CDC® CYBER 200 MODEL 205 COMPUTER SYSTEM

HARDWARE REFERENCE MANUAL

60256020

COMPUTER INSTRUCTION INDEX

Instruction Code	Page Number	Instruction Code	Page Number	Instruction Code	Page Number	Instruction Code	Page Number	Instruction Code	Page Number
00	4-155	34	4-35	62	4-38	8F †	4-73	BD	4-129
03	4-153	35	4-58	63	4-39	90	4-76	BE	4-33
04	4-153	36	4-58	64	4-38	91	4-76	BF	4-33
05	4-154	37	4-123	65	4-38	92	4-76	C0	4-101
06	4-155	38	4-33	66	4-38	93	4-83	C1	4-101
08	4-156	39	4-124	67	4-39	94	4-86	C2	4-101
09	4-58	3A	4-124	68	4-38	95	4-86	C3	4-101
0A	4-159	3B	4-55	69	4-38	96	4-83	C4	4-143
0C	4-157	3C	4-122	6B	4-38	97	4-83	C5	4-143
0D	4-157	3D	4-122	6C	4-38	98	4-76	C6	4-143
0E	4-157	3E	4-32	6D	4-36	99	4-76	C7	4-143
0F	4-158	3F	4-32	6E	4-37	9A	4-76	C8	4-144
10	4-42	40	4-38	6F	4-38	9B	4-81	C9	4-144
11	4-42	41	4-38	70	4-39	9C	4-83	CA	4-144
12	4-122	42	4-38	71	4-39	9D	4-81	CB	4-144
13	4-122	44	4-38	72	4-39	A0 †	4-93	CC	4-148
14	4-131	45	4-38	73	4-42	A1 †	4-93	CD	4-33
15	4-133	46	4-38	74	4-47	A2 †	4-93	CE	4-33
16	4-133	48	4-38	75	4-47	A4 †	4-93	CF †	4-126
1C	4-150	49	4-38	76	4-42	A5 †	4-93	D0	4-109
1D	4-150	4B	4-38	77	4-42	A6 †	4-93	D1	4-107
1E	4-151	4C	4-38	78	4-39	A8 †	4-97	D4	4-109
1F	4-153	4D	4-32	79	4-39	A9 †	4-97	D5	4-107
20	4-50	4E	4-32	7A	4-39	AB †	4-97	D8 †	4-149
21	4-50	4F	4-38	7B	4-42	AC †	4-97	D9 †	4-149
22	4-50	50	4-39	7C	4-42	AF †	4-97	DA	4-104
23	4-50	51	4-39	7D	4-123	B0	4-60,63 140,142	DB	4-105
24	4-50	52	4-39	7E	4-122	B1	4-60,63 140,142	DC	4-116
25	4-50	53	4-42	7F	4-122	B2	4-60,63 140,142	DF	4-112
26	4-50	54	4-47	80 †	4-73	B3	4-60,63 140,142	F0	4-119
27	4-50	55	4-47	81 †	4-73	B4	4-60,63 140,142	F1	4-119
28	4-150	56	4-137	82 †	4-73	B5	4-60,63 140,142	F2	4-119
2A	4-50	58	4-39	83	4-74	B6	4-64	F3	4-119
2B	4-50	59	4-39	84 †	4-73	B7	4-114	F4	4-119
2C	4-34	5A	4-39	85 †	4-73	B8	4-110	F5	4-119
2D	4-34	5B	4-42	86 †	4-73	BA	4-113	F6	4-119
2E	4-34	5C	4-42	87	4-74	BB	4-125	F7	4-119
2F	4-51	5D	4-42	88 †	4-73	BC	4-125	F8	4-118
30	4-35	5E	4-122	89 †	4-73				
31	4-58	5F	4-122	8A	4-75				
32	4-55	60	4-38	8B †	4-73				
33	4-52	61	4-38	8C †	4-73				

† These instructions have sign control capability

CDC® CYBER 200 MODEL 205 COMPUTER SYSTEM

HARDWARE REFERENCE MANUAL

60256020

REVISION RECORD

REVISION	DESCRIPTION
01	Preliminary manual.
(09-29-80)	
A	Manual released. This edition obsoletes all previous editions.
(03-02-81)	
B	Manual revised to make miscellaneous technical and editorial changes.
(10-15-82)	
C	Manual revised to include the Series 600 computer system and to document metal-oxide semiconductor
(11-11-83)	(MOS) memory.

Publication No.
60256020

REVISION LETTERS I, O, Q, S, X AND Z ARE NOT USED.

Address comments concerning this manual to:
Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Inside Front Cover	B
Title Page	-
ii	C
iii	C
iv	C
v	C
vi	C
vii	C
viii	C
ix	C
x	C
xi	C
xii	C
xiii	C
xiv	C
Divider	-
1-1	C
1-2	C
1-3	C
1-4	C
1-5	C
1-6	C
1-7	C
1-8	C
1-9	C
1-10	C
1-11	C
1-12	C
1-13	C
Divider	-
2-1	A
2-2	A
2-3	A
2-4	A
2-5	A
2-6	A
2-7	C
2-8	A
2-9	A
2-10	A
2-11	A
2-12	A
2-13	A
2-14	B
2-15	B
2-16	B
2-17	B
2-18	A
2-19	A
2-20	A
2-21	A
2-22	A
2-23	A
2-24	A
2-25	A
2-26	A
2-27	A

PAGE	REV
2-28	A
2-29	A
2-30	A
2-31	A
2-32	A
2-33	A
2-34	A
2-35	A
2-36	A
2-37	A
2-38	A
2-39	A
2-40	B
2-41	A
2-42	B
2-43	C
2-44	B
2-45	A
2-46	B
2-47	C
2-48	C
2-49	C
2-50	C
2-51	C
2-52	C
2-53	C
2-54	C
2-55	C
2-56	C
2-57	C
2-58	C
2-59	C
2-60	C
2-61	C
2-62	C
2-63	C
2-64	C
2-65	C
2-66	C
2-67	C
2-68	C
2-69	C
2-70	C
2-71	C
2-72	C
2-73	C
Divider	-
3-1	A
3-2	A
3-3	A
3-4	B
3-5	A
Divider	-
4-1	A
4-2	A
4-3	A
4-4	A
4-5	A
4-6	A

PAGE	REV
4-7	A
4-8	A
4-9	B
4-10	A
4-11	A
4-12	A
4-13	A
4-14	A
4-15	A
4-16	B
4-17	A
4-18	A
4-19	A
4-20	A
4-21	A
4-22	A
4-23	A
4-24	A
4-25	B
4-26	A
4-27	B
4-28	A
4-29	B
4-30	A
4-31	A
4-32	A
4-33	A
4-34	A
4-35	B
4-36	A
4-37	A
4-38	A
4-39	A
4-40	A
4-41	A
4-42	A
4-43	A
4-44	A
4-45	A
4-46	A
4-47	B
4-48	A
4-49	A
4-50	A
4-51	A
4-52	A
4-53	A
4-54	A
4-55	B
4-56	B
4-57	A
4-58	A
4-59	A
4-60	A
4-61	A
4-62	A
4-63	A
4-64	A
4-65	A

PAGE	REV
4-66	A
4-67	B
4-68	A
4-69	B
4-70	A
4-71	A
4-72	A
4-73	A
4-74	A
4-75	A
4-76	A
4-77	A
4-78	A
4-79	A
4-80	A
4-81	A
4-82	A
4-83	A
4-84	A
4-85	A
4-86	A
4-87	A
4-88	A
4-89	A
4-90	A
4-91	A
4-92	A
4-93	A
4-94	A
4-95	A
4-96	A
4-97	A
4-98	A
4-99	A
4-100	B
4-101	A
4-102	A
4-103	A
4-104	A
4-105	A
4-106	A
4-107	A
4-108	A
4-109	A
4-110	A
4-111	A
4-112	A
4-113	A
4-114	A
4-115	A
4-116	A
4-117	A
4-118	A
4-119	A
4-120	A
4-121	A
4-122	A
4-123	A
4-124	A

PAGE	REV
4-125	A
4-126	A
4-127	B
4-128	A
4-129	A
4-130	A
4-131	A
4-132	A
4-133	A
4-134	A
4-135	A
4-136	A
4-137	A
4-138	A
4-139	B
4-140	A
4-141	A
4-142	A
4-143	A
4-144	A
4-145	A
4-146	A
4-147	A
4-148	B
4-149	A
4-150	B
4-151	A
4-152	A
4-153	A
4-154	B
4-155	B
4-156	B
4-157	A
4-158	A
4-159	A
Divider	-
5-1	A
5-2	A
5-3	A
5-4	A
5-5	A
5-6	A
5-7	A
5-8	A
5-9	A
5-10	A
5-11	A
5-12	A
5-13	B
5-14	A
5-15	A
5-16	B
5-17	A
5-18	A
5-19	A
5-20	A
5-21	B
5-22	A
5-23	A

PAGE	REV
5-24	B
5-25	A
5-26	A
5-27	A
5-28	A
5-29	A
5-30	A
5-31	A
5-32	A
5-33	A
5-34	B
5-35	A
5-36	A
5-37	A
5-38	A
5-39	A
5-40	A
5-41	A
5-42	A
5-43	A
Divider	-
A-1	A
A-2	A
A-3	A
A-4	A
A-5	A
A-6	A
A-7	A
A-8	A
A-9	A
A-10	A
A-11	B
A-12	A
A-13	A
A-14	B
A-15	B
A-16	A
A-17	A
A-18	A
A-19	A
Divider	-
B-1	A
B-2	B
B-3	A
B-4	B
B-5	A
B-6	A
B-7	A
B-8	B
B-9	A
B-10	A
B-11	A
B-12	A
B-13	A
B-14	A
B-15	A
B-16	A
B-17	A
B-18	A
B-19	A
B-20	A
B-21	A
B-22	A
Divider	-
C-1	A
C-2	A
C-3	A
C-4	A
C-5	A
C-6	A
C-7	B

PAGE	REV
C-8	A
C-9	A
C-10	A
Divider	-
D-1	A
D-2	A
D-3	A
D-4	A
Index-1	B
Index-2	B
Comment Sheet	C
Back Cover	-

PAGE	REV

PAGE	REV

PAGE	REV

PREFACE

This manual contains hardware reference information for the CDC ® CYBER 200 Model 205 Computer System, Series 400 and Series 600.

RELATED PUBLICATIONS

Other manuals applicable to the CYBER 200 Model 205 Computer System and associated equipment include the following:

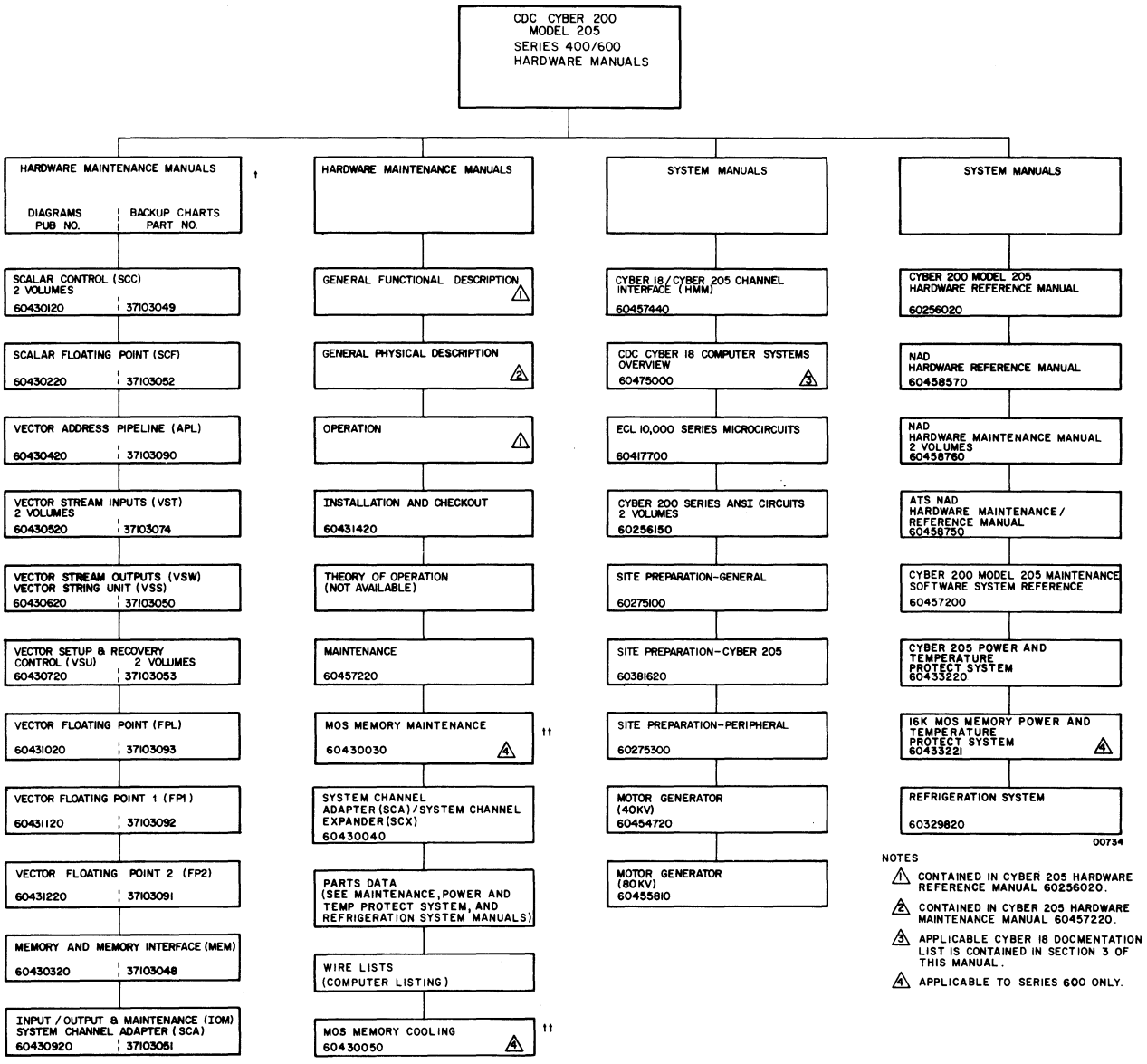
Control Data Publication	Publication Number
CYBER 200 Model 205 Maintenance Software System Reference Manual	60457200
CYBER 200 Model 205 General Physical Description Manual	60256120
CYBER 200 Model 205 Power and Temperature Protect Systems Manual	60433220
CYBER 200 Model 205 Refrigeration System Manual	60329820
CYBER 200 Model 205 16K MOS Hardware Maintenance Manual	60430030
CYBER 200 Model 205 MOS Memory Cooling Hardware Maintenance Manual	60430050
CYBER 18 Computer Systems Overview Manual (section 3 contains a list of applicable documents)	60475000
Programming Reference Aids Manual	60158600

These manuals are available from:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, MN 55103
(612) 292-2100

SYSTEM PUBLICATION INDEX

THE SYSTEM PUBLICATION INDEX PROVIDES A LISTING OF THE RELATED CDC CYBER 200 MODEL 205 SERIES 400/600 HARDWARE MANUALS. THIS INDEX ALSO INCLUDES CYBER 205 DIAGRAM BACKUP CHARTS.



NOTES

1. CONTAINED IN CYBER 205 HARDWARE REFERENCE MANUAL 60256020.
2. CONTAINED IN CYBER 205 HARDWARE MAINTENANCE MANUAL 60457220.
3. APPLICABLE CYBER 18 DOCMENTATION LIST IS CONTAINED IN SECTION 3 OF THIS MANUAL.
4. APPLICABLE TO SERIES 600 ONLY.

† AVAILABLE FROM:
ADVANCED DESIGN LABS
4290 FERNWOOD STREET
ST. PAUL, MINNESOTA 55112

†† CONTACT LITERATURE AND DISTRIBUTION SERVICES FOR AVAILABILITY OF THIS MANUAL.

REVISED 11/1/83

CONTENTS

1. SYSTEM DESCRIPTION 1-1

Introduction 1-1
- Series 400 1-4
- Series 600 1-4

Physical Characteristics 1-4
- Series 400 1-4
- Series 600 1-4
- Measurements and Weight 1-7
- Power System 1-7
- Cooling 1-7

Functional Characteristics 1-7
- CPU Characteristics 1-7
- Virtual Addressing Mechanism 1-8
- Instruction Repertoire 1-8
- Central Memory 1-8
- I/O Ports 1-8

Major System Component Description 1-9
- CPU 1-9
  - Scalar Processor 1-10
  - Vector Processor 1-12
  - I/O Ports 1-13
- Central Memory 1-13
- MCU 1-13

2. FUNCTIONAL DESCRIPTIONS 2-1

General 2-1

CPU Description 2-1

Scalar Processor Description 2-2
- Priority Unit 2-2
  - Bank Busy Checks 2-3
  - Memory Interface Buffer Checks 2-4
  - Memory Interface Signals 2-4
  - Absolute Bounds Address 2-5
  - Retry Unit 2-5
- RNS/Branch Unit 2-6
- Instruction Stack 2-6
- Instruction Issue Unit 2-6
- Associative Unit 2-8
  - Searching the Page Tables 2-9
  - Multiple-Match Fault 2-12
- Load/Store Unit 2-12
- Register File 2-12
- Scalar Floating Point 2-13
  - Scalar Floating-Point Unit Control Interface 2-13
- Scalar Processor Microcode Memories 2-15
  - Scalar Microcode Memories (PM00, PM01) 2-15
  - Associative Microcode Memory (HM00) 2-16
  - Floating-Point and Divide Microcode Memories (DM00, GM00) 2-17

Vector Processor 2-18
- Vector Setup and Recovery Control (VSU) 2-18
  - Inputs to VSU 2-20
  - VSU Operation 2-21
  - Interrupt and Branch Control 2-22
  - Timers 2-23
  - Data Flag Register and Control 2-24
  - VSU Microcodes 2-26
- Stream Addressing Pipeline (APL) 2-26
  - Stream Input Operation 2-28
  - Stream Output Operation 2-28
- Vector Stream Input (VST) 2-29
  - VST SECDED 2-29
  - VST Expansion Networks 2-29
  - VST Scale Network 2-29
  - Field Length Registers 2-30
  - Register File Reads/Writes 2-30
  - Halts/Interrupts 2-30
- Vector Floating-Point Pipeline 2-31
  - Pipeline Data Interchange 2-33
  - Add Unit 2-33
  - Multiply/Divide Unit 2-34
  - Shift Unit 2-35
  - Logical Unit 2-35
  - Delay Unit 2-35
  - Vector Floating-Point Control 2-37
- Vector Stream Output (VSW) 2-37
  - Write One (Pipelines and Register File) 2-37
  - Write Two (String, VSS) 2-38

Single Error Correction Double Error Detection (SECDED) 2-40
- CPU Word Address Bits (36 through 58) 2-40
- SECDED Error Latching Hardware 2-41
- SECDED Usage 2-41
  - Mode 1 2-41
  - Mode 2 2-41
  - Double Error Log (Mode 2A) 2-41
- SECDED Faults 2-42
  - Block Write Enables 2-42

Input/Output 2-42
- I/O Ports 2-42
- System Channel Adapter 2-43
  - CYBER 205 Interface Lines 2-43
- External Device Transmission Sequence 2-45

System Communication 2-45
Storage and Maintenance Access 2-47
I/O Priority 2-47
Central Memory - Series 400 2-47
Central Memory - Series 600 2-48
Memory Operation 2-48
Memory Access and Control 2-52
Stack Request - Series 400 2-53
Stack Request - Series 600 2-53
Bank Address - Series 400 2-53
Bank Address - Series 600 2-53
Absolute Address - Series 400 2-53
Absolute Address - Series 600 2-53
Clock - Series 400 2-53
Clock - Series 600 2-53
Write Control - Series 400 2-54
Write Control - Series 600 2-54
Write Data 2-54
Sync - Series 400 2-54
Sync - Series 600 2-54
Master Clear - Series 400 2-54
Master Clear - Series 600 2-54
Read Data 2-54
Memory Interface 2-55
Memory Degradation 2-55
Maintenance Control Unit 2-58
System Channel Interface (SCI) 2-58
Interfacing Between SCA and SCI 2-58
Control From A 2-60
Functions From B 2-61
Status Words 2-62
Function Word 2-63
Maintenance Data Transfers 2-64

3. OPERATING INSTRUCTIONS 3-1

Controls and Indicators 3-1
Startup Procedures 3-1
Operating Procedures 3-4
System Stop (Normal) 3-4
Emergency Stop 3-5

4. INSTRUCTION DESCRIPTIONS 4-1

General 4-1
Instruction Word Formats 4-1
Instruction Designators 4-1
Unused Bit Areas 4-1
Instruction Types 4-9
Instruction Descriptions 4-31
Index Instructions 4-32
3E Enter (R) with I (16 Bits) 4-32
3F Increase (R) by I (16 Bits) 4-32
4D Half-Word Enter (R) with I (16 Bits) 4-32
4E Half-Word Increase (R) by I (16 Bits) 4-32
CD Half-Word Enter (R) with I (24 Bits) 4-33
CE Half-Word Increase (R) with I (24 Bits) 4-33
BE Enter (R) with I (48 Bits) 4-33
BF Increase (R) with I (48 Bits) 4-33
38 Transmit (R Bits 00-15) to (T Bits 00-15) 4-33
Register Instructions 4-34
2C Logical Exclusive OR (R), (S) to (T) 4-34
2D Logical AND (R), (S), to (T) 4-34
2E Logical Inclusive OR (R), (S) to (T) 4-34
30 Shift (R) Per (S) to (T) 4-35
34 Shift (R) Per (S) to (T) 4-35
6D Insert Bits from (R) to (T) Per (S) 4-36
6E Extract Bits from (R) to (T) Per (S) 4-37
40/60 Add U; (R) + (S) to (T) 4-38
41/61 Add L; (R) + (S) to (T) 4-38
42/62 Add N; (R) + (S) to (T) 4-38
44/64 Sub U; (R) - (S) to (T) 4-38
45/65 Sub L; (R) - (S) to (T) 4-38
46/66 Sub N; (R) - (S) to (T) 4-38
48/68 Mpy U; (R) • (S) to (T) 4-38
49/69 Mpy L; (R) • (S) to (T) 4-38
4B/6B Mpy S; (R) • (S) to (T) 4-38
4C/6C Div U; (R) / (S) to (T) 4-38
4F/6F Div S; (R) / (S) to (T) 4-38
63 Add Address (R) + (S) to (T) 4-39
67 Sub Address (R) - (S) to (T) 4-39
58/78 Transmit (R) to (T) 4-39
59/79 Absolute (R) to (T) 4-39
51/71 Floor (R) to (T) 4-39
52/72 Ceiling (R) to (T) 4-39
5A/7A Exponent of (R) to (T) 4-39
50/70 Truncate (R) to (T) 4-39
5B/7B Pack (R), (S) to (T) 4-42
5C Extend 32 Bit (R) to 64 Bit (T) 4-42
5D Index Extend 32 Bit (R) to 64 Bit (T) 4-42
76 Contract 64 Bit (R) to 32 Bit (T) 4-42
77 Rounded Contract 64 Bit (R) to 32 Bit (T) 4-42
7C Length of (R) to (T) 4-42

53/73 Significant Square Root of (R) to (T) 4-42
10 Convert BCD to Binary Fixed Length 4-42
11 Convert Binary to BCD, Fixed Length 4-42
54/74 Adjust Significance of (R) Per (S) to (T) 4-47
55/75 Adjust Exponent of (R) Per (S) to (T) 4-47
2A Enter Length of (R) with I (16 Bits) 4-50
2B Add to Length Field 4-50
Branch Instructions 4-50
20/24 Branch if (R) ≠ (S) (32/64 Bit FP) 4-50
21/25 Branch if (R) = (S) (32/64 Bit FP) 4-50
22/26 Branch if (R) ≥ (S) (32/64 Bit FP) 4-50
23/27 Branch if (R) < (S) (32/64 Bit FP) 4-50
2F Register Bit Branch and Alter 4-51
33 Data Flag Register Bit Branch and Alter 4-52
3B Data Flag Register Load/ Store 4-55
32 Bit Branch and Alter 4-55
36 Branch and Set (R) to Next Instruction 4-58
31 Increase (R) and Branch if (R) ≠ 0 4-58
35 Decrease (R) and Branch if (R) ≠ 0 4-58
09 Exit Force 4-58
B0 Compare Integer, Branch if (A) + (X) = (Z) 4-60
B1 Compare Integer, Branch if (A) + (X) ≠ (Z) 4-60
B2 Compare Integer, Branch if (A) + (X) ≥ (Z) 4-60
B3 Compare Integer, Branch if (A) + (X) < (Z) 4-60
B4 Compare Integer, Branch if (A) + (X) ≤ (Z) 4-60
B5 Compare Integer, Branch if (A) + (X) > (Z) 4-60
B0 Compare FP, Branch if (A) = (X) 4-63
B1 Compare FP, Branch if (A) ≠ (X) 4-63
B2 Compare FP, Branch if (A) ≥ (X) 4-63
B3 Compare FP, Branch if (A) < (X) 4-63
B4 Compare FP, Branch if (A) ≤ (X) 4-63
B5 Compare FP, Branch if (A) > (X) 4-63
B6 Branch to Immediate Address (R) + I (48 Bits) 4-64
Vector Instructions 4-64
Instruction Formats 4-64
Subfunction Bits 4-66
Field Lengths, Base Address, and Offsets 4-68
Control Vector 4-69
Vector Instruction Termination 4-70
Example of Vector Instruction Operation 4-70
80 Add U; A + B→C 4-73
81 Add L; A + B→C 4-73
82 Add N; A + B→C 4-73
84 Sub U; A - B→C 4-73
85 Sub L; A - B→C 4-73
86 Sub N; A - B→C 4-73
88 Mpy U; A • B→C 4-73
89 Mpy L; A • B→C 4-73
8B Mpy S; A • B→C 4-73
8C Div U; A/B→C 4-73
8F Div S; A/B→C 4-73
83 Add A; A + B→C 4-74
87 Sub A; A - B→C 4-74
8A Shift; A/B→C 4-75
98 Transmit A→C 4-76
99 Absolute A→C 4-76
91 Floor A→C 4-76
92 Ceiling A→C 4-76
9A Exponent of A→C 4-76
90 Truncate A→C 4-76
9B Pack A, B→C 4-81
9D Logical, A, B→C 4-81
9C Extend 32 Bit A→64 Bit C 4-83
96 Contract 64 Bit A→32 Bit C 4-83
97 Rounded Contract 64 Bit A→32 Bit C 4-83
93 Significant Square Root of A→C 4-83
94 Adjust Significance of A Per B→C 4-86
95 Adjust Exponent of A Per B→C 4-86
Sparse Vector Instructions 4-88
Sparse Vector Instruction Format 4-91
Base Addresses and Field Lengths 4-92
Sparse Vector Instruction Termination 4-92
Instructions A0 through AF 4-93
A0 Add U; A + B→C 4-93
A1 Add L; A + B→C 4-93
A2 Add N; A + B→C 4-93

A4 Sub U; A - B → C 4-93
A5 Sub L; A - B → C 4-93
A6 Sub N; A - B → C 4-93
A8 Mpy U; A • B → C 4-97
A9 Mpy L; A • B → C 4-97
AB Mpy S; A • B → C 4-97
AC Div U; A/B → C 4-97
AF Div S; A/B → C 4-97
Vector Macro Instructions 4-101
C0 Select EQ; A = B, Item Count to (C) 4-101
C1 Select NE; A ≠ B, Item Count to (C) 4-101
C2 Select GE; A ≥ B, Item Count to (C) 4-101
C3 Select LT; A < B, Item Count to (C) 4-101
DA Sum ($A_0 + A_1 + A_2 + \ldots A_n$) to (C) and (C + 1) 4-104
DB Product ($A_0, A_1, A_2, \ldots A_n$) to C 4-105
D5 Delta $[A_{(n+1)} - A_n] \rightarrow C_n$ 4-107
D1 Adj. Mean $[A_{(n+1)} + A_n]/2 \rightarrow C_n$ 4-107
D0 Average $(A_n + B_n)/2 \rightarrow C_n$ 4-109
D4 Ave. Diff $(A_n - B_n)/2 \rightarrow C_n$ 4-109
B8 Transmit Reverse A → C 4-110
DF Interval A Per B → C 4-112
BA Transmit Indexed List → C 4-113
B7 Transmit List → Indexed C 4-114
DC Vector Dot Product to (C) and (C + 1) 4-116
String Instruction 4-118
F8 Move Bytes Left; A → C 4-118
Logical String Instructions 4-119
F0 Logical Exclusive OR A, B → C 4-119
F1 Logical AND A, B → C 4-119
F2 Logical Inclusive OR A, B → C 4-119
F3 Logical Stroke, A, B → C 4-119
F4 Logical Pierce A, B → C 4-119
F5 Logical Implication A, B → C 4-119
F6 Logical Inhibit A, B → C 4-119
F7 Logical Equivalence A, B → C 4-119
Nontypical Instructions 4-122
3D Index Multiply (R) • (S) to (T) 4-122
3C Half-Word Index Multiply (R) • (S) to (T) 4-122
5E/7E Load (T) Per (S), (R) 4-122
5F/7F Store (T) Per (S), (R) 4-122
12/13 Load/Store Byte (T) Per (S), (R) 4-122
37 Transmit Job Interval Timer to (T) 4-123
7D Swap S → T, R → S 4-123
39 Transmit Real-Time Clock to (T) 4-124
3A Transmit (R) to Job Interval Timer 4-124
BB Mask A, B → C Per Z 4-125
BC Compress A → C Per Z 4-125
CF Arith. Compress A → C Per B 4-126
BD Merge A, B → C; Per Z 4-129
14 Bit Compress 4-131
15 Bit Merge 4-133
14 Bit Mask 4-133
56 Select Link 4-137
Compare Instructions (B0 through B5) 4-140
B0 Compare Integer, Set Condition if (A) + (X) = (Z) 4-140
B1 Compare Integer, Set Condition if (A) + (X) ≠ (Z) 4-140
B2 Compare Integer, Set Condition if (A) + (X) ≥ (Z) 4-140
B3 Compare Integer, Set Condition if (A) + (X) < (Z) 4-140
B4 Compare Integer, Set Condition if (A) + (X) ≤ (Z) 4-140
B5 Compare Integer, Set Condition if (A) + (X) > (Z) 4-140
B0 Compare FP, Set Condition if (A) = (X) 4-142
B1 Compare FP, Set Condition if (A) ≠ (X) 4-142
B2 Compare FP, Set Condition if (A) ≥ (X) 4-142
B3 Compare FP, Set Condition if (A) < (X) 4-142
B4 Compare FP, Set Condition if (A) ≤ (X) 4-142
B5 Compare FP, Set Condition if (A) > (X) 4-142
C4 Compare EQ: A = B, Order Vector → Z 4-143
C5 Compare NE: A ≠ B, Order Vector → Z 4-143
C6 Compare GE: A ≥ B, Order Vector → Z 4-143
C7 Compare LT: A < B, Order Vector → Z 4-143
C8 Search EQ; A = B, Index List → C 4-144
C9 Search NE; A ≠ B, Index List → C 4-144
CA Search GE; A ≥ B, Index List → C 4-144
CB Search LT; A < B, Index List → C 4-144

CC Mask Binary Compare; (A EQ/NE (B) Per (C) 4-148
D8 Max. of A to (C) Item Count to (B) 4-149
D9 Min. of A to (C) Item Count to (B) 4-149
28 Scan Equal 4-150
1C Form Repeated Bit Mask with Leading Zeros 4-150
1D Form Repeated Bit Mask with Leading Ones 4-150
1E Count Leading Equals R 4-151
1F Count Ones in Field R, Count to T 4-153
03 Keypoint - Maintenance 4-153
04 Breakpoint - Maintenance 4-153
05 Void Stack and Branch 4-154
Monitor Instructions 4-155
00 Idle 4-155
06 Fault Test - Maintenance 4-155
08 Input/Output Per R 4-156
0C Store Associative Registers 4-157
0D Load Associative Registers 4-157
0E Translate External Interrupt 4-157
0F Load Keys from (R), Translate Address (S) to (T) 4-158
0A Transmit (R) to Monitor Interval Timer 4-159

5. PROGRAMMING INFORMATION 5-1

General 5-1
Monitor and Job Modes 5-1
Exchange from Monitor Mode to Job Mode 5-1
Illegal Instruction in Monitor Mode 5-2
Exchange from Job Mode to Monitor Mode 5-2
Interrupts 5-3
Storage Access Interrupts 5-4
External Interrupts 5-5
I/O Channel Interrupt Lines 5-5
Monitor Interval Timer Interrupt 5-6
Invisible Package 5-6
Addressing Modes 5-13
Virtual Addressing 5-13
Pages 5-13
Virtual Address Format 5-14
Associative Words 5-16
Page Table 5-19
Associative Registers 5-19
Space Table 5-19
Operation of Virtual Addressing 5-21
Absolute Address 5-21
Real-time Counters 5-22
Free-Running Clock Counter 5-23
Monitor Interval Timer 5-23
Job Interval Timer 5-23
Register File 5-24
Register File Restrictions 5-24
Register 0 (Trace Register) Restrictions 5-25
Register 0 Content Resulting from an Exchange Operation 5-25
Register 0 Content Resulting from a Swap (7D) Instruction 5-26
Register 0 when Referenced by an Instruction Designator 5-26
Registers 1 and 2 (64-Bit), 2 through 5 (32-Bit) Restrictions 5-32
Registers 0 through 7 (64-Bit), 0 through F (32-Bit) Monitor Mode Restrictions 5-32
Register 1 (32-Bit) Rightmost Half of 64-Bit Register 0 5-32
Common Register File for Monitor and Job Modes 5-33
Data Flag Branch Register 5-33
Data Flags 5-34
Mask Bits 5-36
Product Bits 5-36
Dynamic Inclusive OR for Product Bits 5-37
Scalar Divide, Square Root, Convert Instruction Flag 5-37
Data Flag Branch Enable Bit 5-37
Free Data Flags 5-37
Data Flag Branch Operation 5-39
Data Flag Branch Timing Considerations 5-39
General Definitions and Programming Guides 5-40
Overlap of Operand and Result Fields 5-40
Illegal Instructions 5-40
Instructions which Cause Undefined Results or Operations 5-41
Item Count 5-41
Field Length and Offset 5-42
Index 5-42
Operand Size Definition 5-42
Restriction on Self-Modifying Programs 5-43
Result Vector 64-Sword Look-Ahead 5-43

APPENDIXES

A. NUMBER SYSTEMS AND TABLES A-1
B. FLOATING-POINT ARITHMETIC B-1
C. G BITS AND TERMINATING CONDITIONS C-1
D. DATA FLAG APPLICATIONS TO INSTRUCTIONS D-1

FIGURES

1-1 CYBER 205 Central Computer (Series 400) 1-2
1-2 CYBER 205 Central Computer (Series 600) 1-3
1-3 CYBER 205 Central Computer Floor Plan (Series 400) 1-5
1-4 CYBER 205 Central Computer Floor Plan (Series 600) 1-6
1-5 CYBER 205 Block Diagram 1-10
1-6 Scalar Processor Block Diagram 1-11
1-7 Simplified Diagram - Vector Processor 1-12
2-1 Functional Components of Scalar Processor 2-3
2-2 Page Table Search Examples 2-10
2-3 Vector Processor 2-19
2-4 VSU Block Diagram 2-20
2-5 Floating-Point Pipeline Basic Block Diagram 2-32
2-6 Add Unit Block Diagram 2-33
2-7 Multiply/Divide Unit Block Diagram 2-34
2-8 Shift Unit Block Diagram 2-35
2-9 Logical Unit Block Diagram 2-36
2-10 Delay Unit Block Diagram 2-36
2-11 String Unit Old Data 2-39
2-12 System Channel Adapter 2-43
2-13 I/O Transmission Sequence 2-45
2-14 Section Configuration (Series 400) 2-49
2-15 Section Configuration (Series 600) 2-50
2-16 Two-Sword, Sword, and Word Configuration 2-51
2-17 Memory Interface Stack Connections (Series 400) 2-52
2-18 Memory Interface Module Connections (Series 600) 2-52
2-19 Memory Interface Configuration and Connections for a Two-Pipeline Configuration 2-55
2-20 System Channel Interface (SCI) 2-59
2-21 Status Words 2 and 3 2-63
4-1 Instruction Formats 4-2
4-2 Instruction Listing Format 4-9
4-3 Example of Register Content for an Insert Bits from (R) to (T) Per (S) Instruction 4-36
4-4 Example of Register Content for an Extract Bits from (R) to (T) Per (S) Instruction 4-37
4-5 Example of Register Content for a Ceiling (R) to (T) Instruction 4-40
4-6 Example of Register Content for a Truncate (R) to (T) Instruction 4-41
4-7 Example of Register Content for an Extend 32-Bit (R) to 64-Bit (T) Instruction 4-43
4-8 Example of Register Content for a Contract 64 Bit (R) to 32 Bit (T) Instruction 4-44
4-9 Example of Register Content for a Rounded Contract 64 Bit (R) to 32 Bit (T) Instruction 4-45
4-10 Example of Register Content for a Convert BCD to Binary, Fixed-Length Instruction 4-47
4-11 Example of Register Content for an Adjust Exponent of (R) Per (S) to (T) 4-49
4-12 Example of Bit Branch and Alter Instruction 4-57
4-13 General Vector Instruction Format 4-65
4-14 Operand Field Length, Base Address, and Offset Formats 4-68
4-15 Vector Field Address Format 4-68
4-16 Control Vector Base Address Format (Z) 4-69
4-17 Vector Instruction Example of Register Content and Instruction Format 4-71
4-18 Vector Address Fields for Vector Instruction Example 4-72
4-19 Example of an Add A; A + B → C Instruction 4-75
4-20 Example of Floor A → C Instruction with Negative Exponent 4-77
4-21 Example of a Ceiling A → C Instruction with Negative Exponent 4-79

4-22 Example of Source and Result Elements for a Truncate A→C Instruction 4-80
4-23 Example of Pack A, B→C Instruction 4-82
4-24 Example of Extend 32 Bit A→64 Bit C Instruction 4-84
4-25 Example of Vector Elements for a Rounded Contract 64 Bit A→32 Bit C Instruction 4-86
4-26 Example of Adjust Exponent of A Per B→C Operation 4-88
4-27 Example of Compressing Initial Vector Field into Sparse Vector Field 4-90
4-28 General Sparse Vector Instruction Format 4-91
4-29 Sparse Vector Field Length and Base Address Formats 4-92
4-30 Example of an Add U; A + B →C Sparse Vector Instruction when G Bit 1 = 0 and G Bit 2 = 1 (AND) 4-95
4-31 Example of an Add U; A + B →C Sparse Vector Instruction when G Bit 1 = 1 and G Bit 2 = 0 (Exclusive OR) 4-96
4-32 Example of a Div or Mpy U Sparse Vector Instruction when G Bit 1 = 0 and G Bit 2 = 1 (OR) 4-100
4-33 Example of Select EQ; A=B Item Count to C 4-103
4-34 Example of Delta Instruction 4-108
4-35 Example of a Transmit Reverse A→C Instruction 4-111
4-36 Example of a Transmit Indexed List→C Instruction 4-115
4-37 Example of General Format of a Data String Field 4-118
4-38 Example of Logical String Instruction (Logical Exclusive OR) 4-121
4-39 Example of Arithmetic Compress A→C Per B Instruction 4-128
4-40 Examples of BD Merge Instruction 4-130
4-41 Example of Bit Compress Instruction 4-132
4-42 Example of Bit Merge Instruction 4-134
4-43 Example of Bit Mask Instruction 4-136
4-44 Link Selection 4-138
4-45 Example of Compare GE; A ≥ B; Order Vector→Z Instruction 4-145
4-46 Example of Search EQ; A = B, Index List→C 4-147
4-47 Example of Repeated Bit Mask Data Format (Leading Zeros) 4-151
4-48 Example of Count Leading Equals Data and Register Format 4-152
4-49 Breakpoint Register Format 4-153
4-50 Register Formats for the OF Instruction 4-158
5-1 Invisible Package Word $xx...xxE_{16}$ Format for Access Interrupt 5-4
5-2 Invisible Package Format 5-7
5-3 Virtual Address Formats 5-15
5-4 Associative Word Formats 5-16
5-5 Virtual Address Key Register Format 5-18
5-6 Page Table Format 5-20
5-7 Virtual Address to Absolute Address 5-22
5-8 Register File 5-24
5-9 Virtual/Absolute Address Zero 5-25
5-10 DFB Register Format 5-33

TABLES

2-1 Scalar/Vector Processor Instruction Responsibility 2-8
2-2 Data Flag Register 2-24
2-3 Channel Flag Assignments 2-46
2-4 Memory Port Transfer Modes 2-56
2-5 Series 400 Memory Degradation Bits (4K Chips) 2-56
2-6 Series 600 Memory Degradation Bits 2-57
2-7 Control From A 2-60
2-8 Functions From B 2-61
2-9 Status Word 1 Bits and Descriptions 2-62
2-10 Function Word Bits and Descriptions 2-63
2-11 MCU to CPU Data 2-65
2-12 CPU to MCU Data 2-71
3-1 Startup Procedures 3-1
3-2 System Stop 3-4
3-3 Emergency Stop 3-5
4-1 Instruction Designators 4-5
4-2 Instruction List by Function Code 4-10
4-3 Instruction List by Instruction Type 4-21
4-4 Bit Branching Conditions 4-51

4-5 Bit Altering Conditions 4-52
4-6 DFBR Bit Branch Conditions 4-53
4-7 DFBR Bit Altering Conditions 4-53
4-8 DFBR Branch Address Source Conditions 4-54
4-9 Bit Branching Conditions 4-56
4-10 Bit Altering Conditions 4-56
4-11 Branch Address Source Conditions 4-56
4-12 Index Branch Instruction Designators 4-62
4-13 Integer Ranges 4-62
4-14 Index Branch Instruction Designators 4-63
4-15 Vector Instruction Designators 4-65
4-16 Subfunction Bits 4-66
4-17 Sign Control Subfunction Bits 4-67
4-18 Sparse Vector Instruction Designators 4-91
4-19 G Bit 1 and 2 Operations 4-93
4-20 Results of the Logical Operations (A0 through A6) 4-94
4-21 Results of the Logical Operations (A8 through AB) 4-98
4-22 Results of the Logical Operations (AC, AF) 4-99
4-23 Truth Table for Logical String Instructions 4-119
4-24 DFB Conditions for F0 through F7 Instructions 4-120
4-25 Destruction Used in a Link Operation 4-138
4-26 Combinations of R, G1, and G2 Bits 3 and 4 that can be Selected 4-139
4-24 DFB Conditions for F0 through F7 Instructions 4-120
4-25 Instructions Used in a Link Operation 4-138
4-26 Combinations of R, G1 and G2 Bits 3 and 4 that can be Selected 4-139
4-27 Input Configurations 4-139
4-28 Search Iteration Starting Designator Conditions 4-146
4-29 R Designator Bit Definitions 4-156
5-1 External Interrupt Lines 5-5
5-2 Page Size Specification 5-14
5-3 Associative Word Usage Codes 5-17
5-4 Lockout Codes 5-18
5-5 Page Table Restrictions and Requirements 5-19
5-6 Results for Specified Register Zero 5-27
5-7 Data Flag Register Bit Assignments 5-34
5-8 Free Data Flag Bit Assignments 5-37

SYSTEM DESCRIPTION 1

INTRODUCTION

The CYBER 205 central computer is a large-scale, high-speed, arithmetic-computing system. The basic central computer consists of the following components.

- Central processing unit (CPU)
- One million 64-bit words of central memory
- Eight input/output (I/O) ports
- Maintenance control unit (MCU)

The CPU is available in models with 1, 2, or 4 vector pipelines, and 8 I/O ports that are expandable to 16 I/O ports.

Large-scale integrated (LSI) circuits are used in the CPU to provide high performance and reliability. The CPU contains separate scalar and vector processors that operate on a single instruction stream to provide sequential and parallel operations on single bits, 8-bit bytes, and 32-bit or 64-bit operands and vector elements. The central memory is a high-performance semiconductor memory with single-error correction and double-error detection (SECDED) on each 32-bit half-word for high-storage integrity. The CYBER 205 uses a virtual addressing high-speed mapping technique to allow programs to appear logically contiguous while not being physically contiguous in the storage system.

There are two series of the CYBER 205 computer: the Series 400 and the Series 600. Both series share the same operating features; the main difference is in the physical construction of the memories.

Figures 1-1 and 1-2 show the CYBER 205 Central Computer, Series 400 and Series 600, respectively.

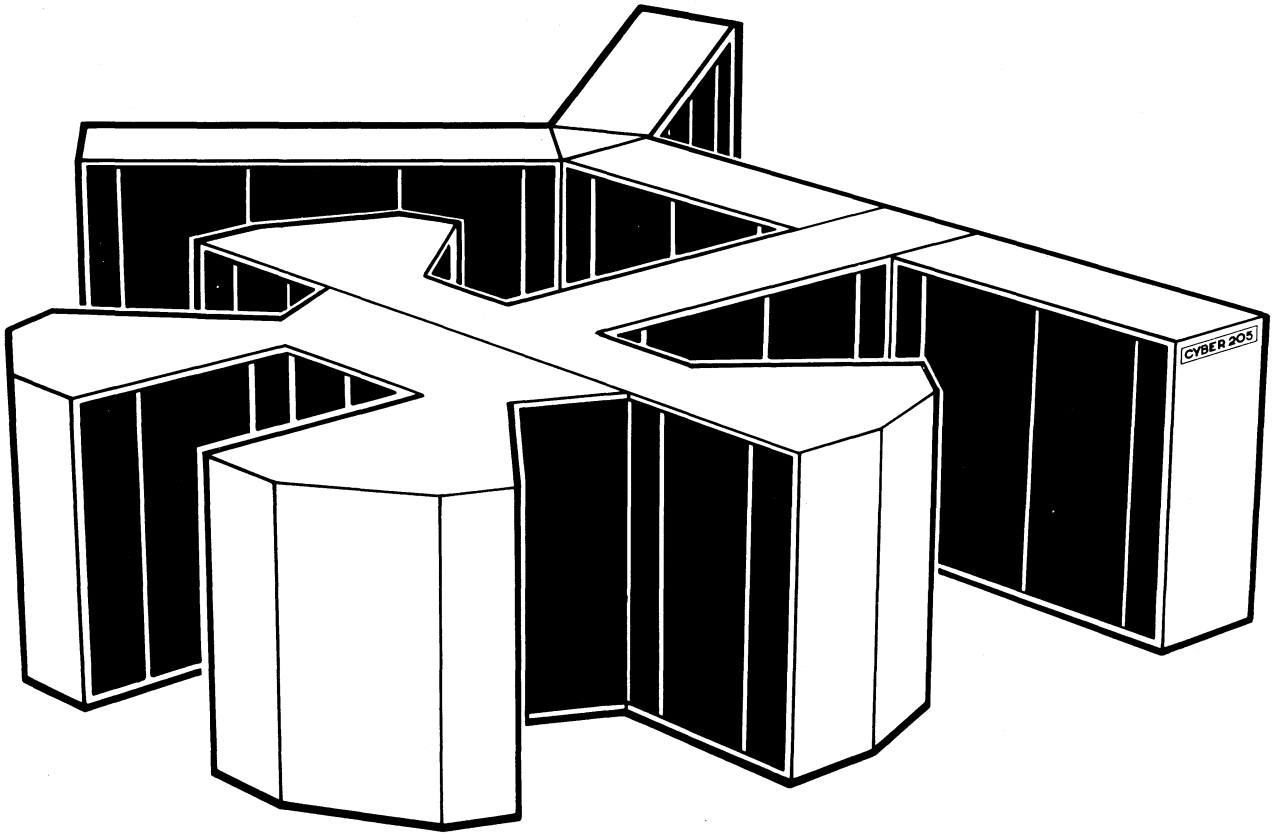


Figure 1-1. CYBER 205 Central Computer (Series 400)

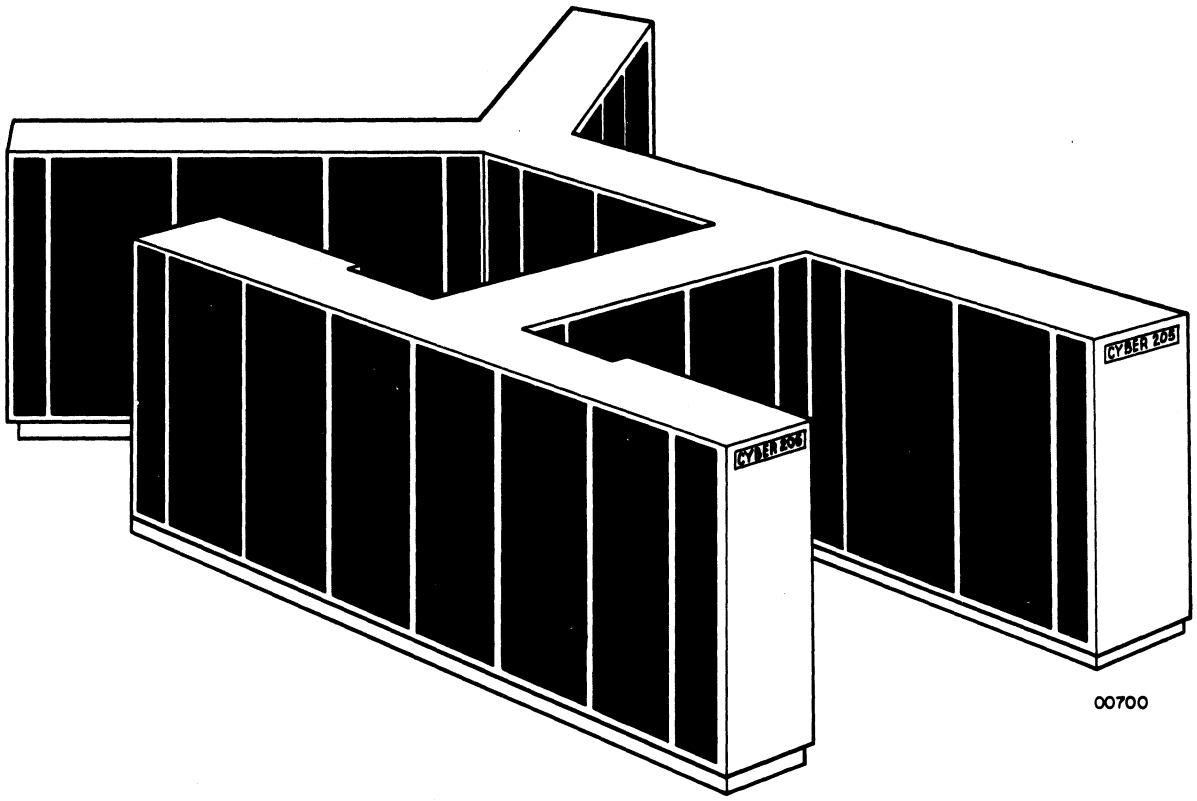


Figure 1-2. CYBER 205 Central Computer (Series 600)

SERIES 400

The Series 400 central memory is composed of 4K bipolar random-access memory (RAM) chips. The basic memory size is 1 million words. The memory is field-expandable to 2 million or 4 million words.

SERIES 600

The Series 600 central memory is composed of 16K metal-oxide semiconductor (MOS) RAM chips. The basic memory size is 1 million words. The memory is field-expandable to 2, 4, or 8 million words.

NOTE

Because of the similarities of the Series 400 and the Series 600 central computers, hereafter when a feature is discussed that is common to both series, a general heading appears at the beginning of the paragraph. However, when a feature is discussed that is peculiar to only one series, the heading Series 400 or Series 600 appears.

PHYSICAL CHARACTERISTICS

Figure 1-3 shows the physical layout of the CYBER 205 Series 400 central memory. Figure 1-4 shows the physical layout of the CYBER 205 Series 600 central computer only. The scalar and vectors of the Series 400 and the Series 600 are identical.

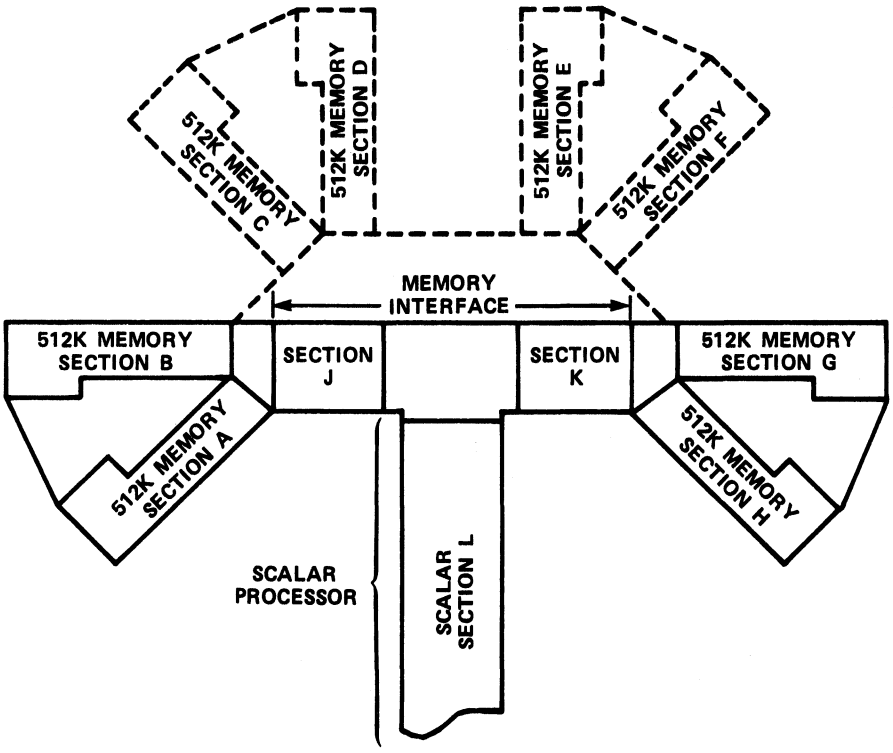
SERIES 400

Central memory is contained in individual sections around the memory interface (sections J and K). The basic 1 million, 64-bit words of central memory are contained in sections A and H. A 1-million-word option is located in sections B and G, and a 2-million-word option is located in sections C, F, D, and E.

The scalar processor is located in section L. The basic vector processor is located in sections N, P, and R. Vector floating-point pipeline 2 (optional) is located in section R. Section S contains optional vector floating-point pipelines 3 and 4. Sections T and U contain the optional divide enhancements for vector floating-point pipelines 1 and 2, and 3 and 4, respectively.

SERIES 600

Central memory is contained in two or four individual cabinets located on either side of the memory interface cabinets (section J and K). The basic 1 million, 64-bit words of central memory are contained in two cabinets (sections A and H). The cabinets are field-expandable to 2 million words. Four million and 8 million words of central memory are contained in four cabinets; sections A, B, G, and H.



NOTE:

SCALAR AND VECTOR FLOORPLAN OF SERIES 400 IDENTICAL TO FLOORPLAN OF SERIES 600.

Figure 1-3. CYBER 205 Central Computer Floor Plan (Series 400)

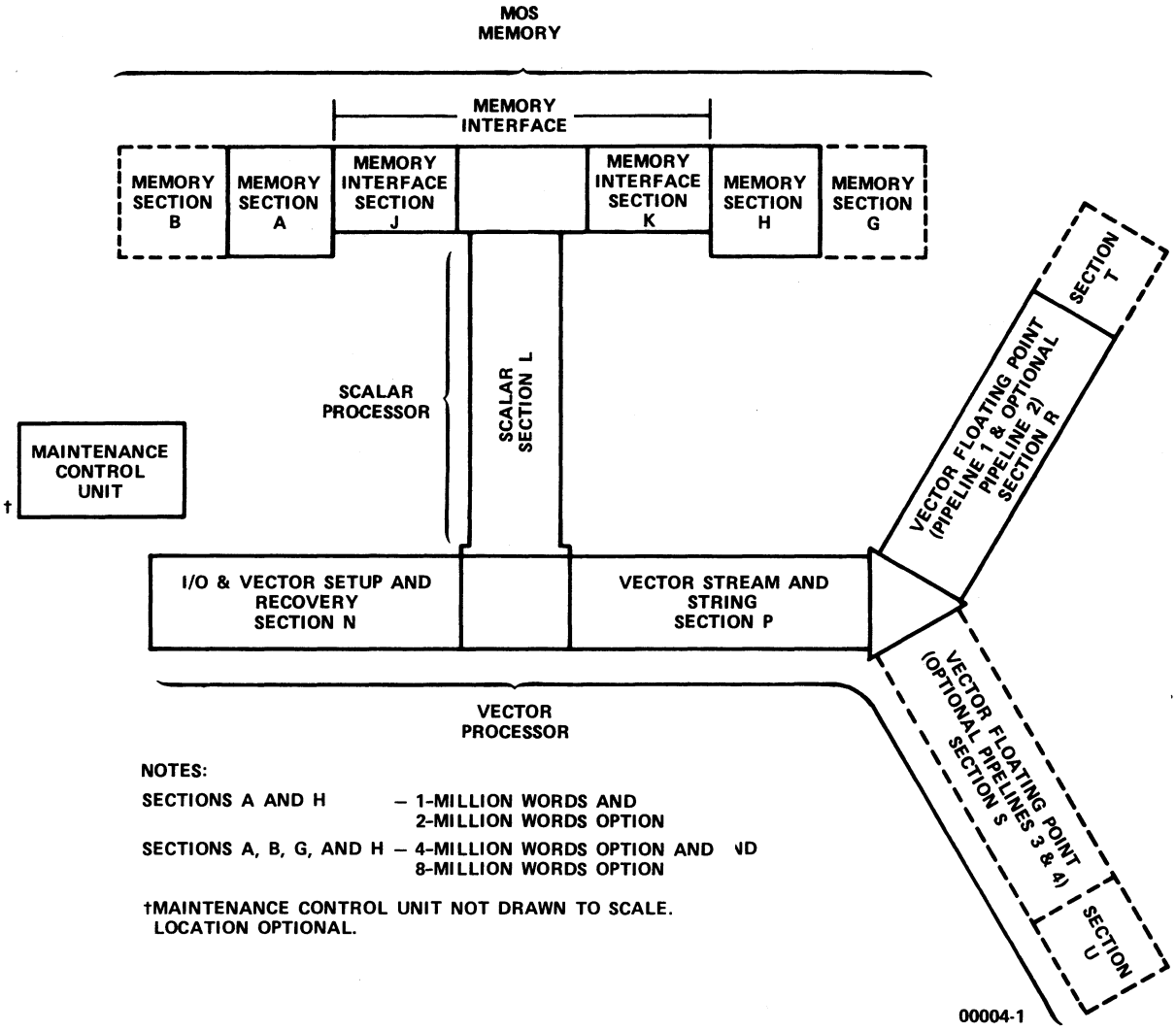


Figure 1-4. CYBER 205 Central Computer Floor Plan (Series 600)

MEASUREMENTS AND WEIGHT

For a complete physical description of the CYBER 200 Model 205 central computer, refer to the CYBER 205 General Physical Description manual listed in the preface.

POWER SYSTEM

For a description of the power system, refer to the CYBER 200 Model 205 Power and Temperature Protect Systems manual listed in the preface.

COOLING

For a description of the cooling system, refer to the CYBER 200 Model 205 Refrigeration System manual (Series 400) or the CYBER 200 Model 205 MOS Memory Cooling Hardware Maintenance manual (Series 600) listed in the preface.

FUNCTIONAL CHARACTERISTICS

The functional characteristics of the CYBER 205 are summarized below. The functional characteristics are described in detail in sections 2 and 4.

CPU CHARACTERISTICS

- Synchronous internal logic with a 20-nanosecond clock period (minor cycle).
- Two's complement arithmetic.
- One, two, or four parallel vector pipelines.
- Hardware macro instructions.
- Sequential stream processing.
- Bit, byte, half-word, or 64-bit floating-point operations.
- Independent scalar and vector instruction execution for no-conflict operations.
- High-speed register file with 256 64-bit registers (2 reads and 1 write per clock period).
- Sixty-four 64-bit word instruction stack for the optimization of programmed scalar loop iteration.
- Monitor and job modes.

VIRTUAL ADDRESSING MECHANISM

- Forty-eight-bit virtual address.
- Program protection via lock and key.
- Sixteen registers for simultaneous virtual to physical mapping.
- Selectable page sizes - small page sizes of 512, 2048, and 8192 words and large page size of 65 536 words.

INSTRUCTION REPERTOIRE

- Thirty-two-bit and 64-bit floating-point arithmetic.
- Vector and sparse vector.
- Vector macros (for example, dot products, inner products, and so on).
- Dot product.
- Square root instructions.
- Integer arithmetic.

CENTRAL MEMORY

- 80-nanosecond access time.
- SECDED for each 32 bits for high reliability.
- Memory sizes of 1, 2, and 4 million 64-bit words (Series 400).
- Memory sizes of 1, 2, 4, and 8 million 64-bit words (Series 600).
- High memory bandwidth to support scalar, and simultaneous vector and I/O operations.
- Data transferred to/from memory in 32-bit half-words, 64-bit words, 512-bit super words (sword), or 1024-bit two-sword quantities.

I/O PORTS

- I/O ports expandable to 16.
- Each port capable of 200 x 10^6 bits per second maximum transfer rate.
- Front-end computer for communications and job entry.
- One channel used for MCU.

MAJOR SYSTEM COMPONENT DESCRIPTION

The following are the CYBER 205 major system components. They are described in detail in the following paragraphs.

- CPU
- Central memory
- MCU

Figure 1-5 shows the CYBER 205 basic block diagram.

The CPU contains the scalar processor, vector processor, and I/O ports.

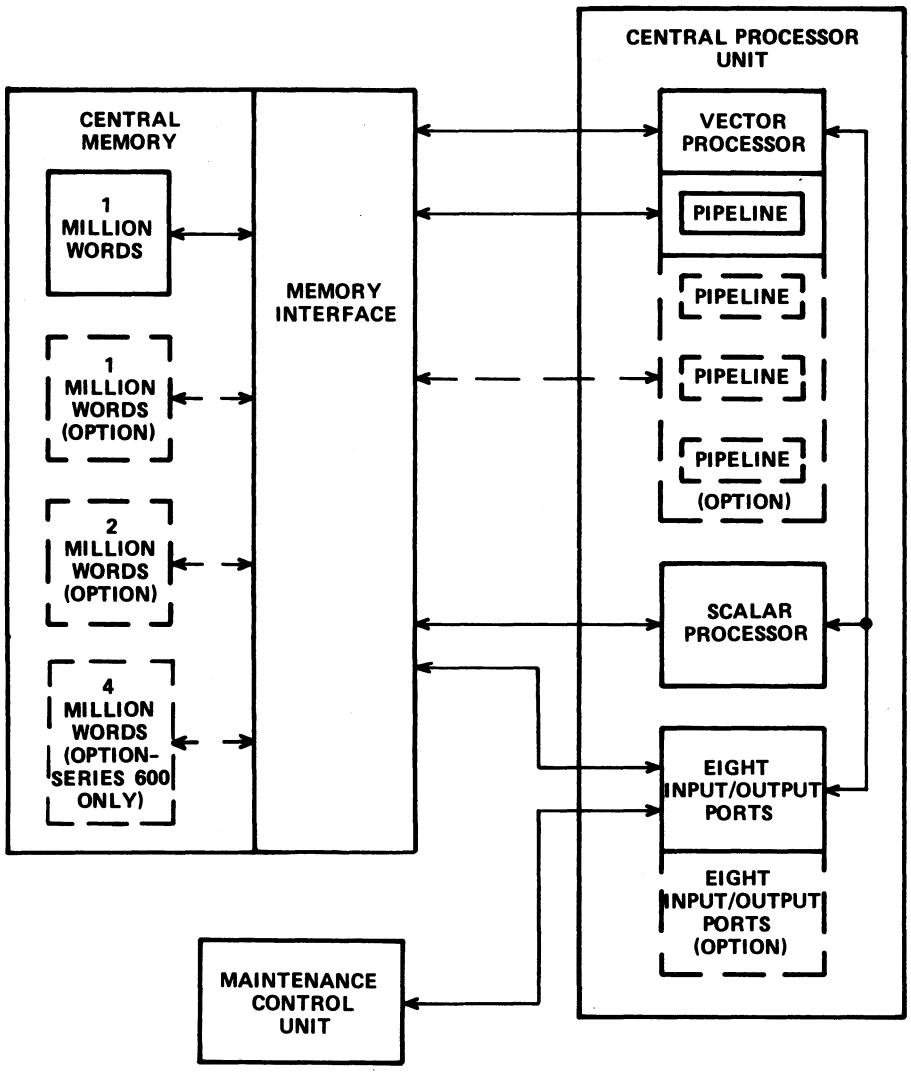


Figure 1-5. CYBER 205 Block Diagram

Scalar

The scalar processor performs the primary system control functions of the CPU in addition to providing the execution of scalar operations. Figure 1-6 shows the scalar processor block diagram.

The scalar processor contains a 64-word instruction stack segmented into 8 superwords (swords). The instruction stack is capable of holding up to 128 32-bit instructions, 64 64-bit instructions, or a combination of both, and provides a 16-word instruction read ahead.

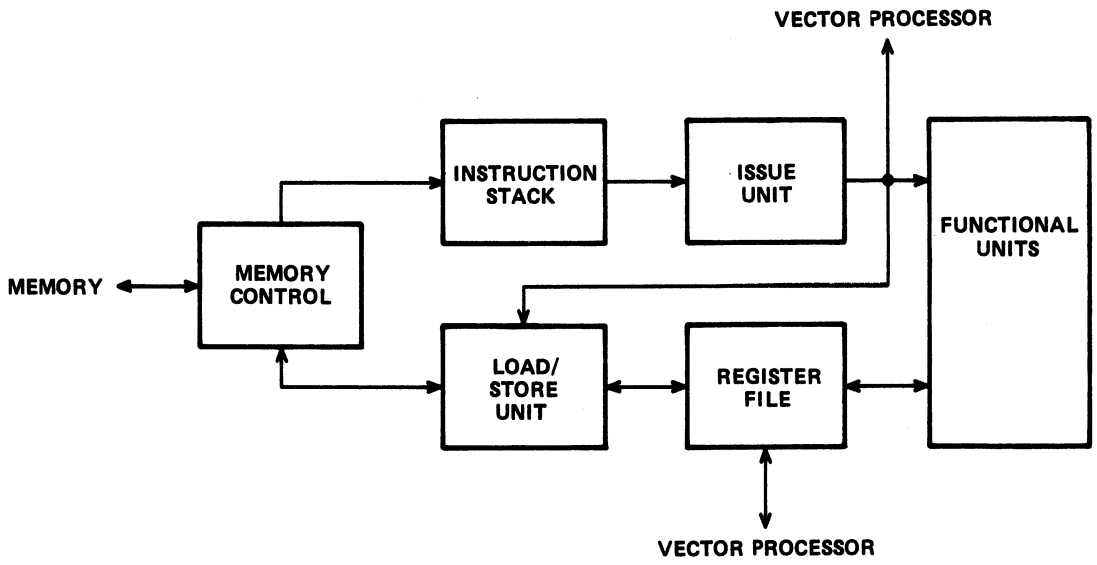


Figure 1-6. Scalar Processor Block Diagram

The issue unit retrieves instructions from the instruction stack. The issue unit decodes all instructions, initiates scalar operations with the appropriate functional unit, and directs decoded vector/string instructions to the vector processor for execution. The issue unit is capable of issuing instructions at the rate of one instruction every minor cycle.

The register file provides 256 64-bit registers for use in instruction and operand addressing, indexing, field lengths, and as source and destination registers for the functional unit operands and results. The register file is capable of two reads and one write every minor cycle.

The scalar arithmetic unit contains independent functional units to attain high scalar performance. These units are used for floating-point arithmetic and logical operations. The functional units can accept a new pair of operands every minor cycle. The functional units receive their operands from and transmit their results to the register file.

The load/store unit receives instructions from the issue unit. It controls data transfers between central memory and the register file. It is capable of accepting one load request every minor cycle or one store request every two minor cycles.

Included in the memory control area are the virtual memory addressing mechanism and the priority unit. Virtual addressing converts a logical address to an absolute storage address to allow programs to appear logically contiguous to the user while not being physically contiguous in the storage system. The priority unit receives memory requests from various sections of the system and resolves memory conflicts.

Vector Processor

The CYBER 205 vector processor is used to process arrays or strings of data. High performance is achieved by specialized hardware operating in parallel to accomplish what otherwise would require issuing a sequence of machine instructions. Thus, for example, two sets of operands can be multiplied to produce a third set of results by issuing a single instruction to the vector processor.

A simplified functional diagram of the vector processor is shown in figure 1-7. Vector instructions are received from the issue unit of the scalar processor. These instructions specify the operation to be performed and the addresses of the operands and results. The vector processor uses buffers in the vector stream input and output units to position operands and results for transmission between the processing elements and memory.

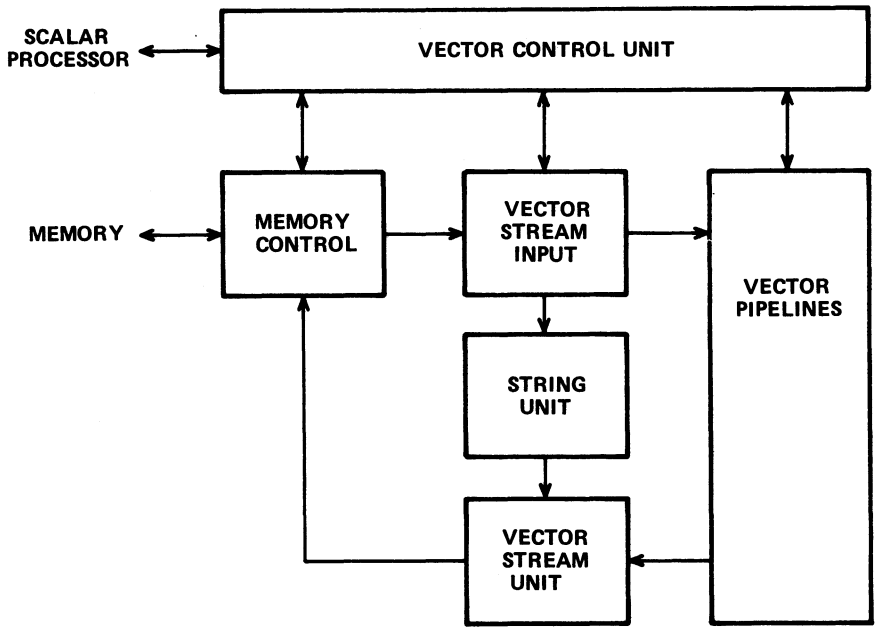


Figure 1-7. Simplified Diagram - Vector Processor

The vector processor has two types of processing elements. The first uses one, two, or four vector pipelines, depending upon the model. When more than one vector pipeline is included, these units operate in parallel on alternating data elements. With the exception of divide and square root, these pipelines each accept a new pair of operands every minor cycle for 64-bit floating-point operations. For 32-bit floating-point operations, the rate is doubled.

The second type of processing element is the string unit. The string unit performs logical operations on strings of data to allow bit operations on bit boundaries.

I/O Ports

The I/O ports provide the control and data paths for communication between central memory and the external devices. The standard configuration provides eight I/O ports with an option for eight more. Each I/O port has a maximum transfer rate of 200 megabits.

Channel interfaces for attachment of the MCU, peripherals, and front end computers are accomplished by the use of SCA.

CENTRAL MEMORY

Central memory is a random-access memory using bipolar, 1K-bit or 4K-bit integrated circuits (Series 400) or 16K MOS circuits (Series 600). The memory words are 78 bits providing a 64-bit data word with 7 bits of SECDED for each 32-bit half-word. The basic memory size is 1 million words with expansions to 2 and 4 million words (Series 400), and 2, 4, or 8 million words (Series 600 only).

MCU

The primary purpose of the MCU is to support the reliability, availability, and maintainability of the system. MCU provides system autoload and system performance monitoring capabilities. The MCU also provides the capability of loading, controlling, and monitoring the central processing unit. It is connected to the CPU through a standard I/O channel by the SCA.

FUNCTIONAL DESCRIPTIONS 2

GENERAL

This section provides a detailed description of the CYBER 205 major system components.

This section starts with a general description of the CPU and then gives a detailed description of the scalar processor, the vector processor, SECDED, I/O ports, central memory, and the MCU.

When reading this section, it should be noted that the following references appear throughout the text.

Half-word	32 bits
Word	64 bits
Superword (sword)	512 bits (eight words)
Two-sword	1024 bits

CPU DESCRIPTION

The CPU contains all string and streaming instruction control, arithmetic units, storage control, and I/O communication control. The CPU consists of the following functional areas.

- Scalar processor
- Vector processor
- I/O ports

The scalar processor is physically contained in a cabinet next to the central memory in order to reduce transfer delays and gain performance. The scalar processor contains the initial instruction decode, five independent arithmetic functional units, a semiconductor register file, and the high performance load/store pipeline unit.

The vector processor performs multioperand instruction by streaming data through functional units. The vector processor contains setup and recovery control, stream addressing pipelines, stream inputs and outputs, input/output control, and one floating-point pipeline with an option of one or three additional pipes.

The eight I/O ports (with an option of eight additional ports) provide the CPU with the physical connection to the external devices.

The scalar and vector processors each contain independent instruction controls. Therefore, operating on a single instruction stream, the scalar processor can execute scalar instructions in parallel with most vector instructions if there are no memory references to load or store operands from the register file. There are two exceptions to the parallel execution of vector and scalar instructions on a single instruction stream.

- The scalar processor cannot execute any scalar register file load or store instructions in parallel with a vector operation requiring references to memory (0F, 12, 13, 5E, 5F, 7E, and 7F instructions).
- The scalar processor cannot issue any instruction while the vector processor executes an instruction that actively uses the register file (7D, B7, or BA instructions).

Register conflict within the register file always delays the issue of a vector or scalar instruction.

SCALAR PROCESSOR DESCRIPTION

The scalar processor provides the central computer instruction control. The scalar processor receives and decodes all instructions from central memory, directs decoded vector/string instructions to the vector processor for execution, and provides orderly buffering and execution of the load and store instructions.

Figure 2-1 shows the functional units of the scalar processor. Each functional unit is described in this section.

PRIORITY UNIT

The priority unit receives memory requests from the various functional units of the machine. After screening out nonvalid requests, the priority unit interprets the requests and drives the memory interface to produce the proper memory activity. A memory request consists of a request line and a set of control bits defining the amount of data to be transferred and whether a read or write is to be performed.

The priority unit upon receiving a valid memory request, responds with an accept to the requesting source to indicate that memory activity is initiated. If, due to a memory busy, the request is not immediately honored, the source or the retry unit in priority repeats the request.

If two requests arrive at the priority unit simultaneously, only one is processed, the other request receives no accept. In all cases of simultaneous request, I/O initiates one of the requests and is always given the accept. Because the I/O request has the highest priority, and is always honored immediately, an I/O accept line is not needed.

When a request has passed the simultaneous request check, an accept is returned to the source, provided the following two conditions are met.

1. The request has an absolute address or virtual address for which a match exists in the associative register. (Refer to the associative unit description for the requirements of a virtual address match.)
2. There is no bank busy conflict.

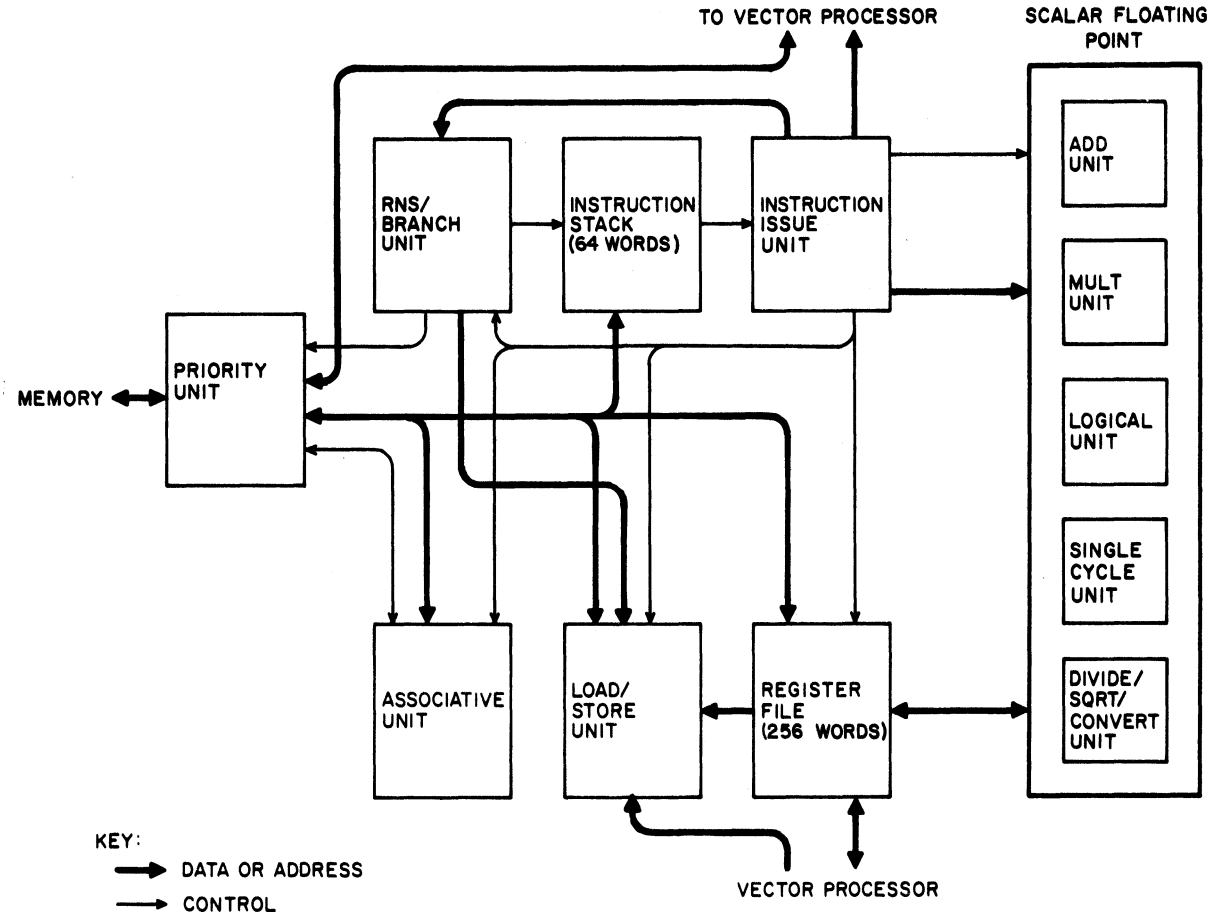


Figure 2-1. Functional Components of Scalar Processor

Requests into the priority unit are either immediate issue or delayed issue. Immediate issue requests consist of all read requests (except I/O read) and short write requests (half-word or word). Delayed issue requests are the long writes (sword or two-sword) which issue out of the priority unit four minor cycles later than an immediate issue request. This extra delay allows the data buffers in the memory interface to accumulate a full sword or two-sword of data before cycling memory. The priority unit also delays I/O read making I/O issue time independent of the read/write nature of the request. The priority unit immediately returns an accept, if necessary, even if it is delaying the issue of the request.

Bank Busy Checks

Because the memory busy time is four minor cycles, the priority unit issues a request to a particular bank of memory at intervals of no less than four minor cycles.

The priority unit conducts two bank busy checks termed the preissue check and the postissue check to prevent an immediate issue request from being honored if the request occurs during a bank busy.

When the priority unit accepts a delayed issue request, the banks of memory to be activated by that request are immediately reserved for three minor cycles. The preissue check compares the banks required by immediate issue requests against the banks reserved by pending delayed issue requests; a match prevents the immediate issue request from being honored.

The postissue check detects requests of a particular issue type attempting to reference a bank which has been referenced, within three minor cycles, by another request of the same issue type. An immediate issue request to bank X will be checked against bank X reference by other immediate issue requests within the preceding three minor cycles. Similarly, a delayed issue request will have its bank compared against the banks of other delayed issue requests arriving in the preceding three minor cycles. Detection by the postissue check of a bank conflict prevents a request from being honored.

A third check performed by the bank busy hardware is actually an address bus busy test. There is only one address bus from the priority unit to the memory interface, so if an immediate issue request follows a delayed issue request by four minor cycles, the immediate issue request will not be honored since it would require the address bus at the same time that the delayed issue request is utilizing the bus.

A single request may activate more than one memory bank. Since each successive half-word resides in a different bank, a word, sword, or two-sword request will activate 2, 16, or 32 banks of memory, respectively. If a write request specified a sword of data, then all 16 of the referenced banks must be found clear for both the preissue and postissue busy checks. Only those banks activated by a request are made busy, and only those banks required by a request are checked for busy.

Load/store (L/S) operands must be processed in the correct sequence. Any L/S request occurring in the three-minor cycle period following the initial request that was not accepted will be ignored by the priority unit.

Memory Interface Buffer Checks

The memory interface contains three read buffers and two write buffers for assembling and disassembling data. The scalar and vector processors control these read and write buffers. However, because a conflict can exist at read buffer three, the priority unit checks all read buffer three requests to ensure it will not be in conflict with a prior I/O request.

Memory Interface Signals

After an accept signal is honored by the priority unit, appropriate control signals are sent to the memory interface causing the requested data transfer to be performed. These control signals are grouped as follows:

- Buffer control signals - Provide the data buffer in the memory interface with information concerning the direction and quantity of data flow.
- Nine bank address bits - Define the lowest-numbered memory bank involved in the data transfer.
- Cycle memory signal - Causes the preselected memory banks to cycle.

Absolute Bounds Address

The absolute bounds address unit notifies the MCU of a memory reference (read or write) to a specified block of memory. The block of memory is specified by an upper bounds sword address and a lower bounds sword address. The addresses are absolute sword addresses. The bounds unit provides resolution to the sword level for a one or two-pipeline machine and resolution to the two-sword level for a four-pipeline machine.

There are two classes referenced: read and/or write requests and CPU and/or I/O requests (all non-I/O requests are CPU requests). None of the requests are mutually exclusive.

The MCU transmits the false state of the upper bounds limit and the true state of the lower bounds limit on two separate 24-bit trunks, DFW4 and DFW5 (refer to Maintenance Data Transfers in this section for a listing of the channel bits). The 24 bits correspond to address bits 35 through 58. Referring to the true state of both limits, bits 55 through 58 must always be zero. Also, bit 54 must be a zero if the CPU is configured for a four-pipeline operation.

An address is in bounds if it is greater than or equal to lower bounds, and less than upper bounds. Any bounds hit is latched until the occurrence of a master clear or an error clear from the MCU.

Retry Unit

All vector memory requests originate in the vector APL unit. When an APL request is not honored (no accept returned), it is the function of the retry unit to automatically reinitiate the request to the priority unit. The APL unit has seven separate requests which it transmits to the priority unit. They are: Read 1, Read 2, Read 3, RNS, Look-ahead, Write 1, and Write 2.

When an APL request is not honored, the retry unit immediately directs the APL unit to terminate the request stream. Because as many as three additional requests may arrive before the flow ceases, the retry unit contains a retry buffer with capacity to hold four requests, along with their control bits and addresses. In order to prevent operands from being processed in the wrong sequence, the retry unit directs the associative unit and priority unit to ignore all APL requests received during the three minor cycles after the initial unhonored request.

During the fourth minor cycle, the retry unit retransmits the initial request and bank address (via the load/store unit) to the priority unit, and transmits (via the load/store unit) the nonbank address to the associative unit. The remaining requests in the retry buffer flow out to the priority unit and the associative unit in successive minor cycles, unless the original request again receives no accept. In this case, the entire retry sequence will be repeated.

When the initial request eventually receives an accept, the request stream hold to APL is dropped, and the first new request arrives just after the last stored request is transmitted from the retry buffer.

After the initial request is honored, any subsequent request, including those in the retry buffer, may cause a retry sequence comprised of the request itself and any other requests arriving in the next three minor cycles.

When a virtual APL request cannot be mapped into absolute memory by the associative unit, no accept will be received and the retry sequence is initiated. Before the unhonored request is retransmitted the associative unit may inform the retry unit that a space table search is required. In this case, the retry sequence is suspended, and the address of the original request is locked into a register which presents that address to the associative unit (via the load/store unit) for the duration of the space table search. If a match is found in the space table, the retry sequence is resumed.

If an end of table is encountered in the space table before a match is found, the associative unit sends the retry unit an access interrupt signal, and terminates the space table search. The retry sequence is resumed for write requests only; no read requests residing in the retry buffer, or received later from APL, are processed. Also, if a B7 instruction is being executed, no write requests will be processed. This access interrupt mode of processing continues until all of the buffer busy lines from APL drop, indicating that the exchange operation is imminent.

RNS/BRANCH UNIT

The read next sword (RNS) portion of the RNS/BRANCH unit provides the control for loading the instruction stack. To maintain the instruction issue rate, a two-sword look-ahead is done by reading the two swords following the sword being executed. Issue of instructions is not blocked if the swords following the look-ahead are not in the stack.

The branch portion performs branch condition testing and executes the branch instructions. An address is maintained for each of the eight swords in the instruction stack, allowing out-of-the-stack jumps to be taken without voiding the stack. For example, it is possible to call a subroutine of up to three swords (48 instructions of 32 bits each) several times from a three-sword instruction stream and never jump out of the stack.

INSTRUCTION STACK

The semiconductor instruction stack provides the buffering for eight virtually addressed swords (512 bits), which can contain up to 128 32-bit instructions, 64 64-bit instructions or a combination of each. The instruction stack can contain up to six nonadjacent swords with two swords lookahead.

INSTRUCTION ISSUE UNIT

The instruction issue unit decodes all instructions and directs decoded vector/string instructions to the vector processor for execution. The instruction issue unit knows the length of scalar operations and schedules operands to and from the register file in the scalar processor. This is accomplished over a three minor cycle pipelined period.

The instruction issue unit issues instructions at a rate of one instruction per minor cycle, unless it is blocked by instruction or memory conflicts. The instruction issue pipe must resolve three register file conflicts:

Source operand conflict	An instruction requiring the result of a previous instruction as an input operand must wait until the operand is available in register file.

Output operand conflict	An instruction result, destined for the same register file location as a previously issued instruction must wait until the previous instruction stores its result into the register file, unless it also has a source operand conflict; then it will go at the shortstop time.†
Register file write conflict	An instruction result, arriving at the register file at the same minor cycle as the result of a previously issued but slower instruction, cannot issue.

To resolve these conflicts, 16 result address registers (RARs) hold the register file addresses for the output operands of previously issued instructions. Before an instruction is issued, its source operand addresses are simultaneously checked against all 16 RARs (source operand conflict) and its output operand address is checked against the operand result position timing chain (output operand and register file write conflicts) for possible conflicts. If a conflict exists, the issue is blocked until the conflict is resolved.

The instruction issue unit allows parallel operation of scalar and most vector/string instructions provided there are no register file reference conflicts and no central memory references made by the scalar instruction.

The register instructions 7D and B7/BA with G bit 7 set do not permit parallel operation. This parallel load operation requires two separate program address counters: one for vector/string instructions and one for scalar instructions. On interrupt, these counters are stored in the invisible package†† along with the operation code and G bits designator of the vector in process. The content of the scalar processor's current instruction register is also stored in the invisible package. This allows for program restart following an interrupt.

The parallel operation of the scalar and vector processors is controlled by the instruction issue unit as follows: when the instruction control unit in the scalar processor decodes a vector instruction and the vector processor is not busy, the scalar processor immediately supplies the vector processor with the decoded instruction function code and the contents of all register file locations per the instruction descriptors. As soon as the vector processor is not busy, it begins to process the vector instruction and releases the scalar processor. The scalar processor reserves any register file locations that the vector instruction may want to use as index, field length, and so forth, and continues with the next instruction in the instruction control unit.

Table 2-1 indicates which instructions are executed in the scalar processor and which in the vector processor.

---

†Shortstop is defined in this section under scalar floating point.

††The invisible package contains the address and control information necessary to begin a new job or to continue a job interrupted during execution in job mode. Refer to section 5 for description.

TABLE 2-1. SCALAR/VECTOR PROCESSOR INSTRUCTION RESPONSIBILITY

First Digit of Instruction Code	Second Digit of Instruction Code															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	S	I	I	S	S	S	S	I	V	V	V	I	S	S	V	S
1	S	S	S	S	V	V	V	I	I	I	I	I	V	V	V	V
2	S	S	S	S	S	S	S	S	V	I	S	S	S	S	S	S
3	S	S	S	V	S	S	S	V	S	V	V	V	S	S	S	S
4	S	S	S	I	S	S	S	I	S	S	I	S	S	S	S	S
5	S	S	S	S	S	S	S	I	S	S	S	S	S	S	S	S
6	S	S	S	S	S	S	S	S	S	S	I	S	S	S	S	S
7	S	S	S	S	S	S	S	S	S	S	S	S	S	V	S	S
8	V	V	V	V	V	V	V	V	V	V	V	V	V	I	I	V
9	V	V	V	V	V	V	V	V	V	V	V	V	V	V	I	I
A	V	V	V	I	V	V	V	I	V	V	I	V	V	I	I	V
B	S	S	S	S	S	S	S	V	V	I	V	V	V	V	S	S
C	V	V	V	V	V	V	V	V	V	V	V	V	V	S	S	V
D	V	V	I	I	V	V	I	I	V	V	V	V	V	I	I	V
E	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
F	V	V	V	V	V	V	V	V	V	I	I	I	I	I	I	I

S Executed within the scalar processor. (Note that data flag information is passed to the data flag register in the vector processor for appropriate instructions.)

V The scalar processor initiates the vector processor to execute portions (or all) of the instructions.

I Illegal instruction (processed by scalar in monitor mode, by vector in job mode).

ASSOCIATIVE UNIT

The associative unit provides the page table virtual addressing mechanism consisting of 16 associative address registers and a space table extension (located in a restricted area of central memory). Virtual addressing converts a logical address to an absolute storage address to allow programs to appear logically contiguous to the user while being physically not contiguous in the storage system.

The page table is an ordered list of the associative words necessary to define the pages in absolute memory. The page table uses a last used push down algorithm, thus the most recently used associative words are at the top of the table. The space table is an extension of the page table containing the associative words necessary to define pages in memory that have not been in recent use. The associative unit is capable of comparing the associative registers in one minor cycle and the space table entries at the rate of two entries every minor cycle.

For the user, the paging mechanism and the operating system software permit the most active portions (pages) of a user program to reside in the central memory. These program portions can reside in nonadjacent areas of the central memory. The virtual addressing facility, through the page table, makes these areas of memory appear to be adjacent. The paging mechanism ensures that a large number of users can have simultaneous access to the central computer with minimum page swapping overhead.

Searching the Page Tables

The 16 associative registers (ARs), labeled 00 through 15, are each one word in length. They are loaded from absolute bit addresses 4000_{16} through $43FF_{16}$ (word addresses 100_{16} through $10F_{16}$) of memory by a load AR (0D) instruction. They can also be stored into the same absolute addresses by a store AR (0C) instruction.

The associative words in the ARs are moved dynamically using the following scheme. Whenever a virtual address is presented for association and a match (hit) is made, the content of the AR containing the hit is moved to the top AR (AR00). Simultaneously, the content of each AR, from AR00 to (but not including) the hit AR, is moved down one AR (for example, 00 to 01, 01 to 02, 02 to 03, and so on). Thus, the associative words in AR00 through AR15 are in descending order of most recent use. If the end-of-table (END) is contained in the ARs and no hit is made, the contents of the ARs remain unchanged and access interrupt is taken, unless the request is for a read-ahead sword of instructions negated by the branch. Whenever an address is presented with no hit made and no END is contained in the ARs, a search through the space table is begun using a ripple method. Each AR from AR00 through 14 is moved down one AR and AR15 is placed in a buffer register. A NULL (vacant location, no entry) is placed into AR00 and then AR00 through AR15 are stored in memory locations 4000_{16} through $43FF_{16}$. The content of the space table is rippled through the ARs. The first associative word of the space table is read and examined; its spot in memory is filled by the old content of the buffer register (AR15). If the first word read from the space table is not a hit, the second word is read, is replaced in memory by the first word read, and so on, until a hit is made or an end-of-table is reached.

When a hit is made, the content of the hit address is temporarily stored in the buffer register and is replaced in memory by the associative word which precedes it in the space table. The contents of locations 4000_{16} through $43FF_{16}$ are loaded into the ARs, and the content of the buffer register (content of the hit address) is transferred to AR00. Entries in the space table beyond the hit address are not modified.

If an end-of-table is read before a hit is made, the entire space table, including the sword containing the END, is pushed down by one word position, and a NULL is placed in AR00. If the unsuccessful search was initiated by a memory reference in job mode, the NULL may be pushed out of AR00 before the exchange to monitor mode is performed. This unsuccessful search condition and the cause bits are sent to the main control and an access interrupt results.

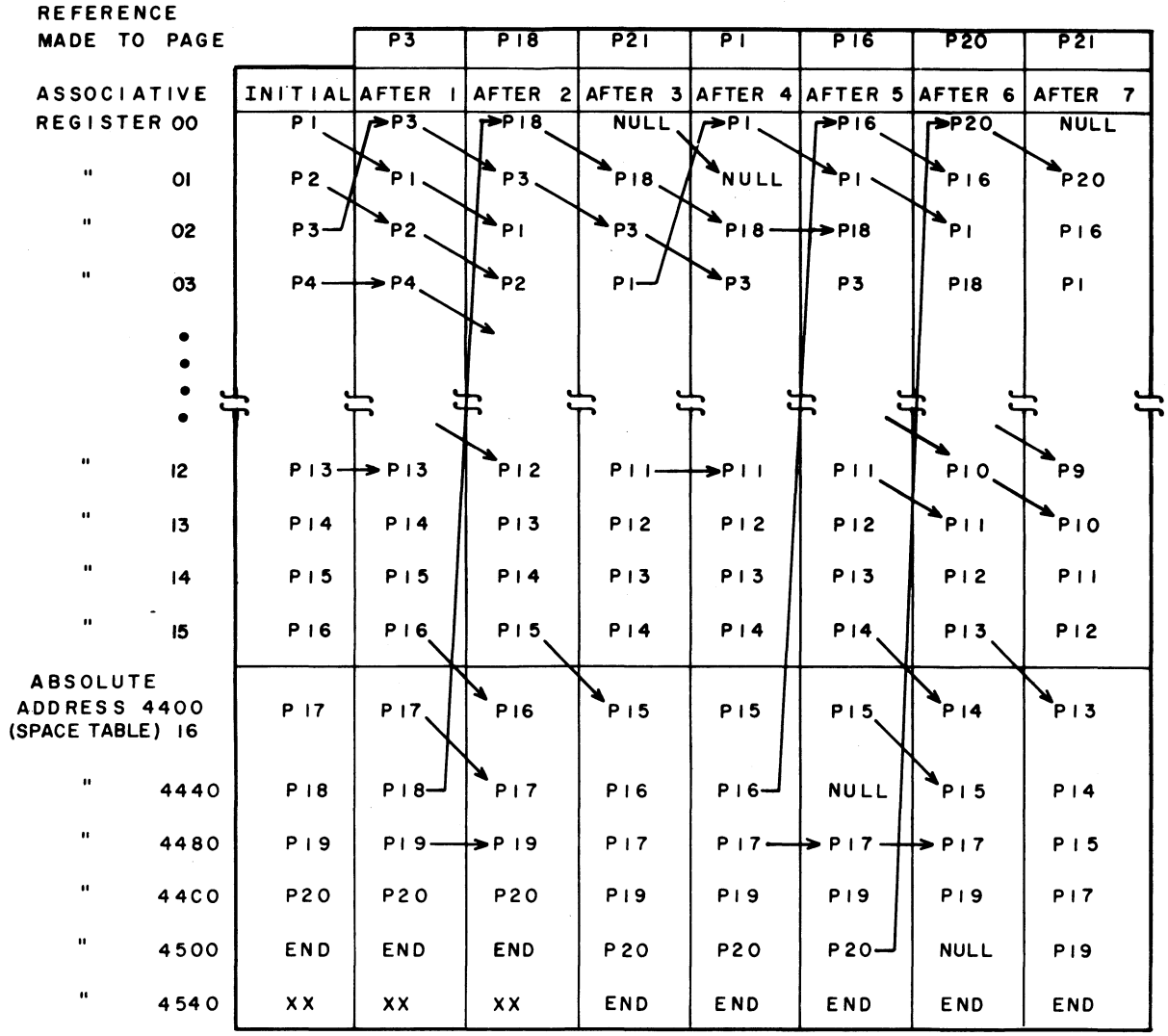
If a NULL exists in the ARs and no hit is made in the ARs, the space table is not pushed down. A read and compare takes place until a hit is made and the NULL replaces that word in the space table.

If a hit is not made in the ARs and a NULL is encountered in the space table, the operation changes from a ripple to a read only (no push down); if no hit is found, the NULL remains in AR00, as before. If a hit is made deeper in the space table, the NULL replaces it. Only one NULL need exist at any given time in the page table.

If the monitor sets up the page table with one NULL, and it does not add or delete a NULL, the END remains at a fixed address for any given number of associative words in the page table.

At the termination of an unsuccessful space table search, there is a NULL in AR00 if the unsuccessful search was initiated by a 0F (load keys, translate address) instruction.

Figure 2-2 is an example of a page table search. The contents of the ARs and the contiguous entries in the space table are depicted as P1, P2, and so on. NULL and END, where P1 represents the associative word for page 1, NULL is a NULL associative word, and END is an end-of-table entry.



NOTE: 1. PAGE TABLE IS MADE UP OF ASSOCIATIVE REGISTERS AND THE SPACE TABLE.

3AP6A

Figure 2-2. Page Table Search Examples

The example shows seven consecutive virtual address page references and the resulting page table transfers. Assume that there are 21 associative words in the page table (16 in the associative registers and 5 in the space table) and that no lockout bits are set; the last entry is an end-of-table.

1. The first reference is to page 3. P3 is in AR02 and is moved to AR00; the content of AR00 through AR01 is moved down one word. The space table is not altered.

2. The next reference is to page 18. No hit is made in the ARs so the ARs are pushed down one and the content of AR15 (P16) is pushed down into the space table. P17 is read and replaced with P16. Since P17 is not a hit, it is swapped with the next entry in the space table, P18. P18 caused a hit so it is replaced by P17 and moved to AR00.

3. The third reference is to P21, which is not in the page table. The result is that the entire page table, including the END, is examined and pushed down, AR00 is set to a NULL, and an access interrupt is generated.

4. Assume that the access interrupt is properly handled by the monitor program and the page table is not altered. The next storage reference in job mode is to P1. Since P1 is in AR03 when the reference is made, it is moved to AR00, and AR01 through AR02 are moved down one word.

5. The fifth reference is to P16 which is now the second entry of the space table. This time there is a NULL in the ARs. The NULL is moved to AR00 and AR00 is moved down one word. P14 is not moved into the space table and the space table is not pushed down. A read and compare takes place until the hit is found; the NULL then replaces the selected associative word in the space table.

6. The next reference is to P20. Since there is no hit or NULL in the ARs, the page table is pushed down until the NULL is encountered. Push down ceases and read and compare takes place until P20 is read, causing a hit. P20 is moved to AR00 and is replaced by a NULL.

7. The last reference is to P21 which is not in the page table. The page table is pushed down until the NULL is encountered. Push down and searching cease when the END is read.

   AR00 is set to a NULL and an access interrupt is generated.

For page table restrictions and requirements, refer to section 5.

Multiple-Match Fault

In the central computer, any given combination of lock and virtual page identifier in an associated word may occur in only one associative word in the page table. Whenever a violation of the rule is detected, a multiple-match fault occurs and the CPU is stopped. When two keys are identical, their lockout bits must be the same (refer to section 5 for a description of locks and keys). Otherwise, a reference to the differing lockout bits generates a multiple-match fault, resulting in an undefined condition. There are two types of multiple-match faults.

1. One virtual address, lock, and key matches more than one register in the associative registers.

2. A virtual address makes a successful match with the associative registers, and at least one additional match combination exists, but the reference is locked out by the key lockout bits.

