60458890

CONTROL DATA

CDC® CYBER 170 COMPUTER SYSTEMS MODELS 815, 825, 835, 845, AND 855

CDC® CYBER 180 COMPUTER SYSTEMS MODELS 810, 830, 835, 840, 845, 850, 855, 860, AND 990

VIRTUAL STATE

VOLUME II
INSTRUCTION DESCRIPTIONS
PROGRAMMING INFORMATION

HARDWARE REFERENCE MANUAL

REVISION RECORD

REVISION	DESCRIPTION
01 (06-06-83)	Manual released.
A (04-15-84)	Manual updated to add support of CYBER 170 Model 845 and CYBER 180 Models 810, 830, 835, 845, 855, and 990. Due to extensive changes, revision bars and dots are not used and all pages reflect the latest revision level. This edition obsoletes all previous editions.
B (11-02-84)	Manual updated to add support of CYBER 180 Models 840, 850, and 860.
C (05-03-85)	Manual revised; includes Engineering Change Order 46891. Front Cover through 5, 7/8, 14, 16, II-1-30, II-2-1, II-2-78, II-2-127, II-2-129, II-2-130, II-2-134, II-2-136 through II-2-138, and II-2-145 through II-2-147 are revised. Page II-2-149 is added.

Publication No.
60458890

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	-
Title Page	-
2	C
3	C
4	C
5	C
6	A
7/8	C
9	B
10	A
11	A
12	A
13	B
14	C
15	B
16	C
II-1-1	A
II-1-2	A
II-1-3	A
II-1-4	A
II-1-5	B
II-1-6	A
II-1-7	A
II-1-8	A
II-1-9	B
II-1-10	B
II-1-11	A
II-1-12	A
II-1-13	A
II-1-14	A
II-1-15	A
II-1-16	A
II-1-17	A
II-1-18	A
II-1-19	A
II-1-20	A
II-1-21	A
II-1-22	B
II-1-23	B
II-1-24	B
II-1-25	B
II-1-26	B
II-1-26.1/ II-1-26.2	B
II-1-27	A
II-1-28	A
II-1-29	A
II-1-30	C
II-1-31	A
II-1-32	A
II-1-33	A
II-1-34	A
II-1-35	A
II-1-36	A
II-1-37	A
II-1-38	A
II-1-39	A
II-1-40	A
II-1-41	A
II-1-42	A
II-1-43	A
II-1-44	A
II-1-45	A
II-1-46	A
II-1-47	A
II-1-48	A
II-1-49	A
II-1-50	A
II-1-51	A
II-1-52	A
II-1-53	A
II-1-54	A
II-1-55	A
II-1-56	A
II-1-57	A
II-1-58	B
II-1-59	A
II-1-60	A
II-1-61	A
II-1-62	A
II-1-63	B
II-1-64	A
II-1-65	A
II-1-66	A
II-1-67	A
II-1-68	A
II-1-69	A
II-1-70	A
II-1-71	A
II-1-72	A
II-1-73	A
II-1-74	A
II-1-75	A
II-1-76	A
II-1-77	A
II-1-78	A
II-1-79	A
II-1-80	A
II-1-81	A
II-1-82	A
II-1-83	A
II-1-84	A
II-1-85	A
II-1-86	A
II-1-87	A
II-1-88	A
II-1-89	A
II-1-90	B
II-1-91	A
II-1-92	A
II-1-93	A
II-1-94	A
II-1-95	A
II-1-96	A
II-1-97	A
II-1-98	A
II-1-99	A
II-1-100	A
II-1-101	A
II-1-102	A
II-1-103	A
II-1-104	A
II-1-105	A
II-1-106	A
II-1-107	A
II-1-108	A
II-1-109	A
II-1-110	A
II-1-111	A
II-1-112	A
II-1-113	A
II-1-114	A
II-1-115	A
II-1-116	A
II-1-117	A
II-1-118	A
II-1-119	A
II-1-120	A
II-1-121	A
II-1-122	A
II-1-123	A
II-1-124	A
II-1-125	A
II-1-126	A
II-1-127	A
II-1-128	A
II-1-129	A
II-1-130	A
II-1-131	A
II-1-132	A
II-1-133	A
II-1-134	A
II-1-135	A
II-1-136	A
II-1-137	A
II-2-1	C
II-2-2	A
II-2-3	A
II-2-4	A
II-2-5	A
II-2-6	A
II-2-7	A
II-2-8	A
II-2-9	A
II-2-10	A
II-2-11	A
II-2-12	A
II-2-13	A
II-2-14	B
II-2-15	A
II-2-16	A
II-2-17	A
II-2-18	A
II-2-19	A
II-2-20	A
II-2-21	A
II-2-22	A
II-2-23	A
II-2-24	A
II-2-25	A
II-2-26	A
II-2-27	A
II-2-28	A
II-2-29	A
II-2-30	A
II-2-31	A
II-2-32	A
II-2-33	A
II-2-34	A
II-2-35	A
II-2-36	A
II-2-37	A
II-2-38	A
II-2-39	A
II-2-40	A
II-2-41	A
II-2-42	A
II-2-43	A
II-2-44	A
II-2-45	A
II-2-46	A
II-2-47	A
II-2-48	A
II-2-49	A
II-2-50	A
II-2-51	A
II-2-52	A
II-2-53	A
II-2-54	A
II-2-55	A
II-2-56	A
II-2-57	A
II-2-58	A
II-2-59	A
II-2-60	A
II-2-61	A
II-2-62	A
II-2-63	A
II-2-64	A
II-2-65	A
II-2-66	A
II-2-67	A
II-2-68	A
II-2-69	A
II-2-70	A
II-2-71	A
II-2-72	A
II-2-73	A
II-2-74	A
II-2-75	A
II-2-76	A
II-2-77	A
II-2-78	C
II-2-79	A
II-2-80	A
II-2-81	A
II-2-82	A
II-2-83	A
II-2-84	A
II-2-85	A
II-2-86	A
II-2-87	A
II-2-88	A
II-2-89	A
II-2-90	A
II-2-91	A
II-2-92	A
II-2-93	A
II-2-94	A
II-2-95	A
II-2-96	A
II-2-97	A
II-2-98	A
II-2-99	A
II-2-100	A
II-2-101	A
II-2-102	A
II-2-103	B
II-2-104	A
II-2-105	A
II-2-106	A
II-2-107	A
II-2-108	A
II-2-109	A
II-2-110	A
II-2-111	A
II-2-112	A
II-2-113	A
II-2-114	A
II-2-115	B
II-2-116	A
II-2-117	A
II-2-118	A
II-2-119	A
II-2-120	A
II-2-121	B
II-2-122	A
II-2-123	A
II-2-124	A
II-2-125	A
II-2-126	A
II-2-127	C
II-2-128	A
II-2-129	C
II-2-130	C
II-2-131	A
II-2-132	A
II-2-133	A
II-2-134	C
II-2-135	A
II-2-136	C
II-2-137	C
II-2-138	C
II-2-139	A
II-2-140	A

PAGE	REV
II-2-141	A
II-2-142	A
II-2-143	A
II-2-144	A
II-2-145	C
II-2-146	C
II-2-147	C
II-2-148	C
II-2-149	C
II-A-1	A
II-A-2	A
II-A-3	A
II-B-1	A
II-B-2	A
II-B-3	A
II-B-4	A
II-B-5	A
II-C-1	B
II-C-2	A
II-C-3	A
II-C-4	A
II-C-5	A
II-C-6	A
II-C-7	A
II-C-8	A
II-C-9	A
II-C-10	A
II-C-11	A
II-D-1	A
II-D-2	A
II-D-3	B
II-D-4	B
II-D-5	B
II-D-6	A
II-D-7	B
II-D-8	A
II-D-9	A
II-D-10	A
II-D-11	A
II-D-12	A
II-D-13	A
II-D-14	A
II-D-15	A
II-D-16	A
Index-1	B
Index-2	B
Index-3	B
Index-4	B
Index-5	B
Index-6	B
Index-7	B
Comment Sheet	C
Back Cover	–

PAGE	REV

PAGE	REV

PAGE	REV

PAGE	REV

This manual contains hardware reference information for the CDC® CYBER 170 Models 815, 825, 835, 845, and 855 computer systems, and the CYBER 180 Models 810, 830, 835, 840, 845, 850, 855, 860, and 990 computer systems, in their Virtual State of operation.

This manual provides model-independent instruction descriptions and programming information relative to the computer systems hardware. Additional hardware reference information regarding operation of the computer systems in both their CYBER 170 State and Virtual State environments is available in manuals listed in the system publications index on the following page.

AUDIENCE

This manual is for use by programming and engineering services personnel who operate, program, and maintain the computer systems.

Other manuals applicable to the CYBER 170 and CYBER 180 computer systems are:

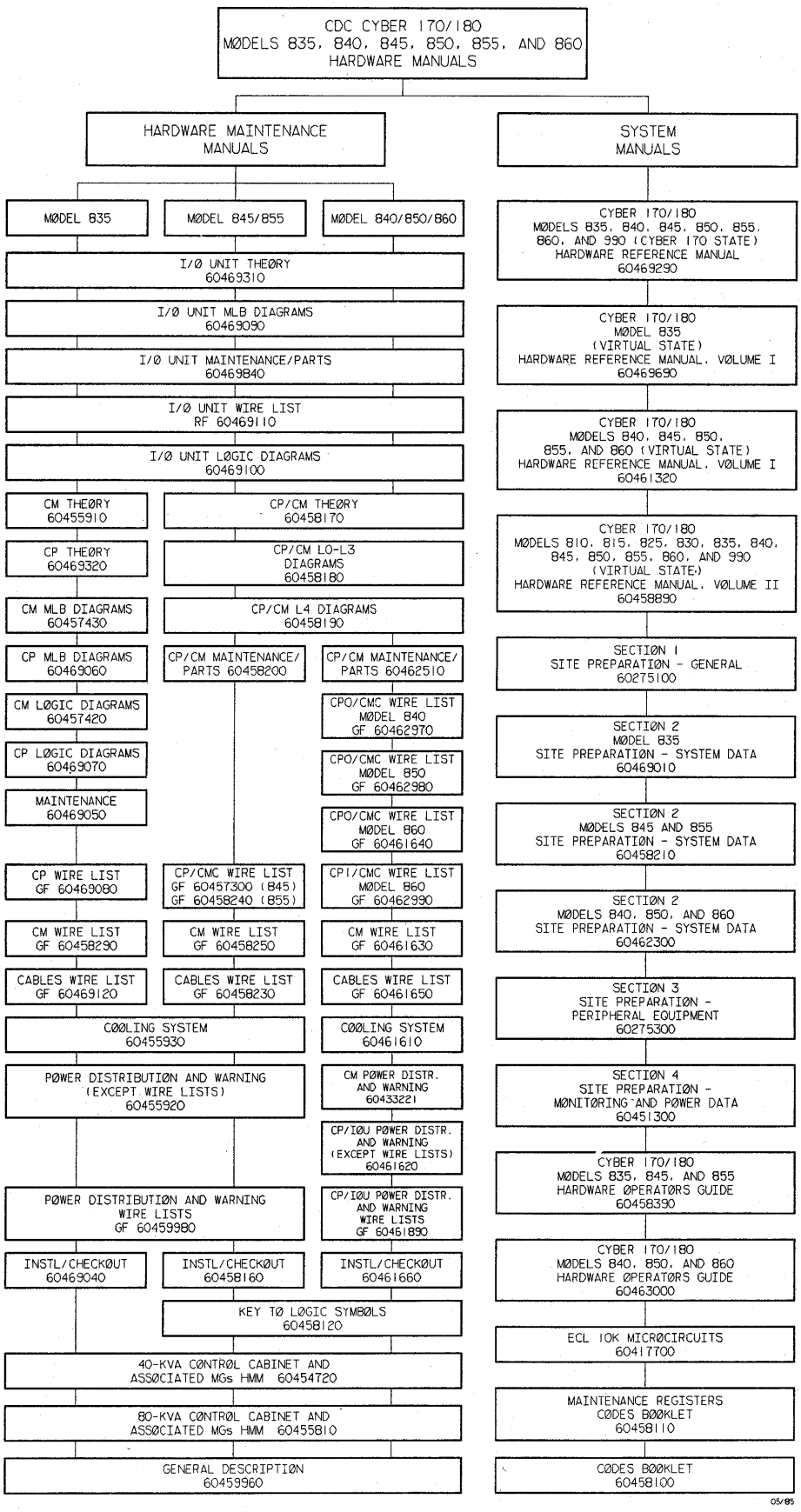
Control Data Publication	Publication Number
NOS Version 2 Operator/Analyst Handbook	60459310
NOS Version 2 Systems Programmer's Instant	60459370
NOS Version 1 Operator's Guide	60457700
NOS Version 1 Systems Programmer's Instant	60457790
NOS/BE Version 1 Operator's Guide	60457380
NOS/BE Version 1 System Programmer's Reference Manual, Volume 1	60458480
NOS/BE Version 1 System Programmer's Reference Manual, Volume 2	60458490
NOS/VE Analysis Usage	60463915
NOS/VE Operations Usage	60463914
Codes Booklet	60458100
Maintenance Register Codes Booklet	60458110
CDC® 721 Enhanced Display Terminal (CC634B) HRM	62950102

Publication ordering information and latest revision levels are available from the Literature Distribution and Services catalog, publication number 90310500.

WARNING

This equipment generates, uses and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation, it has not been tested for compliance with the limits for Class A computing device pursuant to Subpart J of Part 15 of the FCC Rules which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

SYSTEM PUBLICATIØN INDEX



CONTENTS

1. INSTRUCTION DESCRIPTIONS II-1-1

Virtual State CP Instructions II-1-1
CP Instruction Formats II-1-1
Instruction Description Nomenclature II-1-2
Interrupts II-1-3
CP General Instructions II-1-4
CP Load and Store Instructions II-1-5
Load/Store Multiple II-1-6
Load/Store Word II-1-7
Load/Store Word, Indexed II-1-7
Load/Store Address II-1-8
Load/Store Address, Indexed II-1-8
Load/Store Bytes II-1-9
Load/Store Bytes, Immediate II-1-9
Load Bytes, Relative II-1-9
Load/Store Bit II-1-10
CP Integer Arithmetic Instructions II-1-11
Half-Word Integer Sum II-1-12
Integer Sum II-1-13
Half-Word Integer Difference II-1-14
Integer Difference II-1-14
Half-Word Integer Product II-1-14
Integer Product II-1-15
Half-Word Integer Quotient II-1-15
Integer Quotient II-1-16
Half-Word Integer/Integer Compare II-1-16
CP Branch Instructions II-1-17
Branch Relative II-1-17
Branch Intersegment II-1-18
Branch on Half-Word II-1-18
Branch II-1-19
Branch and Increment II-1-19
Branch on Segments Unequal II-1-20
CP Copy Instructions II-1-20
Copy Address II-1-21
Copy Half Word II-1-21
Copy Full Word II-1-22
CP Address Arithmetic Instructions II-1-22
Address Increment, Indexed II-1-22
Address Increment, Signed Immediate II-1-23
Address Relative II-1-23
Address Increment, Modulo II-1-23
CP Enter Instructions II-1-23
Enter Zeros/Ones/Signs II-1-24
Enter, Immediate Positive/Negative II-1-24
Enter X1/X0, Immediate Logical II-1-24
Enter X1/X0, Signed Immediate II-1-25
Enter, Signed Immediate II-1-25
CP Shift Instructions II-1-25
Shift Word, Circular II-1-26
Shift End-Off, Word/Half-Word II-1-27
CP Logical Instructions II-1-27
Logical Sum/Difference/Product II-1-28
Logical Complement II-1-28
Logical Inhibit II-1-29
CP Register Bit String Instructions II-1-29
Bit String Descriptor II-1-29
Isolate Bit Mask II-1-30
Isolate II-1-30
Insert II-1-30
CP Mark to Boolean Instruction II-1-30
BDP Instruction Descriptions II-1-31
BDP Nomenclature II-1-32
BDP Numeric Instructions II-1-32
Decimal Arithmetic II-1-34
Decimal Compare II-1-35
Numeric Move II-1-36
Decimal Scale II-1-37
BDP Byte Instructions II-1-38
Byte Compare II-1-39
Byte Translate II-1-40
Move Bytes II-1-40
Edit II-1-41
Edit Mask II-1-42
Edit Operation II-1-42
MOP Description Nomenclature II-1-42
End Suppression Toggle II-1-43
Special Characters Table II-1-43
Symbol II-1-43
Negative Sign Toggle II-1-43
Zero Field Toggle II-1-43
Skipping of Signs II-1-44
Microoperation 0 II-1-44
Microoperation 1 II-1-44
Microoperation 2, 3 II-1-44
Microoperation 4 II-1-44
Microoperation 5 II-1-44
Microoperation 6 II-1-45
Microoperation 7 II-1-45