LOAD/STORE UNIT

The load/store (L/S) unit accepts addresses and transfers data between its registers and main memory. The L/S unit provides orderly buffering and execution of the load and store instructions; 12, 13, 32, 5E, 5F, 7E, and 7F. Six address registers in the L/S unit enable requests to be stacked and executed in the proper order. The load instructions 12, 5E, and 7E require one register and can be executed (with no memory conflicts) at a rate of one load per minor cycle. The store instructions 5F and 7F require two address registers and can be executed at one store per two minor cycles. The 13 and 32 instructions require two address registers which are busy for 17 minor cycles after selection.

The L/S is capable of streaming L/S instructions (other than 13 and 32) at one minor cycle per load and two minor cycles per store assuming no memory busy, access interrupt, or register file write bus busy conflicts exist. For example, a stream of n loads executes in n+14 minor cycles from the issue of the first load until the operand from the last load is available in the register file. A stream of n stores executes in $2n + \left\lfloor \frac{n-1}{3} \right\rfloor$ minor cycles from issue of the first store until issue of the last store.

REGISTER FILE

The register file of the central computer contains 256 64-bit words. This register file is capable of accomplishing two read operations and one write operation every minor cycle.

A scalar result written into the register file can be used by subsequent scalar instructions before the result is available in the register file when the read and write addresses are equal. The scalar result bypass of the register file occurs at the same time the result is written into the file.

The scalar arithmetic result for one scalar instruction is often used as an input operand for the next scalar (arithmetic) instruction. A special data path (shortstop) between the output and input areas of the scalar arithmetic unit permits immediate use of an arithmetic result operand prior to it being written into the register file.

SCALAR FLOATING POINT

Scalar floating point performs all nonvector arithmetic and logical operations in the CYBER 205. Scalar floating point contains five arithmetic units. The following table lists each unit and the time (in minor cycles) required to produce a 32- or 64-bit result.

Unit	Time (Minor Cycles)
Add/Subtract	5
Multiply	5
Logical	3
Single Cycle	1
Divide/Square Root/Convert	21-54

All times listed are shortstop times. Shortstopping saves time by making it unnecessary to store a unit's result in the register file before the result is used in the next arithmetic operation. Instead, the result is returned directly to the input of any of the arithmetic units (the result is also stored in the register file).

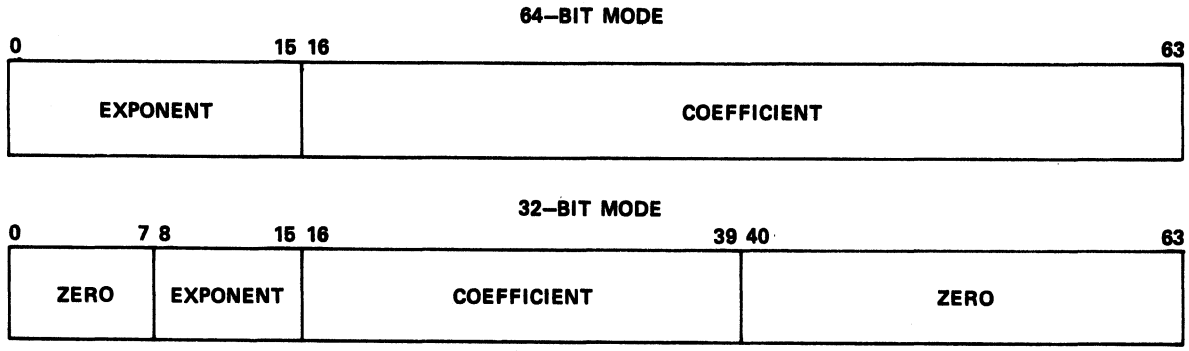
The first four units listed above are segmented, and each segment is independently controlled by microcode. Microcode data bits are read out from the scalar floating-point microcode memories and transmitted through timing chains in parallel with the movement of the operands through the segments of the arithmetic unit. This allows one arithmetic unit to accept a new pair of operands, and to issue a result of a previous pair of operands every minor cycle. The divide/square root/convert unit is not segmented and can accept operands only when it completes the previous operation.

Scalar Floating-Point Unit Control Interface

There are three input and two output trunks to the scalar floating-point unit. All input operands are 64- or 32-bit floating-point quantities, except where otherwise specified. If an indefinite or machine-zero floating-point operand is received, the coefficient is set with zeros.

A-Input Trunk

This 64-bit trunk receives data bits from register location R in the following format.



B-Input Trunk

The B-input trunk, identical to the A trunk, receives data from register S.

Control Trunk

The control trunk carries the signals that control the scalar floating-point unit. It is composed of the following signals.

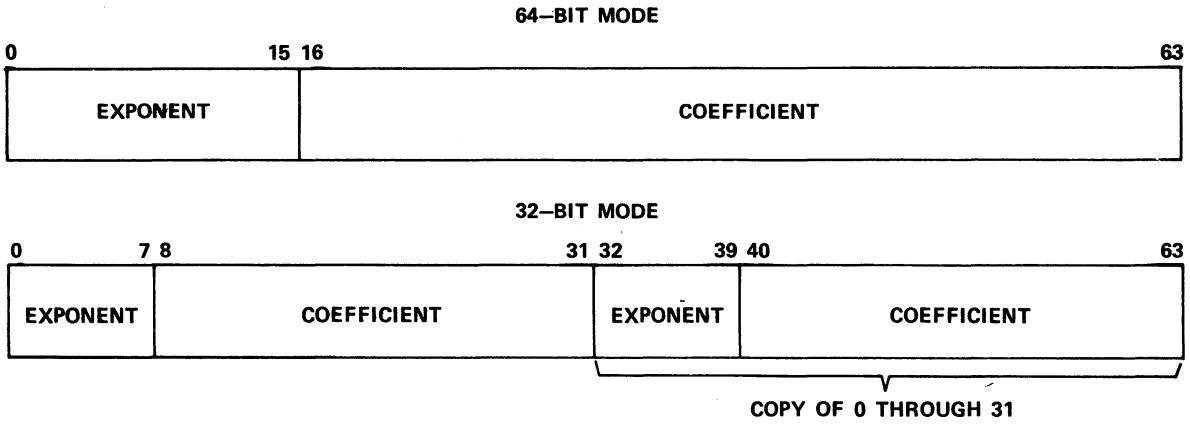
Control Address - The control address bits select the set of internal control signals for the floating-point instruction being executed. A set of unique codes exist for each instruction (refer to table 3-4). Using the input data to the floating-point unit as a reference, these control bits must arrive at the floating-point logic 1.5 cycles before the data and must be valid for 20 nanoseconds.

Mode Controls - The mode controls are Mode 64 In, Mode 64 out, G-bit, and Divide. The Mode 64 and G-bit signals must lead the input data by one minor cycle and the Divide signal must lead by 1.5 minor cycles.

Issue Controls - The issue controls are Shortstop, R-shortstop, S-clockgate, R-clockgate, S-shortstop Enable, R-shortstop Enable, and Go. All these controls must be valid one minor cycle before the data. Shortstop is the process by which a result from any arithmetic unit may be returned directly to either input of any arithmetic unit. The shortstop enable signals enable the setting or clearing of the shortstop control flip-flops. The clockgate signals cause data to be clocked into the floating-point input registers. The Go signal allows processing of operands in the input registers.

Output Trunk

The output trunk is 64 bits and transmits output data to the stream unit. Data remains on this trunk for one minor cycle. The formats for the output trunk are as follows:



Output Control Trunk

The output control trunk transmits control or fault bits associated with results generated by the scalar floating-point unit. These signals come up with data and are held up for one minor cycle. The following signals are transmitted on the output trunk.

Signal	Meaning of a 1 on Signal Line
Branch Condition Met	The operands meet the compare condition. This line is zero when a compare is not being done.
Exit Condition Met	The operands do not meet the compare condition. This line is zero when a compare is not being done.
Divide Timing Pulse	Divide operands follow this timing pulse by 14 cycles.
Divide Unit Busy	The divide unit cannot accept new operands during the time this signal is 1.
Data Flags 39, 41, 42, 43, 45, 46	Refer to appendix D.
Data Flag 58	A divide, square root, or convert operation occurred and resulted in data flag 39, 41, 42, 43, 45, or 46 being set.

SCALAR PROCESSOR MICROCODE MEMORIES

The central computer uses microcode memories to start and control the execution of all intructions. The scalar processor contains five microcode memories: scalar microcode memories PM00 and PM01; associative microcode memory HM00; and floating-point and divide microcode memories DM00 and GM00. Each memory operates independently during CPU instruction execution and is addressed simultaneously during writing or sweeping operations. The MCU loads the microcode memories via a block transfer. All microcode memories operate at the computer clock cycle rate.

Scalar Microcode Memories (PM00, PM01)

Scalar microcode (SMIC) is composed of two memories: PM00 and PM01. Both memories operate simultaneously, and each memory contains 256 120-bit words.

SMIC memory is a read-only memory; writing into SMIC is reserved for loading systems or diagnostic microcode programs. SMIC provides the starting address for SMIC, FMIC, DMIC, and AMIC during the load operation.

SMIC Operation

When the instruction stack has an instruction ready for execution, the function (F) code is sent to the PM00 address register. If the issue unit is ready to execute an instruction, the SMIC output is switched to PM00 and the execution is started.

If the instruction has one cycle of issue, SMIC output remains switched to PM00 and the next instruction begins execution (assuming the instruction stack has the next instruction available).

If the instruction has multicycles of issue, SMIC output is switched to PM01 where the remaining cycles of that instruction are executed. When the remaining cycles are completed, SMIC output switches back to PM00 and the next instruction begins execution (assuming the instruction stack has the next instruction available).

If the instruction has variable cycles of issue (for example, vector processor instructions, some of which execute in the associative unit, and so on), SMIC output is switched to PM01, and the remaining cycles of SMIC control are executed. When PM01 has completed its functions, it waits for the conditions indicating the end of the operation and switches to PM00 to execute the next instruction.

SMIC controls the flow of data from the instruction word to the functional unit. For example, SMIC:

- Selects designators to and from their points of use (register file read address, register file write address, address adder inputs, and so on).
- Selects register file data to functional areas, such as scalar pipeline and address adder.
- Selects register file data, such as address and field lengths, to the vector processor.

SMIC also controls the operations performed by other functional units. For example, SMIC:

- Provides starting addresses for scalar floating-point microcodes FMIC and DMIC.
- Informs load/store unit which operation to perform.

SMIC Address Control

PM00 addresses are controlled by the instruction stack.

PM01 addresses are controlled by SMIC bits. The next address to be used can be:

- The next sequential address (via incrementer).
- The address contained in the M01 field.
- An address made from the M01 field (most significant 4 bits) and an index based on sense condition status (least significant 4 bits).

SMIC Parity

SMIC has five parity bits forming odd parity.

Associative Microcode Memory (HM00)

The associative microcode (AMIC) is a 256-word by 96-bit memory.

AMIC Operation

AMIC is active during the following associative operations.

- Space table search.
- Load associative registers (0D instruction).
- Store associative registers (0C instruction).

The AMIC memory is initialized into an idle loop and waits for a load, store, space table search, or an exchange operation. The memory supplies control to the associative registers (ARs), branches on conditions from ARs, and returns to the idle loop upon completion of an operation.

AMIC Address Control

AMIC bits control HM00 addresses. The next address to be used is one of the following:

- The starting address.
- The address from the AD1 field.
- The address from the space table mode address register.

AMIC Parity

AMIC has two parity bits forming odd parity.

Floating-Point and Divide Microcode Memories (DM00, GM00)

The floating-point microcode (FMIC) and divide microcode (DMIC) control scalar floating-point pipeline segment operations and iterative operations such as divide, square root, and BCD/binary conversion.

The floating-point microcode memory (DM00) and the divide microcode memory (GM00) contain 256 48-bit memory words each. Both memories are read-only memories. Writing is reserved for loading systems or diagnostic microprograms.

The main functions of FMIC and DMIC are as follows:

FMIC — Selects data paths for operand processing.

Provides constants for exponent correction and coefficient shifting.

Enables hardware checks for end case conditions such as machine zero operands, overflow conditions, and so on.

DMIC — Selects data paths for operand processing.

Preconditions logic properly for divide, square root, and BCD/binary convert algorithms.

Indicates that the divide unit is busy processing operands and enters the RAR number into the result timing chain when results are available.

FMIC Operation

FMIC receives its address from an 8-bit field (M02) in scalar microcode memory. If an instruction requires the scalar floating-point unit, the issue unit causes one 48-bit microinstruction word to be read from FMIC. This word controls the segments of the floating-point pipeline as the operands are processed. Floating-point operations are initiated only when result output bus conflicts cannot occur.

DMIC Operation

If the floating-point operation is divide, square root, or BCD/binary conversion, DMIC microcode memory is used with FMIC to control the iterative segments of the pipeline that perform these operations. The 8-bit field (M02) sent from SMIC to the floating-point unit is used for the starting address. Each iterative operation controlled by DMIC requires the execution of several microinstructions. There is a field in each DMIC microinstruction (GMA) that points to the next microinstruction. This linkage continues until the last microinstruction required is completed. The GMA field of the last microinstruction points to location 0 of DMIC, a one instruction idle loop. DMIC remains in this idle loop until the next divide, square root, or BCD/binary conversion instruction is received, at which time a new starting address is received from SMIC.

VECTOR PROCESSOR

The vector processor executes most multioperand instructions in parallel with the scalar processor and certain scalar operations. The vector processor processes data at the rate of four words, two words, one word, one half-word, or 16 bits per minor cycle, depending on the type of operation. The vector processor consists of the following sections (refer to figure 2-3).

VSU - Vector Setup and Recovery Control

APL - Stream Addressing Pipeline

VST - Vector Stream Inputs

VSW - Vector Stream Output

FPL - Floating-Point Pipeline

VSS - String

IOM - Input-Output Ports and Maintenance

VECTOR SETUP AND RECOVERY CONTROL (VSU)

The vector setup and recovery control unit controls the execution of all nonscalar processor instructions including vector instructions, string instructions, data flag register instructions, job and monitor interval timer instructions, and real-time clock instructions. The VSU unit contains three microcode memories to control the VSU hardware. They are the VSU microcode, the VEX microcode and the VSC microcode. For a description refer to VSU microcodes later in this section.

The VSU unit is divided into two sections, setup and execute. Figure 2-4 shows the VSU block diagram. The setup section calculates all starting addresses, extension field length and an overall field length. As soon as the execute section is able to take on data for another operation, a continue is sent back to the scalar processor which can now proceed to the next instruction. When the execute section of VSU takes over an instruction, it is in control until completion of the operation. The execute section sends out decode and control information to the rest of the units in the vector processor. Along with starting address and lengths being sent to the APL units, the overall length is transmitted to VST.

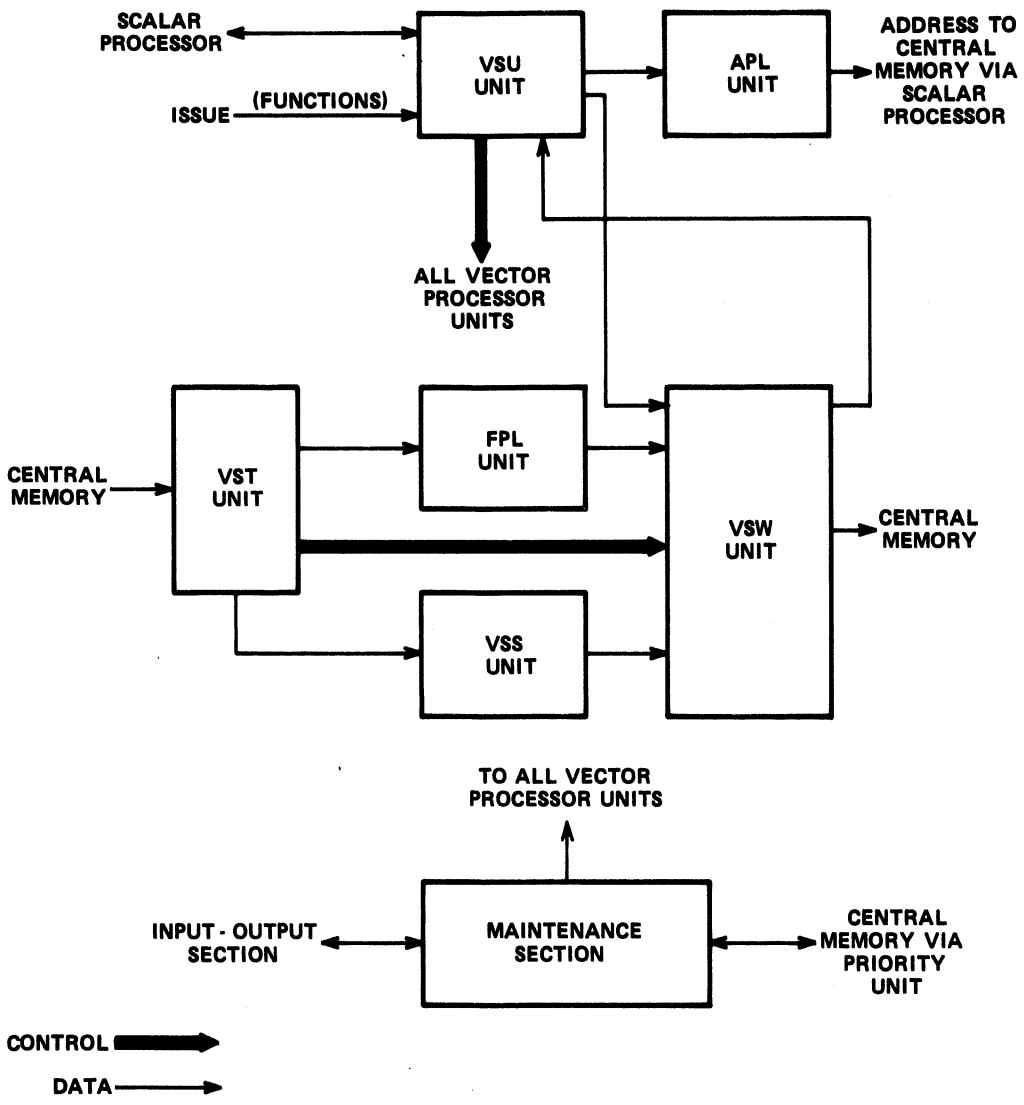


Figure 2-3. Vector Processor

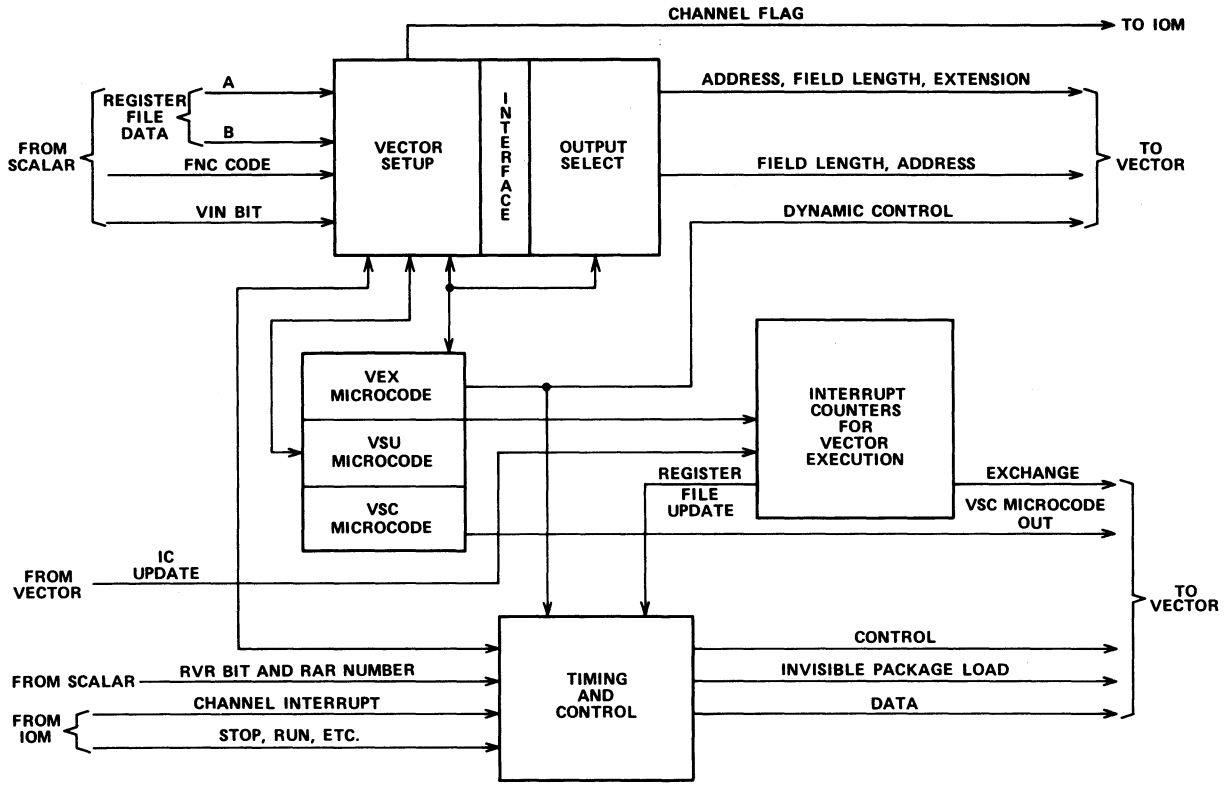


Figure 2-4. VSU Block Diagram

The VSU unit contains nine interrupt counters that are incremented/decremented during a vector processor instruction execution. The counters contain the current field lengths, addresses, and broadcast or extension data required to restart the instruction execution after an interrupt.

Inputs to VSU

Upon translation of a vector processor type of instruction, the scalar processor issue/decode unit immediately transmits the function code and required register contents from the register file to the VSU hardware in the vector processor. Machine zero or all zero extension data required by the instruction is included in the transfer. The information is transmitted through two 64-bit buses from the register file fanout to two sets of input registers in the VSU hardware. The register file data transfer occurs two registers at a time for up to six cycles depending upon the type of instruction. If broadcast is specified by the instruction, then the appropriate broadcast data is transferred in lieu of the extension data. The VSU hardware forces normalized-one extension when applicable. During an instruction execution, all broadcast data is treated as extension data by the vector hardware.

The issue/decode unit controls the transfer of data into the VSU input registers by means of the vector increment (VIN) control line. The VIN bit is sent for each cycle of register data transfer to the VSU. The VIN bit controls the input register addressing and gates the function code into the VSU function code register (SRFC) and, using the function code as a starting address, instructs the VSU microcode to begin operation.

The format of registers transferred from the scalar processor to the vector processor VSU input registers is instruction dependent. Registers that are to be updated at the completion of the instruction execution are all preread to assure their availability at update time.

Depending upon the instruction, there may be one, two, three, five, or six cycles of register transfer to the vector processor. A typical 80 instruction uses six cycles to transfer all the required register file data with the broadcast or extension data transfer occurring during the fifth and sixth cycle. In order to provide some standardization and to minimize decode unit microcode requirements, the register transfers for some instructions are padded out to five or six cycles. In the case of an F0 instruction, registers C and Z are transferred twice to permit use of a common routine.

The scalar processor issues instructions to the VSU unit without waiting for instructions already in progress to be completed. It conflict checks, reads, and transmits register file registers to the VSU unit at a rate of two every minor cycle until all descriptor data is transmitted. The scalar processor sends function data and a timing bit (VIN), then waits for a continue (release) from VSU.

VSU Operation

Most vector processor instructions process data using two data input streams, one control vector input stream, and one data output stream. The instructions specify these streams in terms of field lengths, base addresses, and offsets or indexes. All address and field length calculations occur under VSU microcode control. All of the VSU microcode control bits for one operation are read at the same time and applied through delay chains as required.

Vector Setup

To generate the field length used for instruction execution, the offset is subtracted from the field length. Field length calculations that produce negative results or results greater than a 16-bit count are forced to zero. The correctly shifted offset or index, is added to the base address to generate the address used for instruction execution. The field length and address are combined and loaded into the selected output register.

The VSU microcode checks field lengths and addresses for instruction no-op or instruction illegal. The VEX microcode takes appropriate action as required. The VSU microcode selects the field length, base address, and offset for a calculation from the appropriate input registers.

All field lengths and addresses that go to the APL, VSS, and VST units must pass through the output registers. Multipass instructions and restart of a vector processor instruction after an interrupt both require transfer of the interrupt counter register contents into the output registers. A 64-bit data path from the interrupt counters is used for this transfer.

Vector Termination

The scalar processor, upon sending an instruction to the vector processor, waits to be released by the vector processor. The vector processor sends a release code (REL) and a full bit (SMIC-GO) to the scalar processor, and a signal to increment the current instruction address register (CIAR).

Most vector processor instructions release the scalar processor for parallel operation as soon as the instruction is interruptible. The 7D, B7, and BA instructions with G bit 7 set release the scalar processor when instruction execution is complete. These instructions actively use the register file during instruction execution; parallel operation is not permitted.

Some vector processor instructions require the update of an index, or the storing of an arithmetic result, in register file at the completion of an instruction. The scalar processor, upon receiving the release code, reserves the appropriate register file location(s) for the vector processor instruction. The vector processor records the RAR number(s) for later transmission to the scalar processor result timing chain.

The release (REL) code sent by the vector processor to the scalar processor specifies which instruction designators are to be used for reserving register file locations.

The termination of most vector instructions occurs in two steps: the data inputs go empty and the output buffer goes empty. For most vector instructions, the VEX microcode status becomes not busy when all input data is used. VEX microcode can begin processing the next instruction as soon as it is not busy. However, the output for one vector instruction must be complete through APL output deactive before VEX microcode initiates the output for the next instruction.

Interrupt and Branch Control

The vector processor contains the hardware to recognize conditions that cause an interrupt or branch. The vector processor sends the indicated GET code to the scalar processor which responds accordingly.

An interrupt operation occurs in job mode while a branch operation can occur in either job mode or the monitor mode.

A branch operation, once begun, yields only to a branch with a higher priority. Similarly, an interrupt operation yields only to an interrupt of higher priority. The occurrence of a simultaneous branch and interrupt results in an interrupt. An interrupt is always completed regardless of the time delay. A branch, once recognized, is rejected after a 25-cycle delay. The branch or an interrupt can then be recognized again.

Interrupt Counters

The vector processor contains interrupt counters to record current field lengths and addresses of an executing vector processor instruction. Upon interrupt, the interrupt counter contents are stored in the invisible package for that job. The interrupted instruction restarts from the point of interrupt using the interrupt counters when the job is reloaded.

There are nine 64-bit interrupt counters in the vector processor (IC0 through IC7 and ninth IC). IC0 through IC5 each have a 16-bit adder for field length update of bits 0 through 15 and a 48-bit adder for address update of bits 16 through 63. The IC update consists of an update count and full bit from the VSS, VST, or VSW units. VEX microcode also can update the ICs. The VSC microcode enables the increment/decrement control on each of the ICs per the instructon requirements. IC6 and IC7 do not have update capability, but are used for extension data or other instruction related constants. The ninth IC is used only for the AX, CC, and C8 through CB instructions.

Timers

The monitor interval timer, job interval timer, and real time clock are located in the vector processor. The VEX microcode is required to load or store any of these timers.

The three timers all operate at a 1-MHz rate, but each timer has its own 1-MHz timing clock. The 1-MHz timing clock for the monitor interval timer is synchronized (set to zero) when the 0A instruction load occurs. The job interval timer operates similarly.

Real Time Clock

The real time clock is a 47-bit (with a positive sign bit for a total of 48 bits) counter incremented at a 1-MHz rate. The clock runs continuously and cannot be cleared. The clock time is sampled with the 39 instruction which stores it in the register file.

Monitor Interval Timer

The monitor interval timer is a 32-bit counter loaded from the register file by the 0A instruction in monitor mode. It is decremented at a 1-MHz rate, and causes an external interrupt when counted down to zero. No interrupt occurs when it is loaded to zero.

Job Interval Timer

The job interval timer is a 32-bit counter loaded from the register file by the 3A instruction in job mode, and stored into the register file by the 37 instruction. The counter is decremented at a 1-MHz rate, and sets data flag bit 36 when counted down to zero. The data flag bit does not set when the job interval timer is loaded to zero.

Data Flag Register and Control

The data flag register is a 48-bit register containing hardware status bits with associated mask bits, free data flag bits, and miscellaneous bits. The data flag register is set, stored, or swapped with a register file location(s) by the 3B instruction. The 33 instruction can set, clear, or toggle a selected data flag register bit or can cause a conditional branch depending upon the state of a selected bit (or both). Both instructions require the use of VEX microcode for instruction execution. VEX microcode releases the scalar processor with either a branch or no-branch GET code for both instructions.

The data flag register can produce automatic branch operations, in both job and monitor modes, when properly enabled.

Table 2-2 defines the bits in the data flag register.

TABLE 2-2. DATA FLAG REGISTER

Bit	Assignment/Description
0-15	Product bits that are automatically set when the corresponding mask and hardware status bits are both set. Example: Bit 4 is set when both bits 20 and 36 are set.
16-31	Mask bits for bits 32 through 47 respectively.
32-34	Undefined bits. Must be set to zero.
35	Soft interrupt. The operating system can set this bit in the job's invisible package.
36	Set by the job interval timer counting down to zero.
37	Selected condition not met (C0 through C3 and CC instructions).
38	Not used and must be set to zero.
39	The 10 instruction binary exceeds 48 bits.

TABLE 2-2. DATA FLAG REGISTER (Contd)

Bit	Assignment/Description
40	Inclusive OR of bits 37, 38, and 39.
41	Floating-point divide fault.
42	Exponent overflow.
43	Machine zero result.
44	Inclusive OR of bits 41, 42, and 43.
45	Square root result imaginary.
46	Indefinite result.
47	Breakpoint compare occurred.
48-50	Undefined. Must be set to zero.
51	Dynamic exclusive OR of all the bits in the product field (bits 0 through 15).
52	Enable bit for an automatic data flag branch (ADFB). An ADFB can occur when both bits 51 and 52 are set. Bit 52 is cleared automatically when an ADFB is processed.
53-55	Free data flags used by several instructions to indicate the result of the instruction. Every instruction using these bits clears them prior to selectively setting any of them. These bits are sampled with the 33 instruction.
56	Not used and must be set to zero.
57	Not used and must be set to zero.
58	A scalar divide, square root, or convert fault occurred.
59	Vector processor floating-point divide fault.
60	Vector processor exponent overflow.
61	Vector processor machine zero result.
62	Vector processor square root result imaginary.
63	Vector processor indefinite result.

Data flag bits 41, 42, 43, 45, and 46 are set from both the scalar and vector arithmetic units. Data flags 59-63 are corresponding bits set only from the vector arithmetic unit.

VSU Microcodes

The VSU Unit contains three microcode memories to control the VSU hardware for address and field length calculations, for instruction start and execution monitoring, and for function decode purposes.

VSU Microcode

VSU microcode (Vector Setup) - Performs address and field length calculations. The VSU microcode is 1024 addresses by 120 bits.

VEX Microcode

VEX microcode (Vector Execution) - Provides execution monitoring. The VEX microcode is 1024 addresses by 120 bits.

VSC Microcode

VSC microcode (Vector Static Control) - Provides microcoded function decode bits to all vector units for vector instruction execution. The VSC microcode is 512 addresses by 96 bits.

STREAM ADDRESSING PIPELINE (APL)

The APL unit's function is to manage the memory to maintain a maximum flow of data between memory and the vector processor. The APL unit takes the address and length parameters and redefines them on a sword or two sword basis, then along with control information (such as starting quarter/half sword, element, and so forth) makes memory references via the priority and associative units. The read data and control information is then transmitted to the VST unit after the memory cycle.

The APL unit is physically located in the vector processor. It schedules the memory and register file for data used during the operation of the nonscalar instructions. A list of these types of operations is as follows:

- A - vector, sword/two-sword input on read one.
- B - vector, sword/two-sword input on read two.
- C - vector, sword/two-sword output on write one.
- CS - string, sword output on write two.
- Z - string, sword input on read three (first request on R2).
- CSLA - small page, look-ahead for CS output.
- CLA - small page, look-ahead for C output.

- X - string, sword input on read three.
- Y - string, sword input on read three.
- Random load, word/half-word input on read one (BA instruction).
- Random store, word/half-word output on write one (B7 instruction).
- First/last old data, half-word input on read one.
- RNS, sword input on read three.

During the operation of a general vector/string instruction, all the above types of input, outputs, and look-aheads are in operation together depending on instruction types. CSLA, CLA, and the first/last old data operations occur only once or twice per instruction execution where applicable.

The type of operations used by a particular instruction are determined by the VEX microcode of the VSU unit. For each of the types of operations necessary in a particular instruction, a unique control word is received by APL. The control word contains all the information needed to define the operation of the particular type of input/output during the entire instruction (or entire pass of multipass instructions). A control word contains the following:

- Virtual address - 48 bits.
- Termination field length - 22 bits.
- Extension field length - 22 bits.
- GO active code - 9 bits.
- Number of control lines:
  - Request size.
  - Operand size.
  - Register file address (consecutive).
  - Delta, field length, and address for random load/store.
  - Field length equal zero or infinity.
  - Right to left addressing.
  - Override output before input.
  - Override read lockout.
  - Force abort.

Stream Input Operation

For each input setup, there is a buffer in the VST unit for the data to be entered. Associated with each buffer is a counter in APL which counts quarter swords requested by counting up from zero. It counts quarter swords used by counting down as quarter swords are removed from the buffer for use. The size of the buffers are such that they can hold all the data requested but not used. The APL unit starts entering requests for an input at a maximum rate of one for every four cycles for sword/two-sword or one every cycle for word/half-word. The maximum rate is reduced by competition for the same bank of memory and by competition for the same request phase time. If the counter shows the buffer is full, then the requests must wait.

Stream Output Operation

Similar to the inputs, each output setup has a buffer in VSW and a counter in APL. APL sends a request for each sword/two-sword, word/half-word in the buffer/counter. Requests run at a maximum of one for every four cycles for sword/two-sword, and one per each cycle for word/half-word. The maximum rate is reduced by competition for memory banks and the request phase time. The output is also reduced by the rate of data into the buffer.

The APL unit is designed to receive and start each input/output as an independent operation. The requests of the separate operations avoid each other such that (except for a short time after a successful space table search when I/O may cause bank busies) no bank busies are caused. A request of lower priority must look both before and after the request time it wants to use to make sure there are no requests of higher priority to the same bank within three cycles.

To accomplish this, a request of higher priority must pass through a timing chain before being sent to memory. The chain must be seven cycles long; three cycles before, the one cycle and three cycles after.

Some types of operations have requests of equal priority. They operate on a first come, first served basis.

The chart below shows the priority (highest on top) of all the types of operations listed earlier. I/O is listed because, even though it does not originate in APL, it passes through APL's conflict timing chain because it is the top priority request.

- I/O.
- A-vector, B-vector.
- C-vector.
- CS-string.
- CSLA, CLA, X-string, Y-string, Z-string.
- Random load, FOD, LOD, RNS.

VECTOR STREAM INPUT (VST)

All input buffers write data immediately upon receiving data from the appropriate SECDED nets. The input buffers read data only if their read/write address registers are not equal and if there is a data request from the appropriate input alignment net. When the input buffers read data, they generate a data full bit and send it to the input alignment networks along with the data. The following list describes the buffers.

- Input data buffers are provided for all input (A,B,X,Y,Z) data streams.
- A and B buffers each are 48 addresses in length by 256 bits wide.
- X and Y buffers each are 12 addresses in length by 128 bits wide.
- The Z buffer is 48 addresses in length by 128 bits wide.
- The A and B buffers receive data directly from the R1 and R2 SECDED networks.
- The X and Y buffers receive data directly from the R3 SECDED network.
- The Z buffer receives data directly from either R2 or R3 SECDED networks.

VST SECDED

Read 1, Read 2, and Read 3 in VST use SECDED as described earlier in this section. The characteristics of R1, R2, and R3 are as follows:

- R1, R2, and R3 are 128 bits wide (one- and two-pipeline machines).
- R1, R2, and R3 are 256 bits wide (four-pipeline machine).
- R1, R2, and R3 SECDED networks and error recording are physically located in VST.

VST Expansion Networks

The A and B data streams each contain an expansion network between the input shift network and the transmission to the register file or pipelines.

Expansion is used for the 7D, BC, BD, and AX instructions, and on the exchange operation with 1 million words of memory.

VST Scale Network

The scale network processes control vector and input order vector data. The scale network provides the following:

- Control vector and order vector data, W1 size, W2 size (use count), and leading zeros count to the backend (VSW).
- Shift and expansion data to the A and B data streams in VST.

- W1 size and W2 size to increment the W1 and W2 buffer size counters in VST.
- Element count to decrement field length counter 1 (in VST) for sparse vector instructions.
- Interrupt count updates to the VSU unit, for all input data streams for nonstring type instructions.

This network processes one, two, four, or eight control vector bits per minor cycle, depending on the instruction and pipe size.

It processes from 1 to 16 input order vector bits per minor cycle, depending on instruction type, pipeline size, and the population of one bits in the OR of the X/Y order vectors.

Field Length Registers

VST contains two 16-bit field length registers. These two registers are referred to as FL1 and FL2.

FL1 and FL2 are set up at the beginning of an instruction via VEX microcode. They always contain an element length. They are decremented by VST at the time it sends data to any functional unit. When they are decremented to zero, the VST unit stops sending data to the functional units, clears all fulls in VST, sends an empty signal to VEX microcode in VSU, and sends a terminate signal to VSW and APL.

NOTE

FL1 and FL2 are used only for nonstring instructions.

Register File Reads/Writes

VST contains a 9-bit register file read address register/incrementer. VST also contains an 8-bit register file write enable register.

All register file references for the 7D, B7, BA, and exchange operations use the VST register file hardware.

Halts/Interrupts

Any halt caused by a vector memory reference is sent to VST by the associative unit. The halt causes an immediate space table search by associative, and it stops VST from sending any more data to the functional units.

If the space table search results in a find (hit), then the associative unit sends a clear halt signal to VST. VST then resumes sending data to the functional units and data processing continues.

If the space table search results in no find, an access interrupt occurs.

VST receives one interrupt line from VSU. This line includes all interrupts (access, external, illegal, and so forth).

The VST unit stops sending data to the functional units upon receipt of an interrupt signal. It then checks to see if a termination occurred prior to the interrupt. If a termination occurred prior to the interrupt, then the VST unit ignores the interrupt.

If a termination has not occurred prior to the interrupt, then the interrupt forces a termination and sends an interrupt signal to VSU, VSW, and APL. All data in the input buffers is discarded, and VST prepares to receive the next instruction (an exchange is just another instruction to VST).

NOTE

Halts, interrupts, and termination are handled by the VSS unit (not VST) for all string instructions.

VECTOR FLOATING-POINT PIPELINE

The vector floating-point pipeline provides logical and arithmetic operand processing for vector instructions. The vector floating-point pipelines have three configurations; one pipeline, two pipelines, and four pipelines. The one- and two-pipeline configurations are structured as shown in figure 2-5. The pipeline contains five operand processing units, a data interchange to connect these units, and control logic.

The bus width for the A and B operands is 128 bits for the one-pipeline and two-pipeline versions. The four-pipeline version is essentially two two-pipeline processors, thereby providing a data path of 256 bits. The one-pipeline processor can process one 64-bit mode operand for A and B or two 32-bit mode operands for A and B. The two-pipeline processor can process two 64-bit mode operands for A and B or four 32-bit mode operands for A and B and the four-pipeline processor can process four 64-bit mode operands for A and B or eight 32-bit mode operands for A and B at the same time.

The minimum interval required between operands supplied to the vector floating-point pipeline is less than the time required to produce a result; therefore, at any given time, the segmented operand processing logic can contain a number of operands in various stages of processing. The amount of operand logic used depends on the type of vector instruction. The required operand processing logic unit is connected between the input and output of the vector floating-point pipeline by selecting the appropriate data interchange path. More complex instructions, such as the DC instruction which requires more arithmetic capabilities than a single operand processing unit possesses, may pass through several operand processing units from input to output.

The following descriptions of the components of the vector floating-point pipeline assume the typical two-pipeline version.

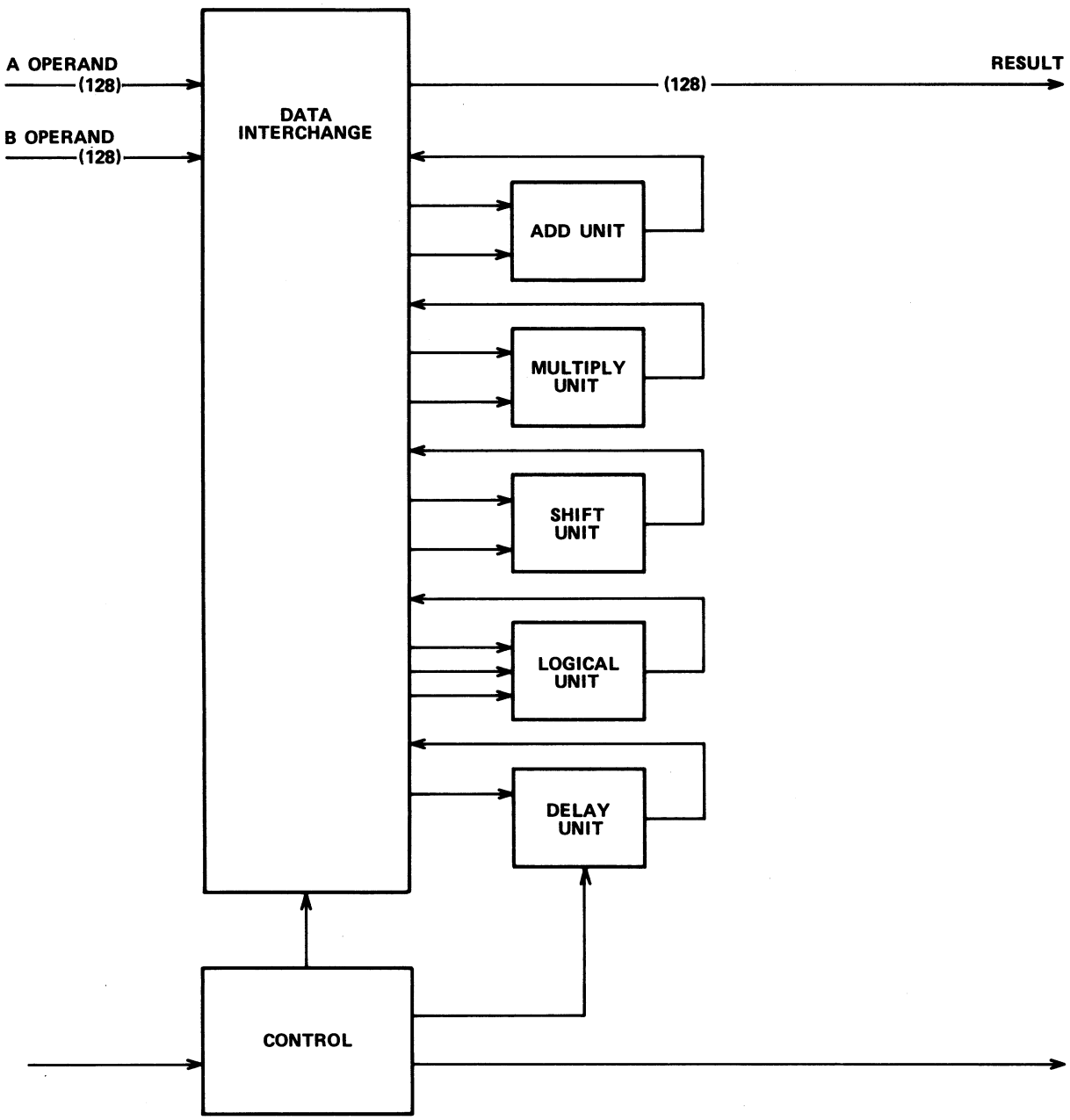


Figure 2-5. Vector Floating-Point Pipeline Basic Block Diagram

Pipeline Data Interchange

For normal vector instructions, the data interchange is configured to connect the input and output trunks to the appropriate processing unit. A link operation causes the data interchange to connect the output of one unit to the input of a second unit. For the CF, D8, D9, DA, DB, DC, and DF instructions, more complex connections are utilized and may be switched dynamically.

Add Unit

The add unit receives operands from and delivers results to the data interchange over 128-bit data paths. A block diagram of the add unit is shown in figure 2-6.

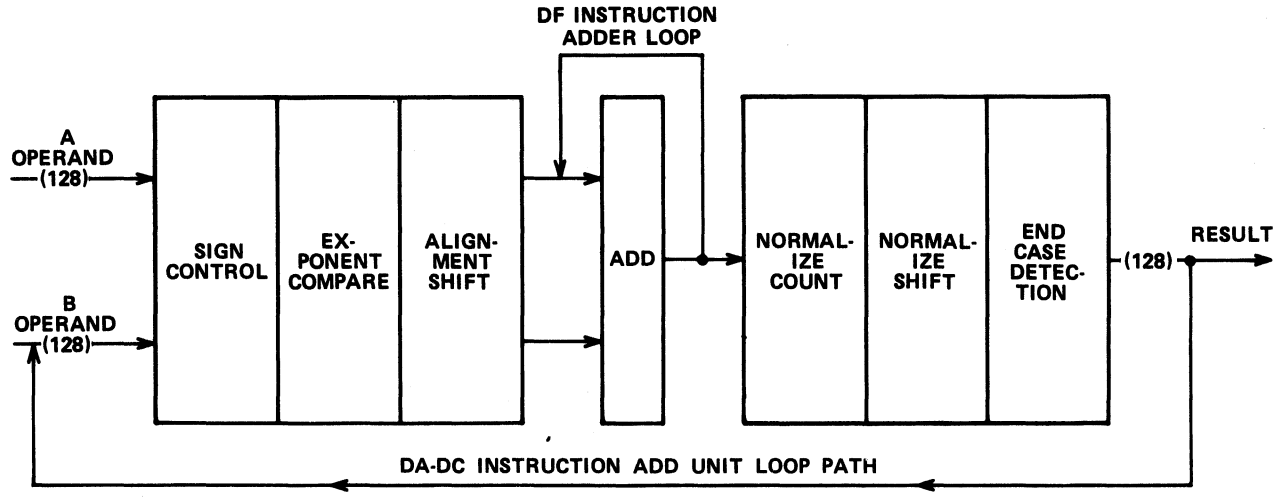


Figure 2-6. Add Unit Block Diagram

For instructions with sign control, the appropriate complementing is done in the sign control segment. This segment also complements the B operand for subtract operations.

The A and B exponents are compared in the exponent compare segment. The difference between the two exponents is used as a shift count which determines the amount the coefficient with the smaller exponent is right shifted in the alignment shift segment.

A one-cycle adder loop path is provided around the add segment for the DF (interval) instruction.

The normalize count segment produces a shift count which controls the normalize shift segment and modifies the result exponent if normalization is required.

The end case detection segment determines if an end case has been encountered and forces the result accordingly.

An eight-cycle add unit loop path is provided for use by the DA and DC instruction.

The 90, 91, and 92 (truncate, floor, and ceiling) instructions are implemented by forcing the exponent of the B operand to zero.

Multiply/Divide Unit

The multiply/divide unit receives operands from and delivers results to the data interchange over 128-bit data paths. A block diagram of the multiply/divide unit is shown in figure 2-7.

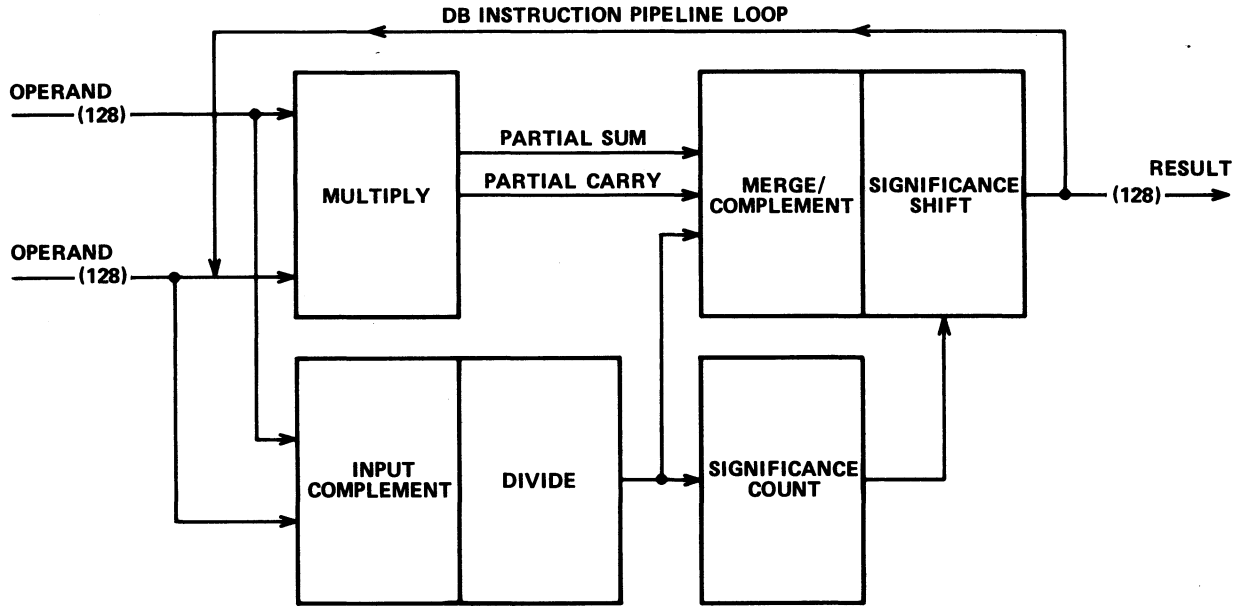


Figure 2-7. Multiply/Divide Unit Block Diagram

The multiply segment performs the coefficient multiply for the 88, 89, 8B, A8, A9, AB, DB, and DC instructions. It also serves as a pass through for the adjust significance (94) and adjust exponent (95) instructions.

For divide and square root instructions (8C, 8F, 93, AC, AF), the operands are made positive in the input complement segment. The coefficient divide or square root is performed in the divide segment.

The merge/complement segment has two functions. It merges the partial sums and carries from the multiply segment and selectively complements the result coefficient as required by the input sign bits and the sign control specification in G bits 5, 6, and 7.

The significance shift segment adjusts the significance of the result coefficient and modifies the result exponent for those operations specifying significant results. A seven-cycle short stop is provided for the DB instruction.

Shift Unit

The shift unit receives a 128-bit A operand and a 14-bit B operand (two 7-bit shift counts) and returns 128 bits of result to the data interchange. A block diagram of the shift unit is shown in figure 2-8.

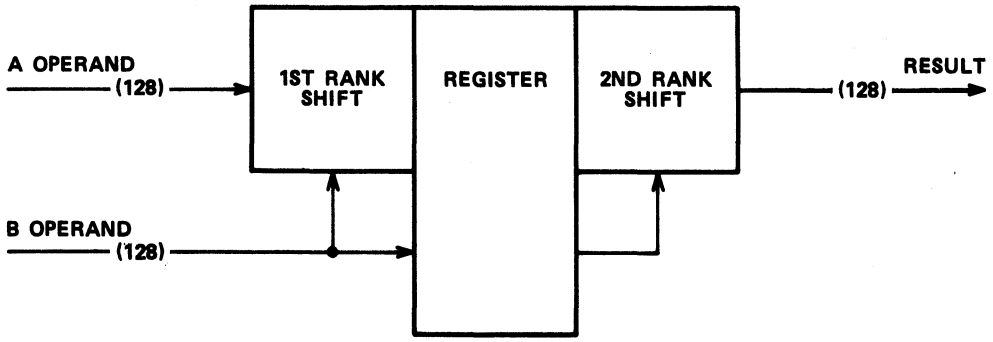


Figure 2-8. Shift Unit Block Diagram

The low order 3 bits of the shift count control the shift in the first rank shift segment. The next three higher order shift count bits control the shifting in the second rank shift segment.

Logical Unit

The logical unit receives operands from and delivers results to the data interchange over 128-bit data paths. In addition, a 2-bit path is used for the result of the masked compare (CC) instruction. A block diagram of the logical unit is shown in figure 2-9.

The 9A and 9B instructions are performed in the pack/unpack network. The vector logical instruction (9D) is performed in the Boolean network. For the masked compare instruction (CC), an exclusive OR operation is performed in the Boolean network and then combined with a broadcast mask in the masked compare network. For certain other instructions, the logical unit provides a pass through path.

Delay Unit

The delay unit receives operands from and delivers results to the data interchange over 132-bit data paths. The 132 bits consists of 128 bits of data and four control bits. A block diagram of the delay unit is shown in figure 2-10.

The delay function is implemented by offsetting the read and write addresses of the buffer memory by the required number of cycles of delay.

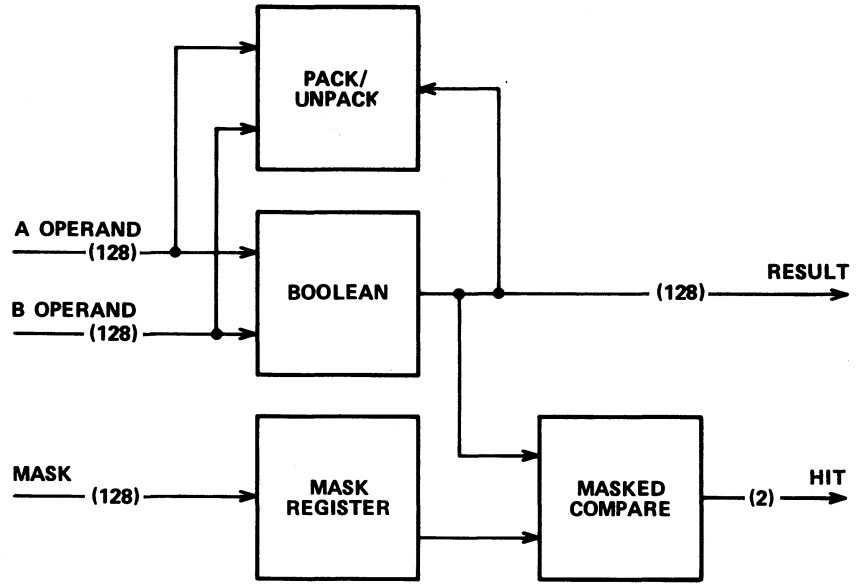


Figure 2-9. Logical Unit Block Diagram

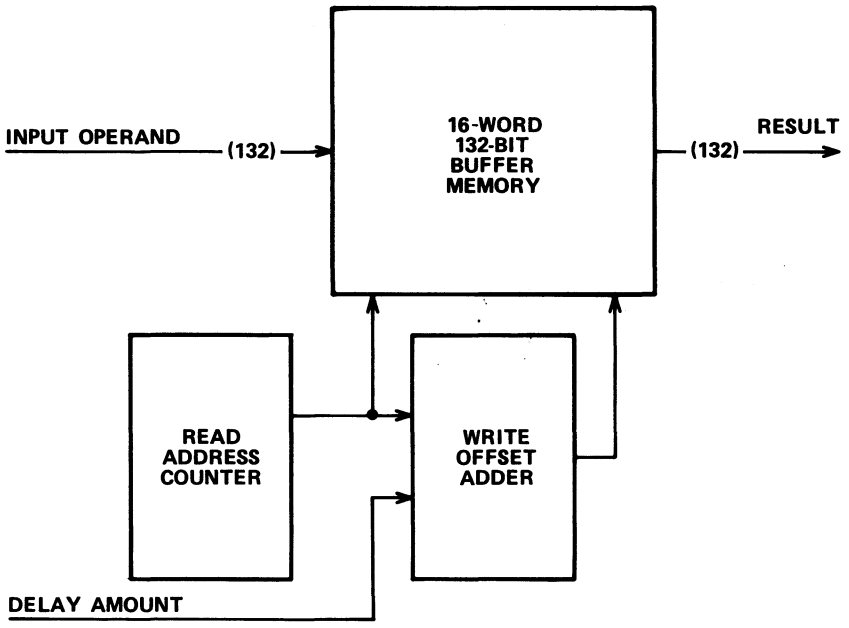


Figure 2-10. Delay Unit Block Diagram

Vector Floating-Point Control

The vector floating-point control contains the necessary logic to manage the operand processing. This includes interconnection of operand processing units, unit startup, interruption of processing operations, resumption of processing operations after interruption, and processing operation shut down.

VECTOR STREAM OUTPUT (VSW)

The CYBER 205 has two output streams: write one (pipelines and register file output) and write two (string output). The pipes and register file output stream accepts word/half-word data from the functional units and aligns it on a 32-bit address for storage to memory. The string output accepts bit data from the functional units and aligns and merges the data to form a half-word for storing to memory.

Write One (Pipelines and Register File)

The pipes and register file output stream can process up to 256 bits of data per minor cycle (four-pipeline machine) or 128 bits of data per minor cycle (one- or two-pipeline machine). The function of the pipes and register file output is to accept data from the functional units, compress out the unwanted data (sparse vector), align and buffer the data for storing into memory. The pipes and register file output are separated into five subunits. They are referred to as the output selection, compression, alignment, buffers, and SECDED.

Output Selection

The data for the pipes and register file output stream may be transmitted from three functional units. They are the pipelines with up to 256 bits for a four-pipeline system, register file output of 256 or 128 bits, and the vector setup unit output of 128 bits. The pipeline bus is used for all pipeline type vector instructions creating a word or half-word result. The register file bus will be used for the exchange operation and storing (in memory) of a single register file read operand. The vector setup unit bus is used for the storing of the invisible package information and instructions that create an index output.

Compression

The compress networks main purpose is to compress out unwanted data for the sparse vector instructions. This is accomplished by examining the sparse vector instruction order vector. Where a one appears in the sparse vector, the data is stored into central memory. Where a zero is examined, the data is discarded. The register file input may use the compress network to align the data starting at bit zero for a half-word, word and quarter sword register file address.

Alignment

The alignment network consist of a 256-bit bus allowing alignment of any half-word of data into any one of eight locations for storing into memory. A secondary purpose of the alignment network is to broadcast a 32/64-bit quantity over the entire 256-bit bus during one cycle.

Buffers

The pipes and register file output stream buffer is 264 bits by 64 words. The 256 bits are used for data and 8 bits are for memory write enables. (The VST unit complements control vector bits sent to the output when zeros are permissive.) Normal operation accumulates a sword or two-sword quantity of data in the buffer before it is read and sent to the priority unit for storing into central memory. During a space table search operation, the buffer must accumulate the data that was in the pipelines when the space table search operation started. This may be up to several swords/two-sword quantities depending on the pipeline length.

Write Two (String, VSS)

The string output receives from 0 to 16 bits of data from the functional units, and aligns and assembles the data into 32-bit groups for storage into the buffer. In the string buffer, 128 bits are assembled for storage into central memory. The string output unit consists of six subunits; old data, selection, alignment, merge, buffer, and SECDED.

Old Data

Data written into central memory must be in multiples of 32-bit quantities. The string unit data can begin and end anywhere within 32-bit groups. The nonstring data within a 32-bit group is referred to as old data.

The old data consists of FOD/IOD, (first old data/interrupt old data) and LOD (last old data). FOD/IOD is used to assemble from the half-word address to the string output starting address. LOD is used to fill out the half-word from the point where the string output data ended (refer to figure 2-11). IOD is similar to FOD except that it refers to the partial output data existing at the time an exchange occurs due to an interrupt.

Data Selection

The data selection subunit of the string output is used to select the data from the three functional units, string, FOD/LOD and the front end. Each of these units can send from 0 to 16 bits per cycle.

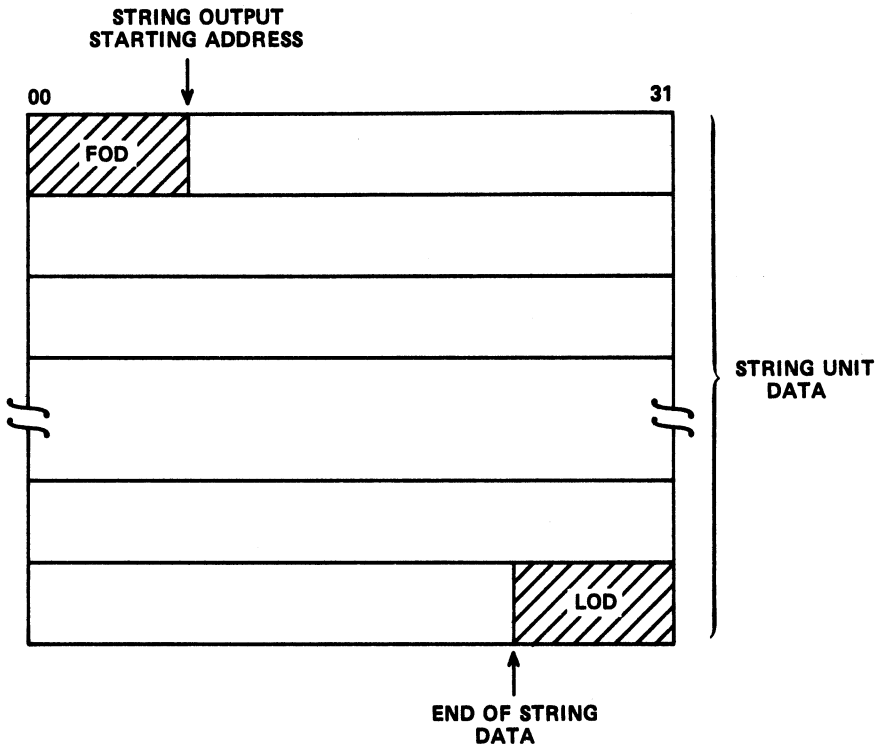


Figure 2-11. String Unit Old Data

Alignment

The string alignment network shifts the string data to the correct bit address, allowing the merge network to assemble 32 bits of data for storage.

Merge

The merge network receives from 0 to 16 bits of data from the alignment network and assembles it into a 32-bit half-word for storage in the string output buffer.

Buffer

The string output buffer is 16 words by 132 bits, 128 bits of data and four memory write enables. The buffer assembles 32-bit half-words into a sword of data for storage into central memory. The buffer is also used to accumulate data when a space table search is in progress.

SINGLE ERROR CORRECTION DOUBLE ERROR DETECTION (SECDED)

SECDED provides automatic correction of a single bit error in a memory word and optional latching of the single bit error and multiple bit errors. SECDED checking and generating is done in both the scalar processor and vector processor. SECDED generates seven check bits for each 32 bits of data. These data and check bits are then transmitted to the scalar priority unit for storing into central memory. When the data and check bits are read from central memory, the check bits are used for error correction and detection.

There are eight SECDED units within the CPU; three write and five read units. The three write or generate units are:

1. Write 1 vector
2. Write 2 vector
3. Write scalar

The five read or checker units are:

1. Read 1 vector
2. Read 2 vector
3. Read 3 vector
4. Read next sword (RNS) for next instruction
5. Read scalar

The SECDED error information is stored by the maintenance control unit (MCU). The stored information is the syndrome word, single error or double error bits, read bus code, and CPU physical address bits 36 through 58. The I/O ports have no SECDED generators and checkers. Equipment attached to the I/O ports perform SECDED.

CPU WORD ADDRESS BITS (36 THROUGH 58)

The word address bits (bits 36 through 58) indicate the following:

Bit	Description
36-37	Select 1 of 4 memory chip/bank
38-49	Select 1 or 4096 words/chip
50	2048K select
51	1024K select
52-54	Bank select
55-56	Quarter sword select
57-58	Half-word select

SECDED ERROR LATCHING HARDWARE

The SECDED error latching hardware has two modes of operation; mode 1 and mode 2. Mode selection is accomplished through the MCU/CPU maintenance line called Select SECDED Error Log Mode Two.

In modes 1 and 2 for simultaneous SECDED errors, the error latch information to be latched is dependent on the relative priority of the data buses or half-words containing the errors. It is possible to encounter a single and double error simultaneously and latch the single error; the double error flag sets unconditionally. Therefore, if the double error flag sets, the syndrome bits must be checked to determine if a single or double error was latched.

SECDED USAGE

The SECDED mode best suited for a system is based on the error rate of the memory.

Mode 1

Mode 1 is normally used during system processing for a memory with a low error rate. All error log information is correct, but mode 1 does not latch a double error if it follows a single error within the cycle time of the MCU. The first error occurring after a master clear or error clear has its error information latched. The information is correct regardless of subsequent errors. If a double error follows a single error before an error clear, the double error information is lost.

Mode 2

Mode 2 is used for a memory with a higher error rate. All single errors latched are correct, and all double errors following a single error by more than eight minor cycles are correct. A double error occurring before a single error is also latched correctly.

Double Error Log (Mode 2A)

Mode 2A should be used to locate defective storage after a high error rate has occurred. This mode misses the double error only if there is a simultaneous single error with a higher latching priority.

After a master clear or error clear, the MCU creates a single error using the maintenance function to toggle a check bit. This single error is not cleared, and blocks detection of all subsequent single errors. Therefore, when the MCU detects the double error flag, the error log information is correct for that double error.

SECDED FAULTS

Executing an 06 instruction with bits 9 through 15 of the R designator selected, causes a word or words to be written into memory with incorrect SECDED code. This allows checking the SECDED networks on any or all read buses. All read bus SECDED networks are disabled by setting bit 8.

Block Write Enables

The MCU can block write enable if a SECDED error occurs. Depending on the mode, there are two options:

- Mode 1 - The write enable is blocked when SECDED receives its first single or double error.
- Mode 2 - The write enable is blocked when SECDED receives its first double error.

INPUT/OUTPUT

The CPU provides data, function, maintenance, and status communication between the CPU and other external system elements through I/O channels. A channel is made up of a number of devices and networks and at least one I/O port. The I/O port is physically located in the CPU chassis. The device outside of the CPU and cabled directly to the I/O port is referred to as the system channel adapter.

I/O PORTS

There are eight bidirectional I/O ports in the CPU. Another eight ports with identical performance, timing, and equal priorities may be added as an option. Any port may be used as a maintenance control channel by setting the maintenance line coming into that particular port. This causes data transfers with the System Channel Adapter (SCA) to be made with the IOM maintenance registers rather than with the CYBER 205 memory. The maintenance port can disable the capability of any or all ports from making central memory transfers or sending interrupts. The maintenance port does not disable the maintenance control channel operation.

A portion of the memory bandwidth is dedicated to the I/O ports and is unaffected by other system activities. All other memory requests yield to I/O requests. This bandwidth is equally divided among the ports providing 200-megabit transfer rates on any or all ports simultaneously. The transfer length limits are: minimum transfer length of one sword and a maximum transfer of 4096 swords. All data transfers start and terminate on sword address boundaries.

SYSTEM CHANNEL ADAPTER

The system channel adapters (SCA) are located in a stand-alone cabinet. This cabinet contains a maximum of six SCAs. The minimum number of SCAs in a system is 6 and the maximum number is 16 when the system has the optional 8 I/O ports. Each SCA interfaces an I/O port with an external device for transfer of data and maintenance information. (Refer to figure 2-12.) Each SCA operates independently and consists of an A interface for transfer of 16-bit parallel data (with two parity bits) to/from the external device and a CYBER 205 interface for transfer of 32-bit parallel data to/from the CPU. Two internal buffer memories allow simultaneous data transfers with the external device and CPU. Each memory has a capacity of 128 data words (8 swords). Word size is 37 bits (32 data bits, 4 parity bits, and 1 SECDED error bit). SECDED is provided and checkbits are generated and accompany each 32-bit data word sent to the CPU. Checkbits received with each word are used for error detection and correction. A loop mode feature allows testing of the buffer memories, by the external device, without initiating a transfer request to the CPU.

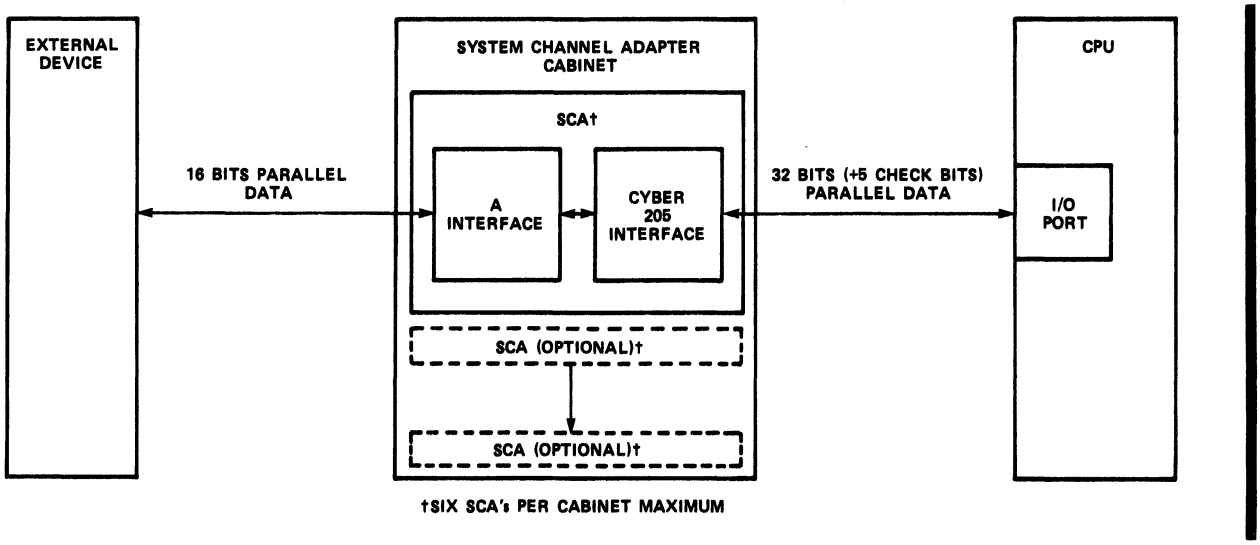


Figure 2-12. System Channel Adapter

CYBER 205 Interface Lines

The interface lines from the SCA to the I/O port (46 lines) are defined as follows:

- Write Enable (one line).

  The corresponding half-word is written into the CPU memory or maintenance interface.

- Write Data (32 lines).

  Write data is a group of 32 lines containing the starting memory address and the field length for a read or write. Following the start address is the data in half-words to be written into the CPU memory or maintenance interfaces.

- Write SECDED check bits (seven lines).

  Write data check bits, with write data, for checking SECDED operation. The data and check bits are written into the CPU memory where later reads of the same data and check bits will exercise the SECDED hardware.

- Write (one line).

  This signal indicates a transfer of data from the I/O port to the SCA if a logical 0 (read) or a transfer from the SCA to the I/O port if a logical 1 (write).

  SCA to the I/O port if a logical 1 (write).

- Initiate (one line).

  This signal is sent at the beginning of any data transfer to indicate that the sword address, sword count, and the write and maintenance signals are available. Initiate must remain a logical 1 until a data valid signal is returned.

- Write Strobe (one line).

  A timing pulse that strobes all other signals sent to the I/O port. It is derived from and synchronized with the read strobe.

- Maintenance (one line).

  A logical 1 indicates that data will be read from or written into the IOM maintenance registers. A logical 0 indicates a data transfer with CPU memory.

- Interrupt (one line).

  This line signals an interrupt is being sent to the CPU. If the interrupt signal is received during a write operation, it is not sent to the CPU until the write is complete.

- Master Clear (one line) not used.

The interface lines from the I/O port to the SCA (45 lines) are defined as follows:

- Read Data (32 lines).

  Read data is a group of 32 lines containing half-word read data being sent from the CPU memory to the channel device.

- Read SECDED check bits (seven lines).

  Read data check bits used by SECDED to check read data and correct and record errors.

- Data Valid (one line).

  Notifies the SCA it can begin sending data during a write operation or that the I/O port is sending data during a read operation.

- Read Strobe (one line).

  This is a timing pulse that strobes all other signals (except control function and control function strobe) sent from the I/O port to the channel device. It is also used to synchronize the write strobe.

- Control Function (two lines).

  A 2-bit code used to indicate system control functions.

- Control Function Strobe (one line).

  Used as a timing pulse to strobe the control function.

- Inactive (one line).

  Notifies the SCA that the I/O port is not active and will not respond to any signals sent by the SCA.

EXTERNAL DEVICE TRANSMISSION SEQUENCE

Data transfers from the external device occur in multiples of 32-bit words consisting of two transfers of 16 bits each. Operations are initiated by the external device sending a 32-bit address field accompanied by a function code. Data transfers occur in either direction as determined by the function code.

The address field is transferred in two 16-bit words (figure 2-13). The upper portion of the address is sent in the first 16-bit word and the lower portion is sent in the second word. Data transfers occur in the same sequence with the upper half of a 32-bit word appearing in the first 16-bit word and lower half in the second word.

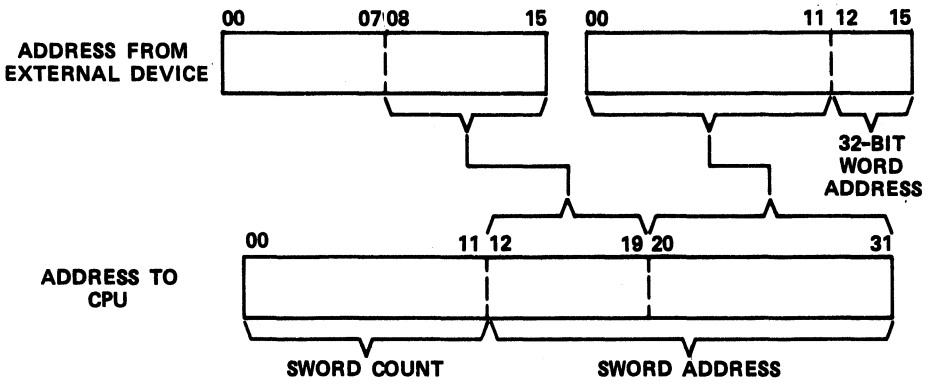


Figure 2-13. I/O Transmission Sequence

SYSTEM COMMUNICATION

The CPU communicates with the SCA's through two encoded control lines that specify channel flag, external flag, and suspend functions.

These functions are defined as follows:

Channel Flag — A channel flag is transmitted by the execution of a 08 instruction. The 08 instruction designates which port the flag will be sent through. Table 2-3 gives R designator to port correspondence.

TABLE 2-3. CHANNEL FLAG ASSIGNMENTS

08 Instruction R Designator	Port
00	NONE
01	Port 1
02	Port 2
03	Port 3
04	Port 4
05	Port 5
06	Port 6
07	Port 7
08	Port 8
09	Port 9
0A	Port 10
0B	Port 11
0C	Port 12
0D	Port 13
0E	Port 14
0F	Port 15
10	Port 16
11 through FF	UNDEFINED

Suspend — A suspend is transmitted by all ports when a master clear is performed.

STORAGE AND MAINTENANCE ACCESS

The SCA sends the start address for a CPU memory reference to the I/O port on the right 20 bits (bits 12 through 31) of the write data lines. The address is a sword address with the capability of addressing up to 1 million swords of memory. All transfers are in sword increments with half-word writes controlled by write enables.

The left 12 bits (bits 00-11)on the write data lines received at the I/O port contain the data transfer length in swords. The maximum data transfer size is 4096 swords specified by a 12-bit data transfer length of zero. The actual usable maximum data transfer is determined by the external peripheral devices. A data transfer with the maintenance interface uses a length of one sword.

A write enable is sent on the write enable line with each 32 bits of write data. A logical 1 on the write enable line causes that half-word to be written into CPU memory or the maintenance interface. A logical zero prevents a write.

An access function accompanies each start address. The access functions are defined as follows:

Write/Read - Data is written into the CPU memory or maintenance interface if the write line is a logical one. Data is read from the CPU memory or maintenance interface if the write line is a logical zero.

Maintenance - Write or read the maintenance interface within the CPU through the I/O port if the maintenance line is a logical one. Write or read the CPU memory if the maintenance line is a logical zero.

For maintenance access the 20-bit sword address is not used, but the field length must be set to one sword.

I/O PRIORITY

The top memory access priority in the CPU is assigned to I/O operations. All other memory requests yield to I/O requests.

The CPU I/O control sequentially services all 8 or 16 I/O ports for memory requests. The I/O control can maintain maximum data transfer rates on all I/O ports simultaneously with vectors running in parallel on any CPU operation.

CENTRAL MEMORY - SERIES 400

Each 1 million words of central memory contains 16 memory stacks, each having 128K 39-bit half-words (32 data bits plus 7 SECDED bits). Each 128K stack is arranged in eight phased banks. Memory can assign sequential addresses to different banks by using bank phasing. Because the banks are independent, a bank can begin a memory cycle before adjacent banks have completed previously initiated cycles. In streaming mode, a reference is made simultaneously to the same address in each of the 16 memory stacks obtaining a superword (sword) of 512 data bits (one, two, or four pipelines) or 32 memory stacks obtaining a two-sword quantity of 1024 data bits (four pipelines). Each 1 million words of memory contains 128 phased half-word banks. Figure 2-14 shows the chassis configuration for 1 million words of memory.

CENTRAL MEMORY - SERIES 600

The basic 1 million words of the Series 600 central memory contain 16 memory modules. Each module contains 128K 39-bit half-words (32 data bits plus 7 SECDED bits) called a rank. An additional 1-million word option can be added to the memory by installing one additional rank to the 16 modules. Another 2 million words can be added by installing two additional memory cabinets. The full complement of 8 million words is reached by adding two ranks to the four cabinets of memory. Memory modules are divided into eight phased banks. A 1-million-word memory contains 128 phased half-word banks (16 modules times 8 banks).

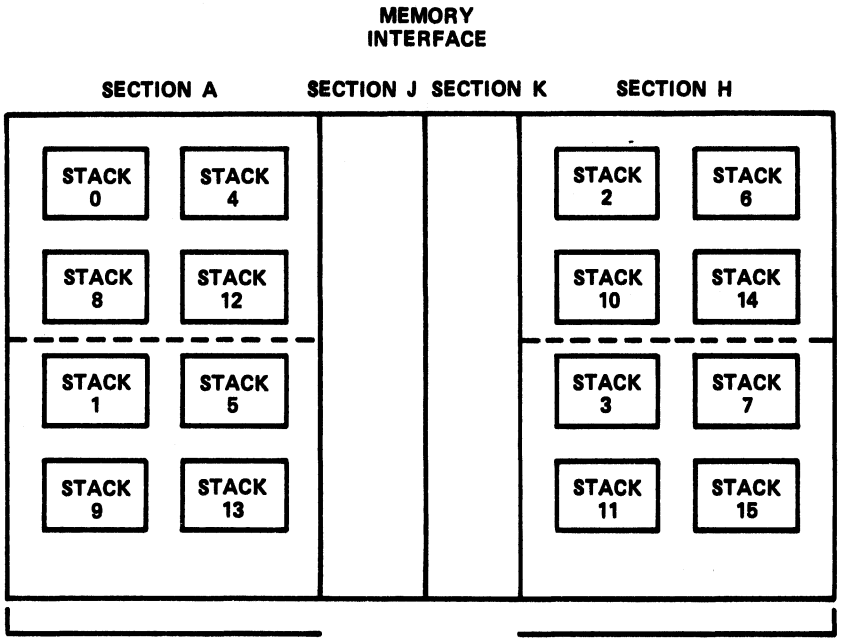
Memory can assign sequential addresses to different banks by using bank phasing. Because the banks are independent, a bank can begin a new memory cycle before adjacent banks have completed a previously initiated cycle. In the streaming mode, a reference made simultaneously to the same address in each of the 16 memory modules obtains a superword (sword) of 512 data bits (1, 2, and 4 pipeline machines) or 32 memory modules obtaining a two-sword quantity of 1024 data bits (four pipeline machines only). Figure 2-15 shows the chassis configuration for 1 million words of memory.

MEMORY OPERATION

The memory request determines the amount of data that is transferred. A memory request can be for a two-sword, a sword, a word, or a half-word quantity. One sword contains 8 words, 1 word contains 78 bits (68 data bits, 14 SECDED bits) addressed from left to right (refer to figure 2-16).

When the memory interface performs a write/read operation in sword mode, it addresses a half-word in each stack (Series 400) or a half-word in each of the modules (Series 600). For a write/read operation in word mode, the memory interface addresses 2 of the stacks (Series 400) or 2 of the 16 modules (Series 600), and in half-word mode only 1 stack or module is addressed. In addition to the memory stack or module addresses, the memory interface sends a bank address signal to select one of eight banks within a stack or module.

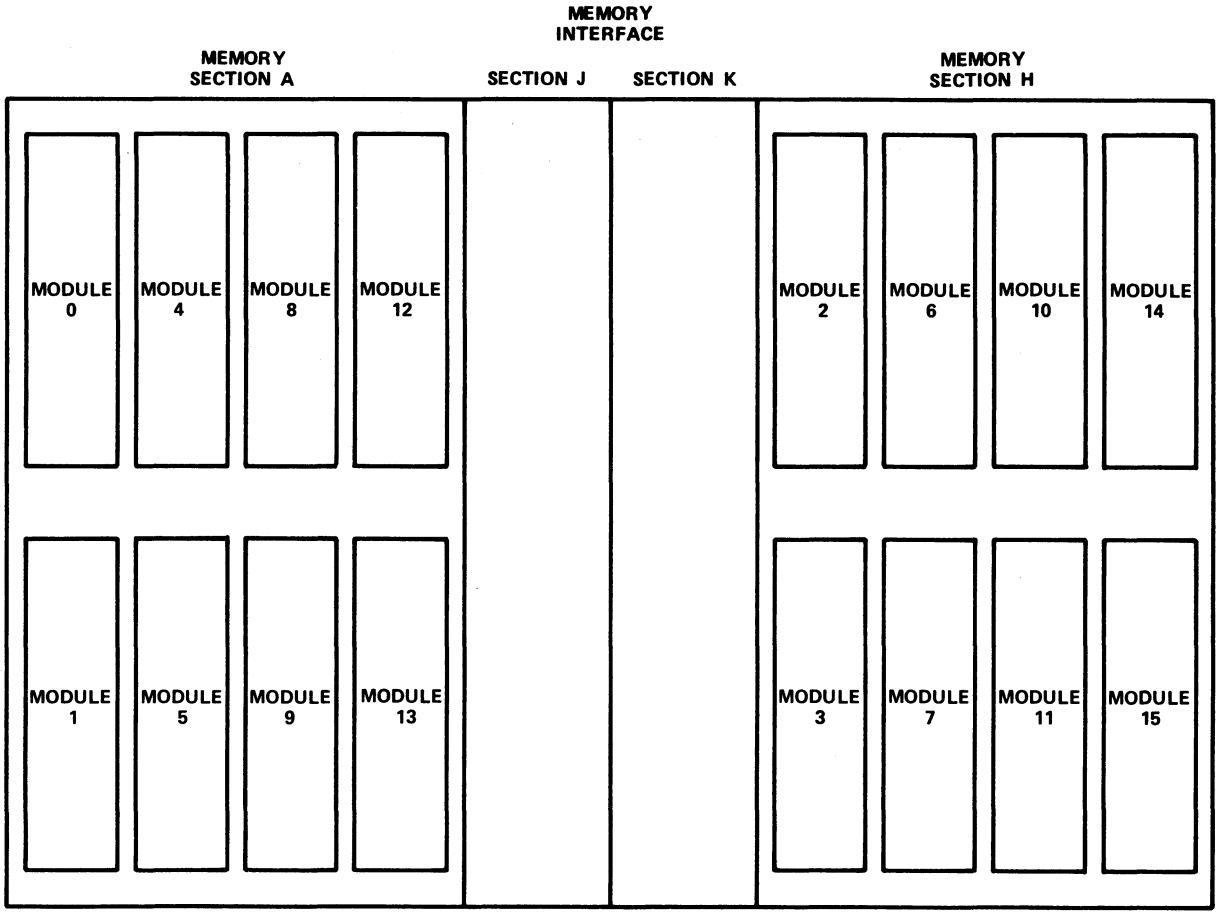
Depending upon the mode selected, all of the bits are transferred to and from central memory even though not all the bits are used. For example: in sword mode, all 624 bits of one sword are transferred to and from the central memory during each write and read operation although only part of the sword may actually be stored or transferred. The memory interface enables the proper control line for each half-word of the sword that is to be stored or transferred.



NOTES:

1. EACH SECTION HAS EIGHT STACKS.
2. TWO SECTIONS COMPRISE 1 MILLION WORDS OF MEMORY.

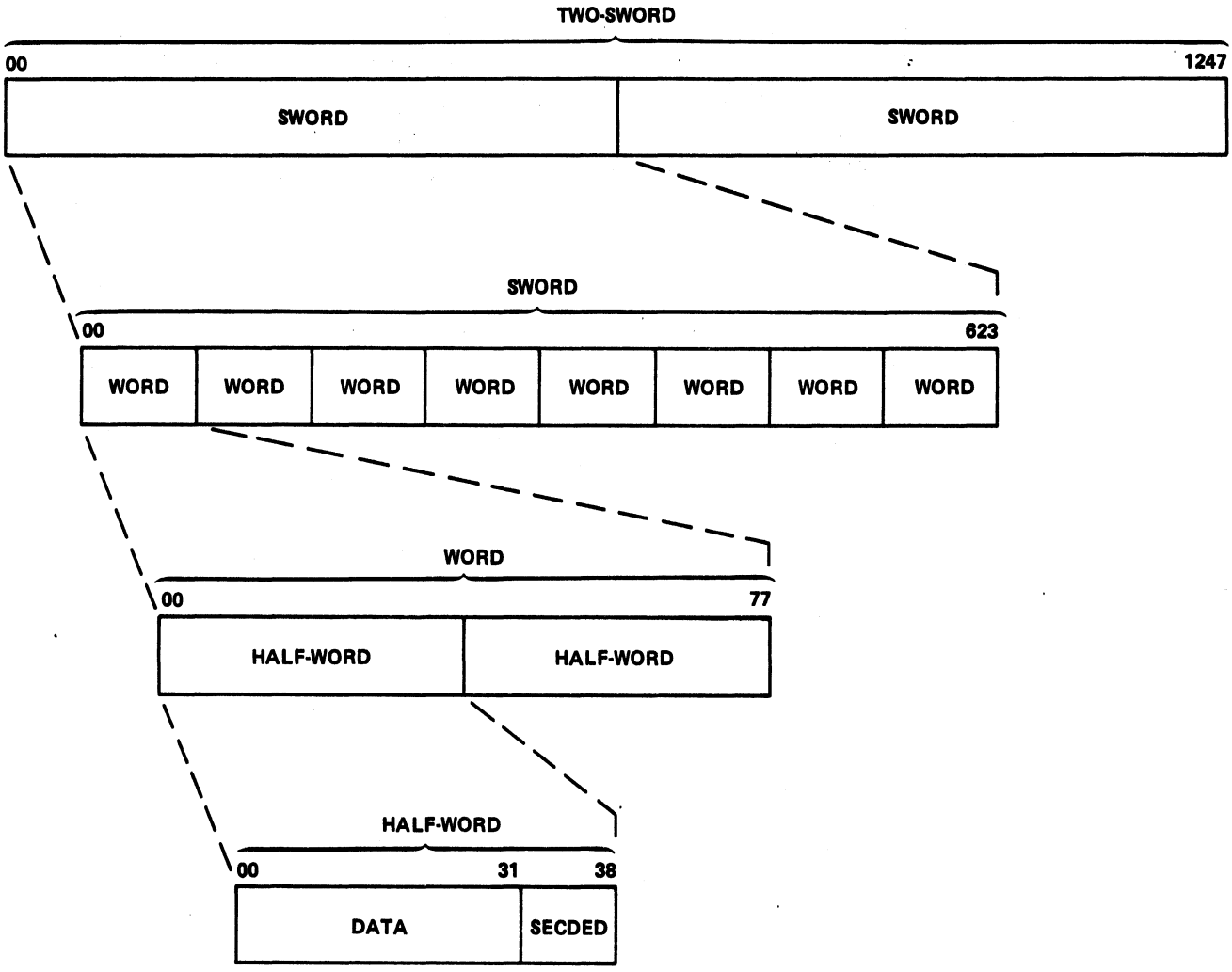
Figure 2-14. Section Configuration (Series 400)



NOTES:

1. EACH SECTION HAS EIGHT MODULES.
2. TWO SECTIONS CAN COMPRISE 1, 2, OR 4 MILLION WORDS OF MEMORY DEPENDING UPON THE NUMBER OF RANKS INSTALLED.

Figure 2-15. Section Configuration (Series 600)



NOTES:

1. EACH TWO-SWORD QUANTITY CONTAINS 2 SWORDS.
2. EACH SWORD CONTAINS EIGHT WORDS OR 16 HALF-WORDS.
3. EACH WORD CONTAINS TWO HALF-WORDS.
4. A HALF-WORD CONTAINS 39 BITS (32 DATA BITS AND 7 SECDED BITS).
5. MEMORY TRANSFERS MAY BE IN TWO-SWORD, SWORD, WORD, OR HALF-WORD DATA QUANTITIES.

Figure 2-16. Two-Sword, Sword, and Word Configuration

MEMORY ACCESS AND CONTROL

Figure 2-17 shows the control signals sent to each memory stack. Figure 2-18 shows the control signals sent to each module. All signals except read data are sent from the memory interface to the stacks or modules. The read data signal is sent back to the memory interface.

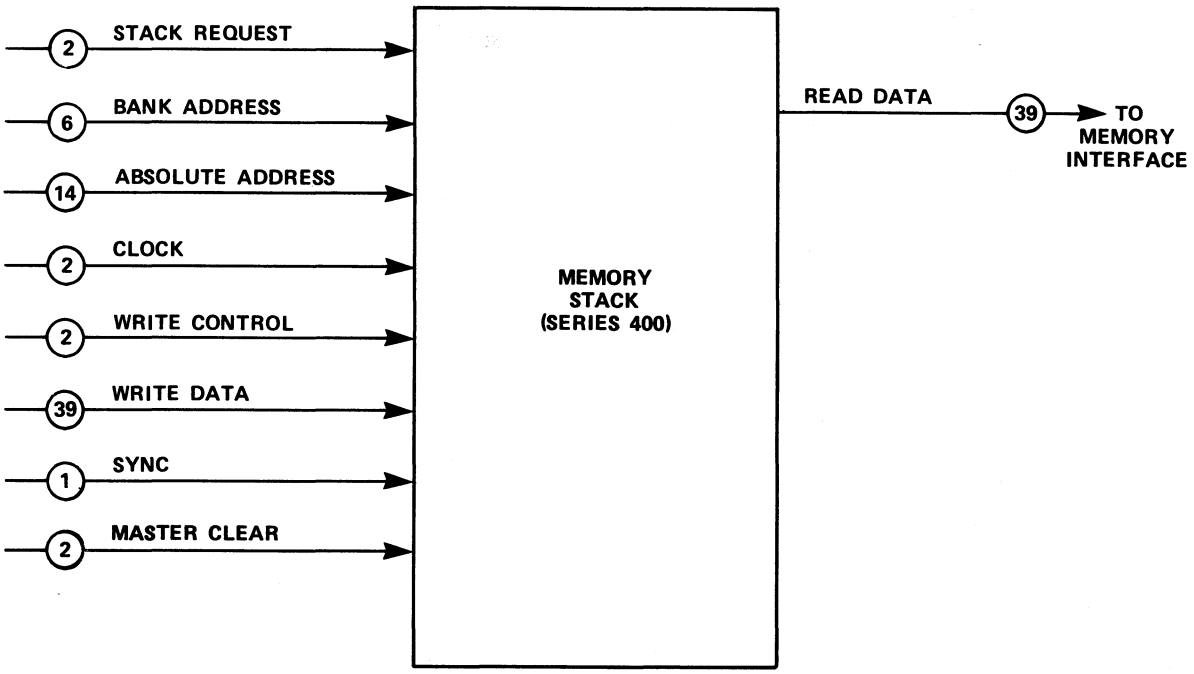


Figure 2-17. Memory Interface Stack Connections (Series 400)

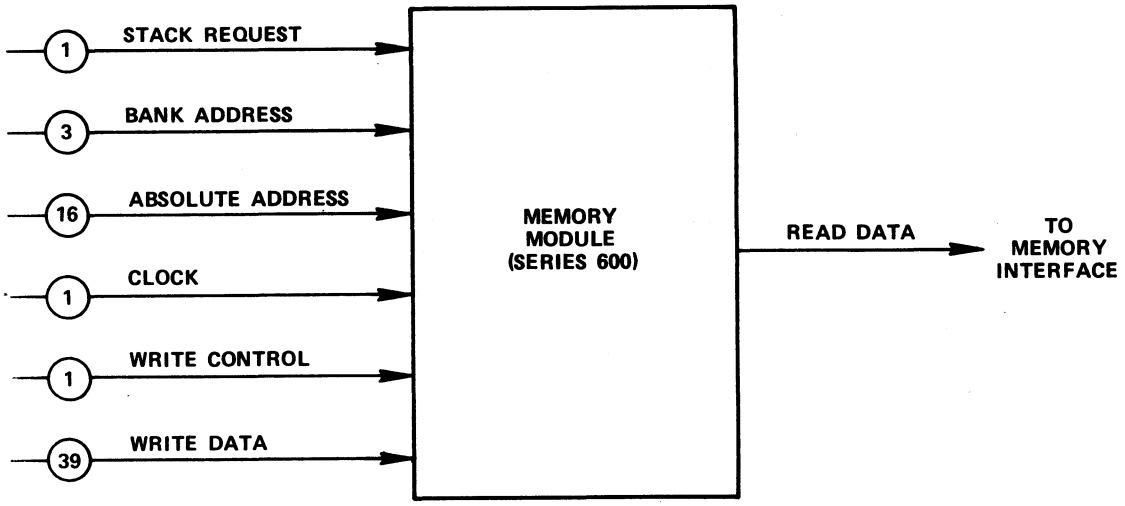


Figure 2-18. Memory Interface Module Connections (Series 600)

STACK REQUEST - SERIES 400

There are two stack request lines for each memory stack. This signal determines which stack has been selected.

STACK REQUEST - SERIES 600

There is one stack request line for each memory module. This signal determines which module to select.

BANK ADDRESS - SERIES 400

There are six (two sets of three) bank address lines for each memory stack. This signal determines which bank of the eight banks within a stack has been selected.

BANK ADDRESS - SERIES 600

There are three bank address lines for each memory module. This signal determines which bank with a module has been selected.

ABSOLUTE ADDRESS - SERIES 400

There are 14 bits that determine the absolute address; 2 bits determine which of the four banks of memory chips has been selected and 12 bits determine the address in memory selected.

ABSOLUTE ADDRESS - SERIES 600

There are 16 bits that determine the absolute address; 2 bits determine which of the 4 ranks of memory has been selected and 16 bits determine the address in memory selected.

CLOCK - SERIES 400

There are two identical clock lines for each memory stack. This signal synchronizes the memory stack to the memory interface.

CLOCK - SERIES 600

There is one clock line for each memory module. This signal synchronizes the memory module to the memory interface.

WRITE CONTROL - SERIES 400

There are two identical write control lines for each memory stack. This signal informs the memory stack of a write memory cycle.

WRITE CONTROL - SERIES 600

There is one write control line for each memory module. This signal informs the memory module of a write memory cycle.

WRITE DATA

There are 39 write data bit lines for each memory stack or module: 32 for data and 7 for SECDED.

SYNC - SERIES 400

This signal provides a point of reference for maintenance purposes.

SYNC - SERIES 600

None.

MASTER CLEAR - SERIES 400

There are two identical master clear lines for each memory stack. The memory interface pulses the master clear signal continuously whenever a master clear is present in the CPU.

MASTER CLEAR - SERIES 600

None.

READ DATA

The 39 read data bits are obtained from the read data registers on the output, and the information is sent back to the memory interface.

MEMORY INTERFACE

The memory interface provides ports for access to central memory. The scalar processor, vector processor, and I/O ports are connected to central memory through the memory interface as shown in figure 2-19. Data transmissions are controlled by the priority unit in the scalar processor. SECDED for each 32 bits of data on the memory ports is done in the scalar or vector processor. SECDED for data through the I/O ports is done externally to the central computer. Data can be transferred to and from the memory ports in 32-bit half-word, 64-bit word, 512-bit sword, or 1024-bit two-sword quantities (refer to table 2-4).

Each memory port is connected to memory through a one sword/two-sword buffer located in the memory interface. If a buffer is shared by multiple ports, the priority unit provides proper port selection to the memory interface selection network. Data is transmitted between the buffers and the processor in sword/two-sword quantities at a rate of one half-word, word, or quarter-sword/half sword per minor cycle.

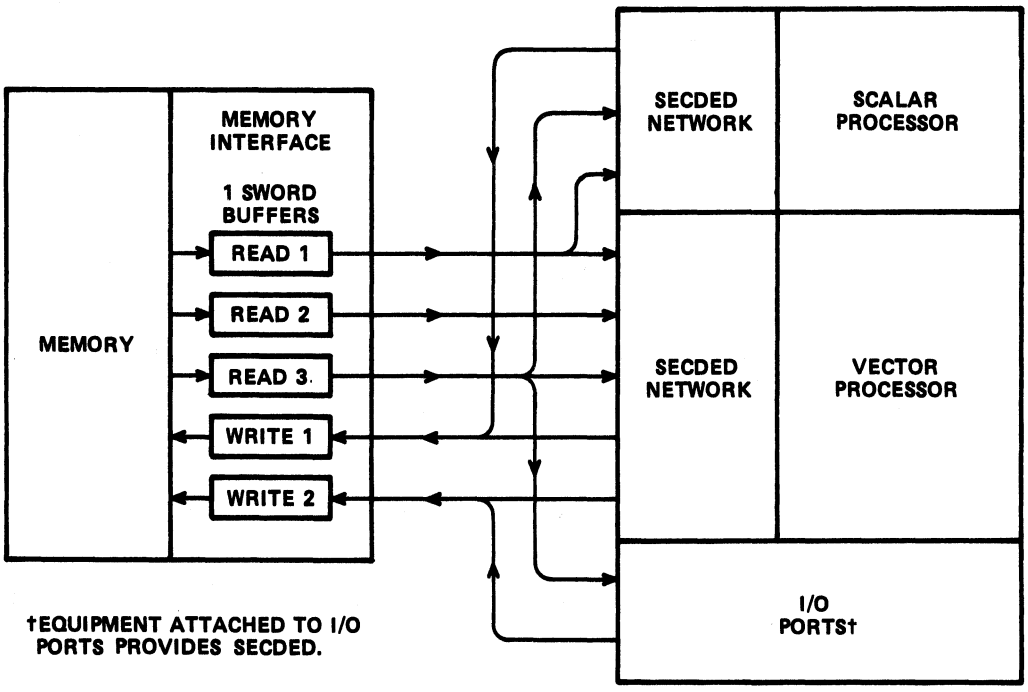


Figure 2-19. Memory Interface Configuration and Connections for a Two-Pipeline Configuration

MEMORY DEGRADATION

If more than 1 million words of memory are present, degradation may be selected. Degradation allows the amount of usable memory to be less than the total memory in the system. The amount of usable memory is controlled by a degradation code from the MCU along with a strobe bit. Tables 2-5 and 2-6 shows the memory degradation codes and their descriptions for Series 400 and Series 600, respectively.

TABLE 2-4. MEMORY PORT TRANSFER MODES

Memory Interface Buffer	Memory Port	Transfer Mode
Read 1	Scalar processor	Half-word, word, sword, two-sword
	Vector processor	Half-word, word, sword, two-sword
Read 2	Vector processor	Sword, two-sword
Read 3	Read next sword (RNS) (scalar processor)	Sword
	I/O ports	Sword
	Vector processor	Sword
Write 1	Scalar processor	Half-word, word, sword, two-sword
	Vector processor	Half-word, word, sword, two-sword
Write 2	I/O ports	Sword
	Vector processor	Sword

NOTE: Although the I/O ports have a sword transfer mode, this mode is modified to be synchronized with vector streams operating in two-sword mode (four pipelines) with minimal effect to the I/O transfer rates.

TABLE 2-5. SERIES 400 MEMORY DEGRADATION BITS (4K CHIPS)

Memory Sections Used†	Degradation Code	Usable Memory
A, H and B, G and C, F and D, E	6	4 million words
C, F and D, E	5	2 million words
A, H and B, G	4	2 million words
D, E	3	1 million words
C, F	2	1 million words
B, G	1	1 million words
A, H	0	1 million words

†Refer to figure 1-2.

TABLE 2-6. SERIES 600 MEMORY DEGRADATION BITS

Ranks Used	Degradation Code	Sections A and H Usable Memory	Sections A,H,B and G Usable Memory
Rank 0	0	1 million words	2 million words
Rank 1	1	1 million words	2 million words
Rank 2	2	1 million words	2 million words
Rank 3	3	1 million words	2 million words
Ranks 0,1	4	2 million words	4 million words
Ranks 2,3	5	2 million words	4 million words
Ranks 0,1,2,3	6		8 million words

MAINTENANCE CONTROL UNIT

The maintenance control unit (MCU) provides system autoload and system performance monitoring capabilities. The MCU also provides the capability of loading, controlling, and monitoring the central processor unit (CPU) diagnostics. Connections from the MCU to the central computer are made through I/O ports. Any I/O port may be used by setting the maintenance line for that particular port. The interfaces allow the MCU to monitor CPU status.

The primary purpose of the MCU is to support the reliability, availability, and maintainability of the central computer. The MCU provides operators with the means of autoloading the operating system and checking the CPU status.

The MCU operates in offline and online software modes.

- In an offline mode, the MCU loads CPU diagnostic routines and then controls and monitors the diagnostic operations and furnishes the results of the operations to a display unit or a line printer.

- During normal online site operation the CYBER 18 system used as the MCU on the CYBER 205 provides the operator with the means of autoloading the operating system and checking CPU status.

- During periods of degraded or intermittent system operation, the primary purpose of the MCU is to support the reliability, availability, and maintainability (RAM) CPU standards. To minimize repair time during these periods, the customer engineers supporting the site must be given top priority in using the MCU.

SYSTEM CHANNEL INTERFACE (SCI)

The System Channel Interface (SCI) provides the interface between the MCU and the SCA. Refer to the section on Input/Output for a description of the SCA. The SCI occupies two printed circuit board slots in the MCU chassis. Control logic and input/output buffering are contained on one board and the receiving and transmitting circuits necessary to interface with the SCA are contained on the second board.

INTERFACING BETWEEN SCA AND SCI

There are 51 lines, not counting the maintenance line, used to transfer the data and control functions between the SCA (which contains an A interface) and the SCI (which contains a B interface). Refer to figure 2-20. The interface lines from the SCA to the SCI (25 lines) are defined as follows: (References to A apply to the A interface and references to B apply to the B interface.) The same lines and transfer sequences are used between the other SCAs and other external devices except that there is no maintenance line.

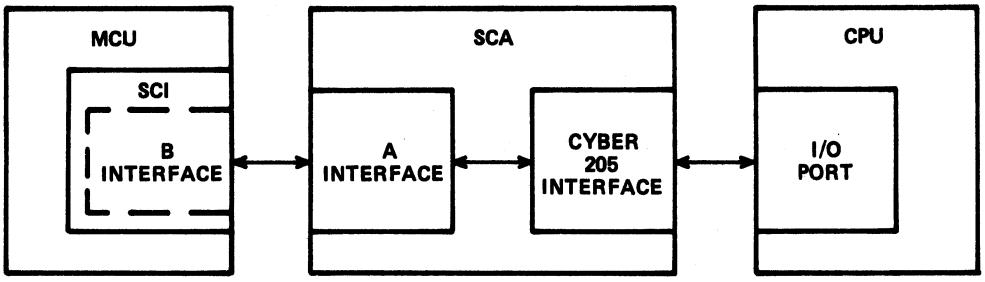


Figure 2-20. System Channel Interface (SCI)

- Data from A (DFA, 16 lines).

  A group of 16 pulsed lines, DFA0 through DFA15, that transmit data to the SCI. DFA0 is the most significant.

- Data Parity from A (DPFA, two lines).

  Two pulsed lines, DPFA0 and DPFA1. DPFA0 forms odd parity with DFA0 through DFA7, and DPFA1 forms odd parity with DFA8 through DFA15. DPFA is transmitted with DFA.

- Request from A (RFA, one line).

  Signals the subsequent presence of data and error information.

- Accept from A (AFA, one line).

  Acknowledges the receipt of a request from B (RFB) and associated information and also signals the subsequent presence of error information. On transmission of AFA, A can accept a new RFB.

- Parity Error from A (PEFA, one line).

  Indicates that one or more of the following errors has been detected.

  - SECDED error
  - Buffer memory parity error
  - Transmission parity error
  - Function parity error

- Illegal from A (IFA, one line)

  Indicates an illegal or invalid function code has been detected.

- Control Strobe from A (CSFA, one line).

  Signals the subsequent presence of control information.

- Control from A (CFA, two lines).

  Two pulsed lines, CFA0 and CFA1, that are coded to indicate control functions. Refer to table 2-6 and Control From A in this section for codes.

The interface lines from the SCI to the SCA (26 lines) are defined as follows:

- Data from B (DFB, 16 lines).

  A group of 16 pulsed lines, DFB0 through DFB15, that transmit data, address fields, and function words to the SCA. DFB0 is the most significant.

- Data Parity from B (DPFB, two lines).

  Two pulsed lines, DPFB0 and DPFB1. DPFB0 forms odd parity with DFB0 through DFB7 and DPFB1 forms odd parity with DFB8 through DFB15. DPFB is transmitted with DFB.

- Request from B (RFB, one line).

  Signals the subsequent presence of function, data, or address information.

- Accept from B (AFB, one line).

  Acknowledges the receipt of an RFA and associated information. On transmission of AFB, B can accept a new RFA.

- Function from B (FFB, three lines).

  Three pulsed lines, FFB0 through FFB2 that are coded to indicate the type of operation to be performed. FFB is transmitted with DFB. Refer to function from B in this section for further explanation.

- Function Parity from B (FPFB, one line).

  FPFB forms odd parity with FFB0 through FFB8. FPFB is transmitted with FFB.

- Interrupt from B (IFB, one line).

  Single pulsed line that is passed on to the CPU.

- Master Clear from B (MCFB, one line).

  Single pulsed line that master clears the SCA.

Control From A

Control from A (CFA) is generated by the CPU and is passed on to the SCI via the SCA along with the CSFA signal. Table 2-7 lists CFA codes.

TABLE 2-7. CONTROL FROM A†

CFA0	CFA1	Function
0	0	Control Flag††
0	1	Channel Flag
1	0	External Flag††
1	1	Suspend

†Refer to system communication (section 2 of this manual) for definition of terms in Function column of this table.
††Not used with CYBER 205 system.

Functions From B

Functions from B (FFB) are transmitted with data from B (DFB) and indicate the type of CYBER 205 memory access operation to be performed. Table 2-8 lists FFB codes.

TABLE 2-8. FUNCTIONS FROM B

FFB0	FFB1	FFB2	Function
0	0	0	Null
0	0	1	Read†
0	1	0	Write
0	1	1	Status read
1	0	0	Data
1	0	1	Block read †
1	1	0	Function write
1	1	1	End of operation

†These functions perform the same operation.

The following is a description of FFB functions:

- Null (FFB 000)

  Null accompanies the second half of an address field or data word and follows the end of operation FFB.

- Read/Block Read (FFB 001/101)

  These functions indicate that DFB contains the most significant 16 bits of the address field and that one or more words are to be transferred from the CYBER 205 to the MCU starting at the specified address. The least significant 16 address bits are accompanied by a null FFB.

- Write (FFB 010)

  This function indicates that DFB contains the most significant 16 bits of the address field and that one or more words are to be transferred from the MCU to the CYBER 205 beginning at the specified address. The least significant 16 address bits are accompanied by a null FFB.

- Status Read (FFB 011)

  The status read function initiates the transfer of three 16-bit status words to the MCU from the SCA. A null FFB follows the status read.

- Data (FFB 100)

  This function indicates that DFB contains the most significant 16 bits for a write data transfer. The least significant 16 bits are accompanied by a null FFB.

- Function Write (FFB 110)

  This function indicates that DFB contains a 16-bit function word from the MCU. A null FFB follows the function write.

- End of Operation (FFB 111)

  This function indicates that the current read or write operation has ended. The end of operation FFB is followed by null FFB.

Status Words

The status read function (FFB 011) initiates transfer of three 16-bit status words from the SCA to the MCU. Error status bits are cleared by the SCA internal master clear signal generated at the beginning of a read or write operation. A null function (FFB 000) must follow the status read.

Table 2-9 is a bit description of the status word:

TABLE 2-9. STATUS WORD 1 BITS AND DESCRIPTIONS

Bit	Description
00-06	Syndrome bits 00-06.
07	When set, this bit indicates detection of SECDED single error.
08	When set, this bit indicates detection of SECDED double error.
09	When set, this bit indicates detection of a parity error (PE0) on bits 00-07 read from the SCA buffer memory.
10	When set, this bit indicates detection of a parity error (PE1) on bits 08-15 read from the SCA buffer memory.
11	When set, this bit indicates detection of a parity error (PE2) on bits 16-23 read from the SCA buffer memory.
12	When set, this bit indicates detection of a parity error (PE3) on bits 24-31 read from the SCA buffer memory.
13	When set, this bit indicates a transmission parity error was detected in data or an address received from the MCU.
14	When set, this bit indicates the CYBER 205 I/O port is disabled (inactive) and will not recognize any data transfer requests.
15	Not used.

- Status Words 2 and 3

  Status word 2 contains the upper 8 address bits of the 32-bit word which had a SECDED error. Status word 3 contains the lower 16 address bits of the 32-bit word which had a SECDED error. Refer to figure 2-21.

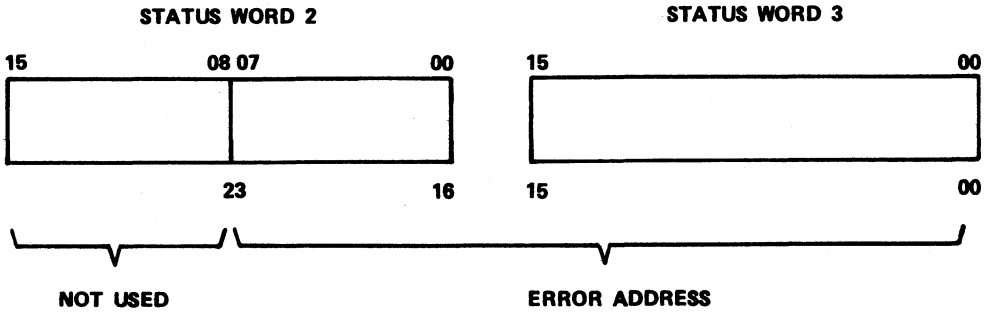


Figure 2-21. Status Words 2 and 3

Function Word

The function write function (FFB 110) indicates that the channel contains a 16-bit function word from the MCU. A null function (FFB 000) must follow the function write.

Table 2-10 is a bit description of the function word.

TABLE 2-10. FUNCTION WORD BITS AND DESCRIPTIONS

Bit	Description
00-06	Complement SECDED checkbits 0-6 sent to the CYBER 205 during a write operation.
07	When set, this bit disables the error correction logic during a read operation.
08	When set, this bit inhibits sending a PEFA signal to the MCU.
09	When set, this bit disables access to SCA buffer memory 1.
10	When set, this bit disables access to SCA buffer memory 2.
11	When set, this bit allows writing or reading the SCA buffer memories without initiating a request to the CYBER 205.
12	When set, this bit prevents setting the SECDED double error bit when a double error is detected.
13	When set, this bit enables setting the SECDED single error bit when a single error is detected.
14	When set, this bit limits the capacity of each buffer memory to one sword (16 words).
15	When set, this bit sends maintenance signal to the CYBER 205 upon initiation of a data transfer to indicate transfer of maintenance information. This bit also limits the capacity of each buffer memory to one sword (16 words).

MAINTENANCE DATA TRANSFERS

Maintenance data transfers occur over the same lines as the CPU memory access and use the same transfer sequences. The difference is one additional line, the maintenance signal. The maintenance signal must accompany all maintenance data transfers. When data are read or written through an I/O port and the maintenance line is a logical one, the data will pass to or from the maintenance data registers instead of central memory. There are two one-sword maintenance data registers; one input register (consisting of DFW0 through DFWD), and one output register (consisting of DFR0 through DFR3). The input register connects to various parts of the CPU to provide control. The output register holds status information to be monitored by the MCU.

The input register is loaded in 32-bit segments through the use of the write enable line sent with each 32-bit half-word of maintenance control data. The output register is connected statically to the CPU and is constantly updated. The MCU software reads this status at the appropriate time.

Tables 2-11 and 2-12 show the bit assignments for the input and output maintenance registers. The first 16 bits of the input register are pulse shaped within the maintenance hardware. All other bits must be set and cleared by alternate transfers of ones and zeros into the register by the maintenance software.

TABLE 2-11. MCU TO CPU DATA

Bit	DFW0 (Sword Bits 00-31)	Bit	DFW1 (Sword Bits 32-63)
00	Master clear	00	Not used
01	Stop	01	↓
02 †	Step	02	
03 †	Run	03	
04 †	Store associative registers	04	Not used
05 †	Load associative registers	05 †	Clock frequency select code Bit 00
06	CPU master clear	06 †	Clock frequency select code Bit 01
07	Clear faults	07 †	Clock frequency select code Bit 02
08	Clear external interrupt initial exchange	08	Not used
09	Not used	09	Increment RTC per minor cycle
10	↓	10	Increment JIT per minor cycle
11		11	Increment MIT per minor cycle
12		12	Not used
13		13 †	Memory size select code Bit 00
14	Not used	14 †	Memory size select code Bit 01
15	Pulse clear (after 1 microsecond DFW0 bits 00-15)	15 †	Memory size select code Bit 02
16	Not used	16	Stop on single SECDED error
17	Not used	17	Stop on double SECDED error
18	Not used	18	SECDED correction enable
19	Not used	19	Block write enables on error
20	Test mode loop count Bit 00	20	Select SECDED mode 2
21	01	21	Clear SECDED single error
22	02	22	Complement memory address bit 39
23	03	23	Not used
24	04	24	↓
25	05	25	
26	06	26	
27	07	27	
28	08	28	
29	09	29	
30	10	30	
31	Test mode loop count Bit 11	31	Not used

†CPU must be stopped before executing these commands.

TABLE 2-11. MCU TO CPU DATA (Contd)

Bit	DFW2 (Sword Bits 64-95)		Bit	DFW3 (Sword Bits 96-127)
00	Send external flag	CH1	00	Not used
01		CH2	01	
02		CH3	02	
03		CH4	03	
04		CH5	04	
05		CH6	05	
06		CH7	06	
07		CH8	07	Not used
08		CH9	08	Stop on microcode parity error
09		CH10	09	Stop on instruction stack parity error
10		CH11		
11		CH12	10	Stop on multiple match
12		CH13	11	Stop on bounds hit
13		CH14	12	Check bounds on CPU references
14		CH15	13	Check bounds on I/O references
15	Send external flag	CH16	14	Check bounds on read
16	Channel enable	CH1	15	Check bounds on write
17		CH2	16	Not used
18		CH3	17	Not used
19		CH4	18	Block external interrupt
20		CH5	19†	Interrupt gate
21		CH6	20†	Force instruction stack parity error
22		CH7		
23		CH8	21†	Swap register file on exchange
24		CH9	22†	Store register file 20000
25		CH10	23	Not used
26		CH11	24	
27		CH12	25	
28		CH13	26	
29		CH14	27	
30		CH15	28	
31	Channel enable	CH16	29	
			30	
			31	
			32	Not used

†The CPU must be stopped before executing these commands.

TABLE 2-11. MCU TO CPU DATA (Contd)

Bit	DFW4 (Sword Bits 128-159)	Bit	DFW5 (Sword Bits 160-191)
00	Not used	00	Not used
01	Not used	01	Not used
02	Not used	02	Not used
03	Upper bounds address 35†	03	Lower bounds address 35
04	36	04	36
05	37	05	37
06	38	06	38
07	39	07	39
08	40	08	40
09	41	09	41
10	42	10	42
11	43	11	43
12	44	12	44
13	45	13	45
14	46	14	46
15	47	15	47
16	48	16	48
17	49	17	49
18	50	18	50
19	51	19	51
20	52	20	52
21	53	21	53
22	54	22	54
23	55	23	55
24	56	24	56
25	57	25	57
26	Upper bounds address 58	26	Lower bounds address 58
27	Not used	27	Not used
28		28	
29		29	
30		30	
31	Not used	31	Not used

†The MCU transmits the false state of the upper bounds address, DFW4 bits 03 through 26.

TABLE 2-11. MCU TO CPU DATA (Contd)

Bit	DFW6 (Sword Bits 192-223)	Bit	DFW7 (Sword Bits 224-255)
00	Not used	00	Not used
01	↓	01	↓
02		02	
03		03	
04		04	
05		05	
06		06	
07		07	
08		08	
09		09	
10		10	
11		11	
12		12	
13		13	
14		14	
15		15	
16		16	
17		17	
18		18	
19		19	
20		20	
21		21	
22		22	
23		23	
24		24	
25		25	
26		26	
27		27	
28		28	
29		29	
30	↓	30	↓
31	Not used	31	Not used

TABLE 2-11. MCU TO CPU DATA (Contd)

Bit	DFW8 (Sword Bits 256-287)	Bit	DFW9 (Sword Bits 288-319)
00	Auxiliary board select code Bit 00	00	Write microcode data word 1 Bit 00
01	↓	01	↓
02	↓	02	↓
03	↓	03	↓
04	↓	04	↓
05	Auxiliary board select code Bit 05	05	↓
06 †	Write enable select code Bit 00	06	↓
07 †	Write enable select code Bit 01	07	↓
08 †	Write microcode	08	↓
09 †	Sweep microcode	09	↓
10 †	Clear microcode address	10	↓
11 †	Enable PM01	11	Write microcode data word 1 Bit 11
12 †	Read microcode	12	Write microcode data word 2 Bit 00
13	Not used	13	↓
14	↓	14	↓
15	↓	15	↓
16	↓	16	↓
17	↓	17	↓
18	↓	18	↓
19	↓	19	↓
20	↓	20	↓
21	↓	21	↓
22	↓	22	↓
23	↓	23	Write microcode data word 2 Bit 11
24	↓	24	Write microcode data word 3 Bit 00
25	↓	25	↓
26	↓	26	↓
27	↓	27	↓
28	↓	28	↓
29	↓	29	↓
30	↓	30	↓
31	Not used	31	Write microcode data word 3 Bit 07

†CPU must be stopped before executing these commands.

TABLE 2-11. MCU TO CPU DATA (Contd)

Bit	DFWA (Sword Bits 320-351)	Bit	DFWB (Sword Bits 352-383)
00	Write microcode data word 3 Bit 08	00	Write microcode data word 6 Bit 04
01	09	01	05
02	10	02	06
03	Write microcode data word 3 Bit 11	03	07
04	Write microcode data word 4 Bit 00	04	08
05	01	05	09
06	02	06	10
07	03	07	Write microcode data word 6 Bit 11
08	04	08	Write microcode data word 7 Bit 00
09	05	09	01
10	06	10	02
11	07	11	03
12	08	12	04
13	09	13	05
14	10	14	06
15	Write microcode data word 4 Bit 11	15	07
16	Write microcode data word 5 Bit 00	16	08
17	01	17	09
18	02	18	10
19	03	19	Write microcode data word 7 Bit 11
20	04	20	Write microcode data word 8 Bit 00
21	05	21	01
22	06	22	02
23	07	23	03
24	08	24	04
25	09	25	05
26	10	26	06
27	Write microcode data word 5 Bit 11	27	07
28	Write microcode data word 6 Bit 00	28	08
29	01	29	09
30	02	30	10
31	Write microcode data word 6 Bit 03	31	Write microcode data word 8 Bit 11

NOTE: Half-words DFWC and DFWF are not used.

TABLE 2-12. CPU TO MCU DATA

Bit	DFR0 (Sword Bits 00-127) (If DFW8 Bit 12 is a Zero)
00	SECDED fault
01	Microcode parity error
02	Instruction stack parity error
03	Multiple match fault
04	Bounds hit
05	Not used
06	Not used
07	Not used
08	Temperature dewpoint alarm
09	Section power failure
10	Main memory 60-Hz power
11	Optional memory 60-Hz power
12	Not used
13	Monitor mode
14	Idle
15	Stopped
16	CIAR bit 00
↓	↓
63	CIAR bit 47
64	CIR bit 00
↓	↓
127	CIR bit 63

Bit	DFR0 Bits 48-127 (If DFW8 Bit 12 is a One)		
48	PM00/01	BRD1	Bit 00
49	PM00/01	BRD1	Bit 01
50	PM00/01	BRD2	Bit 00
51	PM00/01	BRD2	Bit 01
52	PM00/01	BRD3	Bit 00
53	PM00/01	BRD3	Bit 01
54	PM00/01	BRD4	Bit 00
55	PM00/01	BRD4	Bit 01
56	PM00/01	BRD5	Bit 00
57	PM00/01	BRD5	Bit 01
58	DM00	BRD5	Bit 00
59	DM00	BRD5	Bit 01

Bit	DFR0 (Sword Bits 48-127) (If DFW8 Bit 12 is a One) (Contd)		
60	DM00		Bit 02
61	DM00		Bit 03
62	GM00		Bit 00
63	GM00		Bit 01
64	GM00		Bit 02
65	GM00		Bit 03
66	HM00	BRD1	Bit 00
67	HM00	BRD1	Bit 01
68	HM00	BRD1	Bit 02
69	HM00	BRD1	Bit 03
70	HM00	BRD2	Bit 00
71	HM00	BRD2	Bit 01
72	HM00	BRD2	Bit 02
73	HM00	BRD2	Bit 03
74	Not used		
↓	↓		
89	Not used		
90	SM00	BRD1	
91	SM00	BRD2	
92	SM00	BRD3	
93	SM00	BRD4	
94	SM00	BRD5	
95	SM00	BRD6	
96	SM00	BRD7	
97	SM00	BRD8	
98	SM00	BRD9	
99	SM00	BRD10	
100	VM00	BRD1	
101	VM00	BRD2	
102	VM00	BRD3	
103	VM00	BRD4	
104	VM00	BRD5	
105	VM00	BRD6	
106	VM00	BRD7	
107	VM00	BRD8	
108	VM00	BRD9	
109	VM00	BRD10	
110	WM00	BRD1	Bit 00
111	WM00	BRD1	Bit 01

TABLE 2-12. CPU TO MCU DATA (Contd)

Bit	DFR0 (Sword Bits 48-127) (If DFW8 Bit 12 is a One) (Contd)	
112	WM00 BRD2	Bit 00
113	WM00 BRD2	Bit 01
114	WM00 BRD3	Bit 00
115	WM00 BRD3	Bit 01
116	WM00 BRD4	Bit 00
117	WM00 BRD4	Bit 01
118	Not used	
↓	↓	
127	Not used	

Bit	DFR1 (Sword Bits 128-255)	
00	Data flag register	Bit 00
↓	↓	↓
63	Data flag register	Bit 63
64	Not used	
65	Not used	
66	Not used	
67	Invisible package address	Bit 00
↓	↓	↓
86	Invisible package address	Bit 19
87	Not used	
↓	↓	
98	Not used	
99	Page zero address	Bit 35
↓	↓	↓
112	Page zero address	Bit 48
113	Not used	
↓	↓	
127	Not used	

Bit	DFR2 (Sword Bits 256-383)	
00	External interrupt register	CH1
01		CH2
02		CH3
03	External interrupt register	CH4

Bit	DFR2 (Sword Bits 256-383) (Contd)		
04	External interrupt register		CH5
05			CH6
06			CH7
07			CH8
08			CH9
09			CH10
10			CH11
11			CH12
12			CH13
13			CH14
14			CH15
15	External interrupt register		CH16
16	Monitor timer interrupt		
17	Not used		
18	SECDED double error		
19	SECDED single error		
20	Space table search in process		
21	Bus code	Bit 00	
22	Bus code	Bit 01	
23	Bus code	Bit 02	
24	Not used		
25	Syndrome	Bit 00	
26		01	
27		02	
28		03	
29		04	
30		05	
31	Syndrome	Bit 06	
32	Not used		
33	Not used		
34	Not used		
35	SECDED fault address	Bit 35	
↓	↓	↓	
63	SECDED fault address	Bit 63	

TABLE 2-12. CPU TO MCU DATA (Contd)

Bit	DFR2 (Sword Bits 256-383) (Contd)		Bit	DFR3 (Sword Bits 384-511)	
64	Channel read active	CH1	00	Microcode parity error	Board 00
65		CH2	↓		
66		CH3	08	Microcode parity error	Board 08
67		CH4	09	Not used	
68		CH5	↓		
69		CH6	12	Not used	
70		CH7	13	Microcode parity error	Board 13
71		CH8	↓		
72		CH9	36	Microcode parity error	Board 36
73		CH10	37	Not used	
74		CH11	38	Microcode address	Bit 00
75		CH12	39		01
76		CH13	40		02
77		CH14	41		03
78		CH15	42		04
79	Channel read active	CH16	43		05
80	Channel write active	CH1	44		06
81		CH2	45		07
82		CH3	46		08
83		CH4	47	Microcode address	Bit 09
84		CH5	48	PM01	
85		CH6	49	Scalar microcode parity error	
86		CH7	50	Not used	
87		CH8	↓		
88		CH9	↓		
89		CH10	127	Not used	
90		CH11			
91		CH12			
92		CH13			
93		CH14			
94		CH15			
95	Channel write active	CH16			
96	Not used				
97	Not used				
98	Not used				
99	Bounds hit address	Bit 35			
↓	↓	↓			
127	Bounds hit address	Bit 63			

OPERATING INSTRUCTIONS 3

CONTROLS AND INDICATORS

All of the controls and indicators necessary to operate the CYBER 205 are located on the MCU console and wall box. Refer to the appropriate manual listed in the preface for a description of the MCU controls and indicators.

STARTUP PROCEDURES

The system startup procedures are explained in table 3-1.

TABLE 3-1. STARTUP PROCEDURES

Step	Action Required	Result
1.	Press SYSTEM START button at wall box.	a. Applies 400 Hz, 208 V ac to system. b. POWER ON light illuminates. c. SYSTEM START button remains energized for 1 to 2 seconds. d. Power is applied to intersection 24 V dc power supply. The 24 VDC light illuminates e. After a 3- to 5-minute delay, motor generator lights. MG 1, 2, 3, and 4 illuminate.
2.	Run command buffer diagnostics (provided by site customer engineer).	Verifies that machine is operational and capable of taking autoload.

TABLE 3-1. STARTUP PROCEDURES (Contd)

Step	Action Required	Result
3.	Autoload MCU as follows: **NOTE** The control terminal is the terminal connected to CLA port 0 of the MCU. a. Press control terminal keyboard PAGE key so that key stays up. b. Perform either step 3c or 3d depending on which is more convenient. c. Press the following MCU console switches in specified order. STOP MASTER AUTOLOAD RUN **NOTE** In order to use the control terminal, MCU-OS must have been previously loaded and reside, intact, in MCU memory. d. Use control terminal keyboard to perform the following activity in specified order. Press: ESC key **NOTE** Press SHIFT and ? keys simultaneously. Enter: ? Press: ESC key Enter: J11G HG K8000G I@	

TABLE 3-1. STARTUP PROCEDURES (Contd)

Step	Action Required	Result
	e. Press ECS key at control terminal and enter the following: J28@	
	f. Enter date and time in following format: mmddyyhhmm (mm=month, 01-12; dd=day, 01-31; yy=year, 00-99; hh=hour, 00-23; and minute, 00-59)	
	g. Press CARRIAGE RETURN key.	
	h. Press CONTROL key and G key simultaneously.	Causes manual interrupt.
	i. Press control terminal PAGE key so that key stays down.	
	j. At control terminal enter the following: C205,x x = 0 to enable line 0/1 and SECDED monitor (default). = 1 to disable line 0/1 display. = 2 to disable SECDED monitor (RSR, CRE, and SLOG commands made inoperative). = 3 to disable line 0/1 display and SECDED monitor.	C205,x
	k. Press CARRIAGE RETURN key.	MCU commands are now executable from the terminals.
4.	Master clear the system from MCU.	Initializes CPU (clears all control flip-flops, data flags, interrupts, and error flip-flops). Sets monitor mode in CPU (Job Mode flip-flop cleared by master clear).

TABLE 3-1. STARTUP PROCEDURES (Contd)

Step	Action Required	Result
5.	Load microcode into the vector setup unit and the scalar unit from MCU.	MCU sends an external flag via CPU I/O ports to I/O devices required on-line. I/O devices receiving this flag will autoload and enter an idle loop waiting for a channel flag from the CPU. An alternative approach is to manually autoload each of the I/O devices required on-line. MCU loads operating kernel into central memory, and then interrupts the CPU. CPU recognizes the interrupt and executes a partial exchange to start execution in monitor mode. This exchange is the same as a normal job to monitor exchange except the contents of the register file and invisible package are not stored. Program execution starts at the address contained in monitor's register six just as it does after a normal I/O interrupt.

OPERATING PROCEDURES

Refer to the appropriate manual listed in the preface for operating procedures.

SYSTEM STOP (NORMAL)

Table 3-2 describes procedures for a normal system stop.

TABLE 3-2. SYSTEM STOP

Step	Action Required	Result
1.	Press SYSTEM STOP button at wall box.	a. SYSTEM STOP button remains illuminated for 1 to 2 seconds. b. Power is removed from motor generators. MG 1, 2, 3, and 4 and POWER ON lights go out. c. Power is removed from the peripherals. d. Power is removed from the 24 V dc power supply. The 24 VDC light remains energized for approximately 3 minutes.

EMERGENCY STOP

Table 3-3 gives the procedure for an emergency system stop.

TABLE 3-3. EMERGENCY STOP

Step	Action Required	Result
1.	Press SYSTEM OFF button at wall box.	Removes 400 Hz, 208 V ac power from wall box. All systems power down without delay.

INSTRUCTION DESCRIPTIONS 4

GENERAL

This section describes instruction word formats, instruction types, and instruction descriptions. The instruction word format description explains the content of 32-bit and 64-bit instruction formats used in the computer. The instruction type description explains the instruction groups according to the operations they perform. The instruction description gives detailed explanations and examples of individual instructions.

As an aid in finding instruction designator information and individual instruction descriptions, refer to:

- Table 4-1 for instruction designators.
- Table 4-2 or inside front cover for locating instructions by function code.
- Table 4-3 for locating instructions by instruction type.

INSTRUCTION WORD FORMATS

The 32-bit and 64-bit instruction words have 12 types of formats (figure 4-1). The formats have hexadecimal numbers, 1 through C, which are used as references in tables 4-2 and 4-3. The bits in the instruction word formats number from left to right, 0 through 31 or 0 through 63.

INSTRUCTION DESIGNATORS

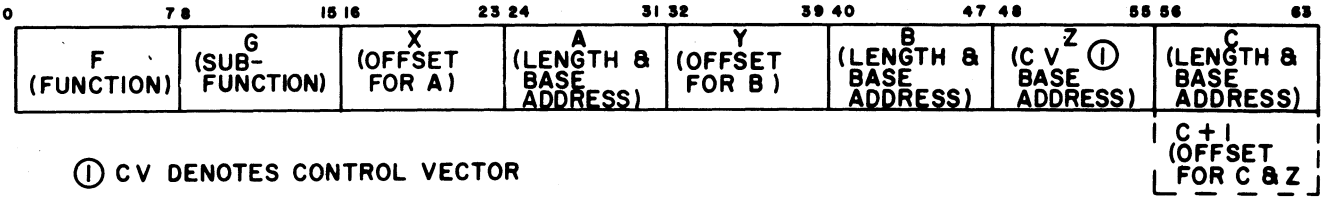
Each instruction word format is divided into bit groups that have assigned instruction designators shown in figure 4-1. The designator letters (such as F, R, S, and T in format 4) and their definitions are listed in table 4-1. The definitions are general and may vary between instructions. The instruction descriptions give more specific designator information as it applies to individual instructions.

When the C + 1 designator is used, the C designator must specify an even-numbered register. If the C designator specifies an odd-numbered register, the results of the instruction become undefined.

Bits 0 through 7 are commonly used by each instruction word as the function code designator (F). The computer uses function codes in the range of 00 through FF. The function codes in the range of 00 through 7F use 32-bit instruction word formats. The function codes in the range of 80 through FF use 64-bit instruction word formats.

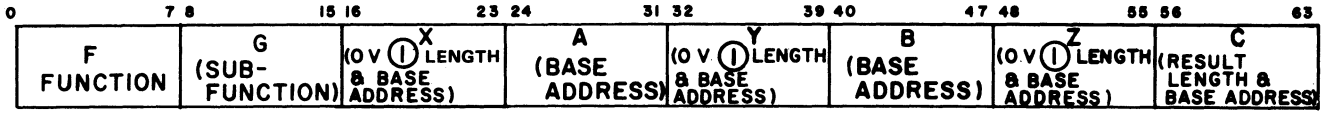
UNUSED BIT AREAS

Cross-hatched lines like those shown in formats A, B, and C of figure 4-1 indicate unused bit areas. These areas must be cleared to all zeros or the instructions may cause undefined results or operations.



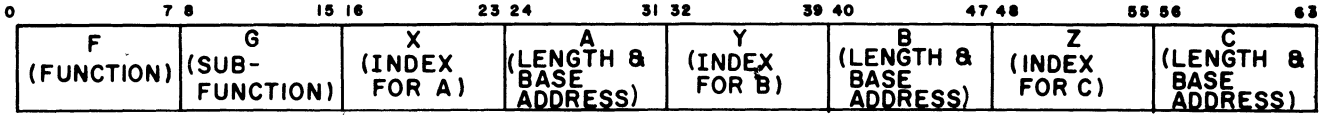
① C V DENOTES CONTROL VECTOR

FORMAT 1 - USED FOR VECTOR, VECTOR MACRO, AND SOME NONTYPICAL INSTRUCTIONS

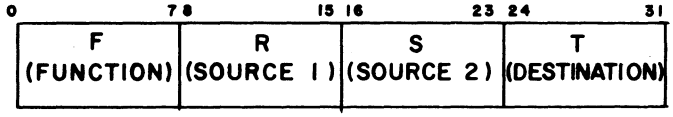


① 0 V DENOTES ORDER VECTOR

FORMAT 2 - USED FOR SPARSE VECTOR AND SOME NONTYPICAL INSTRUCTIONS

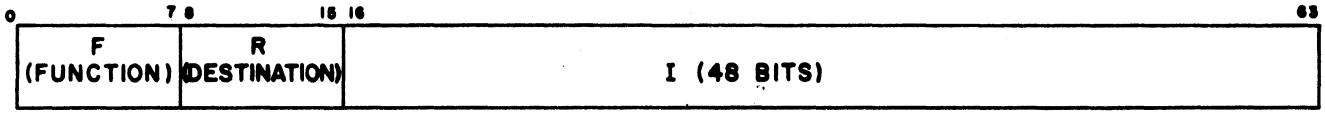


FORMAT 3 USED FOR LOGICAL STRING AND STRING INSTRUCTIONS

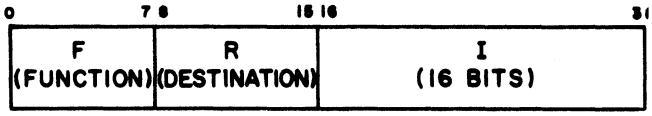


FORMAT 4 USED FOR SOME REGISTER, ALL MONITOR, THE 3D AND 04 NONTYPICAL INSTRUCTIONS

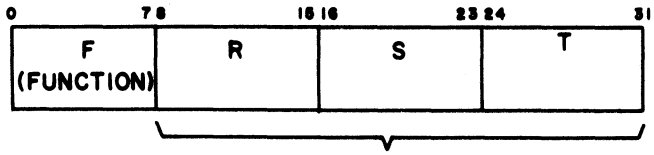
Figure 4-1. Instruction Formats (Sheet 1 of 3)



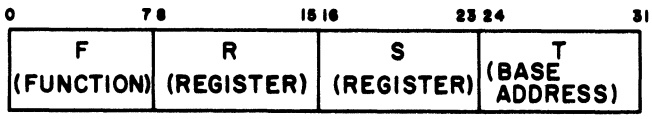
FORMAT 5 USED FOR THE BE, BF, CD, AND CE INDEX INSTRUCTIONS AND FOR THE B6 BRANCH INSTRUCTION



FORMAT 6 USED FOR THE 3E, 3F, 4D, AND 4E INDEX INSTRUCTIONS AND THE 2A REGISTER INSTRUCTION

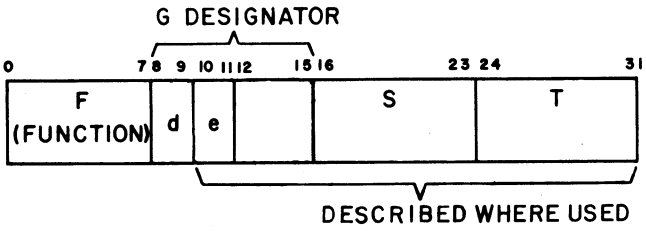


FORMAT 7 USED FOR SOME BRANCH AND NONTYPICAL INSTRUCTIONS

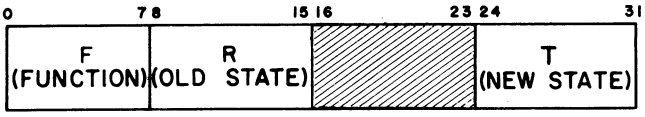


FORMAT 8 USED FOR SOME BRANCH INSTRUCTIONS

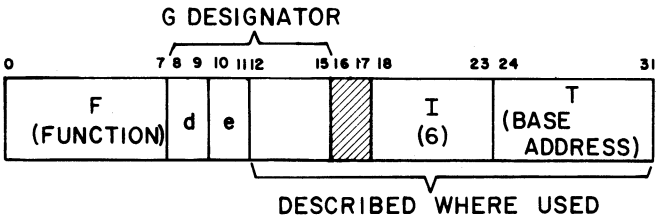
Figure 4-1. Instruction Formats (Sheet 2 of 3)



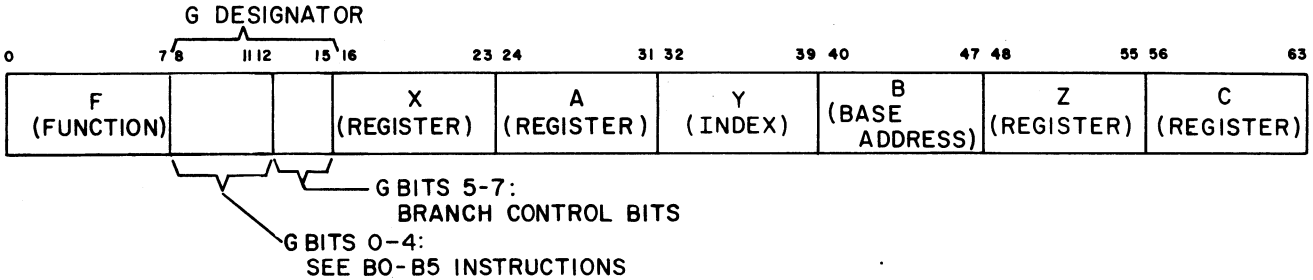
FORMAT 9 USED FOR THE 32 BRANCH INSTRUCTION



FORMAT A USED FOR SOME INDEX, BRANCH, AND REGISTER INSTRUCTIONS



FORMAT B USED FOR THE 33 BRANCH INSTRUCTION



FORMAT C USED FOR THE BO – B5 BRANCH INSTRUCTIONS

Figure 4-1. Instruction Formats (Sheet 3 of 3)

TABLE 4-1. INSTRUCTION DESIGNATORS

Designator	Format Type	Definition
A	1 and 3	This 8-bit designator specifies a register that contains a field length and base address for the corresponding source vector or string field.
	2	This 8-bit designator specifies a register that contains the base address for a source sparse vector field.
	C	Specifies a register that contains a two's complement integer in the rightmost 48 bits.
B	1 and 3	This 8-bit designator specifies a register that contains a field length and base address for the corresponding source vector or string field.
	2	This 8-bit designator specifies a register that contains the base address for a source sparse vector field.
	C	This 8-bit designator specifies a register that contains the branch base address in the rightmost 48 bits.
C	1,2, and 3	This 8-bit designator specifies a register that contains the field length and base address for storing the result vector, sparse vector, or string field.
	C	Specifies the register that contains the two's complement sum of (A) + (X) in the rightmost 48 bits. The leftmost 16 bits are cleared.
C+1	1	This 8-bit designator specifies a register that contains the offset for the C and Z vector fields.
d	9 and B	This 2-bit designator is contained within the G designator and specifies the branch conditions for the corresponding branch instructions.

TABLE 4-1. INSTRUCTION DESIGNATORS (Contd)

Designator	Format Type	Definition
e	9 and B	This 2-bit designator is contained within the G designator and specifies the object bit altering conditions for the corresponding branch instructions.
F	1 - C	This 8-bit designator is used in all instruction format types to specify the instruction function code. This designator is always contained in the leftmost 8 bits of the instruction and is expressed in hexadecimal for all instruction descriptions. Thus, the function code range is $00\text{-}FF_{16}$. However, not all of the possible function codes are used.
G	1,2,3,9,B, and C	This 8-bit designator specifies certain subfunction conditions for the corresponding instruction. The subfunctions include the length of the operands (32- or 64-bit), normal or broadcast source vectors, etc. The number of bits that are used in the G designator vary with individual instructions. (Appendix C lists the G bit usage codes according to function code.) The G designator bits have bit positions 8 through 15 in the word format. The manual references these bits as G bits 0 through 7. G bit 0 corresponds to bit position 8 in the word format. Other G bits follow, in order, from left to right.
I	5	This 48-bit designator functions as an index used to form the branch address in a B6 branch instruction. In the CD and CE index instructions, operand I is contained in the rightmost 24 bits. In the BE and BF index instructions, I is a 48-bit operand.
	6	In the 3E, 3F, 4D, and 4E index instructions, I functions as a 16-bit operand.
	B	In the 33 branch instruction, the 6-bit I designator specifies the number of the data flag branch register bit used in the branching operation.

TABLE 4-1. INSTRUCTION DESIGNATORS (Contd)

Designator	Format Type	Definition
R	4	This 8-bit designator specifies a register that contains an operand to be used in arithmetic operation in the register and 3D instructions.
	5 and 6	In the BE, BF, CD, CE, 3E, 3F, 4D, and 4E index instructions, R functions as a destination register for the transfer of an operand or operand sum. In the B6 branch instruction, R specifies a register that contains an item count which is used to form the branch address.
	7,8, and A	In these format types, R specifies registers and branching conditions that are described in the individual instruction descriptions.
S	4	This 8-bit designator specifies a register that contains an operand to be used in an arithmetic operation in the register and 3D instructions.
	7,8, and 9	In these format types, S specifies registers and branching conditions that are described in the individual instruction descriptions.
T	4	This 8-bit designator specifies a destination register for the transfer of the arithmetic results.
	7,8,9, and B	In these formats, T specifies a register that contains the base address, and in some cases, the field length of the corresponding result field or branch address.
	A	In this format, T specifies a register that contains the old state of a register, data flag branch register, etc.; in an index, branch or inter-register transfer operation.

TABLE 4-1. INSTRUCTION DESIGNATORS (Contd)

Designator	Format Type	Definition
X	1 and 3	This 8-bit designator specifies a register that contains the offset or index for vector or string source field A.
	2	In this case, X specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field A.
	C	In the B0 through B5 branch instructions, X specifies a register that contains a signed, two's complement integer in the rightmost 48 bits which is used as an operand in the branching operation.
Y	1 and 3	This 8-bit designator specifies a register that contains the offset or index for vector or string field B.
	2	In this format, Y specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field B.
	C	In the B0 through B5 branch instructions, Y specifies a register that contains an index that is used to form the branch address.
Z	1	This 8-bit designator specifies a register that contains the base address for the control vector used to control the result vector in field C.
	2	In this case, Z specifies a register that contains the length and base address for the order vector corresponding to sparse vector field C.
	3	In this format, Z specifies a register that contains the index for result field C.
	C	In the B0 through B5 branch instructions, Z specifies a register that contains a signed two's complement integer in the rightmost 48 bits. This integer is used as the comparison operand in determining whether the branch condition is met.

INSTRUCTION TYPES

The following 10 types of instructions are grouped according to the operations they perform.

1. Index (IN)
2. Register (RG)
3. Branch (BR)
4. Vector (VT)
5. Sparse vector (SV)
6. Vector macro (VM)
7. String (ST)
8. Logical string (LS)
9. Nontypical (NT)
10. Monitor (MN)

Table 4-2 lists each instruction code in the central computer instruction repertoire; the list is in the numerical order (hexadecimal) of the function code. Table 4-3 lists the instruction codes according to general type; the general types are in the same order as previously listed. The unused and illegal function codes are omitted from tables 4-2 and 4-3.

A page number is given for each instruction code in tables 4-2 and 4-3. These page numbers refer to the description of the corresponding instruction. Figure 4-2 provides additional explanations for using the tables.

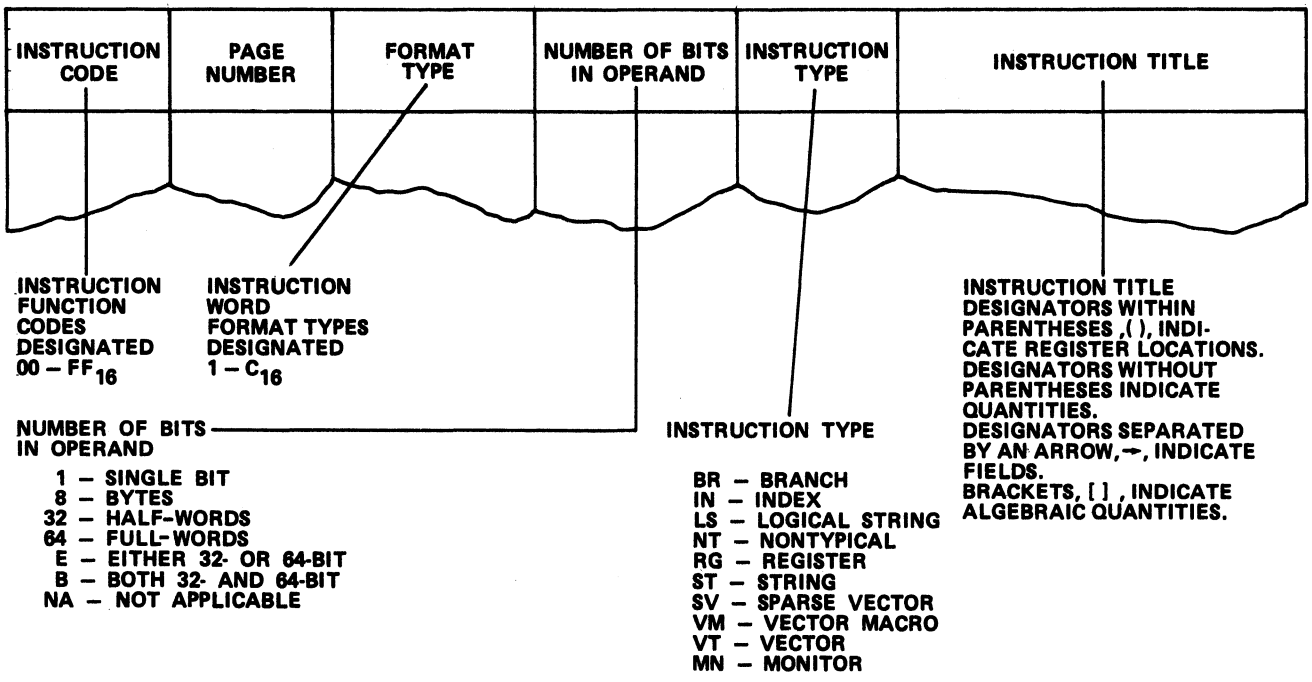


Figure 4-2. Instruction Listing Format

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
00	4-155	4	NA	MN	IDLE
03	4-153	4	64	NT	KEYPOINT - MAINTENANCE
04	4-153	4	64	NT	BREAKPOINT - MAINTENANCE
05	4-154	4	64	NT	VOID STACK AND BRANCH
06	4-155	7	NA	MN	FAULT TEST - MAINTENANCE
08	4-156	4	64	MN	INPUT/OUTPUT PER R
09	4-58	4	64	BR	EXIT FORCE
0A	4-157	4	64	MN	TRANSMIT (R) TO MONITOR INTERVAL TIMER
0C	4-157	4	64	MN	STORE ASSOCIATIVE REGISTERS
0D	4-157	4	64	MN	LOAD ASSOCIATIVE REGISTERS
0E	4-157	4	64	MN	TRANSLATE EXTERNAL INTERRUPT
0F	4-157	4	64	MN	LOAD KEYS FROM (R), TRANSLATE ADDRESS (S) TO (T)
10	4-42	A	64	RG	CONVERT BCD TO BINARY, FIXED LENGTH
11	4-42	A	64	RG	CONVERT BINARY TO BCD, FIXED LENGTH
12	4-122	7	64	NT	LOAD BYTE (T) PER (S), (R)
13	4-122	7	64	NT	STORE BYTE (T) PER (S), (R)
14	4-131	7	1	NT	BIT COMPRESS
15	4-133	7	1	NT	BIT MERGE
16	4-133	7	1	NT	BIT MASK

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
1C	4-150	7	1	NT	FORM REPEATED BIT MASK WITH LEADING ZEROS
1D	4-150	7	1	NT	FORM REPEATED BIT MASK WITH LEADING ONES
1E	4-151	7	1	NT	COUNT LEADING EQUALS
1F	4-153	7	1	NT	COUNT ONES IN FIELD R, COUNT TO (T)
20	4-51	8	32	BR	BRANCH IF (R)=(S)(32 BIT FP)
21	4-51	8	32	BR	BRANCH IF (R)≠(S)(32 BIT FP)
22	4-51	8	32	BR	BRANCH IF (R)≥(S)(32 BIT FP)
23	4-51	8	32	BR	BRANCH IF (R)<(S)(32 BIT FP)
24	4-51	8	64	BR	BRANCH IF (R)=(S)(64 BIT FP)
25	4-51	8	64	BR	BRANCH IF (R)≠(S)(64 BIT FP)
26	4-51	8	64	BR	BRANCH IF (R)≥(S)(64 BIT FP)
27	4-51	8	64	BR	BRANCH IF (R)<(S)(64 BIT FP)
28	4-150	7	8	NT	SCAN EQUAL
2A	4-50	6	64	RG	ENTER LENGTH OF (R) WITH I (16 BITS)
2B	4-50	4	64	RG	ADD TO LENGTH FIELD
2C	4-34	4	64	RG	LOGICAL EXCLUSIVE OR (R), (S), TO (T)
2D	4-34	4	64	RG	LOGICAL AND (R), (S), TO (T)
2E	4-34	4	64	RG	LOGICAL INCLUSIVE OR (R), (S), TO (T)
2F	4-51	9	1	BR	REGISTER BIT BRANCH AND ALTER
30	4-35	7	64	RG	SHIFT (R) PER S TO (T)
31	4-58	7	64	BR	INCREASE (R) AND BRANCH IF (R) ≠ 0

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
32	4-55	9	1	BR	BIT BRANCH AND ALTER
33	4-52	B	1	BR	DATA FLAG REGISTER BIT BRANCH AND ALTER
34	4-35	4	64	RG	SHIFT (R) PER (S) TO (T)
35	4-58	7	64	BR	DECREASE (R) AND BRANCH IF (R) $\neq$ 0
36	4-58	7	64	BR	BRANCH AND SET (R) TO NEXT INSTRUCTION
37	4-123	A	64	NT	TRANSMIT JOB INTERVAL TIMER TO (T)
38	4-34	A	64	IN	TRANSMIT (R BITS 00-15) TO (T BITS 00-15)
39	4-124	A	64	NT	TRANSMIT REAL-TIME CLOCK TO (T)
3A	4-124	A	64	NT	TRANSMIT (R) TO JOB INTERVAL TIMER
3B	4-55	A	64	BR	DATA FLAG REGISTER LOAD/STORE
3C	4-122	4	32	NT	HALF-WORD INDEX MULTIPLY (R)$\cdot$(S) TO (T)
3D	4-122	4	64	NT	INDEX MULTIPLE (R)$\cdot$(S) TO (T)
3E	4-32	6	64	IN	ENTER (R) WITH I (16 BITS)
3F	4-32	6	64	IN	INCREASE (R) BY I (16 BITS)
40	4-38	4	32	RG	ADD U; (R) + (S) TO (T)
41	4-38	4	32	RG	ADD L; (R) + (S) TO (T)
42	4-38	4	32	RG	ADD N; (R) + (S) TO (T)
44	4-38	4	32	RG	SUB U; (R) - (S) TO (T)
45	4-38	4	32	RG	SUB L; (R) - (S) TO (T)
46	4-38	4	32	RG	SUB N; (R) - (S) TO (T)

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
48	4-38	4	32	RG	MPY U; (R)·(S) TO (T)
49	4-38	4	32	RG	MPY L; (R)·(S) TO (T)
4B	4-38	4	32	RG	MPY S; (R)·(S) TO (T)
4C	4-38	4	32	RG	DIV U; (R)/(S) TO (T)
4D	4-32	6	32	IN	HALF-WORD ENTER (R) WITH I (16 BITS)
4E	4-32	6	32	IN	HALF-WORD INCREASE (R) BY I (16 BITS)
4F	4-38	4	32	RG	DIV S; (R)/(S) TO (T)
50	4-39	A	32	RG	TRUNCATE (R) TO (T)
51	4-39	A	32	RG	FLOOR (R) TO (T)
52	4-39	A	32	RG	CEILING (R) TO (T)
53	4-42	A	32	RG	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
54	4-48	4	32	RG	ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)
55	4-48	4	32	RG	ADJUST EXPONENT OF (R) PER (S) TO (T)
56	4-137	7	NA	NT	SELECT LINK
58	4-39	A	32	RG	TRANSMIT (R) TO (T)
59	4-39	A	32	RG	ABSOLUTE (R) TO (T)
5A	4-39	A	32	RG	EXPONENT OF (R) TO (T)
5B	4-42	4	32	RG	PACK (R), (S) TO (T)
5C	4-42	A	B	RG	EXTEND 32 BIT (R) TO 64 BIT (T)
5D	4-42	A	B	RG	INDEX EXTEND 32 BIT (R) TO 64 BIT (T)
5E	4-122	7	32	NT	LOAD (T) PER (S), (R)
5F	4-122	7	32	NT	STORE (T) PER (S), (R)

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
60	4-38	4	64	RG	ADD U; (R) + (S) TO (T)
61	4-38	4	64	RG	ADD L; (R) + (S) TO (T)
62	4-38	4	64	RG	ADD N; (R) + (S) TO (T)
63	4-39	4	64	RG	ADD ADDRESS (R) + (S) TO (T)
64	4-38	4	63	RG	SUB U; (R) - (S) TO (T)
65	4-38	4	64	RG	SUB L; (R) - (S) TO (T)
66	4-38	4	64	RG	SUB N; (R) - (S) TO (T)
67	4-39	4	64	RG	SUB ADDRESS (R) - (S) TO (T)
68	4-38	4	64	RG	MPY U; (R)•(S) TO (T)
69	4-38	4	64	RG	MPY L; (R)•(S) TO (T)
6B	4-38	4	64	RG	MPY S; (R)•(S) TO (T)
6C	4-38	4	64	RG	DIV U; (R) / (S) TO (T)
6D	4-36	4	64	RG	INSERT BITS FROM (R) TO (T) PER (S)
6E	4-37	4	64	RG	EXTRACT BITS FROM (R) TO (T) PER (S)
6F	4-38	4	64	RG	DIV S; (R) / (S) TO (T)
70	4-39	A	64	RG	TRUNCATE (R) TO (T)
71	4-39	A	64	RG	FLOOR (R) TO (T)
72	4-39	A	64	RG	CEILING (R) TO (T)
73	4-42	A	64	RG	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
74	4-48	4	64	RG	ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)
75	4-48	4	64	RG	ADJUST EXPONENT OF (R) PER (S) TO (T)
76	4-42	A	B	RG	CONTRACT 64 BIT (R) TO 32 BIT (T)

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
77	4-42	A	B	RG	ROUNDED CONTRACT 64 BIT (R) TO 32 BIT (T)
78	4-39	A	64	RG	TRANSMIT (R) TO (T)
79	4-39	A	64	RG	ABSOLUTE (R) TO (T)
7A	4-39	A	64	RG	EXPONENT OF (R) TO (T)
7B	4-42	4	64	RG	PACK (R), (S) TO (T)
7C	4-42	A	64	RG	LENGTH OF (R) TO (T)
7D	4-123	7	64	NT	SWAP S→T AND R→S
7E	4-122	7	64	NT	LOAD (T) PER (S), (R)
7F	4-122	7	64	NT	STORE (T) PER (S), (R)
80 †	4-73	1	E	VT	ADD U; A + B→C
81 †	4-73	1	E	VT	ADD L; A + B→C
82 †	4-73	1	E	VT	ADD N; A + B→C
83	4-74	1	64	VT	ADD A; A + B→C
84 †	4-73	1	E	VT	SUB U; A - B→C
85 †	4-73	1	E	VT	SUB L; A - B→C
86 †	4-73	1	E	VT	SUB N; A - B→C
87	4-74	1	64	VT	SUB A; A - B→C
88 †	4-73	1	E	VT	MPY U; A·B→C
89 †	4-73	1	E	VT	MPY L; A·B→C
8A	4-75	1	64	VT	SHIFT; A PER B→C
8B	4-73	1	E	VT	MPY S; A·B→C
8C †	4-73	1	E	VT	DIV U; A/B→C
8F †	4-73	1	E	VT	DIV S; A/B→C
90	4-76	1	E	VT	TRUNCATE A→C
91	4-76	1	E	VT	FLOOR A→C

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
92	4-76	1	E	VT	CEILING A→C
93†	4-83	1	E	VT	SIGNIFICANT SQUARE ROOT OF A→C
94	4-86	1	E	VT	ADJUST SIGNIFICANCE OF A PER B→C
95	4-86	1	E	VT	ADJUST EXPONENT OF A PER B→C
96	4-83	1	B	VT	CONTRACT 64 BIT A→32 BIT C
97	4-83	1	B	VT	ROUNDED CONTRACT 64 BIT A→32 BIT C
98	4-76	1	E	VT	TRANSMIT A→C
99	4-76	1	E	VT	ABSOLUTE A→C
9A	4-76	1	E	VT	EXPONENT OF A→C
9B	4-81	1	E	VT	PACK A, B→C
9C	4-83	1	B	VT	EXTEND 32 BIT A→64 BIT C
9D	4-82	1	E	VT	LOGICAL; A, B→C
A0†	4-93	2	E	SV	ADD U; A + B→C
A1†	4-93	2	E	SV	ADD L; A + B→C
A2†	4-93	2	E	SV	ADD N; A + B→C
A4†	4-93	2	E	SV	SUB U; A - B→C
A5†	4-93	2	E	SV	SUB L; A - B→C
A6†	4-93	2	E	SV	SUB N; A - B→C
A8†	4-97	2	E	SV	MPY U; A·B→C
A9†	4-97	2	E	SV	MPY L; A·B→C
AB†	4-97	2	E	SV	MPY S; A·B→C
AC†	4-97	2	E	SV	DIV U; A/B→C
AF†	4-97	2	E	SV	DIV S; A/B→C

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
B0	4-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) = (Z)
B1	4-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) ≠ (Z)
B2	4-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) ≥ (Z)
B3	4-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) < (Z)
B4	4-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) ≤ (Z)
B5	4-60	C	64	BR	COMPARE INTEGER, BRANCH IF (A) + (X) > (Z)
B0	4-63	C	64	BR	COMPARE FP, BRANCH IF (A) = (X)
B1	4-63	C	64	BR	COMPARE FP, BRANCH IF (A) ≠ (X)
B2	4-63	C	64	BR	COMPARE FP, BRANCH IF (A) ≥ (X)
B3	4-63	C	64	BR	COMPARE FP, BRANCH IF (A) < (X)
B4	4-63	C	64	BR	COMPARE FP, BRANCH IF (A) ≤ (X)
B5	4-63	C	64	BR	COMPARE FP, BRANCH IF (A) > (X)
B0	4-140	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) = (Z)
B1	4-140	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) ≠ (Z)
B2	4-140	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) ≥ (Z)
B3	4-140	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) < (Z)
B4	4-140	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) ≤ (Z)

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
B5	4-140	C	64	NT	COMPARE INTEGER, SET CONDITION (A) + (X) > (Z)
B0	4-142	C	64	NT	COMPARE FP, SET CONDITION (A) = (X)
B1	4-142	C	64	NT	COMPARE FP, SET CONDITION (A) ≠ (X)
B2	4-142	C	64	NT	COMPARE FP, SET CONDITION (A) ≥ (X)
B3	4-142	C	64	NT	COMPARE FP, SET CONDITION (A) < (X)
B4	4-142	C	64	NT	COMPARE FP, SET CONDITION (A) ≤ (X)
B5	4-142	C	64	NT	COMPARE FP, SET CONDITION (A) > (X)
B6	4-64	5	NA	BR	BRANCH TO IMMEDIATE ADDRESS (R) + I (48 BITS)
B7	4-114	1	E	VM	TRANSMIT LIST → INDEXED C
B8	4-110	1	E	VM	TRANSMIT REVERSE A → C
BA	4-113	1	E	VM	TRANSMIT INDEXED LIST → C
BB	4-125	2	E	NT	MASK A, B → C PER Z
BC	4-125	2	E	NT	COMPRESS A → C PER Z
BD	4-129	2	E	NT	MERGE A, B → C PER Z
BE	4-33	5	64	IN	ENTER (R) WITH I (48 BITS)
BF	4-33	5	64	IN	INCREASE (R) BY I (48 BITS)
C0	4-101	1	E	VM	SELECT EQ; A = B, ITEM COUNT TO (C)
C1	4-101	1	E	VM	SELECT NE; A ≠ B, ITEM COUNT TO (C)
C2	4-101	1	E	VM	SELECT GE; A ≥ B, ITEM COUNT TO (C)

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
C3	4-101	1	E	VM	SELECT LT; A < B, ITEM COUNT TO (C)
C4	4-143	1	E	NT	COMPARE EQ; A = B, ORDER VECTOR→Z
C5	4-143	1	E	NT	COMPARE NE; A ≠ B, ORDER VECTOR→Z
C6	4-143	1	E	NT	COMPARE GE; A ≥ B, ORDER VECTOR→Z
C7	4-143	1	E	NT	COMPARE LT; A < B, ORDER VECTOR→Z
C8	4-144	1	E	NT	SEARCH EQ; A = B, INDEX LIST→C
C9	4-144	1	E	NT	SEARCH NE; A ≠ B, INDEX LIST→C
CA	4-144	1	E	NT	SEARCH GE; A ≥ B, INDEX LIST→C
CB	4-144	1	E	NT	SEARCH LT; A < B, INDEX LIST→C
CC	4-148	3	64	NT	MASKED BINARY COMPARE: A EQ/NE (B) PER (C)
CD	4-33	5	32	IN	HALF-WORD ENTER (R) WITH I (24 BITS)
CE	4-33	5	32	IN	HALF-WORD INCREASE (R) BY I (24 BITS)
CF†	4-126	1	E	NT	ARITH. COMPRESS A→C PER B
D0	4-109	1	E	VM	AVERAGE $(A_n+B_n)/2 \rightarrow C_n$
D1	4-107	1	E	VM	ADJ. MEAN $[A_{(n+1)}+A_n]/2 \rightarrow C_n$
D4	4-109	1	E	VM	AVE. DIFF. $(A_n-B_n)/2 \rightarrow C_n$
D5	4-107	1	E	VM	DELTA $[A_{(n+1)}-A_n] \rightarrow C_n$
D8†	4-149	1	E	NT	MAX. OF A TO (C), ITEM COUNT TO (B)

TABLE 4-2. INSTRUCTION LIST BY FUNCTION CODE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Type	Instr Title
D9†	4-149	1	E	NT	MIN. OF A TO (C), ITEM COUNT TO (B)
DA	4-104	1	E	VM	SUM ($A_0+A_1+A_2...A_n$) TO (C) AND (C + 1)
DB	4-105	1	E	VM	PRODUCT ($A_0,A_1,A_2,...A_n$) TO (C)
DC	4-116	1	E	VM	VECTOR DOT PRODUCT TO (C) AND (C + 1)
DF	4-112	1	E	VM	INTERVAL A PER B → C
F0	4-119	3	1	LS	LOGICAL EXCLUSIVE OR A, B → C
F1	4-119	3	1	LS	LOGICAL AND A, B → C
F2	4-119	3	1	LS	LOGICAL INCLUSIVE OR A, B → C
F3	4-119	3	1	LS	LOGICAL STROKE A, B → C
F4	4-119	3	1	LS	LOGICAL PIERCE A, B → C
F5	4-119	3	1	LS	LOGICAL IMPLICATION A, B → C
F6	4-119	3	1	LS	LOGICAL INHIBIT A, B → C
F7	4-119	3	1	LS	LOGICAL EQUIVALENCE A, B → C
F8	4-118	3	8	ST	MOVE BYTES LEFT A → C

† These instructions have sign control capability.

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
INDEX INSTRUCTIONS (IN)				
3E	4-32	6	64	ENTER (R) WITH I (16 BITS)
3F	4-32	6	64	INCREASE (R) BY I (16 BITS)
4D	4-32	6	32	HALF-WORD ENTER (R) WITH I (16 BITS)
4E	4-32	6	32	HALF-WORD INCREASE (R) BY I (16 BITS)
CD	4-33	5	32	HALF-WORD ENTER (R) WITH I (24 BITS)
CE	4-33	5	32	HALF-WORD INCREASE (R) BY I (24 BITS)
BE	4-33	5	64	ENTER (R) WITH I (48 BITS)
BF	4-33	5	64	INCREASE (R) BY I (48 BITS)
38	4-34	A	64	TRANSMIT (R BITS 00-15) TO (T BITS 00-15)
REGISTER INSTRUCTIONS (RG)				
2C	4-34	4	64	LOGICAL EXCLUSIVE OR (R), (S), TO (T)
2D	4-34	4	64	LOGICAL AND (R), (S), TO (T)
2E	4-34	4	64	LOGICAL INCLUSIVE OR (R), (S), TO (T)
30	4-35	7	64	SHIFT (R) PER (S) TO (T)
34	4-35	4	64	SHIFT (R) PER (S) TO (T)
6D	4-36	4	64	INSERT BITS FROM (R) TO (T) PER (S)
6E	4-37	4	64	EXTRACT BITS FROM (R) TO (T) PER (S)
40/60	4-38	4	32/64	ADD U; (R) + (S) TO (T)
41/61	4-38	4	32/64	ADD L; (R) + (S) TO (T)
42/62	4-38	4	32/64	ADD N; (R) + (S) TO (T)
44/64	4-38	4	32/64	SUB U; (R) - (S) TO (T)
45/65	4-38	4	32/64	SUB L; (R) - (S) TO (T)

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
46/66	4-38	4	32/64	SUB N; (R) - (S) TO (T)
48/68	4-38	4	32/64	MPY U; (R)·(S) TO (T)
49/69	4-38	4	32/64	MPY L; (R)·(S) TO (T)
4B/6B	4-38	4	32/64	MPY S; (R)·(S) TO (T)
4C/6C	4-38	4	32/64	DIV U; (R)/(S) TO (T)
4F/6F	4-38	4	32/64	DIV S; (R)/(S) TO (T)
63	4-39	4	64	ADD ADDRESS (R) + (S) TO (T)
67	4-39	4	64	SUB ADDRESS (R) - (S) TO (T)
58/78	4-39	A	32/64	TRANSMIT (R) TO (T)
59/79	4-39	A	32/64	ABSOLUTE (R) TO (T)
51/71	4-39	A	32/64	FLOOR (R) TO (T)
52/72	4-39	A	32/64	CEILING (R) TO (T)
5A/7A	4-39	A	32/64	EXPONENT OF (R) TO (T)
50/70	4-39	A	32/64	TRUNCATE (R) TO (T)
5B/7B	4-42	4	32/64	PACK (R), (S) TO (T)
5C	4-42	A	B	EXTEND 32 BIT (R) TO 64 BIT (T)
5D	4-42	A	B	INDEX EXTEND 32 BIT (R) TO 64 BIT (T)
76	4-42	A	B	CONTRACT 64 BIT (R) TO 32 BIT (T)
77	4-42	A	B	ROUNDED CONTRACT 64 BIT (R) TO 32 BIT (T)
7C	4-42	A	64	LENGTH OF (R) TO (T)
53/73	4-42	A	32/64	SIGNIFICANT SQUARE ROOT OF (R) TO (T)
10	4-42	A	64	CONVERT BCD TO BINARY, FIXED LENGTH
11	4-42	A	64	CONVERT BINARY TO BCD, FIXED LENGTH

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
54/74	4-48	4	32/64	ADJUST SIGNIFICANCE OF (R) PER (S) TO (T)
55/75	4-48	4	32/64	ADJUST EXPONENT OF (R) PER (S) TO (T)
2A	4-49	6	64	ENTER LENGTH OF (R) WITH I (16 BITS)
2B	4-49	4	64	ADD TO LENGTH FIELD
BRANCH INSTRUCTIONS (BR)				
20/24	4-51	8	32/64	BRANCH IF(R) = (S)(32/64 BIT FP)
21/25	4-51	8	32/64	BRANCH IF(R) $\neq$ (S)(32/64 BIT FP)
22/26	4-51	8	32/64	BRANCH IF(R) $\geq$ (S)(32/64 BIT FP)
23/27	4-51	8	32/64	BRANCH IF(R) < (S)(32/64 BIT FP)
2F	4-51	9	1	REGISTER BIT BRANCH AND ALTER
33	4-53	B	1	DATA FLAG REGISTER BIT BRANCH AND ALTER
3B	4-55	A	64	DATA FLAG REGISTER LOAD/STORE
32	4-55	9	1	BIT BRANCH AND ALTER
36	4-58	7	64	BRANCH AND SET (R) TO NEXT INSTRUCTION
31	4-58	7	64	INCREASE (R) AND BRANCH IF (R) $\neq$ 0
35	4-58	7	64	DECREASE (R) AND BRANCH IF (R) $\neq$ 0
09	4-57	4	64	EXIT FORCE
B0	4-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) = (Z)
B1	4-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) $\neq$ (Z)
B2	4-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (Z) $\geq$ (Z)
B3	4-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) < (Z)

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
B4	4-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) $\leq$ (Z)
B5	4-60	C	64	COMPARE INTEGER, BRANCH IF (A) + (X) > (Z)
B0	4-63	C	64	COMPARE FP, BRANCH IF (A) = (X)
B1	4-63	C	64	COMPARE FP, BRANCH IF (A) $\neq$ (X)
B2	4-63	C	64	COMPARE FP, BRANCH IF (A) $\geq$ (X)
B3	4-63	C	64	COMPARE FP, BRANCH IF (A) < (X)
B4	4-63	C	64	COMPARE FP, BRANCH IF (A) $\leq$ (X)
B5	4-63	C	64	COMPARE FP, BRANCH IF (A) > (X)
B6	4-63	5	NA	BRANCH TO IMMEDIATE ADDRESS (R) + I (48 BITS)
VECTOR INSTRUCTIONS (VT)				
80†	4-73	1	E	ADD U; A + B→C
81†	4-73	1	E	ADD L; A + B→C
82†	4-73	1	E	ADD N; A + B→C
84†	4-73	1	E	SUB U; A - B→C
85†	4-73	1	E	SUB L; A - B→C
86†	4-73	1	E	SUB N; A - B→C
88†	4-73	1	E	MPY U; A·B→C
89†	4-73	1	E	MPY L; A·B→C
8B†	4-73	1	E	MPY S; A·B→C
8C†	4-73	1	E	DIV U; A/B→C
8F†	4-73	1	E	DIV S; A/B→C
83	4-74	1	64	ADD A; A + B→C
87	4-74	1	64	SUB A; A - B→C
8A	4-75	1	64	SHIFT; A PER B→C

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
98	4-75	1	E	TRANSMIT A→C
99	4-75	1	E	ABSOLUTE A→C
91	4-75	1	E	FLOOR A→C
92	4-75	1	E	CEILING A→C
9A	4-75	1	E	EXPONENT OF A→C
90	4-75	1	E	TRUNCATE A→C
9B	4-80	1	E	PACK A, B→C
9D	4-80	1	E	LOGICAL; A, B→C
9C	4-81	1	B	EXTEND 32 BIT A→64 BIT C
96	4-81	1	B	CONTRACT 64 BIT A→32 BIT C
97	4-81	1	B	ROUNDED CONTRACT 64 BIT A→32 BIT C
93 †	4-81	1	E	SIGNIFICANT SQUARE ROOT OF A→C
94	4-85	1	E	ADJUST SIGNIFICANT OF A PER B→C
95	4-85	1	E	ADJUST EXPONENT OF A PER B→C
SPARSE VECTOR INSTRUCTIONS (SV)				
A0 †	4-93	2	E	ADD U; A + B→C
A1 †	4-93	2	E	ADD L; A + B→C
A2 †	4-93	2	E	ADD N; A + B→C
A4 †	4-93	2	E	SUB U; A - B→C
A5 †	4-93	2	E	SUB L; A - B→C
A6 †	4-93	2	E	SUB N; A - B→C
A8 †	4-97	2	E	MPY U; A·B→C
A9 †	4-97	2	E	MPY L; A·B→C
AB †	4-97	2	E	MPY S; A·B→C
AC †	4-97	2	E	DIV U; A / B→C
AF †	4-97	2	E	DIV S; A / B→C

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
VECTOR MACRO INSTRUCTIONS (VM)				
C0	4-101	1	E	SELECT EQ; A = B, ITEM COUNT TO (C)
C1	4-101	1	E	SELECT NE; A $\neq$ B, ITEM COUNT TO (C)
C2	4-101	1	E	SELECT GE; A $\geq$ B, ITEM COUNT TO (C)
C3	4-101	1	E	SELECT LT; A < B, ITEM COUNT TO (C)
DA	4-104	1	E	SUM ($A_0 + A_1 + A_2 + \ldots A_n$) TO (C) AND (C + 1)
DB	4-105	1	E	PRODUCT ($A_0, A_1, A_2, \ldots A_n$) TO (C)
D5	4-107	1	E	DELTA $[A_{(n+1)}-A_n] \rightarrow C_n$
D1	4-107	1	E	ADJ. MEAN $[A_{(n+1)}+A_n]/2 \rightarrow C_n$
D0	4-109	1	E	AVERAGE $(A_n+B_n)/2 \rightarrow C_n$
D4	4-109	1	E	AVE. DIFF. $(A_n-B_n)/2 \rightarrow C_n$
B8	4-110	1	E	TRANSMIT REVERSE A $\rightarrow$ C
DF	4-112	1	E	INTERVAL A PER B $\rightarrow$ C
BA	4-113	1	E	TRANSMIT INDEXED LIST $\rightarrow$ C
B7	4-114	1	E	TRANSMIT LIST $\rightarrow$ INDEXED C
DC	4-116	1	E	VECTOR DOT PRODUCT TO (C) AND (C + 1)
STRING INSTRUCTION (ST)				
F8	4-118	3	8	MOVE BYTES LEFT; A $\rightarrow$ C

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
LOGICAL STRING INSTRUCTIONS (LS)				
F0	4-119	3	1	LOGICAL EXCLUSIVE OR A, B→C
F1	4-119	3	1	LOGICAL AND A, B→C
F2	4-119	3	1	LOGICAL INCLUSIVE OR A, B→C
F3	4-119	3	1	LOGICAL STROKE A, B→C
F4	4-119	3	1	LOGICAL PIERCE A, B→C
F5	4-119	3	1	LOGICAL IMPLICATION A, B→C
F6	4-119	3	1	LOGICAL INHIBIT A, B→C
F7	4-119	3	1	LOGICAL EQUIVALENCE A, B→C
NONTYPICAL INSTRUCTIONS (NT)				
3D	4-122	4	64	INDEX MULTIPLY (R)·(S) TO (T)
3C	4-122	4	32	HALF-WORD INDEX MULTIPLY (R)·(S) TO (T)
5E/7E	4-122	7	32	LOAD (T) PER (S), (R)
5F/7F	4-122	7	32	STORE (T) PER (S), (R)
12/13	4-122	7	64	LOAD/STORE BYTE (T) PER (S), (R)
37	4-123	A	64	TRANSMIT JOB INTERVAL TIMER TO (T)
7D	4-123	7	64	SWAP S→T AND R→S

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
39	4-124	A	64	TRANSMIT REAL-TIME CLOCK TO (T)
3A	4-124	A	64	TRANSMIT (R) TO JOB INTERVAL TIMER
BB	4-125	2	E	MASK A, B → C PER Z
BC	4-125	2	E	COMPRESS A → C PER Z
CF	4-126	1	E	ARITH. COMPRESS A → C PER B
BD	4-129	2	E	MERGE A, B → C PER Z
14	4-131	7	1	BIT COMPRESS
15	4-133	7	1	BIT MERGE
16	4-133	7	1	BIT MASK
56	4-137		NA	SELECT LINK
B0	4-140	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) = (Z)
B1	4-140	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) ≠ (Z)
B2	4-140	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) ≥ (Z)
B3	4-140	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) < (Z)
B4	4-140	C	64	COMPARE INTEGER, SET CONDITION (A) + (X) ≤ (Z)
B5	4-140	C	64	COMPARE INTEGER, SET CONDITION (A) > (X)
B0	4-142	C	64	COMPARE FP, SET CONDITION (A) = (X)
B1	4-142	C	64	COMPARE FP, SET CONDITION (A) ≠ (X)
B2	4-142	C	64	COMPARE FP, SET CONDITION (A) ≥ (X)
B3	4-142	C	64	COMPARE FP, SET CONDITION (A) < (X)

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
B4	4-142	C	64	COMPARE FP, SET CONDITION (A) $\leq$ (X)
B5	4-142	C	64	COMPARE FP, SET CONDITION (A) > (X)
C4	4-143	1	E	COMPARE EQ; A = B, ORDER VECTOR → Z
C5	4-143	1	E	COMPARE NE; A ≠ B, ORDER VECTOR → Z
C6	4-143	1	E	COMPARE GE; A > B, ORDER VECTOR → Z
C7	4-143	1	E	COMPARE LT; A < B, ORDER VECTOR → Z
C8	4-144	1	E	SEARCH EQ; A = B, INDEX LIST → C
C9	4-144	1	E	SEARCH NE; A ≠ B, INDEX LIST → C
CA	4-144	1	E	SEARCH GE; A > B, INDEX LIST → C
CB	4-144	1	E	SEARCH LT; A < B, INDEX LIST → C
CC	4-148	3	64	MASKED BINARY COMPARE: A EQ/NE (B) PER (C)
D8	4-149	1	E	MAX. OF A TO (C) ITEM COUNT TO (B)
D9	4-149	1	E	MIN. OF A TO (C) ITEM COUNT TO (B)
28	4-150	7	8	SCAN EQUAL
1C	4-150	7	1	FORM REPEATED BIT MASK WITH LEADING ZEROS
1D	4-150	7	1	FORM REPEATED BIT MASK WITH LEADING ONES
1E	4-151	7	1	COUNT LEADING EQUALS
1F	4-153	7	1	COUNT ONES IN FIELD R. COUNT TO (T)

TABLE 4-3. INSTRUCTION LIST BY INSTRUCTION TYPE (Contd)

Instr Code	Page No.	Format Type	No. of Bits in Operand	Instr Title
NONTYPICAL INSTRUCTIONS (NT)				
03	4-153	4	64	KEYPOINT - MAINTENANCE
04	4-153	4	64	BREAKPOINT - MAINTENANCE
05	4-154	4	64	VOID STACK AND BRANCH
MONITOR INSTRUCTIONS (MN)				
00	4-155	4	NA	IDLE
06	4-155	7	NA	FAULT TEST - MAINTENANCE
08	4-156	4	64	INPUT/OUPTUT PER R
0C	4-157	4	64	STORE ASSOCIATIVE REGISTERS
0D	4-157	4	64	LOAD ASSOCIATIVE REGISTERS
0E	4-157	4	64	TRANSLATE EXTERNAL INTERRUPT
0F	4-157	4	64	LOAD KEYS FROM (R), TRANSLATE ADDRESS (S) TO (T)
0A	4-157	4	64	TRANSMIT (R) TO MONITOR INTERVAL TIMER

† These instructions have sign control capability.

INSTRUCTION DESCRIPTIONS

The instruction descriptions are grouped in the following order.

- Index Instructions
- Register Instructions
- Branch Instructions
- Vector Instructions
- Sparse Vector Instructions
- Vector Macro Instructions
- String Instructions
- Logical String Instructions
- Nontypical Instructions
- Monitor Instructions

The description of each of the general types of instructions contains the instruction formats, operating parameters, and instruction termination conditions that are applicable to the instruction. The individual instructions within a general type are grouped according to the specific functions they perform within that group. Instructions that differ slightly in the functions they perform have a common description.

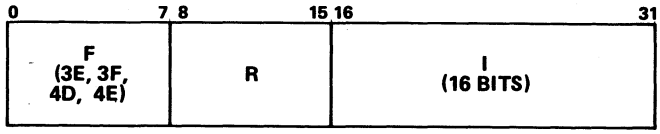
Each description begins with a listing of the function code (hexadecimal) and title of the instruction. This listing is followed by the instruction format. The formats specifically apply to the listed instructions and show the variations from the general format types shown in the beginning of this section.

Where applicable, the instruction descriptions include examples. These examples show a simplified illustration of the instruction operation using arbitrarily assumed operands, register contents, indexes, and so forth. The assumed operands and operating parameters are selected mainly to illustrate the instruction operation and are not necessarily typical operating values. The numbers used in the examples are in hexadecimal notation unless otherwise noted.

INDEX INSTRUCTIONS

The index instructions manipulate 16 24- or 48-bit operands in the designated operational registers. These instructions are used primarily in performing numerical calculations on field lengths and addresses.

3E Enter (R) with I (16 Bits)
3F Increase (R) by I (16 Bits)
4D Half-Word Enter (R) with I (16 Bits)
4E Half-Word Increase (R) by I (16 Bits)



3E Enter (R) with I (16 Bits)

This instruction enters the 16-bit operand I into the rightmost 48 bits of the 64-bit register designated by R. The sign bit of the immediate 16-bit operand is extended through bit 16 of the destination register R. Register R is cleared before the transfer of I.

3F Increase (R) with I (16 Bits)

This instruction replaces the rightmost 48 bits of the 64-bit register designated by R with the sum of these bits and the 16-bit operand I. The leftmost 16 bits of register R are unaltered. The sign bit of the immediate 16-bit operand is extended through bit 16 in the addition. Arithmetic overflow is ignored if it occurs.

4D Half-Word Enter (R) with I (16 Bits)

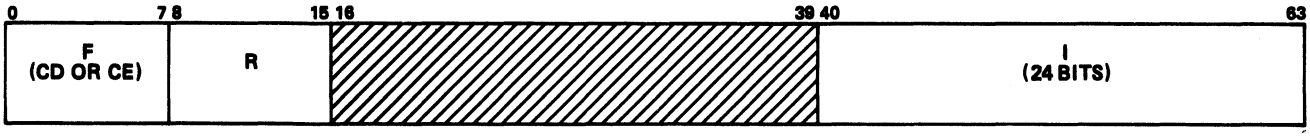
This instruction enters the 16-bit operand I into the rightmost 24 bits of the 32-bit register designated by R. The sign of the immediate 16-bit operand is extended through bit 8 of the destination register R. Register R is cleared before the transfer of I.

4E Half-Word Increase (R) by I (16 Bits)

This instruction replaces the rightmost 24 bits of the 32-bit register designated by R with the sum of these bits and the 16-bit operand I. The leftmost 8 bits of register R are unaltered. The sign of the operand is extended through bit 8 for the addition. Arithmetic overflow is ignored if it occurs.

CD Half-Word Enter (R) with I (24 Bits)

CE Half-Word Increase (R) with I (24 Bits)



CD Half-Word Enter (R) with I (24 Bits)

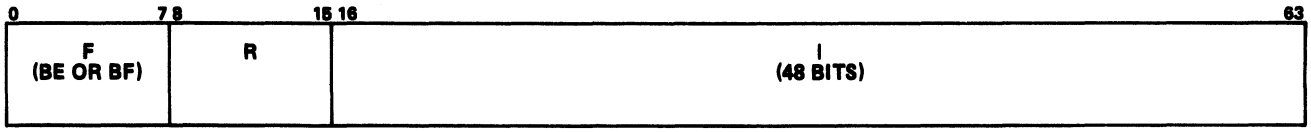
This instruction clears the 32-bit register designated by R and enters the operand I, contained in the rightmost 24 bits of this instruction, into the rightmost 24 bits of register R.

CE Half-Word Increase (R) with I (24 Bits)

This instruction replaces the rightmost 24 bits of the 32-bit register designated by R with the sum of these bits and operand I contained in the rightmost 24 bits of this instruction. The leftmost 8 bits of register R are unaltered. Arithmetic overflow is ignored if it occurs.

BE Enter (R) with I (48 Bits)

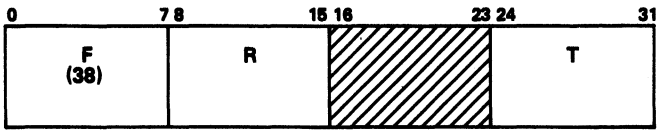
BF Increase (R) with I (48 Bits)



The BE instruction enters the 48-bit operand I into the rightmost 48 bits of the R register. Register R is cleared before the transfer of I.

The BF instruction replaces the rightmost 48 bits of the R register with the sum of these bits and the 48-bit operand I. The leftmost 16 bits of R are unaltered. Arithmetic overflow is ignored.

38 Transmit (R Bits 00-15) to (T Bits 00-15)



This instruction replaces the leftmost 16 bits of register T with the leftmost 16 bits of register R. The remaining bits of register T are unaltered.

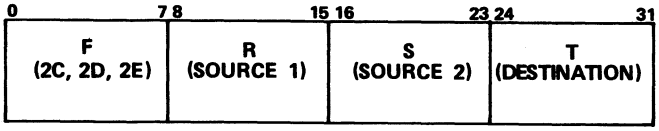
REGISTER INSTRUCTIONS

The source and result operands of register instructions are contained in specified registers in the register file. The 8-bit R, S, and T designators, contained in the instructions, denote the numbers of the registers to be used in the operation. For example, if a 64-bit, floating-point, add upper instruction is executed (instruction code 60) with R = 02, S = 03, and T = 7F, the content of register 02 is added to the contents of register 03 (floating-point format), and the upper result is stored in destination 7F.

A register may contain one or both source operands as well as the result. Register 00 provides a special case. If this register is designated as containing the source operand, the instruction uses machine zero as the source operand (8X 000000 for 32-bit operands and 8XXX 000000 000000 for 64-bit operands where X represents any hexadecimal digit). If the instruction specifies 00 as the destination register, no result is stored. However, the instruction sets the corresponding data flags if applicable.

Unless the individual instruction description states differently, register-to-register operations do not change the content of the source registers. These operations clear the destination register before the result is transferred into it.

2C Logical Exclusive OR (R), (S) to (T)
2D Logical AND (R), (S), to (T)
2E Logical Inclusive OR (R), (S) to (T)

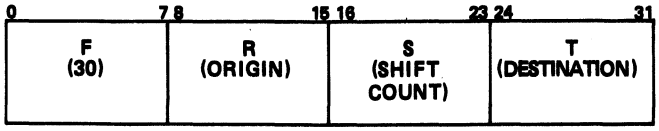


These instructions perform the following logical functions. The function occurs bit by bit on the 64-bit operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

R	S	Exclusive OR R-S	AND R•S	Inclusive OR R+S
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	1

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

30 Shift (R) Per S to (T)



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

If the S designator is in the range from 0 through $3F_{16}$ (0 through 63_{10}), the operand from register R shifts left end-around the number of specified places and then stores in register T.

If the S designator is in the range from FF_{16} through $C1_{16}$ (-1 through -63_{10}), the operand from register R shifts right with sign extension and then stores into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the S designator.

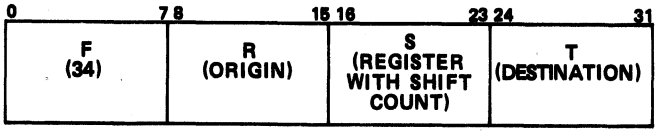
If, for example, S is equal to FE_{16}, the operand from register R shifts right two places.

If the S designator is greater than $3F_{16}$ or less than $C1_{16}$, the results of this instruction are undefined.

If the R designator is equal to zero, register zero provides machine zero.

This instruction does not test for machine zero, indefinite or does not set any data flags.

34 Shift (R) Per (S) to (T)



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

If the rightmost byte of register S is in the range from 0 through $3F_{16}$ (0 through 63_{10}), the operand from register R shifts left end-around the number of specified places and then stores into register T.

If the rightmost byte of register S is in the range from FF_{16} through $C1_{16}$ (-1 through -63_{10}), the operand from register R shifts right with sign extension and then stores into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the rightmost byte of register S.

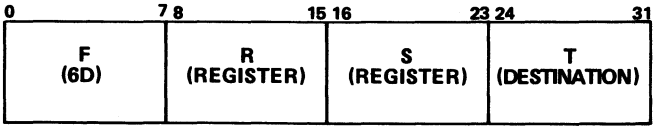
If the rightmost byte of register S is greater than $3F_{16}$ or less than $C1_{16}$, the results of this instruction are undefined.

The leftmost seven bytes of register S are ignored.

If the R designator is equal to zero, register zero provides machine zero.

This instruction does not cause a test for machine zero, indefinite or does not set any data flags.

6D Insert Bits from (R) to (T) Per (S)



This instruction inserts a number of rightmost bits (m) from the register designated R to the register designated T (figure 4-3). In the register designated S, bits 10 through 15 specify the number of bits (m) to be inserted, and bits 58 through 63 specify the location (n) in register T for the leftmost bit of the inserted bits. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zeros.

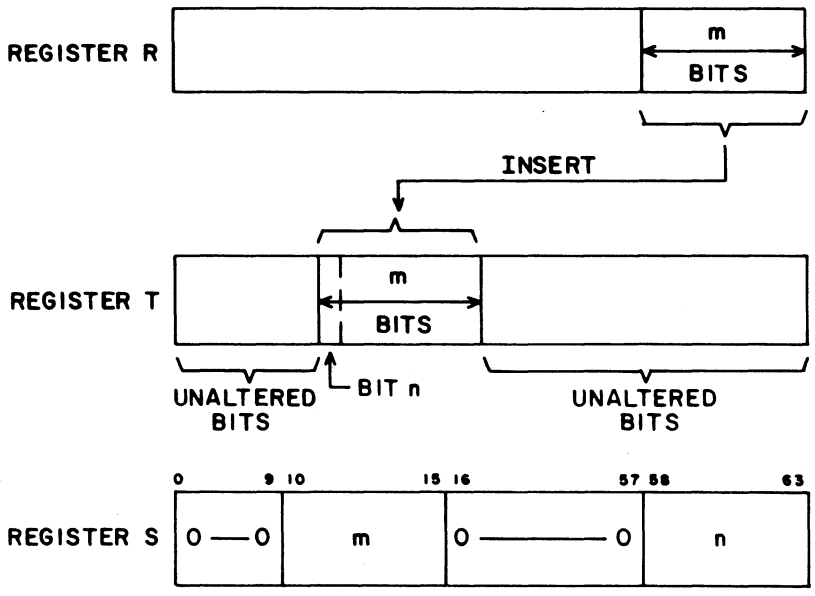
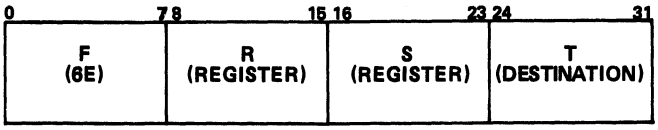


Figure 4-3. Example of Register Content for an Insert Bits from (R) to (T) Per (S) Instruction

If the R designator is equal to zero, register zero provides machine zero. If m plus n is greater than 64_{10}, or if m is equal to zero, the results of this instruction are undefined. The maximum number of bits specified by m is 63_{10}.

6E Extract Bits from (R) to (T) Per (S)



This instruction extracts a number of bits (m) from the register designated R and stores them in the rightmost part of the register designated T (figure 4-4). Register T is cleared before receiving the extracted bits. In the register designated S, bits 10 through 15 contain the number of bits (m) to be extracted and bits 58 through 63 specify the leftmost bit number of the extracted bits in register R. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zeros.

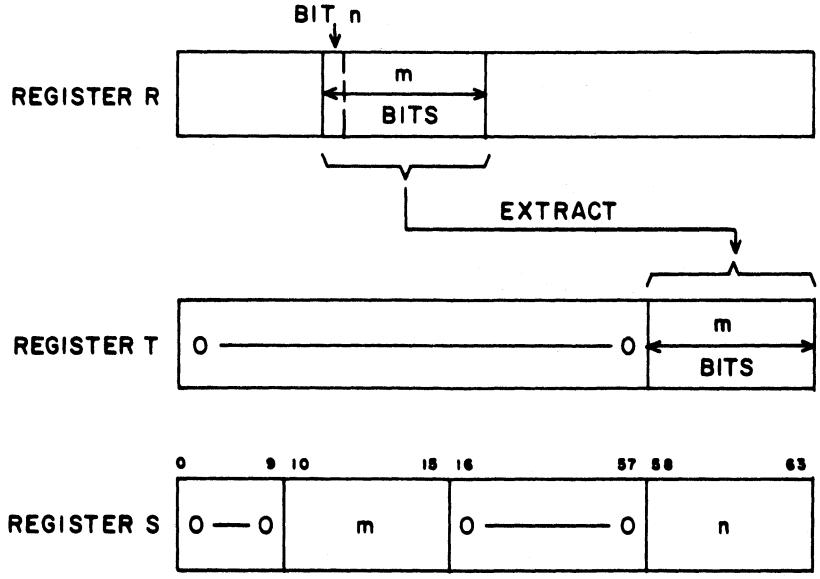


Figure 4-4. Example of Register Contents for an Extract Bits from (R) to (T) Per (S) Instruction

If the R designator is equal to zero, register zero provides machine zero. If m plus n is greater than 64_{10}, or if m is equal to zero, the results of this instruction are undefined. The maximum number of bits specified by m is 63_{10}.

40/60 Add U; (R) + (S) to (T)

41/61 Add L; (R) + (S) to (T)

42/62 Add N; (R) + (S) to (T)

44/64 Sub U; (R) - (S) to ( )

45/65 Sub L; (R) - (S) to (to (T)

46/66 Sub N; (R) - (S) to (T)

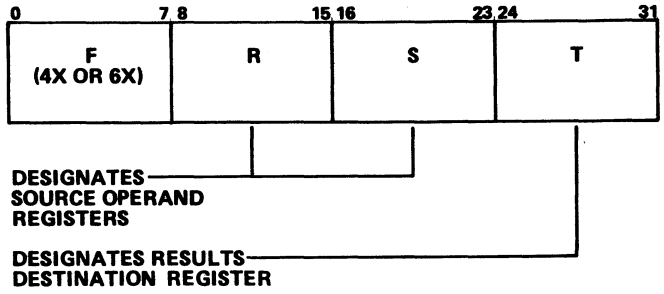
48/68 Mpy U; (R) • (S) to (T)

49/69 Mpy L; (R) • (S) to (T)

4B/6B Mpy S; (R) • (S) to (T)

4C/6C Div U; (R) / (S) to (T)

4F/6F Div S; (R) / (S) to (T)



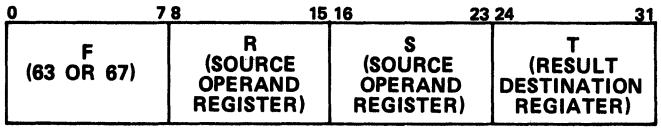
These instructions perform the indicated floating-point arithmetic operation on the 32-bit (4X function codes) or 64-bit (6X function codes) operands contained in the registers designated by R and S. Appendix B describes the floating-point operations and operand formats. This appendix also describes how certain instructions are order-dependent and will result in unexpected answers unless the execution order is known. An example is shown in the appendix under Order-Dependent Result Considerations. The arithmetic operation is the same for the 32-bit or 64-bit operands with adjustment for bit length of the result. The instruction, in each case, stores the arithmetic result in destination register T.

Designator U signifies that the upper result is stored, L signifies that the lower result is stored, N signifies that the normalized upper result is stored, and S signifies the significant result is stored. Appendix B of this manual defines the U, L, N, and S results.

Data flag bits 41 (floating-point divide fault), 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result) are set by the applicable instructions if the necessary operating and result conditions are present.

63 Add Address (R) + (S) to (T)

67 Sub Address (R) - (S) to (T)



These instructions add/subtract bits 16 through 63 in register S to/from bits 16 through 63 in register R. The instructions then store the result in corresponding bits of register T. The instructions operate on bits 16 through 63 as 48-bit, positive, unsigned integers. Arithmetic overflow is ignored if it occurs. The instructions transmit bits 0 through 15 of register R to corresponding bit positions of register T without modification.

58/78 Transmit (R) to (T)

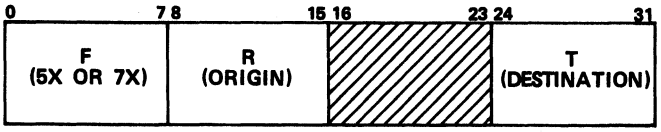
59/79 Absolute (R) to (T)

51/71 Floor (R) to (T)

52/72 Ceiling (R) to (T)

5A/7A Exponent of (R)to (T)

50/70 Truncate (R) to (T)



58/78 Transmit (R) to (T)

This instruction transmits the 32-bit (58) or 64-bit (78) operand in the register designated by R to the register designated by T.

59/79 Absolute (R) to (T)

This instruction transmits the absolute value of the 32-bit (59) or 64-bit (79) floating-point operand in register R to register T. If the coefficient of the initial operand is negative, the operand is complemented and is transmitted to register T. If the initial coefficient is positive, it is sent to register T as it is. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

51/71 Floor (R) to (T)

This instruction transmits the closest integer less than or equal to the 32-bit (51) or 64-bit (71) floating-point operand in register R to register T. This integer (T) is expressed by an unnormalized 32-bit or 64-bit floating-point number with a positive exponent.

If the exponent of the source operand is positive (greater than or equal to zero), the operand is transmitted directly to register T. If the exponent of the source operand is negative, the machine right-shifts the coefficient end-off and increases the exponent by one for each shift. Sign bits are extended on the left during the shift. When the exponent becomes zero, the shifting stops and the machine transmits the shifted coefficient and zero exponent to register T. If machine zero is used as the source operand, 32/64 zeros are transmitted to register T.

The applicable data flag bit is 46 (indefinite result).

52/72 Ceiling (R) to (T)

This instruction transmits the closest integer greater than or equal to the 32-bit (64-bit for 72 function code) operand in origin register R to destination register T. This integer is represented as an unnormalized 32-bit (64-bit) floating-point number with a positive exponent.

If the source operand exponent is positive (greater than or equal to zero), the instruction transmits the source operand directly to register T.

If the source operand exponent is negative, the machine right-shifts the two's complement of the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. The shift operation extends the sign. The instruction then recomplements the shifted coefficient and transmits it with zero exponent to register T. Figure 4-5 shows the results of a ceiling (R) to (T), 52/72, instruction with a source operand having a negative exponent. In this example, a shift of four was necessary to reduce the exponent to zero. The example shows the complement of the shifted coefficient with zero exponent in register T.

If machine zero is used as the source operand, the machine transmits 32/64 zeros as a result. The applicable data flag bit is 46 (indefinite result).