Microoperation 8 II-1-45
Microoperation 9 II-1-45
Microoperation A II-1-46
Microoperation B II-1-46
Microoperation C II-1-46
Microoperation D II-1-46
Microoperation E II-1-47
Microoperation F II-1-47
Edit Function NUMERIC Termination of the Edit Instruction II-1-47
Byte Scan While Nonmember II-1-48
BDP Subscript and Immediate Data Instructions II-1-48
Calculate Subscript and Add II-1-49
Move Immediate Data II-1-50
Compare Immediate Data II-1-51
Add Immediate Data II-1-52
Floating-Point Instruction Descriptions II-1-52
Double-Precision Register Designators II-1-53
Floating-Point Conversion Instructions II-1-53
Convert From Integer to FP II-1-53
Convert From FP to Integer II-1-53
Floating-Point Arithmetic Instructions II-1-54
Floating-Point Sum/Difference II-1-55
Floating-Point Product II-1-56
Floating-Point Quotient II-1-57
Floating-Point Branch II-1-58
Normal Exit II-1-58
Branch Exit II-1-58
Group Interrupt Conditions II-1-58
Floating-Point Branch on Comparison II-1-59
Floating-Point Branch on Condition II-1-59
Floating-Point Compare II-1-60
Vector Instruction Descriptions II-1-60
Vector Instruction Format II-1-61
Integer Vector Arithmetic II-1-63
Integer Vector Compare II-1-63
Logical Vector Arithmetic II-1-64
Integer/Floating-Point Vector Conversion II-1-64
Floating-Point Vector Arithmetic II-1-64
Special Purpose Vector Instructions II-1-65
System Instruction Descriptions II-1-71
Nonprivileged System Instructions II-1-72
Program Error II-1-72
Scope Loop Sync II-1-73
Exchange II-1-73
Return II-1-74
Pop II-1-75
Copy Free Running Counter II-1-76
Test and Set Bit II-1-77
Test and Set Page II-1-77
Call Relative II-1-78
Compare Swap II-1-80
Call Indirect II-1-81
Reserved Operation Codes II-1-83
Execute Algorithm II-1-83
Local Privileged System Instructions II-1-83
Load Page Table Index II-1-84
Global Privileged System Instruction II-1-84
Processor Interrupt II-1-84
Monitor Mode Instructions II-1-85
Mixed-Mode Instructions II-1-86
Purge Buffer II-1-86
Copy to/from State Buffer II-1-87
Branch on Condition Register II-1-88
Peripheral Processor Instruction Descriptions II-1-90
PP Instruction Formats II-1-90
PP Data Format II-1-90
PP Relocation Register Format II-1-91
PP Load and Store Instructions II-1-93
PP Arithmetic Instructions II-1-97
PP Logical Instructions II-1-103
PP Replace Instructions II-1-108
PP Branch Instructions II-1-113
PP Central Memory Access Instructions II-1-116
PP Input/Output Instructions II-1-124
Other IOU Instructions II-1-135
Exchange Jumps II-1-136

2. PROGRAMMING INFORMATION II-2-1

CP Exchange Operations II-2-1
Virtual State Job-to-Monitor Exchange Operations II-2-3
Virtual State Monitor-to-Job Exchange Operations II-2-3
Exchange Packages II-2-3
CP Registers II-2-6
Process State Registers II-2-6
CP Base Constant (BC) Register II-2-6
CP Debug Index (DI) Register II-2-6
CP Debug List Pointer (DLP) Register II-2-6
CP Debug Mask Register (DM) Register II-2-7
CP Flag Register II-2-8
Critical Frame Flag (CFF) II-2-8

On-Condition Flag (OCF) II-2-8
Process-Not-Damaged (PND) Flag II-2-8
CP Largest Ring Number (LRN) Register II-2-8
CP Last Processor Identification (LPID) Register II-2-8
CP Monitor Condition Register (MCR) II-2-8
CP Monitor Mask Register II-2-8
Operand X Registers II-2-9
CP Process Interval Timer (PIT) II-2-9
CP Program Address (P) Register II-2-9
CP Segment Table Address (STA) Register II-2-10
CP Segment Table Length (STL) Register II-2-10
CP Top-of-Stack (TOS) Pointer Register II-2-10
CP Trap Enable (TE) Register II-2-10
CP Trap Pointer (TP) Register II-2-10
CP Untranslatable Pointer (UTP) Register II-2-11
CP Untranslatable Virtual Machine Identifier (UVMID) Register II-2-11
CP User Condition Register (UCR) II-2-11
CP User Mask Register (UMR) II-2-11
CP Virtual Machine Identifier (VMID) Register II-2-12
CP Processor State Registers II-2-12
CP Options Installed (OI) Register II-2-13
CP Page Size Mask (PSM) Register II-2-13
CP Page Table Address (PTA) Register II-2-13
CP Page Table Length (PTL) Register II-2-13
CP Processor Fault Status (PFS) Registers II-2-13
CP Processor Identifier (PID) Register II-2-13
CP Processor Test Mode (PTM) Register II-2-13
CP Status Summary (SS) Register II-2-13
CP Cache/Map Corrected Error Log (CCEL/MCEL) Register II-2-14
CP Dependent Environment Control (DEC) Register II-2-14
CP Element Identifier (EID) Register II-2-14
CP Job Process State (JPS) Register II-2-14
CP Model Dependent Word (MDW) Register II-2-14
CP Monitor Process (MPS) Register II-2-15
CP System Interval Timer (SIT) Register II-2-15
CP Virtual Machine Capability List (VMCL) II-2-15
CM Registers II-2-16
CM Corrected Error Log (CEL) Register II-2-16
CM Element Identifier (EID) Register II-2-17
CM Environment Control (EC) Register II-2-17
CM Free-Running Counter Register II-2-17
CM Options Installed (OI) Register II-2-17
CM Port Bounds Register II-2-17
CM Status Summary Register II-2-17
CM Uncorrectable Error Log (UEL) Register II-2-18
IOU Registers II-2-18
IOU Element Identifier (EID) Register II-2-19
IOU Environment Control (EC) Register II-2-19
IOU Fault Status (FS) Registers II-2-19
IOU Fault Status Mask Register II-2-19
IOU Options Installed (OI) Register II-2-19
IOU OS Bounds Register II-2-19
IOU Status Summary Register II-2-20
IOU Test Mode (TM) Register II-2-20
CP Condition and Mask Registers II-2-20
CP Condition Register Bit Grouping II-2-23
CP Interrupts II-2-25
Exchange Interrupts II-2-25
Trap Interrupts II-2-25
Interrupt Conditions II-2-26
Access Violation (MCR 54) II-2-26
Address Specification Error (MCR 52) II-2-27
Arithmetic Loss-of-Significance (UCR 62) II-2-27
Arithmetic Overflow (UCR 57) II-2-27
Critical Frame Flag (UCR 53) II-2-28
Debug (UCR 56) II-2-28
Divide Fault (UCR 55) II-2-28
Environment Specification Error (MCR 55) II-2-28
Exponent Overflow (UCR 58) II-2-29
Exponent Underflow (UCR 59) II-2-29
External Interrupt (MCR 56) II-2-29
Floating-Point Indefinite (UCR 61) II-2-29

Floating-Point Loss-of-Significance (UCR 60) II-2-30
Free Flag (UCR 50) II-2-30
Instruction Specification Error (MCR 51) II-2-30
Inter-Ring Pop (UCR 52) II-2-31
Invalid BDP Data (UCR 63) II-2-31
Invalid Segment/Ring Number Zero (MCR 60) II-2-31
Not Assigned (MCR 49) II-2-31
Outward Call/Inward Return (MCR 61) II-2-31
Page Table Search Without Find (MCR 57) II-2-32
Privileged Instruction Fault (UCR 48) II-2-32
Process Interval Timer (UCR 51) II-2-32
Detected Uncorrectable Error (MCR 48) II-2-32
CYBER 170 State Exchange Request (MCR 53) II-2-32
Short Warning (MCR 50) II-2-32
Soft Error Log (MCR 62) II-2-33
System Interval Timer (MCR 59) II-2-33
Trap Exception (MCR 63) II-2-33
Unimplemented Instruction (UCR 49) II-2-33
Multiple Interrupt Conditions II-2-33
Flags II-2-35
Stack Manipulating Operations II-2-36
Stack Frames and Save Areas II-2-36
Stack Frame Save Area Format II-2-36
Stack Frame Save Area Descriptor Field II-2-37
Virtual Machine Identifier (VMID) Field II-2-38
User Mask/Condition and Monitor Condition Fields II-2-39
Assigned Registers During Stack Operation II-2-39
Top-of-Stack Pointers II-2-39
Dynamic Space Pointer (A0) II-2-39
Current Stack Frame Pointer (A1) II-2-39
Previous Save Area Pointer (A2) II-2-39
Binding Section Pointer (A3) II-2-40
Argument Pointer (A4) II-2-40
Exceptions During Stack Operations II-2-40
Business Data Processing Programming II-2-40
BDP Data Descriptors II-2-40
BDP Data Types II-2-41
Data Type 0: Packed Decimal, Unsigned II-2-42
Data Type 1: Packed Decimal, Unsigned Slack Digit II-2-42
Data Type 2: Packed Decimal, Signed II-2-43
Data Type 3: Packed Decimal, Signed, Slack Digit II-2-43
Data Type 4: Unpacked Decimal, Unsigned II-2-43
Data Type 5: Unpacked Decimal, Trailing Sign Combined Hollerith II-2-44
Data Type 6: Unpacked Decimal, Trailing Sign Separate II-2-44
Data Type 7: Unpacked Decimal, Leading Sign Combined Hollerith II-2-44
Data Type 8: Unpacked Decimal, Leading Sign Separate II-2-44
Data Type 9: Alphanumeric II-2-45
Data Type 10: Binary, Unsigned II-2-45
Data Type 11: Binary, Signed II-2-45
Slack Digit II-2-45
Undefined Results II-2-45
Overlap II-2-45
Invalid Data II-2-45
Vector Programming II-2-46
Vector Length (Number of Operations) II-2-47
Vector Page Size II-2-48
Vector Broadcast II-2-48
Vector Interrupts II-2-48
Vector Overlap II-2-48
Floating-Point Programming II-2-48
Floating-Point Data Formats II-2-49
Standard and Nonstandard FP Numbers II-2-51
Floating-Point Zero II-2-51
Floating-Point Nonzero II-2-51
Floating-Point Infinite II-2-51
Floating-Point Indefinite II-2-52
Double-Precision Non-standard FP Numbers II-2-52
Exponent Arithmetic II-2-52
Normalization II-2-52
Floating-Point Sum and Difference II-2-52
Floating-Point Multiply II-2-53
Floating-Point Divide II-2-53
Floating-Point End Cases II-2-54
Program Monitoring II-2-64
Debug II-2-64
Debug List II-2-64
Debug List Pointer Register II-2-65

Debug Index Register II-2-65
Debug Mask Register II-2-66
Enabling Debug II-2-68
Debug Scan Operation II-2-68
Interrupts During Debug Scan II-2-69
Debug-Software Interaction, Debug Enabled II-2-70
Debug-Software Interaction, Debug Disabled II-2-70
Virtual and Central Memory Programming II-2-77
Process Virtual Memory II-2-78
System Virtual Memory II-2-78
Real Memory II-2-79
Address Tables II-2-83
Segment Descriptor Table II-2-84
System Page Table II-2-86
Page Table Search II-2-86
Page Table Entries II-2-87
PTE Control Fields II-2-88
PTE Segment/Page Identifier Field II-2-88
PTE Page Frame KMA Field II-2-89
Listing of Pages in Page Table II-2-90
Process Binding Section II-2-90
Access Protection II-2-91
Ring Structure II-2-95
Ring Voting II-2-95
Effect of RN = 0 II-2-96
RN for Read/Write Access II-2-96
RN for Execute Access II-2-96
RN Effect on Pop Instruction II-2-97
Effect of RN Violations II-2-98
Execute Access Privilege/Mode II-2-98
Keys/Locks II-2-99
Interstate Programming II-2-102
Operation in CYBER 170 State II-2-102
Memory Addressing in CYBER 170 State II-2-103
Cache Invalidation in CYBER 170 State (Models 835 through 860 Only) II-2-103
State-Switching Operations II-2-104
Virtual State Monitor Mode-to-CYBER 170 State Exchange II-2-104
CYBER 170 State-to-Virtual State Monitor Mode Exchange II-2-104
Exchanges Within CYBER 170 State II-2-104
Call from Virtual State to CYBER 170 State II-2-105
Trap Interrupt from CYBER 170 State to Virtual State II-2-105
Return from Virtual State to CYBER 170 State II-2-105
Exchange Packages used in CYBER 170 State II-2-107
Interstate Exchange Package II-2-107
Program Address (P) Register II-2-109
Stack Pointers II-2-109
EM Register II-2-109
Flags II-2-110
Unified Extended Memory (UEM) Enable Flag II-2-110
Expanded Addressing Select Flag II-2-110
Enhanced Block Copy Flag II-2-110
Software Flag (Word 4, Bit 28) II-2-110
Instruction Stack Purge Flag II-2-111
Software Flag (Word 4, Bit 26) II-2-111
CYBER 170 State Monitor Flag II-2-111
Exit Mode Halt Flag II-2-111
RAC Register II-2-111
FLC Register II-2-111
Monitor Address (MA) Register II-2-111
Address (A) Registers II-2-112
RAE Register II-2-112
FLE Register II-2-112
Virtual State Ring Numbers II-2-112
Index (B) Registers II-2-112
Operand (X) Registers II-2-112
CYBER 170 State Exchange Package II-2-112
Interstate Stack Frame Save Area II-2-113
Code Modificaton in CYBER 170 State II-2-115
Debug/Performance Monitoring II-2-115
Exception Handling in CYBER 170 State II-2-115
Software Exception Conditions II-2-115
Address Errors II-2-119
Illegal Instructions II-2-120
Extended Memory Transfer Exceptions II-2-120
Hardware Exceptions in CYBER 170 State II-2-121
IOU Peripheral Processor Programming II-2-121
Central Memory Addressing by PPs II-2-121
Absolute and Relocation Addressing II-2-121
OS Bounds Test II-2-122
PP Central Memory Read II-2-122
PP Central Memory Write II-2-122
PP Memory Addressing by PPs II-2-123

Direct 6-Bit Operand II-2-123
Direct 18-Bit Operand II-2-123
Direct 6-Bit Address II-2-123
Direct 12-Bit Address II-2-123
Indexed 12-Bit Address II-2-123
Indirect 6-Bit Address II-2-123
Channel Input/Output Operations II-2-124
Channel Flags II-2-124
Channel Active Flag II-2-124
Register-Full Flag II-2-124
Channel (Marker) Flag II-2-125
Error Flag II-2-125
Programming for Channel Input/Output II-2-125
Inter-PP Communications II-2-126
PP Program Timing Consideration II-2-127
Cache Invalidation II-2-127
Error Detection and Recovery II-2-128
PP Hardware Errors II-2-128
Channel Parity Errors II-2-128
Parity Errors on Output Data II-2-128
Parity Errors on Input Data II-2-129
Timeout II-2-129
Initialization II-2-129
Display Station Programming (Channel 10_8) II-2-130
Keyboard II-2-130
Data Display II-2-130
Character Mode II-2-130
Dot Mode II-2-130
Codes II-2-133
Programming Example II-2-134
Program Timing Consideration II-2-134
Real-Time Clock Programming II-2-136
IOU Dedicated Channels II-2-136
Two-Port Multiplexer Programming II-2-138
Function Words II-2-139
Terminal Select (7XXX) II-2-139
Terminal Deselect (6XXX) II-2-139
Calendar Clock/Auto Dial-Out (1XXX) II-2-140
Read Summary Status (00XX) II-2-141
PP Terminal Data (01XX) II-2-141
PP Write Output Buffer (02XX) II-2-142
Set Operation Mode to Terminal (03XX) II-2-142
Set/Clear Data Terminal Ready (DTR) (04XX) II-2-143
Set/Clear Request to Send (RTS) (05XX) II-2-143
Master Clear (07XX) II-2-143
Programming Considerations II-2-143
Data Output II-2-143
Data Input II-2-144
Maintenance Channel Programming II-2-144
MCH Function Words II-2-144
MCH Control Words II-2-145
MCH Programming for Halt/Start (Opcode 0/1) II-2-145
MCH Clear LED (Opcode 3) II-2-145
MCH Programming for Read/Write (Opcode 4/5) II-2-146
MCH Programming for Master Clear/Clear Errors (Opcode 6/7) II-2-147
MCH Echo (Opcode 8) II-2-147
MCH Programming for Read IOU Summary Status (Opcode C,IOU Only) II-2-147

APPENDIXES

A. GLOSSARY II-A-1

B. EDIT EXAMPLES II-B-1

C. INTERFACE INFORMATION II-C-1

Interfaces II-C-1
Twelve-Bit External Interface II-C-1
Maintenance Channel Interface II-C-1
Two-Port Multiplexer Interface II-C-1
Signals II-C-2
Twelve-Bit Channel Control Signals II-C-2
Maintenance Channel Signals II-C-3
Control Signals II-C-3
Signals and Cables II-C-3
Data Signals II-C-5
PP and Channel Interaction II-C-5
Active Flag II-C-5
Full Flag II-C-5
Function Instructions II-C-5
External Channel Input/Output Sequences II-C-5