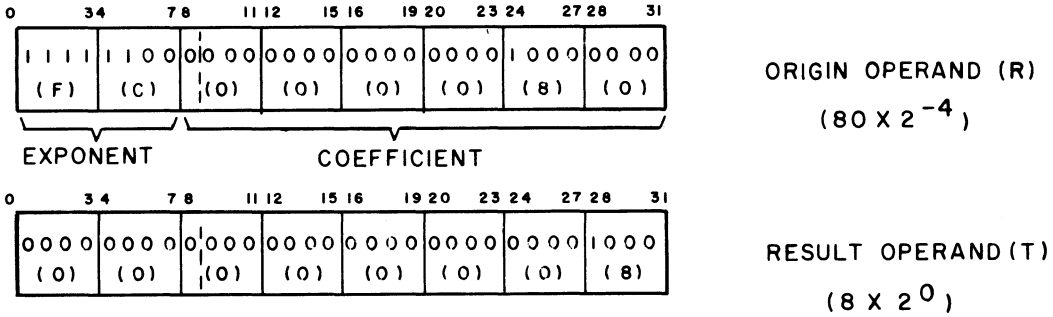


Figure 4-5. Example of Register Content for a Ceiling (R) to (T) Instruction

5A/7A Exponent of (R) to (T)

This instruction transmits the exponent in the leftmost 8 bits (16 bits for 64-bit operands) of register R to the rightmost 8 bits (16 bits for 64-bit operands) of register T. The instruction extends the sign of the exponent through bit 8 (16 bits for 64-bit operands) of register T. The exponent portion (leftmost 8 or 16 bits) of register T is cleared.

50/70 Truncate (R) to (T)

This instruction transmits the closest integer the magnitude of which is less than or equal to the 32-bit (64-bit for 70 function code) operand in origin register specified by R to destination register T. This integer is represented by an unnormalized 32-bit (64-bit) floating-point number with a positive exponent.

If the origin operand exponent is positive (greater than or equal to zero), the instruction transmits the origin operand directly to register T.

If the origin operand exponent is negative, the machine right-shifts the magnitude of the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. The operation extends zeros on the left during the shift. If the coefficient of the origin operand was positive, the shifted coefficient with zero exponent is transmitted to the destination register. If the coefficient of the origin operand was negative, the two's complement of the shifted coefficient and zero exponent is transmitted to the destination register. If machine zero is used as the origin operand, 32/64 zeros are transmitted as a result.

Figure 4-6 shows the results of a truncate (R) to (T), 50/70, instruction with an origin operand having a negative exponent and positive coefficient. A right shift of eight is required to reduce the negative exponent to zero.

The applicable data flag bit is 46 (indefinite result).

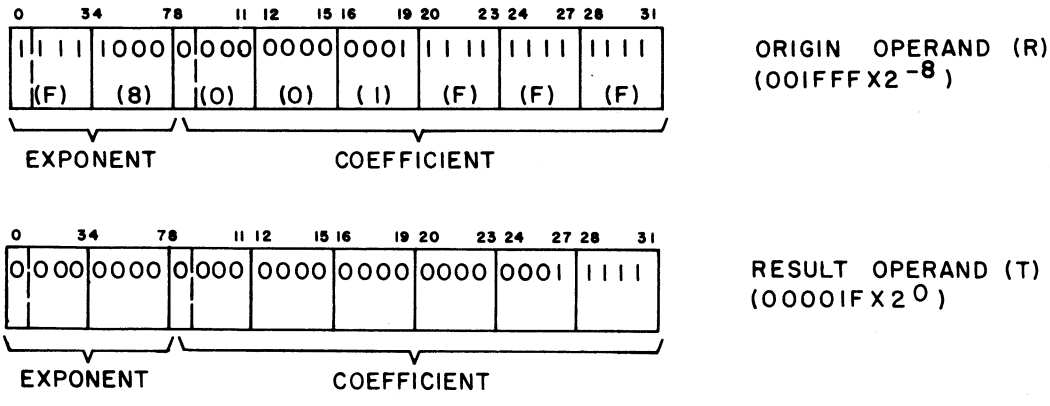
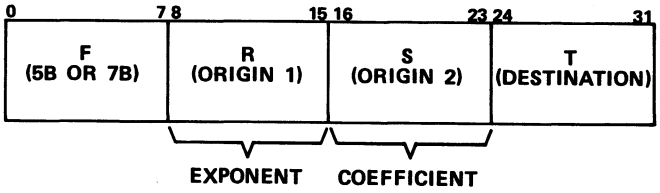


Figure 4-6. Example of Register Content for a Truncate (R) to (T) Instruction

5B/7B Pack (R), (S) to (T)



This instruction transmits a 32-bit (64-bit for the 7B function code) floating-point number to the destination register T. The instruction transmits the exponent of the number from the rightmost 8 bits (16 bits for 7B) of register R and the coefficient from the rightmost 24 bits (48 bits for 7B) of register S.

5C Extend 32 Bit (R) to 64 Bit (T)

5D Index Extend 32 Bit (R) to 64 Bit (T)

76 Contract 64 Bit (R) to 32 Bit (T)

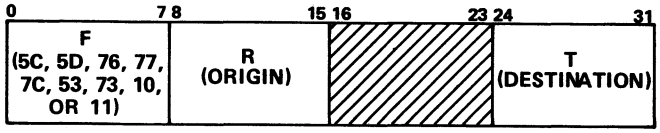
77 Rounded Contract 64 Bit (R) to 32 Bit (T)

7C Length of (R) to (T)

53/73 Significant Square Root of (R) to (T)

10 Convert BCD to Binary, Fixed Length

11 Convert Binary to BCD, Fixed Length



5C Extend 32 Bit (R) to 64 Bit (T)

This instruction extends the 32-bit floating point number from register R into a 64-bit floating-point number and transmits the result to 64-bit register T (figure 4-7). The value of the resulting exponent is 24_{10} less than the exponent of the origin operand. The result coefficient results from the transmission of the origin coefficient to bits 16 through 39 of register T. The instruction clears the rightmost 24 bits of the destination register.

If the contents of register R is indefinite, the result in register T is also indefinite and data flag bit 46 (indefinite result) is set. If the contents of register R is machine zero, register T contains machine zero, and data flag bit 43 (result machine zero) is set.

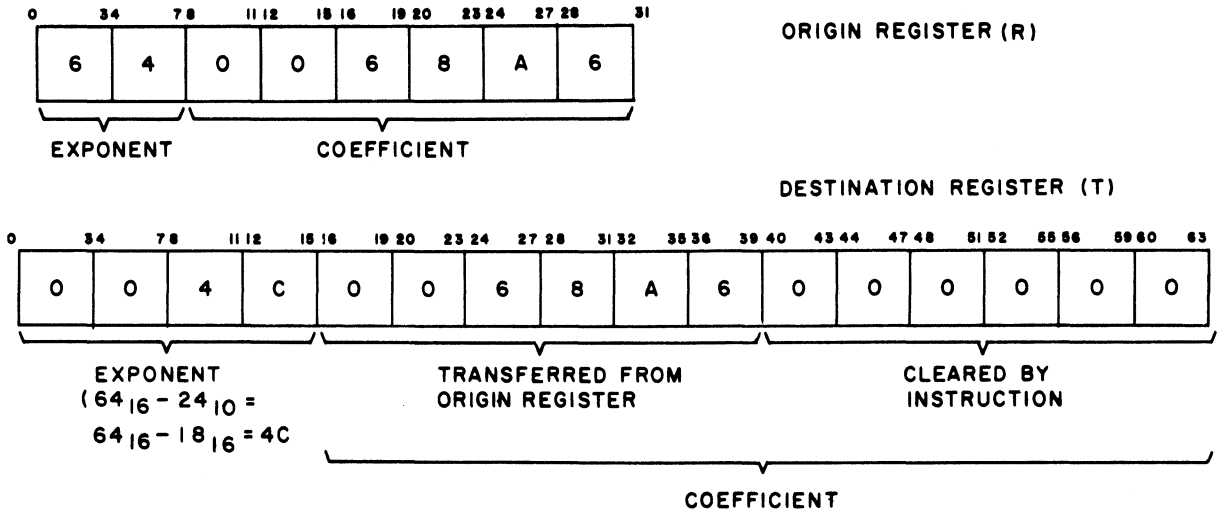


Figure 4-7. Example of Register Content for an Extend 32-Bit (R) to 64-Bit (T) Instruction

5D Index Extend 32 Bit (R) to 64 Bit (T)

This instruction extends the 32-bit floating-point number from register R into a 64-bit floating-point number and transmits the result to 64-bit register T. The value of the resulting 16-bit exponent is the same as the origin operand's exponent with the sign bit extended through bit 0 of the result exponent.

The result coefficient results from the transmission of the rightmost 24 bits of the origin register into bits 40 through 63 of the destination register. Bits 16 through 39 of the destination register are set to the sign of the origin coefficient.

If the contents of register R is indefinite, the result in register T is also indefinite and data flag bit 46 (indefinite result) is set. If the contents of register R is machine zero, register T contains machine zero and data flag bit 43 (result machine zero) is set.

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

This instruction (figure 4-8) contracts the 64-bit floating-point number from register R into a 32-bit floating-point number. The instruction then transmits the result to a 32-bit register designated by T. The resulting 8-bit exponent represents the sum of the least-significant 8 bits of the origin exponent and 24_{10}. If the result exponent cannot be contained in 8 bits, exponent overflow or underflow is detected.

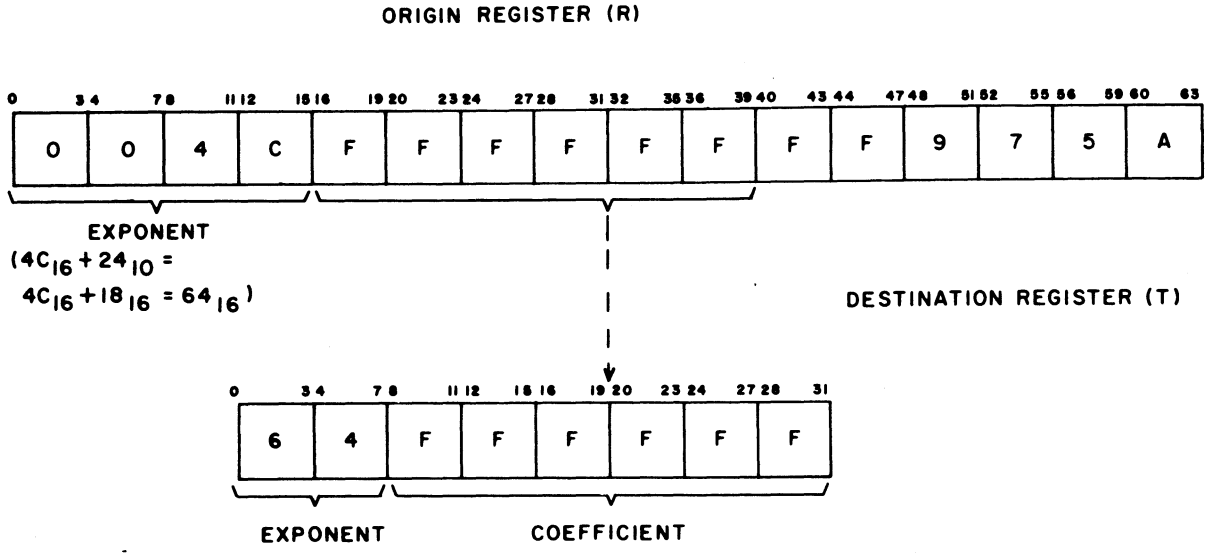


Figure 4-8. Example of Register Content for a Contract 64 Bit (R) to 32 Bit (T) Instruction

The following input exponent conditions are listed with the corresponding results of the 76 instruction execution.

Input Exponent	Result
7FFF . . 7000	Result indefinite Indefinite data flag bit 46 (indefinite result) is set.
6FFF . . 0058	Result indefinite Data flag bits 42 (exponent overflow) and 46 (indefinite result) are set.
0057 . . FF78	Result exponent is 24_{10} larger than the input exponent. The leftmost 24 bits of the input coefficient are transferred.
FF77 . . 8000	Result is machine zero. Data flag bit 43 (result machine zero) is set.

Bits 16 through 39 of the origin are transmitted directly to the rightmost 24 bits of register T as the result coefficient. This operation contracts to minus one all source operands having a negative coefficient with an absolute value of less than or equal to 2^{24} and to zero source operands having positive coefficients with an absolute value of less than 2^{24}.

77 Rounded Contract 64-Bit (R) to 32-Bit (T)

This instruction performs a rounded contract operation on the 64-bit, floating-point operand in origin register R and transmits the 32-bit floating-point result to destination register T (figure 4-9). The resulting 8-bit exponent represents the sum of the least-significant 8 bits of the origin exponent and 24_{10}. If the result exponent cannot be contained in 8 bits, exponent overflow or underflow is detected. The instruction then adds a +1 to bit position 40 of the origin operand and coefficient. If overflow occurs, the instruction increases the exponent by one and right-shifts the coefficient one place. The leftmost 24 bits of the shifted result coefficient are transmitted to the corresponding bits of the destination register. The 8-bit exponent of each nonend case result element is 24_{10} (25_{10} if overflow occurred) greater than the exponent of the corresponding source element.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

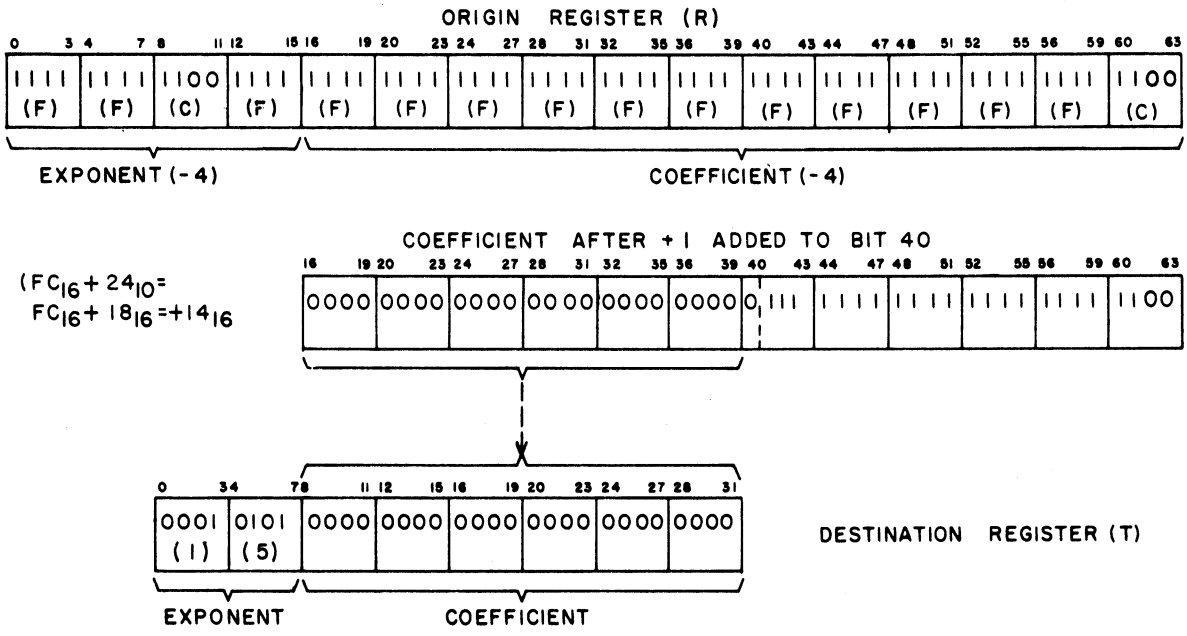


Figure 4-9. Example of Register Content for a Rounded Contract 64 Bit (R) to 32 Bit (T) Instruction

7C Length of (R) to (T)

This instruction transmits the leftmost 16 bits of origin register R to the rightmost 16-bit positions of destination register T. The leftmost 48 bits of register T are cleared.

53/73 Significant Square Root of (R) to (T)

This instruction transmits the square root† of a 32-bit (53 function code) or 64-bit (73 function code) operand in register R to register T. The result contains the same number of significant bits as the source operand. Applicable data flag bits are 45 (square root result imaginary), 46 (indefinite result), and 43 (result machine zero).

10 Convert BCD to Binary, Fixed Length

This instruction converts the packed BCD number in register R to a signed (two's complement) binary number and transfers the result to the rightmost 48 bits of register T. Figure 4-10 shows an example of the register contents following a convert BCD to binary, fixed length instruction. The leftmost 16 bits of register T are cleared by this instruction. The conversion is undefined for binary results greater than $+(2^{47}-1)$ or less than $-(2^{47}-1)$. Thus, the largest decimal number that this instruction can convert is $\pm$ 140, 737, 488, 355, 327. The instruction sets data flag bit 39 (refer to data flag register bit assignments in section 5) for numbers outside this range.

If the input number is not a valid BCD number, the results are undefined. The ASCII/EBCDIC sign code for the BCD number is in bits 60 through 63 of register R.

The following are signs recognized for BCD to binary conversion.

+ 1010	+1100	+1110
-1011	-1101	+1111

11 Convert Binary to BCD, Fixed Length

This instruction converts the rightmost 48 bits (two's complement, binary number) of register R to a packed BCD number and transfers the result to register T. The result is a number containing 15 packed BCD digits (4 bits per digit and the sign in bits 60 through 63). Figure 4-10 shows the packed BCD format; the binary range is $\pm$ $(2^{47}-1)$.

During job mode, the sign bits generated are conditioned by the ASCII/EBCDIC bit in the job's invisible package. During monitor mode, only ASCII codes will be generated.

† Appendix B describes the floating-point square root operation.

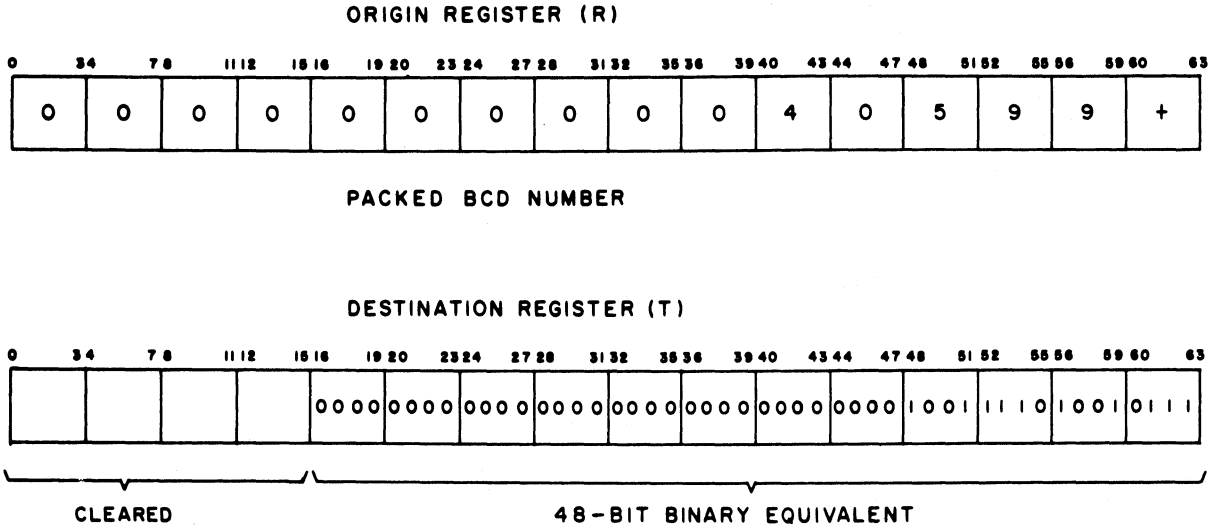


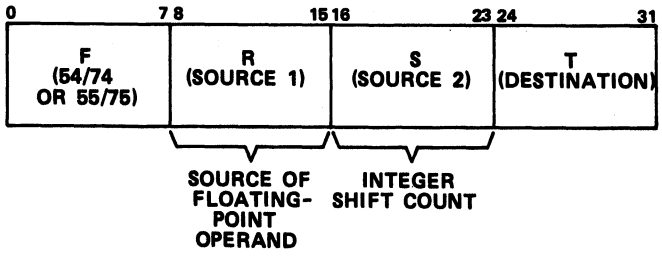
Figure 4-10. Example of Register Content for a Convert BCD to Binary, Fixed-Length Instruction

The following are signs generated for binary to BCD conversion.

	ASCII MODE	EBCDIC MODE
POSITIVE	1010	1100
NEGATIVE	1011	1101

54/74 Adjust Significance of (R) Per (S) to (T)

55/75 Adjust Exponent of (R) Per (S) to (T)



54/74 Adjust Significance of (R) Per (S) to (T)

This instruction adjusts the significance of the floating-point operand in register R and transmits the adjusted result to register T. The rightmost 24 bits (48 bits for 74 function code) of register S contains a signed, two's complement integer. The absolute value of this integer is a shift count.

If the shift count is positive, the machine shifts the coefficient of the operand left the number of positions specified by the shift count or the number of positions needed to normalize† the coefficient, whichever is the smaller number.

In either case, the instruction reduces the exponent of the operand by one count for each position shifted. The instruction left-shifts an all zero coefficient the number of positions specified.

If the shift count is negative, the instruction shifts the coefficient of the operand right the number of positions specified by the shift count and increases the exponent of the operand by one count for each position shifted. If (R) is indefinite, the machine sets the (T) to indefinite and sets data flag bit 46 (indefinite result). If (R) equals machine zero, the machine sets (T) to machine zero and data flag bit 43 will be set.

This instruction is undefined if the absolute value of the shift count is greater than 23_{10} for the 54 or 47_{10} for the 74 instruction. The addition of the shift count can cause either exponent overflow or exponent underflow.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

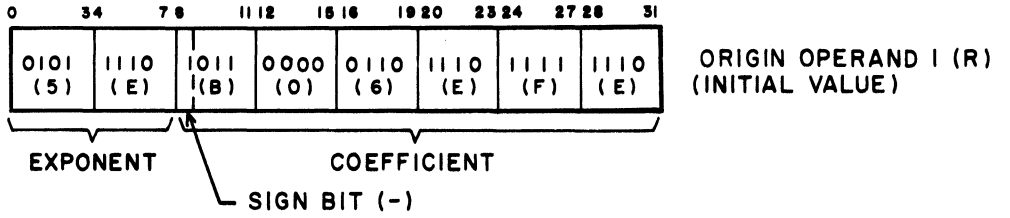
55/75 Adjust Exponent of (R) Per (S) to (T)

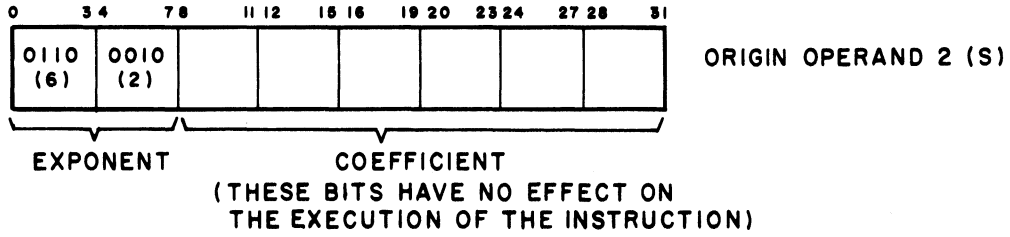
This instruction transmits the adjusted operand from register R to result register T. The instruction sets the result exponent equal to the exponent of the operand in register S. The machine forms the coefficient of the result by shifting the coefficient of the operand from register R.

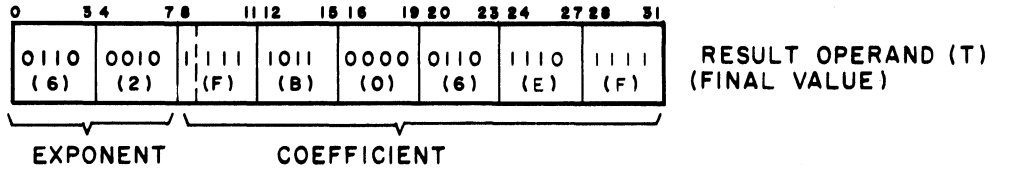
The shift count is the difference between the exponents in registers R and S. If the exponent in register R is greater than the exponent in register S, the machine shifts the coefficient left. The shift is to the right if the exponent in register R is less than the exponent in register S. If register R contains a zero coefficient, the exponent in register S is transferred to register T with an all zero coefficient. Figure 4-11 shows that the exponent in register S exceeds the exponent in register R by 4 (62 - 5E = 4); thus, the machine right-shifts the coefficient in register R four positions.

If a left shift exceeds the number of positions required for normalization, the machine sets the result to indefinite and sets data flag bit 42 (exponent overflow). If either or both operands are indefinite or machine zero, the machine also sets the result to indefinite. However, in this case, data flag bit 46 (indefinite result) is set and data flag bit 42 (exponent overflow) is not set.

†Appendix B describes the process of adjusting a floating-point operand for significance and of normalizing a floating-point number.



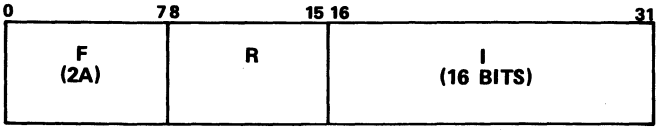




NOTE: NUMBERS IN PARENTHESES REPRESENT
HEXADECIMAL EQUIVALENTS OF BINARY GROUPS

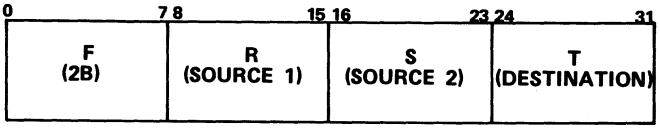
Figure 4-11. Example of Register Content for an Adjust Exponent of (R) Per (S) to (T)

2A Enter Length of (R) with I (16 Bits)



This instruction transfers operand I contained in the rightmost 16 bits of the instruction word to the leftmost 16 bits of the 64-bit register specified by R. The rightmost 48 bits of register R are left unchanged.

2B Add to Length Field



This instruction adds bits 0 through 15 of the 64-bit register specified by R to bits 48 through 63 of 64-bit register S and stores the result in bits 0 through 15 of register T. Overflow is ignored if it occurs. Bits 16 through 63 of register R are transferred to bits 16 through 63 of register T.

BRANCH INSTRUCTIONS

The branch instructions compare or examine single bits, a 48-bit index, 32-bit floating-point operands, or 64-bit operands. The results of the comparison or examination determine whether the program continues with the next sequential instruction (branch condition not met) or branches to a different instruction sequence (branch condition met). The different instruction sequence may consist of a single instruction or a series of instructions beginning at the branch address specified in the branch instruction format.

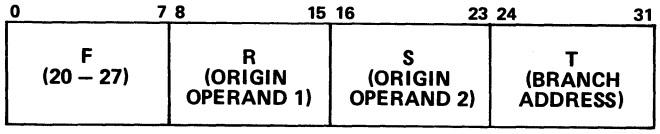
A special branch instruction provides for entering or leaving the monitor program. All item counts in branch instructions are in half-words.

20/24 Branch if (R) = (S) (32/64 Bit FP)

21/25 Branch if (R) ≠ (S) (32/64 Bit FP)

22/26 Branch if (R) ≥ (S) (32/64 Bit FP)

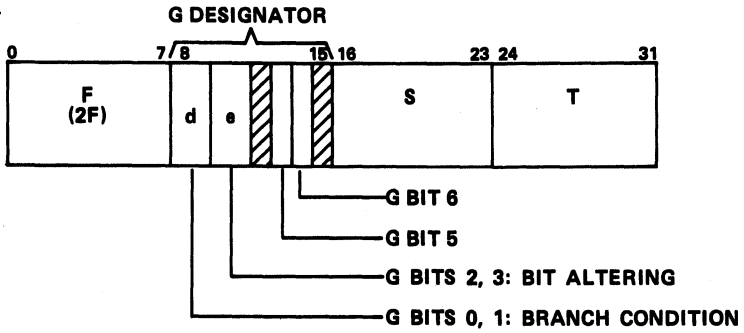
23/27 Branch if (R) < (S) (32/64 Bit FP)



These instructions perform the indicated comparison of the 32-bit (64-bit for the 24 through 27 function codes) floating-point operands in the registers designated by R and S. If the specified comparison condition is met, the next instruction is read from the branch address, contained in the rightmost 48 bits of 64-bit register T. Register T is a 64-bit register for the 20 through 27 instruction codes. The byte and bit portions of the address (bits 59 through 63) are ignored in the reading of an instruction. If the specified comparison condition is not met, the next instruction is read from the next sequential program address. The comparison of (R) and (S) is based on the floating-point compare rules in appendix B. An example of a 22 instruction is also in appendix B.

If either or both of the compared operands are indefinite, data flag bit 46 is set.

2F Register Bit Branch and Alter



This instruction examines bit 63 of register T as specified by the G designator. A branch is made to the address contained in the rightmost 48 bits of register S. The branch occurs according to G bits 0 and 1 (table 4-4).

TABLE 4-4. BIT BRANCHING CONDITIONS

G Designator		Branch Conditions
Bit 0	Bit 1	
0	0	No branch
0	1	Unconditional branch
1	0	Branch if object bit = 1
1	1	Branch if object bit = 0

After the branch decision has been made and regardless of the decision, the object bit is altered according to G bits 2 and 3 (table 4-5).

TABLE 4-5. BIT ALTERING CONDITIONS

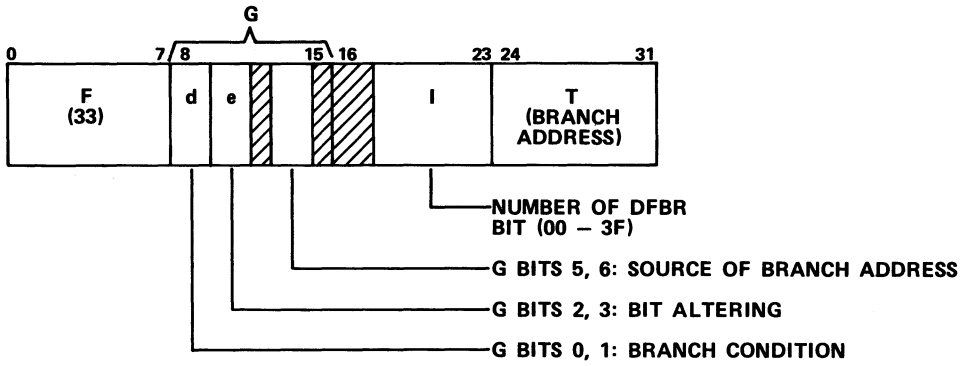
G Designator		Altering Conditions
Bit 2	Bit 3	
0	0	No altering
0	1	Toggle the bit
1	0	Set the bit 1
1	1	Clear the bit 0

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

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

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

33 Data Flag Register Bit Branch and Alter



This instruction examines the state of a specified bit in the data flag branch register (DFBR). If the designated branch condition is met, the next instruction is read from the half-word address as specified by G designator bits 5 and 6. If the designated branch condition is not met, the next instruction is read from the next sequential program address. In either case, the state of the DFBR bit is altered as specified by G bits 2 and 3.

The 6-bit designator I specifies the number of the DFBR bit. The bit numbers range from 00 through 3F (00 through 63_{10}). The 2-bit designator denotes the branch condition (table 4-6).

TABLE 4-6. DFBR BIT BRANCH CONDITIONS

G Designator		Branch Conditions
Bit 0	Bit 1	
0	0	No branch
0	1	Unconditional branch
1	0	Branch if selected DFBR bit = 1
1	1	Branch if selected DFBR bit = 0

After the branch decision is made, the instruction alters the DFBR bit according to G designator bits 2 and 3 (table 4-7). The bit altering occurs regardless of the branch decision.

TABLE 4-7. DFBR BIT ALTERING CONDITIONS

G Designator		Altering Conditions
Bit 2	Bit 3	
0	0	No altering
0	1	Toggle the bit
1	0	Set the bit 1
1	1	Clear the bit 0

NOTE

Do not attempt to alter bits in the DFBR product field since the altering of these bits is only a function of the corresponding data flag and flag mask bits.

Since the 33 instruction may begin execution without waiting until the machine has completed all operations (for example, the scalar divide's data flags may not have reached the Data Flag Register), the data flag bits (except free data flag bits 53, 54, and 55) may set on any minor cycle during or after execution of the 33 instruction. Consequently, any data flag bits that set after the object bit is sampled will not affect the operation of the 33 instruction, but will be retained in the Data Flag Register for follow on sampling.

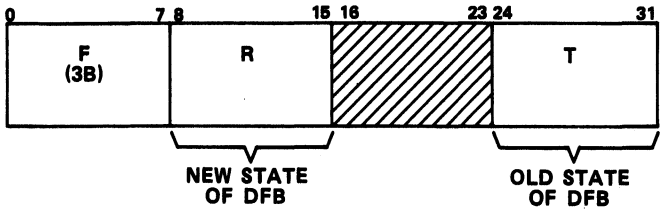
Operations that alter free data flag bits 53, 54, and 55 are always completed before the 33 instruction checks or alters these bits.

The source of the branch address is determined by the state of G designator bits 5 and 6 (table 4-8).

TABLE 4-8. DFBR BRANCH ADDRESS SOURCE CONDITIONS

G Designator		Branch Address Source Conditions
Bit 5	Bit 6	
0	0 or 1	Register T contains the branch address.
1	0	Branch address is formed by addition of the T designator, used as a half-word item count, to the content of the program address register.
1	1	Branch address is formed by the subtraction of the T designator, used as a half-word item count, from the contents of the program address register.

3B Data Flag Register Load/Store



This instruction transfers the content of register R to the DFB register. The 3B instruction also transmits the previous content of the DFB to the T register. Since the DFB is a 64-bit register, both R and T must be 64-bit registers. The R and T designators may be equal which exchanges data flag values.

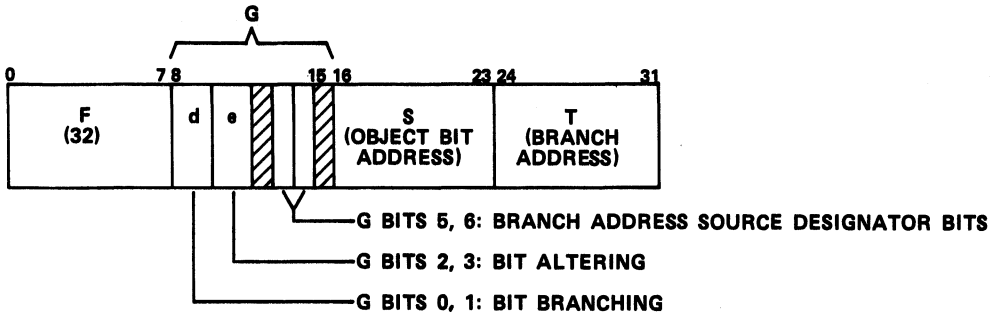
CAUTION

The data flag bit 36 sets asynchronously in the Data Flag Register (DFR); meaning that it can set any time either prior to or after the 3B instruction load/unload minor cycle. This can cause a problem concerning the retention of the original data from the DFR. There is no problem if the data flag bit sets after the load/unload minor cycle. In this case the original data remains in the DFR. However, if the data flag bit sets prior to the load/unload cycle the original data in the DFR moves into the T register. The programmer must specify a number other than 00_{16} in the T designator in order to retain the data in this register. If 00_{16} is in the T designator the data will be lost.

NOTE

An immediate data flag branch results at the termination of this instruction if the new content of the DFB register meets the appropriate branch conditions.

32 Bit Branch and Alter



This instruction reads the word from the address contained in the register designated by S and examines the specified object bit. The remaining bits are not used in the instruction.

If the object bit meets the branch condition specified by G designator bits 0 and 1, the next instruction is read from the branch address contained in the T register. If the branch condition is not met, the next instruction is read from the next sequential program address. In either case, G designator bits 2 and 3 determine the final state of the object bit. Tables 4-9 and 4-10 list the bit branching and altering conditions, respectively. Table 4-11 lists branch address source conditions.

TABLE 4-9. BIT BRANCHING CONDITIONS

G Designator		Branch Conditions
Bit 0	Bit 1	
0	0	No branch
0	1	Unconditional branch
1	0	Branch if object bit = 1
1	1	Branch if object bit = 0

TABLE 4-10. BIT ALTERING CONDITIONS

G Designator		Altering Conditions
Bit 2	Bit 3	
0	0	No altering
0	1	Toggle the bit
1	0	Set the bit 1
1	1	Clear the bit 0

NOTE

If G bits 0, 2, and 3 = 0, the word containing the object bit is not read and the object bit is not altered. If G bit 0 = 1 and G bits 2 and 3 = 0, the word is read but the object bit is not written.

TABLE 4-11. BRANCH ADDRESS SOURCE CONDITIONS

G Designator		Branch Address Source Conditions
Bit 5	Bit 6	
0	0 or 1	Register T contains the branch address.
1	0	Branch address is formed by addition of the T designator, used as a half-word item count, to the contents of the program address register.
1	1	Branch address is formed by the subtraction of the T designator, used as a half-word item count, from the contents of the program address register.

Figure 4-12 shows an example of the bit branch and alter instruction with assumed register content and branch conditions. The object bit is located in bit 7 of byte 3 of word 100000. Since G bit 0 equals 1 and G bit 1 equals 0 and the object bit is a 1, a branch takes place to the assumed branch address which is contained in the T register as specified by G designator bits 5 and 6.

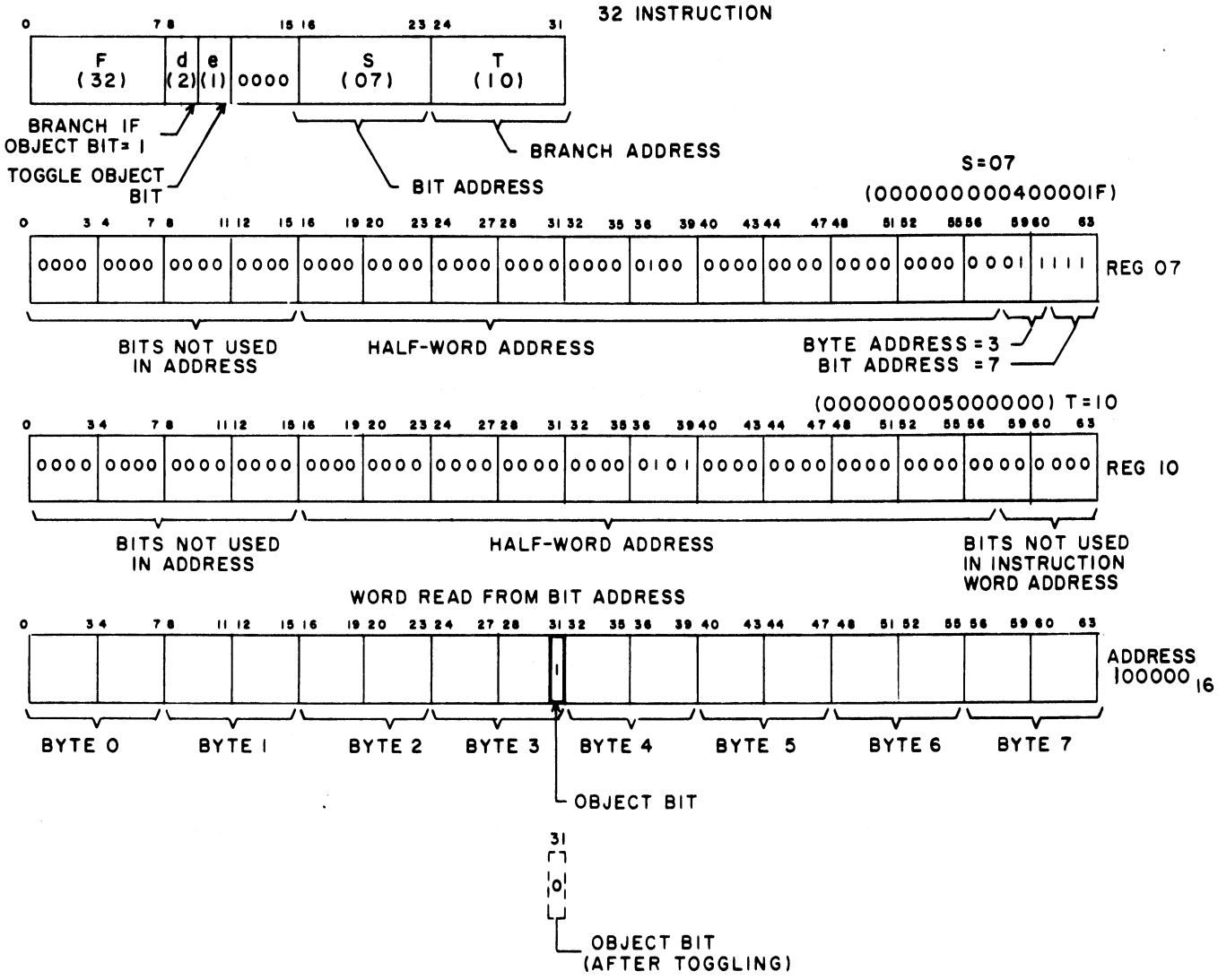
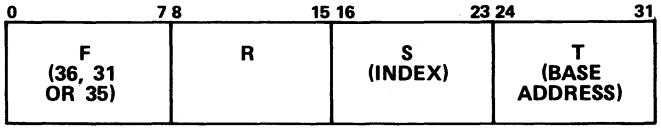


Figure 4-12. Example of Bit Branch and Alter Instruction

36 Branch and Set (R) to Next Instruction

31 Increase (R) and Branch if (R) ≠ 0

35 Decrease (R) and Branch if (R) ≠ 0



36 Branch and Set (R) to Next Instruction

This instruction first stores the address of the next sequential instruction into register R. The program then branches to (S) + (T), where (S) represents an item count (index) of half-words and (T) specifies the base address. The machine forces bits 0 through 15 of register R to zeros. Bits 59 through 63 are undefined. If the instruction designator R is equal to the designator S, the results of this instruction are undefined.

If S = 0 and R = T, this instruction sets register R to the half-word address of the next instruction. The program then continues at the next instruction. This method provides a means of sampling the program address register.

31 Increase (R) and Branch if (R) ≠ 0

35 Decrease (R) and Branch if (R) ≠ 0

This instruction first increments (31 function code) or decrements (35 function code) the rightmost 48 bits of register R by one. The leftmost 16 bits of register R are not altered and arithmetic overflow (if it occurs) is ignored.

If the increment/decrement operation produces zeros in the rightmost 48 bits of R, the program reads the next sequential instruction. If the rightmost 48 bits of R are not all zeros, the program branches to (S) + (T), where (S) represents an item count in half-words and (T) specifies the base address.

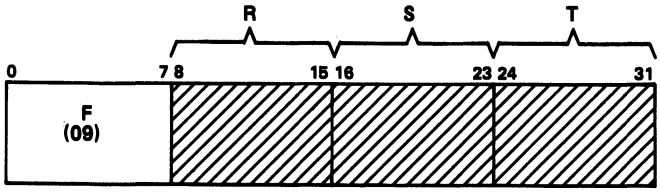
The resulting address for the branch is undefined if the R designator is equal to either the S designator or the T designator.

09 Exit Force

This instruction provides a means of exchanging program control between a job and monitor program. For example, if the machine is operating in the job mode, the exit force instruction causes a branch to the beginning address of a portion of the monitor program. Similarly, in a monitor program, the exit force performs a branch to a job program. The starting address of the invisible package and register file for the job is defined by the content of the register designated by T and S, respectively. For either type of exchange (job to monitor or monitor to job), the invisible package and register file for the current job are transferred to/from central storage. (Refer to section 5 for a more comprehensive description of monitor and job operations.)

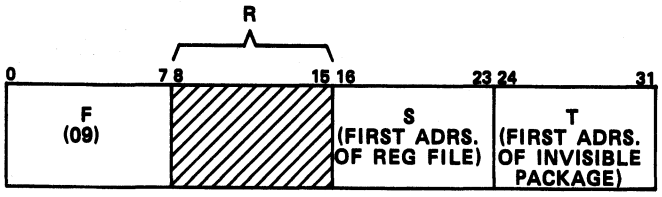
Job to Monitor

The following exit force instruction format is an exchange from a job to a monitor program. The R, S, and T designators are unused and must be zeros. In this case, the instruction switches the machine to the monitor mode and unconditionally branches to the address specified by the rightmost 48 bits of register 05 in the register file. Register 05 address is an absolute bit address since the machine was switched to the monitor mode. The monitor program then proceeds from this beginning address.



Monitor to Job

The following instruction format is an exchange from the monitor to a job program. The R designator is unused and must be zeros.

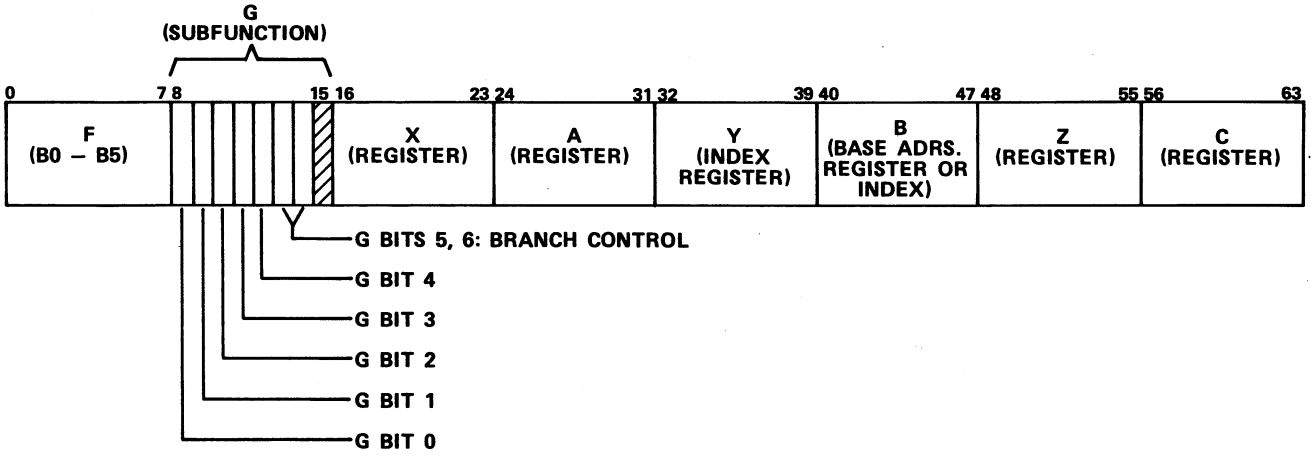


When exchanging from the monitor mode to a job, this instruction loads the registers from the register file stored in central storage, beginning at the address contained in the register specified by S. The instruction also loads the invisible package for the applicable job from central storage, beginning at the address in the register specified by T. The S and T addresses are absolute bit addresses.

B0 Compare Integer, Branch if (A) + (X) = (Z)
B1 Compare Integer, Branch if (A) + (X) ≠ (Z)
B2 Compare Integer, Branch if (A) + (X) ≥ (Z)
B3 Compare Integer, Branch if (A) + (X) < (Z)
B4 Compare Integer, Branch if (A) + (X) ≤ (Z)
B5 Compare Integer, Branch if (A) + (X) > (Z)

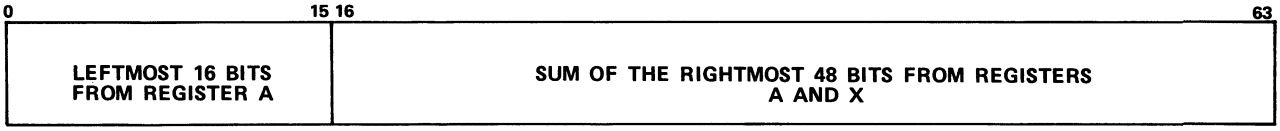
NOTE

B0-B5 instructions also have nonbranch compare capabilities. Refer to the description of compare instructions in this section.



For these instructions, G bits 1 and 2 are 0. If G bit 0 is cleared (0), registers A, X, C, and Z are 64 bits. If G bit 0 is set (1), registers A, X, C, and Z are 32 bits. Registers B and Y are 64 bits.

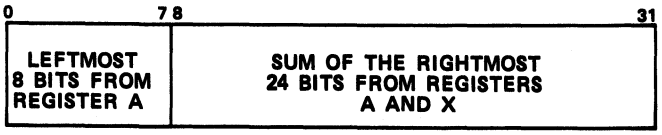
If G bit 0 is 0, the sum of the rightmost 48-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 48 bits of register Z, according to the specified branch condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 48 bits of register C. The leftmost 16 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



Then the sum of the rightmost 48 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values:

G bit 3 = 0	The integers compared are the 48-bit result of registers A and X and the rightmost 48 bits read from register Z.
G bit 3 = 1	The integers compared are the 64 bits stored in register C and the 64 bits read from register Z. This compare is defined for the B0 and B1 instructions only.
G bit 4 = 0	The integers compared are interpreted as signed two's complement numbers.
G bit 4 = 1	The integers compared are interpreted as unsigned numbers.

If G bit 0 is 1, the sum of the rightmost 24-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 24 bits of register Z, according to the specified branch condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 24 bits of register C. The leftmost 8 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



Then the sum of the rightmost 24 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values.

G bit 3 = 0	The integers compared are the 24-bit result of registers A and X and the rightmost 24 bits read from register Z.
G bit 3 = 1	Undefined.
G bit 4 = 0	The integers compared are interpreted as signed two's complement numbers.
G bit 4 = 1	The integers compared are interpreted as unsigned numbers.

If the specified branch condition is met, the program address branches to the address specified by the branch control bits in the G designator (table 4-12). In all cases, the index is an item count in half-words that is left-shifted five places before the addition or subtraction.

If designators A and/or X equal zero, machine zero will be supplied. If designator Z is equal to zero, 48 (24 for 32-bit operands) zeros are read as the rightmost bits.

TABLE 4-12. INDEX BRANCH INSTRUCTION DESIGNATORS

G Designator Bit State	Branch Address
Bit 5 = 0	Branch to address formed by adding the item count in register Y to the base address in register B. The item count is left-shifted five places before the addition. Overflow, if any, is ignored. If the Y or B designator is equal to the C designator, the instruction is undefined.
Bit 5 = 1	Branch according to the state of G designator bit 6 as follows:
Bit 6 = 0	Branch to address formed by adding 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before addition.
Bit 6 = 1	Branch to address formed by subtracting 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before subtraction.

If the branch condition is not met, the program reads the next sequential instruction.

If one of the following conditions occur, the operation becomes undefined.

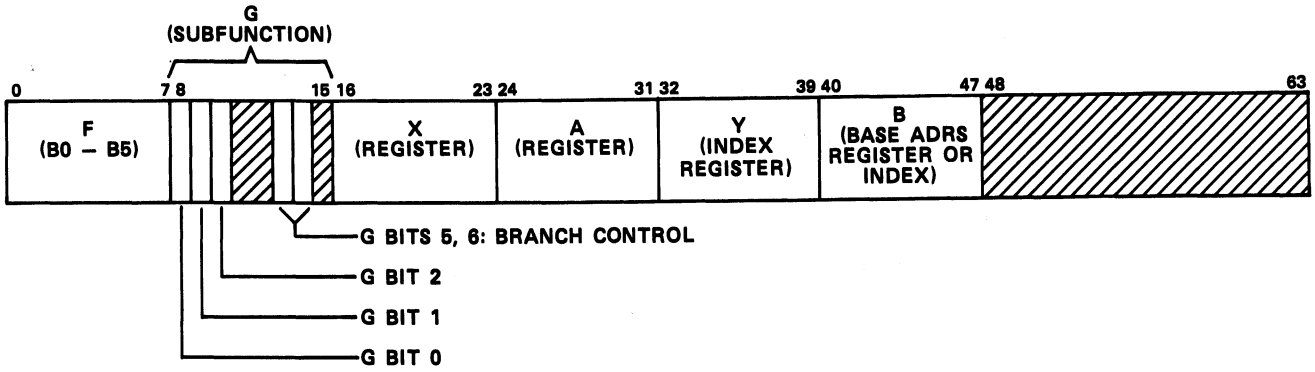
- G bit 0 is 1 and G bit 3 is 1
- G bit 3 is 1 for instructions B2, B3, B4, and B5
- G bit 5 is 0 and G bit 6 is 1

Table 4-13 relates integer ranges to the state of G bit 4.

TABLE 4-13. INTEGER RANGES

	48-bit/24-bit hexadecimal quantities in descending order from the largest to the smallest, from top to bottom.	
	G bit 4 = 0	G bit 4 = 1
Largest	7F - - - - FF	FF - - - - FF
	7F - - - - FE	FF - - - - FE
	00 - - - - 01	80 - - - - 01
	00 - - - - 00	80 - - - - 00
	FF - - - - FF	7F - - - - FF
Smallest	80 - - - - 01	00 - - - - 01
	80 - - - - 00	00 - - - - 00

B0 Compare FP, Branch if (A) = (X)
B1 Compare FP, Branch if (A) ≠ (X)
B2 Compare FP, Branch if (A) ≥ (X)
B3 Compare FP, Branch if (A) < (X)
B4 Compare FP, Branch if (A) ≤ (X)
B5 Compare FP, Branch if (A) > (X)



If G bit 1 is 1 and G bit 2 is 0, these instructions compare the two floating-point operands from registers A and X according to the floating-point compare rule in appendix B. If G bit 0 is clear (0), the registers contain 64 bits. If G bit 0 is set (1), the registers contain 32 bits. Registers B and Y are always 64 bits.

If the specified branch condition is met, the program address branches to the address specified by the branch control bits in the G designator (table 4-14). In all cases, the index is an item count in half-words that is left-shifted five places before the addition or subtraction.

TABLE 4-14. INDEX BRANCH INSTRUCTION DESIGNATORS

G Designator Bit State	Branch Address
Bit 5 = 0	Branch to address formed by adding the half-words item count in register Y to the base address in register B. The item count is left-shifted five places before the addition. Overflow, if any, is ignored. If the B or Y designator is equal to the C designator, the instruction is undefined.
Bit 5 = 1	Branch according to the state of G designator bit 6 as follows:
Bit 6 = 0	Branch to address formed by adding 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before addition.
Bit 6 = 1	Branch to address formed by subtracting 16-bit item count designators Y and B (bits 32 through 47) to the address of this instruction. The item count is left-shifted five places before subtraction.

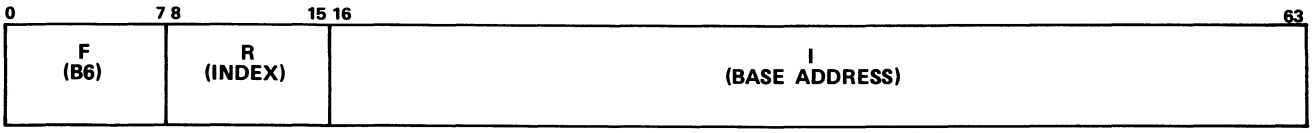
If the branch condition is not met, the program reads the next sequential instruction.

If one of the following conditions occur, the operation becomes undefined.

- G bit 3=1, G bit 4=1, or G bit 7=1.
- Designator C and/or Z not equal to 0.
- G bit 5=0 and G bit 6=1.

The applicable data flag bit is 46 (indefinite result).

B6 Branch to Immediate Address (R) + I (48 Bits)



This instruction branches unconditionally to the address formed by the sum of the rightmost 48 bits of register R as the index and I as the base address. The index represents an item count of half-words which is shifted left five positions before being added to the base address. Overflow, if any, is ignored.

The instruction makes a direct branch to the base address if the R designator is zero or if the rightmost 43 bits of register R are zeros.

VECTOR INSTRUCTIONS

The vector instructions perform operations on ordered scalars. Generally, the vector instructions read the scalars, which are in the form of 32-bit or 64-bit floating-point operands, from consecutive storage locations over a specified address range (field). These instructions perform the designated operation on each set of operands and store the results in consecutive addresses of a result field, beginning at a specified starting address. Thus, a single vector instruction can perform operations on two source fields of vector operands and automatically store the results in a result field of storage.

Instruction Formats

All vector instructions use the same general instruction format (figure 4-13). Table 4-15 lists each of the 8-bit designators in the vector instructions and gives a brief description of the function.

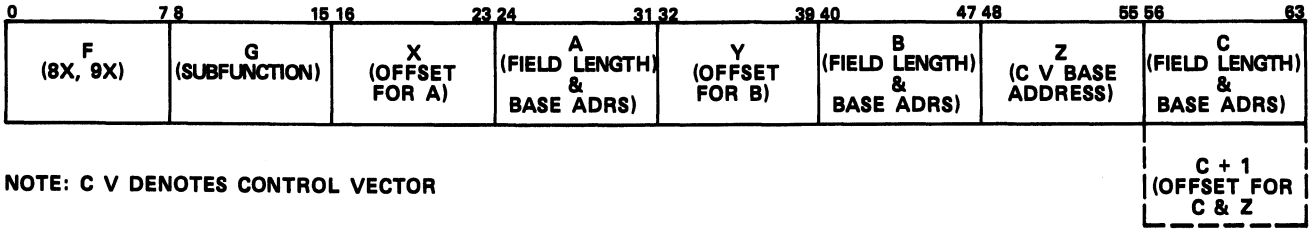


Figure 4-13. General Vector Instruction Format

TABLE 4-15. VECTOR INSTRUCTION DESIGNATORS

Designator	Function
F	Function code.
G	Subfunction code.
X, Y	Specify registers that hold address offsets for corresponding source operand fields.
A, B	Specify registers that hold base addresses and field lengths for source operand fields.
Z	Specifies register that contains the base address of the control vector (CV).
C	Specifies register that contains the base address and field length of the result field. If C+1 is used by the instruction, C must be an even number since the machine forms C+1 by forcing the rightmost bit of C to a 1. If the C designator specifies an odd-numbered register, the results of the instruction become undefined.
C+1	Specifies register that holds offset for the control vector and the result field; C+1 always references an odd register.

Subfunction Bits

Table 4-16 lists the subfunction bits and their general usage. Table 4-17 gives the sign control subfunction bits.

If the Z designator is zero, no control vector is used; thus, G-bit 1 becomes undefined. If G bit 3 and/or G bit 4 = 1, the A and/or B designator denotes a constant which is used as each element of the respective vector field. The instruction ignores the associated offsets in this case. The registers specified by A and B, respectively, contain these constants. Registers A and B are always 64-bit registers except when G bits 3 and 4 indicate a broadcast. When broadcasting, the size of registers A and B track the size specified by G bit 0 (refer to table 4-16).

Appendix C gives a composite listing of the G designator bits usage according to function code.

TABLE 4-16. SUBFUNCTION BITS

Bit Number	State	Subfunction
0	0	64-bit operands (words)
	1	32-bit operands (half-words)
1	0	Control vector operates on ones †
	1	Control vector operates on zeros †
2	0	No offset for result field and control vector
	1	Offset for result field and control vector
3	0	Normal source vectors A
	1	Broadcast repeated (A)††
4	0	Normal source vectors B
	1	Broadcast repeated (B)††
5	X	Sign control (refer to table 4-17)
6	X	
7	X	

†If the 8-bit designator Z is zero, no control vector is used, so bit 1 of G is undefined. (All output operands are stored.)

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

TABLE 4-17. SIGN CONTROL SUBFUNCTION BITS

Bit 5	Bit 6	Bit 7	Control Operation
0	0	X	The operands from the A stream are used in the normal manner.
0	1	X	The coefficients of the operands from the A stream are complemented before they are used.
1	0	X	The magnitude of the coefficients of the operands from the A stream is used.
1	1	X	The coefficients of all positive operands from the A stream are made negative before they are used. Negative operands are not altered.
X	X	0	The operands from the B stream are used in the normal manner.
X	X	1	The magnitude of the coefficients of the operands from the B stream is used.

NOTES

1. X denotes that the bit can be either a 0 or a 1.

2. Any required complementing is two's complement. Complementing is performed before the operand is used in the specified arithmetic operation. If the complement of the coefficient 8000 0000 0000 is required, the operand is used as 4000 0000 0000 with 1 added to the exponent.

3. Any necessary significance calculation is performed before the previous complementing is performed.

Field Lengths, Base Address, and Offsets

Figures 4-14 and 4-15 show the formats of the register contents for the field lengths, base addresses, and offsets. The computer allows 16-bit field lengths to be specified and assumes them to be positive. The field lengths are in the range of 0 through $2^{16}-1$ before any offset adjustments. The offsets are taken from a 48-bit register and must have at least 32 identical sign bits, otherwise the instruction is undefined. The offsets are in the range of -2^{16} to $2^{16}-1$.

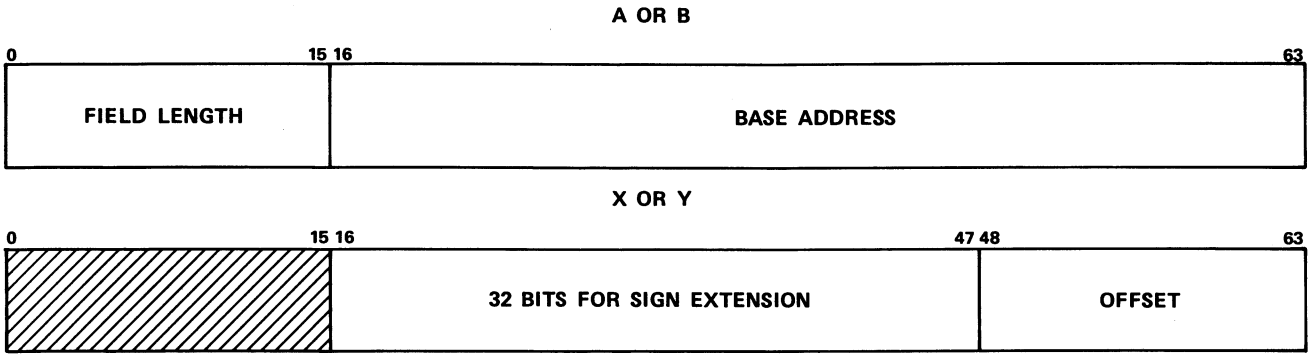


Figure 4-14. Operand Field Length, Base Address, and Offset Formats

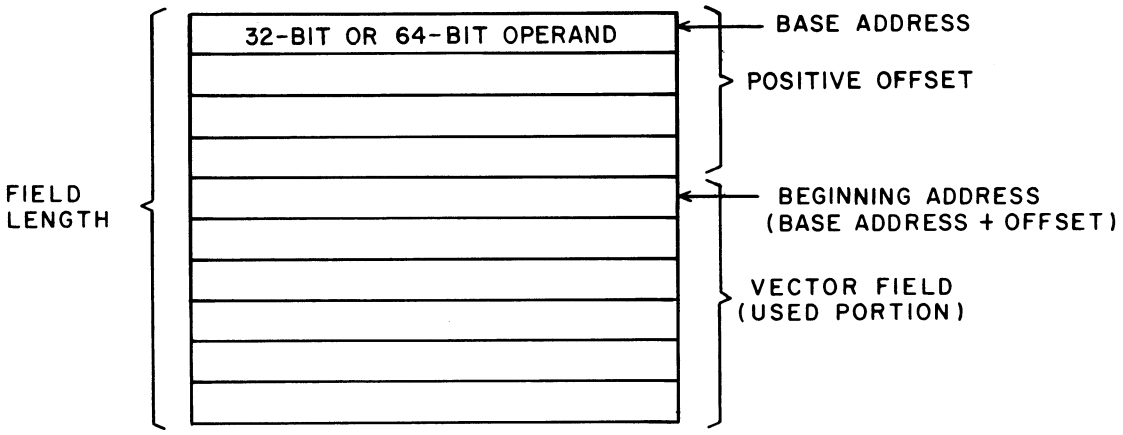


Figure 4-15. Vector Field Address Format

The operation of subtracting the offset from the field length must result in a field length which is positive and less than 2^{16}. If the resulting vector length is not positive and less than 2^{16}, it is treated as a zero vector length. The instruction obtains the beginning address by adding the offset (including sign extension) to the base address (figures 4-15 and 4-18). In the (offset + base address) addition, the offset is first shifted left five (half-words) or six (words) places since the bit and byte bits are not used in the vector operand field address.

The C and C+1 registers are identical in format to the A or B and X or Y content, respectively. If bit 2 specifies that vector field C is to be offset, register C+1 contains the offset.

Control Vector

When the instruction specifies a control vector (Z designator $\neq$ 0), a single bit from the control vector controls the storing of each element in the result field. When a bit from the control vector prohibits the storing of a result element, the instruction does not alter the previous content of the corresponding storage address and a data flag bit cannot be set for that result element. Thus, the nth bit read from the control vector prohibits or allows the storing of the nth result in the result vector field.

Bit 1 of the G designator selects whether a 0 or a 1 control vector bit allows the storing of the result (table 4-11). If bit 1 of the G designator is a 0 or a 1, the instruction stores the nth result if the nth bit of the control vector is a 1 or a 0, respectively.

The rightmost 48 bits of the register designated by Z contains the base address of the control vector (figure 4-16). The control vector uses the same field length as result vector C.

The addition of the offset and base address provides the starting bit address of the control vector. Since offsets are item counts, the result vector and control vector use the same offset; however, the control vector offset represents a bit offset.

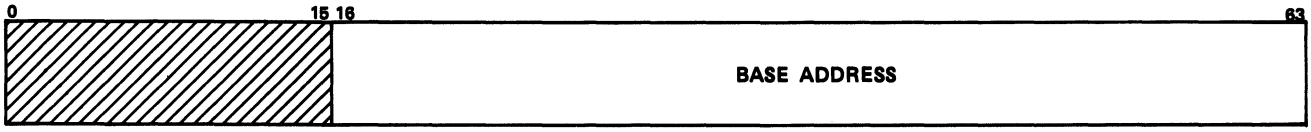


Figure 4-16. Control Vector Base Address Format (Z)

Vector Instruction Termination

Vector instructions terminate when the result vector field is exhausted.†

1. Exhausting a vector which has an offset.

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

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

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

   A vector is exhausted prior to the first operand fetch if the field length is read from register zero.

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

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

Example of Vector Instruction Operation

Figure 4-17 shows the register content and figure 4-18 shows the resulting vector address fields of an assumed add U, A+B → C (80) vector instruction. Although an 80 instruction is used, the general sequence of operations is the same for all vector instructions.

The G designator bits used in the example specify the following conditions for the operation of the instruction.

G-Designator Bit	Condition
0 = 1	32-bit, floating-point operands.
1 = 0	Control vector operates on ones (ones in control vector enable storage of corresponding control vector).
2 = 1	Result vector and control vector fields are offset (C+1 designator is used).
3 = 0	Normal vector source stream A.
4 = 0	Normal vector source stream B.
5 = 0 6 = 0	Use the operands from the A stream in the normal manner.
7 = 0	Use the operands from the B stream in the normal manner.

†Appendix C provides a complete listing of the various vector instruction field conditions and the resulting termination conditions.

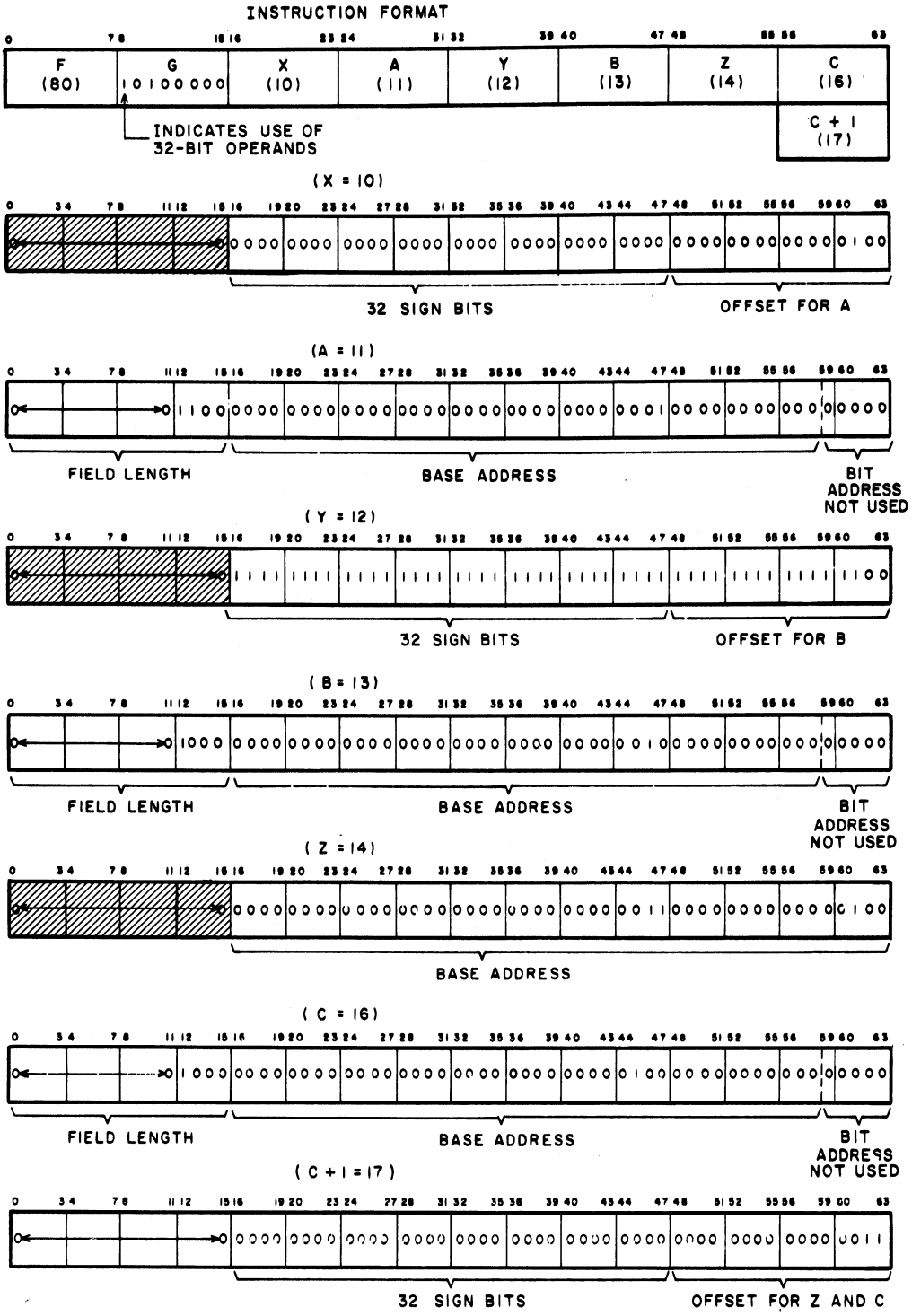


Figure 4-17. Vector Instruction Example of Register Content and Instruction Format

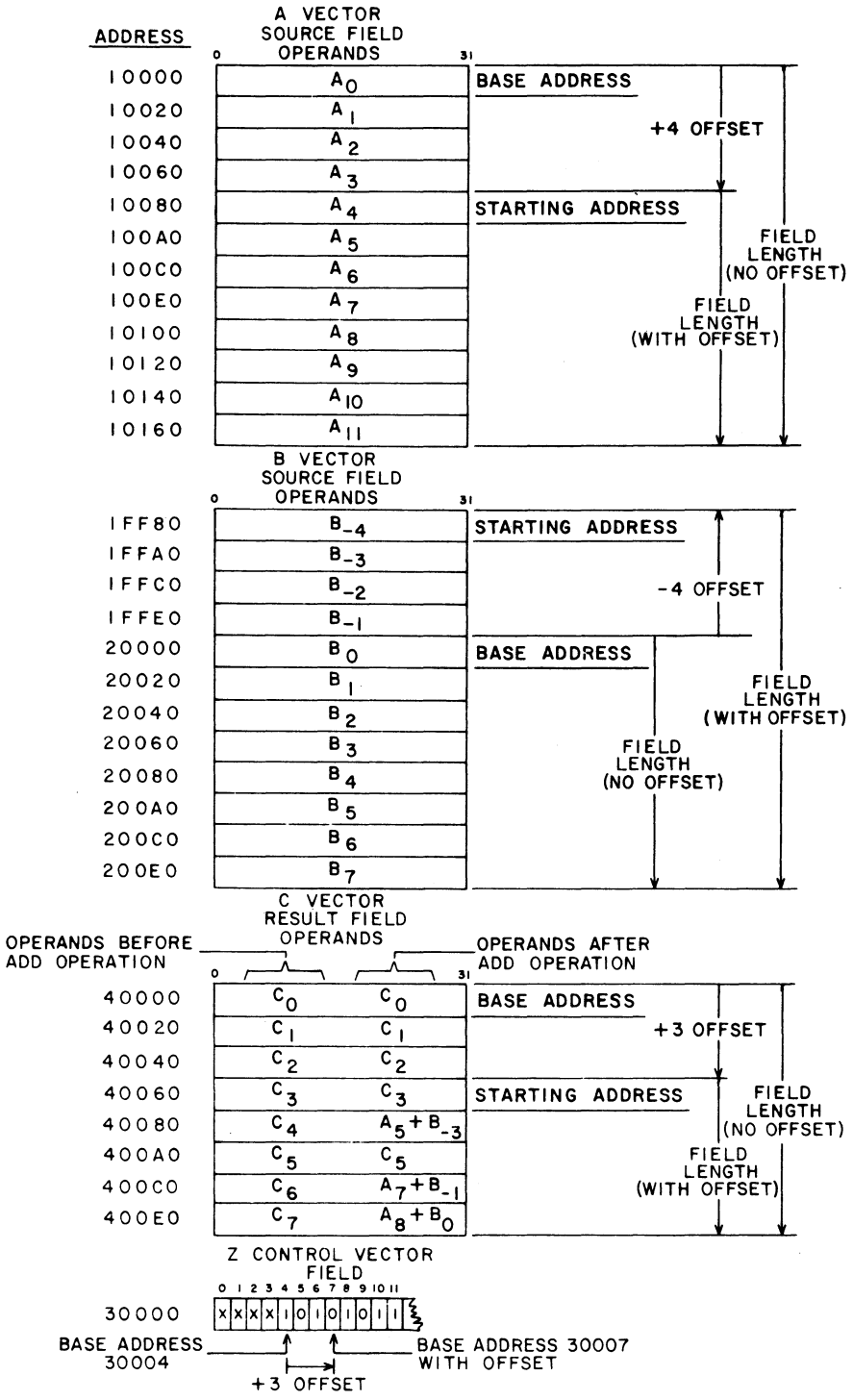


Figure 4-18. Vector Address Fields for Vector Instruction Example

The X, A, Y, B, Z, and C register designator numbers are shown in parentheses. Thus, register 10 contains the offset for vector field A, register 11 contains the base address for vector field A, and so forth.

Since the bit and byte address bits are not used in the vector field addresses, successive half-word addresses are shown. Thus, incrementing address 10000_{16} by a half-word count gives 10020_{16} as the next successive address.

With the A vector offset equal to +4 and the B vector offset equal to -4 (figures 4-17 and 4-18), the first vector add U, A+B → C operation adds the A and B operands from the respective addresses 10080_{16} and $1FF80_{16}$. The result of the first add operation does not store, because bit 7 of the addressed control vector field is a zero. Successive add operations add successive A and B operands, storing the results only when a corresponding one appears in the control vector.

80 Add U; A + B → C

81 Add L; A + B → C

82 Add N; A + B → C

84 Sub U; A - B → C

85 Sub L; A - B → C

86 Sub N; A - B → C

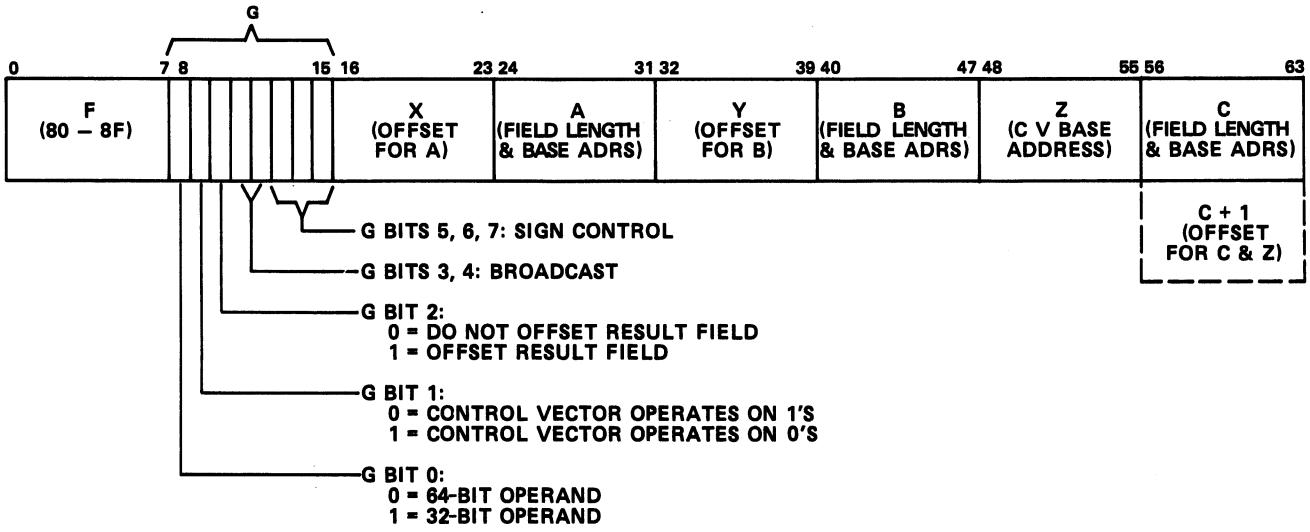
88 Mpy U; A • B → C

89 Mpy L; A • B → C

8B Mpy S; A • B → C

8C Div U; A/B → C

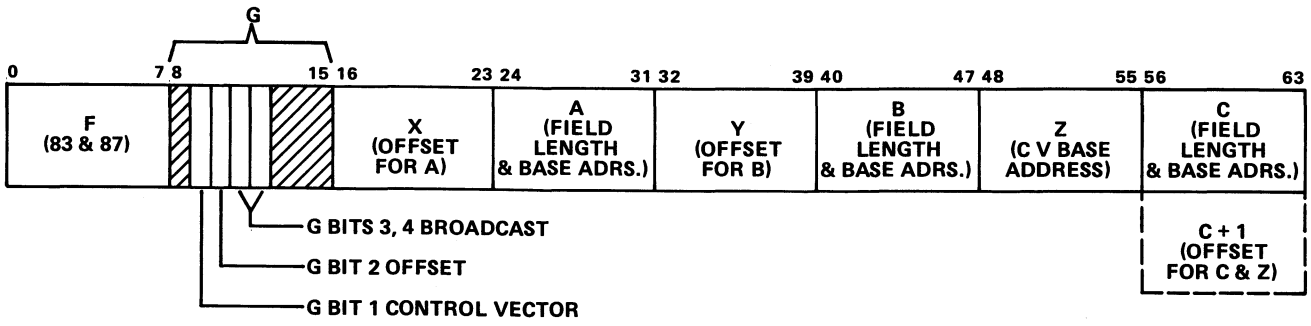
8F Div S; A/B → C



These instructions perform the indicated floating-point† arithmetic operations on the elements of vector fields A and B. The instructions store the result elements in vector field C. All of the vector elements are in the form of 32-bit or 64-bit floating-point operands. The U, L, N, and S designators specify the upper, lower, normalized upper, or significant results, respectively.

Applicable data flag bits are 41 (floating-point divide fault), 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

83 Add A; A + B → C
87 Sub A; A + B → C



These instructions add/subtract bits 16 through 63 of the B vector elements to/from bits 16 through 63 of the A vector elements (figure 4-19). The instructions store the results in bits 16 through 63 of the C vector elements. Bits 16 through 63 of the source vector elements are treated as 48-bit, unsigned, positive integers. Arithmetic overflow is ignored if it occurs.

The instructions transmit bits 0 through 15 of the A vector elements to corresponding portions of the C vector elements. G bits 0, 5, 6, and 7 are undefined and must be set to zero.

†Appendix B describes the floating-point arithmetic operations.

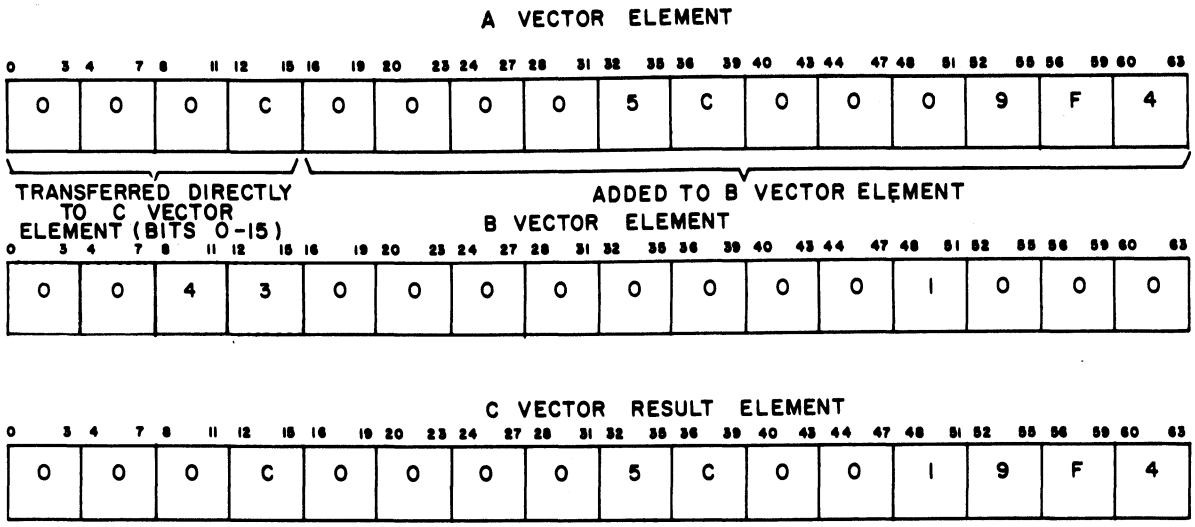
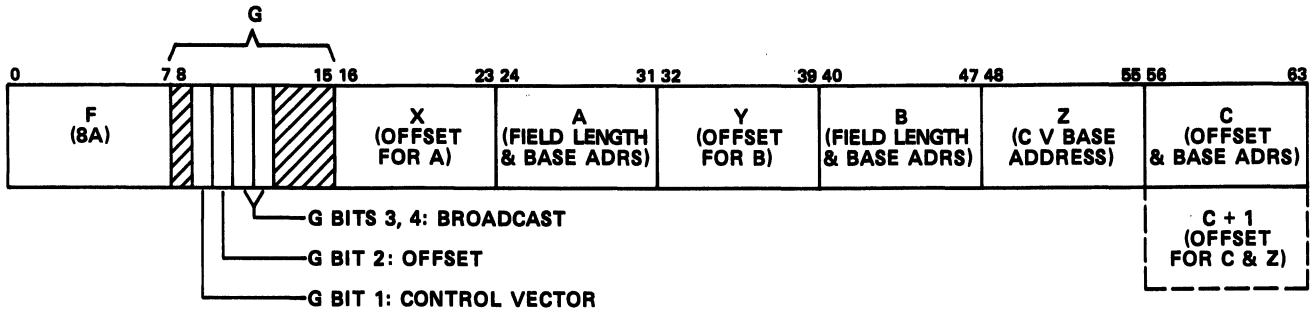


Figure 4-19. Example of an Add A; A + B → C Instruction

8A Shift; A/B → C



This instruction shifts the 64-bit elements from source vector A by corresponding elements from source vector B and stores them into result vector C. If the rightmost byte of the element in vector B is in the range from 0 through 3F base 16 (0 through 63 base 10), the element from vector A is shifted left end-around the number of specified places. If the rightmost byte of the element in vector B is in the range from FF through C1 base 16 (-1 through -63 base 10), the element from vector A is shifted right with sign extension. Bit 0 of operands in vector A is the sign bit for extension and the number for right shifts is equal to the two's complement of the rightmost bytes of operands in vector B. If the rightmost byte of elements from vector B is greater than 3F or less than C1 base 16, the results are undefined. The leftmost seven bytes of elements in vector 8 are ignored.

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

98 Transmit A → C

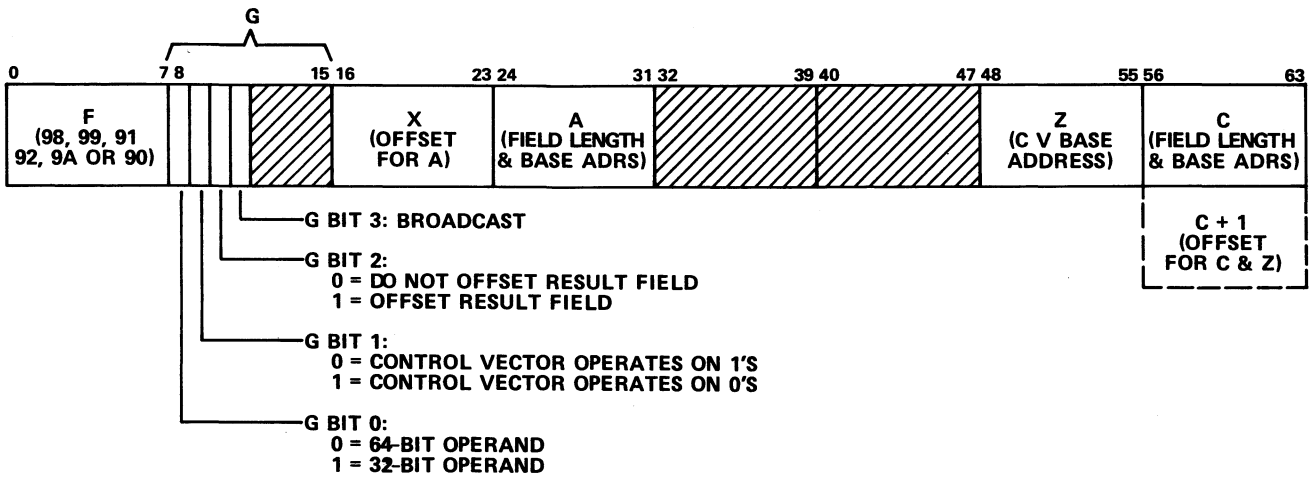
99 Absolute A → C

91 Floor A → C

92 Ceiling A → C

9A Exponent of A → C

90 Truncate A → C



98 Transmit A → C

This instruction transmits each element of the source field A to successive elements of result field C. The Y and B designators and G bits 4 through 7 are unused and must be zeros.

99 Absolute A → C

This instruction transmits the absolute value of each element of the source field A to successive elements of result field C. All vector elements are 32- or 64-bit, floating-point operands. If the coefficient of the source operand is positive, the element is transmitted directly to the result vector field; if the coefficient is negative, the coefficient is complemented before transmission. The Y and B designators and G bits 4 through 7 are unused and must be zeros.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

91 Floor A → C

This instruction converts each floating-point element of source field A to the nearest integer less than or equal to it. The resulting integers are transmitted to corresponding elements of result field C. The resulting integer is always an unnormalized, floating-point number with a positive exponent.

If the exponent of the source element is positive (greater than or equal to zero), the instruction transmits the element directly to the result field. If the exponent of the source element is negative, the instruction right-shifts the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The instruction then transmits the shifted coefficient with zero exponent to the corresponding element of result field C.

The Y and B designators and G bits 4 through 7 are unused and must be zeros. If zero is used as a source element, the instruction transmits all zeros as the corresponding result element.

Figure 4-20 shows an example of a floor A→C (91) operation with one assumed source vector element. Since the exponent of the source element is negative, the instruction right-shifts the coefficient three places and increments the exponent plus three. The sign bits are extended on the left. The result element becomes a minus one. Thus, the floor A→C (91) instruction provides a means of converting positive fractions to zero and negative fractions to a minus one.

The applicable data flag bit is 46 (indefinite result).

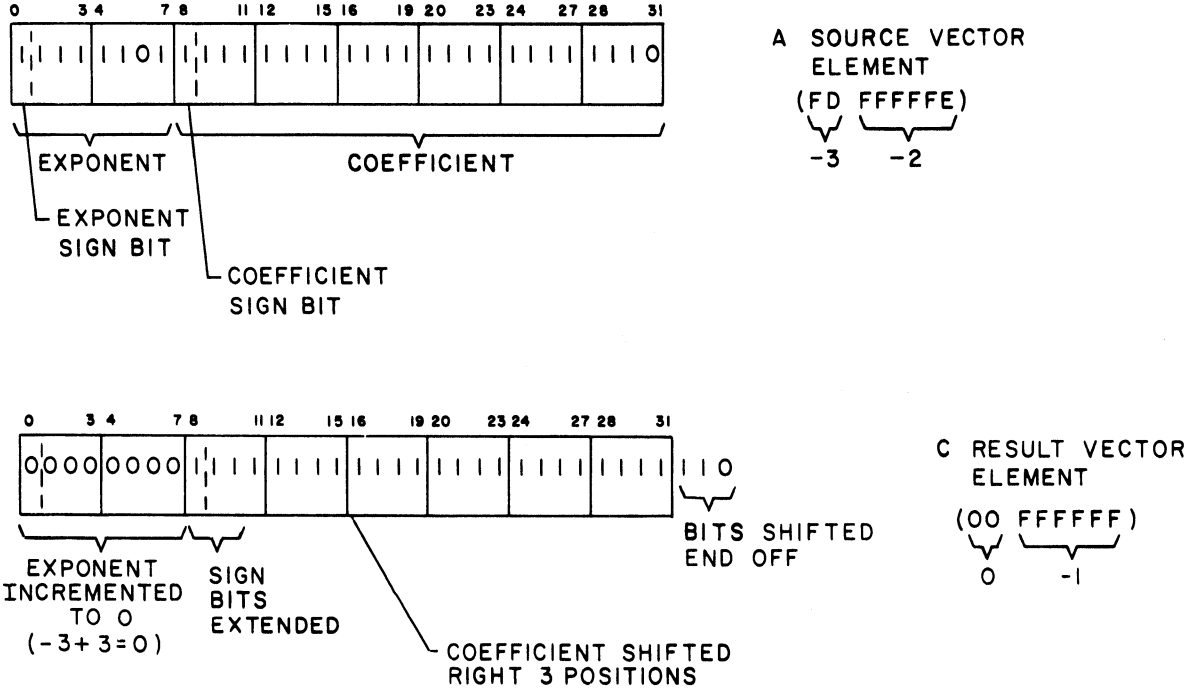


Figure 4-20. Example of Floor A→C Instruction with Negative Exponent

92 Ceiling A → C

This instruction converts each floating-point element of source field A to the nearest integer greater than or equal to it. The resulting integers are transmitted to corresponding elements of result field C. The resulting integer is always an unnormalized floating-point number with a positive exponent.

If the exponent of the source element is positive, the instruction transmits the element directly to the result field. If the exponent of the source element is negative, the instruction right-shifts the two's complement of the coefficient end-off and increases the exponent by one for each position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The instruction then recomplements the shifted coefficient and transmits it with zero exponent to the corresponding element of the result field.

The Y and B designators and G bits 4 through 7 are undefined and must be zeros. If machine zero is used as a source element, the instruction transmits all zeros as the corresponding result element.

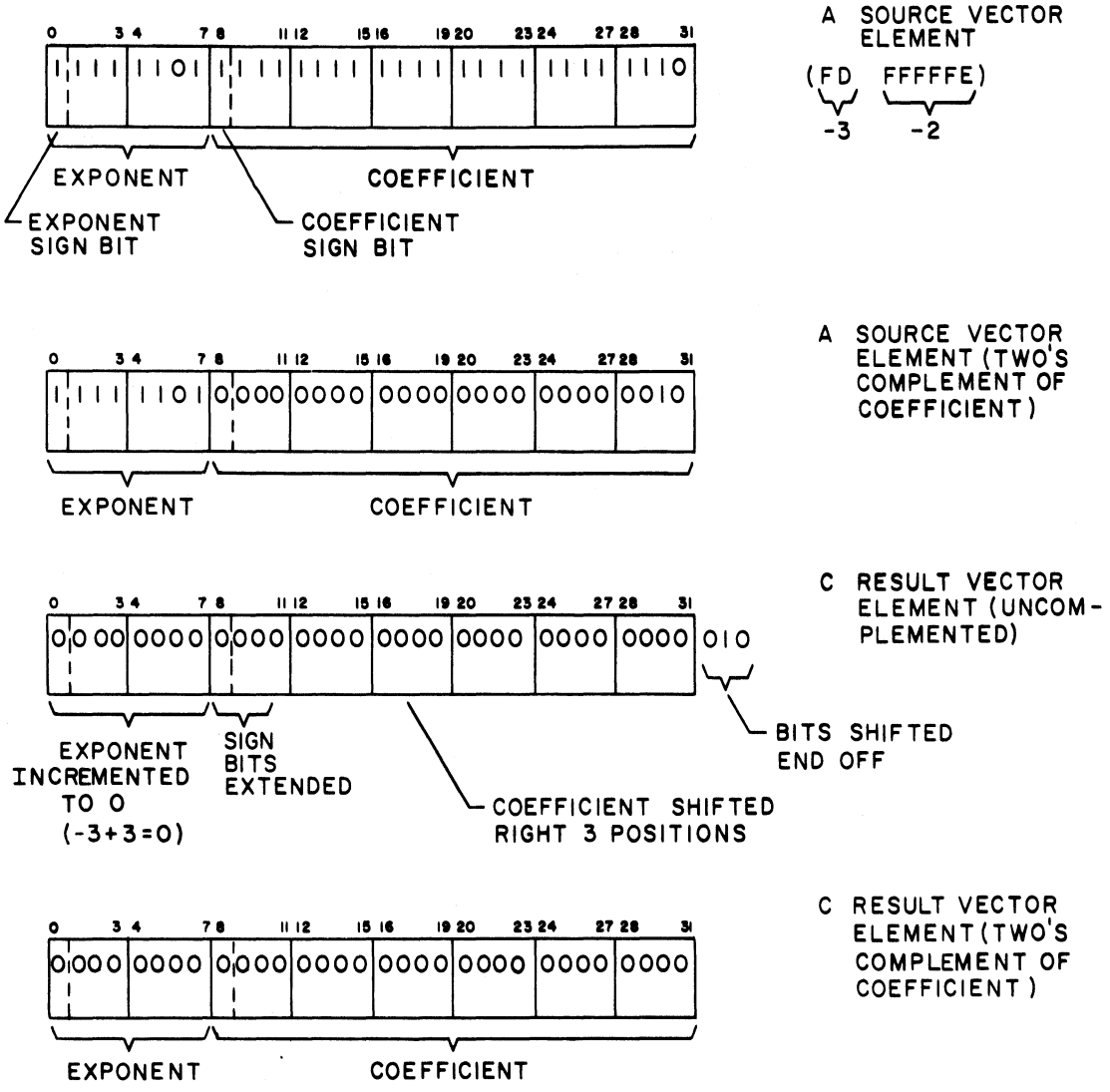
Figure 4-21 shows an example of a ceiling A → C (92) operation with one assumed source vector element. Since the exponent of the source element is negative, the instruction right-shifts the two's complement of the coefficient three places and increments the exponent by plus three. The zero sign bits are extended on the left. The result element becomes all zeros. Thus, zero is the closest integer greater than the A source vector element. The ceiling A → C (92) instruction provides a means of converting negative fractions to zero and positive fractions to plus one.

The applicable data flag bit is 46 (indefinite result).

9A Exponent of A → C

The elements of result vector C are formed by storing the exponents from input vector A into the rightmost position of the coefficients of vector C. The sign of the exponent is extended left to the coefficient sign bit position. The exponent portion of each element of vector C is cleared to zero.

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



NOTE: 32-BIT OPERANDS ARE ASSUMED.

Figure 4-21. Example of Ceiling A → C Instruction with Negative Exponent

90 Truncate A → C

This instruction transmits to elements of vector C the nearest integer the magnitude of which is less than or equal to the corresponding elements of source vector A. These integers are represented by unnormalized floating-point numbers having positive exponents.

If the origin-operand exponent is positive (greater than or equal to zero), the instruction transmits the source element directly to the corresponding result elements.

If the source-element exponents are negative, the machine right-shifts the magnitude of the corresponding coefficients end-off and increases the exponent by one for each position shifted until the exponent becomes zero.

The operation extends zeros on the left during the shift after complementing if the coefficient is negative. If the coefficient of a source element is positive, the shifted coefficient with zero exponent is transmitted to the corresponding result element. If the coefficient of a source element is negative, the two's complement of the shifted coefficient and zero exponent are transmitted to the corresponding result element. If zeros are transmitted as a source element, zero is also transmitted as the corresponding result element.

Figure 4-22 shows a typical source element and the corresponding result element for a truncate A → C (90) instruction. A 32-bit source element with a positive coefficient and negative exponent is assumed. A right shift of eight is required to reduce the negative exponent to zero.

The Y and B designators and G bits 4 through 7 are undefined and must be set to zeros.

The applicable data flag bit is 46 (indefinite result).

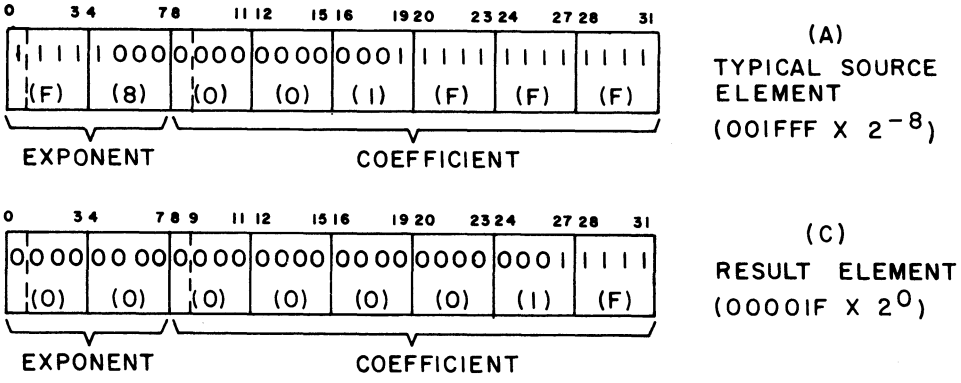
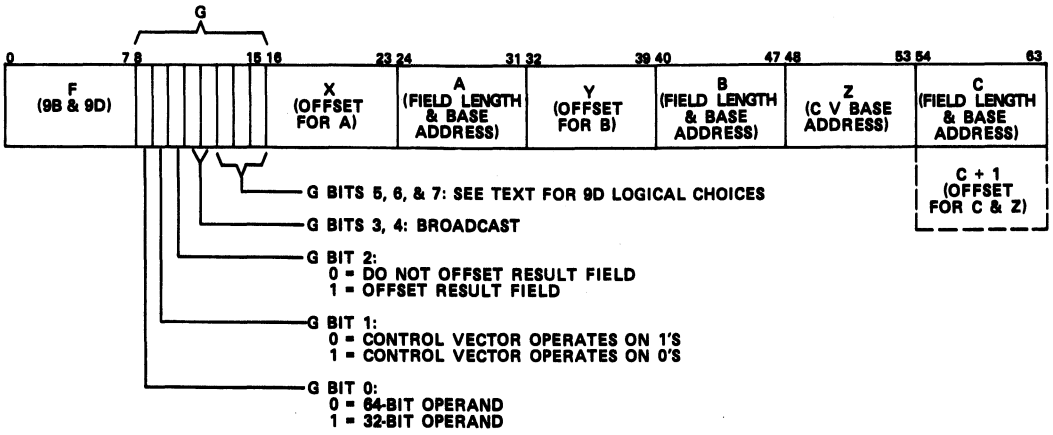


Figure 4-22. Example of Source and Result Elements for a Truncate A → C Instruction

9B Pack A, B → C

9D Logical; A, B → C



9B Pack A, B → C

This instruction forms the elements of a floating-point result vector C. The elements of result vector C consist of exponents from the rightmost 16 bits (64-bit operands) or 8 bits (32-bit operands) of source vector A elements and coefficients from the rightmost 48 bits/24 bits of the corresponding elements of source vector B.

Figure 4-23 shows an example of an assumed A source and B source vector element used in forming a C result vector element in a pack A, B → C instruction.