D. INSTRUCTION INDEX II-D-1

INDEX

FIGURES

II-1-1	Vector Instruction Format	II-1-61
II-1-2	Gather Instruction	II-1-67
II-1-3	Scatter Instruction	II-1-69
II-1-4	PP Instruction Formats and Nomenclature	II-1-91
II-1-5	PP Data Format	II-1-92
II-1-6	PP Relocation Register Format	II-1-92
II-1-7	Relocation and Address Formation	II-1-117
II-2-1	CP Calls, Returns and Interrupts	II-2-2
II-2-2	Virtual State Exchange Package	II-2-5
II-2-3	Interrupt Flowchart	II-2-34
II-2-4	Format of X0 for Call Instructions	II-2-37
II-2-5	Virtual State Stack Frame Save Area	II-2-37
II-2-6	Stack Frame Save Area Descriptor	II-2-38
II-2-7	BDP Data Descriptor Format	II-2-41
II-2-8	Floating-Point Data Formats	II-2-49
II-2-9	Debug List Entry	II-2-67
II-2-10	Debug Condition Select	II-2-66
II-2-11	Central Memory Addressing from CP	II-2-77
II-2-12	Process Virtual Address (PVA) Format	II-2-78
II-2-13	System Virtual Address (SVA) Format	II-2-79
II-2-14	Segment/Page Identifier (SPID) Format	II-2-80
II-2-15	Real Memory Address (RMA) Format	II-2-81
II-2-16	Virtual BN-to-Page Number/ Page Offset Conversion	II-2-82
II-2-17	PVA-to-RMA Conversion	II-2-83
II-2-18	Segment Descriptor Table Entry Format	II-2-85
II-2-19	Page Table Search, Start RMA Formation	II-2-87
II-2-20	Page Table Entry Format	II-2-89
II-2-21	Code Base Pointer Format	II-2-91
II-2-22	PVA-to-SVA Conversion, Read/Write	II-2-93
II-2-23	PVA-to-SVA Conversion, Execute	II-2-94
II-2-24	Call Indirect Access Requirements	II-2-101
II-2-25	Interstate Calls, Returns and Interrupts	II-2-106
II-2-26	Interstate Exchange Package	II-2-108
II-2-27	CYBER 170 State Exchange Package	II-2-113
II-2-28	Interstate Stack Frame Save Area	II-2-114
II-2-29	Display Station Output Function Code	II-2-133
II-2-30	Coordinate Data Word	II-2-133
II-2-31	Character Data Word	II-2-133
II-2-32	Receive and Display Program Flowchart	II-2-135
II-2-33	IOU Dedicated Channels, Models 810, 815, 825, and 830	II-2-136
II-2-34	IOU Dedicated Channels, Models 835, 840, 845, 850, 855, 860, and 990	II-2-137
II-C-1	Data Sequences Timing	II-C-11

TABLES

II-1-1	CP Load and Store Instructions	II-1-5
II-1-2	CP Integer Arithmetic Instructions	II-1-11
II-1-3	CP Branch Instructions	II-1-17
II-1-4	CP Copy Instructions	II-1-20
II-1-5	CP Address Arithmetic Instructions	II-1-22
II-1-6	CP Enter Instructions	II-1-23
II-1-7	CP Shift Instructions	II-1-25
II-1-8	CP Logical Instructions	II-1-27
II-1-9	CP Register Bit String Instructions	II-1-29
II-1-10	Compare j Field and X1 Bits 32 and 33	II-1-31
II-1-11	BDP Numeric Instructions	II-1-32
II-1-12	BDP Divide Fault	II-1-35
II-1-13	BDP Byte Instructions	II-1-38
II-1-14	BDP Subscript and Immediate Data Instructions	II-1-48
II-1-15	Floating-Point Conversion Instructions	II-1-53
II-1-16	Floating-Point Arithmetic Instructions	II-1-54
II-1-17	Floating-Point Branch Instructions	II-1-58
II-1-18	Vector Instructions	II-1-62
II-1-19	Nonprivileged Instructions	II-1-72

II-1-20	Local Privileged Instructions	II-1-83
II-1-21	Mixed Mode Instructions	II-1-86
II-1-22	PP Load and Store Instructions	II-1-93
II-1-23	PP Arithmetic Instructions	II-1-98
II-1-24	PP Logical Instructions	II-1-103
II-1-25	PP Replace Instructions	II-1-108
II-1-26	PP Branch Instructions	II-1-113
II-1-27	PP Central Memory Access Instructions	II-1-116
II-1-28	PP Input/Output Instructions	II-1-124
II-2-1	Process State Registers	II-2-7
II-2-2	Processor State Registers	II-2-12
II-2-3	CM Maintenance Registers	II-2-16
II-2-4	IOU Maintenance Registers	II-2-18
II-2-5	Monitor Condition/Mask Register Bit Assignments	II-2-21
II-2-6	User Condition/Mask Register Bit Assignments	II-2-22
II-2-7	Interrupt Condition Groups	II-2-24
II-2-8	Condition of Flags Following Call, Return, Pop, Exchange, and Trap Operations	II-2-35
II-2-9	BDP Operand Types and Field Lengths	II-2-42
II-2-10	Vector Operations	II-2-47
II-2-11	Floating Point Representation	II-2-50
II-2-12	FP Compare Results	II-2-55
II-2-13	FP Sum Results, UM Clear	II-2-56
II-2-14	FP Sum Results, UM Set	II-2-57
II-2-15	FP Difference Results, UM Clear	II-2-58
II-2-16	FP Difference Results, UM Set	II-2-59
II-2-17	FP Product Results, UM Clear	II-2-60
II-2-18	FP Product Results, UM Set	II-2-61
II-2-19	FP Quotient Results, UM Clear	II-2-62
II-2-20	FP Quotient Results, UM Set	II-2-63
II-2-21	Debug Conditions	II-2-72
II-2-22	System Instruction Privilege and Mode	II-2-99
II-2-23	CYBER 170 State Exceptions	II-2-117
II-2-24	Keyboard Character Codes	II-2-131
II-2-25	Display Character Codes	II-2-132
II-2-26	MCH Function Word Bit Assignments	II-2-148
II-C-1	Maintenance Channel Signals	II-C-3
II-C-2	Data Input Sequence	II-C-6
II-C-3	Data Output Sequence	II-C-7
II-C-4	MCH Input Sequence	II-C-8
II-C-5	MCH Output Sequence	II-C-9

INSTRUCTION DESCRIPTIONS 1

This section contains the Virtual State CP instruction descriptions and PP instruction descriptions.

VIRTUAL STATE CP INSTRUCTIONS

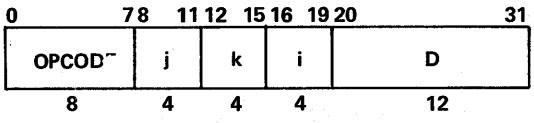
The Virtual State CP instructions comprise the following five groups:

- General
- Business data processing (BDP)
- Floating-point (FP)
- Vector
- System

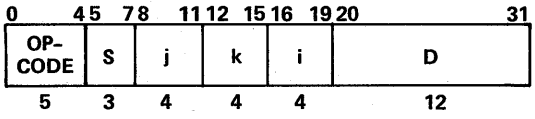
CP INSTRUCTION FORMATS

The CP instructions are 16 or 32 bits long, and have 4 basic formats.

Format jkiD (32 Bits)



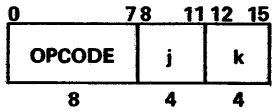
Format SjkiD (32 Bits)



Field	Description
Opcode	Operation code.
j,k,i	Register designators.
D	Signed shift count, positive displacement, or bit string descriptor.
S	Suboperation code.

Business data processing (BDP) instructions using these formats also have one or two 64-bit data descriptor words which are stored in CM immediately after the instruction. (Refer to BDP Data Descriptors in this section.)

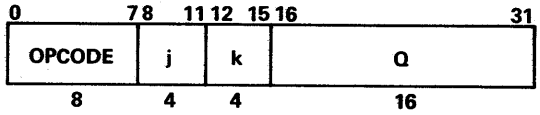
Format jk (16 Bits)



Field	Description
Opcode	Operation code.
j	Register designator, suboperation code, or immediate operand.
k	Register designator or immediate operand.

BDP instructions using this format have two data descriptor words which are stored in CM immediately after the instruction. (Refer to BDP Data Descriptors in this section.)

Format jkQ (32 Bits)



Field	Description
Opcode	Operation code.
j,k	Register designators, suboperation codes, or immediate operand value.
Q	Signed displacement or immediate operand value.

INSTRUCTION DESCRIPTION NOMENCLATURE

The instruction descriptions in this section use the following address, register, and instruction designators:

Designator	Description
j,k,i,Q, D or S	Refer to corresponding field in CP Instruction Formats in this section.
Aj or Ak	One of sixteen 48-bit A registers (A0-AF) specified by j or k field.
Xj or Xk	One of sixteen 64-bit X registers (X0-XF) specified by j or k field.

Designator	Description
XXj or XXk	A double-length X register comprised of Xj and X(j+1), or Xk and X(k+1).
XjL or XkL	Bits 0 through 31 of an X register.
XjR or XkR	Bits 32 through 63 of an X register.
2^n*Xk 2^n*Q 2^n*D	Xk, Q or D left-shifted n places with zero fill on right (e.g., 8*Q shifts Q three places).
BN	Byte number field of a process virtual address.
SEG	Segment number field of a process virtual address.
RN	Ring number field of a process virtual address.
( )	Content of memory location (address is the quantity in parentheses).

Additional designators used with the BDP instructions are listed in BDP Nomenclature in this section.

INTERRUPTS

Refer to CP Interrupts in section 2 of this volume for further information on interrupts. Exceptions caused by instruction execution are listed with instruction descriptions. The following exceptions occur independently of instruction execution and, therefore, are not listed:

Bit	Description
MCR 48	Detected uncorrectable error.
MCR 50	Short warning.
MCR 53	CYBER 170 State exchange request.
MCR 56	External interrupt.
MCR 59	System interval timer.
MCR 62	Soft error log.
MCR 63	Trap exception.
UCR 51	Process interval timer.
UCR 50	Free flag.

CP GENERAL INSTRUCTIONS

The 84 CP general instructions are divided into 10 subgroups. Tables II-1-1 through II-1-10 list the instructions in each subgroup. The subgroups are as follows:

- Load and store.
- Integer arithmetic.
- Branch.
- Copy.
- Address arithmetic.
- Enter.
- Shift.
- Logical.
- Register bit string.
- Mark to Boolean.

CP LOAD AND STORE INSTRUCTIONS

The load and store instructions (table II-1-1) transfer a single bit, byte string, 64-bit word, or multiple 64-bit words between one or more registers and one or more CM locations. Store instructions do not alter any register serving as the source of the data transferred to CM.

Table II-1-1. CP Load and Store Instructions

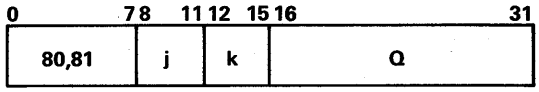
Opcode	Format	Instruction	Mnemonic
80	jkQ	Load multiple	LMULT
81	jkQ	Store multiple	SMULT
82	jkQ	Load word	LX
83	jkQ	Store word	SX
84	jkQ	Load address	LA
85	jkQ	Store address	SA
86	jkQ	Load bytes, relative	LBYTP,j
88	jkQ	Load bit	LBIT
89	jkQ	Store bit	SBIT
A0	jkiD	Load address, indexed	LAI
A1	jkiD	Store address, indexed	SAI
A2	jkiD	Load word, indexed	LXI
A3	jkiD	Store word, indexed	SXI
A4	jkiD	Load bytes	LBYT,X0
A5	jkiD	Store bytes	SBYT,X0
D0-D7	SjkiD	Load bytes, immediate	LBYTS,S
D8-DF	SjkiD	Store bytes, immediate	SBYTS,S

The following interrupt conditions apply to all load and store instructions. Refer to CP Interrupts in section 2 of this volume for descriptions of these conditions.

- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Page table search without find.
- Debug.

Load/Store Multiple

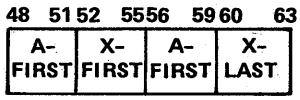
80jkQ	Load Multiple Registers, from (Aj displaced by 8*Q), selectivity per XkR	LMULT Xk,Aj,Q
81jkQ	Store Multiple Registers, at (Aj displaced by 8*Q), selectivity per XkR	SMULT Xk,Aj,Q



These instructions transfer data between the A and X registers and contiguous word locations in CM.

The CM starting address forms by left-shifting Q three places with zero insertion on right, and adding the shifted Q to the byte number (BN) field of the process virtual address (PVA) from Aj.

XkR bits 48 through 63 specify which contiguous A and X registers are transferred as follows:



XkR Bits	Register Transferred
48-51	First A register.
52-55	First X register.
56-59	Last A register.
60-63	Last X register.

The A registers transfer first. When A-first exceeds A-last, no A registers transfer. When X-first exceeds X-last, no X registers transfer.

For example, when: A-first = B_{16}, X-first = 2_{16}
A-last = 4_{16}, X-last = C_{16}

the instruction does not transfer any A registers and transfers X registers 2 through C.

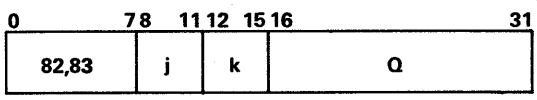
The store multiple instruction clears CM bits 0 through 15 when storing the A registers. The load multiple instruction unconditionally transfers bits 20 through 63 of each CM word to the corresponding bits of the designated A registers. Bits 16 through 19 (RN-field) of each A register are set to the largest of the following:

- Bits 16 through 19 of the CM word.
- Bits 16 through 19 of Aj.
- Bits 8 through 11 (R1 field) of the segment descriptor associated with the PVA in Aj.

During a debug scan operation, the PVA resulting from the addition of Aj and Q is the only data read argument for the load multiple instruction, or the only data write argument for the store multiple registers instruction. Refer to Debug in section 2 of this volume.

Load/Store Word

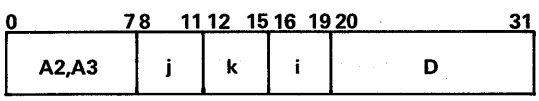
82jkQ	Load Xk, from (Aj displaced by 8*Q)	LX	Xk,Aj,Q
83jkQ	Store Xk, at (Aj displaced by 8*Q)	SX	Xk,Aj,Q



These instructions transfer one word between Xj and CM. The CM address of the word transferred is the sum 8 times Q plus the BN field from Aj.

Load/Store Word, Indexed

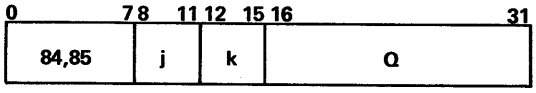
A2jkiD	Load Xk, from (Aj displaced by 8*D and indexed by 8*XiR)	LXI	Xk,Aj,Xi,Q
A3jkiD	Store Xk, at (Aj displaced by 8*D and indexed by 8*XiR)	SXI	Xk,Aj,Xi,Q



These instructions transfer one word between Xk and a word address in CM. The CM address of the word is the BN field from Aj plus 8 times XiR (index), plus 8 times D (displacement). For indexing, these instructions interpret the X0 contents as zeros. Aj bits 61 through 63 must be zeros or an address specification error occurs.

Load/Store Address

84jkQ	Load Ak, from (Aj displaced by Q)	LA	Ak,Aj,Q
85jkQ	Store Ak, at (Aj displaced by Q)	SA	Ak,Aj,Q



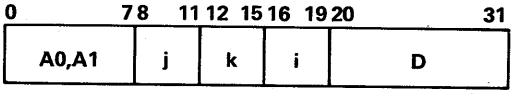
These instructions transfer a 6-byte field between Ak and CM. The field's leftmost byte address is the sum of Q (sign-extended to 32 bits) plus the BN field from Aj.

The load Ak instruction transfers the rightmost 44 bits of the 6-byte CM field to Ak bits 20 through 63. The value transferred to Ak bits 16 through 19 is the largest of the following:

- Leftmost 4 bits of the 6-byte CM field.
- Initial Aj bits 16 through 19.
- Bits 8 through 11 (R1 field) of the segment descriptor associated with the PVA in Aj.

Load/Store Address, Indexed

A0jkiD	Load Ak, from (Aj displaced by D and indexed by XiR)	LAI	Ak,Aj,Xi,D
AljkiD	Store Ak, at (Aj displaced by D and indexed by XiR)	SAI	Ak,Aj,Xi,D



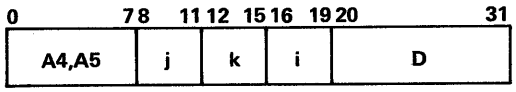
These instructions transfer 6 bytes between Ak and a 6-byte CM field. The starting (leftmost) CM address of the 6-byte field is the sum of the displacement D plus the index value XiR plus the BN field from Aj. For indexing, these instructions interpret the X0 contents as zeros.