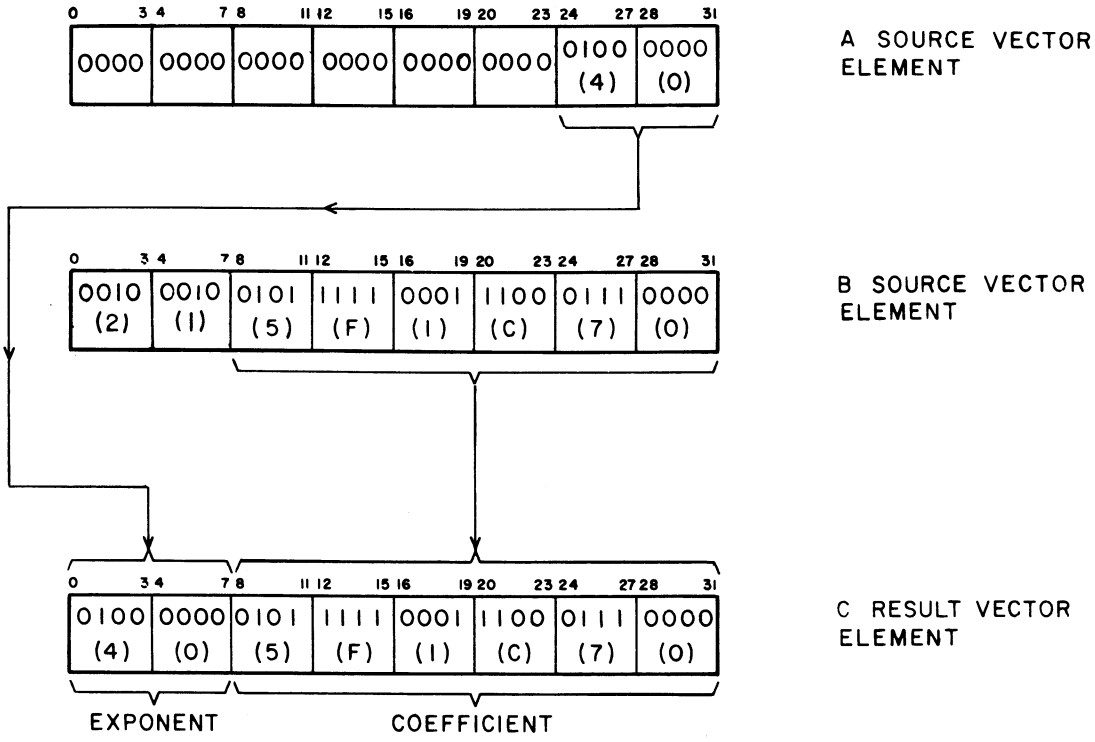


Figure 4-23. Example of Pack A, B → C Instruction

9D Logical; A, B → C

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

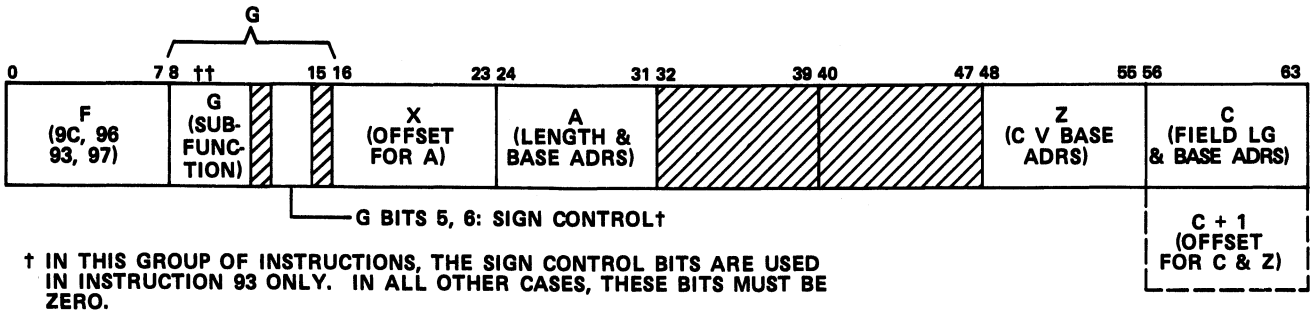
G Bits 5, 6, and 7	000	001	010	011	100	101	110	111
A B	Exclusive OR A-B	AND A.B	OR A+B	Stroke $\overline{(A.B)}$	Pierce $\overline{(A+B)}$	Implication $A+\overline{B}$	Inhibit $A.\overline{B}$	Equivalence $A-\overline{B}$
0 0	0	0	0	1	1	1	0	1
0 1	1	0	1	1	0	0	0	0
1 0	1	0	1	1	0	1	1	0
1 1	0	1	1	0	0	1	0	1

9C Extend 32 Bit A → 64 Bit C

96 Contract 64 Bit A → 32 Bit C

97 Rounded Contract 64 Bit A → 32 Bit C

93 Significant Square Root of A → C



† IN THIS GROUP OF INSTRUCTIONS, THE SIGN CONTROL BITS ARE USED IN INSTRUCTION 93 ONLY. IN ALL OTHER CASES, THESE BITS MUST BE ZERO.

†† G BIT 0 MUST BE A ZERO FOR THE 9C, 96, AND 97, INSTRUCTION BUT MAY BE A ZERO OR A ONE FOR THE 93 INSTRUCTION.

9C Extend 32 Bit A → 64 Bit C

This instruction forms the elements of result vector C by extending the 32-bit, floating-point operands of vector field A into 64-bit, floating-point operands. The instruction reduces the exponent of the result elements by 24_{10}. The 9C instruction transmits the rightmost 24 bits of the corresponding source elements to bits 16 through 39 of the result elements. The rightmost 24 bits of each result element are cleared.

If an element of vector A is indefinite, the instruction sets the corresponding element of vector C to indefinite and sets data flag bit 46. If an element of vector A is machine zero, the instruction stores machine zero as the corresponding element of vector C and sets data flag bit 43 (result machine zero).

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

Since the instruction uses only one source field, the Y and B designators and bits 0, and 4 through 7 of the G designator are not used. These bits must be zeros.

Figure 4-24 shows an example of the extension of one assumed source element into the corresponding result element. The instruction reduces the exponent of the assumed source element ($4F_{16}$) by 24_{10} to 37_{16}. The sign of the result exponent is extended in bits 0 through 7. The 9C instruction always clears bits 40 through 63 of the result-element coefficients.

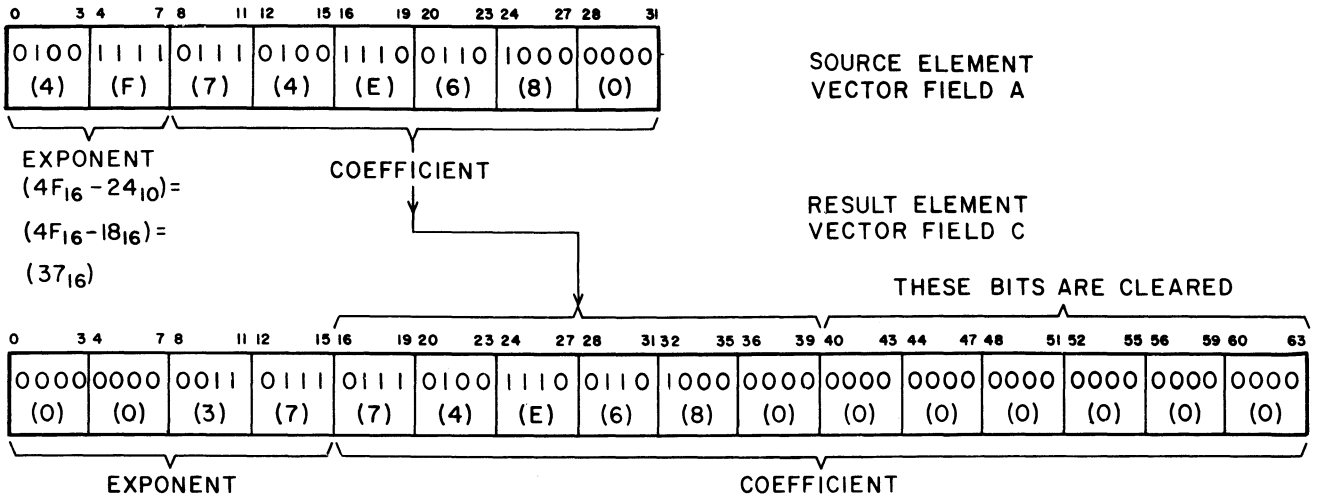


Figure 4-24. Example of Extend 32 Bit A → 64 Bit C Instruction

96 Contract 64 Bit A → 32 Bit C

This instruction contracts each 64-bit, floating-point element of vector field A into its corresponding 32-bit floating-point result. The result element becomes the corresponding element of result vector field C. The instruction increases each non-end case source-element exponent by 24_{10} in forming the 8-bit exponent for the result element.

The following is a list of input exponents and the corresponding result of the 96 instruction execution.

Input Exponent	Result
7FFF . . 7000	Result indefinite. Data flag bit 46 (indefinite result) is set.
6FFF . . 0058	 Data flag bits 42 (exponent overflow) and 46 (indefinite result) are set.
0057 . . FF78	Result exponent is 24_{10} larger than the input exponent. The leftmost 24 bits of the input coefficient are transferred.
FF77 . . 8000	Result is machine zero. Data flag bit 43 (result machine zero) is set.

The coefficient of the result element is the leftmost 24 bits of the source element coefficient. This operation contracts the coefficients of all elements with an absolute value of less than 2^{24} (neglecting the exponent) to minus one for negative coefficients and zero for positive coefficients.

The Y and B designators and bits 0 and 4 through 7 of the G designator are not used and must be zeros. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

97 Rounded Contract 64 Bit A → 32 Bit C

This instruction performs a rounded contract operation on the 64-bit, floating-point elements of vector field A and transmits the 32-bit, floating-point results to elements of vector field C (figure 4-25). Each resulting 8-bit exponent represents the sum of the least significant 8 bits of the source element and 24_{10}. If the result exponent cannot be contained in 8 bits, exponent overflow or underflow is detected.

The instruction then adds a plus one to bit positions 40 of the source-element coefficients. If overflow occurs (figure 4-25), the instruction increases the exponent by one and right-shifts the coefficient one place. (Since the result coefficient in figure 4-25 contains all zeros, the example does not show the right-shift of one place.) The leftmost 24 bits of the shifted result coefficient are transmitted to the corresponding bits of result element C. The exponent of each non-end case result element is 24_{10} (25_{10} if overflow occurred) greater than the exponent of the corresponding source element.

The Y and B designators and bits 0 and 4 through 7 of the G designator are not used and must be zeros. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

93 Significant Square Root of A → C

This instruction forms the square root† of each element of vector field A and places the result in each corresponding element of vector field C. Each result element contains the same number of significant bits as the corresponding source element.

Since the instruction uses only one source field, the Y and B designators and bits 4 and 7 of the G designator are not used and must be zeros. Bits 5 and 6 of the G designator perform sign control functions as given in table 4-17. Applicable data flag bits are 43 (result machine zero), 45 (square root result imaginary), and 46 (indefinite result).

†Appendix B describes the floating-point square root operation.

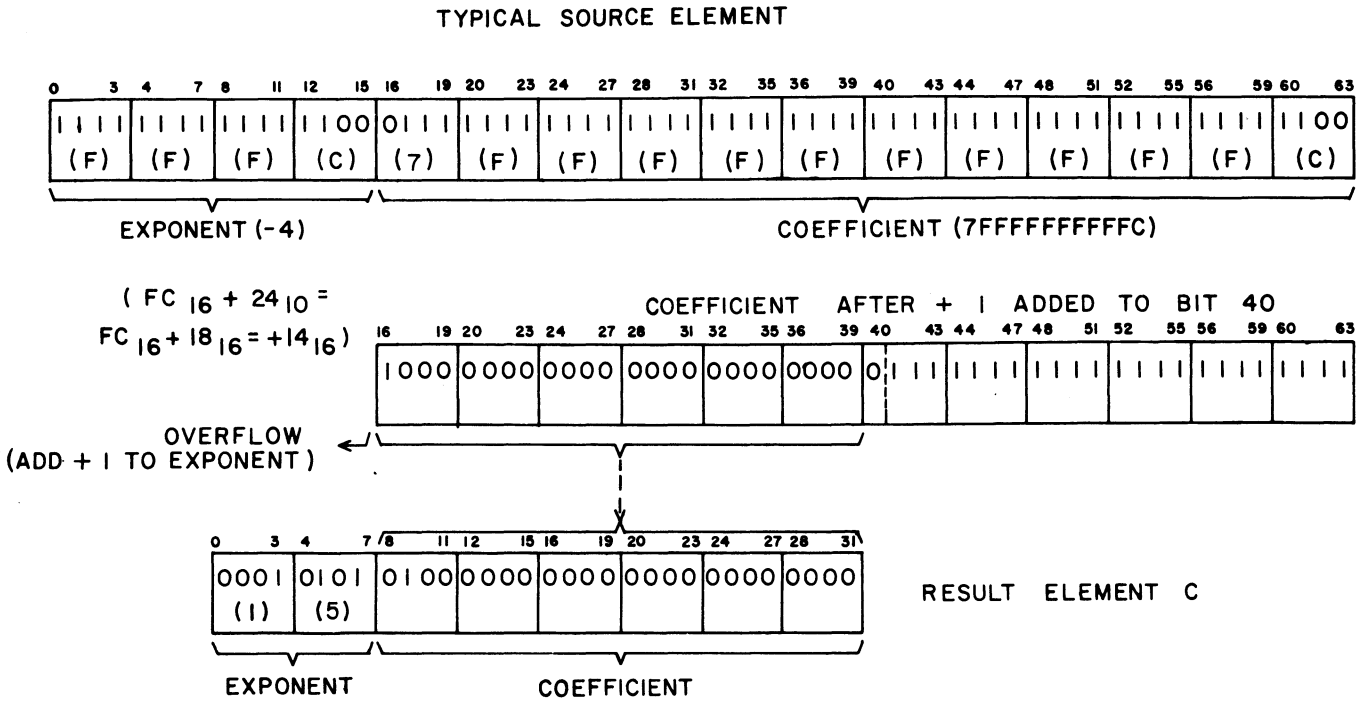
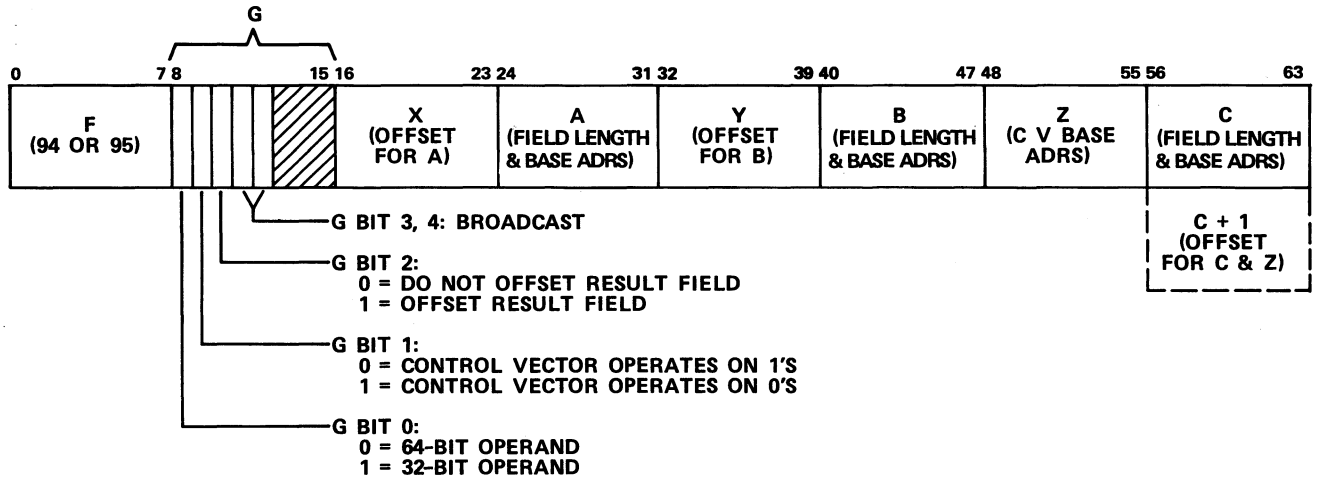


Figure 4-25. Example of Vector Elements for a Rounded Contract 64 Bit A → 32 Bit C Instruction

94 Adjust Significance of A Per B → C

95 Adjust Exponent of A Per B → C



94 Adjust Significance of A Per B →C

This instruction adjusts the significance of floating-point† elements from vector field A and transmits the adjusted elements to corresponding elements of vector field C. The rightmost 48 (64-bit operands)/24 (32-bit operands) bits of the elements in vector field B contain signed, two's complement integers. The absolute values of these integers are shift counts.

If a shift count is positive, the instruction left-shifts the coefficient of the element from vector field A the number of positions specified by the shift count or by the number of positions necessary to normalize the coefficient, whichever is smaller. In either case, the instruction reduces the exponent of the source element by one for each position shifted. The instruction left-shifts an all zero coefficient by the specified number of positions.

If a shift count is negative, the instruction right-shifts the coefficient of the source element by the shift count. The instruction increases the exponent by one for each position shifted. If the absolute value of the shift count is greater than 47_{10} (23_{10} for 32-bit operands), the shift operation is undefined. The addition of the shift count can cause either exponent overflow or underflow.

If the source element is indefinite, the instruction sets the corresponding result element to indefinite and sets data flag bit 46 (indefinite result). If the source element is machine zero, the instruction sets the corresponding result element to machine zero and sets data flag bit 43 (result machine zero). Data flag bit 42 (exponent overflow) is also applicable.

G bits 5 through 7 are undefined and must be set to zeros.

95 Adjust Exponent of A Per B →C

This instruction transmits adjusted source elements from vector field A to corresponding result elements in vector field C. The instruction sets the exponent of a result element equal to the exponent of the associated source element in vector field B. The coefficients of the result elements are formed by shifting the coefficients of the source elements from vector field A.

The difference between the exponents of associated elements from vector fields A and B forms the shift count. If the exponent from A is greater/less than the exponent of the element from B, the shift is to the left/right, respectively. If A contains a zero coefficient, the exponent of the corresponding element of B is transferred to the corresponding element of C with an all zero coefficient. If a left shift exceeds the number of positions required for normalization, the corresponding result element is set to indefinite, and data flag bit 42 (exponent overflow) is set. G bits 5 through 7 are undefined and must be set to zeros.

If either or both source elements are indefinite or machine zero, the instruction sets the result element to indefinite. In this case, data flag bit 46 (indefinite result) is set and data flag bit 42 (exponent overflow) is not set.

†Appendix B describes the operation of adjusting floating-point operands.

Figure 4-26 shows one adjust exponent of A per B → C operation with assumed 32-bit source elements for vector fields A and B. The exponent of the source element in vector field B is greater than the source element from field A by eight. As a result, the instruction right-shifts the coefficient eight positions end-off. The vacated positions on the left are filled with zeros.

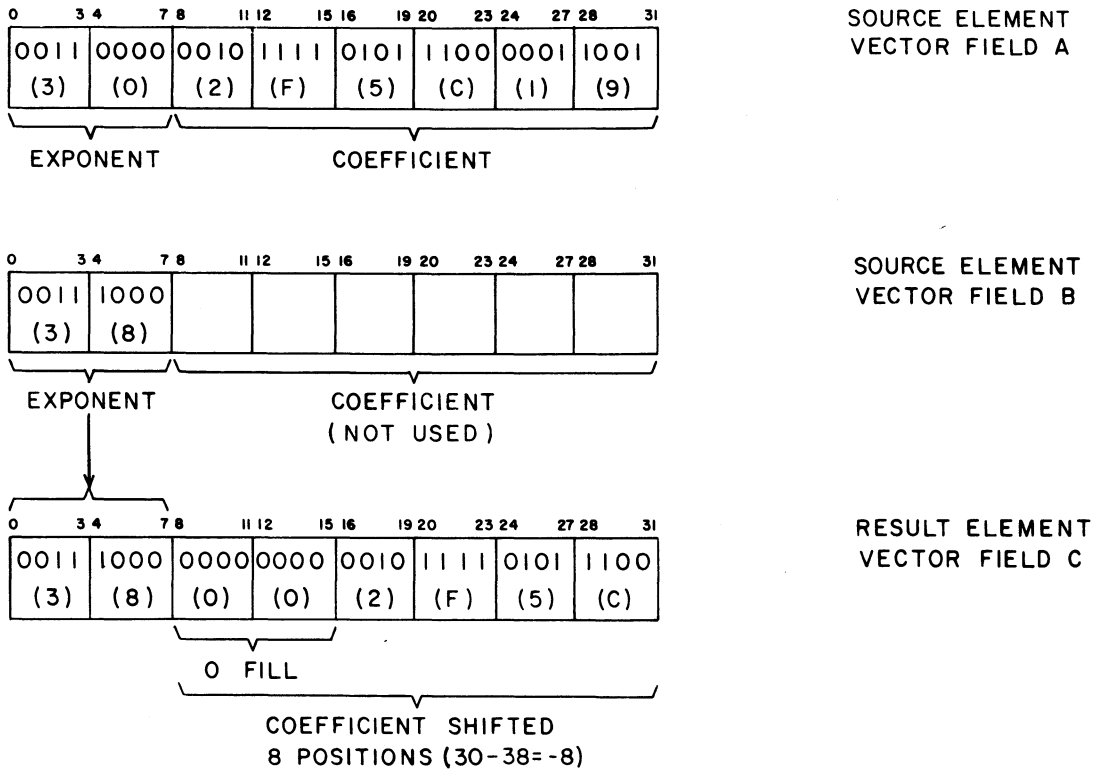


Figure 4-26. Example of Adjust Exponent of A Per B → C Operation

SPARSE VECTOR INSTRUCTIONS

Arithmetic operations may reduce many elements of a vector field to a zero or near-zero value. Except for positional significance, the near zero values need not occupy storage locations as floating-point operands in the vector field. In order to conserve storage space and calculating time, the sparse vector instructions make possible the expansion and compression of vectors of this type into sparse vectors.

A sparse vector consists of a vector pair [one of which is a bit string, identified as an order vector, and the other is a floating-point array (32- or 64-bit) identified as the data vector]. Sparse order vectors determine the positional significance of the segments of the corresponding sparse data vector.

Typically, a sparse vector is formed by the following procedure.

1. The compare instructions generate an order vector.

2. The compress A → C per Z (BC) instruction reduces the corresponding vector to a sparse vector.

3. The BC instruction uses the generated order vector as a means of discarding all near-zero elements and still maintain their positional significance through the order vector.

Figure 4-27 shows an example of compressing an initial vector into a sparse vector. Initial vector elements A_0 through A_8 are contained in consecutive, half-word addresses, beginning at arbitrary address m. A compare instruction first generates an order vector from the initial vector. The compare instruction sets the bits in the order vector corresponding to vector elements that are to be retained in the data vector. Conversely, zeros in the order vector designate the near zero elements that are to be discarded in the sparse vector field.

The compress A →C per Z instruction stores the vector elements in consecutive addresses of the data vector corresponding to ones in the order vector. Thus, the initial vector is now represented or the sparse vector consisting of the order vector and data vector.

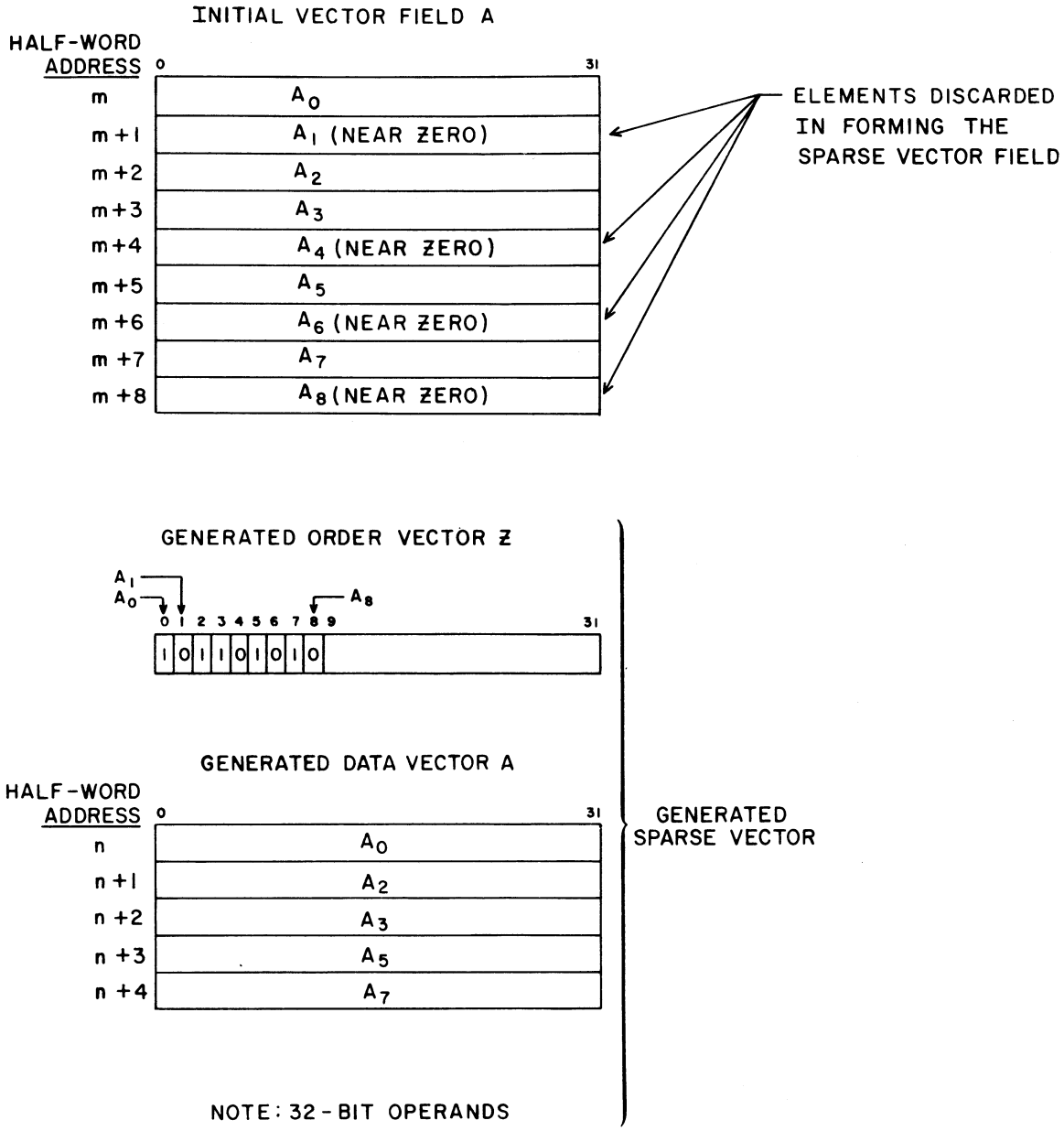


Figure 4-27. Example of Compressing Initial Vector Field into Sparse Vector Field

Sparse Vector Instruction Format

All sparse vector instructions use the same general format as shown in figure 4-28. Table 4-18 lists each of the 8-bit designator portions of the sparse vector instruction format and the corresponding definition.

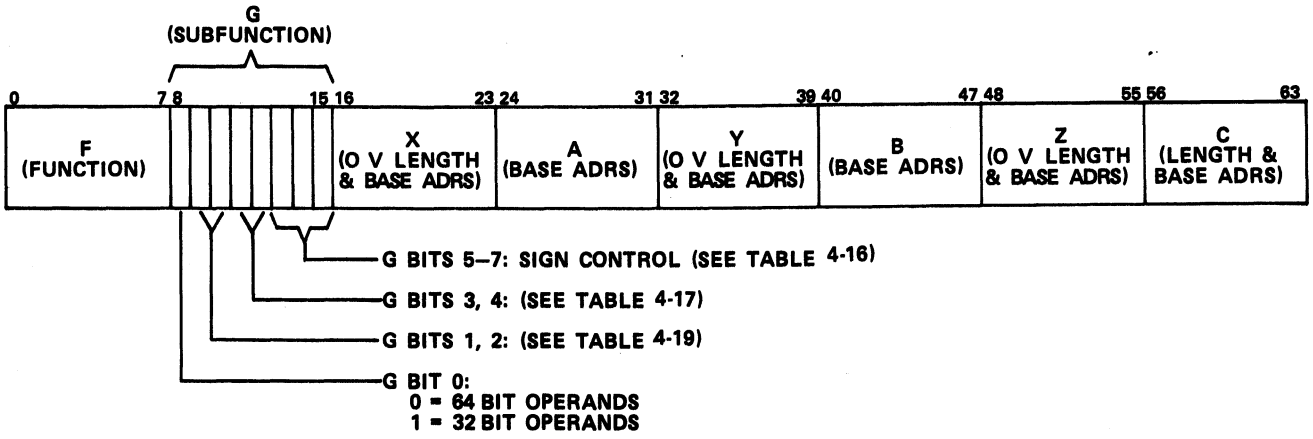


Figure 4-28. General Sparse Vector Instruction Format

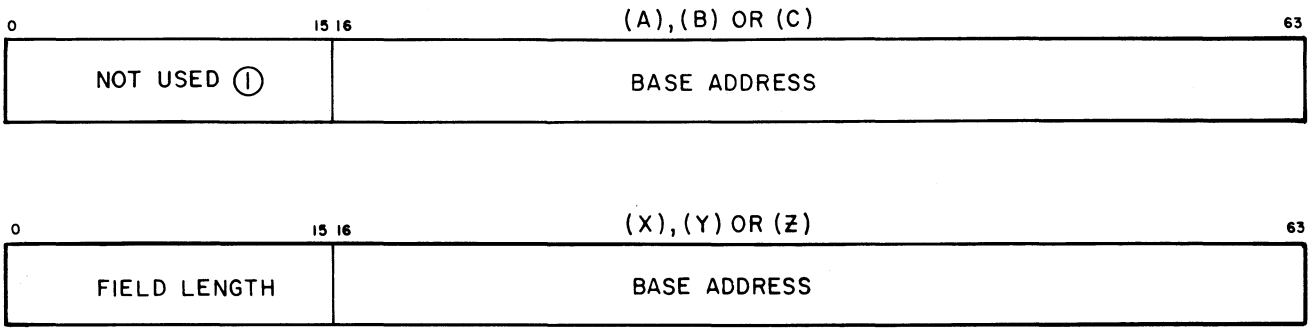
TABLE 4-18. SPARSE VECTOR INSTRUCTION DESIGNATORS

8-Bit Designator	Definition
F	Instruction code.
G	Suboperation code; the state of G bit 0 denotes the following: State — Designation 0 — 64-bit operands 1 — 32-bit operands G bits 1 and 2 are as defined by table 4-19. When bit 3 is set, the function is broadcast A. When bit 4 is set, the function is broadcast B. G bits 5 through 7 function as sign control bits (refer to table 4-17).†
X, Y	Specify the register that contains the base address and field length of the source order vector associated with source sparse data vectors A and B, respectively.
A, B	Specify the register that contains the base address of the corresponding source sparse data vector.
C	Specifies the register that contains the base address of the result sparse data vector.
Z	Specifies the register that contains the base address and the field length of the result sparse order vector associated with result sparse data vector C.

†Appendix C provides a composite listing of the G designator bits usage according to function code.

Base Addresses and Field Lengths

Figure 4-29 shows that the base addresses and field lengths for the sparse data vectors are the same format as the corresponding field lengths and base addresses of the normal vectors. However, the field lengths associated with source sparse data vectors are not used; thus, figure 4-29 shows bits 0 through 15 of the registers designated by A, B, and C as not used. The field lengths for these vectors are determined by the number of ones in the corresponding order vectors. The field lengths of the source order vectors (X and Y) and the result order vector (Z) are item counts in bits. The addresses to these order vectors are bit addresses.



(1) AT THE COMPLETION OF THE SPARSE VECTOR INSTRUCTIONS, THE LENGTH OF THE RESULTING SPARSE VECTOR IS TRANSFERRED TO THIS PORTION OF REGISTER C

Figure 4-29. Sparse Vector Field Length and Base Address Formats

Sparse Vector Instruction Termination

Sparse vector instructions terminate when the result order vector, as defined by corresponding field length, is filled. If the Z designator is zero or if the Z field length is zero, the instructions set no data flag bits and become no-operation (no-op) instructions. The sparse vector instructions terminate differently from the vector or vector macro instructions.

Source order vectors with a zero or short field length are extended with zeros as required. If vector Z contains a nonzero field length and the C designator is zero, the results of the instruction are undefined and an illegal operand will occur if a store in C vector is required.

INSTRUCTIONS A0 THROUGH AF

These instructions have different forms depending on G bits 1 and 2. Table 4-19 shows the operations associated with the values assigned to G bits 1 and 2.

TABLE 4-19. G BIT 1 AND 2 OPERATIONS

G Bit 1	G Bit 2	Operation
0	0	Normal order vector generation (logical OR for ADD/SUB, logical AND for MULT/DIV).
0	1	Reverse logical operation (AND instead of OR for ADD/SUB, OR instead of AND for MULT/DIV).
1	0	Exclusive OR.
1	1	Implication.

A0 Add U; A + B → C

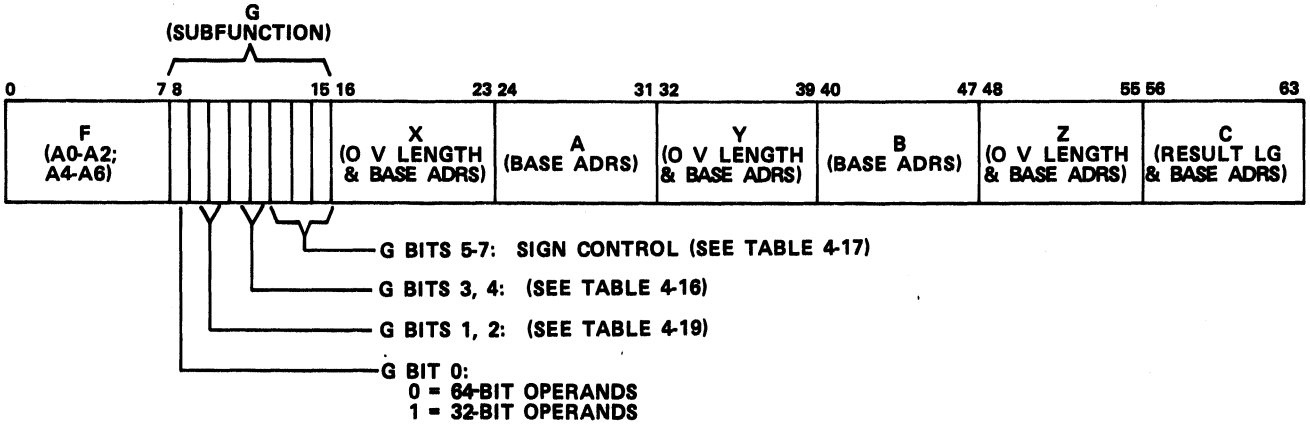
A1 Add L; A + B → C

A2 Add N; A + B → C

A4 Sub U; A - B → C

A5 Sub L; A - B → C

A6 Sub N; A - B → C



These instructions perform the indicated floating-point operations on elements of sparse data vectors A and B. The instructions return the results to elements of sparse data vector C. The instructions read an element from sparse data vector A and/or B when the corresponding sparse order vector X and/or Y contains a one in the associated bit position. A zero in a source order vector causes machine zero to be used as the associated A and/or B element. The instructions generate an element in the C field when a one is in the corresponding bit position of order vector X and/or Y. Each bit position of order vector Z is the bit-by-bit logical function of order vectors X and Y as defined by G bits 1 and 2 in table 4-19. The instruction transfers the resulting field length of sparse vector C to bits 0 through 15 of register C.

In the previously listed sparse vector instructions, U, L, and N denote that upper, lower, and normalized floating-point results are generated, respectively. Applicable data flag bits for the sparse vector instructions are 42 (exponent overflow), 43 (exponent underflow), and 46 (indefinite operand). However, the instructions set the data flag bits only when an element is actually stored in the result vector.

Figures 4-30 and 4-31 show examples of an add U; A + B → C sparse vector instruction operation with assumed register contents and vector address fields for specific values of G bits 1 and 2. Although an A0 instruction is used in the examples, the general execution sequence is the same for all the previous instructions. The dashed lines in figures 4-30 and 4-31 connect the elements of the sparse data vector with the corresponding order vector bits. The results of the logical operations for instructions A0 through A6 are shown in table 4-20.

TABLE 4-20. RESULTS OF THE LOGICAL OPERATIONS (A0 THROUGH A6)

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				
0	0	MZ	MZ	N	N	N	MZ
0	1	MZ	B	+ B	N	+ B	N
1	0	A	MZ	A	N	A	A
1	1	A	B	A + B	A + B	N	A + B

NOTES:

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

†Appendix B describes the normalized floating-point operations.

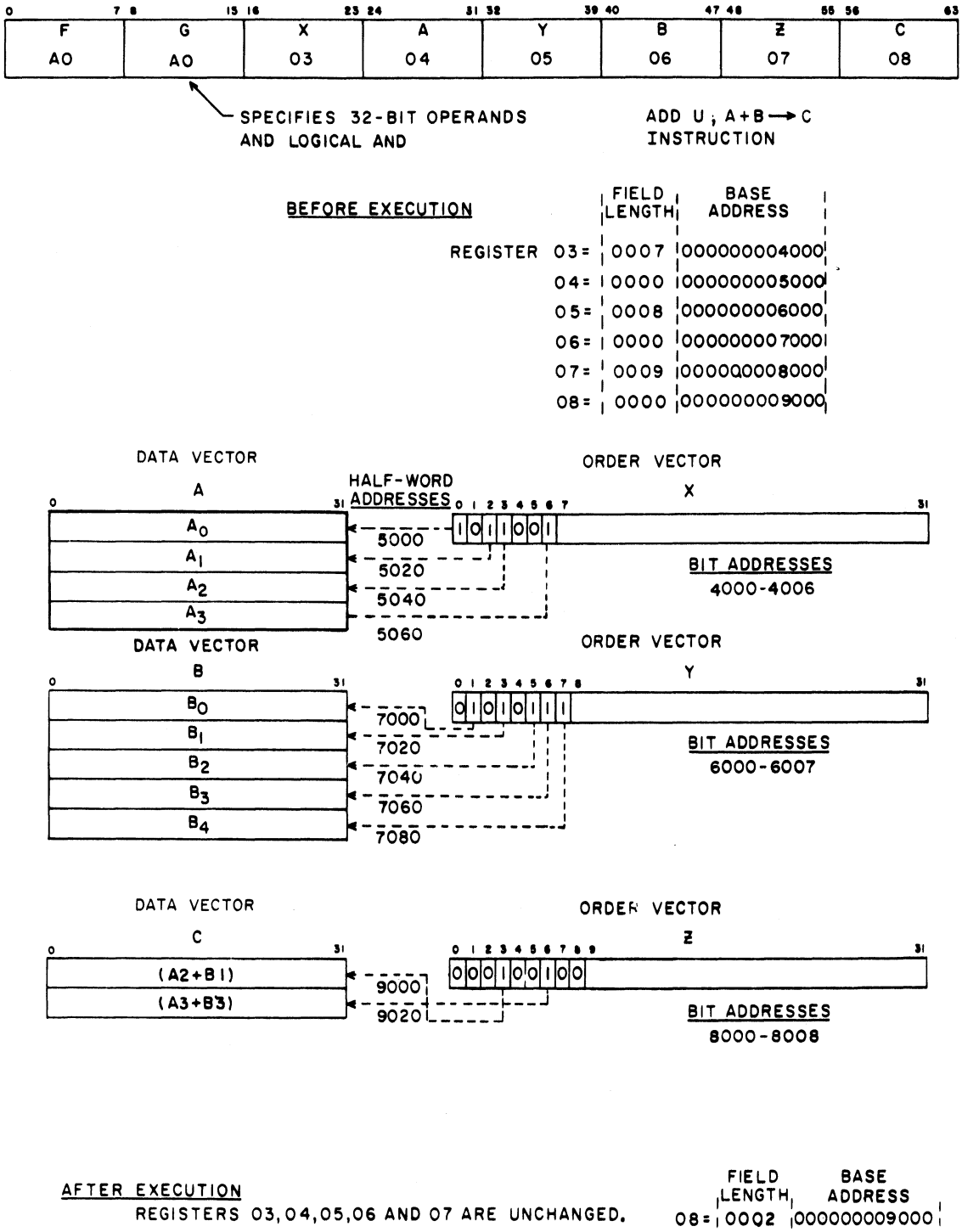


Figure 4-30. Example of an Add U; A + B → C Sparse Vector Instruction when G Bit 1 = 0 and G Bit 2 = 1 (AND)

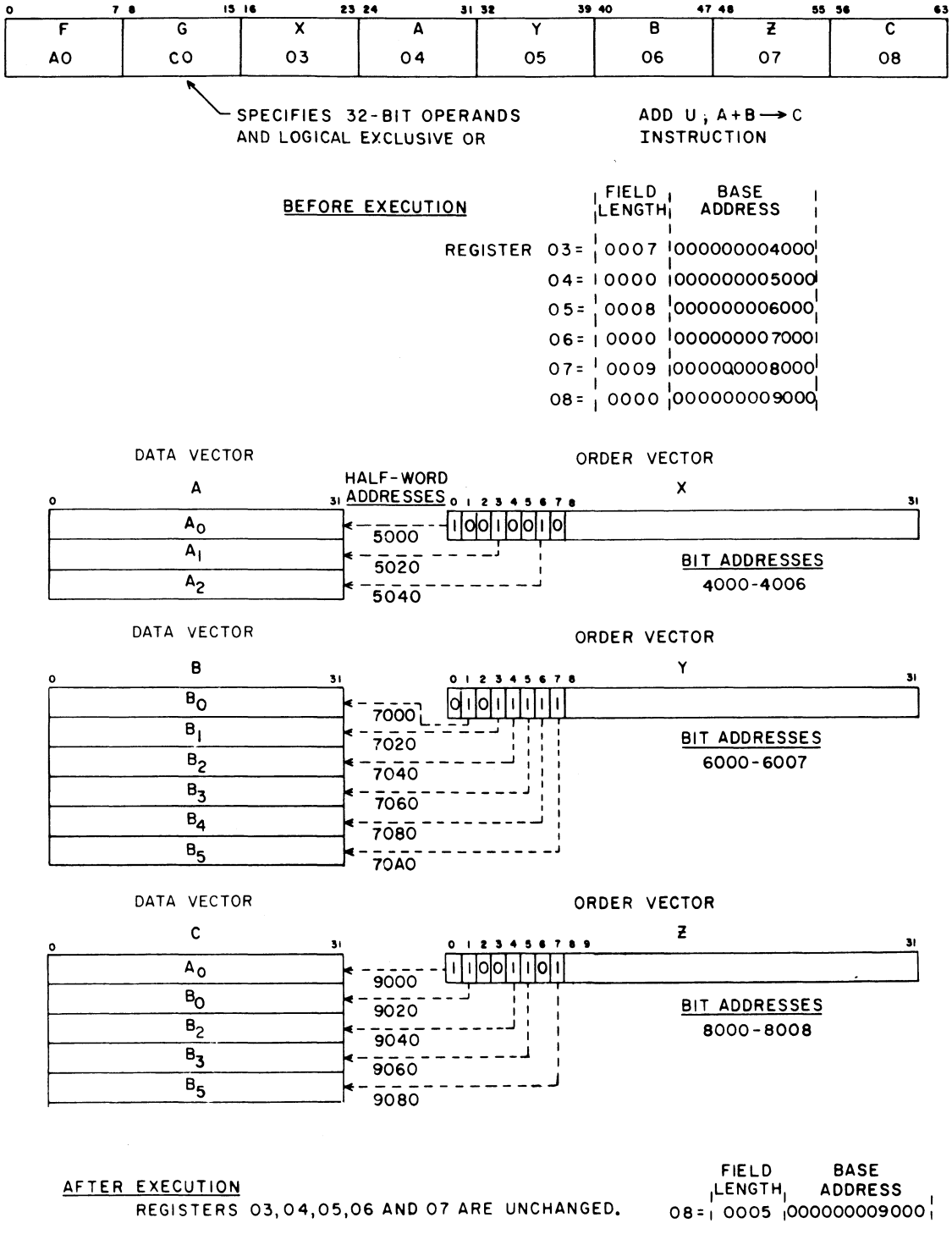


Figure 4-31. Example of an Add U; A + B → C Sparse Vector Instruction when G Bit 1 = 1 and G Bit 2 = 0 (Exclusive OR)

In an A0 instruction operation, an actual addition of an element from data vector A to an element from data vector B takes place only when the corresponding source order vector bits are both ones. For example, the A_3+B_3 addition takes place because bit 3 of X and Y order vectors is a one. In cases where a source order vector bit is a one and the corresponding bit for the other source order vector bit is a zero, machine zero is essentially added to the sparse vector element.

At the end of the sparse vector operation, the resulting output data vector length is inserted in the corresponding portion of the register designated by C. In the example, figure 4-31, the instruction transfers a 0002_{16} to the leftmost 16 bits of register 08. The 0002 denotes the number of elements in the result data vector C.

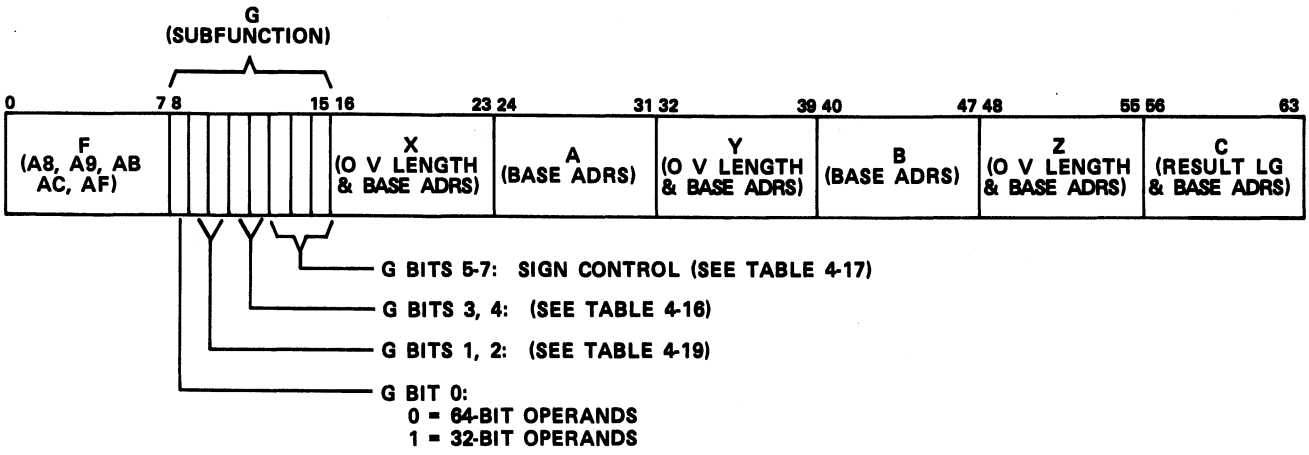
A8 Mpy U; A • B → C

A9 Mpy L; A • B → C

AB Mpy S; A • B → C

AC Div U; A/B → C

AF Div S; A/B → C



These instructions perform the indicated floating-point,† multiply, and divide operations on elements of sparse data vectors A and B. The instructions store the results in elements of sparse data vector C. The instructions read an element from vector A and/or B if the bit position of the corresponding order vector X and/or Y is a one. An element is generated for sparse data vector C according to the entries given in tables 4-21 and 4-22. Result order vector is the bit-by-bit, logical function of order vectors X and Y as defined by G bits 1 and 2 in table 4-19.

†Appendix B describes the floating-point arithmetic operations.

In the sparse vector instructions previously listed, U, L, and S denote that upper, lower, and significant upper floating-point results are generated, respectively. Applicable data flag bits for the multiply and divide sparse vector instructions are 41 (floating-point divide fault), 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). However, the instructions set the data flag bits only when an element is actually stored in the result vector.

Figure 4-32 shows an example of multiply U; A B → C or divide upper A/B → C sparse vector instruction operation with assumed register contents and vector address fields with specific values for G bits 1 and 2. Although an A8 instruction is used, the general execution sequence is the same for all instructions of this type. Dashed lines connect the elements of the sparse data vector with the corresponding order vector bits.

In an A8 operation, an actual product is generated as an element of data vector C only when the corresponding order vector bits for the A and B data elements are both ones. In cases where one or both of the source order vector bits are zero, no multiplication takes place, and the corresponding result order vector bit is cleared. In figure 4-32, only three products are generated by the instruction $(A_1 \bullet B_1)$, $(A_2 \bullet B_4)$, and $(A_3 \bullet B_5)$.

At the end of the sparse vector operations, the resulting output data vector length is inserted in the corresponding portion on the register designated by C. In the example, the instruction transfers a 0007 to the leftmost 16 bits of register 09. The 0007 denotes the number of elements in result data vector C. The results of the logical operations for instructions A8 through AF are shown in tables 4-21 and 4-22.

TABLE 4-21. RESULTS OF THE LOGICAL OPERATIONS (A8 THROUGH AB)

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				
0	0	NO	NO	N	N	N	NO * NO
0	1	NO	B	NO * B	N	NO * B	N
1	0	A	NO	A * NO	N	A * NO	NO * NO
1	1	A	B	A*B	A*B	N	A * B

NOTES:

A A stream operand
B B stream operand
N No result produced
NO Normalized one

TABLE 4-22. RESULTS OF THE LOGICAL OPERATIONS (AC, AF)

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				
0	0	NO	NO	N	N	N	NO / NO
0	1	NO	B	NO / B	N	NO / B	N
1	0	A	NO	A / NO	N	A / NO	NO / NO
1	1	A	B	A/B	A/B	N	A / B

NOTES:

A A stream operand
B B stream operand
N No result produced
NO Normalized one
IND Indefinite

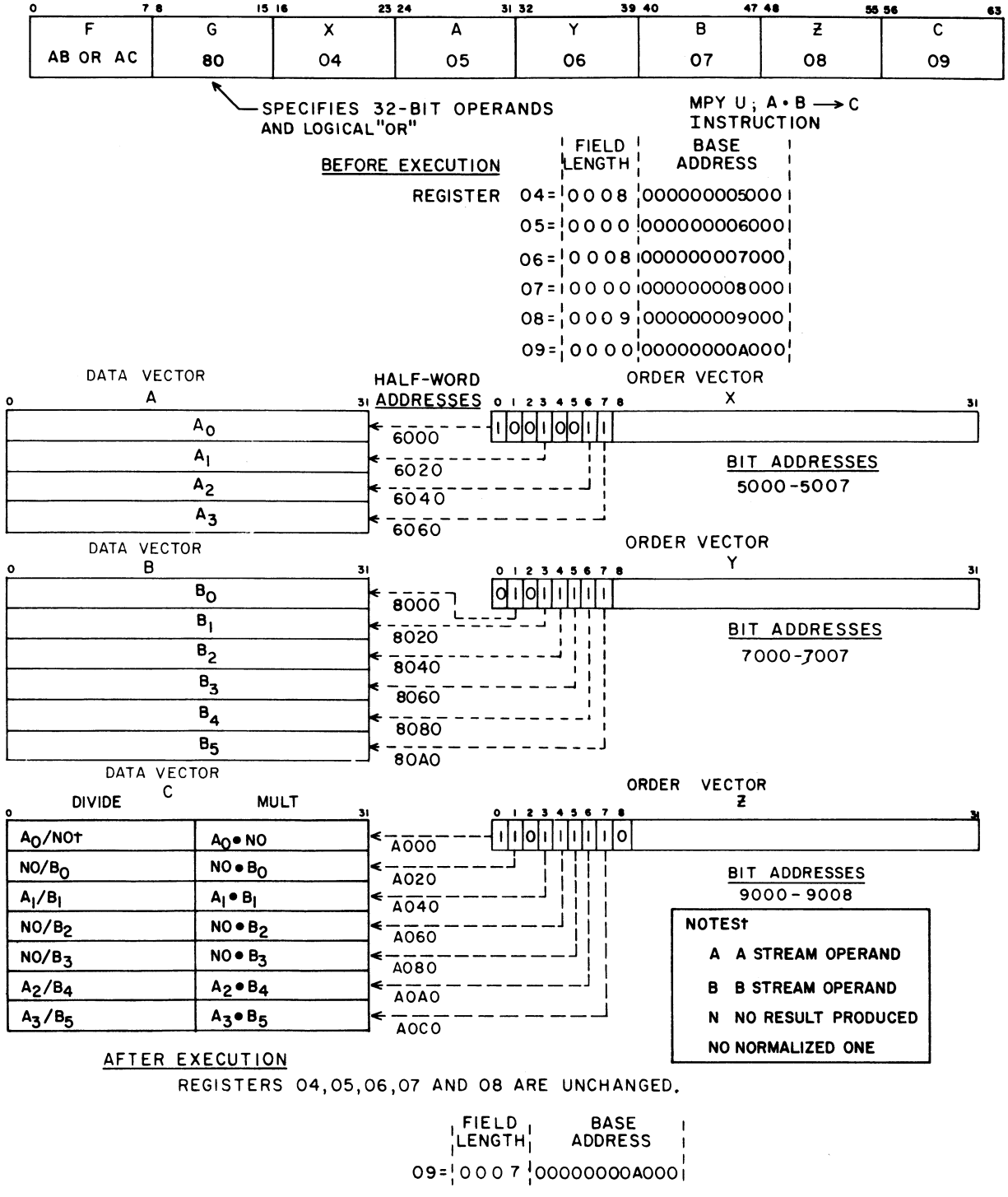


Figure 4-32. Example of a Div or Mpy U Sparse Vector Instruction when G Bit 1 = 0 and G Bit 2 = 0 (OR)

VECTOR MACRO INSTRUCTIONS

Vector macro instructions perform operations similar to vector instructions. However, some vector macro instructions do not form result vector fields, but store the results in one or two registers which are specified by the instruction. In these instructions, the control vector contains neither length nor offset, but controls the use of elements of the source vectors. Bit 2 of the G designator is undefined and must be a zero. Designators C and C + 1 denote 32-bit registers when bit 0 of the G designator† specifies 32-bit operands. In the vector macro instructions that produce result vector fields, the control vector performs the same function as in the vector instructions.

Vector macro instructions with result fields (as opposed to result registers) extend short source fields with machine zeros or normalized ones and terminate in an identical fashion to the vector instructions. The other vector macro instructions do not extend short source vectors but terminate when either source vector is exhausted. For instructions of this type, broadcasting both source fields causes an undefined condition to exist. Appendix C gives a complete listing of the various field conditions and the resulting termination condition.

C0 Select EQ; A = B, Item Count to (C)

C1 Select NE; A ≠ B, Item Count to (C)

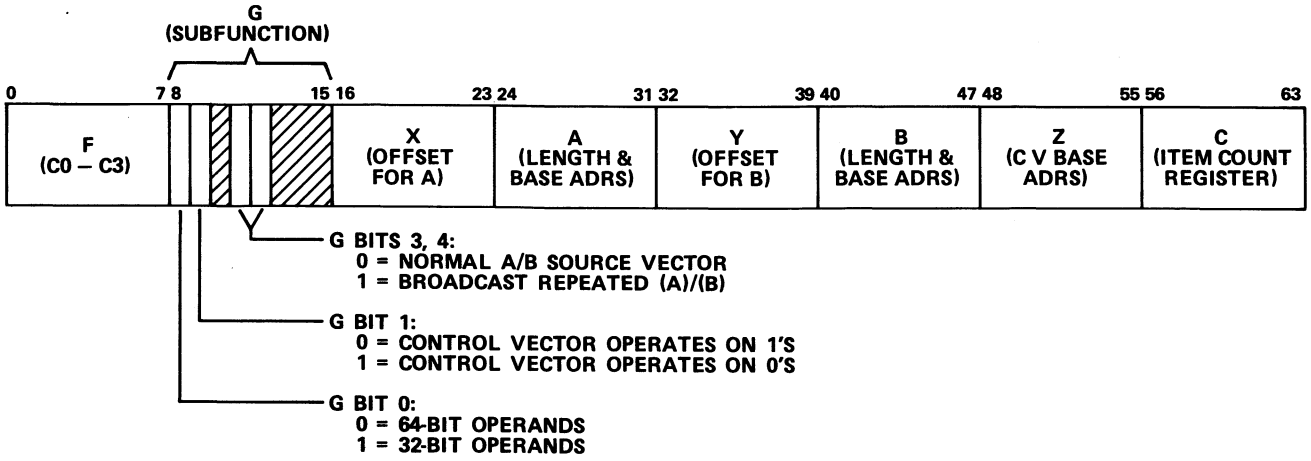
C2 Select GE; A ≥ B, Item Count to (C)

C3 Select LT; A < B, Item Count to (C)

These instructions compare each element of vector field A with its corresponding element of vector field B by subtracting vector B from vector A. The conditions for comparing floating-point operands are described in the Floating-Point Compare Rules, appendix B. The comparing operation proceeds until the compare condition is met (for a pair of elements not inhibited by the corresponding bit of the control vector) or the shorter of the two vector fields is exhausted. If broadcast is selected for field A or B (but not both), the instruction will terminate when the nonbroadcast field terminates.

†Appendix C provides a comprehensive listing of the G designator bits usage according to function code.

G bits 2, 5, 6, and 7 are undefined and must be set to zeros.



If the compare condition is met, the item count equals the number of pairs of elements encountered up to (but not including) the pair meeting the specified condition, including the pairs inhibited by the control vector. If the compare condition is not met, the item count equals the length of the shorter vector after the offset adjustment. The instruction stores the item count into the rightmost 48 bits of a cleared register C.†

The control vector, if used, determines which pairs of elements are compared. For example, if G designator bit 1 equals zero, a 1 bit in the control vector enables the comparison of the corresponding pair of source elements. A zero bit in a control vector disables the comparison of the corresponding pair of source elements. The item count, as previously described, includes all pairs of elements encountered, including the pairs for which the comparison was inhibited. If a control vector is used and either source vector A or B is exhausted before a permissive control vector bit is encountered, the instruction makes no comparisons. In this case, the item count represents the length of the shorter vector field minus the offset. Applicable data flag bits are 37 (select condition not met) and 46 (indefinite result).

Figure 4-33 shows an example of a select EQ; A=B; item count → C(C0) instruction with assumed instruction codes, register contents, and vector fields. The G designator specifies 32-bit operands and broadcast source vector A_0. Since the B offset equals 3, the first comparison takes place between source element B_3 and broadcast vector A_0; this comparison is not met. Element B_5 satisfies the comparison condition, but the zero in bit 5 of the control vector disables the comparison. Element B_6 satisfies the comparison condition, and the control vector enables the comparison. Thus, the item count of three is transmitted to the rightmost 48 bits of register 0A. The item count includes the B_5 comparison although the control vector disabled this comparison.

†If the C designator is zero, this instruction produces undefined results.

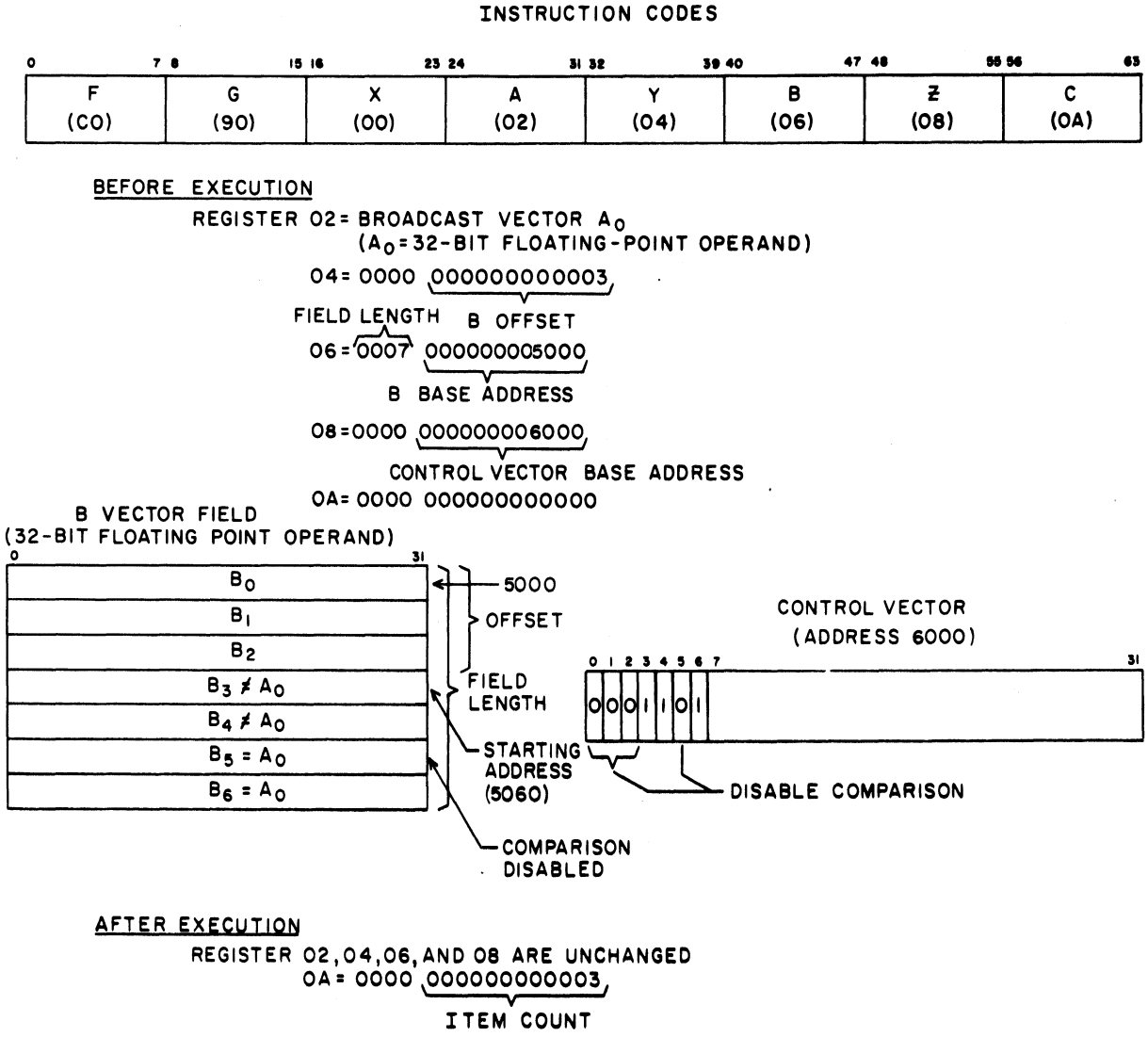
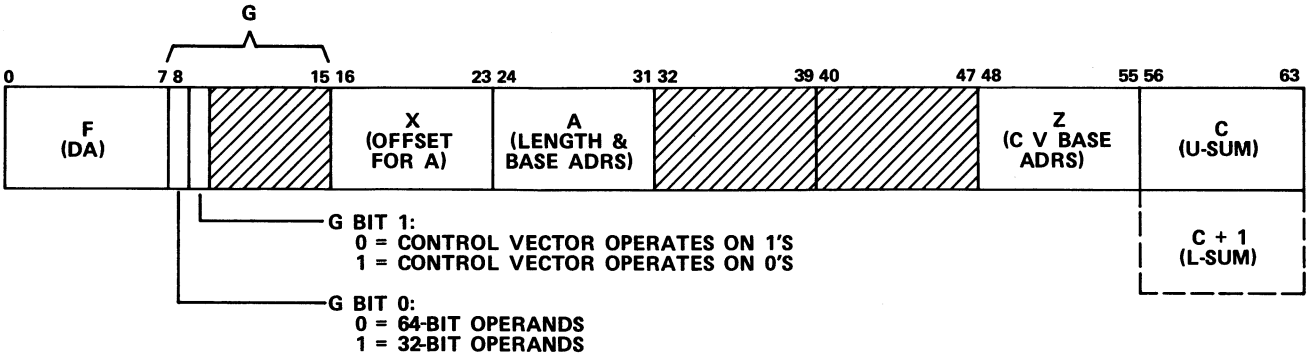


Figure 4-33. Example of Select EQ; A=B, Item Count to C

DA Sum ($A_0 + A_1 + A_2 + \ldots A_n$) to (C) and (C + 1)



NOTE: U DENOTES THE UPPER RESULT.
L DENOTES THE LOWER RESULT.

This instruction forms the double-precision, unnormalized, floating-point sum † of all the elements of vector field A. The instruction is executed in the following manner:

$A_0 + A_8 + A_{16} + \ldots = \text{sum } X_0$ $\quad$ $A_4 + A_{12} + A_{20} + \ldots = \text{sum } X_4$

$A_1 + A_9 + A_{17} + \ldots = \text{sum } X_1$ $\quad$ $A_5 + A_{13} + A_{21} + \ldots = \text{sum } X_5$

$A_2 + A_{10} + A_{18} + \ldots = \text{sum } X_2$ $\quad$ $A_6 + A_{14} + A_{22} + \ldots = \text{sum } X_6$

$A_3 + A_{11} + A_{19} + \ldots = \text{sum } X_3$ $\quad$ $A_7 + A_{15} + A_{23} + \ldots = \text{sum } X_7$

Where A_0, A_1, A_2, . . . are elements of vector A.

Sums X_0 through X_7 (all double precision quantities) are then added together as follows:

$X_6 + X_2 = Y_0$ $\quad$ $X_0 + X_4 = Y_2$

$X_7 + X_3 = Y_1$ $\quad$ $X_1 + X_5 = Y_3$

These precision results are then further added as follows:

$Y_0 + Y_2 = Z_0$ $\quad$ $Y_1 + Y_3 = Z_1$

A final double precision add to Z_0 and Z_1 forms the final sum C, C + 1.

The instruction transmits the upper result portion of the sum to the register specified by C and the lower result to the register designated by C + 1.

Registers C and C + 1 are either 32- or 64-bit registers, depending on the state of G bit 0 in the instruction. Register C must be even; if register C is odd or zero, the instruction results are undefined.

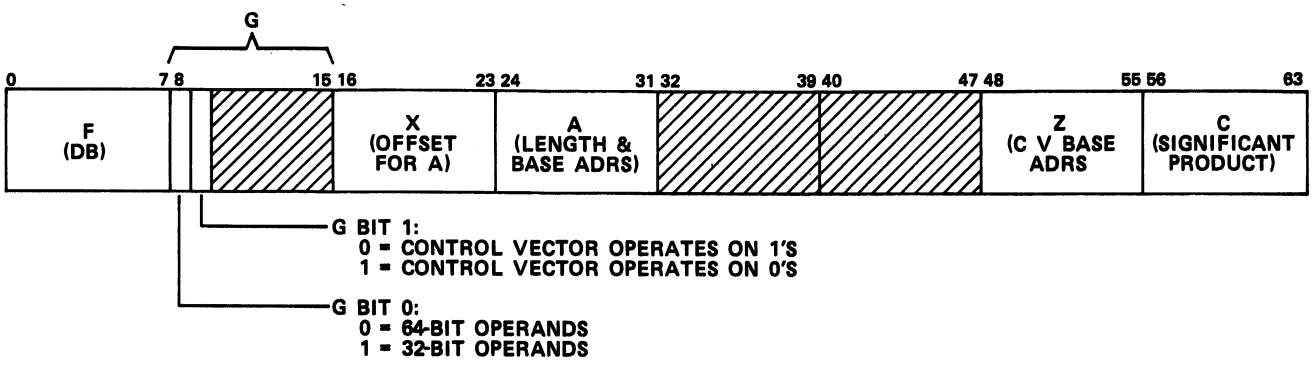
The Y and B designators (bits 32 through 47) and bits 2 through 7 of the G designator are not used and must be zeros. There is no length or offset specification for control vector Z. The instruction terminates when the source vector field A is exhausted. If the control vector allows no elements to be summed, the instruction sets the result to machine zero.

† Appendix B describes the double-precision addition of floating-point operands and order-dependent result considerations.

If a control vector (CV) is specified and contains no permissive elements, the result is machine zero. The instruction does not specify control vector length or offset.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). Data flag bits 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 bit 43 is set if the exponent of the lower result is less than 9000_{16} for 64-bit mode and 90_{16} for 32-bit mode. In this case, the exponent of the upper result may be greater than 9000_{16} and will be stored as is and will not be forced to machine zero. The instruction sets flag bit 42 if any of the add operations overflow.

DB Product (A_0, A_1, A_2, . . . A_n) to C



This instruction forms the significant product† of successive, floating-point elements in source field A. The instruction is executed in the following manner.

$A_0 \bullet A_7 \bullet A_{14} \bullet \ldots$ = product X_0

$A_1 \bullet A_8 \bullet A_{15} \bullet \ldots$ = product X_1

$A_2 \bullet A_9 \bullet A_{16} \bullet \ldots$ = product X_2

$A_3 \bullet A_{10} \bullet A_{17} \bullet \ldots$ = product X_3

$A_4 \bullet A_{11} \bullet A_{18} \bullet \ldots$ = product X_4

$A_5 \bullet A_{12} \bullet A_{19} \bullet \ldots$ = product X_5

$A_6 \bullet A_{13} \bullet A_{20} \bullet \ldots$ = product X_6

Where A_0, A_1, A_2, . . . are the elements of vector A.

Products X_0 through X_6 are then multiplied together as follows:

$X_5 \bullet X_2 = Y_0$

$X_6 \bullet X_3 = Y_1$

$X_0 \bullet X_4 = Y_2$

The results are further multiplied along with Xl as follows:

$X_1 \bullet Y_1 = Z_0$

$Y_0 \bullet Y_2 = Z_1$

†Appendix B describes the floating-point multiplication operation and order-dependent result considerations.

A final multiply produces the result C. Register C is either a 32- or 64-bit register, depending on whether 32- or 64-bit operands are used, respectively.

In the execution of the DB instruction, the following result differences may occur. The central computer may multiply the partial products (X and Y) by a normalized one (EA40 0000 in 32-bit mode or FFD2 4000 0000 0000 in 64-bit mode) an indeterminate number of times, depending on discontinuities in the input data stream. If the coefficients of the partial products are nonzero, the partial products are unchanged by the additional multiply. However, if the coefficient is all zeros, EA or FFD2 is added to the exponent. This is normal under the definition of significant multiply. If the interruptions last long enough, the exponent may decrease to machine zero, setting data flag 43.

Input Stream			Partial Products		
00FF	FFFF		1800	0000	1st
0080	0000				
Interruption occurs here ⟶		(First normalized one)	0200	0000	2nd
			EC00	0000	3rd
			D600	0000	4th
			C000	0000	5th
			AA00	0000	6th
			9400	0000	7th
			8E00	0000	8th

All of the above products are equal under the floating-point compare rules. The last product, however, sets data flag 43 and 46. Data flag 42 sets if any multiply operation overflows.

These discontinuities may be caused by hardware-generated gaps in the input data or by machine interrupts.

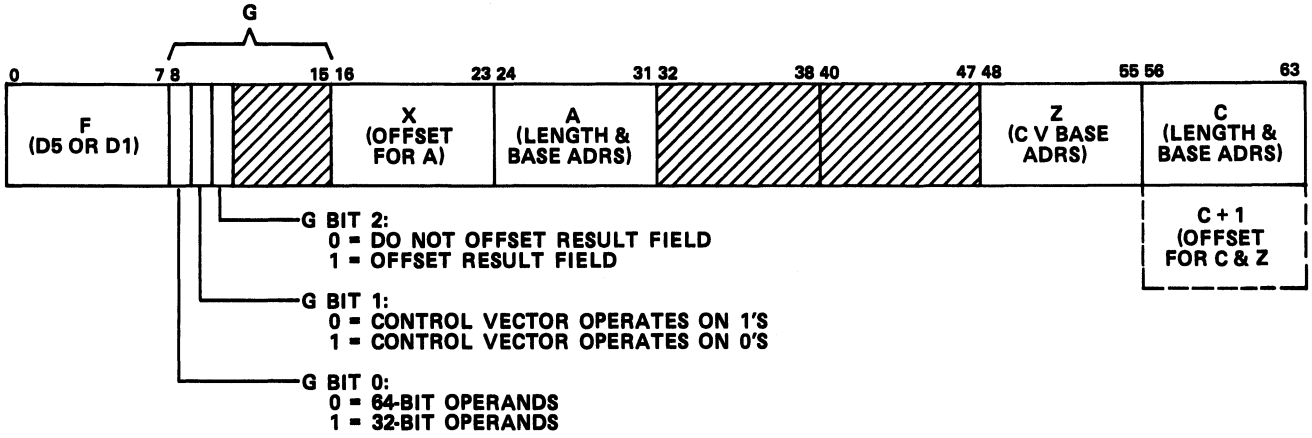
The Y and B designators and bits 2 through 7 of the G designator are undefined and must be zeros. Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

If bit 1 of the G designator is zero, for example, a zero bit in the control vector disables the multiplication of the corresponding source element and the partial product. Thus, the multiplication of a source element and the partial product takes place only when the corresponding control vector bit is enabled. This instruction contains no length specification for the control vector. The instruction terminates when the A field vector is exhausted. If the control vector contains no enabling elements, the result is a normalized one. If the C designator is zero, the results are undefined.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

D5 Delta $[A_{(n+1)}-A_n] \rightarrow C_n$

D1 Adj. Mean $[A_{(n+1)}+A_n] / 2 \rightarrow C_n$



D5 Delta $(A_n + 1 - A_n) \rightarrow C_n$

This instruction forms the nth element of result vector field C by subtracting the nth element of source field A from the n+1th element of A. Normalized, floating-point arithmetic is used in the subtraction. Figure 4-34 shows an example of a delta instruction with assumed instruction codes, operands, and register contents.

The example shows that since there is no offset of the A vector, the first subtraction consists of $A_1 - A_0$ which produces result element C_0. The subtraction of the A vector elements continues in this manner until element C_4 is reached. The corresponding Z control vector bit is a zero which prohibits the storing of the result element C_4 and leaves the C_4 result field location unchanged.

Since the source field is one element shorter than the result field, C_5 becomes minus A_5 and C_6 becomes zero. The delta (D5) instruction terminates when the result field is exhausted.

The Y and B designators and bits 3 through 7 of the G designator are unused and must be zeros.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result).

INSTRUCTION CODE

0 7	8 15	16 23	24 31	32 39	40 47	48 55	56 63
F (D5)	G (80)	X (00)	A (02)	Y (00)	B (00)	Z (03)	C (04)

A VECTOR SOURCE FIELD

0 31	ADDRESS
A_0	6000
A_1	6020
A_2	6040
A_3	6060
A_4	6080
A_5	60A0

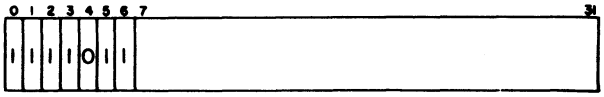
REGISTER CONTENTS

02=0006 000000006000
03=0000 000000007000
04=0007 000000008000

C VECTOR RESULT FIELD

0 31	ADDRESS
$C_0 (A_1 - A_0)$	8000
$C_1 (A_2 - A_1)$	8020
$C_2 (A_3 - A_2)$	8040
$C_3 (A_4 - A_3)$	8060
C_4 NO CHANGE	8080
$C_5 (0 - A_5)$	80A0
$C_6 (0)$	80C0

Z CONTROL VECTOR



ADDRESS
7000

NOTE: VALUES IN PARENTHESES INDICATE A VECTOR ELEMENTS SUBTRACTED FOR CORRESPONDING C VECTOR ELEMENT.

Figure 4-34. Example of Delta Instruction

D1 Adj. Mean $(A_n + 1 + A_n)/2 \rightarrow C_n$

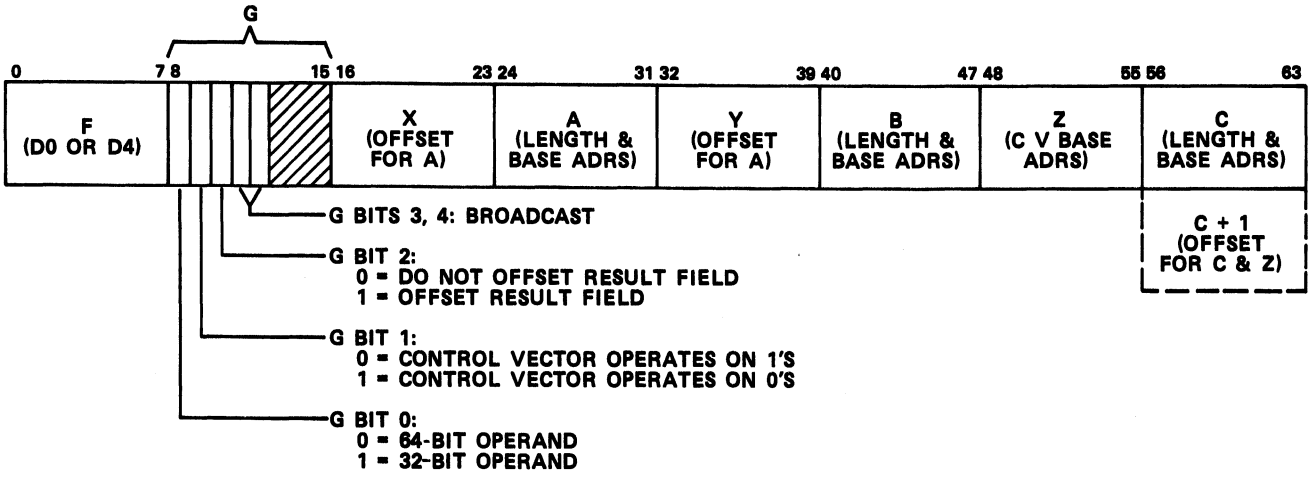
This instruction forms the nth element of result vector field C by the normalized addition of the nth and n+1th elements of source field A. The instruction then divides the result element by two, producing the mean of the two source elements. The mean result is stored as the corresponding result element in vector C. All operands and arithmetic operations are expressed in floating point.

The division by two is accomplished by subtracting one from the exponent of the result element.

The Y and B designators and bits 3 through 7 of the G designator are not used and must be zeros.

Applicable data flag bits are 43 (result machine zero) and 46 (indefinite result).

D0 Average $(A_n + B_n) / 2 \rightarrow C_n$
D4 Ave. Diff. $(A_n - B_n) / 2 \rightarrow C_n$



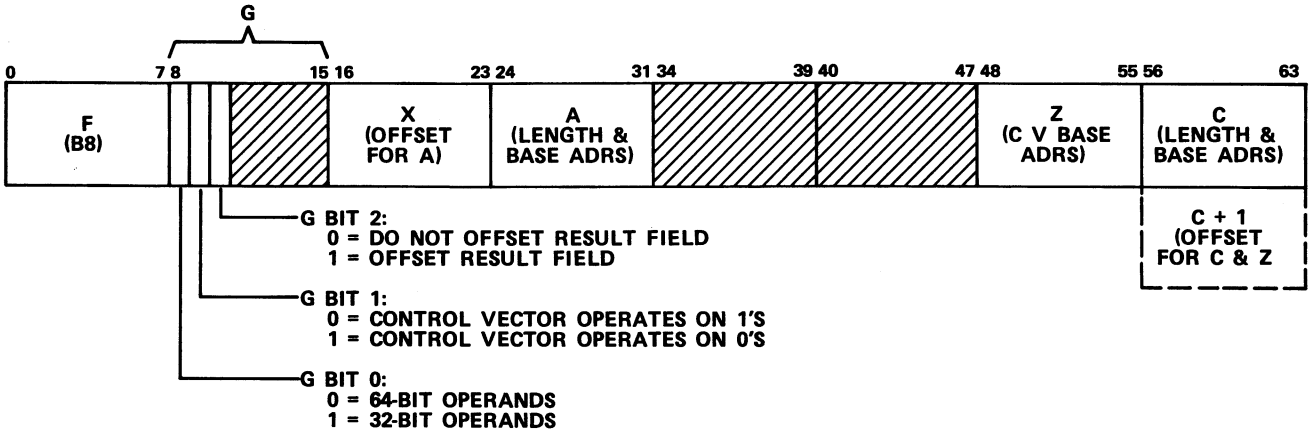
These two instructions form the normalized average and normalized average difference, respectively, of elements A_n and B_n in the A and B vector fields. The sum (D0) or difference (D4) of elements A(n) and B(n) is divided by two. The result elements become corresponding elements of result vector field C. The division by two is accomplished by subtracting one from the exponent.

In all other respects, these instructions function the same as the normal vector instructions described under Vector Instructions in this section. Thus, short source fields are extended with machine zeros. These instructions terminate when the result field is exhausted.

G bits 5 through 7 are undefined and must be set to zeros.

Applicable data flag bits are 43 (result machine zero) and 46 (indefinite result).

B8 Transmit Reverse A → C



This instruction transmits the elements of vector source field A to vector result field C. The elements are transmitted in reverse order from A to C. Thus, the last element of vector A becomes the first element of vector C, the next to the last element of vector A becomes the second element of vector C, and so forth.

This instruction terminates when the result field is exhausted. A short source field is extended with machine zero elements. If the source and result fields overlap in storage, the results of the instruction are undefined.

The Y and B designators and bits 3 through 7 of the G designator are undefined and must be zeros. This instruction sets no data flag bits.

Figure 4-35 shows an example of the operation of a transmit reverse A → C instruction with assumed instruction codes, addresses, field lengths, and vector fields.

Since the offsets for the A and C vector fields are equal (+3), the first operation transmits element A_7 to C_3. The operations continue in this manner until bit 5 of the control vector is reached. Since bit 5 = zero, the transmission of A_5 to C_5 is disabled, and C_5 remains unaltered.

The last three elements of vector field C (C_8, C_9, and C_A) are set to machine zero since the result field length is three elements longer than the source length. The dashed lines show the order of transfer of elements from the A vector source field to the C vector result field.

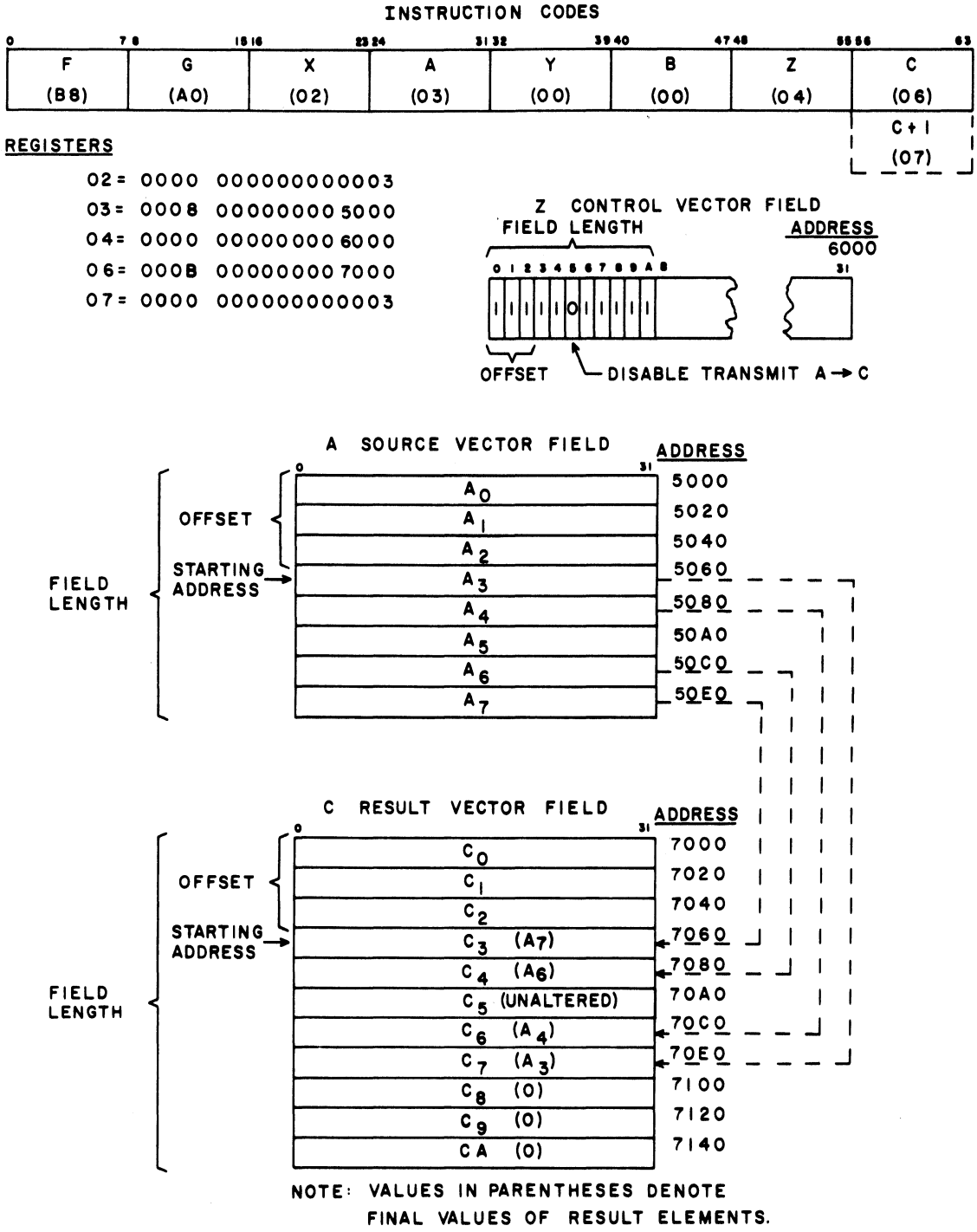
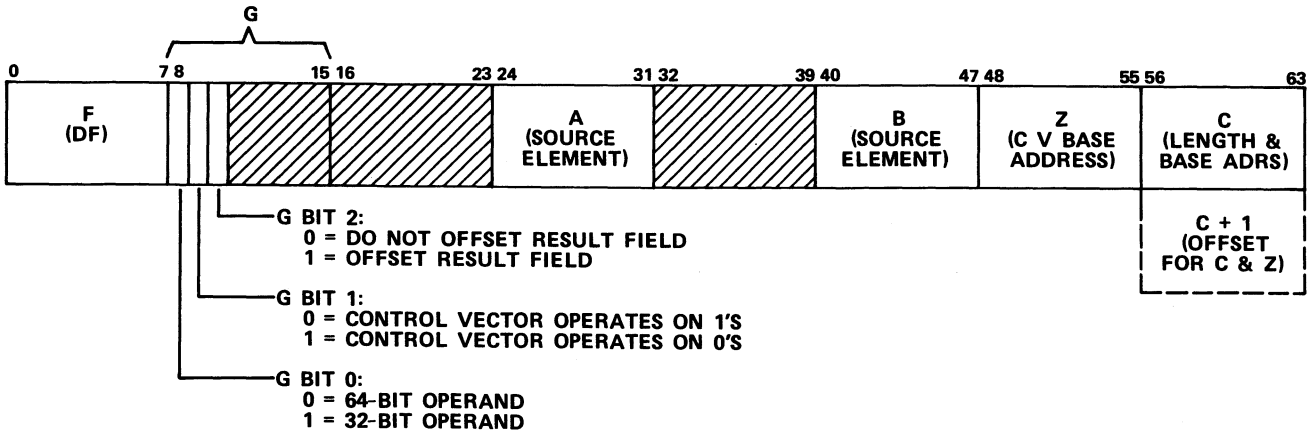


Figure 4-35. Example of Transmit Reverse A → C Instruction

DF Interval A Per B→C



This instruction forms a result vector C whose initial element is equal to the constant from register A and whose succeeding elements are greater than the preceding element of vector C by the constant in register B. If the exponent of the constant in register A is equal to or greater than the exponent of the constant in register B, the initial element of the result vector is identical to the constant in register A. If the exponent of the constant in register A is less than the exponent of the constant in register B, the initial element of the result vector is the constant from register A adjusted to have the same exponent as the constant in register B. The second element of vector C equals the first element plus the content of register B. The third element of vector C equals the second element plus the content of register B, and so forth. The instruction uses unnormalized, floating-point addition.† Thus, the first element of C_0 = B and the succeeding elements are $C_n = C_{n-1}+B$.

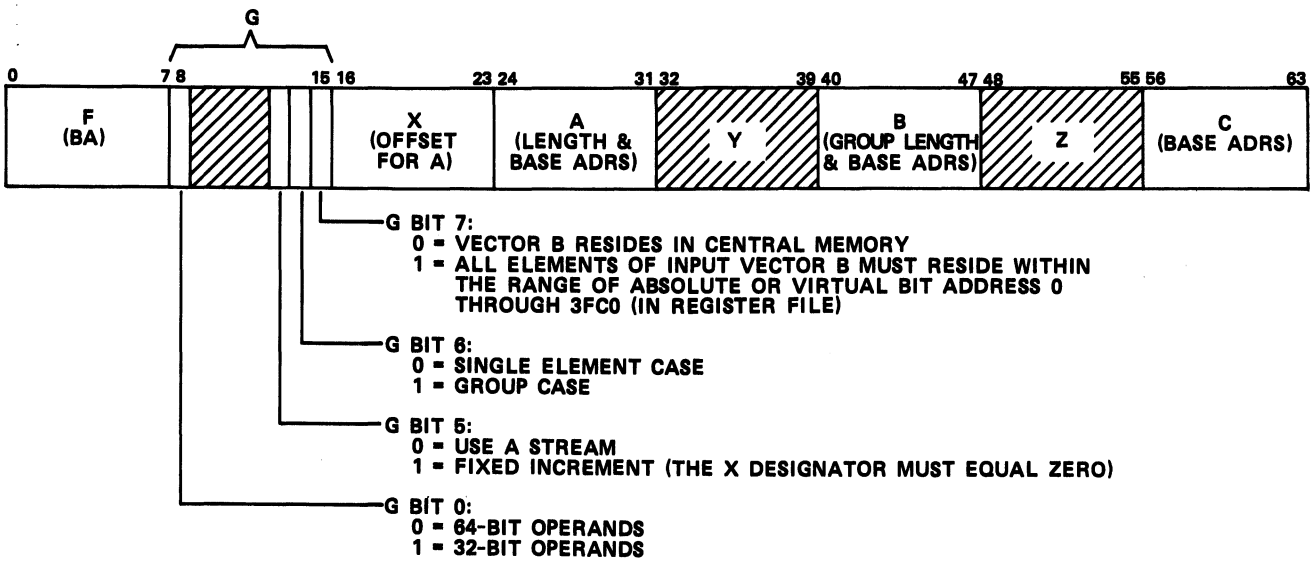
If the instruction uses a control vector, an element is generated for each control bit of the field length, although it may not be stored in the result field. If the instructon detects a nonpermissive bit in the control vector, the addition operation is performed, but the result element is not stored in the result field. If the control vector diables the storing of a result element and this element is indefinite, data flag bit 46 (indefinite result) is not set until a permissive bit is detected in the control vector. Similarly, data flag bit 42 (exponent overflow) or 43 (result machine zero) is set on the next permitted store although the iterative step which overflowed was not stored.

The X and Y designators and bits 3 through 7 of the G designator are not used and must be zeros.

If the A designator is zero, this is treated as a broadcast register and 8000---0 is read from register zero.

†Appendix B describes floating-point arithmetic and order-dependent result considerations.

BA Transmit Indexed List→C



This instruction forms an indexed list of result elements in vector field C by transferring elements from addresses in vector field B as indexed by the item counts in the A-vector field. The rightmost 48 bits (no half-word option) of each element of vector A contains an item count. The instruction adds the first item count in vector A to the base address of vector B. The element at that address is transferred to result vector C. Before the addition of the item count (index) to the base address, the index is left-shifted five places (32-bit operands) or six places (64-bit operands) to form the half-word or full-word address, respectively.

The instruction then adds the next element of vector A to the base address of vector B. The resulting address indexes the second element of vector B. This process continues until vector A is exhausted. Vector C is contiguous for this instruction.

The elements of vector A are always 64-bit operands, while G bit 0 specifies the B and C vector element size.

The Y and Z designators and bits 1 through 4 of G designator are undefined and must be set to zero.

When G bit 5=1 vector A is replaced by a fixed increment specified by the rightmost 48 bits of register A. The X designator must equal zero. The addressing of vector B is then; B,B+A,B+2A,. . . . ,B+(N-1)A where N is the field length specified by the leftmost 16 bits of register A and still determines the total number of groups. The fixed increment A is shifted left 6 (G bit 0=0) or 5 (G bit 0=1) places before it is added to B.

If G bit 6 is a zero, the instruction transmits single elements as previously described. If G bit 6 is set, a group of elements is transmitted from vector B to vector C for each element of vector A. The group length is specified in the upper 16 bits of register B. All groups are of equal length. If the leftmost bits of register B are zero, this instruction is a no-op.

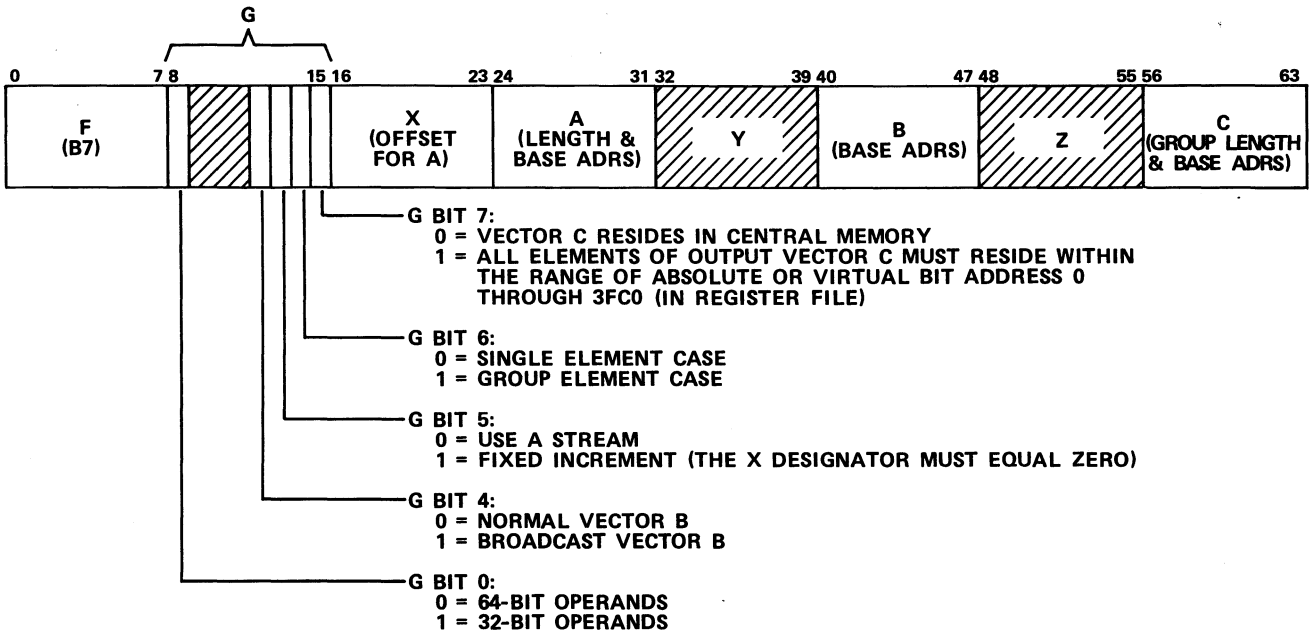
If G bit 7 is set, all elements of input vector B must reside in the register file within the range of absolute or virtual bit addresses 0 through 3FC0. Reference to the register file as central memory is normally not allowed. This instruction and the B7 instruction are the only instructions which permit this type of reference to occur. Refer to section 5 for other register file restrictions. If all the addresses for vector B are not contained in the register file, this instruction is undefined. This instruction is also undefined if G bits 6 and 7 are both set.

The search: index list→C (C8 through CB) instructions may be used to produce the index list for the BA instruction.

Figure 4-36 shows an example of a transmit indexed list→C instruction with assumed instruction codes, register content, and vector fields. The first item count is read from address 4000. This value indexes the B vector base address by five half-words after the left shift of five. Thus, the instruction transfers the first B vector element from address 70A0 to the C vector element address 9000. Six B vector elements are transferred to the C vector.

No data flag bits are set by the BA instruction.

B7 Transmit List →Indexed C



NOTE: THE C+1 DESIGNATOR IS NOT USED

This instruction adds the rightmost 48 bits of the first element of vector field A (no half-word option) to the base address in register C to form the address of the first element of result vector field C. The instruction then transmits the first element of vector field B to the computed address in C. The rightmost 48 bits of each element of vector field A is an item count. Before the addition of the item count (index) to the base address, the index is left-shifted five places (32-bit operands) or six places (64-bit operands) to form the half-word or full-word address, respectively.

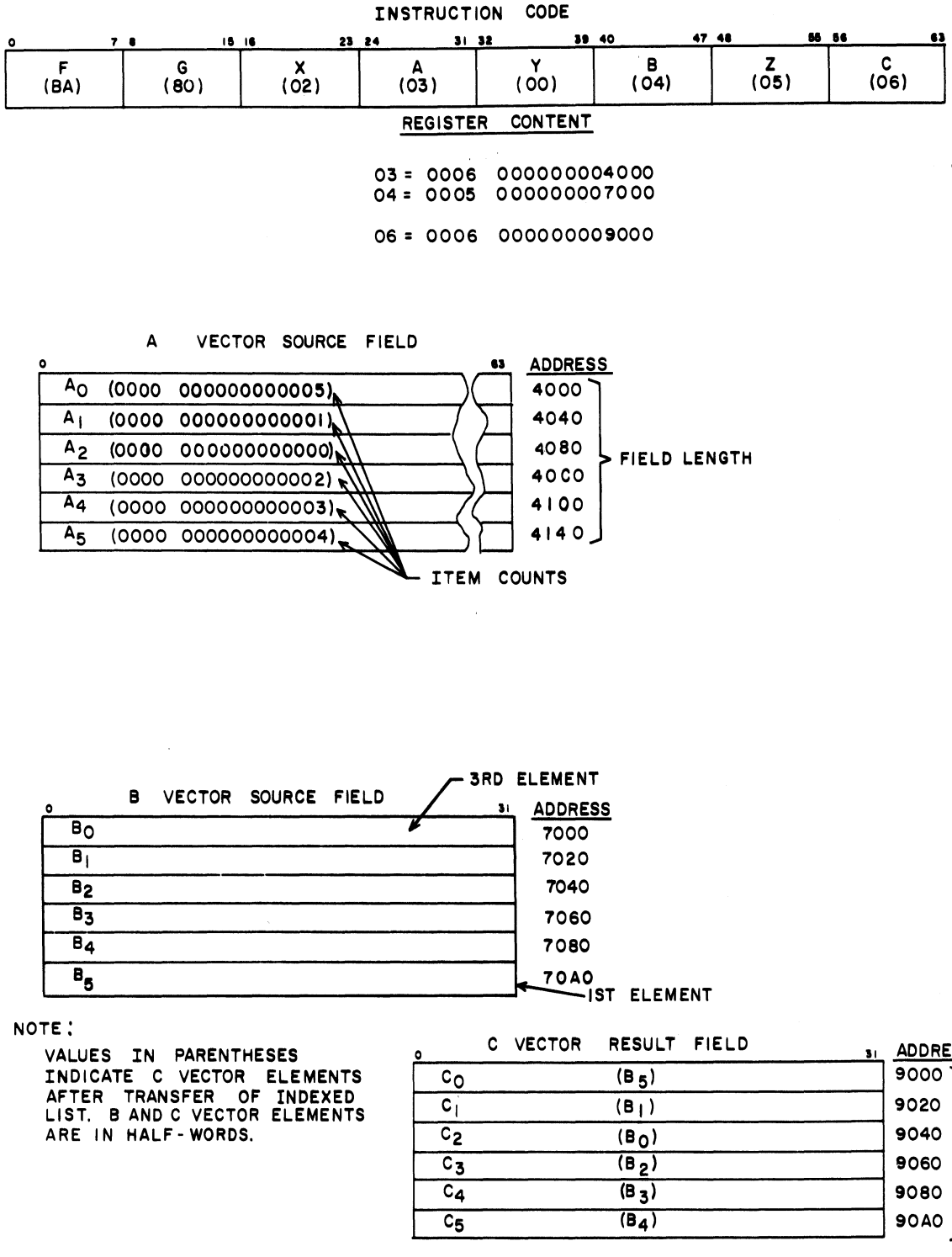


Figure 4-36. Example of Transmit Indexed List → C Instruction

Similarly, the instruction forms the address of the second element of vector field C by adding the second element of vector field A to the base address in register C. The second element of vector field B is then transmitted to the computed address in the result vector field C. The instruction continues in this manner until the A vector field length is exhausted. Vector B is contiguous for this instruction.

The Y and Z designators are undefined and must be zeros. The elements of vector field A are 64 bits while the elements of vectors B and C are 64 bits or 32 bits as specified by G designator bit 0.

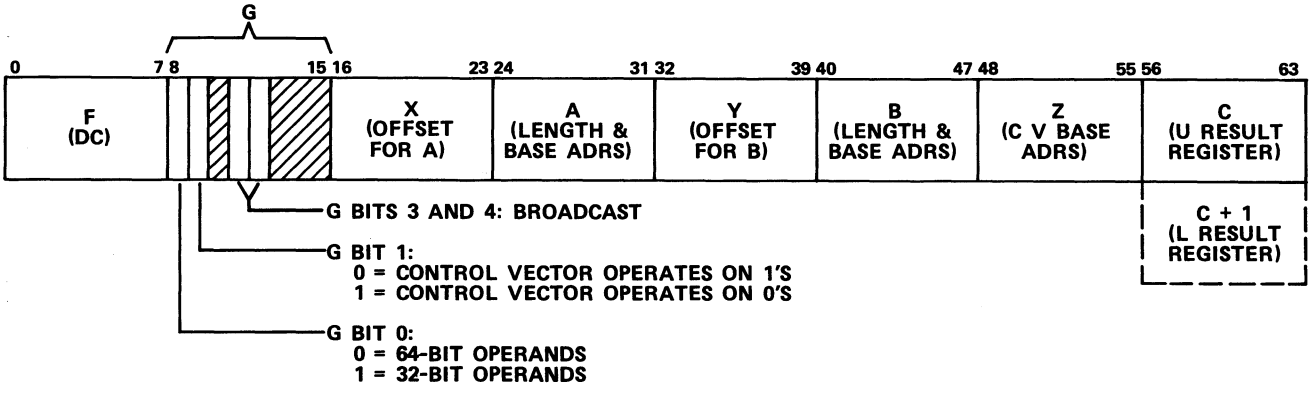
Bits 1, 2, 3, of the G designator are not used and must be set to zero. Vector B is broadcast when bit 4 of the G designator is set and bit 6 is a zero.

When G bit 5=1, vector A is replaced by a fixed increment specified by the rightmost 48 bits of register A. The X designator must equal zero. The addressing of vector C is then: C,C+A,C+2A,....,(C+(N-1)A where N is the field length specified by the leftmost 16 bits of register A and still determines the total number of groups or elements. The fixed increment A is shifted left 6 (G bit 0=0) or 5 (G bit 0=1) places before it is added to B.

If G bit 6 is a zero, the instruction transmits single elements as previously described. If G bit 6 is set, a group of elements is transmitted from vector B to vector C for each element of vector A. The group length is specified in the upper 16 bits of register C. All groups are of equal length. If the sixteen leftmost bits of register C are zero, the instructon is a no-op.

If G bit 7 is set, all elements of output vector C must reside in the register file, within the range of absolute or virtual bit addresses 0 through 3FC0. Reference to the register file as central memory is normally not allowed. This instruction and the BA instruction are the only instructions which permit this type of reference. Refer to section 5 for other register file restrictions. If all the addresses for vector C are not contained in the register file, the instruction is undefined. This instruction is also undefined if either G bits 4 and 6 or G bits 6 and 7 are set.

DC Vector Dot Product to (C) and (C + 1)



NOTE: U DENOTES THE UPPER RESULT.
L DENOTES THE LOWER RESULT.

This instruction multiplies corresponding elements of vector fields A and B and forms the sum of the products. This instruction uses double-precision, unnormalized, floating-point† arithmetic in the operation. The sum of the double-precision products is as follows:

$(A_0 \bullet B_0) + (A_8 \bullet B_8) + (A_{16} \bullet B_{16}) + \ . \ . \ . = \text{sum } X_0$

$(A_1 \bullet B_1) + (A_9 \bullet B_9) + (A_{17} \bullet B_{17}) + \ . \ . \ . = \text{sum } X_1$

$(A_2 \bullet B_2) + (A_{10} \bullet B_{10}) + (A_{18} \bullet B_{18}) + \ . \ . \ . = \text{sum } X_2$

$(A_3 \bullet B_3) + (A_{11} \bullet B_{11}) + (A_{19} \bullet B_{19}) + \ . \ . \ . = \text{sum } X_3$

$(A_4 \bullet B_4) + (A_{12} \bullet B_{12}) + (A_{20} \bullet B_{20}) + \ . \ . \ . = \text{sum } X_4$

$(A_5 \bullet B_5) + (A_{13} \bullet B_{13}) + (A_{21} \bullet B_{21}) + \ . \ . \ . = \text{sum } X_5$

$(A_6 \bullet B_6) + (A_{14} \bullet B_{14}) + (A_{22} \bullet B_{22}) + \ . \ . \ . = \text{sum } X_6$

$(A_7 \bullet B_7) + (A_{15} \bullet B_{15}) + (A_{23} \bullet B_{23}) + \ . \ . \ . = \text{sum } X_7$

Sums X_0 through X_7 (all double precision quantities) are then added together as follows:

$X_6 + X_2 = Y_0$ $\qquad$ $X_0 + X_4 = Y_2$

$X_7 + X_3 = Y_1$ $\qquad$ $X_1 + X_5 = Y_3$

These double precision results are then further added as follows:

$Y_0 + Y_2 = Z_0$ $\qquad$ $Y_1 + Y_3 = Z_1$

A final double precision add to Z_0 and Z_1 forms the final sum C, C+1. The instruction transmits the upper result portion of the sum to the register specified by C and the lower result to the register designated by C+1. The DC instruction terminates when the shorter of the two source fields is exhausted. If the control vector contains no enabling elements, the result is set to machine zero. G bit 0 determines the size of the A and B operands and registers C and C+1.

Bits 2, 5, 6, and 7 of the G designator are not used and must be zeros. The instruction contains no length designator for the control vector Z.

C must specify an even-numbered register. If C specifies an odd-numbered register, the instruction results are undefined.

Applicable data flag bits are 42 (exponent overflow), 43 (result machine zero), and 46 (indefinite result). Data flag bits 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 bit 43 is set if the exponent of the lower result is less than 9000_{16}. In this case, the exponent of the upper result may be greater than 9000_{16} and will be stored as is and will not be forced to machine zero. The instruction sets data flag bit 42 if any of the multiply operations overflow.

† Appendix B describes floating-point arithmetic and order-dependent result considerations.

STRING INSTRUCTION

The string instruction performs arithmetic and logical operations on strings of data in the form of 8-bit bytes. The 8-bit byte size allows handling large alphabets and is compatible with ASCII and EBCDIC codes. The data strings are in the general format shown in figure 4-37.

The field length of the data string can extend beyond one 64-bit word. The field length of the data string can also be less than one data word.

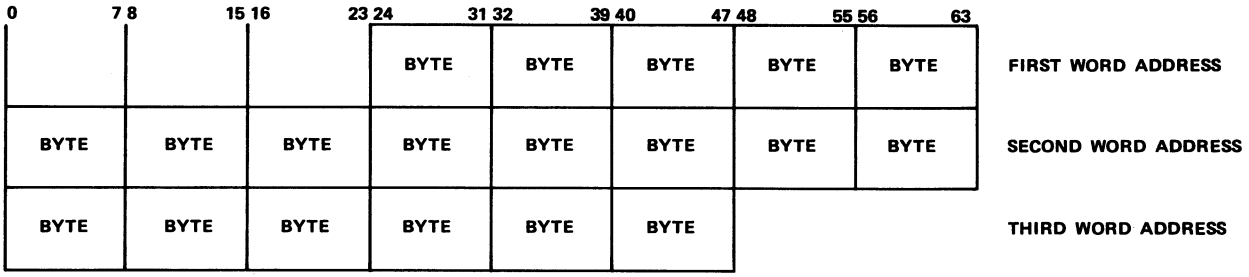
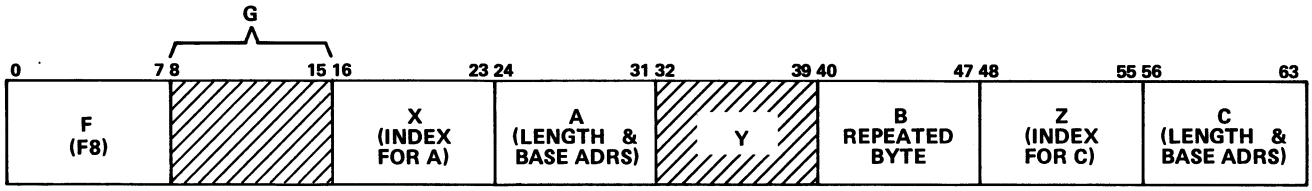


Figure 4-37. Example of General Format of a Data String Field

F8 Move Bytes Left; A→C



F8 Move Bytes Left; A → C

This instruction moves source field A to result field C. The bytes in the field are considered from left to right. Thus, the most significant byte of the source field is moved to the most significant byte position of the result field.

The Y and G designators are not used and must be zeros.

If the origin field is shorter than the destination field, the destination field is filled in with the repeated byte found in the B designator of the instruction.

If the origin field is longer than the destination field, the operation is truncated when the destination field is exhausted.

The field lengths are expressed in bytes.

The 48-bit indexes in the X and Z registers are left shifted 3 bits before being added to the A and C base addresses, respectively.

LOGICAL STRING INSTRUCTIONS

The logical string instructions function in the same general manner as the string instruction. Logical string instructions operate with indexes and data fields identical to those of the string instruction except that the item counts and indexes are expressed in bits instead of bytes. Thus, the logical string instructions perform bit operations on bit boundaries while the string instruction performs byte operations on byte boundaries.

F0 Logical Exclusive OR A, B → C

F1 Logical AND A, B → C

F2 Logical Inclusive OR A, B → C

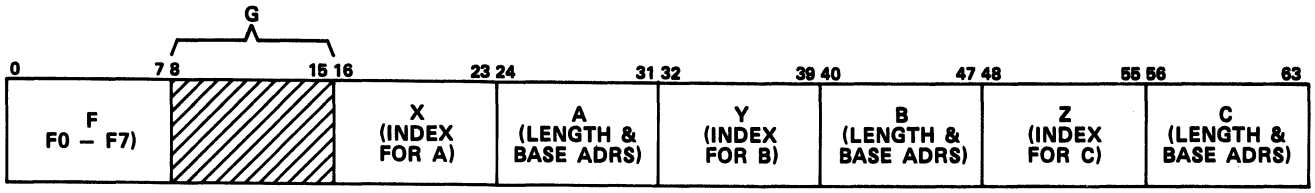
F3 Logical Stroke, A, B → C

F4 Logical Pierce A, B → C

F5 Logical Implication A, B → C

F6 Logical Inhibit A, B → C

F7 Logical Equivalence A, B → C



These instructions perform bit-by-bit logical functions on binary source fields A and B and store the results in binary field C. Table 4-23 lists the variations of source bits A and B with the corresponding result bit for each of the logical string instructions.

TABLE 4-23. TRUTH TABLE FOR LOGICAL STRING INSTRUCTIONS

Source Bits		OR	AND	Exclusive OR	Stroke	Pierce	Implication	Inhibit	Equivalence
A	B	(A+B)	(A • B)	(A-B)	$\overline{(A \bullet B)}$	$\overline{(A+B)}$	$(A+\overline{B})$	$(A \bullet \overline{B})$	$(A-\overline{B})$
0	0	0	0	0	1	1	1	0	1
0	1	1	0	1	1	0	0	0	0
1	0	1	0	1	1	0	1	1	0
1	1	1	1	0	0	0	1	0	1

Fields A, B, and C are strings of bits. The instruction proceeds from left to right and terminates when the result field C is filled. The instruction extends source fields A and/or B with zeros if they are shorter than field C. The G designator is not used and must be all zeros.

Data flag bit 53, 54, or 55 is set according to the condition of the result field as shown in table 4-24.

TABLE 4-24. DFB CONDITIONS FOR F0 THROUGH F7 INSTRUCTIONS

DFB Bit	Condition
53	Result field all zeros.
54	Result field mixed.
55	Result field all ones.

Figure 4-38 shows an example of a logical string instruction operation. A logical exclusive OR (F0) instruction is used for the example. In the example, source field B contains a mask of all ones which is used to complement the binary number in source field A through the exclusive OR function. All indexes and field lengths are item counts, expressed in bits (for example, the source and result field lengths equal 28_{16} bits). The operation proceeds from the starting addresses of A, B, and C to the end of the result field (to address 7030_{16}). Each operation forms the exclusive OR or the corresponding bits in source fields A and B and stores the result in the corresponding position of field C.

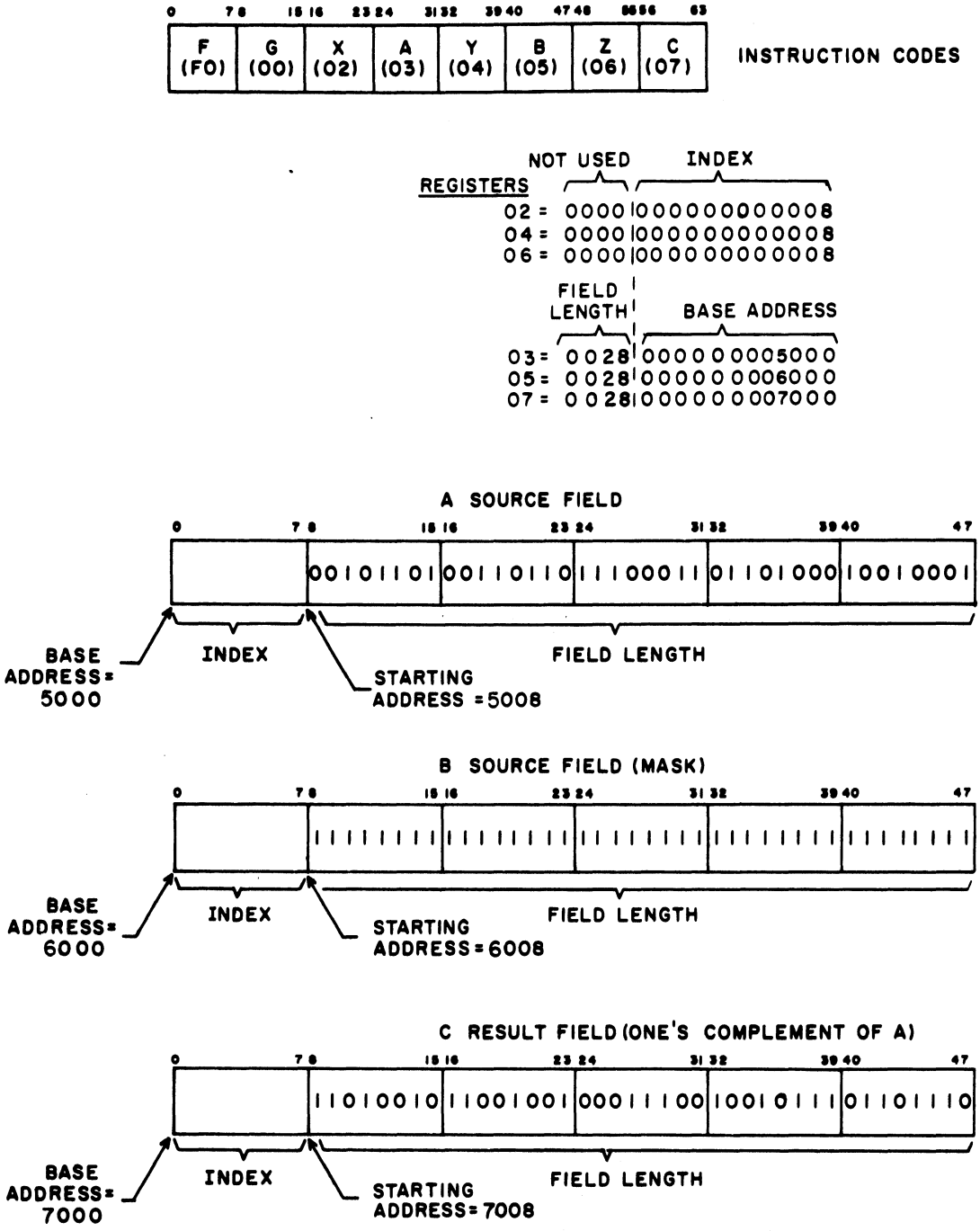


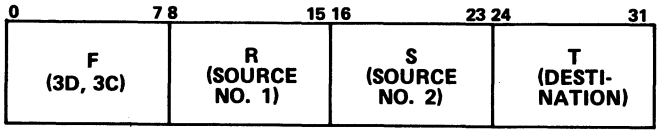
Figure 4-38. Example of Logical String Instruction (Logical Exclusive OR)

NONTYPICAL INSTRUCTIONS

These instructions perform operations such as register to storage transfers, formation of repeated bit masks, and maximum/minimum determinations that do not fall into any of the preceding categories of instructions. The separate instruction descriptions define the format and operation for these instructions. Appendix C provides a complete listing of the various nontypical instruction fields and the resulting termination conditions.

3D Index Multiply (R) • (S) to (T)

3C Half-Word Index Multiply (R) • (S) to (T)



3D Index Multiply (R) • (S) to (T)

This instruction forms the product of the signed two's complement integers contained in the rightmost 48 bits of the registers specified by the R and S designators, respectively. The instruction stores the product in the rightmost 48 bits of register T and clears the leftmost 16 bits.

If the product or either operand exceeds $\pm 2^{47}-1$, the result is undefined.

3C Half-Word Index Multiply (R) • (S) to (T)

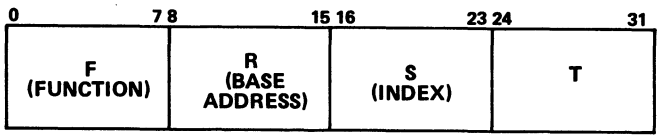
This instruction forms the product of the signed two's complement integers contained in the rightmost 24 bits of the registers specified by the R and S designators, respectively. The instruction stores the product in the rightmost 24 bits of register T and clears the leftmost 8 bits.

If the product or either operand exceeds $\pm 2^{23}-1$, the result is undefined.

5E/7E Load (T) Per (S), (R)

5F/7F Store (T) Per (S), (R)

12/13 Load/Store Byte (T) Per (S), (R)



5E/7E Load (T) Per (S), (R)

These instructions load the 32/64-bit register T with the content of the address specified by (S) + (R), where (R) is the base address. For the 5E instruction, (S) is an item count in half-words, and for the 7E instruction, (S) is an item count in words. The index in S is shifted five/six places to the left before it is added to the base address. S and R are 64-bit registers. Overflow resulting from this addition has no effect if it occurs.

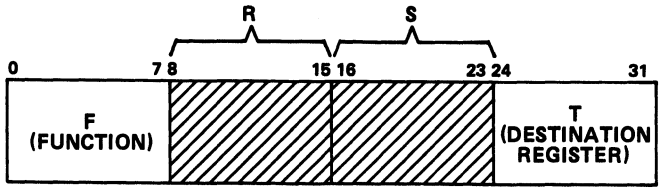
5F/7F Store (T) Per (S), (R)

These instructions store the content of the 32/64-bit register T in the address specified by (S) + (R), where (R) is the base address. For the 5F instruction, (S) is an item count in half-words, and for the 7F instruction, (S) is an item count in words. The index in S is shifted five/six places to the left before it is added to the base address. S and R are 64-bit registers. These instructions do not detect overflow if it occurs.

12/13 Load/Store Byte (T) Per (S), (R)

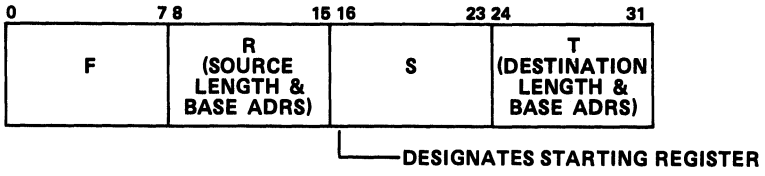
These instructions load/store a byte from/into the address specified by (R) + (S), where (R) is the base address and (S) is an item count in bytes. The index in S is shifted three places to the left before it is added to the base address. The byte is transmitted into/from bits 56 through 63 of register T. The remaining bits in T are cleared on a load and ignored on a store.

37 Transmit Job Interval Timer to (T)



This instruction transmits the contents of the job interval timer into bits 32 through 63 of register T and clears bits 0 through 31 to zero. The designators R and S are undefined and must be set to zero. When executed in monitor mode, the operation of this instruction is undefined. This instruction does not deactivate the timer.

7D Swap S →T, R →S



This instruction moves to destination field T, a portion of the register file beginning at the 64-bit register specified by the rightmost 8 bits of register S. The instruction also transmits source field R to the register file beginning at the 64-bit register specified by the rightmost 8 bits of register S.

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

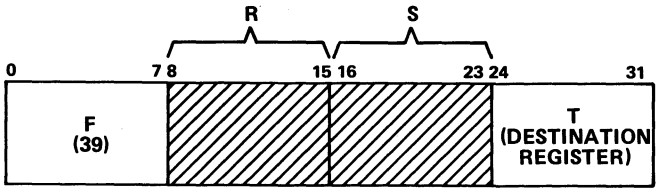
The rightmost 48 bits of registers R and T specify the base address of the source and destination fields, respectively. These addresses must specify an even 64-bit word in central storage. Bits 57 through 63 of registers R and T are undefined and must be set to zero. Overlap of the source and destination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

There are no restrictions relating to registers R, S, or T being in the range of the registers being swapped.

The starting register in the file specified by the rightmost 8 bits of the register specified by S must be an even register, or the instruction will be treated as an undefined instruction.

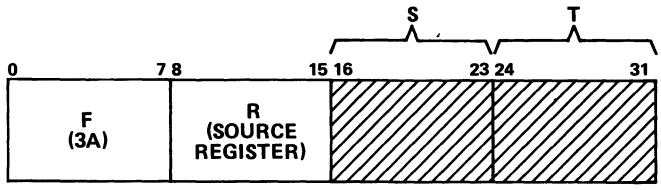
If the source field from the register file includes register zero, the computer transmits the trace register. However, new data from memory is never written into register zero by the swap (7D) instruction.

39 Transmit Real-Time Clock to (T)



This instruction transmits the contents of the real-time clock to bits 16 through 63 of the register designated by T. Bits 0 through 15 of register T are cleared.

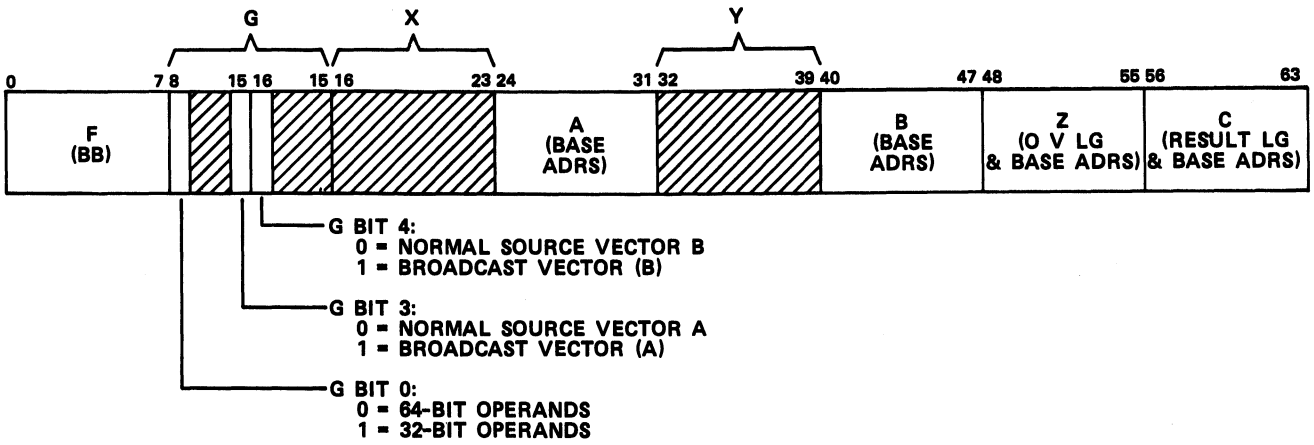
3A Transmit (R) to Job Interval Timer



This instruction transmits bits 32 through 63 of the register designated by R to the job interval timer. When executed in the monitor mode, this instruction functions as a no-op.

Loading the job interval timer with all zeros deactivates the timer without setting data flag bit 36.

BB Mask A, B → C Per Z

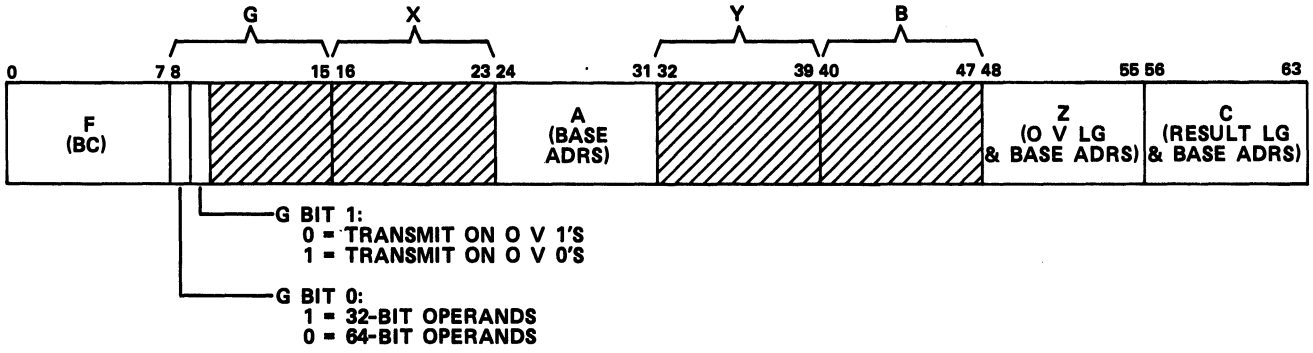


This instruction combines elements of vectors A and B to form result vector C as controlled by order vector Z. When a one is detected in order vector Z, the next element of vector A is inserted into result vector C and the corresponding element of vector B is skipped. When the instruction detects a zero in order vector Z, the instruction inserts the next element of vector B and skips the corresponding element of vector A. When all elements of A and B have been merged, the instruction transmits the resulting length of vector C to the length specification portion of register C as shown in figure 4-31.

Bit 0 of the G designator determines whether 64- or 32-bit operands are used for the A, B, and C vectors. The X and Y designators and bits 1, 2, and 5 through 7 of the G designator are undefined and must be zeros. G bits 3 and 4 determine whether normal vector elements or broadcast elements are used for vectors A and B, respectively. The use of normal or broadcast source vectors are described in Vector Instructions in this section.

This instruction terminates when all bits of the order vector have been examined. The instruction recognizes no lengths for vectors A and B.

BC Compress A → C Per Z



This instruction forms a sparse data vector field C by compressing vector field A. Sparse data vector field C consists of elements of vector field A corresponding to ones in sparse order vector Z. Thus, the elements of vector field A that correspond to the positions of ones in sparse order vector Z transfer in order to corresponding elements of sparse data vector field C if G designator bit 1 equals zero. If this bit is one, the elements of vector field A that correspond to zeros in sparse order vector Z are transferred to corresponding elements of sparse data vector field C.

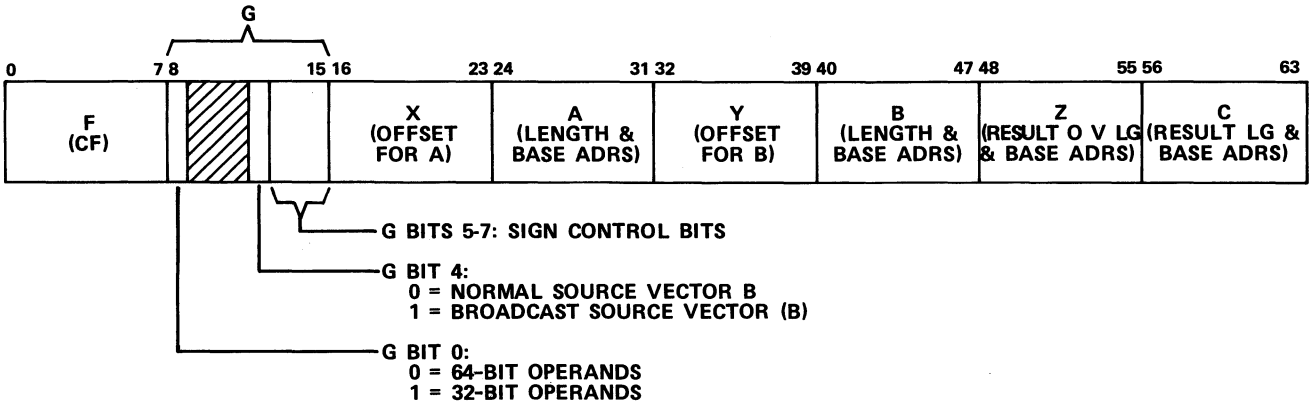
In a typical operation, one of the compare instructions first generates sparse order vector Z. The BC instruction uses the generated order vector as a means of discarding all near-zero elements of vector field A and still maintaining their positional significance through the order vector.

The instruction transfers the resulting length of sparse data vector C to the length specification portion of the register designated by C in the instruction word. If bit 0 of the G designator is zero/one, the operand size (elements of vectors A and C) is 64/32 bits, respectively. As shown in the instruction format, the X, Y, and B designators and bits 2 through 7 of the G designator are undefined and must be zeros.

The instruction terminates when all bits of sparse order vector Z are used. The length specification portion of registers A and C (initial) is not used.

Figure 4-27 shows a simplified example of compressing a vector field into a sparse vector field.

CF Arith. Compress A →C Per B



This instruction forms sparse data vector† C and the associated sparse order vector Z by performing a floating-point compare operation between elements of vector A and the elements of vector B. Each element of vector B is subtracted from the corresponding element of vector A. The conditions for comparing floating-point operands are described in the Floating-Point Compare Rules, appendix B. If an element of vector A is greater than or equal to the corresponding element of vector B (An $\geq$ Bn), the instruction stores the element of A as the corresponding element of sparse data vector C and sets the associated order vector bit. If the element of vector A is less than the corresponding element of vector B (An < Bn), the element of A is not stored in sparse data vector C and the associated order vector bit is cleared. The element of C is not skipped if An < Bn. Thus, in the case of broadcast vector (B), this instruction provides a means of generating a sparse vector field by comparing the elements of a source vector field with a fixed threshold element.

†The sparse vector part of this section describes the general format of sparse vectors.

The registers designated by X and Y contain the offsets for vectors A and B, respectively.

The elements of vectors A and B are in floating-point format.† The sign control bits of the G field may specify operations on the elements of vector A and/or B before the floating-point compare is made. However, the element of A, if stored in C, will be the original element as read from A. The compare operation follows the floating-point compare conditions as described in the branch instruction section. In the comparison, only (R) $\geq$ (S) condition is detected where, in this case, An and Bn are substituted for (R) and (S), respectively. If the instruction detects an indefinite operand for vectors A and B, the indefinite operand is stored as the corresponding element of vector C and the associated bit of the order vector is set.

The instruction format shows that if bit 0 of the G designator is a zero/one, the vector elements are 64-bit/32-bit operands, respectively. If bit 4 of the G designator is a one, a constant element is broadcast for vector B as described in Vector Instructions in this section. In this case, the Y designator is not used. G bits 1 through 3 are not used and must be zeros. G bits 5 through 7 function as sign control bits as described in Vector Instructions.

This instruction terminates when all the elements of vector A have been compared. At termination, the instruction stores the length (in bits) of the generated order vector into the length portion (bits 0 through 15) of the register specified by Z. The number of elements stored in vector C is stored in the length portion of register C, thus providing the field length of the generated sparse vector. If the length of vector B is shorter than the length of vector A, the instruction extends the B field with machine zero elements to equal the A field length. The applicable data flag bit is 46 (indefinite result).

Figure 4-39 is an example of an arithmetic compress instruction with assumed instruction code, register contents, and source vector field A. In this example, a broadcast floating-point constant B is compared with source vector elements A_1 through A_6. Element A0 is not compared because of the offset. The A vector elements are indicated as being An $\geq$ B or An $<$ B. Thus, the instruction in this example generates a 4-element result vector C and a 6-bit order vector Z. The 6 and 4 values are stored in the field length portions of registers 08 and 09, respectively.

†Appendix B describes the floating-point format.

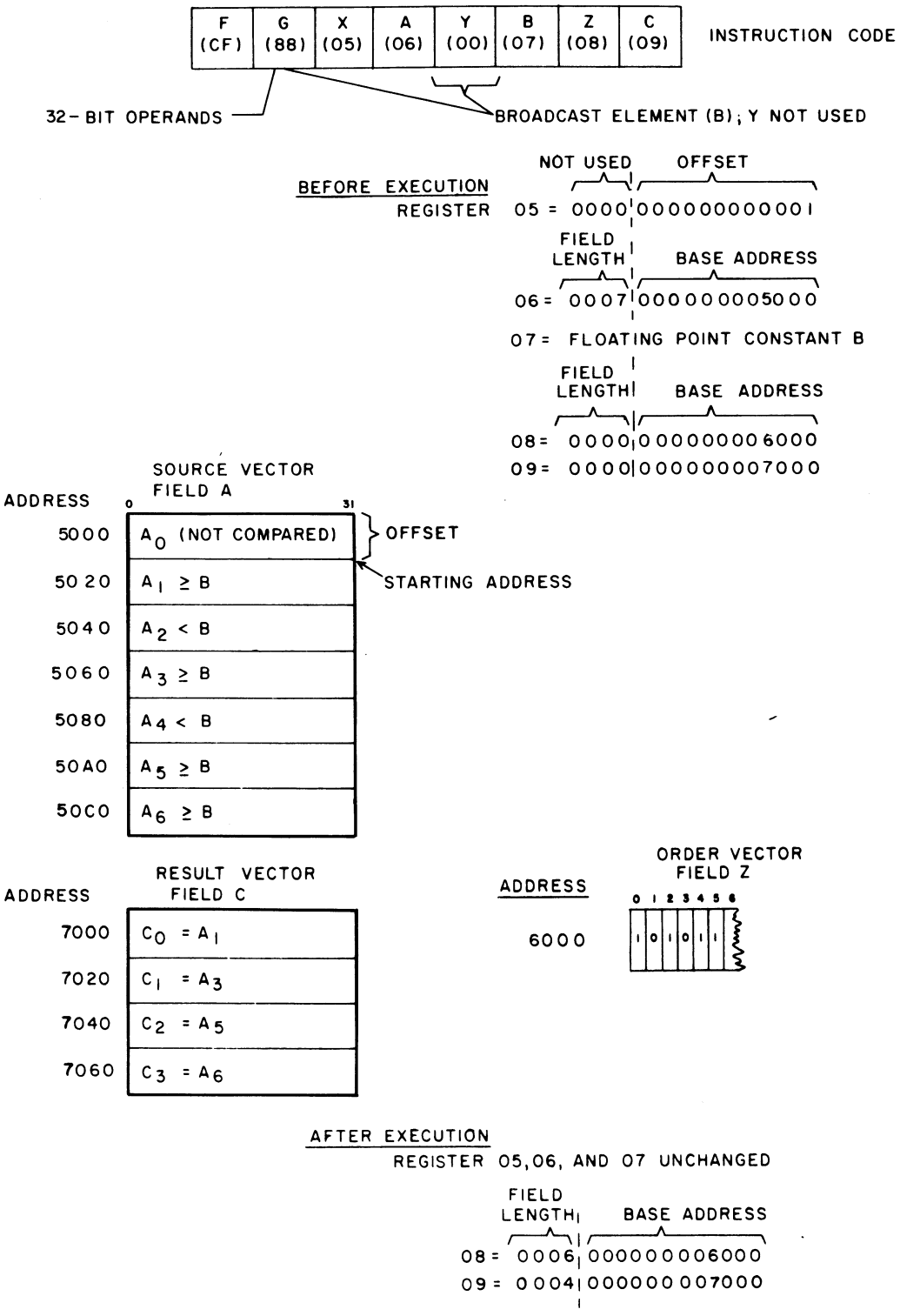
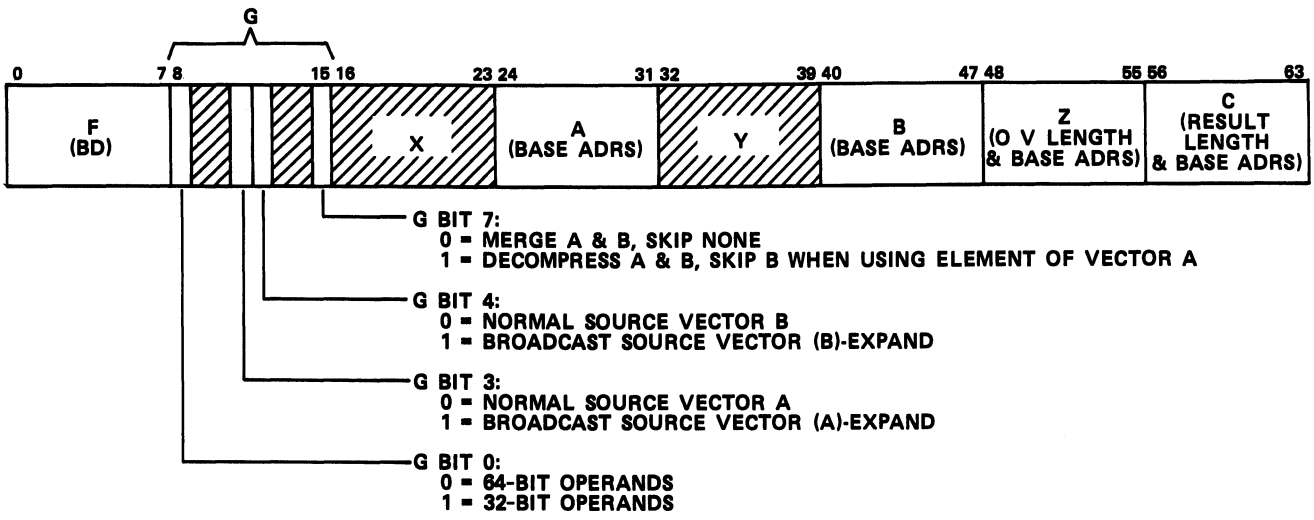


Figure 4-39. Example of Arithmetic Compress A → C Per B Instruction

BD Merge A, B → C; Per Z



This instruction merges the elements of vector field A with the elements of vector field B to form result vector field C as controlled by order vector Z. Thus, this instruction could be used to reform a vector field from a sparse vector with a broadcast near-zero element. When the order vector Z contains a one in a given position, the instruction inserts the next element from vector field A into vector field C. If the order vector contains a zero, the instruction inserts the next element from vector field B in the result field (figure 4-40). The instruction transmits the resulting length of vector C to the length specification portion (bits 0 through 15) of register C.

Field B vector elements are controlled by G bit 7. When G bit 7 is a zero, the operation (called merge) combines vectors A and B. When G bit 7 is set (decompress), an element of vector B is skipped for each element of vector A used. No elements of vector A are skipped when elements of vector B are used.

If bit 0 of the G designator is a zero/one, the operand size for vectors A, B, and C is 64/32 bits, respectively. The X and Y designators and G bits 1, 2, and 5 through 6 are undefined and must be zeros. Bits 3 and 4 of the G designator determine whether a constant element is broadcast from the registers designated by A and B, respectively. If G bit 3 or 4 is a one, the operation is called expand.

The BD instruction terminates when all of the bits of the order vector have been processed. The field length specifications for vectors A and B are not used.

THE Z-BIT STRING IS USED FOR ALL THREE EXAMPLES. G BITS NOT INDICATED ARE ZEROS.

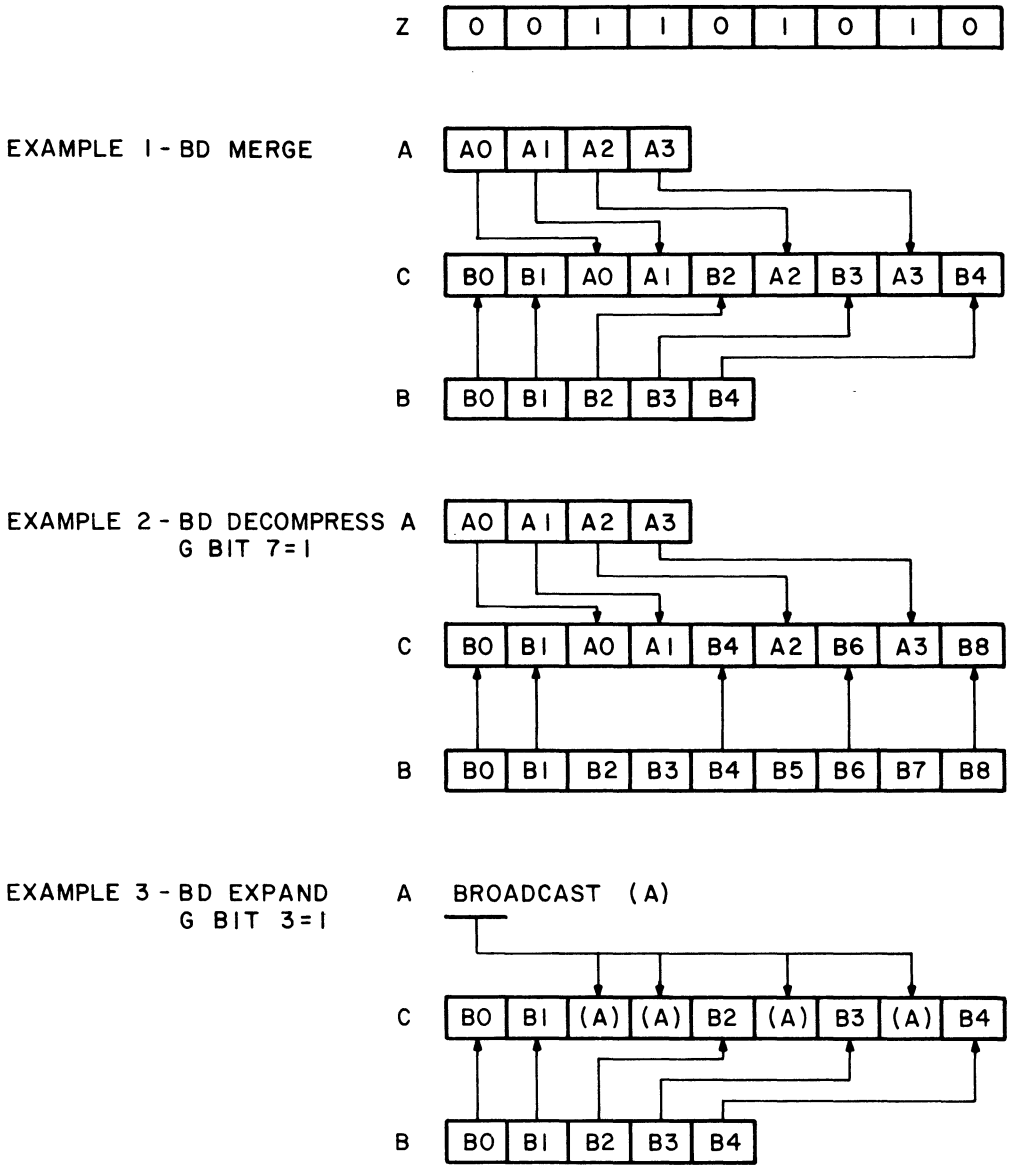
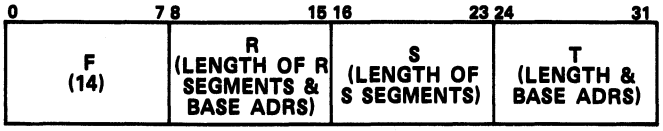


Figure 4-40. Examples of BD Merge Instruction

14 Bit Compress



This instruction compresses specified segment lengths (in bits) of source field R into result field T. The R designator code in the instruction specifies a 64-bit register which contains the length of the R segments in the leftmost 16 bits and the base address of the source field in the rightmost 48 bits (figure 4-41). The register denoted by S contains the length of the segments in the source field to be skipped in the compress operation. The rightmost 48 bits of register S are not used.

Register T contains the destination field length in the leftmost 16 bits and the base address of the destination field in the rightmost 48 bits.

The bit compress operation successively transmits the segment lengths of the source field, as specified by R, to corresponding lengths of the destination field. The instruction moves from left to right in the source and destination fields. The instruction skips the segment lengths of the source field as specified by S.

Figure 4-41 shows that the instruction transfers segments R_1, R_2, and R_3 in the source field to corresponding segment lengths of the result field. Source field segments S_1 and S_2 are skipped. The operation continues until the T field length is filled. If the field length specified by R or T is zero, the instruction functions as a no-op.

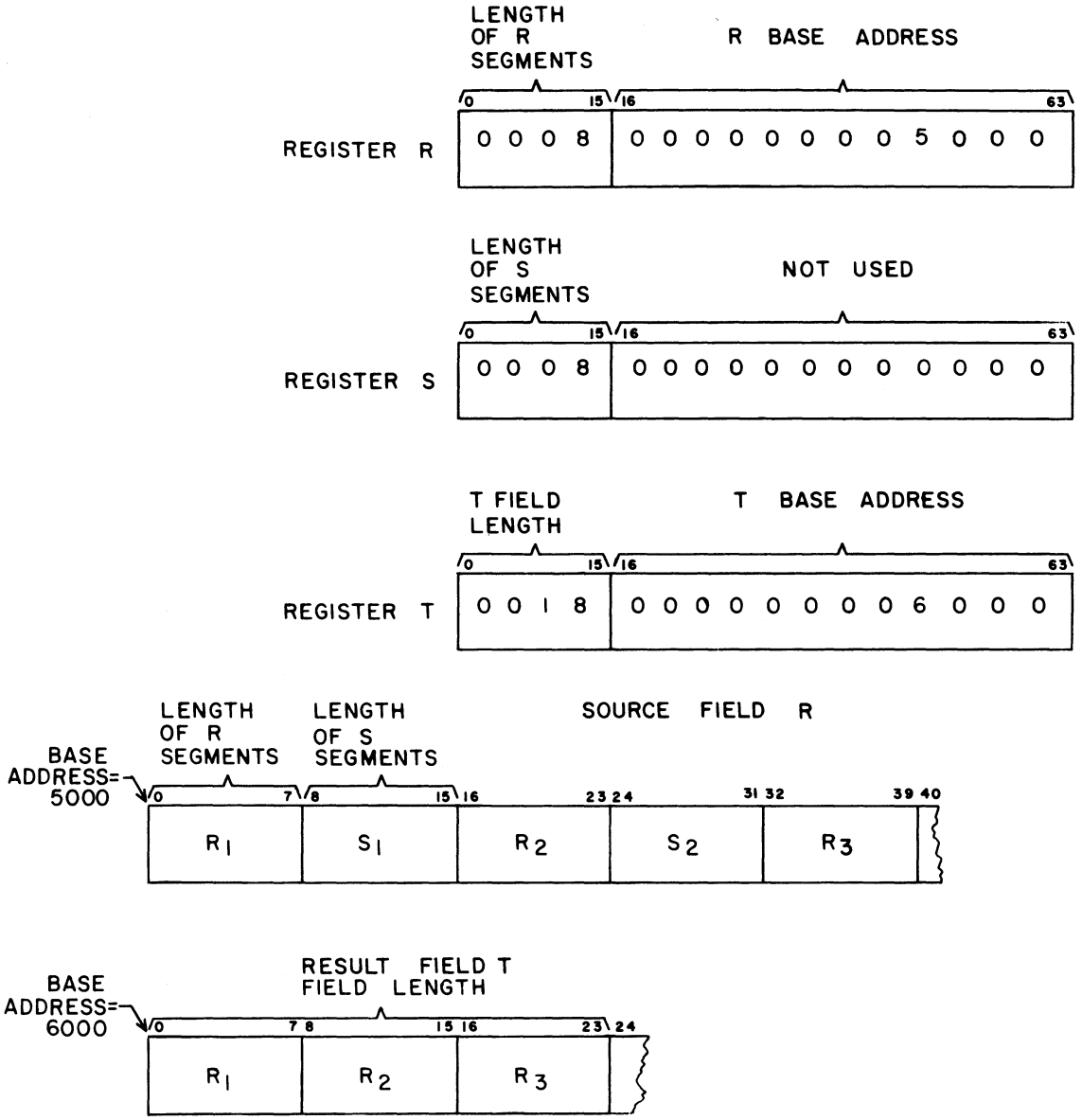
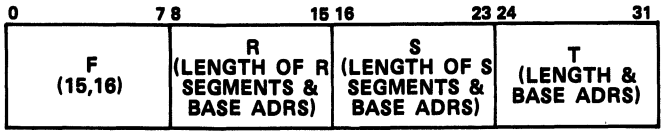


Figure 4-41. Example of Bit Compress Instruction

15 Bit Merge
16 Bit Mask



15 Bit Merge

The bit merge instruction merges specified segment lengths (in bits) of source fields R and S into result field T. The 64-bit register specified by R contains the length of the R segments in the leftmost 16 bits and the base address of the R source field in the rightmost 48 bits (figure 4-42). The register denoted by S contains the length of the S segments in the leftmost 16 bits and the base address of the S source field in the rightmost 48 bits. Register T contains the destination field length in the leftmost 16 bits and the base address of the destination field in the rightmost 48 bits.

The bit merge operation successively merges the segment lengths of the R source field with segment lengths of the S source field into corresponding lengths of the destination field. The instruction moves from left to right in the source and destination field.

Figure 4-42 shows that the 15 instruction merges segments R_1, R_2, and R_3 in source field R with segments S_1 and S_2 into corresponding segment lengths of the destination field. The operation continues until the T field length is filled.

If bits 16 through 63 of the S register are cleared, the instruction transmits zeros to the corresponding segment lengths in the destination field. If the field length specified by the R, S, or T registers is zero, the instruction functions as a no-op.

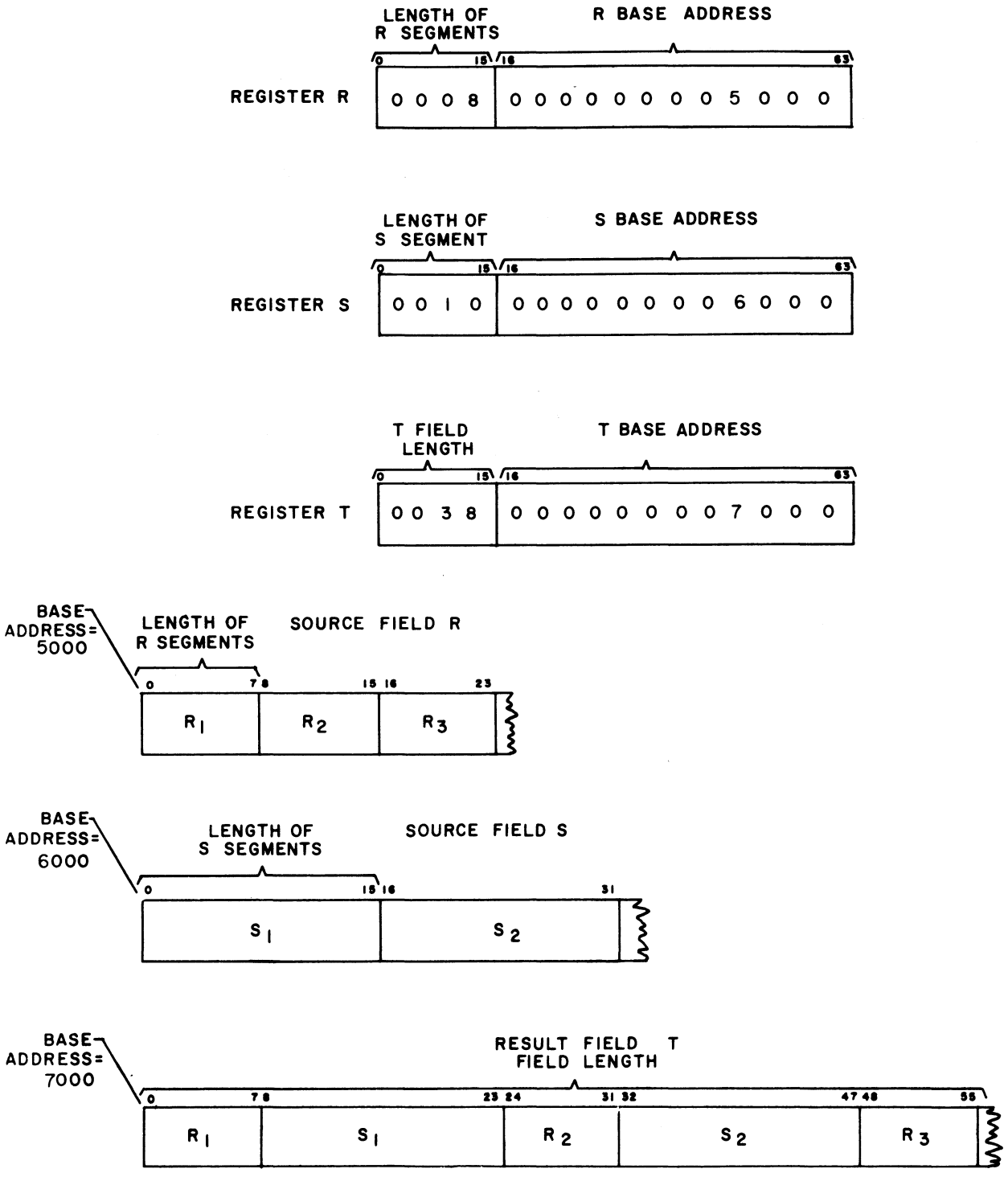


Figure 4-42. Example of Bit Merge Instruction

16 Bit Mask

The bit mask instruction is similar to the bit merge instruction. The specified R, S, and T register contain segment lengths, base address, and field length in the same manner. However, the bit mask instruction (figure 4-43), moving from left to right, transmits a segment equal to the length specified by R to the corresponding segment length in the destination field. The 16 instruction then transmits a segment of field S equal to the segment length specified by the S register starting at an address equal to the base address plus the R segment length. The next segment of the R source field to be transmitted to the destination field starts at an address equal to the R base address plus the R segment length plus the S segment length. As in the bit merge instruction, if bits 16 through 63 of the S register are cleared, the instruction transmits zeros to the corresponding segment lengths in the destination field. In the same manner, if the field lengths specified by the R, S, or T register is zero, the instruction becomes a no-op. The bit mask operation continues in this manner until the destination field is filled.

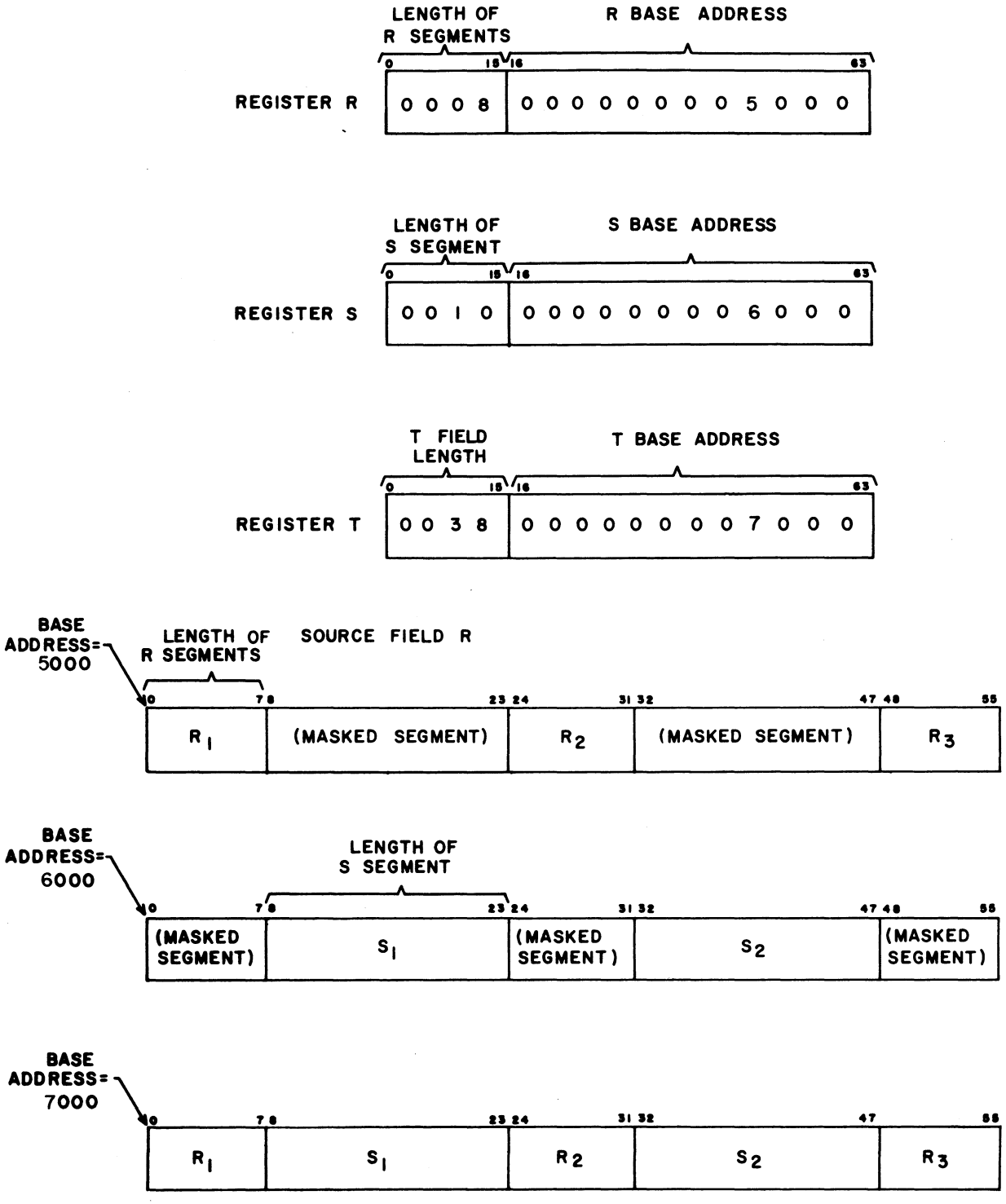
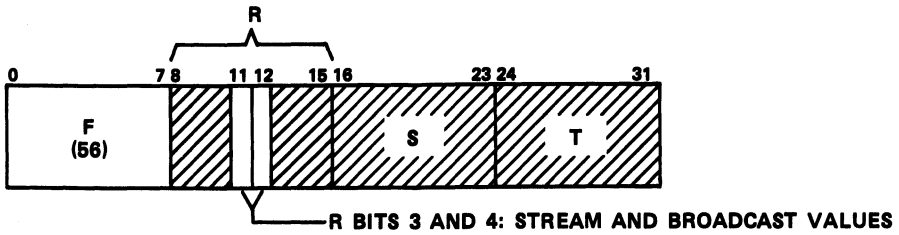


Figure 4-43. Example of Bit Mask Instruction

56 Select Link



For certain vector operations (refer to table 4-25), this instruction provides the ability to combine two vector operations into one single operation. The link instruction accomplishes this by chaining the output of the first vector's function to one of the inputs for the second vector's function. The link instruction must be followed immediately by the two vector instructions (except when R equals zero, which is a pass operation) to be linked such as:

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

The entire operation will be done as one vector with function F1 preceding function F2. Designators Z2, C2, (C+1) 2 and G2 bits 1, 2 will be used to specify the output stream and designators Z1, C1, (C+1)1 and G1 bits 1, 2 will be ignored. Between the two vectors there can only be two input streams (one A and one B) and one broadcast value, or one input stream and two broadcast values (one A or one B). Refer to figure 4-44.

The stream and broadcast values selected are specified by G1 bits 3 and 4, G2 bits 3 and 4 of the vector instructions and R bits 3 and 4 of the link instruction. Refer to table 4-26 for possible combinations.

The two inputs to the first function F1 are A1 (selected by designators X1, A1 and G1 bit 3) and B1 (selected by designators Y1, B1 and G1 bit 4). The two inputs (I2 and J2) to the second function (F2) are the output of F1 and either input A2 (selected by designators X2, A2, and G2 bit 3) or input B2 (selected by designators Y2, B2, and G2 bit 4). The input configuration to F2 is determined by R bits 3 and 4 of the link instruction (refer to table 4-27).

TABLE 4-25. INSTRUCTIONS USED IN A LINK OPERATION

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

†Functional unit number where:

1. Array Shift
2. Array Logical
3. Array Multiply
4. Array Add

NOTES:

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

G1 bit 0 and G2 bit 0 must be equal.

G1 bits 5-7 apply to F1.

G2 bits 5-7 apply to F2.

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

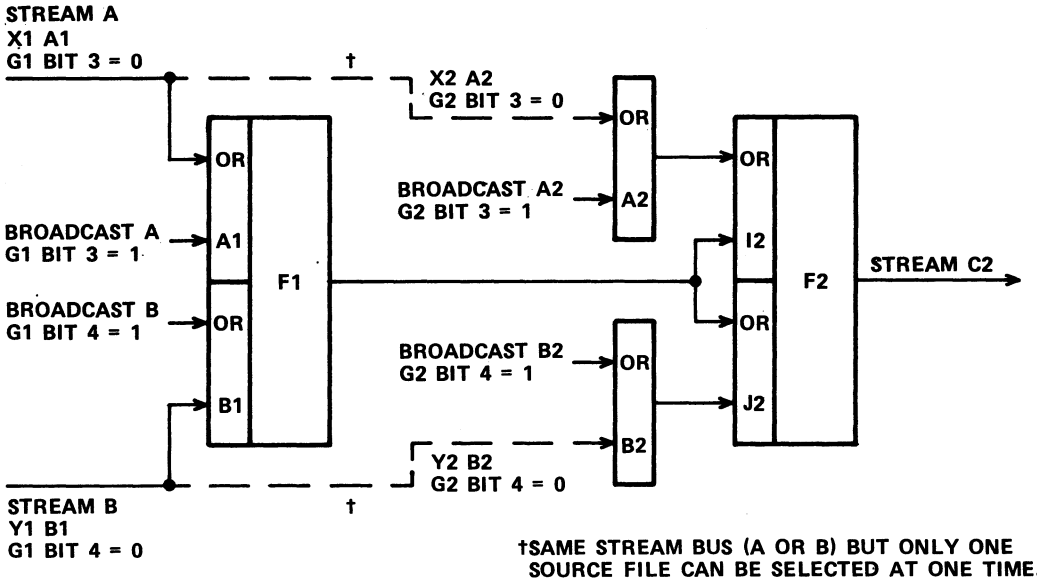


Figure 4-44. Link Selection

TABLE 4-26. COMBINATIONS OF R, G1, AND G2 BITS 3 AND 4 THAT CAN BE SELECTED

	R Bits 3,4=01	R Bits 3,4=10	R Bits 3,4=11
G1 Bit 3 =	0 1 1 1 0	0 0 0 1 1	0 0 1 1
G1 Bit 4 =	0 0 0 1 1	0 1 1 1 0	0 1 0 1
G2 Bit 3 =	1 0 1 0 1	0 0 0 0 0	0 0 0 0
G2 Bit 4 =	0 0 0 0 0	1 0 1 0 1	0 0 0 0
NOTE: Combinations other than above produce undefined results.			

TABLE 4-27. INPUT CONFIGURATIONS

R Bits	State	Instructions
	00	Pass
3,4	01	Select A2 → I2, F1 → J2 and Ignore B2
	10	Select B2 → J2, F1 → I2 and Ignore A2
	11	Select F1 → I2 and F1 → J2 and Ignore B2 and A2
0-2		Undefined and must be set to zero
5-7		Undefined and must be set to zero

Link instruction designators S and T and G bits 0, 1, 2, 5, 6, and 7 are undefined and must be set to zeros.

A link instruction is a pass (no-op) when R bits 3 and 4 are both zeros. The two vector instructions following a no-op link instruction are executed as separate vectors. To operate similarly to a linked vector, the first vector's result field must be stored in memory where it is then read by the second vector instruction as an input vector. To be assured of identical results for the two vectors when unlinked as when linked, the two vectors must use a common control vector and the intermediate output and input field lengths must be equal to, or greater than, the second vector's output field length. When the second vector is a C4, C5, C6, or C7 instruction, the first vector must have all control vector bits permissive for identical linked and unlinked results.

COMPARE INSTRUCTIONS (B0 THROUGH B5)

The central computer has expanded capabilities for the B0 through B5 instructions (refer to Branch Instructions in this section for other use of these instructions). Which set of B0 through B5 instructions to use depends on the values given to G bits 0 through 3.

NOTE

For these instructions:

G bit 1 = 0
G bit 2 = 1

B0 Compare Integer, Set Condition if (A) + (X) = (Z)

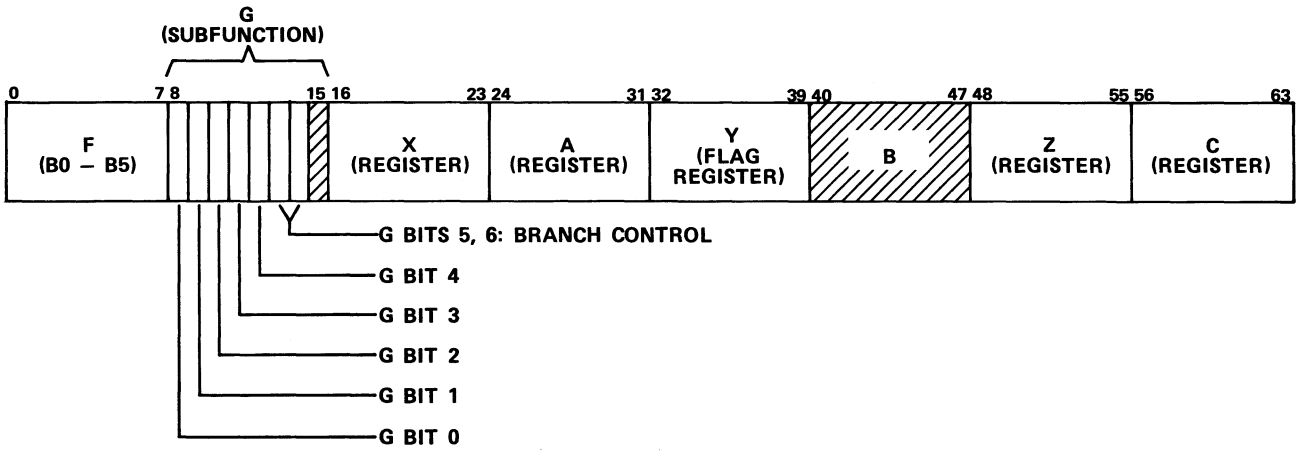
B1 Compare Integer, Set Condition if (A) + (X) ≠ (Z)

B2 Compare Integer, Set Condition if (A) + (X) ≥ (Z)

B3 Compare Integer, Set Condition if (A) + (X) < (Z)

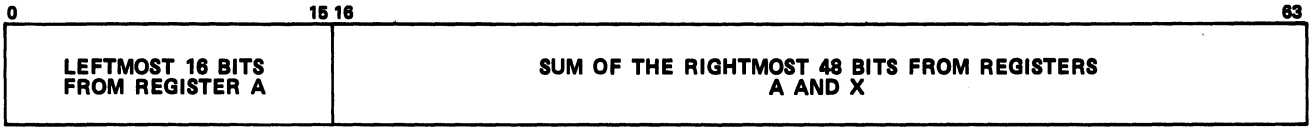
B4 Compare Integer, Set Condition if (A) + (X) ≤ (Z)

B5 Compare Integer, Set Condition if (A) + (X) > (Z)



For these instructions, G bit 1 is clear (0) and G bit 2 is set (1). These instructions compare two integer operands from register A and X. If G bit 0 is clear (0), registers A, X, Y, C, and Z are 64 bits. If G bit 0 is set (1), these registers are 32 bits. Register B is not used and must be set to zero.

If G bit 0 is 0, the sum of the rightmost 48-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 48 bits of register Z according to the specified condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 48 bits of register C. The leftmost 16 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



The sum of the rightmost 48 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values.

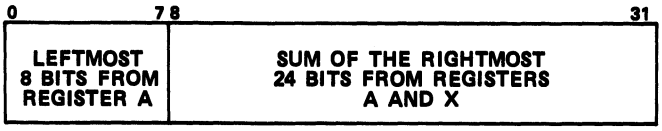
G bit 3 = 0 — The integers compared are the 48-bit result of registers A and X and the rightmost 48 bits read from register Z.

G bit 3 = 1 — The integers compared are the 64 bits stored in register C and the 64 bits read from register Z. This compare is defined for the B0 and B1 instructions only.

G bit 4 = 0 — The integers compared are interpreted as signed two's complement numbers.

G bit 4 = 1 — The integers compared are interpreted as unsigned numbers.

If G bit 0 is 1, the sum of the rightmost 24-bit integers from registers A and X is formed, ignoring overflow. The sum is compared to the rightmost 24 bits of register Z, according to the specified condition. The original content of register Z is read before the sum of registers A and X is stored in the rightmost 24 bits of register C. The leftmost 8 bits of register A are copied into the leftmost bits of register C. Register C contains the following:



Then the sum of the rightmost 24 bits of registers A and X is compared to register Z, based on the following G bit 3 and 4 values:

G bit 3 = 0 — The integers compared are the 24-bit result of registers A and X and the rightmost 24 bits read from register Z.

G bit 3 = 1 — Undefined.

G bit 4 = 0 The integers compared are interpreted as signed two's complement numbers.

G bit 4 = 1 The integers compared are interpreted as unsigned numbers.

Refer to table 4-13 for integer ranges.

If the specified compare condition is met, a 64- or 32-bit quantity (depending on G bit 0) 00....0001 is stored in register Y and the program reads the next sequential instruction.

If the specified compare condition is not met, a 64- or 32-bit quantity (depending on G bit 0) 00....000 is stored in register Y and the program reads the next sequential instruction.

If one of the following conditions occurs, the operation becomes undefined.

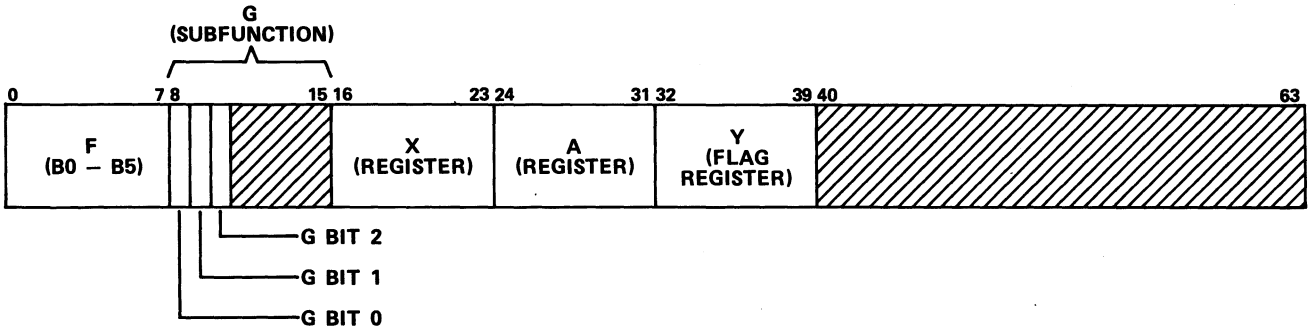
- G bit 0 is 1 and G bit 3 is 1.
- G bit 3 is 1 for B2, B3, B4, and B5.
- G bit 5 is 1, G bit 6 is 1, or G bit 7 is 1.
- The C designator is equal to the Z designator.

NOTE

For these instructions:

G bit 1 = 0
G bit 2 = 1

B0 Compare FP, Set Condition if (A) = (X)
B1 Compare FP, Set Condition if (A) ≠ (X)
B2 Compare FP, Set Condition if (A) ≥ (X)
B3 Compare FP, Set Condition if (A) < (X)
B4 Compare FP, Set Condition if (A) ≤ (X)
B5 Compare FP, Set Condition if (A) > (X)



For these instructions, G bit 1 is set (1) and G bit 2 is set (2). These instructions compare two floating-point operands from registers A and X according to the floating-point compare rule in appendix B. If G bit 0 is clear (0), registers A, X, and Y are 64 bits. If G bit 0 is set (1), these registers are 32 bits. Registers B, C, and Z are unused and must be set to zero.

If the specified compare condition is met, a 64- or 32-bit quantity (depending on G bit 0) 00....0001 is stored in register Y and the program reads the next sequential instruction.

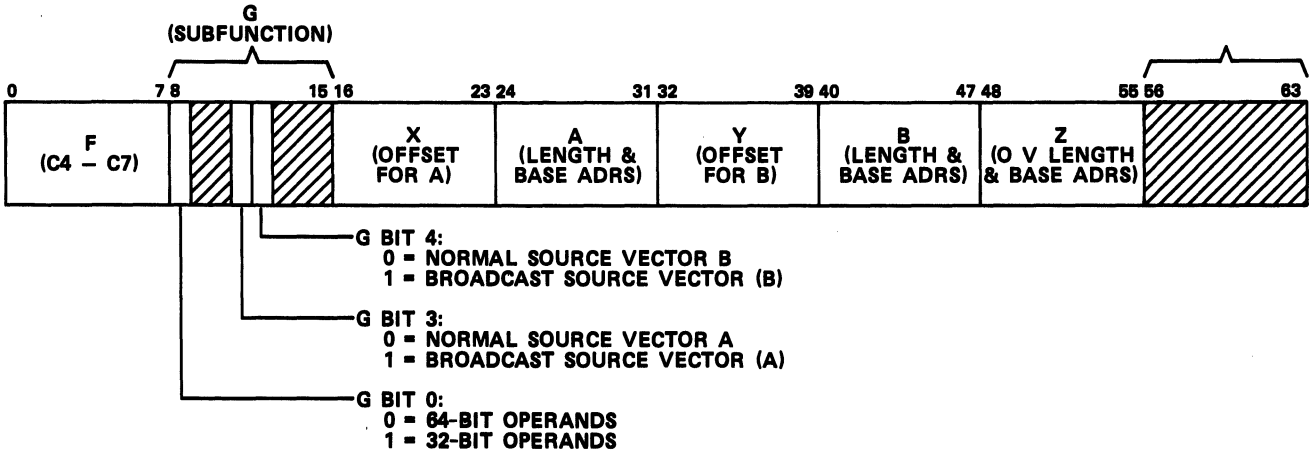
If the specified compare condition is not met, a 64- or 32-bit quantity (depending on G bit 0) 00....000 is stored in register Y and the program reads the next sequential instruction.

If one of the following conditions occurs, the operation becomes undefined.

- Any one of G bits 3 through 7 is set (1).
- Designators B, Z, and/or C are not equal to zero.

Applicable data flag bit is 46 (indefinite result).

C4 Compare EQ; A = B, Order Vector → Z
C5 Compare NE; A ≠ B, Order Vector → Z
C6 Compare GE; A ≥ B, Order Vector → Z
C7 Compare LT; A < B, Order Vector → Z



NOTE: THE C + 1 DESIGNATOR IS NOT USED BY THIS INSTRUCTION.

These instructions compare successive elements of vector A with corresponding elements of vector B by subtracting vector B from vector A. The elements of the vectors are in floating-point format. The conditions for comparing floating-point operands are described in the Floating-Point Compare Rules, appendix B. If the specified compare condition is met (A =, $\neq$, $\geq$, or < B), the instruction sets the corresponding bit of order vector Z. If the condition is not met, the instruction clears the corresponding bit of Z. The instruction terminates when the order vector Z field is filled. Thus, the compare instructions provide a means of generating an order vector for reducing a vector field to a sparse vector field.

G bits 1, 2, 5 through 7, and the C designator are unused and must be zeros. The C + 1 designator is not used. Thus, no offset can be assigned to order vector Z.

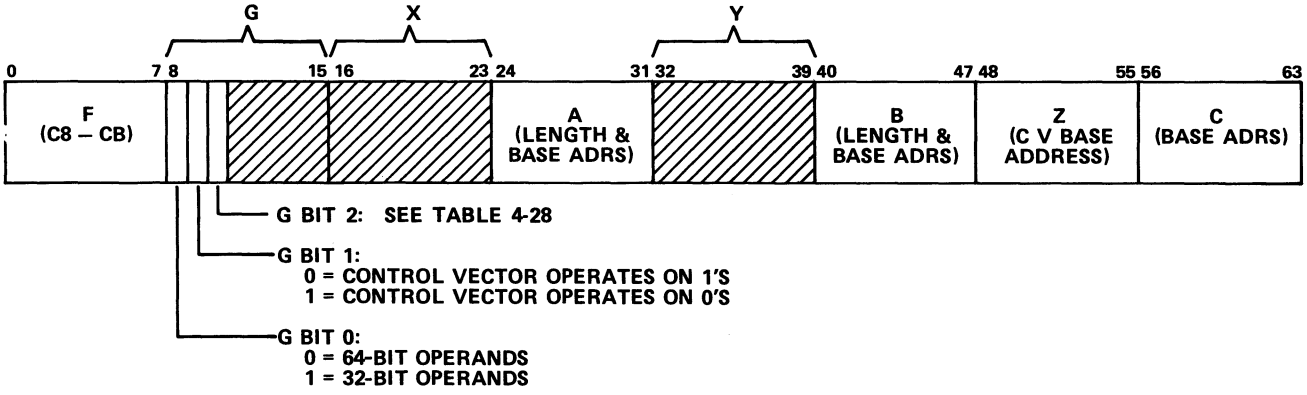
The registers specified by X and Y contain the offsets for vectors A and B, respectively. When a constant is broadcast for either source vector, no field length is specified for that vector, and the offset are undefined and must be set to zero.

The field lengths and base addresses for vectors A, B, and Z are contained in the registers specified by A, B, and Z, respectively. The lengths of vectors A and B are in words (64-bit operands) or half-words (32-bit operands), and the length of order vector Z is in bits.

The applicable data flag bit is 46 (indefinite result).

Figure 4-45 is a simplified example of a compare instruction (C6) with assumed instruction codes, register contents, and source vector field A. In the example, a broadcast constant of +1 is used for vector field B. The elements of vector field A at addresses 5040, 5060, 50E0, and 5100 set the corresponding bits of order vector Z, while the elements at addresses 5080, 50A0, and 50C0 clear the corresponding bits. Although the coefficients of the elements at addresses 5080 and 50A0 are larger than the coefficient of constant B, the negative exponents cause the results of the floating-point subtract operation (normalized upper) to be negative (A < B).

C8 Search EQ; A = B, Index List → C
C9 Search NE; A ≠ B, Index List → C
CA Search GE; A ≥ B, Index List → C
CB Search LT; A < B, Index List → C



NOTE: THE C+1 DESIGNATOR IS NOT USED

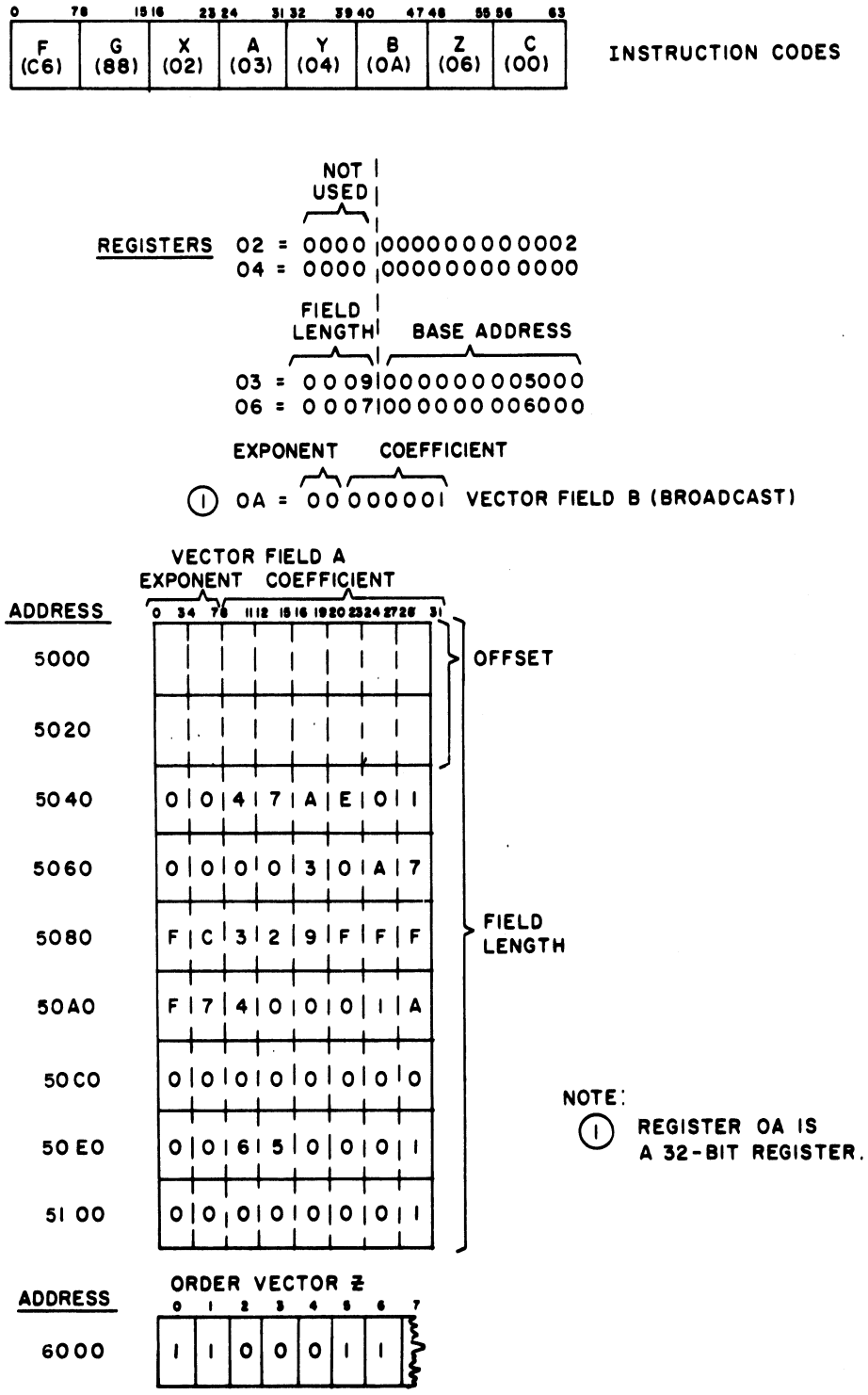


Figure 4-45. Example of Compare GE; A $\geq$ B; Order Vector → Z Instruction

These instructions search and compare each element of vector field A with the successive elements of vector field B by subtracting vector B from vector A. The conditions for comparing floating-point operands are described in the Floating-Point Compare Rules, appendix B. The comparison and search of a given element of A with the elements of B, as specified by G designator bit 2, is defined as one search iteration. Each search iteration terminates when the condition specified by the instruction is found (A =, $\neq$, $\geq$, or < B) or when each element of B has been searched.

After each iteration, the instruction clears the corresponding element of result vector C, if the control vector bit is permissive, and transfers to this element an item count of the number of elements of B that were searched without the specified condition being found (no hit). The item count does not include the hit condition if one is found. Regardless of the operand size (32- or 64-bit elements), the resulting item count is contained in the rightmost 48 bits of a 64-bit word. The leftmost 16 bits of each C vector element are cleared. If no element in the B vector causes a hit condition, the item count equals the field length of the B vector. The control vector controls the storing of the elements of vector C as specified by bit 1 of the G designator. The function of the control vector is described in Vector Instructions in this section.

These instructions use the floating-point compare conditions as described in Branch Instructions in this section. The conditions specified by bits 0 and 1 of the G designator are shown in the previous instruction format. The conditions specified by bit 2 of the G designator are listed in table 4-28. The instruction format also shows that the X and Y designators and G bits 3 through 7 are not used and must be zeros. These instructions use no field lengths or offsets for vectors C and Z. Thus, the C + 1 designator is not used.

TABLE 4-28. SEARCH ITERATION STARTING DESIGNATOR CONDITIONS

G Bit 2	Conditions
0	Start at the beginning of vector B for each element of vector A.
1	Start at the location of the last hit in vector B for each element of vector A.

These instructions terminate when each element of vector A has been compared with each element of vector B. The applicable data flag bit is 46 (indefinite result).

Figure 4-46 is an example of a search equal (C8) instruction with assumed instruction codes, register content, and vector fields. In the example, two search iterations compare the two elements of the A vector with the four elements of the B vector. The comparisons in the first iteration are represented by solid lines while those in the second iteration are indicated by dashed lines. Since bit 2 of the G designator is a zero for this case, each search iteration starts at the beginning of vector B. If the B vector becomes exhausted and G bit 2 = 1, all search iterations start and end with the end of the B vector. If the length of vector B is initially zero, all indexes stored are zero.

In the first iteration, three comparisons take place before the hit condition (A = B) is detected. As a result, an item count of three is entered into the first result element. No hit is detected in the second iteration; thus, the second result element equals the field length of the B vector (4). Since the two corresponding bits of the control vector are set, both result elements are stored.

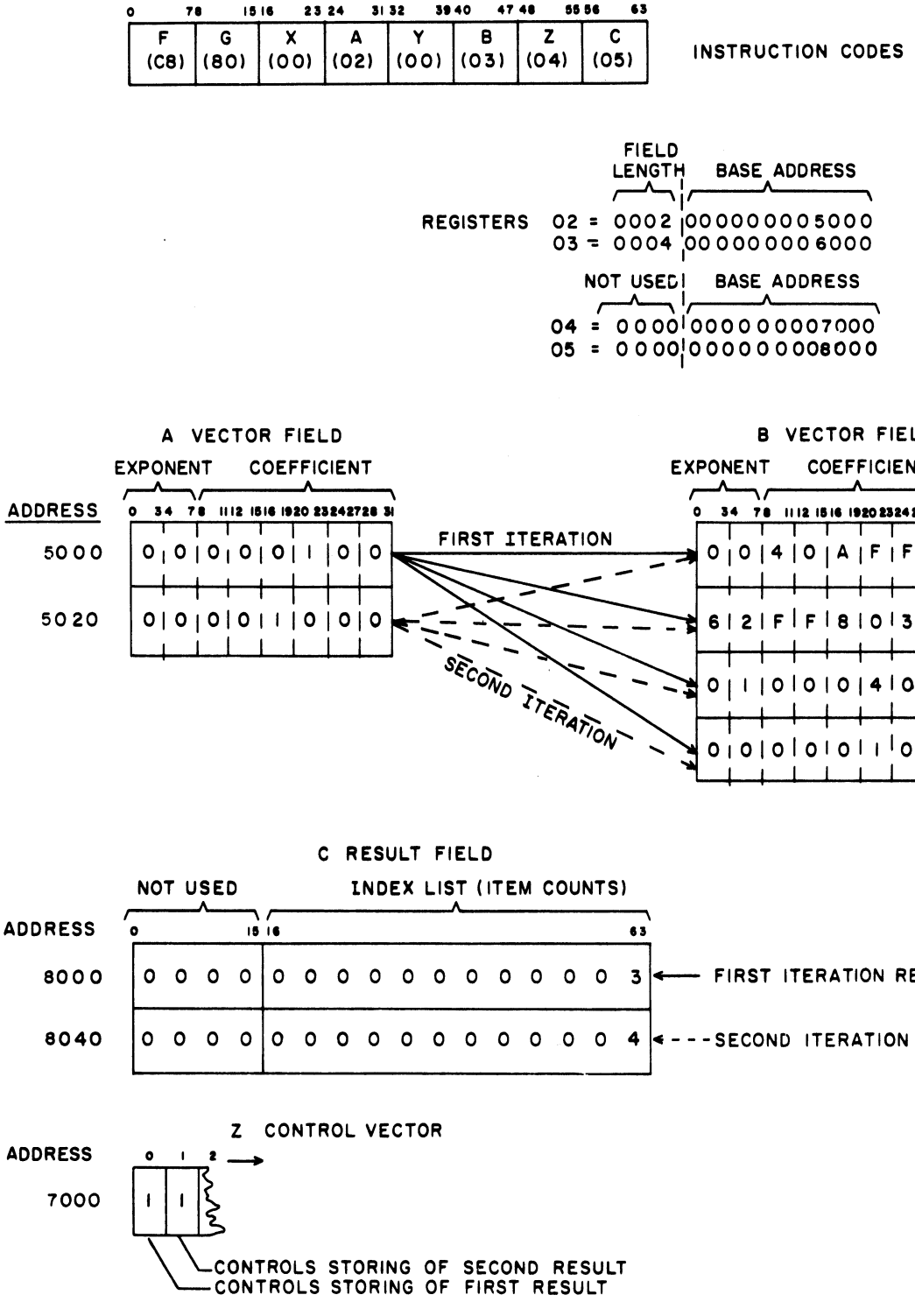
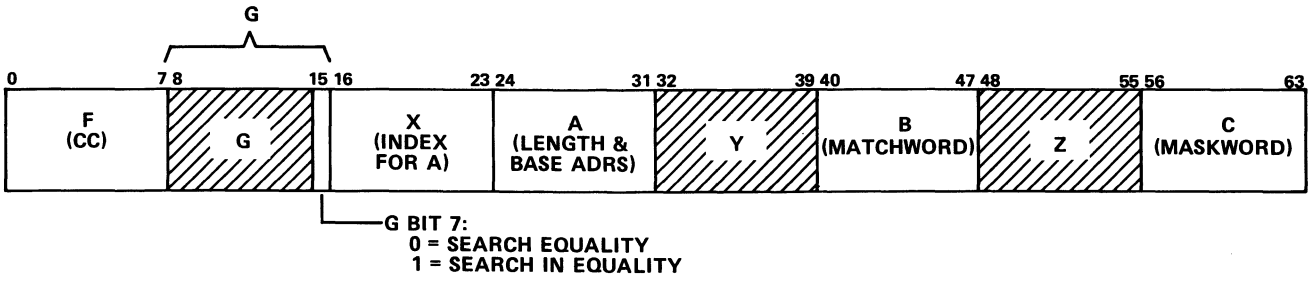


Figure 4-46. Example of Search EQ; A = B, Index List → C

CC Mask Binary Compare; A EQ/NE (B) Per (C)



This instruction searches source field A (Reference Field) for a match with the contents of the register specified by the B designator. The contents of the register specified by the C designator serves as a word mask such that the instruction makes a word-by-word comparison only when there are ones in the corresponding bit positions of the mask. Bits of the reference field and the contents of B are considered to match wherever there is a zero bit in the mask word.

The Y and Z designators and G bits 0 through 6 are not used and must be zeros. G bit 7 equals 0 and 1 to search for equality and inequality, respectively. Registers A, B, and C are 64-bit quantities.

The A index (X) is incremented by one after each word searched not resulting in a match. However, if no match is found, the A index is increased by the length of the A field. When a match is found, the A index provides a means of locating the word of the reference field matching the contents of B. The leftmost bits of the index register are not altered.

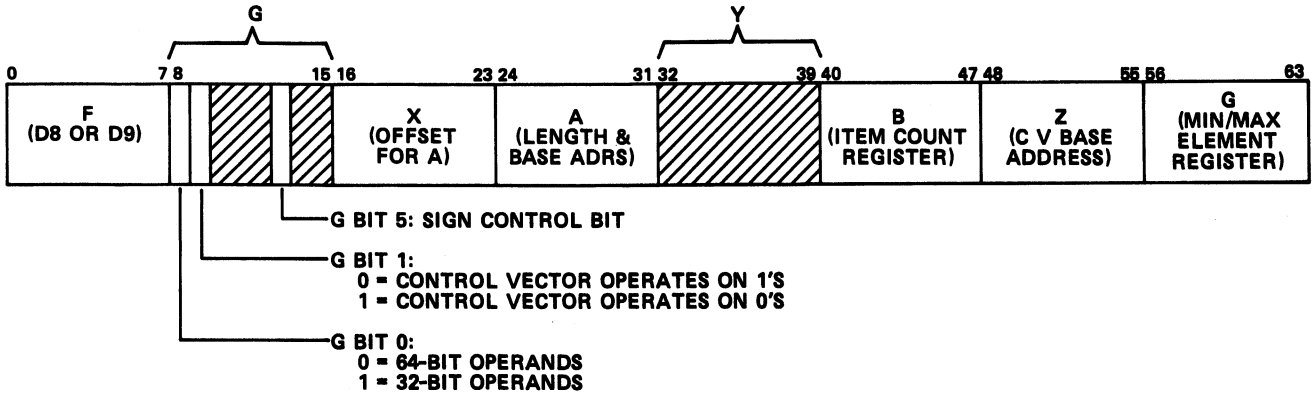
Index increments for mask binary compare.

Field	Data Flag Bit 37	Index Increment
A	1	Full Increment (No Match)
A	0	Partial Increment (Match)

Data flag: bit 37

D8 Max. of A to (C) Item Count to (B)

D9 Min. of A to (C) Item Count to (B)



These instructions search and compare the successive elements of vector A for the maximum/minimum element, using floating-point rules. The instructions then transmit the element to the register designated by C. The number of elements in vector A before, but not including, the maximum/minimum element is the item count which is stored in the rightmost 48 bits of a cleared register designated by B. The instructions terminate when vector A is exhausted.

If an indefinite element is encountered and examined, the register designated by C sets to indefinite and data flag bit 46 (indefinite result) sets. When this happens, the content of the register designated by B and data flag bit 54 is undefined.

The Z designator of the instructions provides the base address for a control vector. If used, the control vector determines which of the vector A elements the instruction compares. This is possible by the association of individual control vector bits with single elements of vector A. Only permissive control vector bits permit compares for their associated vector A elements. The instruction does not use an offset for the control vector.

Bit 0 of the G bits determines the size of the A operands and register C. Bit 5 of the G bits provides sign control. When bit 5 is set, the magnitude of the elements of A vector are compared. The unaltered element as read from A vector is stored in the register designated by C.

Applicable data flag bits are 46 (indefinite result) and free data flag bit 54.

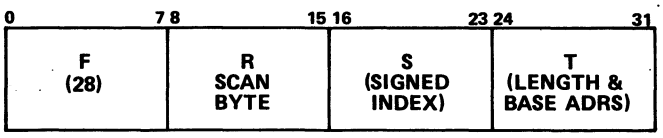
The Y designator and G bits 2 through 4, 6, and 7 of the G designator are undefined and must be zeros. Bit 5 provides sign control for vector A as described in Vector Instructions.

If the instruction specifies a control vector and the control vector contains no enabling bits, the instruction examines no elements of vector A, and the contents of register C becomes undefined. In this case, the item count in register B equals the field length of vector A minus the A offset.

If the instruction examines (enabling bit in control vector) an indefinite element, the instruction sets register C to indefinite and sets data flag bit 46 (indefinite result). In this case, data flag bit 54 is undefined. Data flag bit 54 is set when two maximum/minimum operands are encountered in the operand stream (multiple match). The index stored is for the first maximum/minimum operand of the pair.

The operands are compared by subtracting the current element of vector A from the next element of vector A and checking the result coefficient. If the result is not equal to zero, the maximum or minimum operand (depending upon the instruction) is used for the next compare with a new element of vector A. If the result is equal to zero, the element of A previously determined to be the maximum or minimum (depending on the instruction) is retained and used for the next compare. The relative positions of the elements within the vector dictate the order of the subtract. Since this type of compare operation is order dependent, † the final maximum or minimum can be affected by the order of the elements within the vector.

28 Scan Equal



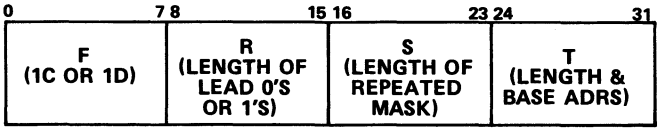
This instruction scans the bytes in field T, from left to right, until the scan operation locates the first byte equal to byte R, contained in the instruction word. The scan operation is indexed by the signed scan index, located in the rightmost 48 bits of the register denoted by S. When the operation locates the first equal byte, the instruction stops scanning and increments the scan index (S) by the number of bytes scanned before the equal byte was found. The leftmost 16 bits of the register are not altered.

The register specified by T contains the field length and base address of the source field in the leftmost 16 bits and rightmost 48 bits, respectively. Since the S index is an item count in bytes, it is left-shifted three places before being added to the base address.

The instruction sets data flag bit 53 if no equal byte is found, and the S index is incremented by the number of bytes in the T field. In this case, the instruction terminates when the entire source field is scanned.

1C Form Repeated Bit Mask with Leading Zeros

1D Form Repeated Bit Mask with Leading Ones



These left-to-right instructions form a repeated mask in field T. The mask consists of a string of zeros/ones followed by a string of ones/zeros. The repeated mask consists of one combined string of zeros and ones or ones and zeros as shown in figure 4-47. All length specifications shown in figure 4-47 are in bits.

---

†Appendix B describes floating-point compare rules.

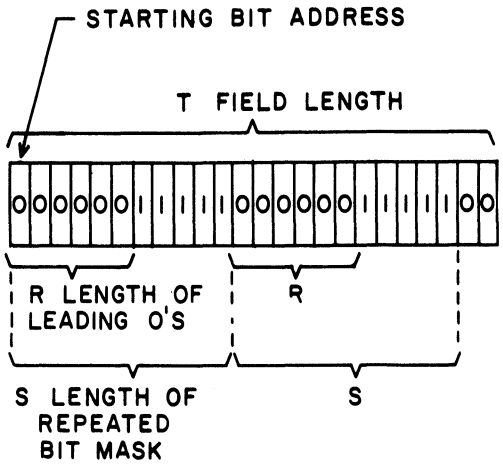
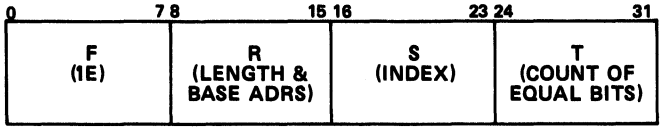


Figure 4-47. Example of Repeated Bit Mask Data Format (Leading Zeros)

The register specified by R (instruction format) contains the length of the string of zeros/ones in the leftmost 16 bits. The length of the repeated mask is contained in the leftmost 16 bits of register S. The rightmost 48 bits of registers R and S are undefined and require clearing before execution of the instruction. If the field length specified by the S register is zero, the instruction becomes a no-op. The register specified by T contains the length and starting bit address of the T field in the leftmost 16 bits and rightmost 48 bits, respectively. The instruction terminates when the T field is filled. If length R is equal to length S, a string of zeros (1C) or ones (1D) is formed. If length R is zero, a string of ones/zeros is formed.

1E Count Leading Equals R



This instruction scans the bits in field R, from left to right, until a bit unequal to the leftmost bit in the field is detected. The scanning operation starts with the bit immediately to the right of the leftmost bit in the field (figure 4-48). The instruction stores the count of the number of bits equal to the leftmost bit of the binary field in the rightmost bits of the register designated by T. The entire T register is cleared before the count is stored into it.

The register designated by R contains the length (in bits) and the base address in the leftmost 16 bits and rightmost 48 bits, respectively. Register S contains an index (in bits) which is added to the base address to form the starting address of the field. The instruction terminates when it either detects a bit unequal to the leftmost bit in the field or scans the entire field. In the latter case, the instruction stores a count equal to the field length minus one. In figure 4-48, a count of B_{16} is stored in register T.

The instruction sets data flag bit 53 if the leftmost bit of the binary field is a one.