The load Ak instruction unconditionally transfers only the rightmost 44 bits of the 6-byte CM field to Ak bit positions 20 through 63. The instruction transfers to Ak bits 16 through 19 a value that is the largest of the following:

- Leftmost 4 bits of the 6-byte CM field.
- Aj bits 16 through 19.
- Bits 8 through 11 (R1 field) of the segment descriptor for the PVA in Aj.

Load/Store Bytes

A4jkiD	Load Bytes, to Xk from (Aj displaced by D and indexed by XiR), length per X0	LBYT,X0 Xk,Aj,Xi,D
A5jkiD	Store Bytes, from Xk at (Aj displaced by D and indexed by XiR), length per X0	SBYT,X0 Xk,Aj,Xi,D

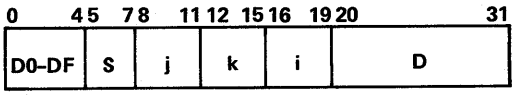


These instructions transfer a field of bytes between Xk and CM. The byte field length equals 1 plus X0 bits 61 through 63. For lengths less than 8, the load byte instruction right-justifies and zero-extends the bytes loaded into Xk.

The beginning (leftmost) CM address of the byte field is the sum of D (zero-extended to 32 bits) plus XiR, plus the BN field from Aj.

Load/Store Bytes, Immediate

DSjkiD	Load Bytes, to Xk from (Aj displaced by D and indexed by XiR), length per S (DS = D0 through D7)	LBYTS,S Xk,Aj,Xi,D
DSjkiD	Store Bytes, from Xk at (Aj displaced by D and indexed by XiR), length per S (DS = D8 through DF)	SBYTS,S Xk,Aj,Xi,D

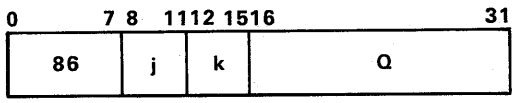


These instructions transfer a field of bytes between Xk and CM. The field length equals S plus one. For lengths less than eight, the load instruction right-justifies and zero-extends the bytes loaded into Xk.

The beginning (leftmost) CM address of the byte string is the sum of D (displacement) plus XiR (index), plus the BN field of Aj. For indexing, these instructions interpret the X0 contents as zeros.

Load Bytes, Relative

86jkQ	Load Bytes, to Xk from (P displaced by Q), length per j	LByTP,j

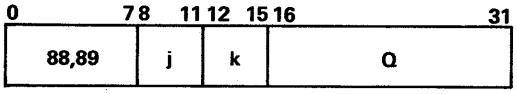


This instruction transfers a field of bytes from CM to Xk. The CM byte field length is the value of the rightmost 3 bits of j plus 1. For lengths less than 8, the byte(s) loaded into Xk are right-justified and zero-extended on the left. The starting (leftmost) CM byte address is the sum of Q (sign-extended to 32 bits) plus the BN field from P.

For this instruction, the CP considers the read operation for the byte field an instruction fetch rather than a data read, and therefore tests the fetch for execute access validity. Refer to Access Protection in section 2 of this volume.

Load/Store Bit

88jkQ	Load Bit, to Xk from (Aj displaced by Q and bit-indexed by XOR)	LBIT Xk,Aj,Q,XO
89jkQ	Store Bit, from Xk at (Aj displaced by Q and bit-indexed by XOR)	SBIT Xk,Aj,Q,XO



These instructions transfer a single bit between XkR bit 63 and a bit position in CM. The load bit instruction also clears Xk bits 0 through 62.

The instructions first generate the CM address of the byte containing the bit loaded or stored as follows:

1. Form byte index by right-shifting XOR 3 bit positions, end-off, and sign-extend to 32 bits.

2. Form the sum of this 32-bit byte index plus Q (sign-extended to 32 bits) plus BN field from Aj.

These instructions then use the original XOR bits 61 through 63 to select the bit position within the addressed byte. Binary values 0 through 7 for these 3 bits select the corresponding bit position (0 through 7) within the byte.

The store bit (89) instruction executes as follows: the byte containing the bit to be stored is read, modified in the appropriate bit position, and written. No other accesses from any port to the addressed byte are permitted between these read and write accesses. Clearing a synchronization lock with this instruction requires preserialization which can be achieved as follows: a test and set bit (14) instruction (which postserializes) issues immediately before the store bit instruction. This postserialization effectively preserializes the lock clearing.

For the store bit instruction, operand access validation consists of write access validation only.

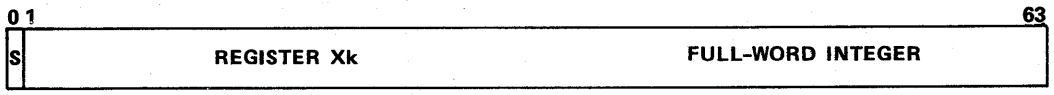
CP INTEGER ARITHMETIC INSTRUCTIONS

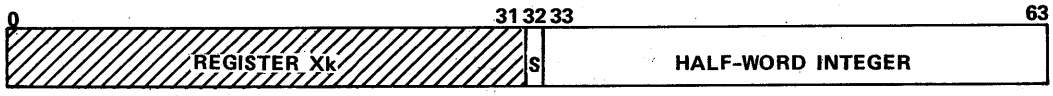
The instructions in this subgroup (table II-1-2) perform integer arithmetic on signed two's complement words or half words in Xk or XkR. The sign bit is bit 0 for full-word integers, or bit 32 for half-word integers.

Table II-1-2. CP Integer Arithmetic Instructions

Opcode	Format	Instruction	Mnemonic
10	jk	Integer sum, immediate	INCX
11	jk	Integer difference	SUBX
20	jk	Half-word integer sum	ADDR
21	jk	Half-word integer difference	SUBR
22	jk	Half-word integer product	MULR
23	jk	Half-word integer quotient	DIVR
24	jk	Integer sum	ADDX
25	jk	Integer difference	SUBX
26	jk	Integer product	MULX
27	jk	Integer quotient	DIVX
28	jk	Half-word integer sum, immediate	INCR
29	jk	Half-word integer difference, immediate	DECR
2C	jk	Half-word integer compare	CMPR
2D	jk	Integer compare	CMPX
8A	jkQ	Half-word integer sum, signed immediate	ADDRQ
8B	jkQ	Integer sum, signed immediate	ADDXQ
8C	jkQ	Half-word integer product, signed immediate	MULRQ
B2	jkQ	Integer product, signed immediate	MULXQ

The integer format is as follows:



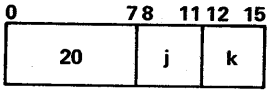


The half-word integer instructions do not alter X register bits 0 through 31.

The arithmetic overflow interrupt condition applies to all integer arithmetic instructions. Individual instruction descriptions list additional interrupt conditions where applicable. (Refer to CP Interrupts in section 2 of this volume for descriptions of these conditions).

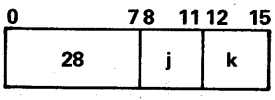
Half-Word Integer Sum

20jk Half-Word Integer Sum,
XkR replaced by XkR plus XjR ADDR Xk,Xj



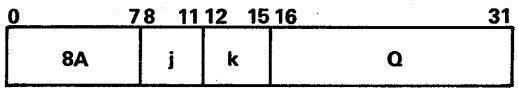
This instruction forms XkR plus XjR and transfers the 32-bit sum to XkR.

28jk Half-Word Integer Sum Immediate,
XkR replaced by XkR plus j INCR Xk,j



This instruction forms XkR plus j zero-extended to 32 bits, and transfers the 32-bit sum to XkR.

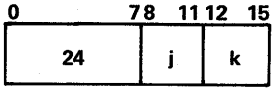
8AjkQ Half-Word Integer Sum, Signed Immediate,
XkR replaced by XjR plus Q ADDRQ Xk,Xj,Q



This instruction forms XjR plus Q sign-extended to 32-bits, and transfers the 32-bit sum to XkR.

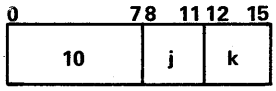
Integer Sum

24jk Integer Sum, Xk replaced by Xk plus Xj ADDX Xk,Xj



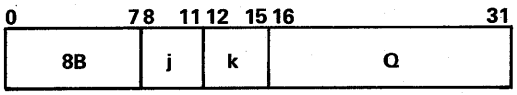
This instruction forms Xk plus Xj and transfers the 64-bit sum to Xk.

10jk Integer Sum Immediate,
Xk replaced by Xk plus j INCX Xk,j



This instruction forms Xk plus j zero-extended on the left to 64 bits, and transfers the 64-bit sum to Xk.

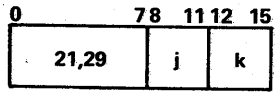
8BjkQ Integer Sum Signed Immediate,
Xk replaced by Xk minus Xj ADDXQ Xk,Xj,Q



This instruction forms Xk plus Q sign-extended to 64 bits, and transfers the 64-bit sum to Xk.

Half-Word Integer Difference

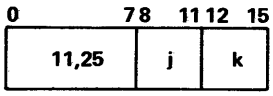
21jk	Half-Word Integer Difference, XkR replaced by XkR minus XjR	SUBR	Xk,Xj
29jk	Half-Word Integer Difference Immediate, XkR replaced by XkR minus j	DECR	Xk,j



These instructions subtract the 32-bit subtrahend in XjR, or in the j field zero-extended to 32 bits on the left, from the 32-bit minuend in XkR, and transfer the 32-bit difference to XkR. The instructions treat each half-word as a signed two's complement integer.

Integer Difference

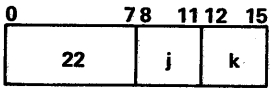
25jk	Integer Difference, Xk replaced by Xk minus Xj	DECX	Xk,j
11jk	Integer Difference Immediate, Xk replaced by Xk minus j	SUBX	Xk,Xj



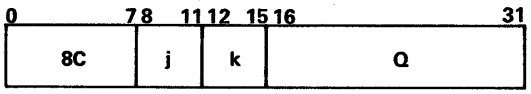
These instructions subtract Xj, or j zero-extended to 64 bits on the left, from Xk, and transfer the 64-bit difference to Xk. The instructions treat each word as a signed two's complement integer.

Half-Word Integer Product

22jk	Half-Word Integer Product, XkR replaced by XkR times XjR	MULR	Xk,Xj



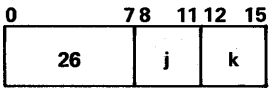
8CjkQ Half-Word Integer Product Signed Immediate,
XkR replaced by XjR times Q MULRQ Xk,Xi,Q



The first instruction multiplies XkR by XjR. The second instruction multiplies the Q field (sign-extended to 32 bits) by XjR. The multiplication forms an algebraically-signed, 64-bit intermediate product. The rightmost 32 bits of this intermediate product transfer to XkR as the final product.

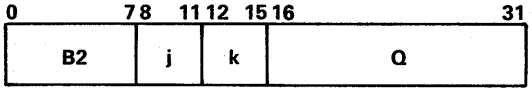
Integer Product

26jk Integer Product, Xk replaced by Xk times Xj MULX Xk,Xj



This instruction multiplies the signed two's complement integers in Xk and Xj to form an algebraically-signed, 128-bit intermediate product. The rightmost 64 bits of this intermediate product transfer to Xk as the final product.

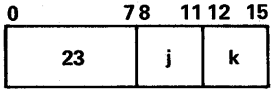
B2jkQ Integer Product Signed Immediate,
Xk replaced by Xj times Q MULXQ Xk,Xj,Q



The first instruction multiplies the signed two's complement integer from Xk by Xj. The second instruction multiplies the Q field sign-extended to 64 bits by Xj. An algebraically-signed, 128-bit intermediate product forms. The rightmost 64 bits of this intermediate product transfer to Xk as the final product.

Half-Word Integer Quotient

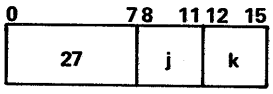
23jk Half-Word Integer Quotient,
XkR replaced by XkR divided by XjR DIVR Xk,Xj



This instruction divides XkR by XjR, and transfers the algebraically-signed, 32-bit quotient to XkR. A divide fault (UCR bit 55) interrupt condition will occur if XjR is equal to zero.

Integer Quotient

27jk Integer Quotient, Xk replaced by Xk divided by Xj — DIVX Xk,Xj

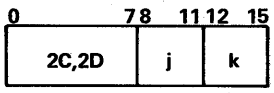


This instruction divides the word in Xk by the 64-bit word in Xj, and transfers the result, consisting of an algebraically-signed, 64-bit quotient to Xk. A divide fault (UCR bit 55) interrupt condition will occur if Xj is equal to zero.

Half-Word Integer/Integer Compare

2Cjk Half-Word Integer Compare, XjR to XkR, result to X1R — CMPR Xi,Xj,Xk

2Djk Integer Compare, Xj to Xk, result to X1R — CMPX Xi,Xj,Xk



These instructions algebraically compare the two's complement binary integer in XjR or Xj to the signed two's complement binary integer in XkR or Xk, respectively. X0 consists of all zeros. Based on the comparison result, XIR sets as follows:

Condition	Action Taken
Xj = Xk	Clear X1R bits 32 through 63.
Xj $\geq$ Xk	Clear X1R bits 32 and 34 through 63, set bit 33.
Xj $\leq$ Xk	Clear X1R bits 34 through 63, set bits 32 and 33.

CP BRANCH INSTRUCTIONS

This subgroup (table II-1-3) consists of both conditional and unconditional branch instructions. Each conditional branch instruction compares the contents of two general registers to determine whether a normal or a branch exit is taken.

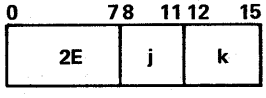
Table II-1-3. CP Branch Instructions

Opcode	Format	Instruction	Mnemonic
2E	jk	Branch relative	BREL
2F	jk	Branch intersegment	BRDIR
90	jkQ	Branch on half-word equal	BRREQ
91	jkQ	Branch on half-word not equal	BRRNE
92	jkQ	Branch on half-word greater than	BRRGT
93	jkQ	Branch on half-word greater than or equal	BRRGE
94	jkQ	Branch on equal	BRXEQ
95	jkQ	Branch on not equal	BRXNE
96	jkQ	Branch on greater than	BRXGT
97	jkQ	Branch on greater than or equal	BRXGE
9C	jkQ	Branch and increment	BRINC
9D	jkQ	Branch on segments unequal	BRSEG

The debug interrupt condition applies to all branch instructions.
Individual instruction descriptions list additional interrupt conditions where applicable.
Refer to CP Interrupts in section 2 of this volume for a description of these conditions.

Branch Relative

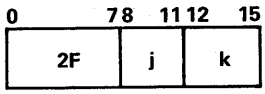
2Ejk Branch to P Indexed by 2*XkR BRREL Xk



This instruction causes an unconditional branch to the CM address formed by adding 2 times XkR to the BN field in P.

Branch Intersegment

2Fjk Branch to Aj indexed by 2*XkR BRDIR Aj,Xk



This instruction causes an unconditional branch by modifying the key (KEY), segment number (SEG), and byte number (BN) fields of the PVA in P, as follows:

1. The key in P copies to the lock of the branched-to segment. The branch is permitted if the key and lock are equal, if the key is a master key, or if the lock is zero or equals no lock.

2. The 12-bit SEG field in Aj (bits 20 through 31) transfers to P bits 20 through 31.

3. A value 2 times XkR adds to the rightmost 32 bits from Aj. X0 consists of all zeros. This sum transfers to bit positions 32 through 63 of P.

This instruction can cause the following exception conditions:

- Address specification error.
- Invalid segment/ring number zero.
- Access violation.

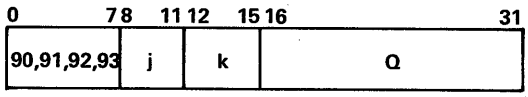
Branch on Half-Word

90jkQ Branch to P Displaced by 2*Q, if XjR equal to XkR BRREQ Xj,Xk,Q

91jkQ Branch to P Displaced by 2*Q, if XjR not equal to XkR BRRNE Xj,Xk,Q

92jkQ Branch to P Displaced by 2*Q, if XjR greater than XkR BRRGT Xj,Xk,Q

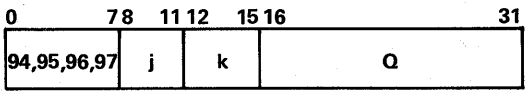
93jkQ Branch to P Displaced by 2*Q, if XjR greater than or equal to XkR BRRGE Xj,Xk,Q



These instructions algebraically compare XjR with XkR, treating each as a signed two's complement binary integer. X0 consists of all zeros. If the comparison between XjR and XkR does not satisfy the branch condition specified, the instruction takes a normal exit by adding 4 to the BN field in P to generate the next instruction address. If the Xj right (XjR) and Xk right (XkR) comparison satisfies the branch condition, the instruction takes a branch exit by adding 2 times Q to the BN field in P to form the next instruction address.