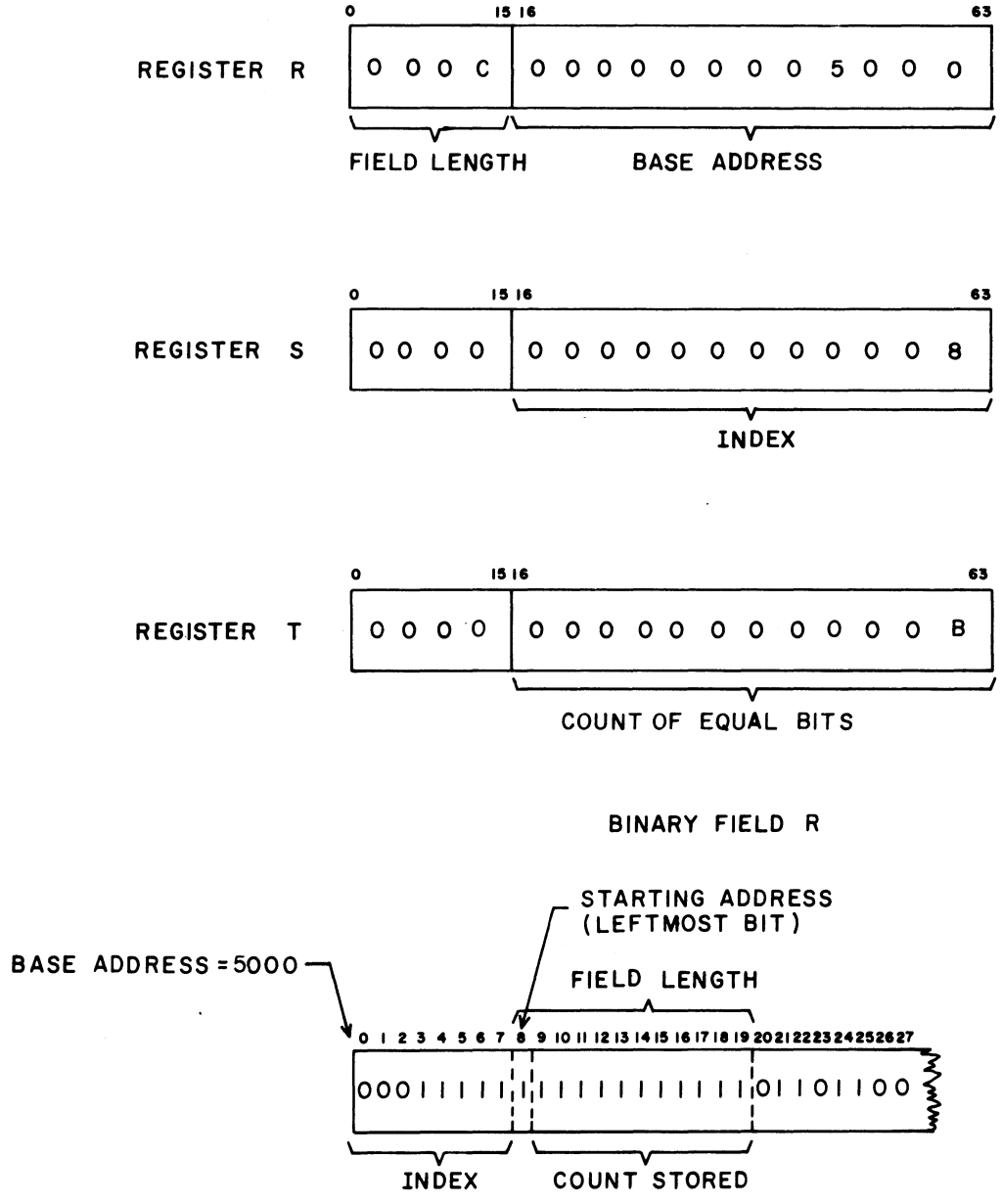
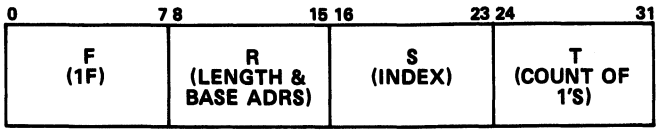


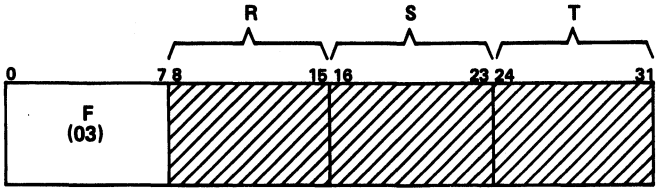
Figure 4-48. Example of Count Leading Equals Data and Register Format

1F Count Ones in Field R, Count to T



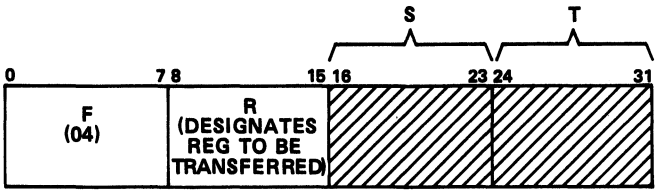
This instruction scans left to right, counts the number of binary ones in field R, and transmits this count to the rightmost bits of the register specified by T. The entire T register is cleared before receiving the count of ones. The register specified by R contains the length and base address of the R field in the leftmost 16 and rightmost 48 bits, respectively. The rightmost 48 bits of register S contain an index (in bits), which the instruction adds to the base address to form the starting address of the R field. The instruction terminates when all bits in the R field have been scanned.

03 Keypoint - Maintenance



This instruction is a one-cycle no-operation instruction. The designators R, S, and T are undefined and must be set to zero.

04 Breakpoint - Maintenance



The breakpoint instruction is a special instruction reserved as a maintenance and program debugging aid. This instruction transfers the content of the 64-bit register designated by R into the breakpoint register. The format of the breakpoint register is shown in figure 4-49. The breakpoint register is initially loaded from the invisible package of a job.

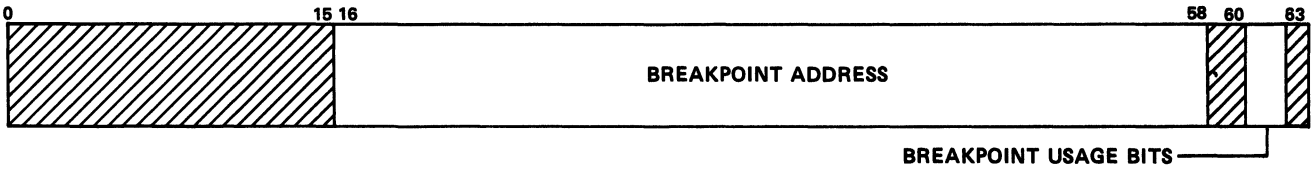


Figure 4-49. Breakpoint Register Format

The breakpoint function compares the addresses of specified categories of requests with the breakpoint address. If a match occurs, bit 47 of the data flag branch register is set.

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

61 - Breakpoint on CPU write operands

62 - Breakpoint on CPU read operands

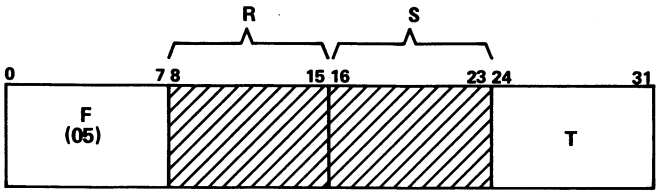
In job mode, virtual addresses are compared with the breakpoint address, while in monitor mode, absolute addresses are compared.

Since the monitor program has no invisible package, the breakpoint register is cleared by a job to monitor exchange. The monitor program may then reload the breakpoint register.

The breakpoint logic nominally resolves addresses to half-word fineness. However actual resolution is blunted in direct proportion to the amount of storage activated by a particular memory request. This amount is a function of the CPU model and of the type of operation which the CPU is performing.

The breakpoint specification of each job is saved in the invisible package.

05 Void Stack and Branch



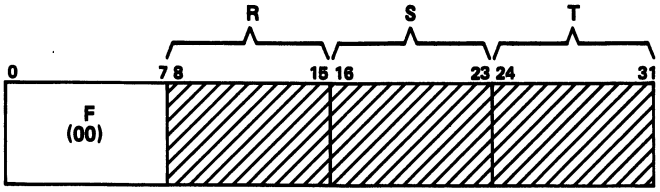
This instruction voids the instruction stack and branches out-of-stack to the contents of the register designated by T. The designators R and S are undefined and must be set to zero.

Programmer note: This instruction should be used immediately after an instruction which stores modified code to ensure that the code executed is the modified code.

MONITOR INSTRUCTIONS

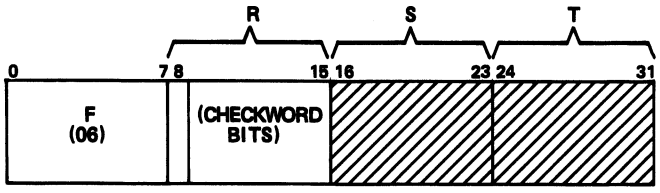
The monitor instructions function only during the monitor mode of operation. When the machine is in the job mode, the attempt to execute a monitor instruction is detected in the same way as an attempt to execute an undefined instruction code. The result of such an attempt is that the function code (F) and virtual program address (P) of the current instruction are stored in the appropriate positions of the invisible package. The machine then exchanges to the monitor program starting at the address contained in register 03. Refer to section 5 for a more detailed description of job to monitor exchange operations.

00 Idle



If in the monitor mode, this instruction enables the external interrupt and halts program operation until an external interrupt occurs. The R, S, and T designators are not used and must be zeros.

06 Fault Test - Maintenance



This instruction is used to complement checkword bits on the scalar write bus so the read SECDED circuitry may be checked. It can also be used to disable the error correction circuitry on all read buses allowing data to be passed through the SECDED hardware without any correction taking place.

In this test, R designator bits 9 through 15 are selected to complement the checkword bits of half-words 0, 1, 2, and 3 on the write scalar bus to central memory. By selection of data bits and complementing checkword bits, SECDED fault generation on all read buses is possible allowing complete checking of the read SECDED hardware and the fault recording hardware for type and address of fault.

The forced complementing of the checkword bits is discontinued by executing the instruction with bit 9 through 15 of the R designator set to 0.

This instruction is enabled during monitor mode only; in job mode it is a no-op.

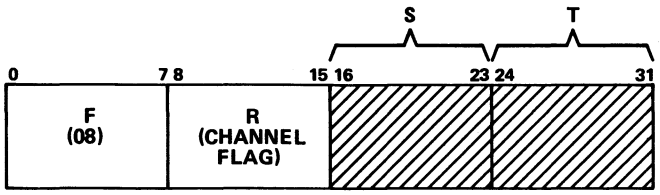
The modes are set by executing the instruction with 1 in the appropriate R designator bit and cleared by a 0 in the same bit location. Table 4-29 shows the R designator bit definitions.

TABLE 4-29. R DESIGNATOR BIT DEFINITIONS

R Designator Bit	Definition
8	Disable error correction on all read buses.
9-15	Checkword bit to be complemented.

This instruction must be executed with the R designator bits set to zero before any monitor to job exchange operation. If these bits are not set to zero via this instruction, the connection network could produce invalid data on the read and invalid data could be written into memory.

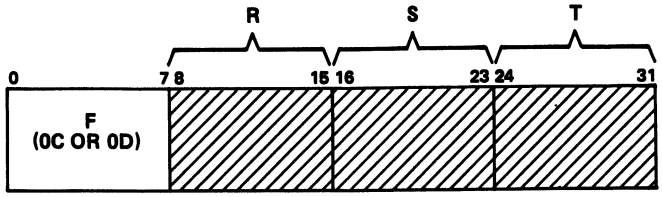
08 Input/Output Per R



In the monitor mode, this instruction sets the channel flag bit in the I/O channel designated by the hexadecimal code in the designator R. The setting of this bit indicates that the CPU has stored data at a predetermined location in central storage for the designated channel. The corresponding I/O channel then processes the stored data. If the R designator specifies a nonexistent channel other than I/O 1 through 16, the instruction becomes undefined. The S and T designators are not used and must be zeros.

0C Store Associative Registers

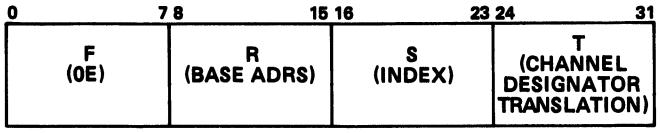
0D Load Associative Registers



These instructions store (0C)/load (0D) the contents of the 16 associative registers into/from absolute addresses of central storage beginning at 4000_{16}. The transfer is an ordered operation; thus, associative register 0 transfers to/from address 4000_{16}, and so forth. The content of the associative registers are undefined following the execution of the 0C instruction.

The R, S, and T designators are not used and must be zeros.

0E Translate External Interrupt



This instruction translates the lowest numbered bit set in the external interrupt register (EIR) into its associated, 5-bit code and transmits the code to the rightmost 5 bits of the register designated by T. The leftmost 59 bits of register T are cleared to zeros. If only one bit in EIR is set, the program branches to the address formed by the sum of the content of the registers designated by S and R. The rightmost 48 bits of register S contain an index in half-words and the rightmost 48 bits of register R contain the base address. If more than one bit in EIR is set, the program executes the next instruction.

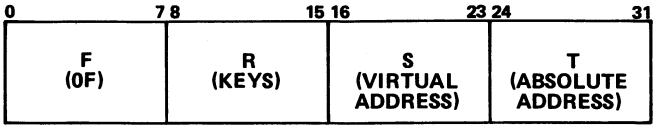
Whether the branch condition is met or not, the instruction clears the EIR bit corresponding to the channel designator that was transmitted to register T. If the T and S designators are equal, the interrupting channel designator is the branch index.

If no bit in EIR is set, the instruction clears register T and performs no branch operation. Bit zero of EIR is never set since this bit is reserved for maintenance purposes.

Each bit in the EIR is associated with one of the I/O channels or the monitor interval timer. The EIR bit assignments are as follows:

Bits	Assignments
0	Not available
1-16	I/O channels 1 through 16
17	Monitor interval timer

0F Load Keys from (R), Translate Address (S) to (T)



This instruction loads the four keys found in the register designated by R into the virtual address key registers. The instruction then translates the virtual address† located in the rightmost 48 bits of register S into an absolute bit address, using the four keys loaded from R and the associative words from the page table. The resulting absolute bit address is transmitted to the rightmost 48 bits of the register designated by T. If no translation is possible before the end of the page table is reached, the instruction clears the rightmost 48 bits of register T. The leftmost 16 bits of register S are transmitted to the corresponding portion of register T. The associative word used to make the translation is placed at the top of the page table (associative register 0). The instruction moves the position of the associative words down in the page table, if necessary, when searching for the associative word used to make the translation. The 3-bit usage code in the associative word is not altered by this instruction. Figure 4-50 shows the formats for the R, S, and T registers as they are used for this instruction.

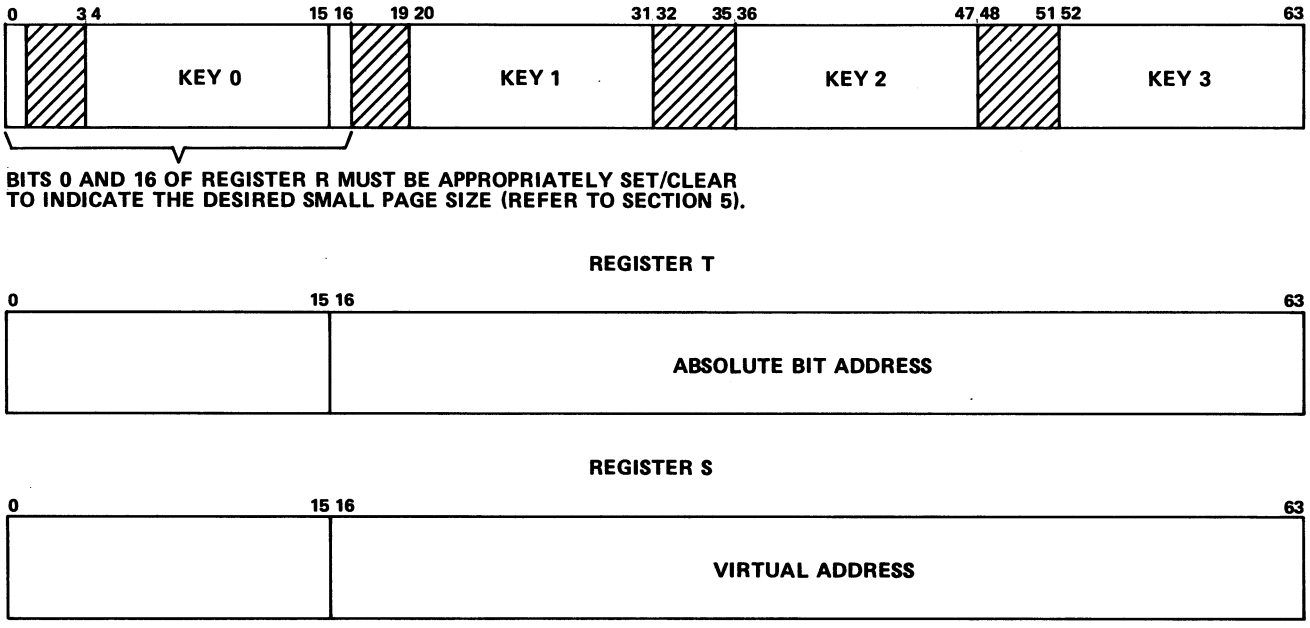
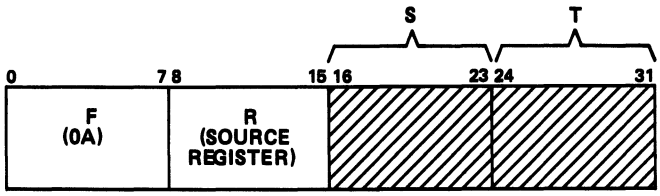


Figure 4-50. Register Formats for the 0F Instruction

†Virtual addressing operation is described in section 5.

0A Transmit (R) to Monitor Interval Timer



In the monitor mode, this instruction transmits bits 32 through 63 of the 64-bit register specified by the R designator to the monitor interval timer. The function of the monitor interval timer is described in section 5. The leftmost 32 bits of register R are ignored.

Loading the monitor interval timer with zeros deactivates the timer without setting bit 17 of the external interrupt register.

PROGRAMMING INFORMATION 5

GENERAL

This section describes the various registers and operations of the central computer that are of particular interest to the programmer. Included are descriptions of job and monitor modes, interrupts, the invisible package, addressing modes, real-time counters, the register files, the data flag branch register, addressing modes, and general definitions and programming guides.

MONITOR AND JOB MODES

The central processor unit (CPU) operates in one of two programming modes.

1. Monitor mode.
2. Job mode.

The CPU automatically exchanges the job mode for the monitor mode when it receives an interrupt or when a job program executes an exit force (09) instruction. The monitor mode disables all interrupts and virtual addressing† and permits absolute addressing†† to central storage. Any interrupts that occur during the monitor mode temporarily store until the monitor program executes an idle (00), translate external interrupt (0E) or an exit force (09) instruction. The idle instruction causes the CPU to wait until an interrupt occurs. The exit force (09) instruction switches the CPU to the job mode and starts executing the selected job program. Switching to the job mode enables the interrupts and virtual addressing.

The purpose of the exchange is to change the prime role of the CPU. In job mode, job tasks are performed. In monitor mode, the system decisions are made and the page table is altered.

Some instructions in progress may be interrupted prior to their completion. The flags stored in the invisible package are used to restart the interrupted instruction exactly where it left off.

EXCHANGE FROM MONITOR MODE TO JOB MODE

This is always accomplished with an exit force (09) instruction. The monitor program must set up the invisible package for the job prior to changing modes for that job via the exit force (09) instruction. The exit force operation is as follows:

1. The 256-word register file for monitor is stored into absolute memory locations 0 through $3FC0_{16}$ (bit address). The register file for the job is loaded from the job's virtual memory locations 0 through $3FC0_{16}$. Any job mode reference to this area of a job's virtual memory causes the executing instruction to be treated as an illegal instruction. The absolute bit address of the job's virtual page zero is in the monitor's register S specified by the exit force instruction. This address is placed in a register for storing the register file on return to monitor mode.

†Absolute and virtual addressing are described later in this section.
††The invisible package is described in detail later in this section.

2. The CPU's major control registers and flags are loaded from the invisible package which is located starting at the absolute bit address in the monitor's register T specified by the exit force instruction. This starting address is saved in a register to provide for storing the current invisible package when returning to the monitor program.

3. The CPU's mode is changed from monitor mode to job mode. This enables the virtual address mechanism and the interrupts.

4. The contents of P (program address register) is then read using virtual addressing, for the next instruction to be issued. If the invisible package contains information for the restart of an interrupted instruction, execution of that instruction is restarted. The instruction from the program address register is issued and executed if no restart of a vector or string instruction is located, or if parallel (scalar and vector/string) operation is permitted on a restart.

ILLEGAL INSTRUCTION IN MONITOR MODE

If an attempt is made by the monitor program to perform an illegal instruction code, an automatic branch is made to the absolute address contained in the monitor's register 4. This hardware trap is to aid in the debugging of the monitor software and to trap some hardware failures. This trap is not to be utilized by the monitor software as a normal branch.

If an illegal instruction code is issued in job mode, an exchange to monitor mode is performed with monitor mode exectuion beginning at the address specified by the contents of absolute register 3.

Any reference to the monitor or job mode register file via an absolute or virtual bit address will be treated as if an illegal instruction had been performed. This hardware trap is an aid to trap some hardware failures and software programming problems. This trap is not to be utilized by job mode software as a normal branch, or an early terminate of a vector/string instruction may occur due to the exchange.

EXCHANGE FROM JOB MODE TO MONITOR MODE

The exit force (09) instruction, channel interrupt, and accesss interrupt are the three normal ways of getting from job mode to the monitor program in monitor mode. Attempting to execute either a monitor-type instruction in job mode or an illegal instruction is the fourth way into the monitor. Except for the starting point in the monitor program, the operations performed in getting to the monitor are identical for the four.

The operation is as follows:

1. The current invisible registers and flags are stored into the invisible package starting at the same address used to load the invisible package when the job was entered.

2. The 256-word register file for the job is stored in virtual memory locations 0 through $3FC0_{16}$. Absolute memory locations 0 through $3FC0_{16}$ are read into the register file.

3. The CPU is changed from job to monitor mode and the virtual addressing mechanism is disabled. Any external interrupts that occur after this point are honored only if the CPU executes an idle instruction. Otherwise, the interrupts are saved until the CPU reverts to job mode, or until the monitor program clears the interrupts with a translate external interrupt (0E) instruction.

4. The monitor program executes starting at the absolute address contained in the rightmost 48 bits of registers 3, 5, 6, or 7 in the monitor's register file. The method used to enter monitor mode determines the register selection. The address in the selected register transfers to the program address register (P register).

Method of Getting to the Monitor	Register in Monitor's Register File used for Starting Address (P Address)
1. Job mode illegal: Illegal instruction, monitor type instruction in job mode, or a reference to the register file as memory (bit address 0000 - $3FFF_{16}$)	Register 3
2. Monitor mode instruction: Illegal instruction in monitor or reference to the register file as memory (bit address 0000 - $3FFF_{16}$)	Register 4
3. Exit force	Register 5
4. External interrupt	Register 6
5. Storage access interrupt	Register 7

INTERRUPTS

Interrupts consist of two main types.

1. Storage access
2. External

The occurrence of either type of interrupt during job mode causes the CPU to switch to monitor mode. The monitor program then processes the interrupt.

During monitor mode, the interrupt system is disabled except during the idle (00) instruction. Any external interrupts that occur are stored until the CPU switches back to the job mode or until the monitor program clears the interrupts with the translate external interrupt (0E) instruction.

STORAGE ACCESS INTERRUPTS

A storage access interrupt occurs when a job mode program attempts to reference a central storage page that does not have a corresponding word in the page table. A storage access interrupt also occurs when a job mode program attempts a storage reference that violates the corresponding lockout code.

Any CPU storage reference can cause an access interrupt even if it occurs in the middle of a vector or string instruction. The virtual address of the reference causing the interrupt and bits indicating the reason for the access interrupt (cause bits) are stored in word address $xx...xxE_{16}$ of the invisible package for the corresponding job (figure 5-1). Refer to the invisible package explanation in this section.

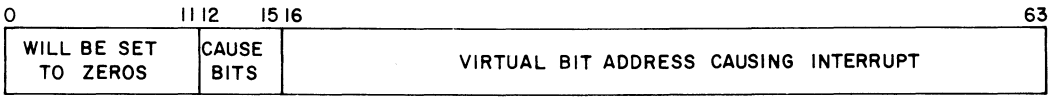


Figure 5-1. Invisible Package Word $xx...xxE_{16}$ Format for Access Interrupt

The condition of the cause bits indicate the type of storage reference that initiated the access interrupt is shown as follows:

Cause Bits				Type of Access Attempted
12	13	14	15	
0	1	0	0	Write operand violation
1	0	0	0	Associative word not in the page table
1	1	0	0†	Associative word not in the page table and reference attempted was a write operation
0	0	1	0	Read operand violation
0	0	0	1	Read instruction violation

Following the access interrupt, the CPU exchanges and switches to the monitor mode. The program then branches to the absolute address contained in the rightmost 48 bits of register 7 in the register file for the monitor program. The monitor program then allocates space for the requested page and/or procures the requested page directly. The monitor program can restart the job where it was interrupted by using the exit force (09) instruction. If the job is to be restarted, however, the monitor program must alter the page table and central storage to include the new page.

† This is the only case where more than one cause bit is set at one time.

EXTERNAL INTERRUPTS

Each input/output (I/O) channel and the monitor interval timer can interrupt the CPU by transmitting an interrupt signal on the assigned interrupt line. The interrupt signal sets the corresponding flag bit in the external interrupt register. The external line assignments are listed in table 5-1.

I/O CHANNEL INTERRUPT LINES

As shown in table 5-1, each I/O channel has an external interrupt line assignment. The transmission of the interrupt signal on the corresponding external interrupt line sets the corresponding external interrupt register flag bit. The setting of this bit indicates to the CPU that the I/O device (station) has stored a message in a predetermined location in central storage.

TABLE 5-1. EXTERNAL INTERRUPT LINES

External Interrupt Line	Assignment
0	Not available
1	I/O channel 1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	I/O channel 16
17	Monitor interval timer

MONITOR INTERVAL TIMER INTERRUPT

When the monitor interval timer (described in this section) decrements to a zero count, an external interrupt signal appears on line 17. The resultant setting of external register flag bit 15 indicates to the CPU that the specified period initially set in the monitor interval timer has elapsed, requiring processing by the monitor program.

INVISIBLE PACKAGE

The invisible package contains the address and control information necessary to begin a new job or to continue a job interrupted during execution in job mode. Each invisible package is associated with a job. The invisible package for a particular job is stored at 40 consecutive word addresses in central storage beginning at the initial address assigned by the monitor program. The invisible package is always stored starting at an even-numbered sword address. Therefore, the rightmost 10 bits of the starting address of the invisible package must be zeros. Refer to the exit force (09) instruction in the instructions section of this manual.

The monitor program must set up an invisible package for each job. There is no invisible package for the monitor program itself.

When the CPU switches from monitor to job mode, the invisible package for the corresponding job is automatically loaded into the appropriate CPU register from central storage beginning at the address assigned to that job.

When the CPU switches from job to monitor mode, as in an interrupt, the contents of the corresponding registers are automatically stored in central storage as the invisible package for that job.

If a job is to be reentered, the monitor should not alter the job's invisible package except for possibly the keys.

If the job is being initialized or reentered by changing the program address from the value it was at the time it returned to monitor mode, the invisible package has to be cleared and the program address and keys set. The setting of the breakpoint, data flag register, and job interval timer is optional.

Figure 5-2 shows the invisible package format.

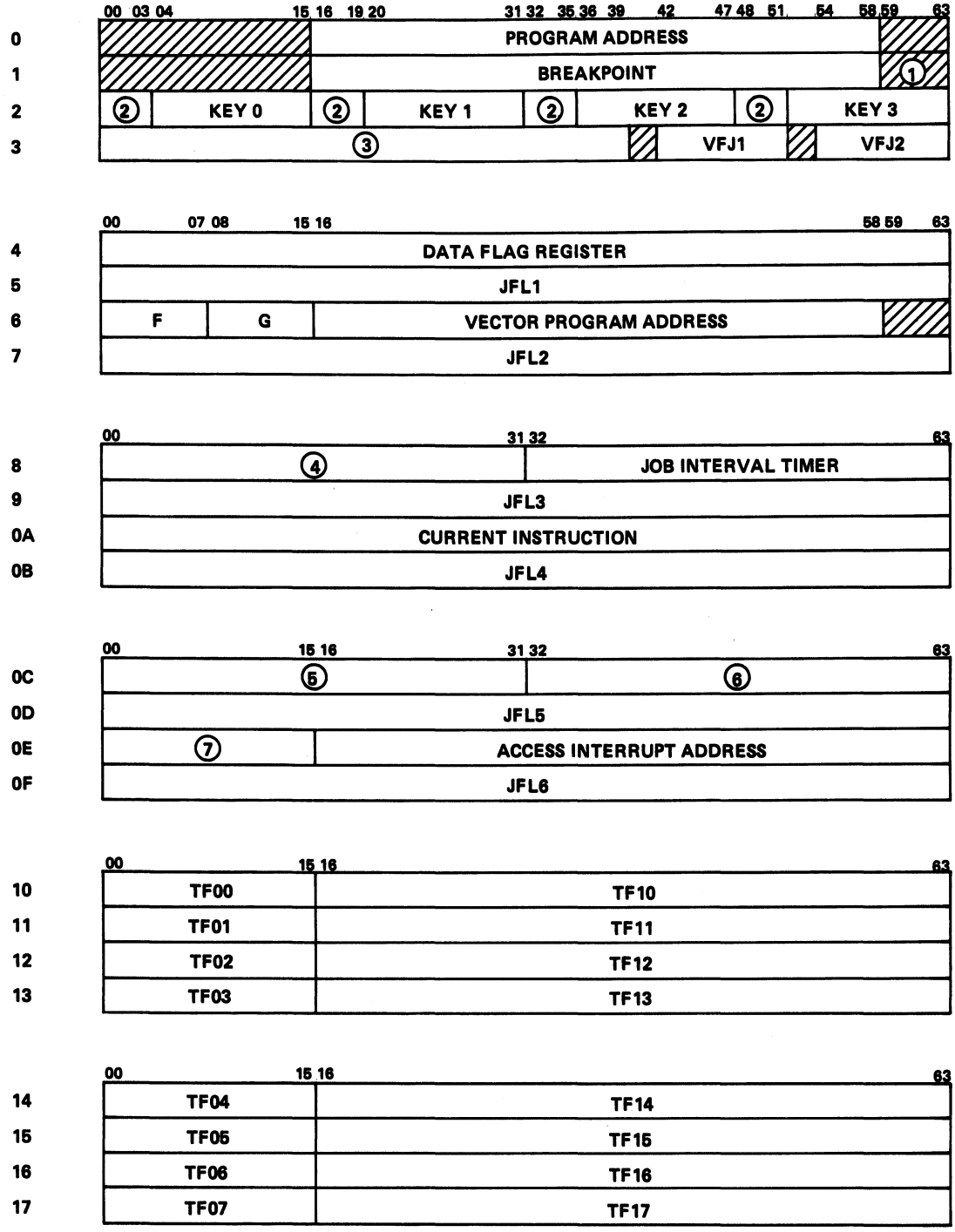


Figure 5-2. Invisible Package Format (Sheet 1 of 2)

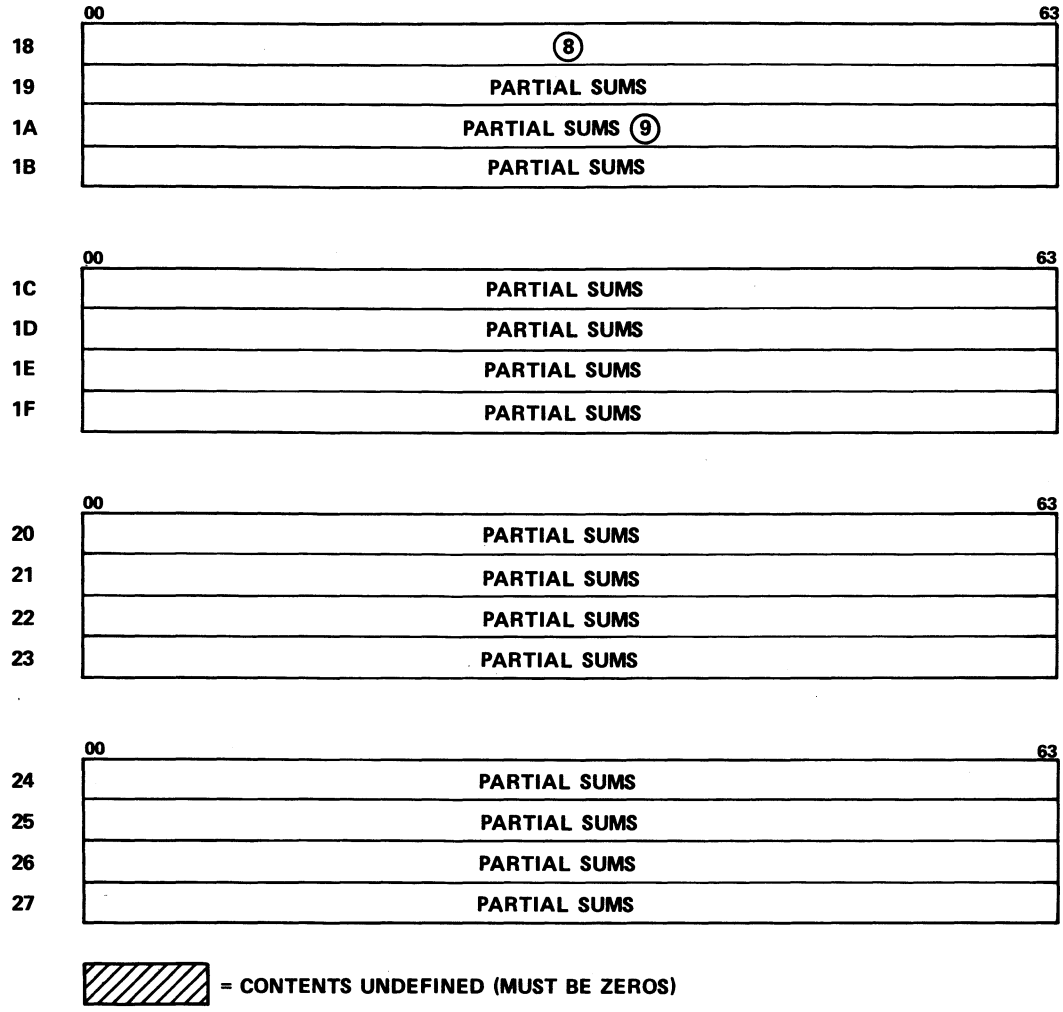


Figure 5-2. Invisible Package Format (Sheet of 2 of 2)

Each word is explained as follows:

- WORD 0

  Bits 00 through 15 are undefined and must be zeros.

  Bits 16 through 58 contain the program address.

  Bits 59 through 63 are undefined and must be zeros.

- Word 1

  Bits 00 through 15 are undefined and must be zeros.

  Bits 16 through 58 contain the breakpoint.

  (1) Bits 59 through 63 are the breakpoint usage bits and are defined as follows:

  Bits 59 and 60 are not used and must be zero.

  Bit 61 is check for breakpoint, compare on write operands.

  Bit 62 is check for breakpoint compare on read operands.

  Bit 63 is not used and must be zero.

- WORD 2

  Bits 04 through 15 contain key 0.

  Bits 20 through 31 contain key 1.

  Bits 36 through 47 contain key 2.

  Bits 52 through 63 contain key 4.

  (2) Usage lockout bits for each key:

  Bits 0 and 16 together define a small page size for all small pages; bits 32 and 48 are not used and are set to zeros.

  Bits 0 and 16

0	0	All small pages are 512 words.
0	1	Undefined.
1	0	All small pages are 2048 words.
1	1	All small pages are 8192 words.

  Bit 1, if set, locks out CPU write operations.

  Bit 2, if set, locks out CPU read operations.

  Bit 3, if set, locks out CPU instruction references.

- WORD 3

  (3) VEX microcode conditions are as follows:

  Bits 00 through 31 are not used and must be zero.

  Bit 32 is interrupt FF (signal to pipelines).

  Bit 33 is link instruction in execution.

  Bit 34 is link instruction R bit 3.

  Bit 35 is link instruction R bit 4.

  Bit 36 is CC instruction in execution.

  Bit 37 is not used and must be set to zero.

  Bit 38 is vector block scalar use of load/store registers.

  Bit 39 is flag 1.

  Bits 40, 41, 52, and 53 are undefined and must be zeros. Bits 42 through 51 and bits 54 through 63 contain VFJ1 and VFJ2. These are temporary address registers used by VEX microcode for restart and so forth.

- WORD 4

  Bits 00 through 63 contain the data register.

- WORD 5

  Bits 00 through 63 contain one of six load/store registers, JFL1.

- WORD 6

  Bits 00 through 07 contain the vector function code.

  Bits 08 through 15 contain the G bits.

  Bits 16 through 58 contain the vector program address.

  Bits 59 through 63 are undefined and must be zeros.

- WORD 7

  Bits 00 through 63 contain one of six load/store registers JFL2.

- WORD 8

  (4) Job/vector instruction status bits are defined as follows:

  Bit 00 is vector restart.

  Bit 01 is not parallel operation.

  Bits 02 through 11 are undefined and must be zeros.

  Bit 12 is stall bit (set for no data processed).

  Bit 13 is D8, D9, or DF execution started.

  Bit 14 is undefined and must be zeros.

  Bit 15 is EBCDIC when set, ASCII when clear.

  Bit 16 is SCR code bit 3 (exit at vector instruction termination).

  Bit 17 is select force of extension field length.

  Bits 18 through 19 are vector instruction register file update disable bits.

  Bit 20 is D8 or D9 multiple match flag.

  Bit 21 is string restart bit (old data flag).

  Bit 22 is undefined and must be zeros.

  Bit 23 is undefined and must be zeros.

  Bit 24 is undefined and must be zeros.

  Bit 25 is undefined and must be zeros.

  Bit 26 is R-record FF.

  Bit 27 is DA-DC toggle code bit 0.

  Bit 28 is DA-DC toggle code bit 1.

  Bit 29 is DA-DC toggle code bit 2.

  Bit 30 is undefined and must be zeros.

  Bit 31 is string invisible package data is valid.

  Bits 32 through 63 contain the job interval timer.

- WORD 9

  Bits 00 through 63 contain one of six load/store registers JFL3.

- WORD 0A

  Bits 00 through 63 contain the current instruction.

- WORD 0B

  Bits 00 through 63 contain one of six load/store registers JFLA.

- WORD 0C

  (5) String partial data or function codes.

  Bits 00 through 31 are string partial data for bit result field instructions.

  (6) Bits 32 through 47 are link (56) instruction F and R codes (for information purposes only).

  Bits 48 through 63 are link F1 instruction F and G codes (for information purposes only).

- WORD 0D

  Bits 00 through 63 contain one of six load/store registers JFL5.

- WORD 0E

  (7) Access interrupt cause bits are defined as follows:

  Bits 00 through 11 are not used and must be zeros.

  Bit 12 is associative word not in page table.

  Bit 13 is write operand violation attempted.

  Bit 14 is read operand violation attempted.

  Bit 15 is read instruction violation attempted.

  Bits 16 through 63 contain the access interrupt address.

- WORD 0F

  Bits 00 through 63 contain one of six load/store registers JFL6.

- WORDS 10 through 17

  Words 10 through 17 contain the vector instruction interrupt counter.

  Bits 00 through 15, TF00 through TF07 contain the field length.

  Bits 16 through 63, TF10 through TF17 contain the address.

- WORDS 18 thrugh 27

  Words 18 through 27 contain the partial sums.

  (8) Partial sum or ninth IC:

  Bits 00 through 63 are the partial sum for DX instructions or special broadcast quantity for link or CC instruction.

  Bits 00 through 15 are output item count for AX or C8 through CB instructions.

  Bits 16 through 63 are C base address for AX instructions. (These bits are undefined in all other applications.)

  (9) Pipelines function control for link instructions.

ADDRESSING MODES

The computer system uses two modes of addressing central storage.

1. Virtual addressing.
2. Absolute addressing.

VIRTUAL ADDRESSING

Virtual addressing provides an efficient, dynamic method of allotting portions of central storage to each job mode program by the monitor program. Virtual addressing is used exclusively when the CPU is in job mode. The switching of the CPU to monitor mode automatically disables virtual addressing. However, central storage recognizes all addresses as being absolute. Thus, the virtual addressing control circuits convert virtual addresses to the corresponding absolute addresses.

Pages

Portions of central memory are logically partitioned into pages; the central computer has small and large page sizes. A small page contains either 512, 2048, or 8192 64-bit words selected by bits 0 and 16 in the third word (keys) of the invisible package. The bits are interpreted as follows:

Bits		
0	16	Description
0	0	Small pages are 512 words.
1	0	Small pages are 2048 words.
1	1	Small pages are 8142 words.
0	1	Undefined.

Only one small page size may reside in the associative page table. The default size is 512 words. A large page contains 65 536 64-bit words.

The monitor program allocates a page or pages to each job program. All of the words in a page are identified by a common page identifier. The common page identifier is an absolute address which locates the page in central memory.

Virtual Address Format

Table 5-2 shows the page size and the virtual page and word identifiers' bit sizes for each word page. This difference results from the number of bits needed to locate the word in the page. Figure 5-3 shows the virtual address formats for the 512-, 2K-, 8K-, and 65K-word pages, respectively. Note that the size of the virtual page identifier varies depending on the word page size.

The bit, byte, half-word, and word identifier portions of the virtual address are absolute. Thus, when the virtual page identifier is converted into an absolute page identifier, these portions of the virtual address are substituted directly into the absolute address.

TABLE 5-2. PAGE SIZE SPECIFICATION

Page Size	Virtual Page Identifier (Bits)	Word Identifier (Bits)
512	33	9
2 048	31	11
8 192	29	13
65 536	26	16

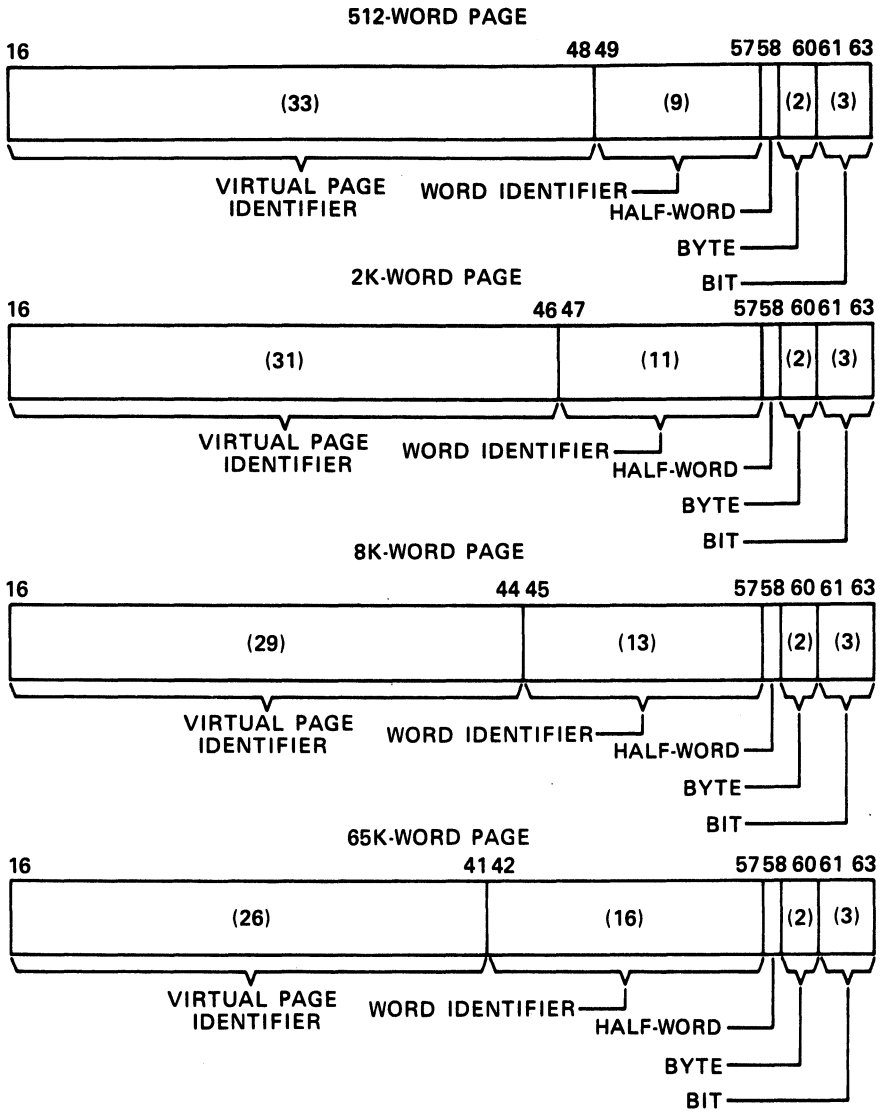
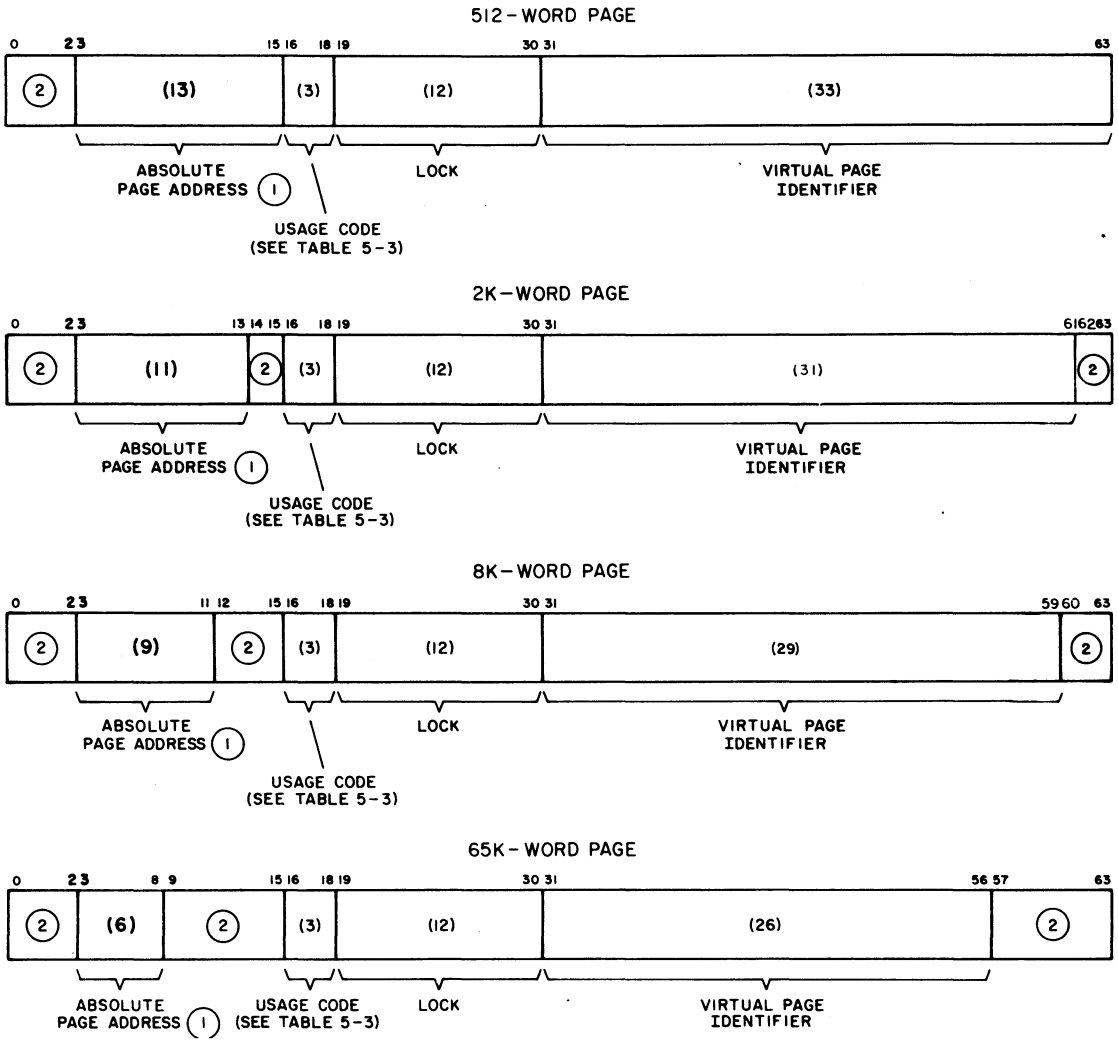


Figure 5-3. Virtual Address Formats

Associative Words

The associative words contain the information necessary to convert a virtual address into an absolute address. The monitor program must assemble the associative words into a page table as necessary for a given run. Figure 5-4 shows the formats of the associative words for the 512-word page, 2K-word page, 8K-word page, and 65K-word page, respectively.



(1) If 1000K word total central storage is used, bits 3 and 4 must be zeros.

(2) Bits not used.

Figure 5-4. Associative Word Formats

If a page has been referenced with code bits in table 5-3, a job program has made at least one storage reference to the page defined by the associative word. If a page is altered, a job program has performed a write operation on at least one bit in the page defined by the associative word. In the monitor mode, the CPU does not use the associative words in addressing. Thus, alteration or referencing storage by the monitor program is not recorded in the associative words.

TABLE 5-3. ASSOCIATIVE WORD USAGE CODES

Code Bits (16 17 18)	Definition
000	End of page table
001	Null associative word
010	Small page has not been referenced by the CPU
011	Large page has not been referenced by the CPU
100	Small page has been referenced by the CPU
101	Large page has been referenced by the CPU
110	Small page has been altered by the CPU
111	Large page has been altered by the CPU

Lock

A lock is a 12-bit quantity contained in each associative word (figure 5-4). The lock associates a page of central storage with a job program or several job programs.

Keys

The monitor assigns four 12-bit keys to each job. The keys for a particular job are read from central storage as part of the invisible package for that job. The monitor program transfers the keys to the virtual address key register (figure 5-5). After the virtual page address portion of an associative word matches with the corresponding portion of a virtual address, one of the four keys for the job must match the lock in the associative word before the storage reference can take place.

Figure 5-5 shows that each key is associated with 4-bit lockout code. The setting of a particular bit in this code locks out the corresponding type of storage reference. Table 5-4 lists each bit of the lockout code and the type of storage reference locked out if the bit is set.

If a key matches the lock of an associative word for a particular storage reference, but the operation is disabled by the lockout code for that type of reference, a storage access interrupt takes place. A storage access interrupt causes an exchange to the monitor mode.

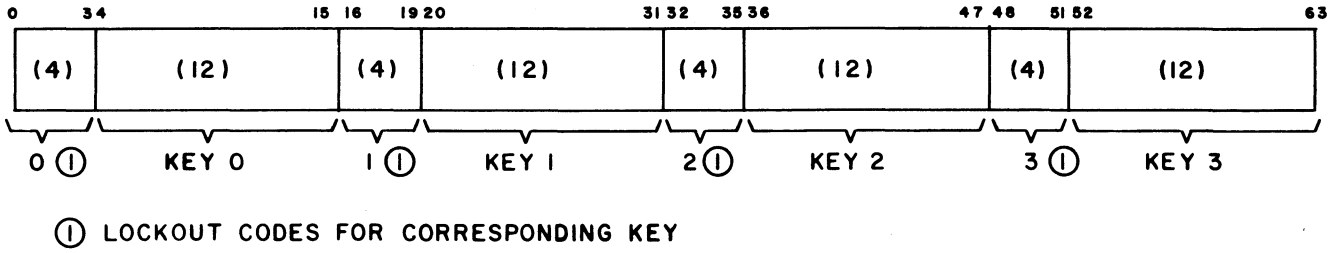


Figure 5-5. Virtual Address Key Register Format

TABLE 5-4. LOCKOUT CODES

Bit Position				Type of Storage Reference Locked Out
0	1	2	3	
1	X	X	X	†
X	1	X	X	CPU write operations
X	X	1	X	CPU read operations
X	X	X	1	CPU instruction references

NOTES

1. The actual bit number depends on the key field to which it corresponds (figure 5-5).
2. X denotes that the bit can be 0 or 1.

† Bits 0 and 16 define the small page size; bits 32 and 48 must be set to zero.

Page Table

The page table contains the complete list of associative words and includes both the associative registers and space table. The associative words contained in the page table define the pages currently allotted space in central storage. Figure 5-6 shows the format of the page table. Note that if the associative words in the associative registers are stored in central storage with the store associative registers (0C) instruction, they are stored in 16 64-bit storage locations beginning at absolute bit addresses 4000_{16}.

Table 5-5 lists page table restrictions and requirements.

TABLE 5-5. PAGE TABLE RESTRICTIONS AND REQUIREMENTS

Number	Restrictions and Requirements
1	There must be at least one END in the page table.
2	A page must appear only once in the page table. If a page appears more than once, the results are undefined, and the multiple match fault may be set.
3	Before looking at the page table, the ARs must be stored in memory. The page table, in memory, starts at address 4000_{16}.
4	Data words, after the end of table word in the same sword and in the sword following, may be altered during space table searches.

Associative Registers

The scalar processor contains 16 64-bit associative registers (ARs). Each AR contains one associative word. The ARs contain the first 16 associative words in the page table. For example, if the computer system consists of 1 million words of central storage and if only 65K-word pages are selected, the associative words for all 16 pages would be contained in the ARs. In the monitor mode, the contents of the ARs can be stored into or loaded from central storage with the store associative registers (0C) or load associative registers (0D) instructions, respectively.

The contents of the ARs cannot be referenced directly for read or write operations except through the 0C and 0D instructions.

Space Table

The space table (figure 5-6) consists of the locations in central storage containing the list of associative words that follow the words loaded into the associative register. The space table extends into central storage until an end-of-page table code is found in the usage bits (table 5-3) of the corresponding associative word. If no end-of-page table entry is found, the search hardware will loop, resulting in a CPU hang. Thus, the space table serves as an extension of the ARs to make up a complete page table.

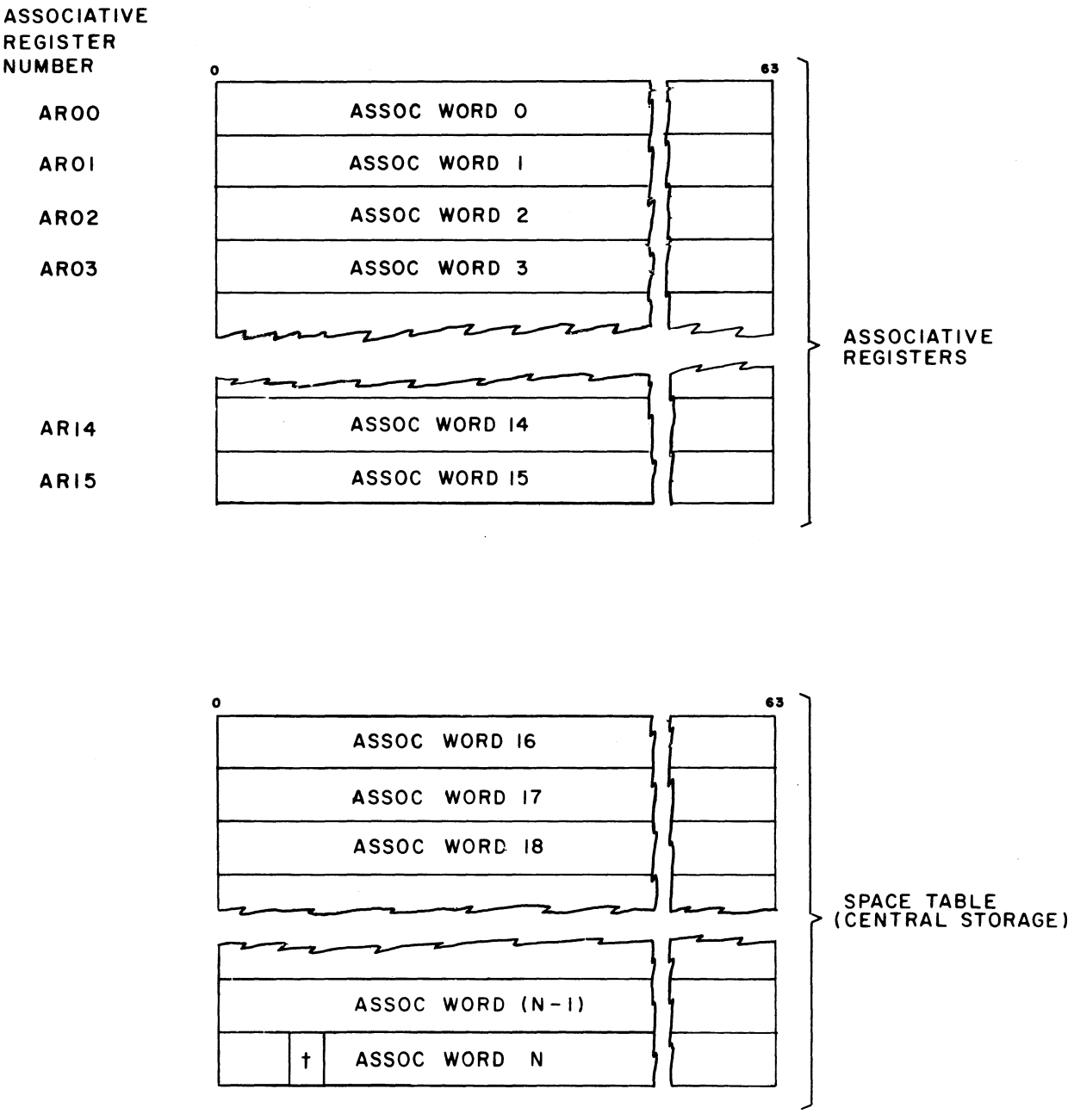


Figure 5-6. Page Table Format

Operation of Virtual Addressing

In the processing of a job mode program, each virtual address is processed by the scalar processor. The scalar processor compares the virtual page identifier in the virtual address (figure 5-3) with the corresponding portion of each associative word (figure 5-4) in the page table. If the virtual page identifiers match and the lock matches one of the four keys, a match condition occurs. If a match results, the absolute page address associated with the match-producing entry in the page table is combined with the applicable portion of the word identifier in the virtual address. The upper 19 bits of this combined address references one sword (eight 64-bit words) from central storage. If the end-of-the-page table is detected with no preceding match condition, or if a match results but the operation is disabled by the lockout code, a storage access interrupt results.

For a description of a page table search, refer to the scalar processor area of the central processor section of this manual (section 3).

ABSOLUTE ADDRESS

The absolute address formed by page table translation receives the page address portion from bits 3 through 15 of the associative word (figure 5-7). For 512 word pages, 13 bits (3 through 15) are placed in bit locations 36 through 48 of the absolute address allowing use of 8192 possible pages in job mode with 4-million-word memory size configuration. Bits 49 through 54 of the absolute address receive the corresponding bits from the virtual address (for a sword address). For 2K word pages, only 11 bits (3 through 13) are placed in bit locations 36 through 46 of the absolute address. Bits 47 through 54 of the absolute address receive the corresponding bits from the virtual address. For 8K word pages, only 9 bits (3 through 11) are placed in bit locations 36 through 44 of the absolute address. Bits 45 through 54 of the absolute address receive the corresponding bits from the virtual address. For 65K word pages, only 6 bits (3 through 8) are placed in bit locations 36 through 41 of the absolute address. Bits 42 through 54 of the absolute address receive the corresponding bits from the virtual address; this allows 64 large pages usable with a 4-million-word memory.

In a 4-million-word memory configuration, bit 36 of the absolute address indicates which upper or lower 2-million-word portion of memory is referenced. In a 1 million memory configuration, bit 38 of the absolute address indicates which upper or lower half-million portion of memory is referenced. If bit 3 or 4 of the absolute page address in the associative word is set for either page size of a 1 million memory configuration, the absolute address formed attempts to reference nonexistent upper words of memory. This type of memory reference is undefined, and an illegal reference to memory occurs. In a 2-million-word memory configuration, if bit 3 of the absolute page address in the associative word is set for either page size, the absolute address formed attempts to reference nonexistent upper words of memory. This type of memory reference is undefined, and an illegal reference to memory occurs.

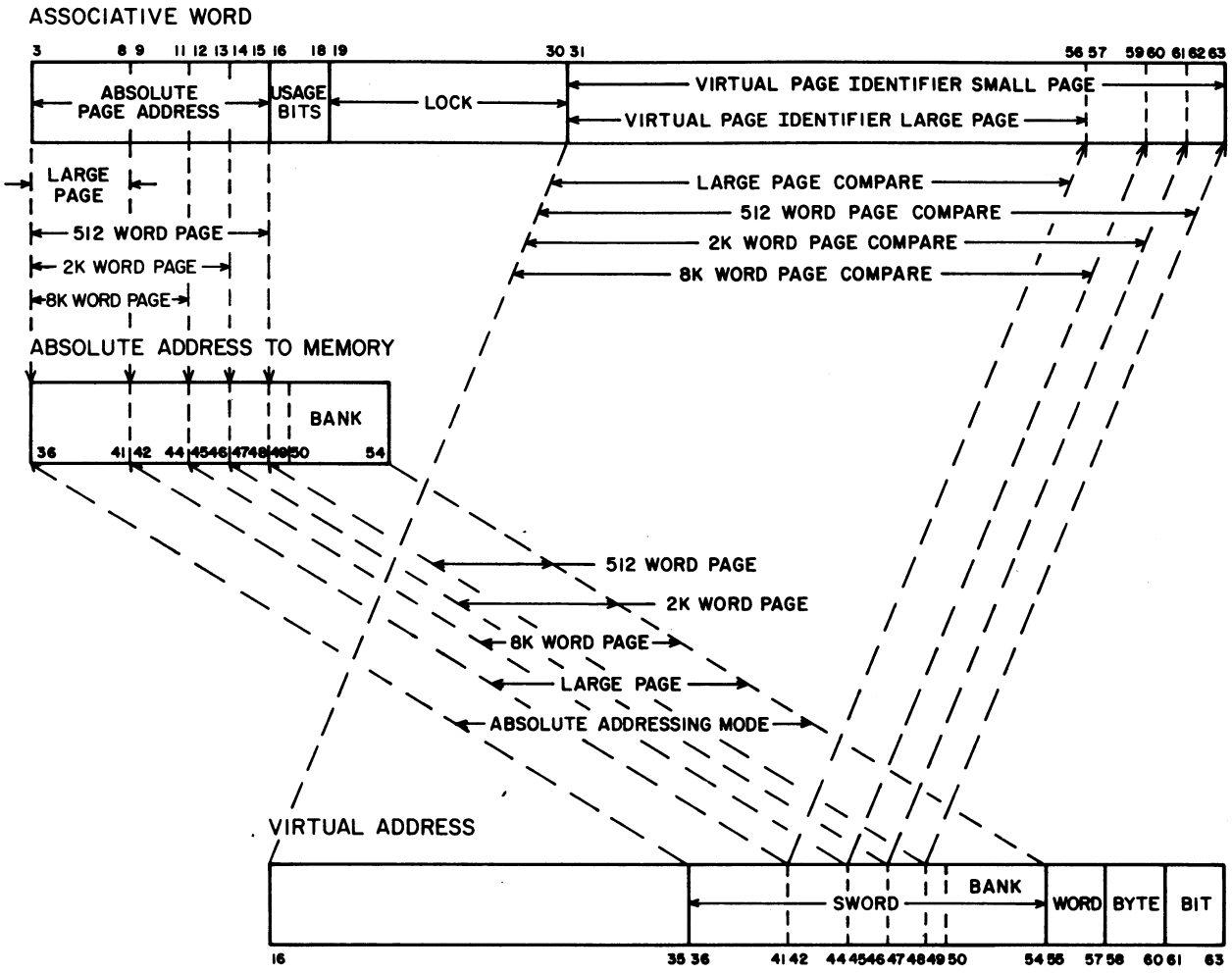


Figure 5-7. Virtual Address to Absolute Address

REAL-TIME COUNTERS

The CPU contains three counters that can be used for real-time programming applications.

1. Free-running clock.
2. Monitor interval timer.
3. Job interval timer.

Each of these counters is described in the following paragraphs.

FREE-RUNNING CLOCK COUNTER

This counter consists of a free-running 47-bit counter that is incremented at a 1-MHz rate, and a positive sign bit for a total of 48 bits. The free-running clock counter is never cleared. The contents of this counter can be stored in a designated register T with the transmit real-time clock to T (39) instruction.

MONITOR INTERVAL TIMER

This 32-bit counter is decremented at a 1-MHz rate. The transmit (R) to monitor interval timer (0A) instruction loads the contents of the designated register R into the monitor interval timer counter when the computer is in the monitor mode. The timer can be activated by loading it with any quantity other than all zeros. Once it is activated, the timer decrements at a 1-MHz rate until it reaches an all zero count. When the counter reaches a zero count, it causes an external interrupt on channel 17 which is processed like any other external interrupt. At this point, the timer is deactivated until it is loaded with some value other than zero.

The monitor interval timer is deactivated by any one of the following three methods.

1. Master clear.
2. Loading it with all zeros.
3. Decrementing it to a zero count. (When decremented to all zeros and causes an external interrupt, it is inactive until loaded with some value other than zero.)

The monitor interval timer count is decremented by a one count 1 microsecond after it is loaded with a nonzero count.

JOB INTERVAL TIMER

This 32-bit counter is decremented at a 1-MHz rate and can be loaded (job mode only) from a designated register R using the transmit R to job interval timer (3A) instruction. Once loaded, the job interval timer continues to decrement until either an exchange to monitor mode occurs, the timer decrements to zero, or the timer is loaded with zeros. If an exchange to monitor mode occurs, the job interval timer stops decrementing and the operation stores the current contents of the timer in the invisible package for that job. When the execution of that job resumes, the operation loads the job interval timer from the invisible package and resumes decrementing it. When the timer is decremented to zero, the CPU sets bit 36 in the data flag branch register. Refer to the data flag branch register description in this section.

Loading zeros deactivates the timer. This action does not set bit 36 of the data flag branch register. Master clear also deactivates the timer.

The job interval timer is deactivated by any one of the following three methods.

1. Master clear.
2. Loading it with all zeros.
3. Decrementing it to a zero count.

The job interval timer count is decremented by a one count 1 microsecond after it is loaded with a nonzero count.

REGISTER FILE

For register operations, the 8-bit instruction designators directly address the 256_{10} registers of the register file. During program execution (monitor or job), these registers reside in the CPU's register file. When an exchange operation occurs, the registers are stored into the first 256_{10} memory locations of the particular job or monitor mode program beginning at bit address zero (absolute address if in monitor mode and virtual if in job mode). The registers may not be referenced as memory by their associated monitor or job program. The only exceptions to this rule are the B7 and BA instructions with Gbit 7 set. (Refer to B7 and BA instructions in section 4 of this manual.)

Figure 5-8 shows a map of the register file and the relationship between the register, its storage address, and its 8-bit designator. The number on the right is the bit address and the number on the left is the value of the 8-bit designator for the 64-bit operand. The number inside the register represents the value of the 8-bit designator for the 32-bit operand. Note that any reference to 32-bit register one is undefined.

REGISTER FILE RESTRICTIONS

Certain registers within the register file have programming restrictions. The restrictions are grouped according to the instruction designator number of the register.

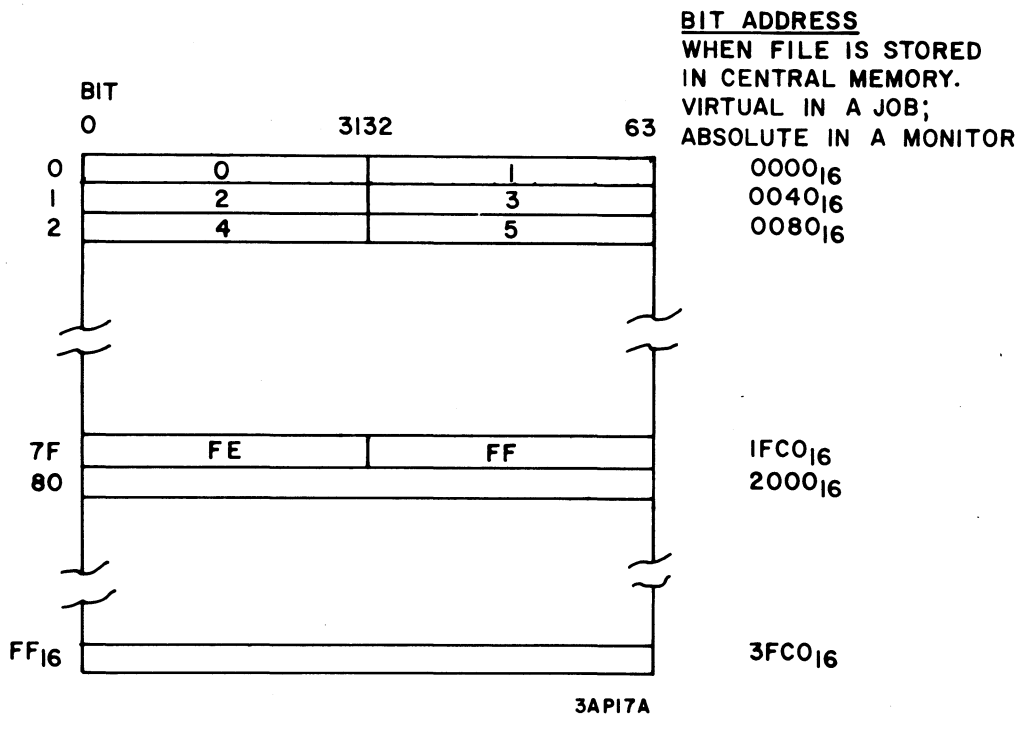


Figure 5-8. Register File

Register 0 (Trace Register) Restrictions

Register file address zero (figure 5-9) is used as the trace register in the 64-bit mode only. The trace register contains the address from which the most recent branch was taken. Register zero can be referenced by executing a 7D instruction. Refer to the instruction section for the mode of the 7D instruction which moves register zero to central memory. The maintenance station reads register zero by storing the register file and reading virtual/absolute zero in central memory. After a job to monitor exchange, the job's virtual address zero in memory contains the address of the last branch taken prior to the exchange operation. After a monitor to job exchange, monitor's address zero (absolute zero) contains the address of the last branch taken prior to the exchange operation. The BA instruction can also read register zero for data.

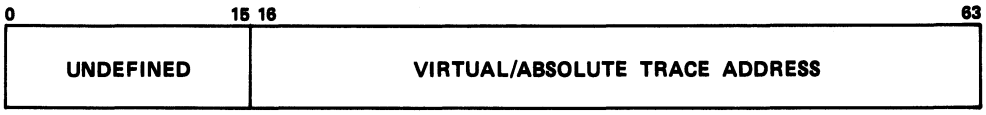


Figure 5-9. Virtual/Absolute Address Zero

Register 0 Content Resulting from an Exchange Operation

During a monitor to job exchange, the content of the trace register and the appropriate memory location for register zero exchange is as follows:

	Content Before Exchange	Content After Exchange
Absolute address zero	A	C
Virtual address zero	B	B
Trace register	C	B

During a job to monitor exchange, the content of the trace register and the appropriate memory location for register zero exchange (swap) is as follows:

	Content Before Exchange	Content After Exchange
Absolute address zero	A	A
Virtual address zero	B	C
Trace register	C	A

If monitor and job mode share a common register file (refer to common register files for job and monitor modes in this section), the following will occur upon a monitor to job or job to monitor exchange.

	Content Before Exchange	Content After Exchange
Absolute address zero	A	B
Virtual address zero	A	B
Trace register	B	B

Register 0 Content Resulting from a Swap (7D) Instruction

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

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

Register 0 when Referenced by an Instruction Designator

When referenced by an instruction designator, register zero provides machine zero as an operand except when used as a source register for a base address or other description for a vector or string instruction. In this case, register zero appears to contain 64 zero bits. The use of a zero address may cause the instruction to be treated as an illegal instruction. The use of a zero field length may cause the instruction to become undefined as when used in the A0 to AF instruction. If register zero is specified as the destination register, the instruction typically performs normally with data flags being set, if warranted, but no data is stored. Some instructions become undefined if register zero is specified as a destination register.

Table 5-6 shows which operand is obtained when register zero is specified for a source operand. To simplify the table, the specifying of register zero as a destination register is ignored since it causes the result to be lost. A blank in the table indicates that register zero cannot be specified or that register zero may only be specified as a destination register. The instruction designators R, S, T, G, X, A, Y, B, Z, and C are used for convenience, although they do not apply to all instructions. The following list contains definitions of symbols in the table.

Symbol	Result when Register Zero is Used as an Operand
A	All zeros are provided.
C	No control vector is used.
M	Machine zero is provided.
	$8000\ 0000\ 0000\ 0000_{16}$ 64-bit mode.
	$8000\ 0000_{16}$ 32-bit mode.
N	Instruction performs as a no-op.
O	A mask of all ones is provided.
Z	All zeros in the used portion. In this instance, the leftmost bit is not used; thus, machine zero and all zeros are undistinguishable.

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO

Op Code	Instruction Designators									
	R	S	T	G	X	A	Y	B	Z	C
00										
03										
04	Z									
05		Z								
06										
08										
09		Z	Z							
0A	Z									
0C										
0D										
0E	Z	Z								
0F	Z	A								
10	M									
11	Z									
12	Z	Z								
13	Z	Z	Z							
14	A	A	A							
15	A	A	A							
16	A	A	A							
1C	A	A	A							
1D	A	A	A							
1E	A	Z								
1F	A	Z								
20	M	M	Z							
21	M	M	Z							
22	M	M	Z							
23	M	M	Z							
24	M	M	Z							
25	M	M	Z							
26	M	M	Z							
27	M	M	Z							
28		Z	A							
2A										
2B	M	Z								
2C	M	M								
2D	M	M								
2E	M	M								
2F		Z	Z							
30	M									
31	Z	Z	Z							

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators									
	R	S	T	G	X	A	Y	B	Z	C
32		Z	Z							
33			Z							
34	M	Z								
35	Z	Z	Z							
36		Z	Z							
37										
38	M									
39										
3A	Z									
3B	Z									
3C	Z	Z								
3D	Z	Z								
3E										
3F	Z									
40	M	M								
41	M	M								
42	M	M								
44	M	M								
45	M	M								
46	M	M								
48	M	M								
49	M	M								
4B	M	M								
4C	M	M								
4D										
4E	Z									
4F	M	M								
50	M									
51	M									
52	M									
53	M									
54	M	Z								
55	M	M								
56										
58	M									
59	M									
5A	M									
5B	Z	Z								
5C	M									
5D	M									

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators									
	R	S	T	G	X	A	Y	B	Z	C
5E	Z	Z								
5F	Z	Z	M							
60	M	M								
61	M	M								
62	M	M								
63	M	Z								
64	M	M								
65	M	M								
66	M	M								
67	M	Z								
68	M	M								
69	M	M								
6B	M	M								
6C	M	M								
6D	M	Z								
6E	M	Z								
6F	M	M								
70	M									
71	M									
72	M									
73	M									
74	M	Z								
75	M	Z								
76	M									
77	M									
78	M									
79	M									
7A	M									
7B	Z	Z								
7C	M									
7D	A	†	A							
7E	Z	Z								
7F	Z	Z	M							
80					Z	A ††	Z	A ††	C	A
81					Z	A ††	Z	A ††	C	A
82					Z	A ††	Z	A ††	C	A
83					Z	A ††	Z	A ††	C	A
84					Z	A ††	Z	A ††	C	A
85					Z	A ††	Z	A ††	C	A
86					Z	A ††	Z	A ††	C	A

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators									
	R	S	T	G	X	A	Y	B	Z	C
87					Z	A ††	Z	A ††	C	A
88					Z	A ††	Z	A ††	C	A
89					Z	A ††	Z	A ††	C	A
8B					Z	A ††	Z	A ††	C	A
8C					Z	A ††	Z	A ††	C	A
8F					Z	A ††	Z	A ††	C	A
90					Z	A ††			C	A
91					Z	A ††			C	A
92					Z	A ††			C	A
93					Z	A ††			C	A
94					Z	A ††	Z	A ††	C	A
95					Z	A ††	Z	A ††	C	A
96					Z	A ††			C	A
97					Z	A ††			C	A
98					Z	A ††			C	A
99					Z	A ††			C	A
9A					Z	A ††			C	A
9B					Z	A ††	Z	A ††	C	A
9C					Z	A ††			C	A
9D					Z	A ††	Z	A ††	C	A
A0					A	Z ††	A	Z ††	A	Z
A1					A	Z ††	A	Z ††	A	Z
A2					A	Z ††	A	Z ††	A	Z
A4					A	Z ††	A	Z ††	A	Z
A5					A	Z ††	A	Z ††	A	Z
A6					A	Z ††	A	Z ††	A	Z
A8					A	Z ††	A	Z ††	A	Z
A9					A	Z ††	A	Z ††	A	Z
AB					A	Z ††	A	Z ††	A	Z
AC					A	Z ††	A	Z ††	A	Z
AF					A	Z ††	A	Z ††	A	Z
B0					Z	M	Z	Z	Z	
B1					Z	M	Z	Z	Z	
B2					Z	M	Z	Z	Z	
B3					Z	M	Z	Z	Z	
B4					Z	M	Z	Z	Z	
B5					Z	M	Z	Z	Z	
B6				Z						
B7					Z	A	Z	A †	Z	A
B8					Z	A			C	A

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators									
	R	S	T	G	X	A	Y	B	Z	C
BA†††					Z	A		A		A
BB						A ††		A ††	A	Z
BC						Z			A	Z
BD						A ††		A ††	A	A
BE										
BF										
C0					Z	A ††	Z	A ††	C	
C1					Z	A ††	Z	A ††	C	
C2					Z	A ††	Z	A ††	C	
C3					Z	A ††	Z	A ††	C	
C4					Z	A ††	Z	A ††	A	
C5					Z	A ††	Z	A ††	A	
C6					Z	A ††	Z	A ††	A	
C7					Z	A ††	Z	A ††	A	
C8						A		A	C	Z
C9						A		A	C	Z
CA						A		A	C	Z
CB						A		A	C	Z
CC					Z	A		A		A
CD										
CE				Z						
CF					Z	A	Z	A ††	Z	Z
D0					Z	A ††	Z	A ††	C	A
D1					Z	A			C	A
D4					Z	A ††	Z	A ††	C	A
D5					Z	A			C	A
D8					Z	A			C	
D9					Z	A			C	
DA					Z	A			C	
DB					Z	A			C	
DC					Z	A ††	Z	A ††	C	
DF						M		M	C	A

TABLE 5-6. RESULTS FOR SPECIFIED REGISTER ZERO (Contd)

Op Code	Instruction Designators									
	R	S	T	G	X	A	Y	B	Z	C
F0					Z	A	Z	A	Z	A
F1					Z	A	Z	A	Z	A
F2					Z	A	Z	A	Z	A
F3					Z	A	Z	A	Z	A
F4					Z	A	Z	A	Z	A
F5					Z	A	Z	A	Z	A
F6					Z	A	Z	A	Z	A
F7					Z	A	Z	A	Z	A
F8					Z	A			Z	A

† Refer to the swap 7D instruction in section 6 of this manual.
†† If register zero is selected to broadcast a constant, machine zero is that constant.
††† The BA instruction can read register zero for data.

Registers 1 and 2 (64-Bit), 2 through 5 (32-Bit) Restrictions

If data flag branches are used, 64-bit registers 1 and 2 must be reserved exclusively for that function. Register 1 stores the data flag branch exit address and register 2 the data flag branch entry address. Refer to the data flag branch register description in this section.

Registers 0 through 7 (64-Bit), 0 through F (32-Bit) Monitor Mode Restrictions

In 64-bit mode, registers 0, 1, and 2 (or in 32-bit mode, registers 0 through 5) have the restrictions during monitor mode as previously described. In 64-bit mode, registers 3 through 7 (or in 32-bit mode, registers 6 through F) are used for the undefined instructions, exit force, external interrupt, and storage access interrupt entry points. Refer to the exchange from job mode to monitor mode description in this section.

Register 1 (32-Bit) Rightmost Half of 64-Bit Register 0

Any reference to 32-bit register 1 is undefined.

COMMON REGISTER FILE FOR MONITOR AND JOB MODES

Monitor and job modes have perfectly overlapping register files if monitor executes an exit force instruction (09) with either designator S or the contents of register S equal to zero. In an exchange from monitor to job mode, the monitor's register file is stored starting at absolute bit address zero. The job's register file is not loaded for a common register file; the monitor's register file remains in the registers unaltered.

When exchanging from job mode back to monitor mode, the job's register file is stored where it came from; in this case, starting at absolute bit address zero. The monitor's register file is not loaded for a common register file; the job's register file remains in the registers unaltered.

DATA FLAG BRANCH REGISTER

The data flag branch (DFB) register is a 64-bit register (figure 5-10) that provides the programmer with an automatic branching feature to a special subroutine for certain operands, results, conditions, and so forth. The DFB register eliminates the time penalty of explicitly checking for special programming conditions. The DFB register is stored in word four of the invisible package. If a condition previously selected to cause an automatic branch occurs during the execution of an instruction, the computer stores the address of the next instruction that would have been executed in the address portion of register 01, and branches to the address contained in register 02.

Many register instructions are executed in parallel, and there may be some uncertainty as to which instruction caused the data flag condition. The data flag set condition may have occurred during an instruction which was issued a number of instructions before the one just completed. A flag on a scalar register instruction (divide, square root, and convert BCD to binary) could have occurred 0 to 35 instructions earlier. A flag on the other register instructions could have occurred 0 to 5 instructions earlier. A flag on a vector instruction could allow the issue of many scalar instructions before the automatic data flag branch occurs. For other data flag branch limitations, refer to the discussion under Data Flag Branch Operation later in this section.

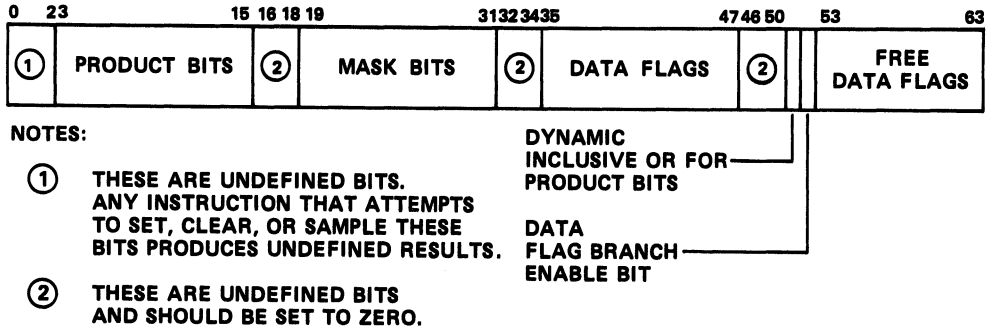


Figure 5-10. DFB Register Format

DATA FLAGS

Data flag bits are bits 35 through 47 of the DFB register. These bits indicate conditions that have occurred. For example, the CPU sets bit 37 at the end of a search for masked binary compare (CC) instruction if the operation detects no match. If a subsequent search for masked key instruction detects a match, the machine does not clear DFB bit 37. Bits 35 through 47 of the DFB register are cleared only by the data flag register bit branch and alter (33) and the data flag register load/store (3B) instructions.

Refer to appendix D for a complete list of data flag applications to instructions. Data flag bit 36 is applicable only to the job interval timer rather than a specific instruction and therefore not listed. Data bit 36 sets asynchronously. Note the CAUTION pertaining to the use of the 3B instruction in section 4 of this manual.

If a control vector (refer to Control Vector under Vector Instruction in section 6) is being used, the current control vector bit must be permissive for the operation to set any of the data flags. For example, if a divide fault occurs but the control vector bit for that result element is not permissive, that result element would not set the divide fault data flag bit.

Table 5-7 lists the data flag register bit assignments and associated mask and product bits described in the following paragraphs.

TABLE 5-7. DATA FLAG REGISTER BIT ASSIGNMENTS

Product Bit	Mask Bit	Data Flag Bit	Assignment/Description
0-2	16-18	32-34	Undefined and must be set to zeros.
3	19	35	Soft interrupt: Monitor software can set bit 35 of a job's DFB register while the register is stored in the job's invisible package. If, after exchanging back to job mode, bit 35 and its corresponding mask bit (19) are set, a normal data flag branch occurs.
4	20	36	Job interval timer.
5	21	37	Selected condition not met. Instructions C0 through C3. No match on instruction CC.
6	22	38	Undefined and must be set to zero.
7	23	39	The binary result exceeds the range of $+ [(2^{47}) - 1]$ to $- (2^{47})$ for the 10 instruction.
8	24	40	Bit 40 is the inclusive OR of bits 37, 38, and 39. Bit 24 masks bit 40. Bit 8 is the logical product of bits 24 and 40.

TABLE 5-7. DATA FLAG REGISTER BIT ASSIGNMENTS (Contd)

Product Bit	Mask Bit	Data Flag Bit	Assignment/Description
9	25	41	Floating-point divide fault: The divisor has an all zero coefficient or the divisor, as read from the register file or from central storage, is machine zero. If the divisor and/or dividend is indefinite, no divide fault exists. If a divisor causes a divide fault, the quotient is set to indefinite. The exponent overflow and result machine zero data flags are not set by a divide operation whose divisor caused a divide fault.
10	26	42	Exponent overflow: The exponent of the result is larger than 6FFF (6F for 32-bit arithmetic). Results are not checked for exponent overflow until after the exponent adjustment for normalization or significance has taken place. In the adjust exponent instructions, if a left shift exceeds the number of places required for normalization, this data flag bit is set. Exponent overflow causes the result to be set to indefinite; thus, the indefinite flag is always set on an exponent overflow. The exponent overflow data flag bit is not set if either source operand from central storage or the register file is indefinite or by a divide instruction whose division causes a divide fault.
11	27	43	Result machine zero: The exponent of the result returned to central storage or to the register file is less than 9000_{16} (90_{16} for 32-bit arithmetic). Exponent underflow causes the result to be set to machine zero. Results are not checked for exponent underflow until after the exponent adjustment for normalization is completed. This data flag bit is not set by a divide whose divisor causes a divide fault.
12	28	44	Bit 44 is the inclusive OR of bits 41, 42, and 43. Bit 28 masks bit 44. Bit 12 is the logical product of bits 28 and 44.

TABLE 5-7. DATA FLAG REGISTER BIT ASSIGNMENTS (Contd)

Product Bit	Mask Bit	Data Flag Bit	Assignment/Description
13	29	45	Square root result imaginary: A negative source operand was detected in a square root instruction. The square root of the absolute value of the operand is formed and the two's complement of this square root is stored as the result.
14	30	46	Indefinite result: An indefinite result was placed in central storage or into the register file. Bit 46 is also set if either or both operands of a floating-point compare were indefinite. An indefinite result may be caused by one or both operands of a floating-point arithmetic operation being indefinite or by the occurrence of either a divide fault or an exponent overflow.
15	31	47	Breakpoint: DFB bit 47 is set on the breakpoint instruction if breakpoint address and usage conditions are met. Applicable instruction: 04.
-	-	48-50	Undefined and must be set to zero.

MASK BITS

The mask bits are bits 16 through 31 of the DFB register. They select the conditions that cause the automatic data flag branch to occur when the selected condition takes place.

The 33 or 3B instruction sets and clears the mask bits. A mask bit need not be set for its corresponding data flag bit to be set when the condition occurs. The mask bits enable the setting of a corresponding bit in the product field when the associated masked data flag bit is set. A product bit is set regardless of the order the mask bit and its associated data flag bit are set.

PRODUCT BITS

Product bits are bits 0 through 15 of the DFB register. Each is the dynamic logical product of a data flag bit and associated mask bit being set. The computer executes a data flag branch when there is at least one bit set in the product field and the data flag branch enable bit is set.

DYNAMIC INCLUSIVE OR FOR PRODUCT BITS

The dynamic inclusive OR for product bits is bit 51 of the DFB register. This bit is set by setting any one of the product bits. It cannot be cleared directly.

SCALAR DIVIDE, SQUARE ROOT, CONVERT INSTRUCTION FLAG

This flag, bit 58 of the DFB register, indicates that one of the other data flags has been set by a scalar divide, square root, or convert instruction. The flag is cleared by the 33 or 3B instructions.

DATA FLAG BRANCH ENABLE BIT

The data flag branch enable bit is bit 52 of the DFB register. This bit must be set for an automatic data flag branch to take place. When bits 51 and 52 are both set, (setting may occur in either order) the data flag branch takes place. The computer automatically clears bit 52 when a data flag branch takes place. To reset the data flag branch enable bit, refer to the discussion under Data Flag Branch Operation later in this section.

FREE DATA FLAGS

Table 5-8 lists each of the free data flag bits and the corresponding assignments. There are no product or mask bits associated with the free data flag bits 53 through 63 of the DFB register.

TABLE 5-8. FREE DATA FLAG BIT ASSIGNMENTS

Free Data Flag Bit	Assignment	Applicable Instructions
53	Ones were counted.	Count leading equals (1E)
54	Undefined.	
55	Undefined.	
53	Whole field scan, no hit.	
54	Undefined.	Scan equal (28)
55	Undefined.	
53	Undefined.	Maximum (D8)
54	Multiple hits.	Minimum (D9)
55	Undefined.	

TABLE 5-8. FREE DATA FLAG BIT ASSIGNMENTS (Contd)

Free Data Flag Bit	Assignment	Applicable Instructions
53	Result field all zeros.	
54	Result field mixed.	Logical string (F0 through F7)
55	Result field all ones.	
56,57	Undefined and must be set to zero.	
58	A scalar divide/SQRT/convert operation set bits 39, 41, 42, 43, 45, and/or 46.	
59	Vector box floating-point divide fault, duplicate of bit 41 caused by a vector.	
60	Vector box exponent overflow, duplicate of bit 42 caused by a vector.	
61	Vector box machine zero result, duplicate of bit 43 caused by a vector.	
62	Vector box square root result imaginary, duplicate of bit 45 caused by a vector.	
63	Vector box indefinate result, duplicate of bit 46 caused by a vector.	

The DFB register bits 53 through 55 are cleared automatically by instructions using these bits prior to selectively setting them. A no-operation (no-op) instruction does not alter these bits. If applicable, bits 53 through 55 must be sampled before executing another instruction which would clear them. The setting of the bits does not cause a data flag branch operation.

The DFB register bits 58 through 63 assist software in determining what operation caused data flag bits 41, 42, 43, 45, and 46 to set. Bit 58 is set due to a scalar divide, square root, or convert instruction setting bits 39, 41, 42, 43, 46, or 46. Bits 59 through 63 are set due to a vector instruction setting bits 41, 42, 43, 45, or 46 respectively. (Scalar instructions do not set bits 59 through 63.) An automatic data flag branch that occurs with any of the bits 58 through 63 being set indicates that many scalar instructions may have been issued since the issue of the instruction causing the automatic data flag branch. This is due to the longer execution times of these instructions. To have the DFB register bits 58 through 63 remain useful to the programmer, the programmer must clear these bits when the corresponding bits causing the automatic data flag branch are cleared.

DATA FLAG BRANCH OPERATION

If a mask field bit and the associated data flag bit are set, the corresponding product field bit is set. Free data flag field bit 51 is also set since this bit is the dynamic inclusive OR of all bits in the product field. Under these conditions, the setting of bit 52 (data flag branch enable bit) initiates an automatic data flag branch operation.

The data flag branch operation begins immediately unless scalar instruction issue is stopped due to a register conflict or vector unit is executing a multipass instruction. The execution of the data flag branch transfers the bit address of the next instruction into the rightmost 48 bits of register 01 of the register file. A branch takes place to the bit address in the rightmost 48 bits of register 02. The data flag branch operation automatically clears bit 52 at this time. The data flag branch also clears the leftmost 16 bits of register 01.

NOTE

The clearing of bit 52 disables the data flag branch operation. Caution must be used to ensure that all data branch conditions are eliminated before resetting bit 52 or the program may enter a tight loop operation. The sampling of bit 51 for a zero before setting bit 52 prevents this situation in all cases except those involving the job interval timer.

When using the job interval timer, the setting of DFB bit 36 occurs asynchronously with respect to instruction execution once the job interval timer is loaded. Thus, the timer may set bit 36 after the check of bit 51 and before the branch to the content of register 01.

This situation can be programmed by examining the content of register 01 upon entering the routine for processing data flag branches. If register 01 indicates that the branch occurred outside the DFB routine, the content of register 01 could be transferred to a temporary storage location.

If register 01 indicates that the branch occurred within the DFB routine, the content of register 01 would not be transferred to a temporary storage location. At the end of the DFB routine, the program would branch to the content of the temporary storage location.

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

DATA FLAG BRANCH TIMING CONSIDERATIONS

The automatic data flag branch (ADFB) can occur up to 35 instructions after the instruction which caused it. The point at which the branch occurs can vary between executions of the same program as a result of the asynchronous I/O activity affecting the load/store operations.

The following points pertain to the central computer use of the data flag register (DFR).

- The content of the DFR, as stored into the register file by a 3B instruction, reflects all previous activity on it. Also, activity prior to the 3B instruction does not affect the new contents of the DFR.
- ADFBs caused by any instruction previous to a 3B instruction will occur prior to the 3B instruction or will be reflected by the DFR as stored off by the 3B instruction.
- An ADFB caused by the 3B instruction data flag register load will occur before the next instruction is issued.
- Sampling or altering a data flag bit with a 33 instruction may occur out of sequence with a previous scalar instruction, up to approximately 40 instructions earlier. However, the sampling of free data flags 53, 54, and 55 by the 33 instruction always occurs in sequence.
- If a 33 instruction alters a bit which causes an immediate ADFB, the branch may occur up to six instructions later.

When registers 1, 2, or 4 in the central computer register file are altered by an instruction, and this instruction is followed by an automatic data flag branch or illegal monitor mode instruction branch, the store operation may occur out of sequence with the branch operation. For example, if a 7E instruction loads register 2, and this instruction is followed by an automatic data flag branch, the automatic branch is to the address specified by either the old or new contents of register 2, depending on the timing of the FE and the branch.

GENERAL DEFINITIONS AND PROGRAMMING GUIDES

The following paragraphs provide general definitions and guides to aid in the programming of the computer system.

OVERLAP OF OPERAND AND RESULT FIELDS

If (in instructions such as vector, string, and so forth) the result field overlays a source field such that elements of the result are stored in the source field before elements in this portion of the source field are read, undefined results may occur. The source elements may be the original elements or they may be the newly-stored elements. In the latter case, the instruction results become undefined. Some instructions prohibit any overlap of source and destination fields. This restriction is included in the instruction descriptions.

ILLEGAL INSTRUCTIONS

Illegal instructions are those with function codes that are not part of the computer instruction set listed in the instruction list table in section 6. An illegal instruction, when used in job mode, causes an exchange to the monitor mode. Instruction execution then begins at the address specified by the content of the register file absolute register 3. An illegal instruction, when used in monitor mode, causes a branch to the register file absolute register 4. Instruction execution then begins at the address specified by the content of the register file absolute register 4.

INSTRUCTIONS WHICH CAUSE UNDEFINED RESULTS OR OPERATIONS

Instructions which contain unused bits must have those bits set to zero or instructions cause undefined results or operations. The unused bit areas of the instructions are shown with cross-hatched lines in the instruction word formats in section 6.

The job mode of operation protects memory from any undefined results or operations with the key-lock virtual addressing mechanism. This mechanism permits memory storage only to pages assigned to the current job for which the write lockout bits are not set.

The monitor mode of operation does not have the protection against undefined results or operations because it makes all memory references with absolute addresses.

ITEM COUNT

Item count is a term used in the instruction descriptions (section 4) to highlight the fact that certain instructions perform operations on a number of items. The term is general and refers to items which may be in bits, bytes, half-words, or words. Descriptions which use the term are those which specify instruction field lengths, offsets, indexes, and/or shift counts.

The size of the items in an item count is specified for applicable instructions in the instruction list tables (located near the front of section 4). The item size is listed under the table heading Number of Bits in Operand. In an example from the tables (shown below), the operand size is 8 which indicates that the field lengths and indexes for the F8 instruction are expressed in bytes.

F8 3 8 ST Move Bytes Left; A → C

In another example (shown below), the operand is E. This indicates that the instruction uses 32-bit or 64-bit items, depending on the status of instruction bit 8 (G bit 0). An item count for a field length of this instruction means that the field contains 100 32-bit items or 100 64-bit items, depending on instruction bit 8.

80 1 E VT ADD U; A+B → C

When an item count (other than a field length) is contained in a 16-bit field, at least one sign bit must be present. Item counts in 16-bit fields are therefore limited to the range of $2^{15}-1$ to -2^{15}. (Refer to the following description of field length.) When an item count other than an index consists of 48 bits, the leftmost 33 bits of the item count must be identical sign bits. Sign bits must always be extended to the left to fill the 16-bit or 48-bit field that contains it.

FIELD LENGTH AND OFFSET

Vector, vector macro, sparse vector, logical string, and some nontypical instructions use a field length. An offset is used in vector, vector macro, and some nontypical instructions. The field length as read from the register file before possible offset modification, is always interpreted as a positive number in the range of 0 to $2^{16}-1$ (65 535).

If a vector or other data field has no offset, the field is considered terminated before the reading of the first operand if the specified field length is zero.

Instructions having offsets must have 32 identical sign bits. The offsets are in the range -2^{16} to $2^{16}-1$. If the offset is not in this range, the operation of the instruction is undefined. The resulting field length after subtracting the offset from the field length (read from register A, B, C) must be positive and less than $2^{16}-1$ or the field length is treated as zero.

INDEX

String, some branch, and some nontypical instructions use an index. The sign of an index may be either positive or negative. The maximum magnitude of an index depends on its use as defined in the instruction descriptions. The machine left shifts the indexes end-off zero, three, five, or six positions before the index is added to the base address. The number of positions shifted depends on whether the unit for the index is bits, bytes, half-words, or words, respectively.

OPERAND SIZE DEFINITIONS

Following is a listing of operand sizes which apply throughout this manual unless otherwise stated.

Word	A 64-bit quantity having the address of the leftmost bit always being a multiple of 64_{10}.
Half-word	A 32-bit quantity having the address of the leftmost bit always being a multiple of 32_{10}.
Byte	An 8-bit quantity having the address of the leftmost bit always being a multiple of 8_{10}.
Digit	A 4-bit binary coded decimal number or sign. In zoned format there is one digit per byte, and in packed BCD format there are two digits per byte (refer to the string instructions descriptions for more detail).
Sword	512 bits (or eight 64-bit words).
Two-sword	1024 bits (two swords).

RESTRICTION ON SELF-MODIFYING PROGRAMS

It is difficult to use self-modifying programs properly in machines utilizing high-speed parallel architecture. Therefore, it is necessary to serialize the operation of the machine. This usually results in reduced performance.

Sophisticated methods requiring intimate familiarity with the machine can be utilized to execute self-modifying routines with less negative impact on performance. Guidelines are presented here to provide a basic method for satisfying most system requirements. The following operations must be performed in the order indicated.

1. Program modification must be performed only with the 13, 32, 5F, or 7F instructions.

2. An instruction must be executed which guarantees that the former 13, 32, 5F, or 7F instruction is completed before the latter 05 instruction starts. One such instruction is the 3284XX01, where XX is any register containing a valid memory address.

3. A 05 instruction must follow the instruction given in step 2, and precede the modified code. This voids the instruction stack and initiates an out-of-stack branch.

RESULT VECTOR 64-SWORD LOOK-AHEAD

The length of the result vector for the following instructions is input data dependent.

- Sparse vector (A0 through AF) and the compress (CF) instruction; the length of the result vector (C) depends on the number of 1 bits in the output order vector (Z).

- Compress (BC) instruction; the length of the result vector (C) depends on the number of 1 bits in the order vector (Z).

As the computer proceeds through the execution of the above instructions, it checks that an extra 64-sword page (small page) of result field is available if needed (64-sword look-ahead). Therefore, it is necessary to provide one more small page for the result vector beyond the expected length.

For the sparse vector (A0 through AF) instructions, it is not necessary to provide an extra small page beyond the maximum possible result field length. The maximum possible length of result vector C is equal to the field length of output order vector Z.

NUMBER SYSTEMS AND TABLES A

GENERAL

Any number system may be defined by the radix or base. The radix or base is the number of unique symbols used in the system. The decimal system has 10 symbols, 0 through 9. Modulus is the number of unique quantities or magnitudes a given device can distinguish. For example, an adding machine with 10 digits, or counting wheels, has a modulus of $10^{10}-1$. The adding machine has a modulus because the highest number which this machine can express is 9,999,999,999.

Most number systems are positional; that is, the relative position of a symbol determines its magnitude. In the decimal system, a 5 in the units column represents a different quantity than a 5 in the 10's column. Quantities equal to or greater than 1 may be represented by using the 10 symbols as coefficients of ascending powers of the base 10. The number 984_{10} becomes:

$$\begin{aligned} 9 \times 10^2 &= 9 \times 100 = 900 \\ +\ 8 \times 10^1 &= 8 \times 10 = 80 \\ +\ 4 \times 10^0 &= 4 \times 1 = 4 \\ &\qquad\qquad 984_{10} \end{aligned}$$

Quantities less than 1 may be represented by using the 10 symbols as coefficients of ascending negative powers of the base 10. The number 0.593_{10} may be represented as:

$$\begin{aligned} 5 \times 10^{-1} &= 5 \times .1 = .5 \\ 9 \times 10^{-2} &= 9 \times .01 = .09 \\ 3 \times 10^{-3} &= 3 \times .001 = .003 \\ &\qquad\qquad .593_{10} \end{aligned}$$

BINARY NUMBER SYSTEM

Internal operations in the computer use the binary number system. This system uses two symbols, 0 and 1; the base is 2. Because of the two-state characteristics, the binary system lends itself well to representation by the electronic switching circuits in the computer. The following numbers show the positional value of the binary number system.

. . .	2^5	2^4	2^3	2^2	2^1	2^0	
	32	16	8	4	2	1	Binary point

The binary number 011010 represents:

$0 \times 2^5 = 0 \times 32 = 0$

$+1 \times 2^4 = 1 \times 16 = 16$

$+1 \times 2^3 = 1 \times 8 = 8$

$+0 \times 2^2 = 0 \times 4 = 0$

$+1 \times 2^1 = 1 \times 2 = 2$

$+0 \times 2^0 = 0 \times 1 = 0$

26_{10}

Fractional binary numbers may be represented by using the symbols as coefficients of ascending negative powers of the base.

	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	- - -
Binary Point	1/2	1/4	1/8	1/16	1/32	

The binary number 0.10110 may be represented as:

$1 \times 2^{-1} = 1 \times 1/2 = 1/2 = 8/16$

$0 \times 2^{-2} = 0 \times 1/4 = 0 = 0$

$1 \times 2^{-3} = 1 \times 1/8 = 1/8 = 2/16$

$1 \times 2^{-4} = 1 \times 1/16 = 1/16 = 1/16$

$0 \times 2^{-5} = 0 \times 1/32 = 0 = 0$

$11/16_{10}$

HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system uses 16 discrete symbols (base 16). Table A-1 shows the 16 hexadecimal symbols with the decimal and binary equivalents. Note that the first 10 hexadecimal symbols are identical to the corresponding decimal symbols. The remaining six symbols are represented by alphabetical characters A through F.

NOTE

To avoid confusion between hexadecimal and decimal numbers in section A, all numbers shown without the base number affixed are hexadecimal numbers. Decimal numbers are shown with the base designator 10 affixed in the conventional manner. For example, the number 79847 represents a hexadecimal number. Conversely, 79847_{10} represents a decimal number.

TABLE A-1. HEXADECIMAL EQUIVALENTS

Binary	Decimal	Hexadecimal
00000	00	00
00001	01	01
00010	02	02
00011	03	03
00100	04	04
00101	05	05
00110	06	06
00111	07	07
01000	08	08
01001	09	09
01010	10	0A
01011	11	0B
01100	12	0C
01101	13	0D
01110	14	0E
01111	15	0F
10000	16	10

With base 16, the positional value of hexadecimal numbers is:

16^5	16^4	16^3	16^2	16^1	16^0
$1{,}048{,}576_{10}$	$65{,}536_{10}$	$4{,}096_{10}$	256_{10}	16_{10}	1

The hexadecimal number 859F is:

$8 \times 16^3 = 8 \times 4{,}096_{10} = 32{,}768_{10}$

$5 \times 16^2 = 5 \times 256_{10} = 1{,}280_{10}$

$9 \times 16^1 = 9 \times 16_{10} = 144_{10}$

$F \times 16^0 = F\dagger \times 1 = 15_{10}$

$34{,}207_{10}$

†To perform this multiplication, the hexadecimal symbol F is first converted to its decimal equivalent 15 (table A-1).

Fractional hexadecimal numbers may be represented by using the symbols as coefficients of ascending negative powers of the base.

16^{-1}	16^{-2}	16^{-3}	16^{-4}	. . .
$1/16_{10}$	$1/256_{10}$	$1/4096_{10}$	$1/65536_{10}$	

The hexadecimal number .48C0 represents:

$$4 \times 16^{-1} = 4 \times 1/16 = \frac{1024}{4096_{10}}$$

$$8 \times 16^{-2} = 8 \times 1/256 = \frac{128}{4096_{10}}$$

$$C \times 16^{-3} = C \times 1/4096 = \frac{12}{4096_{10}}$$

$$\frac{1164}{4096_{10}} = \frac{291}{1024_{10}} = .284$$

Since a group of 4 bits can represent any one of the 16 hexadecimal symbols, this notation is used throughout the instruction manuals for instruction codes, operands, addressing, and so on. Table A-1 shows the hexadecimal equivalents for each unique group of 4 bits.

The hexadecimal number system enables direct substitution of a hexadecimal symbol for a group of 4 bits. Figure A-1 illustrates the substitution of a hexadecimal number for a 32-bit operand. Thus, the equivalent hexadecimal symbol is substituted for each successive group of 4 bits, producing the complete hexadecimal equivalent.

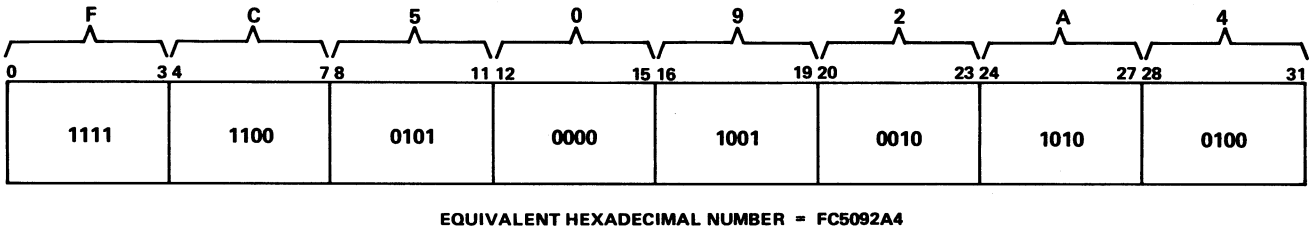


Figure A-1. Example of Hexadecimal Substitution for a Binary Number

Figure A-2 provides an easy way to add or multiply hexadecimal numbers.

ADDITION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION

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

Figure A-2. Hexadecimal Matrices

BINARY ARITHMETIC

The following subparagraphs present a brief description of binary arithmetic, including the one's and two's complement systems.

ADDITION AND SUBTRACTION

Binary numbers are added according to the following rules.

1 + 1 = 0 with a carry of 1

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

The addition of binary numbers proceeds as follows (the hexadecimal and decimal equivalents verify the result).

Augend	1001	(9)
Addend	0101	(5)
Partial Sum	1100	
Carries	0010	
Sum	1110 = E_{16} = 14_{10}	

Binary numbers are subtracted according to rules shown as follows:

0 - 0 = 0

0 - 1 = 1 with a borrow of 1

1 - 0 = 1

1 - 1 = 0

An example of binary subtraction is shown as follows:

Minuend	1001	(9)
Subtrahend	0101	(5)
Partial Difference	1100	
Borrows	1000	
Difference	0100	(4)

Numbers can also be subtracted by adding the complement of the addend as shown below.

Augend	1010	(A) (10_{10})
Addend	1100	(-3) One's complement of +3.
Partial Sum	0110	
Carries	0001	(End around carry)
Sum	0111	(+7)

The example above shows that the carry generated by the most significant stage of the add is added to the least significant stage (end around carry). The procedure for obtaining the one's complement of a binary number is described in the following subparagraphs.

ONE'S COMPLEMENT

In this system, positive numbers are represented by the binary equivalent. The negative numbers are represented in one's complement notation of the corresponding positive number.

The one's complement of a number is found by subtracting each bit of the number from 1. For example:

```
  1111
- 0101     (5)
------
  1010     (One's complement of 5.)
```

The substitution of ones for zeros and zeros for ones also produces the one's complement representation of a negative number.

In general, a negative number in the one's complement system contains a 1 in the most significant bit (sign bit). Conversely, a positive number contains a 0 in the most significant bit. This feature divides the range (modulus) of numbers that a given machine can represent into two halves. One half of the range represents positive numbers while the other half represents negative numbers. A machine with the modulus of 8 has the following range of numbers (refer to figure A-3).

SIGN BIT

$(-7F_{16})$ (-127_{10}) 10000000_2 (Maximum negative number)

$(+7F_{16})$ $(+127_{10})$ 01111111_2 (Maximum positive number)

Figure A-3. Example of a Modulus 8 System

Thus, this machine has a modulus of $\pm (2^7-1)$.

If a 1 is added to the maximum positive number shown in the example, the result equals the maximum negative number as shown in figure A-4.

Such a result is termed an overflow because the result exceeds the modulus of the machine.

```
                    01111111
                          +1
                    --------
Partial Sum         01111110
Carries             11111110
                    --------
Sum                 10000000
                    ^
                    |
                    OVERFLOW
```

Figure A-4. Example of Overflow

In a similar manner, figure A-5 shows that the subtraction of a one from the maximum negative number produces a result that exceeds the modulus of the machine in a negative direction. This result is termed an underflow.

```
                    10000000
                          -1
                    --------
Partial Difference  10000001
Borrows             11111110
                    --------
                    01111111
                    ^
                    |
                    UNDERFLOW
```

Figure A-5. Example of Underflow

In the central computer, underflows and overflows are detected. In most cases, the detection of an overflow or underflow causes forced results. The type of forced results caused by the detection is included with the applicable instruction description.

TWO'S COMPLEMENT

The two's complement system is used exclusively in central computer arithmetic operations. The system is similar to the one's complement system. Positive numbers are represented identically in the two systems. Negative numbers differ by one count. Table A-2 shows a comparison of one's and two's complement representations of counts 0 through 9. In the one's complement system, there are two representations for zero: a +0 and -0. Table A-2 shows the -0 as all ones in parentheses. This feature causes negative numbers in the one's and two's complement systems to vary by one count.

TABLE A-2. COMPARISON OF ONE'S AND TWO'S COMPLEMENT REPRESENTATIONS

Count	Two's Complement Representation	One's Complement Representation
+9	01001	01001
+8	01000	01000
+7	00111	00111
+6	00110	00110
+5	00101	00101
+4	00100	00100
+3	00011	00011
+2	00010	00010
+1	00001	00001
0	00000	00000 (11111)
-1	11111	11110
-2	11110	11101
-3	11101	11100
-4	11100	11011
-5	11011	11010
-6	11010	11001
-7	11001	11000
-8	11000	10111
-9	10111	10110

Positive numbers in the two's complement system can be converted to the equivalent negative numbers by first taking the one's complement of the positive number and then adding +1 to the result. Figure A-6 shows an example of the procedure.

```
00111     (+7)
11000     (One's complement = -7)
  +1      (Add one)
11001     Two's complement = -7)
```

Figure A-6. Example of Converting a Positive Number to a Negative Number in Two's Complement

Addition and subtraction in the two's complement system are performed in the same way as in the one's complement system. However, the end-around carry and borrow features, used in the one's complement system, do not apply to the two's complement system. Figure A-7 shows a comparison of adding a -1 to a +8 in the one's and two's complement systems, respectively.

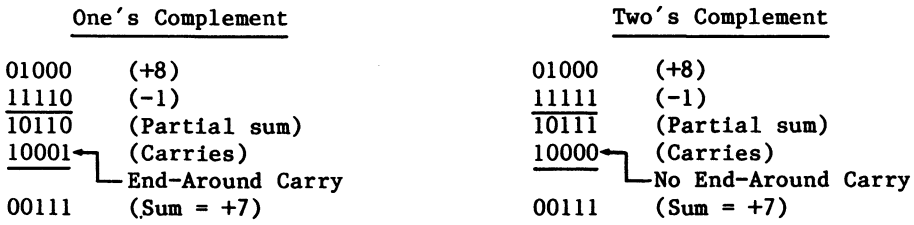


Figure A-7. Comparison of Addition in the One's and Two's Complement Systems

MULTIPLICATION

Binary multiplication proceeds according to the following rules.

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

Multiplication is always performed on a bit-by-bit basis.

Decimal example:

Multiplicand	14	
Multiplier	12	
Partial Products	28	
	14	(shifted left one place)
Product	168_{10}	

Binary example:

Multiplicand	(14_{10})	1110	
Multiplier	(12_{10})	1100	
Partial Products		0000	
		0000	shift to place
		1110	digits in proper
		1110	columns
Product	(168_{10})	10101000_2	

The following example is one method of computer multiplication. The central computer uses variations of this method. However, the following example is valid for explanation.

The computer determines the running subtotal of the partial products. Rather than shifting the partial product to the left to position it correctly, the computer right-shifts the summation of the partial products one place before the next addition is made. When the multiplier bit is a 1, the multiplicand is added to the running total and the result is shifted to the right one place. When the multiplier is a 0, the running total is shifted to the right, effectively multiplying the quantity by 10_2. Figure A-8 shows an example of the multiplication procedure used in the computer.

Multiplicand	1110	(14_{10})	
Multiplier	1100	(12_{10})	
(Multiplier Bit = "0")		0000	First Running Total (Shifted Right One)
(Multiplier Bit = "0")		00000	Second Running Total
		1110	(Shifted Right One)
(Multiplier Bit = "1")		111000	Third Running Total
		1110	(Shifted Right One)
		10101000	Product (168_{10})

Figure A-8. Example of Computer Multiplication Procedure

DIVISION

The following examples show the familiar method of decimal division.

```
                  14          Quotient
Divisor     13 ) 185          Dividend
                 13
                 55           Partial Dividend
                 52
                  3           Remainder
```

The computer performs division in a similar manner (using binary equivalents):

```
                      1110        Quotient (14)
Divisor     1101 ) 10111001       Dividend
                   1101
                   10100
                    1101          Partial Dividends
                    01110
                     1101
                       11         Remainder (3)
```

However, instead of shifting the divisor right to position it for subtraction from the partial dividend (shown above), the computer shifts the partial dividend left, accomplishing the same result. Following each left shift, the divisor is subtracted from the dividend. If the result is positive, the corresponding bit of the quotient is set (1) and the resulting partial dividend is shifted left one position. If the result is negative, indicating that the divisor cannot be contained in the partial dividend, the corresponding bit of the quotient is cleared (0) and the previous partial dividend is shifted left one place. The process continues until the proper number (determined by the number of bits in the dividend) of subtraction and left-shift operations take place.

Figure A-9 shows an example of the division procedure used in the computer. Note that the first subtraction in the example would produce a negative result. Thus, the most significant bit of the quotient is cleared and the previous partial dividend (in this case, the initial dividend) is shifted left one position.

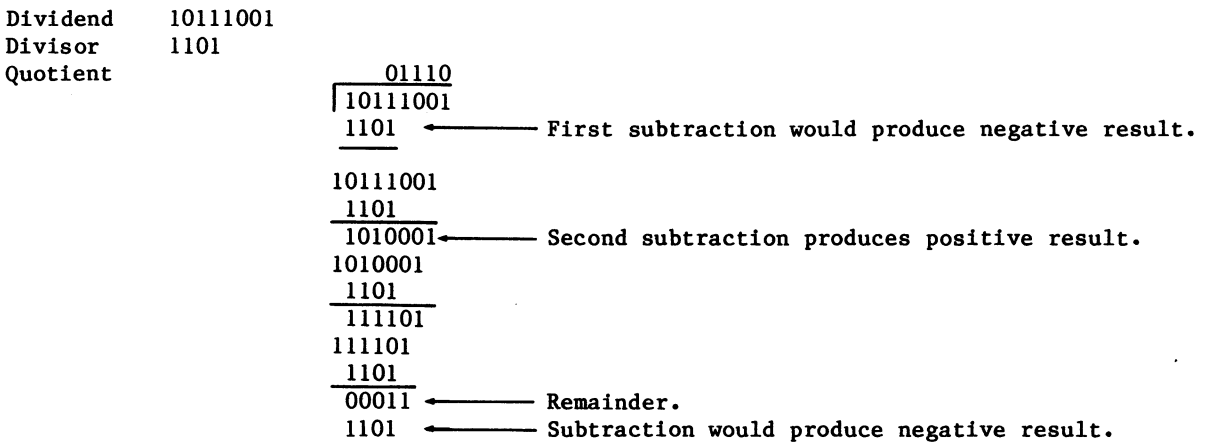


Figure A-9. Example of Computer Division Procedure

The second subtraction produces a positive result. Thus, the next most significant bit of the quotient is set and the result of the subtraction (partial dividend) is left-shifted one place.

Note that the result of the third subtraction is retained as the remainder since the fourth (final) subtraction would produce a negative result.

NUMBER CONVERSIONS

The procedures that may be used when converting a number from one number system to another are power addition, radix arithmetic, and substitution. Table A-3 lists the recommended conversion procedures.

TABLE A-3. RECOMMENDED CONVERSION PROCEDURES
(INTEGER AND FRACTIONAL)

Conversion	Recommended Method
Binary to Decimal	Power Addition
Decimal to Hexadecimal †	Power Addition
Decimal to Binary	Radix Arithmetic
Hexadecimal to Decimal †	Radix Arithmetic
Binary to Hexadecimal	Substitution
Hexadecimal to Binary	Substitution

General Rules

$r_i > r_f$: Use Radix Arithmetic, Substitution

$r_i < r_f$: Use Power Addition, Substitution

r_i = Radix of initial system

r_f = Radix of final system

†Refer to the Programming Reference Aids Manual (listed in the preface) for decimal to hexadecimal conversions for decimal numbers 0 through 40959.

POWER ADDITION

To convert a number from r_i to r_f ($r_i < r_f$), write the number in its expanded r_i polynomial form and simplify using r_f arithmetic.

Example 1: Binary to Decimal (Integer)

$$010111_2 = 1(2^4) + 0(2^3) + 1(2^2) + 1(2^1) + 1(2^0)$$

$$= 1(16) + 0(8) + 1(4) + 1(2) + 1(1)$$

$$= 16 + 0 + 4 + 2 + 1$$

$$= 23_{10}$$

Example 2: Binary to Decimal (Fractional)

$.0101_2 = 0(2^{-1}) + 1(2^{-2}) + 0(2^{-3}) + 1(2^{-4})$

$= 0 + 1/4 + 0 + 1/16$

$= 5/16_{10}$

Example 3: Decimal to Hexadecimal (Integer)

$875_{10} = 8(10^2) + 7(10^1) + 5(10^0)$

$= 8(A^2 16) + 7(A^1 16) + 5(A^0 16)$

$= 8(64_{16}) + 7(A_{16}) + 5(1)$

$= 320_{16} + 46_{16} + 5$

$= 36B_{16}$

NOTE

The base 10 is changed to the hexadecimal equivalent (A). The subsequent arithmetic is then performed in the hexadecimal system.

Example 4: Decimal to Hexadecimal (Fractional)

$.25_{10} = 2(10^{-1}) + 5(10^{-2})$

$= 2(A^{-1}16) + 5(A^{-2}16)$

$=2/A_{16} + 5/64_{16}$

$=19_{16}/64_{16}$

$= .4_{16}$

RADIX ARITHMETIC

To convert a whole number from r_i to r_f ($r_i > r_f$):

1. Divide the number to be converted by r_f, as expressed in r_i notation, using r_i arithmetic.
2. The remainder is the lowest-order digit in the new expression.
3. Divide the integral part from the previous step by r_f, as expressed in r_i notation.

4. The remainder is the next higher-order digit in the new expression.

5. The process continues until the division produces only a remainder which will be the highest-order bit in the r_f expression.

To convert a fractional number from r_i to r_f:

1. Multiply the number to be converted by r_f, as expressed in r_i notation, using r_i arithmetic.

2. The integral part is the highest-order bit in the new expression.

3. Multiply the fractional part from the previous operation by r_f, as expressed in r_i notation.

4. The integral part is the next lower-order bit in the new expression.

5. The process continues until sufficient precision is achieved or the process terminates.

Example 1: Decimal to Binary (Integer)

45 ÷ 2 = 22, remainder 1; record 1

22 ÷ 2 = 11, remainder 0; record 0

11 ÷ 2 = 5, remainder 1; record 1

5 ÷ 2 = 2, remainder 1; record 1

2 ÷ 2 = 1, remainder 0; record 0

1 ÷ 2 = 0, remainder 1; record 1

1 0 1 1 0 1

Thus: $45_{10} = 101101_2$

Example 2: Decimal to Binary (Fractional)

.25 x 2 = 0.5; record 0

.5 x 2 = 1.0; record 1

.0 x 2 = 0.0; record 0

.010

Thus: $.25_{10} = .010_2$

Example 3: Hexadecimal to Decimal (Integer)

9FC	÷	10_{10} (A_{16}) = 0FF remainder 6; record	6
0FF	÷	A_{16} = 19, remainder 5; record	5
019	÷	A16 = 2, remainder 5; record	5
2	÷	A_{16} = 0, remainder 2; record	2
			2556

Thus: $9FC_{16} = 2556_{10}$

Example 4: Hexadecimal to Decimal (Fractional)

.2AC x 10_{10} (A_{16}) = 1.AB8; record		1
.AB8 x A_{16} = 6.B30; record		6
.B30 x A_{16} = 6.FE0; record		6
.FE0 x A_{16} = 9.EC0; record		9
- -	- -	-
- -	- -	-
		.1669- -

Thus: $.2AC_{16} \cong .1669_{10}$

SUBSTITUTION

This method permits easy conversion between hexadecimal and binary numbers. If a binary number is partitioned into groups of 4 bits to the left and right of the binary point, each group of 4 bits converts into a hexadecimal digit. Similarly, each hexadecimal digit converts directly into a group of 4 bits. Table A-1 lists the hexadecimal digits and the corresponding binary equivalents.

Example 1: Binary to Hexadecimal

Binary =	1110	0000	0101.	1011	0010	1001
Hexadecimal =	E	0	5.	B	2	9

Example 2: Hexadecimal to Binary

Hexadecimal =	5	F	8.	7	C	A
	0101	1111	1000.	0111	1100	1010

Tables A-4 and A-5 are translation tables for extended binary coded decimal interchange code (EBCDIC) and American National Standard Code for Information Interchange (ASCII). The double row of squares around the top and left edge of each table show the binary and hexadecimal codes for the characters in the table. The following list gives a description of the control characters in the tables.

NUL	Null
SOH	Start of Heading (CC)
STX	Start of Text (CC)
ETX	End of Text (CC)
EOT	End of Transmission (CC)
ENQ	Enquiry (CC)
ACK	Acknowledge (CC)
BEL	Bell (audible or attention signal)
BS	Backspace (FE)
HT	Horizontal Tabulation (punched card skip) (FE)
LF	Line Feed (FE)
VT	Vertical Tabulation (FE)
FF	Form Feed (FE)
CR	Carriage Return (FE)
SO	Shift Out
SI	Shift In
DLE	Data Link Escape (CC)
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4 (Stop)
NAK	Negative Acknowledge (CC)
SYN	Synchronous Idle (CC)
ETB	End of Transmission Block (CC)
CAN	Cancel
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator (IS)
GS	Group Separator (IS)
RS	Record Separator (IS)
US	Unit Separator (IS)
DEL	Delete†

NOTE

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

Bits in the tables are identified by b_8, b_7, b_6, ... b_1 where b_8 is the highest order or most significant bit. Their numerical significance in binary is as follows:

Bit Identification	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1
Significance	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

† In the strict sense, DEL is not a control character.

FLOATING-POINT ARITHMETIC B

GENERAL

Most programmed arithmetic in the computer system takes place using two's complement, floating-point procedures. The following paragraphs describe the formats and procedures used in performing floating-point operations. Unless otherwise specified, numbers are expressed in hexadecimal notation (base 16).

FLOATING-POINT TECHNIQUE

The floating-point technique allows the computer to represent numbers with variable radix points and to perform computations on these numbers. Using floating-point procedures, the computer automatically places the radix point of the result at the proper position following a computation. Thus, by shifting the radix point and increasing or decreasing the exponent, computations on widely varying quantities which do not exceed the capacity of the machine can be performed.

Floating-point numbers within the computer are represented in a form similar to scientific notation; that is, a coefficient multiplied by a number raised to a power. Since the computer operates only on binary numbers, the numbers are multiplied by powers of two.

$C \cdot 2^E$ Where: C = Coefficient.

E = Exponent.

In floating-point, different coefficients need not relate to the same power of the base as do fixed point numbers. Therefore, the format of a floating-point number includes both the coefficient and exponent. All coefficients and exponents represented in the equipment are signed integers.

OPERAND FORMATS

Floating-point operations are performed on 32-bit and 64-bit operands. The function codes of the corresponding instructions specify whether 32-bit or 64-bit operands are to be used. The following subparagraphs describe the 32-bit and 64-bit formats.

32-BIT FORMAT

Figure B-1 shows the format of the 32-bit floating-point operands. Note that the bit positions of all operands are numbered left to right with the least significant bits in the rightmost bit positions of the word.

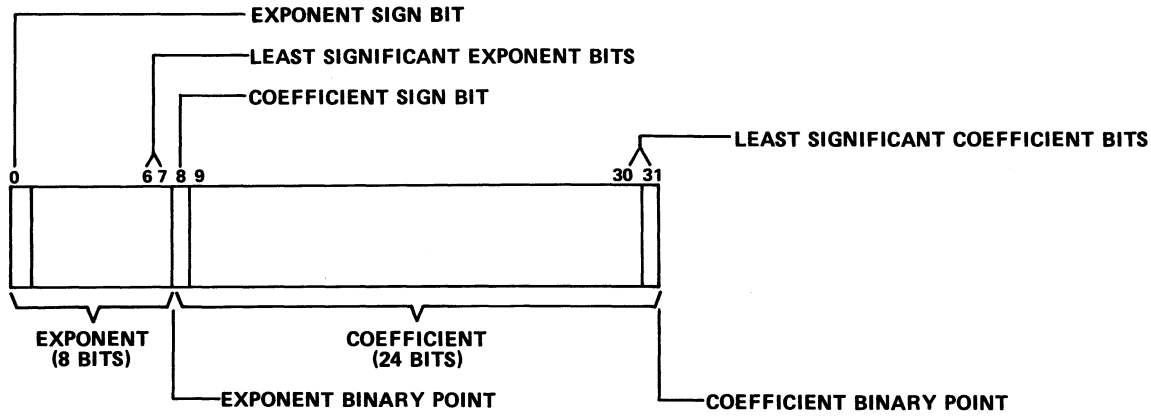


Figure B-1. 32-Bit Floating-Point Operand Format

The range of useful coefficients in the 32-bit format is from 800000 to 7FFFFF which provides a range of $-(2^{23})._{10}$ through $+2^{23}-1)_{10}$.

Useful exponents range from 90 to 6F which gives an exponent range of -112_{10} to $+111_{10}$. Numbers 70 through 8F fall into a special end-case range as listed in table B-1.

TABLE B-1. SPECIAL END CASE RANGE FOR THE 32-BIT FORMAT

Number	Definition
8XXXXXXX	Machine zero
7XXXXXXX	Indefinite
Note: X = Any hexadecimal digit.	

Table B-2 lists some floating-point numbers in the 32-bit format. Unless otherwise indicated, all numbers are in two's complement, hexadecimal notation.

Note that in two's complement notation, a negative number is one more than the corresponding one's complement notation for the same number. For example, in two's complement, -1 = FFFFFF (all ones) while in one's complement -1 = FFFFFE. Positive numbers in two's complement are identical to the corresponding one's complement notation for the same number.

TABLE B-2. FLOATING-POINT NUMBERS IN 32-BIT FORMAT

Number (Base 10)	Floating-Point Format	
	Exponent	Coefficient
+1	00	000001
+1 Normalized †	EA	400000
-1	00	FFFFFF
-1 Normalized †	E9	800000
+26790.0	00	0068A6
$+1/4 = +.25 = +.40_{16}$ Normalized †	E8	400000
256	00	000100

† In these examples, the coefficients are left shifted (normalized) until the sign bit is unequal to the bit immediately to its right. The exponent is reduced by one for each left shift.

64-BIT FORMAT

Figure B-2 shows the format of the 64-bit floating-point operands.

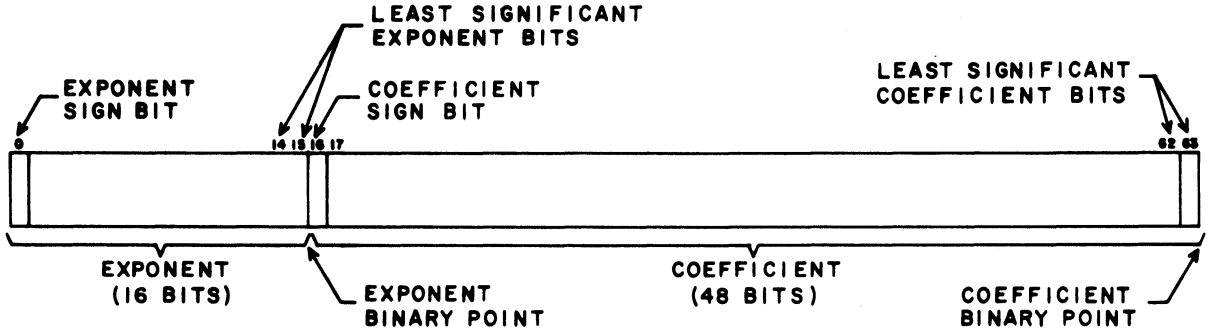


Figure B-2. 64-Bit Floating-Point Operand Format

The range of useful coefficients in the 64-bit format is from 8000 0000 0000 to 7FFF FFFF FFFF which provides a range of $-(2^{47})_{10}$ through $+(2^{47}-1)_{10}$.

Useful exponents range from 9000 to 6FFF which gives an exponent range of $-28,672_{10}$ to $+28,671_{10}$. Numbers 7000 through 8FFF fall into a special end case range as listed in table B-3.

TABLE B-3. SPECIAL END CASE RANGE FOR THE 64-BIT FORMAT

Number	Definition
8XXX XXXX XXXX XXXX	Machine zero.
7XXX XXXX XXXX XXXX	Indefinite.
Note: X = Any Hexadecimal Digit.	

The use of an undefined exponent in an arithmetic operation produces undefined results.

Table B-4 lists some floating-point numbers in the 64-bit format.

TABLE B-4. FLOATING-POINT NUMBERS IN 64-BIT FORMAT

Number Base 10	Floating-Point Format	
	Exponent	Coefficient
+1	0000	0000 0000 0001
+1 Normalized †	FFD2	4000 0000 0000
-1	0000	FFFF FFFF FFFF
-1 Normalized †	FFD1	8000 0000 0000
+26790.0	0000	0000 0000 68A6
$+1/4 = +.25 = +.40_{16}$	FFD0	4000 0000 0000*
$+256_{10}$	0000	0000 0000 0100

†In these examples, the coefficients are left shifted (normalized) until the sign bit is unequal to the bit immediately to its right. The exponent is reduced by one for each shift.

FLOATING-POINT OPERATIONS

In the following descriptions of floating-point operations, the 32-bit format is used for all examples. All descriptions and definitions of the operations apply to 64-bit operands with the adjustment for bit length. The following bit length substitutions are made for operations using 64-bit operands.

Bit Lengths for 32-Bit Operands	Bit Lengths for 64-Bit Operands
22	46
23	47
46	94
47	95
11	23

DOUBLE PRECISION RESULTS

Several instructions produce double-precision results. The double-precision add operation is a floating-point add producing both an upper and a lower result simultaneously and retaining both of these results for the next floating-point add operation. Thus the partial result in 64-bit arithmetic consists of 94 coefficient bits plus sign information. The partial result in 32-bit arithmetic consists of 46 bits plus sign information.

Dot Product instructions add both the upper and lower results of the multiply operations to the partial results of the add operations as described above.

UPPER AND LOWER RESULTS

Floating-point add, subtract, and multiply instructions generate result coefficients twice the length of the source-operand coefficients. The left and right halves of the result operands are called the upper (U) result and lower (L) result, respectively. Figure B-3 shows the format of the result operands.

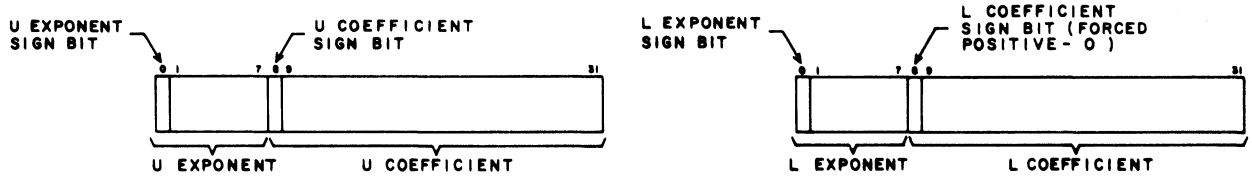


Figure B-3. Add, Subtract, and Multiply Result Format

The sign bit of the lower result coefficient is forced positive. The remaining bits of the lower coefficient are the normal results of the computations. Since the sign bit of the lower result coefficient is forced positive, the lower result is not meaningful alone, but must be used in conjunction with the upper result.

END CASES

If an indefinite operand is used in a floating-point operation, the upper and lower results are indefinite. Table B-5 lists each of the end case conditions and the result of each condition. In table B-5, 0 represents machine zero and N represents an operand that is not machine zero or indefinite. The coefficient of N is not all zeros.

TABLE B-5. END CASE CONDITIONS AND RESULTS

Condition	Result	Condition	Result
0 ± 0	0	$N \cdot 0$	0
$0 \pm N$	$\pm N$	$0 \div 0$	Indefinite
$N \pm 0$	N	$0 \div N$	0
$0 \cdot 0$	0	$N \div 0$	Indefinite
$0 \cdot N$	0		

FLOATING-POINT COMPARE RULES

The rules governing the comparison of floating-point operands are described on the following pages.

Neither Operand Indefinite or Machine Zero

If the signs of the coefficients of the two operands are unlike, the operands are unequal. The operand with the positive exponent is the larger of the two. If the signs of the coefficient are alike, the machine performs a floating-point subtract upper. This operation subtracts operand (S) from operand (R). Each of the arithmetic results are listed below with the corresponding compare results.

Arithmetic Result	Compare Result
Coefficient upper 24 bits all zeros (48 bits for 24 through 27 instructions)	(R) = (S)
Coefficient upper 24 bits not all zeros (48 bits for 24 through 27 instructions)	(R) $\neq$ (S)
Coefficient positive	(R) $\geq$ (S)
Coefficient negative	(R) < (S)

The compare results (R) = (S) and (R) ≠ (S) do not guarantee that (S) = (R) when (R) = (S).

The order of events of the floating-point subtract upper is first to complement the subtrahend, then align the coefficient associated with the smaller exponent, and finally to perform a floating-point add operation. The following is an example of (R) = (S) but (S) ≠ (R) for 64-bit compares.

Operand (R) =	0104	0000	0000	0001	
(S) =	0100	0000	0000	0001	
Complement (S)	0100	FFFF	FFFF	FFFF	
Align (S)	0104	FFFF	FFFF	FFFF	F
Add (R) and	0104	0000	0000	0001	
complemented,	0104	FFFF	FFFF	FFFF	F
aligned (S)	0104	0000	0000	0000	F

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

Operand (R) =	0100	0000	0000	0001	
(S) =	0104	0000	0000	0001	
Complement (S)	0104	FFFF	FFFF	FFFF	
Align R	0104	0000	0000	0000	1
Add aligned (R) and complemented (S)	0104	FFFF	FFFF	FFFF	1

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

Figure B-4 shows an example of the results of a branch if (R) ≥ (S) (32/64 bit FP), 22 instruction with the assumed instruction codes and register content. Note that in the initial comparison of the coefficient signs of (R) and (S) that they are alike. Thus a floating-point subtract operation contains a positive sign which indicates that (R) > (S). Since this result satisfies the assumed branch condition, the program branches to the indicated branch address.

One or Both Operands Indefinite

If one operand is indefinite, the compare condition is not met since indefinite 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) ≥ (S) conditions can be met since indefinite equals indefinite.

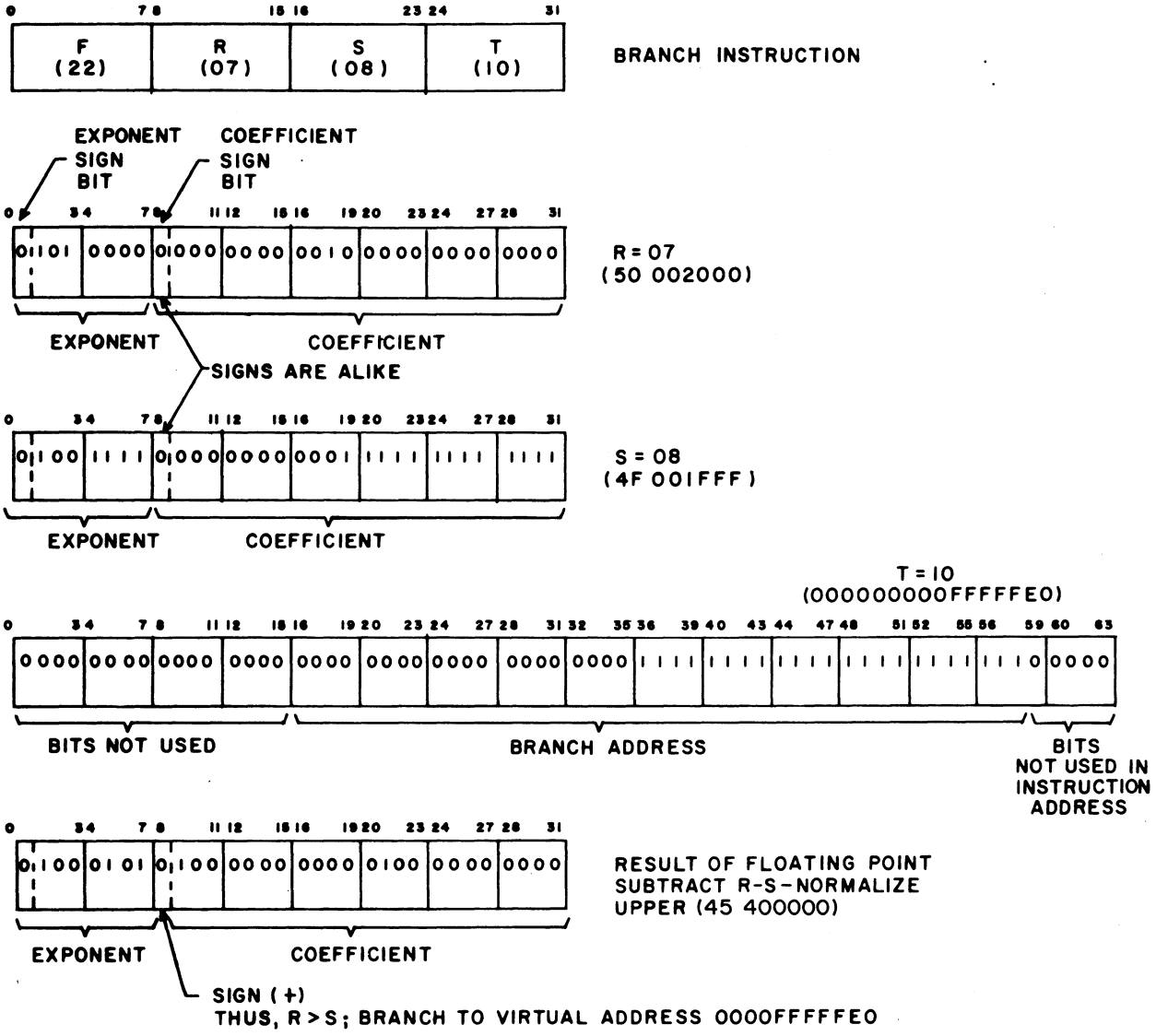


Figure B-4. Example of Branch if (R) $\geq$ (S) (32/64 Bit FP) Instruction

Neither Operand Indefinite but One or Both are Machine Zero

Under this condition, the following definitions apply to the comparison.

1. Any nonindefinite, nonmachine zero operand with a positive, nonzero coefficient is greater than machine zero.

2. Any nonindefinite, nonmachine zero operand with a negative coefficient is less than machine zero.

3. Machine zero is considered equal only to itself and to any number having a finite exponent and an all zero coefficient.

RIGHT NORMALIZATION

When the upper result coefficient overflows, the machine shifts the entire 47-bit result (with sign extension) one place to the right. The upper exponent is increased by one. The machine performs this operation, termed right normalization, when necessary, although normalization may not have been specified by the instruction.

Figure B-5 shows an example of right normalization. In this example, assume that the following floating-point numbers are added, causing the upper result coefficient to overflow.

Note in the example that the sign bit of the lower result is forced positive (0) and bit 23 is shifted around it.

ADD AND SUBTRACT OPERATIONS

Before the computer adds or subtracts floating-point numbers, the exponents are made equal by a procedure called alignment. The alignment procedure successively right shifts the coefficient of the operand with the smallest exponent one bit and increases the exponent by one until the two exponents are equal. The sign of the shifted coefficients is extended from the left to the right during the shift. Negative coefficients approach a minus one and positive coefficients approach zero as they are shifted.

Figure B-6 shows an example of floating-point addition with both operands positive. In figure B-6, the exponent of operand 2 is one less than the exponent of operand 1. The alignment procedure right shifts the coefficient of operand 2 one place to the right and increases its exponent by one, making it equal to the exponent of operand 1. Note that the least significant bit of operand 2 is shifted into bit 25 of the lower result (around the sign bit).

The addition of the coefficients takes place, using conventional binary addition procedures. After right normalization, if required, the result is 46 bits (not including the sign bits). The leftmost 23 bits contain the coefficient for the upper result and the rightmost 23 bits contain the coefficient for the lower result.

The exponent for the upper result equals the larger of the two source operand exponents. Note that right normalization (not necessary in the example) increases this exponent by one. The exponent for the lower result equals the upper result exponent -23_{10} (17_{16}) in all but the following three conditions.

1. Right normalization causes the upper result exponent to overflow. In this case, the computer sets the upper result to indefinite. The lower exponent will equal 59_{16} ($6FD1_{16}$ for 64-bit operands).

2. If the subtraction of 32_{10} from the upper result exponent causes the lower result exponent to underflow, the computer sets the lower result to machine zero.

3. If one or both operands were indefinite, the computer sets the upper and lower results to indefinite.

Figure B-7 shows an example of floating-point addition with one operand negative and the other positive.

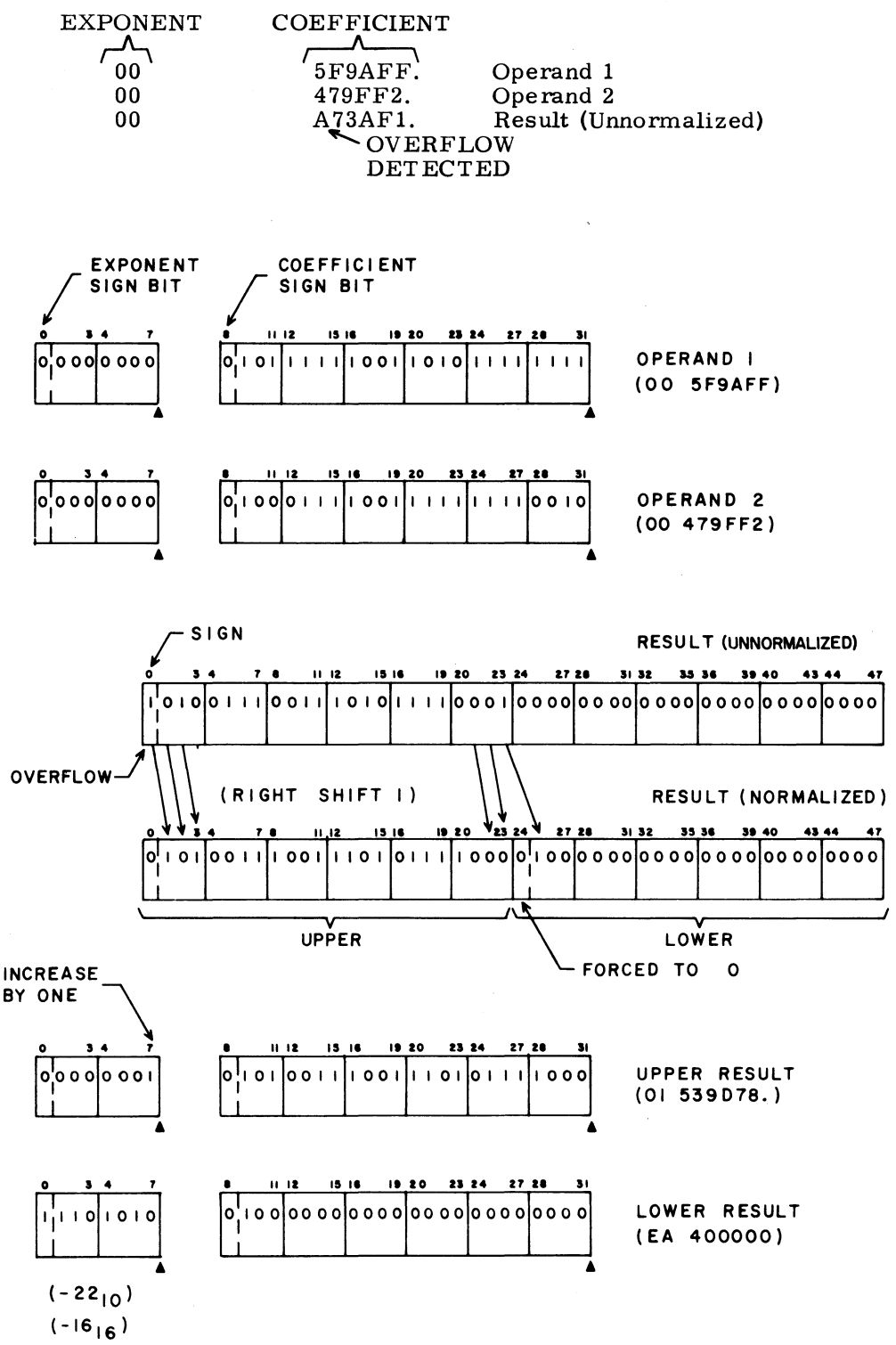


Figure B-5. Example of Right-Normalization

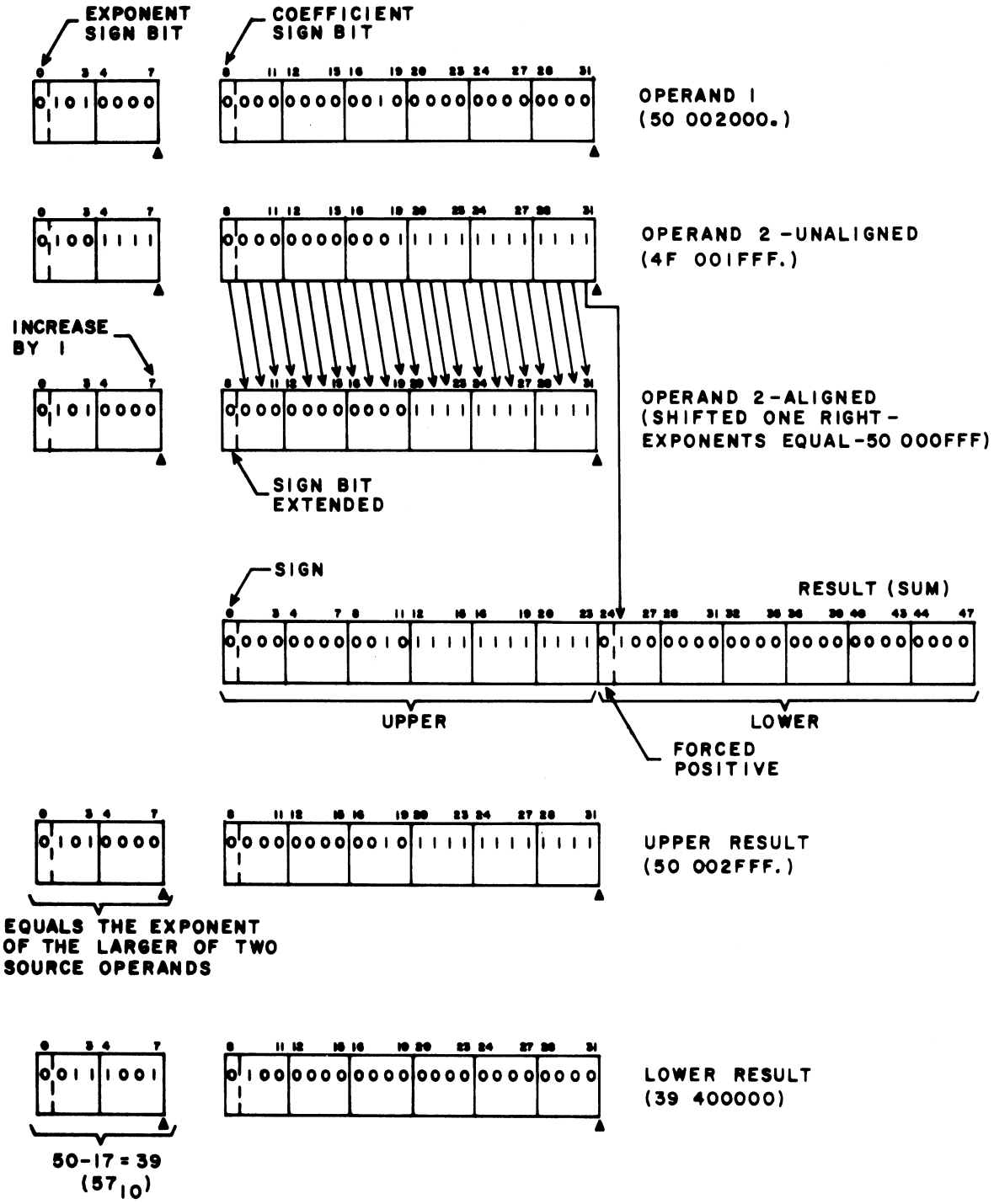


Figure B-6. Example of Floating-Point Addition (Both Operands Positive)

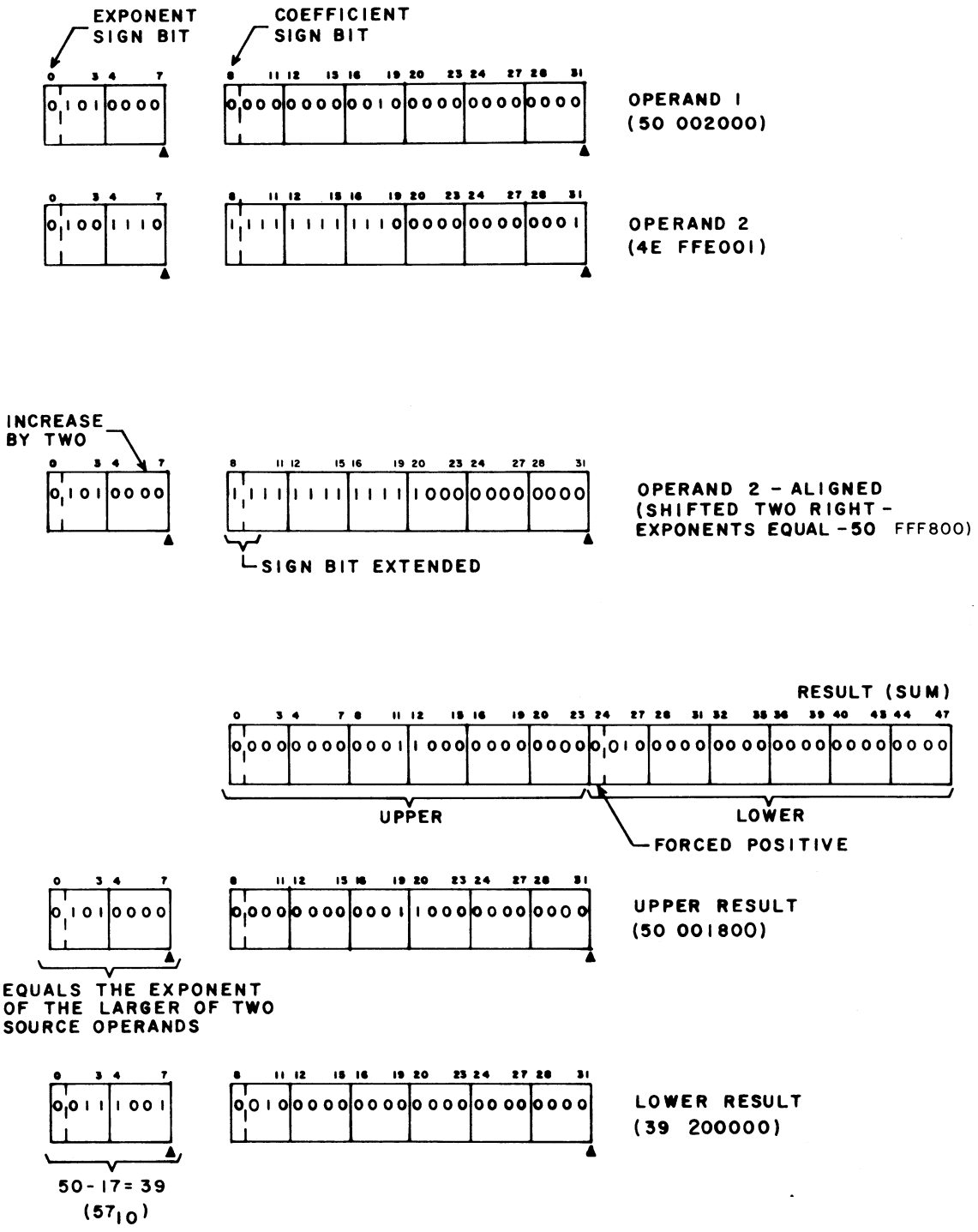


Figure B-7. Example of Floating-Point Addition (One Operand Negative and One Operand Positive)

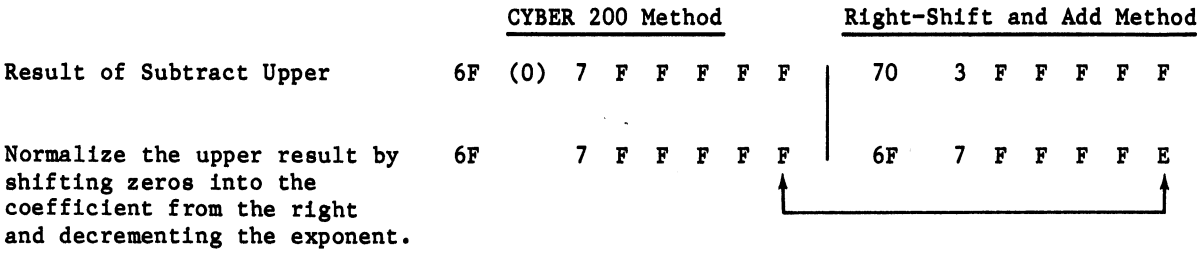
A floating-point subtraction consists of complementing the coefficient of the subtrahend and performing a floating-point addition. In 32-bit format, a 24-bit two's complement operation is performed before the operands are shifted. The complement of an 800000 coefficient is 400000 with one added to the value of the exponent associated with the coefficient.

The central computer hardware used for scalar floating-point add or subtract operations has an extra (or extended) coefficient sign bit. This means that 8000 is complemented without the specified right-shift of one and increase of the exponent by one. This causes a result which, although not mathematically incorrect, may differ from the specified result when all of the following conditions are met for any given pair of operands.

- The operand having the larger exponent must have a coefficient of 8000. If the exponents of the two operands are equal, one of the two must have a coefficient of 8000.
- The operand described in condition 1, having a coefficient of 8000, must be complemented. This may be due to the operand being the subtrahend in a subtract operation or because of sign control in either a subtract or add operation.
- The other operand must have a negative coefficient.

Figure B-8 shows two examples of floating-point subtraction using an extra coefficient sign bit.

If this operation is a subtract upper, the specified result is indefinite (with the appropriate data flags). The central computer result did not overflow. If this operation is a subtract normalized, the following results occur.



The normalized add and subtract instructions generate an intermediate result identical to the final result of the add U and the subtract U instructions. Normalizing of the intermediate, 24-bit result then takes place. In this operation (figure B-9), the computer left-shifts the 24 upper result bits until the sign bit and the bit immediately to the right of the sign bit are different.

The machine attaches zeros to the right of the result as it is shifted. The result exponent is reduced by the number of places shifted. If reducing the exponent by one causes exponent underflow, the result is set to machine zero. If the original coefficient consists of 24 zero bits, the result of the normalization becomes machine zero. If normalization is not specified in an add or subtract instruction, a zero coefficient and any exponent may result, and if reducing the exponent during shifting causes an exponent underflow, the machine sets the result to machine zero.

EXAMPLE 1 A − B

			CYBER 200 METHOD (EXTRA SIGN BIT)	RIGHT SHIFT AND ADD 1 TO EXPONENT METHOD
A		60	F F F 0 0 0	
B		64	8 0 0 0 0 0	
COMPLEMENT B	B	64	(1) 8 0 0 0 0 0	64 8 0 0 0 0 0
	$\overline{B}$	64	(0) 8 0 0 0 0 0	65 4 0 0 0 0 0
ALIGN OPERAND WITH SMALLER EXPONENT		60	(1) F F F 0 0 0	60 F F F 0 0 0
		64	(1) F F F F 0 0	65 F F F F 8 0
ADD A PLUS COMPLEMENT OF B	A	64	(1) F F F F 0 0	65 F F F F 8 0
	$+\overline{B}$	64	(0) 8 0 0 0 0 0	65 4 0 0 0 0 0
		64	(0) 7 F F F 0 0	65 3 F F F 8 0
		64	7 F F F 0 0	65 3 F F F 8 0

EXAMPLE 2 A − B

			CYBER 200 METHOD (EXTRA SIGN BIT)	RIGHT SHIFT AND ADD 1 TO EXPONENT METHOD
A		50	F F F 0 0 0	
B		6F	8 0 0 0 0 0	
COMPLEMENT B	B	6F	(1) 8 0 0 0 0 0	6F 8 0 0 0 0 0
	$\overline{B}$	6F	(0) 8 0 0 0 0 0	70 4 0 0 0 0 0
ALIGN OPERAND WITH SMALLER EXPONENT		50	(1) F F F 0 0 0	50 F F F 0 0 0
		6F	(1) F F F F F F	70 F F F F F F
ADD A PLUS COMPLEMENT OF B	A	6F	(1) F F F F F F	70 F F F F F F
	$+\overline{B}$	6F	(0) 8 0 0 0 0 0	70 4 0 0 0 0 0
		6F	(0) 7 F F F F F	70 3 F F F F F

Figure B-8. Examples of Floating-Point Subtraction Using an Extra Coefficient Sign Bit

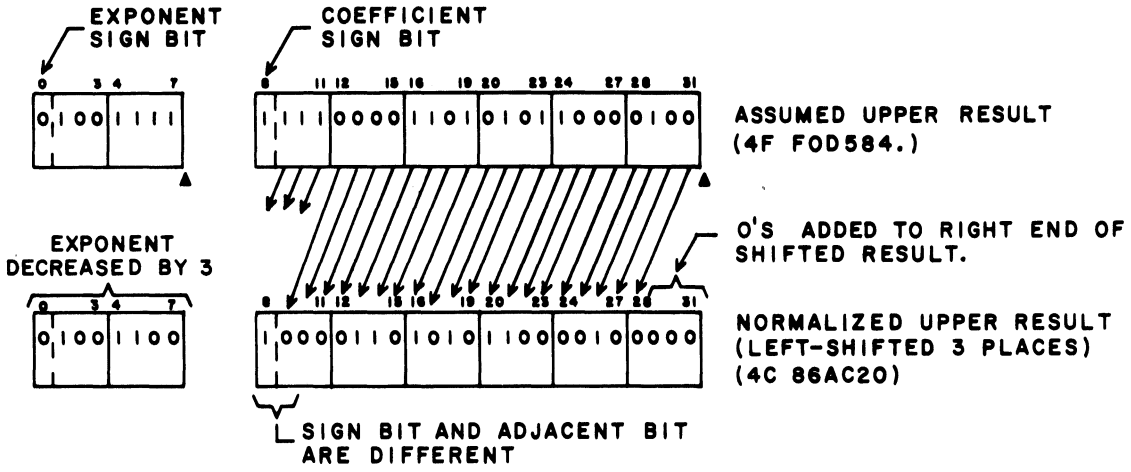


Figure B-9. Example of Normalized Upper Result

ORDER DEPENDENT RESULT CONSIDERATIONS

The result of any sequence of floating-point operations may be operand-order dependent [for instance, (A + B) + C = A + (B + C)].

The following example using 32-bit operands demonstrates this effect.

```
A =   00  000001

B =   00  000003

C =   01  000001

    A 00  000001
   +B 00  000003
   ---------------
  A+B 00  000004
   +C 01  000001
   ---------------
(A+B)+C 01 000003

    B 00  000003 <-- Coefficients not equal
   +C 01  000001
   ---------------
  B+C 01  000002
   +A 00  000001
   ---------------
A+(B+C) 01 000002
```

It is important that this characteristic of floating-point arithmetic be considered when predicting the results of the DA, DB, DC, and DF instructions.

MULTIPLY OPERATIONS

The multiplication of two floating-point operands produces a result coefficient with the least-significant 23 product bits in the lower result and the higher order 23 product bits in the upper result (figure B-10). Note that as in addition and subtraction, the sign bit of the lower result is cleared, forcing the lower result positive. The sign bit of the upper result is determined using the usual procedures of algebraic multiplication. Thus, in the example shown in figure B-10, the sign bit of the upper result is a zero (+) since both source operands are positive.

In the multiply operation, the positive forms of the input operands are used. The signs of the input operands are recorded to determine the sign of the upper result and whether the resultant coefficient should be complemented. If either of the input operands contains a coefficient of 800000, the operation changes the operand to a positive form by right shifting its coefficient by one (with sign extension) and adding one to its exponent. This gives a coefficient of C00000 which will then be complemented to 400000.

The lower result exponent is the sum of the exponents for the two source operands and the upper result exponent equals the lower result exponent plus 17_{16} or 23_{10} with the following exceptions.

1. The sum of the source operands' exponents (plus 23_{10}, if upper result) exceeds $6F_{16}$, in which case the result exponent is set to indefinite.

2. The sum of the source operands' exponents (plus 23_{10}, if upper result) is less than 90_{16}, in which case the result exponent is set to machine zero.

3. Either or both operands are indefinite, in which case the result exponent is set to indefinite.

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

DIVIDE OPERATIONS

In divide operations, a floating-point dividend is divided by a prenormalized divisor, producing a 23-bit coefficient (not including sign bit) of the quotient which appears as the upper result. If one or both source operands are negative, they are complemented and the absolute values are used in the divide operation. The signs of the original source operands determine the sign of the final coefficient according to the normal procedures of algebraic divisions.

Figure B-11 shows an example of floating-point division with both dividend and divisor positive. Note that prenormalization left shifts the divisor until the most significant one bit is adjacent to the sign bit. The normalize count (NC) is stored and will partially determine the exponent of the quotient.

The prenormalized divisor is then subtracted from the dividend and the corresponding bit of the quotient is determined. After each subtraction, the partial dividend is left shifted one position and the subtraction is repeated as in a conventional binary division operation.

After 23 subtract and 22 shift operations have been completed, the absolute value of the quotient coefficient appears as the upper result. If either the original dividend or divisor (but not both) were negative, the coefficient of the quotient is complemented. The rightmost bit of the quotient is neither rounded nor adjusted. The remainder is not retained.

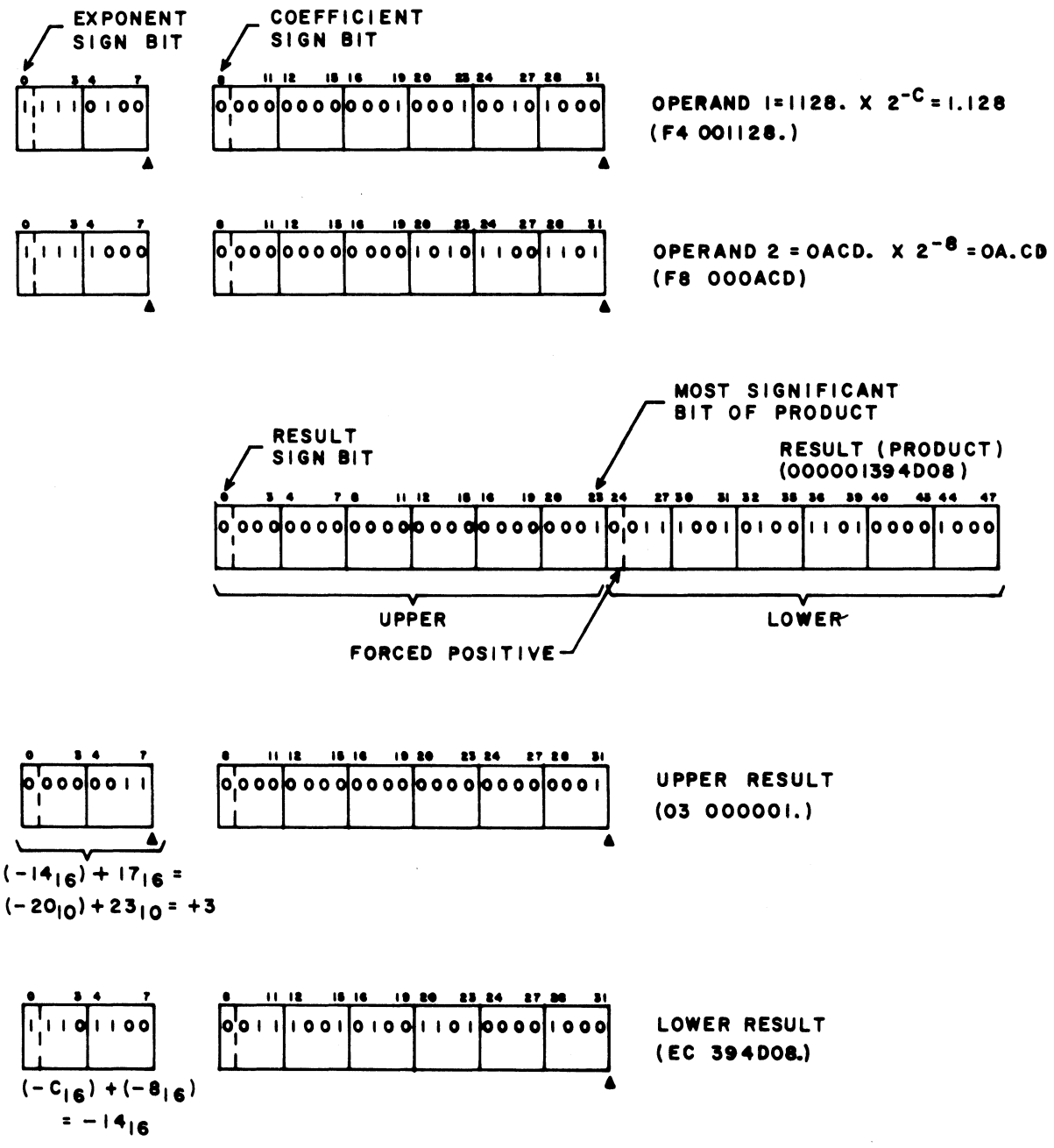


Figure B-10. Example of Floating-Point Multiply

The exponent of the quotient is determined by the equation shown in figure B-11.

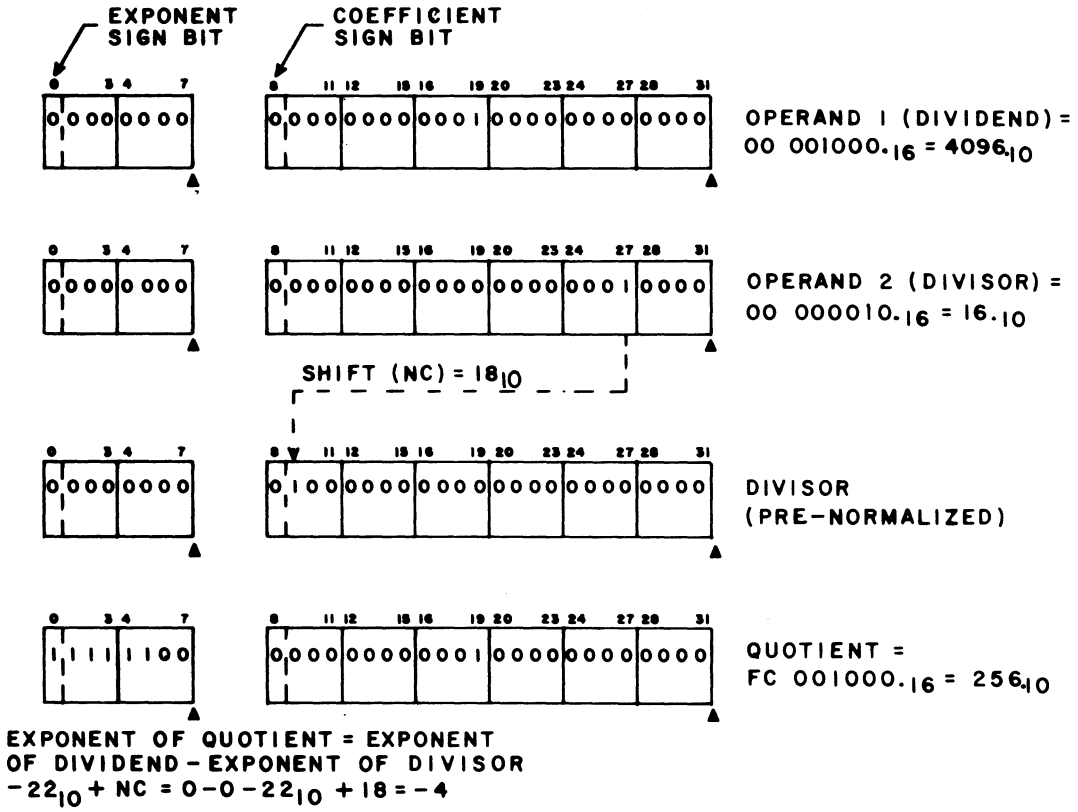


Figure B-11. Example of Floating-Point Divide
(Dividend and Divisor Both Positive)

The prenormalized divisor is then subtracted from the dividend and the corresponding bit of the quotient is determined. After each subtraction, the partial dividend is left-shifted one position and the subtraction is repeated as in a conventional binary division operation.

After 23 subtract and 22 shift operations have been completed, the absolute value of the quotient coefficient appears as the upper result. If either the original dividend or divisor (but not both) were negative, the coefficient of the quotient is complemented. The rightmost bit of the quotient is neither rounded nor adjusted. The remainder is not retained.

The exponent of the quotient is determined by the equation shown in figure B-11. Figure B-12 shows another example of floating-point division. However, in this case, the dividend is positive and the divisor is negative. As a result, the original divisor is complemented before the prenormalization takes place. Note that the quotient is complemented to form the negative final quotient.

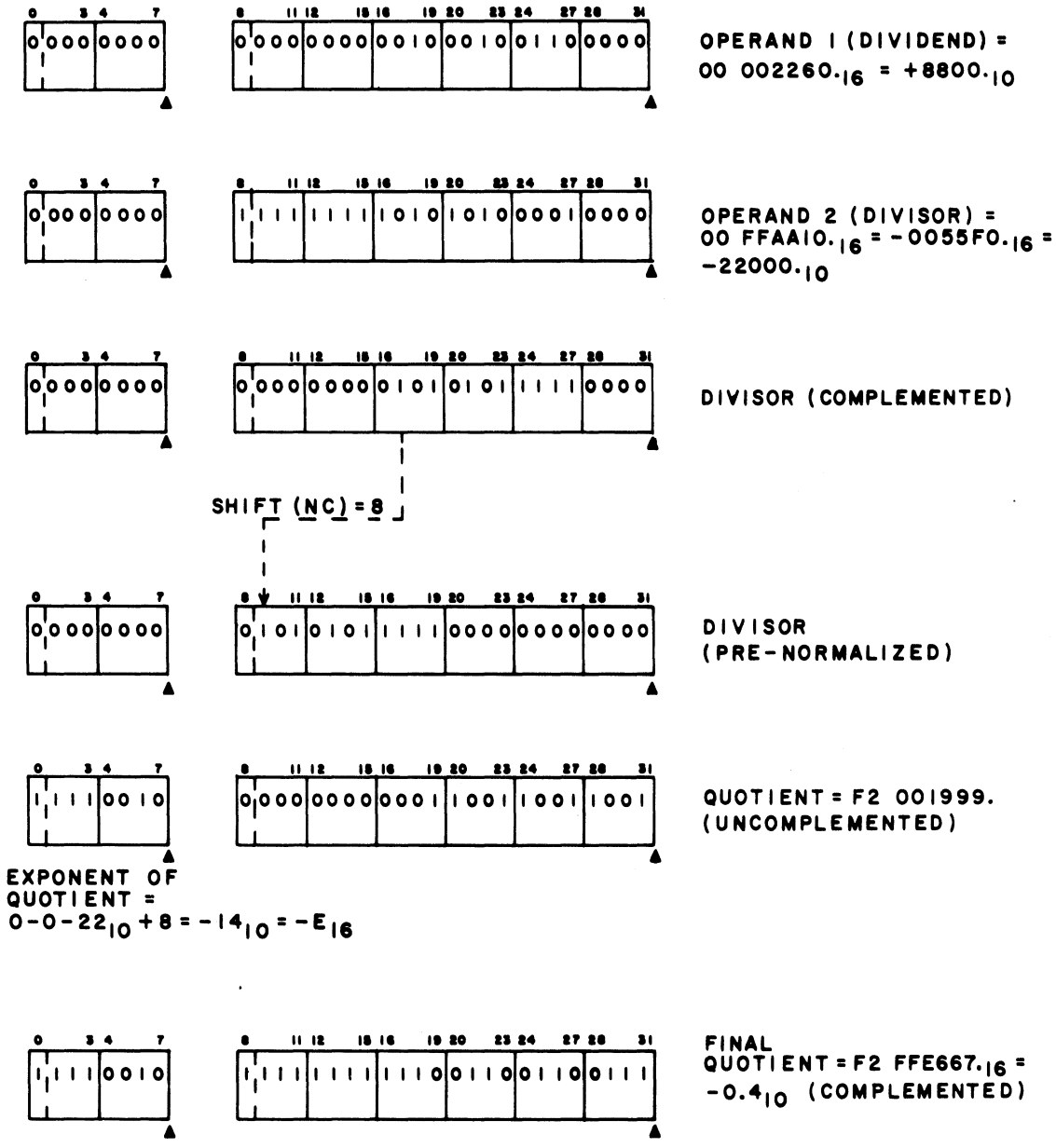


Figure B-12. Example of Floating-Point Divide
(Dividend Positive, Divisor Negative)

SIGNIFICANT RESULTS

Certain multiply and divide instructions specify that the significant results of the product or quotient be obtained. The significant bit count for a floating-point number equals the number of bit positions in the number (excluding sign bit) minus the left-shift count necessary to normalize the number. Refer to example in figure B-13.

A coefficient containing all zeros or all ones has a significant bit count of zero. Note that in a nonzero coefficient that is an exact power of two, the positive form of the coefficient results in a significant bit count that is one greater than the significant bit count of the negative form of the same coefficient. The operation determines the significance of an input operand as originally read from a register or from MCS before any operations such as sign control or the left shift for odd exponents in square root are performed.

Significant arithmetic determines which of the source operands contains the smaller significant bit count and records that count. After the following arithmetic operation, the sequence determines the significant bit count of the result after any necessary sign correction. The input significant bit count and the result significant bit count are then compared. If the significant bit count of the result is less than the significant bit count of the input, the sequence left-shifts (with zeros shifted in) the result coefficient according to the difference in significant bit counts and reduces the exponent accordingly. If the result and input significant bit counts are equal, the sequence does not shift the coefficient and does not adjust the exponent. If the result significant bit count is greater than the input significant bit count, the operation right-shifts (end off with sign extension) and increases the exponent accordingly. Note that for multiply, the entire 95-bit result (47 bits for 32-bit multiply) is shifted as required.

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

SQUARE ROOT OPERATIONS

In floating-point, square root operations, the following steps are performed.

1. The significance of the coefficient of the input operand is determined and recorded.

2. If negative, the input operand is complemented to its positive form.

3. If the exponent of the input operand is odd, it is reduced by one and the coefficient obtained in step 2 is multiplied by two. If the exponent is even, no modification is performed.

4. The machine now obtains the square root of the coefficient from step 3. Note that enough zeros are attached to the right end of the coefficient to produce 23 result bits (47 for 64-bit operands).

5. If the original input operand was negative, the result coefficient is complemented. If the input operand was positive, no modification takes place.

6. The result exponent is formed by dividing the exponent by two and subtracting 11_{10} from the exponent obtained in step 3. (Subtract 23_{10} for 64-bit square root.)

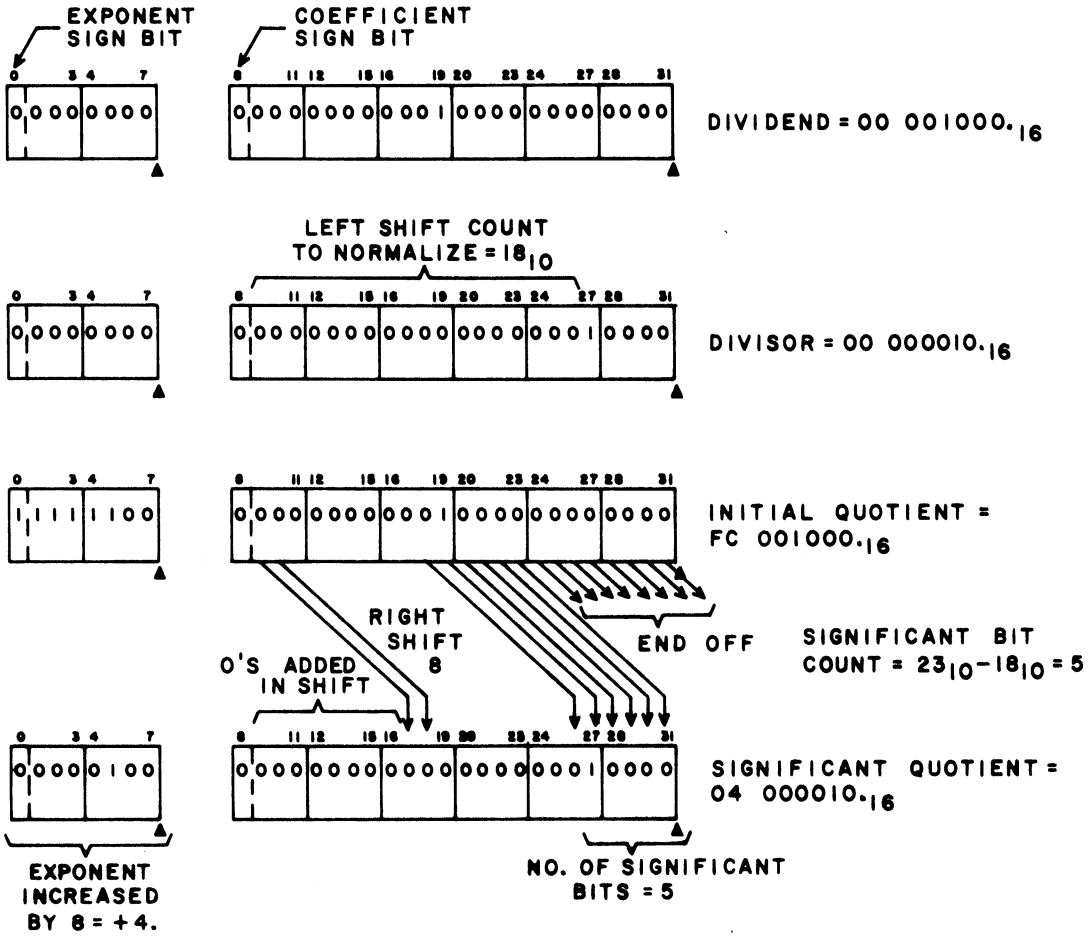


Figure B-13. Example of Significant Results of Floating-Point Divide

7. The result coefficient is adjusted to produce a coefficient with the same significance as the input operand. The significance count obtained in step 1 is used in the operation. The exponent of the result is also adjusted to compensate for the change in magnitude of the result coefficient.

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

Figure B-14 shows an example of a floating point, square root operation. In this example a positive input source operand is used. Thus, no complementing is necessary.

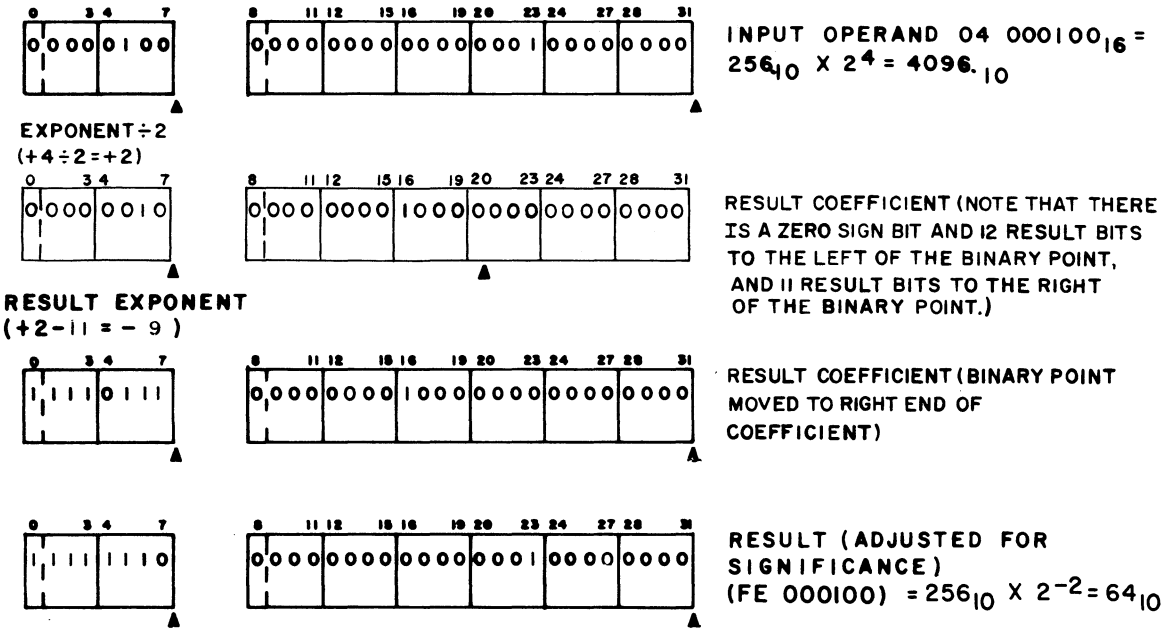


Figure B-14. Example of Floating-Point Square Root

G BITS AND TERMINATING CONDITIONS C

G BIT USAGES

Tables C-1 through C-5 provide the instruction G bit usages in a condensed form. Thus, these tables provide quick lookup charts for determining the G bit control configuration for a particular instruction to which they apply. Note that the G bit usages tables are arranged according to instruction type (vector VT, sparse vector SV, and so on) and according to function code within that type of instructions.

The key to the abbreviations used to designate the G bit usage conditions is given below.

G Bit	Abbreviation	Meaning
0	E	Either 32- or 64-bit operands
1	C	Control vector
2	O	Offset
3, 4	B	Broadcast
5, 6, 7	S	Sign control†
5, 6, 7	I	Optional index increment
Any	X	Defined in individual instruction description

† The operand flowchart (figure C-1) illustrates the order of operations when sign control is selected.

TABLE C-1. G BIT USAGES FOR VECTOR (VT) INSTRUCTIONS

NOTE

A blank space in the table indicates that the corresponding G bit does not apply for that instruction and must be a zero.

Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7
80	E	C	O	B	B	S	S	S
81	E	C	O	B	B	S	S	S
82	E	C	O	B	B	S	S	S
83		C	O	B	B			
84	E	C	O	B	B	S	S	S
85	E	C	O	B	B	S	S	S
86	E	C	O	B	B	S	S	S
87		C	O	B	B			
88	E	C	O	B	B	S	S	S
89	E	C	O	B	B	S	S	S
8A		C	O	B	B			
8B	E	C	O	B	B	S	S	S
8C	E	C	O	B	B	S	S	S
8F	E	C	O	B	B	S	S	S
90	E	C	O	B				
91	E	C	O	B				
92	E	C	O	B				
93	E	C	O	B		S	S	
94	E	C	O	B	B			
95	E	C	O	B	B			
96		C	O	B				
97		C	O	B				
98	E	C	O	B				
99	E	C	O	B				
9A	E	C	O	B				
9B	E	C	O	B	B			
9C		C	O	B				
9D	E	C	O	B	B	X	X	X

TABLE C-2. G BIT USAGES FOR SPARSE VECTOR (VT) INSTRUCTIONS

Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7
A0	E	X	X	B	B	S	S	S
A1	E	X	X	B	B	S	S	S
A2	E	X	X	B	B	S	S	S
A4	E	X	X	B	B	S	S	S
A5	E	X	X	B	B	S	S	S
A6	E	X	X	B	B	S	S	S
A8	E	X	X	B	B	S	S	S
A9	E	X	X	B	B	S	S	S
AB	E	X	X	B	B	S	S	S
AC	E	X	X	B	B	S	S	S
AF	E	X	X	B	B	S	S	S

TABLE C-3. G BIT USAGES FOR BRANCH (BR) INSTRUCTIONS

Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7
B0	X	X	X	X	X	X	X	X
B1	X	X	X	X	X	X	X	X
B2	X	X	X		X	X	X	X
B3	X	X	X		X	X	X	X
B4	X	X	X		X	X	X	X
B5	X	X	X		X	X	X	X

NOTE

Instructions 2F, 32, and 33 are not listed in this table because their G bits are used for control purposes and do not follow the bit definitions at the beginning of this section.

TABLE C-4. G BIT USAGES FOR VECTOR MACRO (VM) INSTRUCTIONS

Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7
B7†	E				B	X	X	X
B8	E	C	O					
BA††	E					X	X	X
C0	E	C		B	B			
C1	E	C		B	B			
C2	E	C		B	B			
C3	E	C		B	B			
D0	E	C	O	B	B			
D1	E	C	O					
D4	E	C	O	B	B			
D5	E	C	O					
DA	E	C						
DB	E	C						
DC	E	C		B	B			
DF	E	C	O					

†This instruction is undefined if G bits 4 and 6 are both set, or if G bits 6 and 7 are both set.
††This instruction is undefined if G bits 6 and 7 are both set.

TABLE C-5. G BIT USAGES FOR NONTYPICAL (NT) INSTRUCTIONS

Function Code	G Bit/Usage							
	0	1	2	3	4	5	6	7
BB	E			B	B			
BC	E	X						
BD	E			B	B			X
C4	E			B	B			
C5	E			B	B			
C6	E			B	B			
C7	E			B	B			
C8	E	C	X					
C9	E	C	X					
CA	E	C	X					
CB	E	C	X					
CC								X
CF	E				B	S	S	S
D8	E	C				S		
D9	E	C				S		

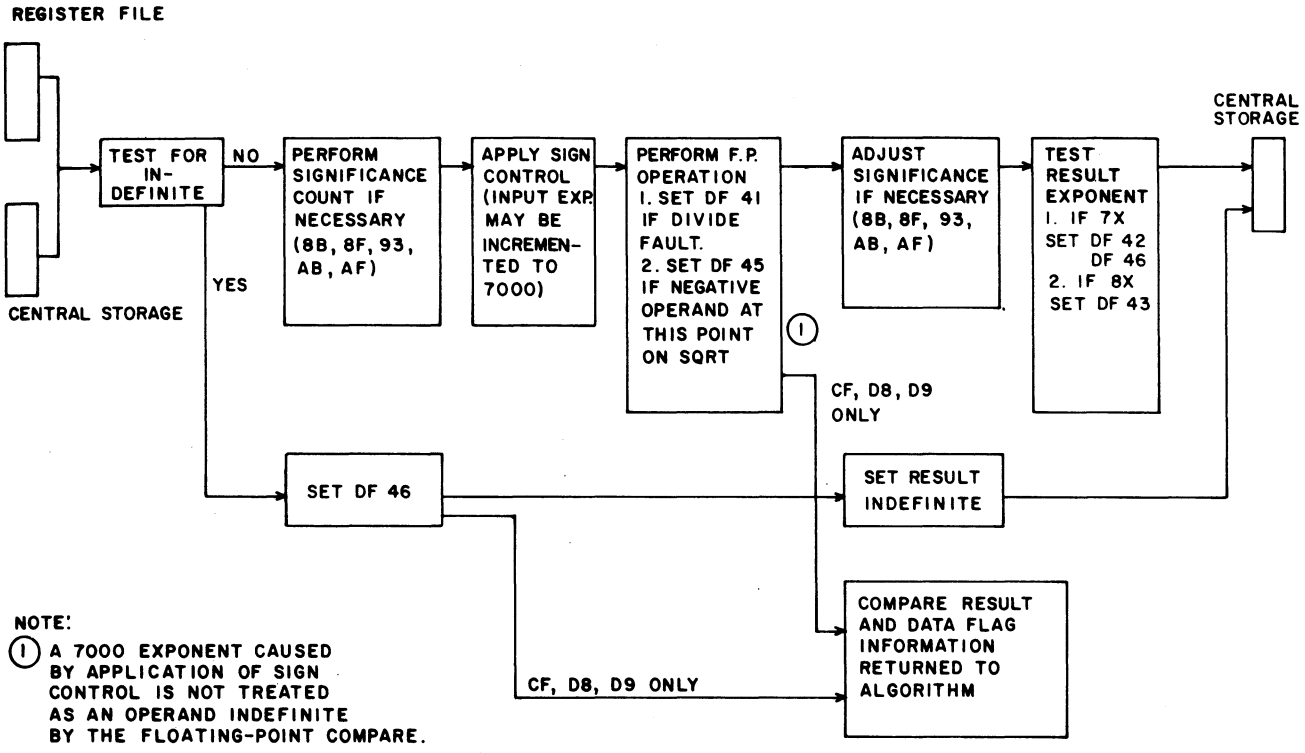


Figure C-1. Operand Flow For Instructions Having Sign Control

INSTRUCTION TERMINATING CONDITIONS

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

The following paragraphs and tables address the termination of multiple operand instructions. Sparse vector instructions terminate as follows:

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

The string instruction terminates when vector C (the result string) is exhausted.

The tables are arranged according to the general instruction types and that the instruction codes within that type are grouped, as much as possible, according to common data field terminating conditions.

Note that in tables C-6 through C-12, M-zero and N-one designate machine zero and normalized one, respectively. In addition, the availability of a control vector for the result field is (C or Z) designated by a yes or no and in the case of the vector macro or nontypical instructions, the yes condition is followed by an I or O designator if the control vector applies to an input or output, respectively.

NOTE

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

TABLE C-6. VECTOR INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field			B Field			C Field		
	Result if A Field is Exhausted	Type of Extension (If Any)	A Field Length Initially Zero	Result if B Field is Exhausted	Type of Extension (If Any)	B Field Length Initially Zero	Result if C Field is Exhausted	C Field Length Initially Zero	Control Vector
80, 81, 82, 83, 84, 85, 86, 87, and 8A	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
88, 89, 8B, 8C, and 8F	Extend	N-One	Extend	Extend	N-One	Extend	Terminate	No-Op	Yes
90, 91, 92, and 93	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes
94 and 95	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
96, 97, 98, 99, and 9A	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes
9B and 9D	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
9C	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes

TABLE C-7. VECTOR MACRO INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field (I)			B Field (I)			C Field (O)		
	Result if A Field is Exhausted	Type of Extension (If any)	A Field Length Initially Zero	Result if B Field is Exhausted	Type of Extension (If any)	B Field Length Initially Zero	Result if C Field is Exhausted	C Field Length Initially Zero	Control Vector
B7	Terminate	NA	No-Op	NA	NA	NA	NA	NA††	No
B8	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes (O)
BA	Terminate	NA	No-Op	NA††	NA	NA	NA	NA	No
C0, C1, C2, and C3	Terminate †	NA	No-Op †	Terminate	NA	No-Op	NA	NA	Yes (I)
D0 and D4	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes (O)
D1 and D5	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes (O)
DA and 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 for reasons other than the exhausting of field length.

††These multipath instructions no-op if Group Length equals zero.

TABLE C-8. STRING INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field (I)			C Field (O)	
	Result if A Field is Exhausted	Type of Extension (If Any)	A Field Length Initially Zero	Result if C Field is Exhausted	C Field Length Initially Zero
F8	Extend	B designator byte	Extend	Terminate	No-Op

TABLE C-9. LOGICAL STRING INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field (I)			B Field (I)			C Field (O)	
	Result if A Field is Exhausted	Type of Extension (If Any)	A Field Length Initially Zero	Result if B Field is Exhausted	Type of Extension (If Any)	B Field Length Initially Zero	Result if C Field is Exhausted	C Field Length Initially Zero
F0, F1, F2, F3, F4, F5, F6, F7	Extend	Zero bits	Extend	Extend	Zero bits	Extend	Terminate	No-Op

TABLE C-10. SPARSE VECTOR INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field (I)			B Field (I)			C Field (O)	
	Result if A/X Field is Exhausted	Type of Extension (If Any)	A/X Field Length Initially Zero	Result if B/Y Field is Exhausted	Type of Extension (If Any)	B/Y Field Length Initially Zero	Result if C/Z Field Exhausted	C/Z Field Length Initially Zero
A0, A1, A2, A4, A5, A6, A8, A9, AB	NA	NA	NA	NA	NA	NA	NA	NA
	X Field (I)			Y Field (I)			Z Field (O)	
AC, AF	Extend	Zero bits	Extend	Extend	Zero bits	Extend	Terminate	No-Op

TABLE C-11. TERMINATING CONDITIONS FOR NONTYPICAL (32-BIT FORMAT) INSTRUCTIONS HAVING MULTIPLE OPERANDS

Instruction Code	R Field (I)		S Field (I)		T Field (O)	
	Result if R Field is Exhausted	R Field Length Initially Zero	Result if S Field is Exhausted	S Field Length Initially Zero	Result if T Field is Exhausted	T Field Length Initially Zero
14	Exit loop	No-Op	Exit loop	Zero R bits skipped	Terminate	No-Op
15 and 16	Exit loop	No-Op	Exit loop	No-Op	Terminate	No-Op
1C and 1D	Exit loop	Strings 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
7D	Terminate data transfer to register file.	No data transfer to register file.	NA	NA	Terminate data transfer from register file.	No data transfer from register file.

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

TABLE C-12. NONTYPICAL (64-BIT FORMAT) INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field (I)			B Field (I)			Z Field (O)		
	Result if A Field is Exhausted	Type of Extension (If Any)	A Field Length Initially Zero	Result if B Field is Exhausted	Type of Extension (If Any)	B Field Length Initially Zero	Result if Z Field is Exhausted	Z Field Length Initially Zero	Control Vector
BB, BC, and BD	NA	NA	NA	NA	NA	NA	Terminate (I)	No-Op (I)	No
C4, C5, C6, and C7	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate (O)	No-Op (O)	No
C8, C9, CA, and CB	Terminate	NA	No-Op	Exit search iteration	NA	Exit search iteration	NA	NA	Yes (O)
CC	Terminate	NA	No-Op	NA	NA	NA	NA	NA	No
CF	Terminate	NA	No-Op	Extend	M-Zero	Extend	NA	NA	No
D8 and D9	Terminate	NA	No-Op	NA	NA	NA	NA	NA	Yes (I)

DATA FLAG APPLICATIONS TO INSTRUCTIONS D

INSTR CODE ↓	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
00 01 02 03										
04 05 06 07									X	
08 09 0A 0B										
0C 0D 0E 0F										
10 11 12 13			X							
14 15 16 17										
13 19 1A 1B										
1C 1D 1E 1F										X

INSTR CODE ↓	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
20 21 22 23								X X X X		
24 25 26 27								X X X X		
28 29 2A 2B										X
2C 2D 2E 2F										
30 31 32 33										
34 35 36 37										
38 39 3A 3B										
3C 3D 3E 3F										

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

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
60					X	X		X		
61					X	X		X		
62					X	X		X		
63										
64					X	X		X		
65					X	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										

INSTR CODE ↓	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
80					X	X		X		
81						X		X		
82					X	X		X		
83										
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										

INSTR CODE ↓	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
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		
B0								X		
B1								X		
B2								X		
B3								X		
B4								X		
B5								X		
B6										
B7										
B8										
B9										
BA										
BB										
BC										
BD										
BE										
BF										

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
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										
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		
DC					X	X		X		
DD										
DE										
DF					X	X		X		

INSTR CODE	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
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										

INDEX

Absolute bounds address 2-25
Add unit 2-33
Addressing modes 5-13
Associative unit 2-8
Associative microcode memory 2-16
Associative registers 5-19

Bank address 2-50
Bank busy checks 2-3
Branch instructions 4-50

Central memory operation 2-48
Central processor unit (CPU) 2-1
Chassis power 1-2
Clock 2-51
Compare instructions 4-140
Cooling 1-2
Common register file for monitor and job modes 5-33
CPU characteristics 1-4
CPU word address (bits 37 through 58) 2-40

Data flag branch register 5-33
Data flag register control 2-24
Delay unit 2-35

External interrupts 5-5

Field length registers 2-30
Floating-point pipeline 2-31
Functional descriptions 2-1

General definitions and programming guides 5-40

Halts/interrupts 2-30

Index instructions 4-32
Inputs to VSU 2-20
Instruction descriptions 4-1
Instruction issue unit 2-6
Instruction stack 2-6
Interfacing between SCA and SCI 2-54
Interrupt counters 2-23
Interrupt and branch control 2-22
Interrupts 5-3
Invisible package 5-6
I/O ports 2-42
I/O priority 2-47

Job interval timer 5-23

Load/store unit 2-12
Logical string instructions 4-118
Logical unit 2-35

Maintenance control unit (MCU) 2-54
Major system components description 1-5
Measurements and weights 1-2
Memory degradation 2-52
Memory interface 2-52
Monitor and job modes 5-1
Monitor instructions 4-155

Nontypical instructions 4-122

Operating instructions 3-1

Priority unit 2-2
Programming information 5-1

Real-time counters 5-22
Register file 5-24

Register file read/writes 2-30
Register instructions 4-34
Retry unit 2-25
RNS/branch 2-6

Scalar 2-2
Scalar floating-point 2-13
Scalar microcode memories 2-15
SECDED 2-40
SECDED error latching hardware 2-41
Shift unit 2-35
Sparse vector instructions 4-88
Startup procedures 3-1
Storage and maintenance access 2-47
Stream input operation 2-28
Stream addressing pipeline (APL) 2-26
Stream output operation 2-28
String instruction 4-118
System channel adapter (SCA) 2-13
System channel interface (SCI) 2-54
System stop 3-4
Systems communication 2-45

Timers 2-23
Vector instructions 4-64
Vector macro instructions 4-101
Vector processor 2-18
Vector setup and recovery unit (VSU) 2-18
Vector stream input (VST) 2-29
Vector stream output (VSW) 2-37
VSU operation 2-21
VSU microcodes 2-26

Write one (pipelines and register file) 2-37
Write two (VSS,string) 2-38

COMMENT SHEET

MANUAL TITLE: CDC CYBER 200 Model 205 Computer System
Hardware Reference Manual
PUBLICATION NO.: 60256020 **REVISION:** C

NAME: _____

COMPANY: _____

STREET ADDRESS: _____

CITY: _____ **STATE:** _____ **ZIP CODE:** _____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

☐ Please Reply ☐ No Reply Necessary

CUT ALONG LINE

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

FOLD FOLD

CORPORATE HEADQUARTERS P.O. BOX 0 MINNEAPOLIS, MINNESOTA 55440