Branch

94jkQ	Branch to P Displaced by 2*Q, if Xj equal to Xk	BRXEQ Xj,Xk,Q
95jkQ	Branch to P Displaced by 2*Q, if Xj not equal to Xk	BRXNE Xj,Xk,Q
96jkQ	Branch to P Displaced by 2*Q, if Xj greater than Xk	BRXGT Xj,Xk,Q
97jkQ	Branch to P Displaced by 2*Q, if Xj greater than or equal to Xk	BRXGE Xj,Xk,Q

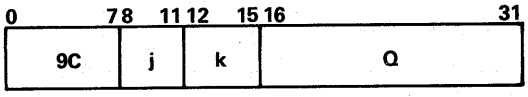


These instructions algebraically compare the Xj word with the Xk word, treating each as a signed two's complement binary integer. X0 consists of all zeros.

If the comparison between Xj and Xk does not satisfy the branch condition specified, the instruction takes a normal exit by adding 4 to the BN field in P to generate the next instruction address. If the comparison satisfies the branch condition, the instruction causes a branch exit by adding 2 times Q to the BN field in P to form the the next instruction address.

Branch and Increment

9CjkQ	Branch to P Displaced by 2*Q and Increment Xk, if Xj greater than Xk	BRINC Xj,Xk,Q



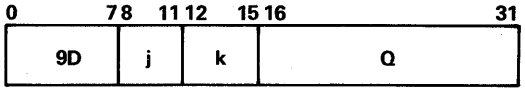
This instruction algebraically compares the Xj word with the Xk word, treating each as a signed two's complement binary integer. For Xj only, the instruction interprets X0 as all zeros.

The comparison results are as follows:

Condition	Action Taken
Xj $\leq$ Xk	Normal exit. Add 4 to BN field in P to form next instruction address.
Xj > Xk	Branch exit. Add 2 times Q to BN field in P to form next instruction address, and increase word in Xk by 1. Overflow is ignored.

Branch on Segments Unequal

9DjkQ Branch to P Displaced by 2*Q, if segments unequal, else compare byte numbers, result to X1R. BRSEG Xi,Aj,Ak,Q



This instruction performs a bit-for-bit comparison between the SEG fields in Aj and Ak (bits 20 through 31). If the SEG fields are unequal, the instruction takes a branch exit by adding 2 times Q to the BN field in P to form the next instruction address.

If the SEG fields are equal, the instruction takes a normal exit by adding 4 to the BN field in P to form the next instruction address. The instruction also algebraically compares Aj bits 32 through 63 with Ak bits 32 through 63, treating each 32-bit quantity as a signed two's complement binary integer, and stores the comparison result in X1R, as follows:

Result	Action Taken
AJ = Ak	Clear X1R.
Aj > Ak	Clear X1R bits 32 and 34 through 63, set bit 33.
Aj < Ak	Clear X1R bits 34 through 63, set bits 32 and 33.

CP COPY INSTRUCTIONS

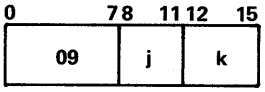
The copy instructions (table II-1-4) transfer information between registers.

Table II-1-4. CP Copy Instructions

Opcode	Format	Instruction	Mnemonic
09	jk	Copy address, A to A	CPYAA
0A	jk	Copy address, X to A	CPYXA
0B	jk	Copy address, A to X	CPYAX
0C	jk	Copy half word	CPYRR
0D	jk	Copy full word	CPYXX

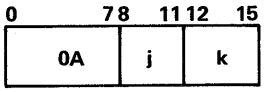
Copy Address

09jk Copy, Ak replaced by Aj CPYAA Ak,Aj



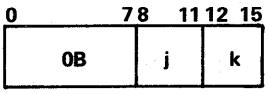
This instruction transfers the 48 bits in Aj to Ak.

0Ajk Copy, Ak replaced by Xj CPYXA Ak,Xj



This instruction unconditionally transfers Xj bits 20 through 63 to the corresponding bit positions of Ak. The instruction also compares Xj bits 16 through 19 with P bits 16 through 19, and transfers the larger field to Ak bits 16 through 19.

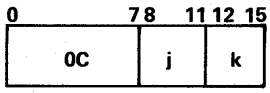
0Bjk Copy, Xk replaced by Aj CPYAX Xk,Aj



This instruction transfers the 48 bits in Aj to Xk bit positions 16 through 63, and clears Xk bits 0 through 15.

Copy Half Word

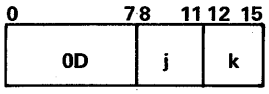
0Cjk Copy, XkR replaced by XjR CPYRR Xk,Xj



This instruction transfers the half-word in XjR to XkR. The XkL content does not change.

Copy Full Word

0Djk Copy, Xk replaced by Xj CPYXX Xk,Xj



This instruction transfers the Xj word to Xk.

CP ADDRESS ARITHMETIC INSTRUCTIONS

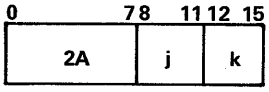
Address arithmetic instructions (table II-1-5) perform address arithmetic in two's complement ignoring overflow.

Table II-1-5. CP Address Arithmetic Instructions

Opcode	Format	Instruction	Mnemonic
2A	jk	Address increment, indexed	ADDAX
8E	jkQ	Address increment, signed immediate	ADDAQ
8F	jkQ	Address relative	ADDPXQ
A7	jkiD	Address increment, modulo	ADDAD

Address Increment, Indexed

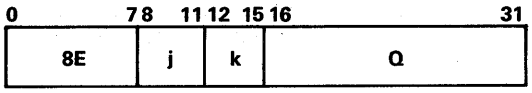
2Ajk Address Ak Replaced by Ak plus XjR ADDAX Ak,Xj



This instruction adds XjR and Ak bits 32 through 63, and returns the sum to Ak bits 32 through 63.

Address Increment, Signed Immediate

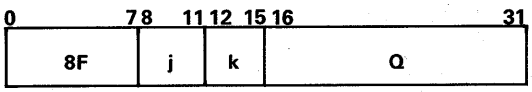
8EjkQ Address Ak replaced by Aj plus Q. ADDAQ Ak,Aj,Q



This instruction transfers Aj bits 16 through 31 to the corresponding Ak bit positions. Also, the instruction adds Q (sign-extended to 32 bits) and Aj bits 32 through 63, and transfers the sum to Ak bits 32 through 63. Overflow is ignored.

Address Relative

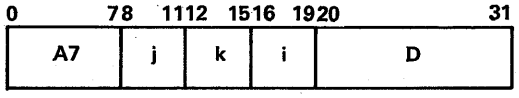
8FjkQ Address Ak replaced by P plus 2*XjR plus 2*Q ADDPXQ Ak,Xj,Q



This instruction transfers P bits 16 through 31 to the corresponding 16 bit positions of Ak. The instruction also adds Q (sign-extended to 32 bits) to the rightmost 32 bits of P, adds this 32-bit sum to 2 times the XkR value, and transfers the final sum to Ak bits 32 through 63. Overflow is ignored. The instruction interprets X0 as all zeros.

Address Increment, Modulo

A7jkiD Address Ak Replaced by Ai plus D per j ADDAD Ak,Ai,D,j



This instruction transfers Ai bits 16 through 31 to the corresponding bit positions of Ak. The instruction also adds D (zero-extended to 32 bits on left) to Ai bits 32 through 63, and transfers bits 32 through 60 of this sum to Ak bits 32 through 60. The instruction performs a logical product (AND) between bits 61 through 63 of this 32-bit sum and the rightmost 3 bits of j, and transfers the 3-bit logical product to Ak bits 61 through 63. Overflow is ignored.

The following is an example of the logical product (AND) operation:

First operand	0011
Second operand	0101
Result (AND)	0001

CP ENTER INSTRUCTIONS

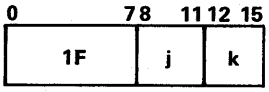
The instructions in this subgroup (table II-1-6) enter immediate operands (consisting of logical quantities or signed two's complement binary integers) into the X registers.

Table II-1-6. CP Enter Instructions

Opcode	Format	Instruction	Mnemonic
1F	jk	Enter zeros	ENTZ
1F	jk	Enter ones	ENTO
1F	jk	Enter signs	ENTZ
39	jk	Enter X1, immediate logical	ENTX
3D	jk	Enter, immediate positive	ENTP
3E	jk	Enter, immediate negative	ENTN
3F	jk	Enter X0, immediate logical	ENTL
87	jkQ	Enter X1, signed immediate	ENTC
8D	jkQ	Enter, signed immediate	ENTE
B3	jkQ	Enter X0, signed immediate	ENTA

Enter Zeros/Ones/Signs

1Fjk	Enter XkL with Zeros	ENTZ	Xk
1Fjk	Enter XkL with Ones	ENTO	Xk
1Fjk	Enter XkL with Signs	ENTS	Xk

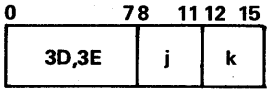


This instruction translates the rightmost 2 bits of j as follows:

j Field	Action Taken
xx00	Clear XkL bits 0 through 31.
xx01	Set XkL bits 0 through 31.
xx10	Copy bit 32 (sign) of XkR to bits 0 through 31 of XkL.

Enter Immediate Positive/Negative

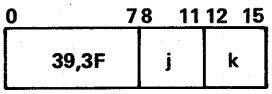
3Djk	Enter Xk with plus j	ENTP Xk,j
3Ejk	Enter Xk with minus j	ENTN Xk,j



These instructions zero-extend j to 64 bits and transfer this result, or the two's complement of this result, to Xk.

Enter X1/X0, Immediate Logical

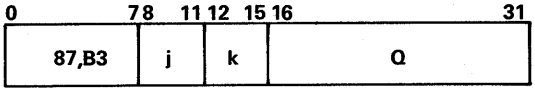
39jk	Enter X1 with Logical jk	ENTX X1,jk
3Fjk	Enter X0 with Logical jk	ENTL X0,jk



These instructions transfer k to bits 60 through 63, and j to bits 56 through 59 of X0 or X1. The instructions clear bits 0 through 55.

Enter X1/X0, Signed Immediate

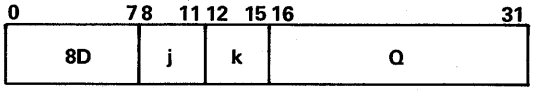
87jkQ	Enter X1 with Sign-Extended jkQ	ENTC X1,jkQ
B3jkQ	Enter X0 with Sign-Extended jkQ	ENTA X0,jkQ



These instructions expand the 24-bit concatenation of j, k, and Q to 64 bits (right-justified) by extending the leftmost bit of j through bits 0 through 39. The 64-bit result transfers to X0 or X1.

Enter, Signed Immediate

8DjkQ Enter Xk with Sign-Extended Q ENTE Xk,Q



This instruction sign-extends Q to 64 bits, and transfers this value to Xk.

CP SHIFT INSTRUCTIONS

The shift instructions (table II-1-7) shift the Xj 64 bits through the number of bit positions determined from a computed shift count. The result transfers to Xk.

Table II-1-7. CP Shift Instructions

Opcode	Format	Instruction	Mnemonic
A8	jkiD	Shift word, circular	SHFC
A9	jkiD	Shift word, end-off	SHFX
AA	jkiD	Shift half word, end-off	SHFR

The computed shift count is the sum of the D field rightmost 8 bits plus XiR bits 56 through 63. An overflow from this 8-bit sum is ignored. The instructions interpret X0 as all zeros. The computed shift count is determined by the following:

1. The leftmost bit of the 8-bit computed shift count determines the shift direction:

   Positive sign: Left shift.

   Negative sign: Right shift.

2. The actual shift count is the two's complement of the rightmost 5 or 6 bits of the computed shift count for 32- and 64-bit operands, respectively. Thus, half words can be shifted from 0 to 31 places left or from 1 to 32 places right. Similarly, full words can be shifted from 0 to 63 places left or from 1 to 64 places right. The shifts are as follows:

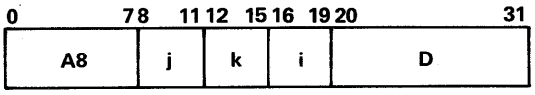
Shift Count	32-Bit Shifts	Shift Count	64-Bit Shifts
0111 1111	Left shift 31 (repeating)	0111 1111	Left shift 63
0010 0000	Left shift 0	0100 0000	Left shift 0
0001 1111	Left shift 31	0011 1111	Left shift 63
0000 0000	Left shift 0	0000 0000	Left shift 0
1111 1111	Right shift 1	1111 1111	Right shift 1
1110 0000	Right shift 32	1100 0000	Right shift 64
1101 1111	Right shift 1 (repeating)	1011 1111	Right shift 1
1000 0000	Right shift 32	1000 0000	Right shift 64

3. If the computed shift count results in an actual shift count of zero, Xj transfers to Xk without shifting.

Shift Word, Circular

A8jkiD Shift Circular, Xk replaced by Xj, direction and count per XiR plus D

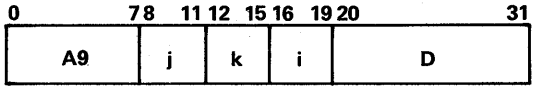
SHFC Xk,Xj,Xi,D



This instruction shifts the Xj word by the computed shift count, and transfers the result to Xk. The shift is circular. Bits that shift out one end of the word transfer into bit positions which become unoccupied at the opposite end of the word.

Shift End-Off, Word/Half-Word

A9jkiD	Shift End-Off, Xk replaced by Xj, direction and count per XiR plus D	SHFX	Xk,Xj,Xi,D
AAjkiD	Shift End-Off, XkR replaced by XjR, direction and count per XiR plus D	SHFR	Xk,Xj,Xi,D



These instructions shift the Xj word, or the XjR half-word, and transfer the result to Xk or XkR. The computed shift count determines the direction and number of bit positions to be shifted. In a right shift, the instruction right-shifts the word end-off on the right, and sign-extends on the left. In a left shift, the instruction left-shifts the word end-off on the left and inserts zeros on the right.

CP LOGICAL INSTRUCTIONS

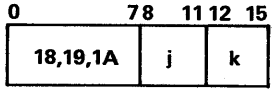
The instructions in this subgroup (table II-1-8) perform logical (Boolean) operations on 64-bit operands in the X registers:

Table II-1-8. CP Logical Instructions

Opcode	Format	Instruction	Mnemonic
18	jk	Logical sum	IORX
19	jk	Logical difference	XORX
1A	jk	Logical product	ANDX
1B	jk	Logical complement	NOTX
1C	jk	Logical inhibit	INHX

Logical Sum/Difference/Product

18jk	Logical Sum, Xk replaced by Xk OR Xj	IORX Xk,Xj
19jk	Logical Difference, Xk replaced by Xk XOR Xj	XORX Xk,Xj
1Ajk	Logical Product, Xk replaced by Xk AND Xj	ANDX Xk,Xj

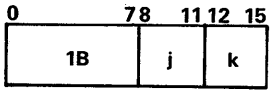


These instructions form the logical sum, difference, or product between the words in Xj and Xk, and return the 64-bit Boolean result to Xk. Examples of these operations are as follows:

	Logical Sum (OR)	Logical Difference (XOR)	Logical Product (AND)
First operand	0011	0011	0011
Second operand	0101	0101	0101
Result	0111	0110	0001

Logical Complement

1Bjk	Logical Complement, Xk replaced by Xj NOT	NOTX Xk,Xj

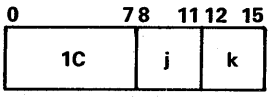


This instruction transfers the one's complement of the Xj word to Xk. The one's complement of a number results from subtracting the original number, bit for bit, from a number consisting of all ones. For example:

	One's Complement
Ones	1111
Xj	0110
Xk	1001

Logical Inhibit

1Cjk Logical Inhibit,
Xk replaced by Xk AND Xj NOT INHX Xk,Xj



This instruction forms the logical product (AND) between the one's complement of the Xj number and the Xk number, and returns the result to Xk. For example:

	Logical Inhibit
Xj	0011
NOT Xj	1100
Xk	0101
Xk AND Xj NOT	0100

CP REGISTER BIT STRING INSTRUCTIONS

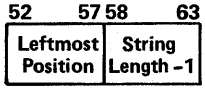
The instructions in this subgroup (table II-1-9) address a contiguous string (field) of bits within a register, beginning and ending at any bit position. The instructions interpret X0 as all zeros.

Table II-1-9. CP Register Bit String Instructions

Opcode	Format	Instruction	Mnemonic
AC	jkiD	Isolate bit mask	ISOM
AD	jkiD	Isolate bit string	ISOB
AE	jkiD	Insert bit string	INSB

Bit String Descriptor

The beginning bit position and bit string length are specified by a bit string descriptor. The descriptor is the rightmost 12 bits of the sum of D (sign-extended) plus XiR, and has the following format:

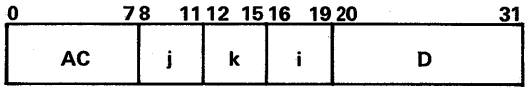


XiR Bits	Description
52-57	Beginning (leftmost) bit position.
58-63	Length, one less than the number of bits in the string.

The instruction specification error interrupt condition applies to all register bit string instructions. Refer to CP Interrupts in section 2 of this volume for a description of this condition.

Isolate Bit Mask

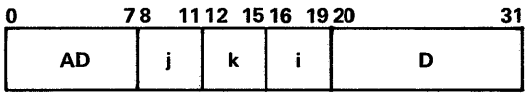
ACjkiD Isolate Bit Mask, into Xk per XiR plus D ISOM Xk,Xi,D



This instruction generates a bit mask consisting of a contiguous field of ones, and places this field into Xk. The bit string descriptor defines the leftmost bit position and the length of the Xk field. All Xk bits outside the specified string clear.

Isolate

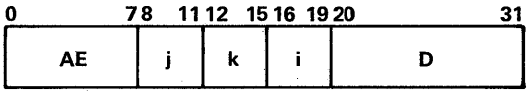
ADjkiD Isolate, into Xk from Xj per XiR plus D ISOB Xk,Xj,Xi,D



This instruction clears Xk and transfers a field of contiguous data from Xj into Xk, right-justified. The bit string descriptor defines the leftmost bit position and Xj field length.

Insert

AEjkiD Insert, into Xk from Xj per XiR plus D INSB Xk,Xj,Xi,D

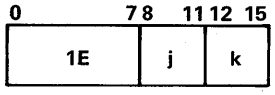


This instruction transfers a field of contiguous bits from Xj to Xk. The field is obtained from the Xj rightmost bit positions, with the length specified by the bit string descriptor. The field inserts into Xk with the leftmost bit position and the field length also specified by the bit string descriptor. All Xk bit positions outside the specified field remain unchanged.

CP MARK TO BOOLEAN INSTRUCTION

The following instruction tests X1R bits 32 through 33 for values specified by the j field.

1Ejk Mark to Boolean, set Xk per j and X1R MARK Xk,Xi,j



This instruction tests XjR bits 32 through 33 for a bit combination by comparing j and X1R bits 32 through 33 for an equal condition (EQ) as shown in table II-1-10. If X1R bits 32 through 33 equal a value specified by j, the instruction clears Xk bits 01 through 63 and sets Xk bit 0. The instruction clears Xk if no equality occurs.

From left to right, the 4 bits of j are individual pointers associated with the 4 possible values of X1R bits 32 and 33 (00, 01, 10, 11). When set, the first bit in the j field tests bits 32 and 33 for a value of 00, the second bit for 01, the third bit for 10, and the fourth bit for 11. For example, if j equals 0101, equality occurs when bits 32 and 33 are either 01 or 11.

Table II-1-10. Compare j Field and X1 Bits 32 and 33

j Field	Register X1R Bits 32, 33			
	00	01	10	11
0000	Unconditional Inequality			
0001				EQ
0010			EQ	
0011			EQ	EQ
0100		EQ		
0101		EQ		EQ
0110		EQ	EQ	
0111		EQ	EQ	EQ
1000	EQ			
1001	EQ			EQ
1010	EQ		EQ	
1011	EQ		EQ	EQ
1100	EQ	EQ		
1101	EQ	EQ		EQ
1110	EQ	EQ	EQ	
1111	Unconditional Equality			

BDP INSTRUCTION DESCRIPTIONS

The business data processing (BDP) instruction group consists of 18 operation codes in 3 subgroups:

- BDP numeric.
- Byte.
- Subscript and immediate data.

Tables II-1-11 through II-1-14 list the instructions within each subgroup. For descriptions of source and destination fields, data descriptors, access types, data formats, and data types, refer to Business Data Processing Programming in section 2 of this volume.

BDP NOMENCLATURE

The BDP instruction descriptions use the following additional terms:

Term	Description
D(Aj)	Source data field addressed by PVA in Aj.
D(Ak)	Other source data field, or destination data field, addressed by PVA in Ak.
D(Ai+D)	Edit mask addressed by PVA in Ai plus displacement D. The edit instruction (ED) uses this term.

BDP NUMERIC INSTRUCTIONS

The instructions in this subgroup (table II-1-11) perform arithmetic, shift, conversion, and comparison operations on byte fields from CM.

Table II-1-11. BDP Numeric Instructions

Opcode	Format	Instruction	Mnemonic
70	jk	Decimal sum	ADDN,Aj,X0
71	jk	Decimal difference	SUBN,Aj,X0
72	jk	Decimal product	MULN,Aj,X0
73	jk	Decimal quotient	DIVN,Aj,X0
74	jk	Decimal compare	CMPN,Aj,X0
75	jk	Numeric move	MOVN,Aj,X0
E4	jkiD	Decimal scale	SCLN,Aj,X0
E5	jkiD	Decimal scale rounded	SCLR,Aj,X0

After completing the required operation, the instructions store the right-justified result in the destination field. These instructions also do the following:

- Zero-fill the high-order destination field positions if the decimal result is shorter than the destination field.
- Truncate the result's leftmost bits if the result exceeds the destination field.
- Treat a decimal numeric value of minus zero as equal to plus zero.
- Do not store minus zero as a result, except when truncation takes place.

An instruction specification error occurs if the length and type fields in the source and destination field data descriptors do not conform to the length and type allowed for a particular instruction. This inhibits instruction execution and initiates the corresponding program interrupt.

The following conditions apply to all BDP numeric instructions:

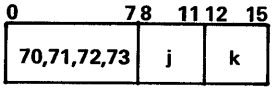
- Instruction specification error.
- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Page table search without find.
- Debug.
- Invalid BDP data.

A destination BDP operand of length zero transforms the instruction into a no-operation. However, when the source field length is nonzero, exception sensing for the source field occurs. This includes testing for arithmetic loss-of-significance and for overflow, but excludes testing for a divide fault.

Individual instruction descriptions list additional interrupt conditions, where applicable. Refer to CP Interrupts in section 2 of this volume for descriptions of these conditions.

Decimal Arithmetic

70jk	(2 descriptors) Decimal Sum, D(Ak) replaced by D(Ak) plus D(Aj)	ADDN,Aj,X0	Ak,Xi
71jk	(2 descriptors) Decimal Difference, D(Ak) replaced by D(Ak) minus D(Aj)	SUBN,Aj,X0	Ak,Xi
72jk	(2 descriptors) Decimal Product, D(Ak) replaced by D(Ak) times D(Aj)	MULN,Aj,X0	Ak,Xi
73jk	(2 descriptors) Decimal Quotient, D(Ak) replaced by D(Ak) divided by D(Aj)	DIVN,Aj,X0	Ak,Xi



These instructions perform arithmetic operations on the initial destination field (an augend, minuend, multiplicand, or dividend) and the source field (an addend, subtrahend, multiplier, or divisor). The decimal result (sum, difference, product, or quotient) transfers to the destination field.

The instructions allow packed and unpacked decimal data types 0 through 6. They do not support unpacked decimal leading sign data types 7 and 8. A numeric move instruction (75) must be used to format operands of these types prior to use in arithmetic operations.

The instruction results are algebraically signed. If the results equal zero with no loss-of-significance, a positive sign is entered. The result translates to the preferred codes of the data type specified by the destination field data descriptor.

These instructions can cause the following exception conditions:

- Arithmetic overflow.
- Divide fault (instruction 73 only, refer to table II-1-12).

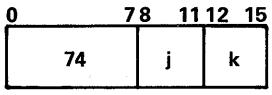
Table II-1-12. BDP Divide Fault

K Field Length	K Value	J Field Length	K Value	Divide Fault
0	*	0	*	No
0	*	Nonzero	0	No
0	*	Nonzero	Nonzero	No
Nonzero	0	0	*	Yes
Nonzero	0	Nonzero	0	Yes
Nonzero	0	Nonzero	Nonzero	No
Nonzero	Nonzero	0	*	Yes
Nonzero	Nonzero	Nonzero	0	Yes
Nonzero	Nonzero	Nonzero	Nonzero	No

*Since field length is zero, the data is disregarded.

Decimal Compare

74jk (2 descriptors) Decimal Compare, D(Aj) to D(Ak), result to X1R — CMPN,Aj,X0 Ak,Xi



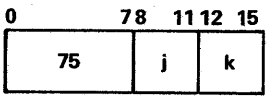
This instruction algebraically compares the decimal contents of the source and destination fields, and depending on the comparison results, transfers a half word to X1R as follows:

Condition	Action Taken
D(Aj) = D(Ak)	Clear X1R.
D(Aj) > D(Ak)	Clear X1R bits 32 and 34 through 63, set bit 33.
D(Aj) < D(Ak)	Clear X1R bits 34 through 63, set bits 32 and 33.

The instruction allows data types 0 through 6. The maximum operand length is a function of the data type. The instruction accommodates unequal field lengths by using decimal zero fill in the leftmost positions of the shorter-length field.

Numeric Move

75jk (2 descriptors) Numeric Move, D(Ak) replaced by D(Aj) after formatting — MOVN,Aj,X0 Ak,Xi



This instruction obtains a number from the source field, validates the number according to the T field from its associated data descriptor, reformats it according to the T field in the destination field data descriptor, and transfers the result to the destination field.

The instruction can convert and format any combination of data types 0 through 8, and 10 or 11. If the conversion is from a decimal data type to a binary data type, the decimal data type determines the maximum length for the source as follows:

Source Field Data Type	Maximum Source Field Length (Bytes)
0 through 3	19
4 through 8	38

The maximum destination field length is eight bytes. The instruction truncates the leftmost bytes if the destination field is not long enough to accommodate the entire binary number, or extends the sign bit on the left if the destination field exceeds the conversion result. When truncation places a negative zero into the destination field, it is not changed to positive zero.

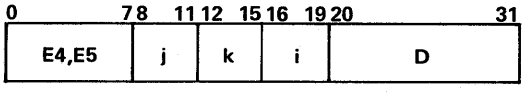
The same length restrictions apply if the source is a binary data type and the destination is a decimal data type, except that if the receiving field exceeds the converted number, the instruction adds leading zeros according to the decimal data type [ASCII character zero (30_{16}) or digit zero (0_{16})].

When both operands are decimal, the destination field fills from right to left. If the field lengths are unequal, the instruction either truncates leading digits or inserts leading zeros according to the destination data type.

This instruction can cause the arithmetic loss-of-significance exception condition.

Decimal Scale

E4jkiD	(2 descriptors) Decimal Scale, D(Ak) replaced by D(Aj), scaled per XiR plus D	SCLN,Aj,X0 Ak,X1,Xi
E5jkiD	(2 descriptors) Decimal Scale Rounded, D(Ak) replaced by rounded D(Aj), scaled per XiR plus D	SCLR,Aj,X0 Ak,X1,Xi



These shift instructions move data from the source field to the destination field, shifting the data under control of a computed shift count. This count is the 8-bit sum of the two's complement 32-bit integer from XiR plus the D-field rightmost 8 bits of the instruction. Any overflow from the 8-bit sum is ignored. The X0 contents interpret as all zeros. The instruction acts as a move instruction if the shift count equals zero.

With positive shift count (bit 56 = 0), the source data left-shifts as determined by bits 57 through 63 of the computed shift count. A negative shift count (bit 56 = 1) causes a shift to the right. In this case, the number of positions is determined by the two's complement of bits 57 through 63 of the computed shift count. A value of 1000 0000 interprets as a right shift of 128 positions.

A positive shift count effectively multiplies the source data by powers of 10; a negative shift count divides the source data by powers of 10. The shifting occurs as the data moves from the source to the destination field. Shifting is end-off with zero-fill as required to accommodate the length and type specified for the destination field. The source field sign moves the destination field unchanged.

The shift counts are interpreted as follows:

Shift Count	Shifts
01111 1111	Left shift 127
↓	↓
0000 0000	Left shift 0
1111 1111	Right shift 1
↓	↓
1000 0001	
1000 0000	Right shift 128

The instruction allows data types 0 through 6 for the source and destination fields.

The decimal scale rounded (E5) instruction rounds upward the absolute value of the right-shift result. This occurs by adding 5 to the last digit shifted end-off, and propagating carries through the decimal result.

These instructions may cause the arithmetic loss-of-significance exception condition.

BDP BYTE INSTRUCTIONS

The instructions in this subgroup (table II-1-13) compare, translate, move, edit, or scan byte fields in CM.

Table II-1-13. BDP Byte Instructions

Opcode	Format	Instruction	Mnemonic
76	jk	Move bytes	MOVB,Aj,X0
77	jk	Byte compare	CMPB,Aj,X0
E9	jkiD	Byte compare, collated	CMPC,Aj,X0
EB	jkiD	Byte translate	TRANB,Aj,X0
ED	jkiD	Edit	EDIT,Aj,X0
F3	jkiD	Byte scan while nonmember	SCNB,X0

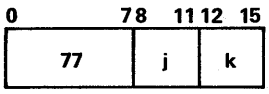
The following conditions apply to all byte instructions:

- Instruction specification error.
- Address specification error.
- Access violation.
- Page table search without find.
- Debug.

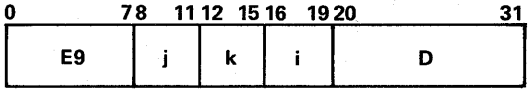
Individual instruction descriptions list additional interrupt conditions where applicable. Refer to CP Interrupts in section 2 of this volume for descriptions of these conditions.

Byte Compare

77jk (2 descriptors) Byte Compare, D(Aj) to D(Ak), result to X1R, index to X0R — CMPB,Aj,X0 Ak,X1



E9jkiD (2 descriptors) Byte Compare Collated, D(Aj) to D(Ak), both translated per (Ai plus D), result to X1R, index to X0R — CMPC,Aj,X0 Ak,X1,Ai,D



These instructions compare the bytes in the source and destination fields, and set X1R according to the result. The comparison proceeds from left to right. When the field lengths are unequal, trailing space characters (20_{16}) are used for the shorter field. The maximum operand length is 256 bytes. Data types are ignored. The comparison continues until the longer field is exhausted or the instructions detect an inequality, as follows:

Compare (77)	The byte comparison ends when the instruction detects an inequality between corresponding bytes in the source and destination field.
Compare Collated (E9)	An inequality detected between corresponding bytes from the source and destination fields results in the translation of both bytes, using a translation table in CM. If the translated bytes are unequal, the comparison stops with the results shown in the following list. If the translated bytes are equal, the comparison continues until the longer field is exhausted, or until the instruction detects another inequality. In the later case, another translation and comparison occurs.

The comparison results are indicated in X1R as follows:

Condition	Action Taken
D(Aj) = D(Ak)	X1R cleared.
D(Aj) > D(Ak)	Clear X1R bits 32 and 34 through 63, set bit 33.
D(Aj) < D(Ak)	Clear X1R bits 34 through 63, set bits 32 and 33.

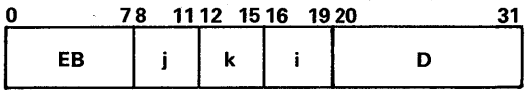
An unequal comparison places the sequence number of the byte causing the inequality into X0R. The instruction adds each field's leftmost byte address to the sequence number in X0R to determine the byte addresses within the source and destination fields causing the unequal comparison. Register X0R does not change if inequalities do not exist.

The user determines the translation table contents used by the compare collated instruction, and preloads the table into CM. The translation table contains 256 bytes. Its starting address forms by adding the BN field in Ai to the zero-extended D field from the instruction. Each translated byte adds as a positive offset to the translation table starting address, forming the address of the translated byte read from CM.

Byte Translate

EBjkiD (2 descriptors) Byte Translate, D(Ak) replaced by D(Aj), translated per (Ai plus D)

TRANB,Aj,X0 Ak,Xi,D



This instruction translates each source field byte according to a user-generated translation table in CM, and transfers the results to the destination field. The source and destination field lengths are limited to 256 bytes. Data types are ignored.

The translation proceeds from left to right. The instruction uses each source field byte as a positive offset which it adds to the translation table address to locate the translated byte. Translated bytes transfer to the destination field. The translation terminates after the destination field length has been exhausted.

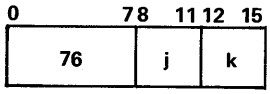
If the source field exceeds the destination field, the instruction truncates the rightmost bytes of the source field. When the source field is shorter than the destination field, the instruction fills the destination field rightmost byte positions with translated space characters.

The user determines the translation table contents and preloads the table into CM. This table contains 256 bytes; its starting address forms by adding the BN field in Ai to the zero-extended D field from the instruction.

Move Bytes

76jk (2 descriptors) Move Bytes, D(Ak) replaced by D(Aj)

MOVB,Aj,X0 Ak,Xi

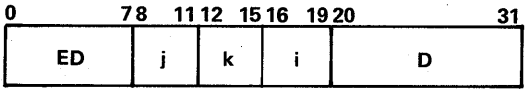


This instruction moves bytes from the source field to the destination field. The move operation is from left to right; data types are ignored. Maximum field lengths are 256 bytes. Unequal field lengths result in truncating trailing characters from the source field or inserting trailing space characters into the destination field.

Edit

EDjkiD (1 descriptor) Edit, D(Ak) replaced by D(Aj) edited per D(Ai+D)

EDIT,Aj,X0 Ak,Xi,Ai,D



Under control of a CM byte field called an edit mask, this instruction edits digits or characters from the source field and transfers the result to the destination field. It can perform the following editing functions:

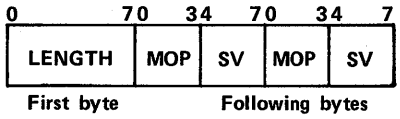
- Move source field digits/characters to destination field.
- Move characters from the edit mask to destination field.
- Specify and insert a string of 0 through 15 characters (symbol) into the destination field.
- Specify an 8-byte special character table (SCT) and insert any character from this table.
- Insert suppression characters and floating signs to the left of the first significant digit.
- Perform insertion of signs, suppression characters, blanks, symbols, or SCT characters based on whether the source field is positive or negative.
- Spread suppression character through the destination field.
- Write suppression characters if destination field is zero.

The source data descriptor type fields are restricted to data types 0 through 9. The instruction ignores the destination data descriptor type fields.

Edit Mask

The edit mask consists of a length-indication byte followed by up to 254 micro-operation bytes. The length is a binary number indicating the number of bytes in the edit mask (including the length-indication byte). If the length-indicating byte is either zero or one, the associated edit instruction results in a no-operation. After the length indicator, the mask contains a string of one-byte microinstructions.

The edit mask address is the sum of the BN field from Ai plus the zero-extended D field from the instruction. The edit mask format is as follows:



Field	Description
LENGTH	Binary number indicating the total number of bytes in the edit mask (0 to 255_{10}).
MOP	Microoperator specifying the editing function.
SV	Binary specification value from 0 through 15. Meaning varies according to the associated MOP.

Edit Operation

The edit operation uses the tables and toggles described in the following paragraphs. Edit control proceeds from left to right on the mask, one character at a time. The instruction performs the editing function specified by the MOP and the SV.

Indexing through the source field is by bytes unless its data type is packed-numeric. Packed-numeric data is indexed by half-bytes. Indexing through the destination field is by bytes.

MOP Description Nomenclature

The MOP descriptions use the following additional terms:

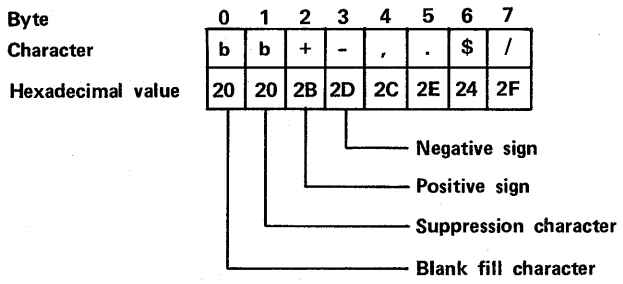
Term	Description
ES	End suppression toggle.
SCT	Special characters table.
SV	Specification value (Refer to Edit Mask, preceding).
SM	Symbol.
SN	Negative sign toggle.
ZF	Zero field.

End Suppression Toggle

The end suppression (ES) toggle controls zero suppression. Hardware sets the ES toggle false at the start of edit. The ES toggle sets true when zero suppression ends, when the first nonzero leading digit is encountered, or by a MOP.

Special Characters Table

The eight-byte special characters table (SCT) is stored in hardware. Entries are written by the micro operation code D. For proper editing, the SCT must be as follows:



Symbol

The symbol (SM) is a string of 0 through 15 characters that the edit instruction creates and inserts into the destination field, under edit mask control. Once the symbol has been inserted, the instruction must recreate it before reinserting it. The symbol has a length of zero when an edit operation begins. The system uses the symbol for the floating-sign and floating-currency editing features, and for sign-sensitive and significance-sensitive character string insertion.

Negative Sign Toggle

The negative sign toggle (SN) provides the source field sign. At start of edit, hardware sets the SN toggle false if the source field is an alphanumeric, an unsigned numeric, or a positive numeric. The SN toggle is initialized true only for a negative numeric source field.

Zero Field Toggle

The zero field (ZF) toggle depicts a zero or nonzero source field. It is initialized true, and sets false after encountering the first nonzero character.

Skipping of Signs

The edit instruction (under edit mask control) automatically skips signs when reading numeric data types. The signs interpret numerically when reading combined signed data types, also under edit mask control.

Microoperation 0

This MOP translates source field characters to ASCII and moves these to the destination field as follows. The translation performs as described in the Edit Function NUMERIC in this section.

1. Set ES true if SV is not equal to zero.
2. Translate SV digits from the source field to the equivalent ASCII characters and copy these into the destination field.

Microoperation 1

This MOP moves type 9 characters as follows:

1. Set ES true if SV is not equal to zero.
2. Move SV characters from the source field to the destination field. The source field must be type 9 or an invalid BDP data condition occurs.

Microoperation 2,3

These MOPs are no-operations.

Microoperation 4

This MOP moves the next edit mask SV bytes to the destination field.

Microoperation 5

This MOP sets the symbol to a single character from SCT, respresenting the source data field sign as follows:

- Negative source data field.
  Copy SCT byte 3 to destination field.
- Positive source data field.
  Copy SCT byte SV into symbol field. The SV rightmost 3 bits provide an index into the SCT.

Microoperation 6

This MOP moves the next edit mask SV bytes to the symbol.

Microoperation 7

This MOP conditionally translates source field SV digits to their equivalent ASCII characters and copies them to the destination field. The translation performs as described in the Edit Function NUMERIC in this section.

- ES false and zero source field digit.
  Copy SCT byte one to destination field.
- ES false and nonzero source field digit.
  Set ES true and copy symbol to destination field followed by the translated digit.
- ES true.
  Copy translated digit to destination field.

Microoperation 8

This MOP conditionally copies the symbol to the destination field as follows:

- ES true.
  No operation.
- ES false.
  Copy symbol to destination field and set ES true.

Microoperation 9

This MOP conditionally copies the symbol or SCT character to the destination field as follows:

- SV > 7
  Copy symbol to destination field.
- SV $\leq$ 7
  Copy SCT byte SV into destination field. The SV rightmost 3 bits provide an index into SCT.

Microoperation A

This MOP conditionally copies the symbol or SCT character to the destination field as follows:

- SV > 7 and source field positive.
  Copy symbol to destination field.

- SV > 7 and source field negative.
  Copy SCT byte 0 to destination field, once for each symbol character.

- SV $\leq$ 7 and source field positive.
  Copy SCT byte SV into destination field. The SV rightmost 3 bits provide an index into SCT.

- SV $\leq$ 7 and source field negative.
  Copy SCT byte 0 into destination field.

Microoperation B

This MOP is identical to MOP A, but with the action caused by a reversal of the source field sign.

Microoperation C

This MOP conditionally copies the symbol or SCT character to the destination field as follows:

- SV > 7 and ES true.
  Copy symbol to destination field.

- SV > 7 and ES false.
  Copy SCT byte 1 character to destination field, once for each symbol character.

- SV $\leq$ 7 and ES true.
  Copy SCT byte SV into destination field. The SV rightmost 3 bits provide an index into SCT.

- SV $\leq$ 7 and ES false.
  Copy SCT byte 1 into destination field.

Microoperation D

This MOP copies the next edit mask character into the SCT byte determined by using the SV rightmost 3 bits as an index into the SCT.

Microoperation E

This MOP copies SCT byte 1 into the destination field, SV times.

Microoperation F

This MOP conditionally copies SCT character into the destination field as follows:

- No-operation when SV = 0.
- ZF false and nonzero source field: terminate the edit instruction.
- ZF true and zero source field: reset to start of destination field and copy SCT byte 1 into destination field SV times.

Edit Function NUMERIC

Micro-operations 0 and 7 translate and move a source digit into the destination field as follows:

- Each source digit is checked. Invalid decimal digits cause an Invalid BDP Data condition. A program interrupt occurs when enabled.
- When the source field is packed-numeric, appropriate ASCII zone bits are supplied for the destination character.
- A nonzero digit causes the ZF toggle to be set false.

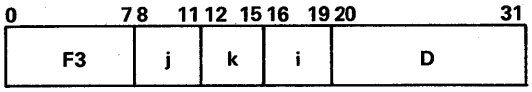
Termination of the Edit Instruction

The edit instruction terminates when the edit mask is exhausted, or when a MOP 15 is read and the zero field (ZF) toggle is false. The CP detects no exception conditions for either condition, even though the instruction may not have exhausted the source or destination fields. If the instruction terminates with the destination field not full, the remaining portion of the destination field is not altered. If the source field is not exhausted when the instruction terminates, the source field is checked for invalid BDP data, and the sign is examined.

The edit instruction may cause the invalid BDP data exception condition.

Byte Scan While Nonmember

F3jkiD (1 descriptor) Byte Scan While Nonmember, D(Ak) for presence bit in (Ai+D), character to X1R, index to X0R — SCNB,Aj,X0 Ak,Xi



This instruction detects possible unwanted characters in a character string by inspecting a 256-bit table in CM. The starting byte address of the table forms by adding the BN field from Ai to the zero-extended D field from the instruction.

The scan proceeds from left to right, one character at a time. The data type is ignored. The binary value of each character addresses a bit in the table. The scan terminates if this bit is a 1 or if the source field has been exhausted.

If the scan terminates because the addressed bit is set, the following occurs:

- The binary value of the sequence number (index) pointing to the byte causing scan termination is placed right-justified into X0R.
- The binary value of the character causing scan termination is placed right-justified into X1R.

If the scan terminates from exhaustion of characters in the byte string, X0R contains the original byte string length, X1R bit 32 sets, and bits 33 through 63 clear.

This instruction can also perform the Byte Scan While Member function. In this case, the table specifying the nonallowed byte string characters is logically complemented before the instruction executes.

BDP SUBSCRIPT AND IMMEDIATE DATA INSTRUCTIONS

The instructions in this subgroup are listed in table II-1-14.

Table II-1-14. BDP Subscript and Immediate Data Instructions

Opcode	Format	Instruction	Mnemonic
F4	jkiD	Calculate subscript and add	CALDF,Aj,X0
F9	jkiD	Move immediate data	MOVI,Xi,D
FA	jkiD	Compare immediate data	CMPI,Xi,D
FB	jkiD	Add immediate data	ADDI,Xi,D

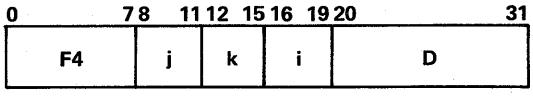
The following conditions apply to all subscript and immediate data instructions.

- Instruction specification error.
- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Page table search without find.
- Debug.
- Invalid BDP data.

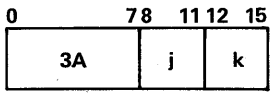
Individual instruction descriptions list additional interrupt conditions where applicable. Refer to CP Interrupts in section 2 of this volume.

Calculate Subscript and Add

F4jkiD (1 descriptor) Calculate Subscript and Add, D(Aj) checked and modified per (Ai plus D), result added to XkR CALDF,Aj,X0 Ak,Xi,Ai,D



This instruction uses a subscript range table (SRT) contained in CM. The SRT contains one or more 64-bit entries with each entry divided into three binary integer values as follows:



Field	Description
SIZE	Sixteen bits, unsigned. Specifies number of elements in one dimension of an array (table).
MIN	Sixteen bits, signed. Specifies minimum allowable value of source field.
MAX	Thirty-two bits, signed. Specifies maximum allowable value of source field.

This instruction forms the PVA of the subscript range table entry using 1) RN and SEG from Ai, and 2) the byte number (BN) generated by adding the BN field from Ai to the instruction D field (expanded to 32 bits using zeros in the high-order bit positions). A signed, 32-bit two's complement binary integer is obtained from the CM source field at location D(Aj). The instruction uses binary source field data unchanged and converts decimal data to its binary equivalent.

The occurrence number is the difference between the binary value of the source field's rightmost 32 bits and the MIN value (sign-extended to 32 bits). The occurrence number is a signed, 32-bit two's complement integer.

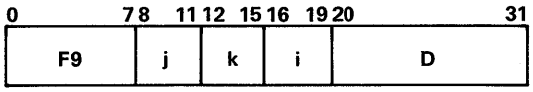
D(Aj) - MIN = OCCURRENCE NUMBER

To calculate the subscript, the instruction multiplies the OCCURRENCE NUMBER by SIZE, and adds the product to the index value in the destination register XkR. The CP does not detect overflow during any arithmetic operation associated with this instruction.

The source field is restricted to data types 0 through 6, 10, and 11, with the maximum field lengths determined by the source field data type.

Move Immediate Data

F9jkiD (1 descriptor) Move Immediate Data, D(Ak) replaced by XiR plus D per j MOVI,Xi,D Ak,Xi,j



The immediate data byte is the two's complement sum of XiR bits 56 through 63, plus the rightmost 8 bits of the instruction D field. Overflow is ignored on this summation. X0 consists of all zeros.

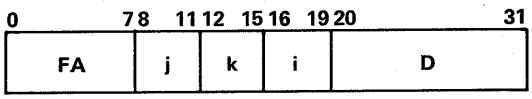
This instruction moves the immediate data to the destination field after format conversion specified by the destination field data type and the j-field suboperation code. The conversion is encoded in the least significant 2 bits (bits 10, 11) of the instruction's j field as follows:

j Field Bits 10,11	Operation
00	The positive, unsigned numeric value (type 10) in the immediate data byte moves right-justified to the destination field. The destination field is restricted to data types 10 or 11.
01	The decimal numeric (type 4) immediate data byte moves right-justified to the destination field after reformatting (if necessary). A positive sign is supplied as required. The destination field is restricted to decimal data types 0 through 6.
10	The ASCII character in the immediate data byte repeats left-to-right in the destination field. Destination data type is ignored.
11	The ASCII character in the immediate data byte moves, left-justified, into the destination field; the remainder of the field fills with space characters. The destination data type is ignored.

The slack digit of destination field types 1 and 3 is unchanged by this instruction. The instruction may cause the arithmetic loss-of-significance exception condition.

Compare Immediate Data

FAjkiD (1 descriptor) Compare Immediate Data, XiR plus D to D(Ak) per j, result to XiR CMPI,Xi,D Ak,Xi,j



The immediate data byte is the two's complement sum of XiR bits 56 through 63, plus the rightmost 8 bits of the instruction D field. Overflow is ignored on this summation. X0 consists of all zeros.

This instruction performs a format conversion on the immediate data byte as specified by destination field data type and the j field suboperation code. The instruction then compares the reformatted immediate data byte to the contents of D(Ak). The instruction j field encodes the operation as follows:

j Field Bits 10,11	Operation
00	The positive, unsigned numeric value (type 10) in the immediate data byte compares to the contents of D(Ak). The destination field is restricted to data types 10 or 11. If field D(Ak) exceeds one byte, the immediate data byte zero-fills in its high-order positions.
01	The decimal numeric (type 4) immediate data byte compares to the contents of D(Ak) after reformatting (if necessary) to match the data type of field D(Ak). A positive sign is supplied as required. The D(Ak) field is restricted to decimal data types 0 through 6. If D(Ak) exceeds one byte, the immediate data byte zero-fills in its high-order positions.
10	The ASCII character in the immediate data byte compares left-to-right to the D(Ak) field. Then (DAk) field data type is ignored.
11	The ASCII character in the immediate data byte compares to the leftmost byte in field D(Ak). If the comparison is equal and field D(Ak) exceeds one byte, a space character compares left-to-right with each successive byte remaining in the D(Ak) field. The D(Ak) field data type is ignored.

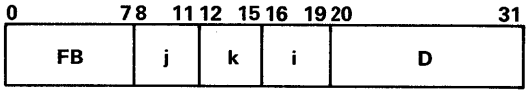
A half-word transfers to X1R to indicate the comparison result as follows:

Results of Compare	Register X1R
Source = Destination	Clear X1R.
Source > Destination	Clear bits 32 and 34 through 63, set bit 33.
Source < Destination	Clear bits 34 through 63, set bits 32 and 33.

Add Immediate Data

FBjkiD (1 descriptor) Add Immediate Data, D(Ak) replaced by D(Ak) plus XiR plus D per j

ADDI,Xl,D Ak,Xi,j



The add immediate instruction converts the source field immediate data to match the destination field data type (if required), and adds the immediate data byte to D(Ak). The immediate data byte stores the integer value of the addend. The instruction j field encodes the data type contained in the immediate data byte.

The j field least significant bit (bit 11) decodes as follows:

j Field Bit 11	Data Type Immediate Data Byte
0	Data type = 10. Unsigned (positive) binary integer value.
1	Data type = 4. One ASCII character representing a decimal digit.

If the source field is data type 10, the destination field is restricted to data types 10 or 11.

If source data is type 4, the destination is restricted to types 0 through 6.

This instruction may cause the arithmetic overflow exception condition.

FLOATING-POINT INSTRUCTION DESCRIPTIONS

Refer to Floating-Point Programming in section 2 of this volume for descriptions of floating-point data formats, standard and nonstandard numbers, and normalization. The floating-point (FP) instructions consists of 18 operation codes in 4 subgroups:

- Conversion.
- Arithmetic.
- Branch.
- Compare.

Tables II-1-15 through II-1-17 list the instructions in the first 3 subgroups.

DOUBLE-PRECISION REGISTER DESIGNATORS

The double-precision FP add, subtract, multiply, and divide instructions operate on double-length registers, designated as follows:

XXk or XXj Two successive registers Xk, X(k+1) or Xj, X(j+1) containing a double-precision FP number. Xk or Xj contains the high order (leftmost) part of this number.

FLOATING-POINT CONVERSION INSTRUCTIONS

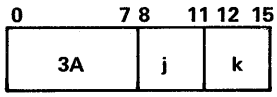
The instructions in this subgroup (table II-1-15) convert 64-bit words between FP and integer formats.

Table II-1-15. Floating-Point Conversion Instructions

Opcode	Format	Instruction	Mnemonic
3A	jk	Convert from integer to FP	CNIF
3B	jk	Convert from FP to integer	CNFI

Convert From Integer to FP

3Ajk Convert, floating-point Xk formed from integer Xj CNIF Xk,Xj

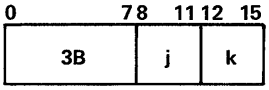


This instruction converts the signed 64-bit two's complement binary integer from Xj to its normalized FP representation, and transfers the 64-bit result to Xk.

During conversion, the instruction truncates the rightmost bits of integers outside the range -2^{48} through $(2^{48})-1$. When Xj is all zeros, it transfers unchanged to Xk.

Convert From FP to Integer

3Bjk Convert, integer Xk formed from floating-point Xj. CNFI Xk,Xj



This instruction converts the 64-bit FP number in Xj to a signed two's complement binary integer and transfers the result to Xk. The fractional part of the binary equivalent truncates. This conversion results in an integer consisting of all zeros if the FP number:

- Is indefinite.
- Has an exponent equal to zero.
- Has a fraction equal to zero.
- Is infinite.

This instruction may cause the arithmetic loss-of-significance, FP indefinite, and FP infinite exception conditions.

FLOATING-POINT ARITHMETIC INSTRUCTIONS

The instructions in this subgroup (table II-1-16) perform arithmetic operations on FP numbers.

Table II-1-16. Floating-Point Arithmetic Instructions

Opcode	Format	Instruction	Mnemonic
30	jk	FP sum	ADDF
31	jk	FP difference	SUBF
32	jk	FP product	MULF
33	jk	FP quotient	DIVF
34	jk	Double-precision FP sum	ADDD
35	jk	Double-precision FP difference	SUBD
36	jk	Double-precision FP product	MULD
37	jk	Double-precision FP quotient	DIVD

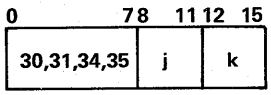
The following conditions apply to all FP arithmetic instructions:

- Exponent overflow.
- Exponent underflow.
- Floating-point loss-of-significance.
- Floating-point indefinite.

Individual instruction descriptions list additional interrupt conditions where applicable. Refer to CP Interrupts in section 2 of this volume.

Floating-Point Sum/Difference

30jk	Floating-Point Sum, Xk replaced by Xk plus Xj	ADDF	Xk,Xj
31jk	Floating-Point Difference, Xk replaced by Xk minus Xj	SUBF	Xk,Xj
34jk	Double-Precision Floating-Point Sum, XXk replaced by XXk plus XXj	ADDD	Xk,Xj
35jk	Double-Precision Floating-Point Difference, XXk replaced by XXk minus XXj	SUBD	Xk,Xj



The following instruction description applies to either single- or double-precision operations. References to Xk or Xj in the description also apply to XXk or XXj for the double-precision instructions.

These instructions algebraically compare the exponents of the two FP operands in Xk and Xj. If the exponents are equal, no adjustment is necessary. If the exponents are unequal, the instruction aligns the coefficients by right-shifting the coefficient with the smaller exponent the number of bit positions designated by the difference between the exponents. The maximum shift is 48 positions for single-precision instructions or 96 positions for double-precision instructions.

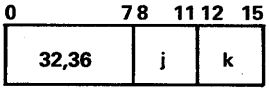
The two aligned coefficients consist of a signed 48-bit single-precision or 96-bit double-precision fraction. The instructions add or subtract the two coefficients as determined by the operation code, using the Xj coefficient as the addend or subtrahend. The algebraic result is a signed coefficient with 48 bits (single-precision) or 96 bits (double-precision), plus an overflow bit. The overflow bit provides the required allowance for true addition (FP sum of coefficients with like signs or FP difference of coefficients with unlike signs).

If coefficient overflow occurs (overflow bit = 1), the instruction right-shifts the coefficient one place, inserts the overflow bit in the high order bit position (bit 16), increases the exponent by one, and places the result in Xk. If the coefficient overflow bit is zero and the coefficient is not all zeros, the instructions normalize the result before placing the result in Xk.

If either or both of the input operands in Xk and Xj consists of an infinite or indefinite FP number, the result transferred to Xk is a nonstandard FP number. Refer to Floating-Point Standard and Nonstandard Numbers in section 2 of this volume.

Floating-Point Product

32jk	Floating-Point Product, Xk replaced by Xk times Xj	MULF	Xk,Xj
36jk	Double Precision Floating-Point Product, XXk replaced by XXk times XXj	MULD	Xk,Xj



The following instruction description applies to either single- or double-precision operations. References to Xk or Xj in the description also apply to XXk and XXj for the double-precision instructions.

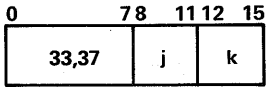
The multiply FP instructions algebraically add the signed exponents for the two FP operands in Xk and Xj, using the result as an intermediate exponent. The instructions multiply the coefficient in Xk by the coefficient in Xj to produce an algebraically-signed product consisting of 96 bits (single-precision) or 192 bits (double precision). If the products high-order bit (bit 16) is a one, the product is already normalized and the high-order 48 bits (single-precision) or 96 bits (double-precision) become an intermediate coefficient.

If the high-order bit is a zero, the instructions left-shift the 96-bit or 192-bit product one bit position, decrease the intermediate exponent by one, and use the high-order 48 bits (single-precision) or 96 bits (double-precision) as the intermediate coefficient. This one-position shift results in a normalized product if both input operands were normalized before executing the multiply instruction. If the intermediate exponent (including the adjustment for normalization) is not equal to an out-of-range value, the intermediate exponent and the intermediate coefficient (with its sign) transfer to Xk to form the final result.

If one or both of the input operands in Xk and Xj consist of an infinite, indefinite, or zero FP number, the result transferred to Xk is a nonstandard FP number. Refer to Floating-Point Standard and Nonstandard Numbers in section 2 of this volume.

Floating-Point Quotient

33jk	Floating-Point Quotient, Xk replaced by Xk divided by Xj	DIVF	Xk,Xj
37jk	Double-Precision Floating-Point Quotient, XXk replaced by XXk divided by XXj	DIVD	Xk,Xj



The following instruction description applies to either single- or double-precision operations. References to Xk or Xj in the description also apply to XXk or XXj for the double-precision instructions.

The divide FP instructions subtract the Xk exponent (divisor) from the Xk exponent (dividend), and use the signed result as an intermediate exponent.

These instructions divide the Xk signed coefficient by the Xj signed coefficient. If the Xj coefficient is unnormalized before instruction execution, and can be divided into the Xk coefficient by a factor exceeding or equal to 2, the CP detects a divide fault.

If the CP does not detect errors, the division results in an algebraically-signed quotient with 48 bits (single-precision) or 96 bits (double-precision), plus an overflow bit. The overflow bit allows for cases in which the divisor can be divided into the dividend by a factor equal to or exceeding one, but less than two. If the overflow bit is a zero, the sign bit and 48- or 96-bit quotient require no further adjustments. If the overflow bit is a one, the instruction right-shifts the quotient one position, end-off, with the overflow bit inserted into the high-order bit position, and the exponent increased by one. The intermediate exponent and intermediate coefficient (with its sign) transfer to Xk to form the final result. When one or both of the input operands in Xk and Xj consist of an infinite, indefinite, or zero FP number, the result transferred to Xk is a nonstandard FP number. (Refer to Floating-Point Standard and Nonstandard Numbers in section 2 of this volume.)

This instruction may cause a divide fault exception condition.

FLOATING-POINT BRANCH INSTRUCTIONS

This subgroup (table II-1-17) consists of five conditional branch instructions. Each instruction compares two FP numbers and performs either a normal or branch exit based on the comparison results.

Table II-1-17. Floating-Point Branch Instructions

Opcode	Format	Instruction	Mnemonic
98	jkQ	FP branch on equal	BRFEQ
99	jkQ	FP branch on not equal	BRFNE
9A	jkQ	FP branch on greater than	BRFGT
9B	jkQ	FP branch on greater than or equal to	BRFGE
9E	jkQ	FP branch on overflow	BROVR
9E	jkQ	FP branch on underflow	BRUND
9E	jkQ	FP branch on indefinite	BRINF
3C	jk	FP compare	CMPF

Normal Exit

The instruction takes a normal exit if the branch condition is not satisfied. The next instruction address forms by adding 4 to the BN field of the current PVA in P.

Branch Exit

The instruction takes a branch exit if the branch condition is satisified. The next instruction address forms by adding 2 times the Q field value (from the branch instruction) to the BN field of the current PVA in P.

Group Interrupt Conditions

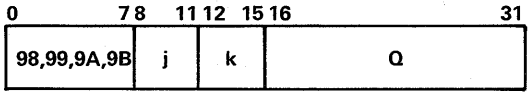
The following interrupt conditions apply to all FP branch instructions.

- Debug.
- Floating-point loss-of-significance.
- Floating-point indefinite.

Refer to CP Interrupts in section 2 of this volume for descriptions of these conditions.

Floating-Point Branch on Comparison

98jkQ	Branch to P Displaced by 2*Q, if floating-point Xj equal to Xk	BRFEQ Xj,Xk,Q
99jkQ	Branch to P Displaced by 2*Q, if floating-point Xj not equal to Xk	BRFNE Xj,Xk,Q
9AjkQ	Branch to P Displaced by 2*Q, if floating-point Xj greater than Xk	BRFGT Xj,Xk,Q
9BjkQ	Branch to P displaced by 2*Q, if floating-point Xj greater than or equal to Xk	BRFGE Xj,Xk,Q



Each compare and branch instruction performs an algebraic comparison between the 64-bit words in Xj and Xk. If the branch conditions are satisfied, the instruction takes a branch exit. If the conditions are not satisified, a normal exit results.

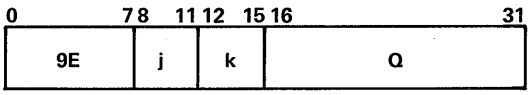
These instructions treat the 64-bit words in Xj and Xk as single-precision FP numbers. If Xj or Xk specifies register X0, these instructions interpret X0 as all zeros.

For the results with the various combinations of comparison input data, refer to Floating-Point Standard and Nonstandard Numbers in section 2 of this volume.

Floating-Point Branch on Condition

9EjkQ Branch to (P) Displaced by 2*Q, if floating-point Xk is exception per j

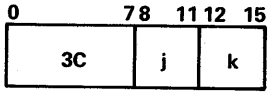
j Field	Xk Tested For	
00	Exponent overflow.	BROVR Xk,Q
01	Exponent underflow.	BRUND Xk,Q
10/11	Exponent indefinite.	BRINF Xk,Q



The instruction takes a branch exit if the exception condition designated by bits 10 and 11 of the instruction j field applies to the 64-bit FP number in Xk. A normal exit occurs if the exception condition designated by j field bits 10 and 11 does not apply to the 64-bit FP number in Xk.

Floating-Point Compare

3Cjk Compare Floating-Point Xj to Xk, result to X1R CMPF Xi,Xj,Xk



This instruction algebraically compares the Xj 64-bit word to the Xk 64-bit word, and indicates the result by setting bits in X1R. The instruction treats the 64-bit words in Xj and Xk as single-precision FP numbers.

If Xj or Xk specifies register X0, the instruction interprets X0 as all zeros. X1R bits are set as follows:

Results of Compare	Register X1R
Xj = Xk	Clear X1R.
Xj > Xk	Clear bits 32 and 34 through 63, set bit 33.
Xj < Xk	Clear bits 34 through 63, set bits 32 and 33.
Indefinite	Clear bits 33 through 63, set bit 32.

If the comparison results are indefinite, the CP records an FP indefinite condition and sets register X1R as shown in the preceding table. If the corresponding user mask bit is set and the trap enabled, the corresponding program interrupt occurs.

VECTOR INSTRUCTION DESCRIPTIONS

The vector instruction group consists of 20 operation codes in 6 subgroups:

- Integer vector arithmetic.
- Integer vector compare.
- Logical vector arithmetic.
- Integer/floating-point vector conversion.
- Floating-point vector arithmetic.
- Special-purpose vector instructions.

Table II-1-18 lists the instructions within each subgroup. For descriptions of vector length, page size, broadcast, interrupts, and overlap, refer to Vector Programming in section 2 of this manual.

VECTOR INSTRUCTION FORMAT

The vector instruction group utilizes the jkiD format (refer to figure II-1-1).

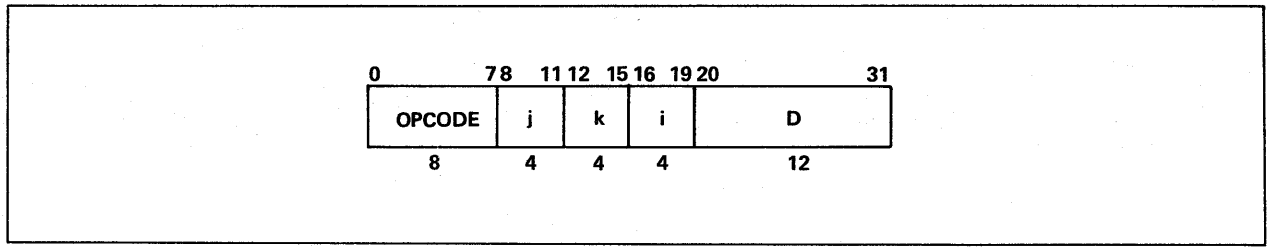


Figure II-1-1. Vector Instruction Format

Field	Description
j	Designates register Aj which contains the starting address of a source vector, VAj.
k	Designates register Ak which contains the starting address of a destination vector, VAk.
i	Designates register Ai which contains the starting address of a second source vector, VAi. May also designate register Xi which contains the interval for gather and scatter instructions.
D	Specifies vector length (number of operations). For further information, refer to Vector Length described under Vector Programming in section 2 of this manual.

Table II-1-18. Vector Instructions

Instruction Name	Opcode	Mnemonic
Integer Vector Sum	44jkiD	ADDXV
Integer Vector Difference	45jkiD	SUBXV
Integer Vector Compare, =	50jkiD	CMPEQV
Integer Vector Compare, <	51jkiD	CMPLEV
Integer Vector Compare, $\geq$	52jkiD	CMPGEV
Integer Vector Compare $\neq$	53jkiD	CMPNEV
Shift Vector Circular	4DjkiD	SHFV
Logical Vector Sum	48jkiD	IORV
Logical Vector Difference	49jkiD	XORV
Logical Vector Product	4AjkiD	ANDV
Convert Vector from Int. to FP	4BjkiD	CNIFV
Convert Vector from FP to Integer	4CjkiD	CNFIV
Floating Point Vector Sum	40jkiD	ADDFV
Floating Point Vector Difference	41jkiD	SUBFV
Floating Point Vector Product	42jkiD	MULFV
Floating Point Vector Quotient	43jkiD	DIVFV
Floating Point Vector Summation	57jkiD	SUMFV
Merge Vector	54jkiD	MRGV
Gather Vector	55jkiD	GTHV
Scatter Vector	56jkiD	SCTV

Integer Vector Arithmetic

The instructions in this subgroup perform arithmetic operations on pairs of integers that compose source vectors from CM. After completing the required operation, the instructions store the results in the destination vector into CM.

44jkiD	Integer Vector Sum, V(Ak) replaced by V(Aj) plus V(Ai)	ADDXV
45jkiD	Integer Vector Difference, V(Ak) replaced by V(Aj) minus V(Ai)	SUBXV

These instructions perform the indicated arithmetic operation on the first element from V(Aj) and V(Ai) and store the result as the first element of V(Ak). This operation repeats for successive elements until the required number of operations has been performed.

Integer Vector Compare

The instructions in this subgroup perform comparisons between pairs of integers that compose source vectors from CM. After completing the required operation, the instructions store the results in a destination vector that returns to CM.

50jkiD	Integer Vector Compare, V(Ak) replaced by V(Aj) equal to V(Ai)	CMPEQV
51jkiD	Integer Vector Compare, V(Ak) replaced by V(Aj) less than V(Ai)	CMPLTV
52jkiD	Integer Vector Compare, V(Ak) replaced by V(Aj) greater than or equal to V(Ai)	CMPGTV
53jkiD	Integer Vector Compare, V(Ak) replaced by V(Aj) not equal V(Ai)	CMPNEV

These instructions perform the indicated integer arithmetic comparison on the first elements from V(Aj) and V(Ai). If the comparison is true, bit 0 is set and bits 1 through 63 are cleared in the first element of V(Ak). If the comparison is false, bits 0 through 63 are cleared in the first element of V(Ak). This operation repeats for successive elements until the required number of operations has been performed. When broadcast of V(Aj) is selected and j=0, the content of X0 interprets as all zeros (refer to Vector Broadcast under Special Purpose Vector Instructions later in this section).

Logical Vector Arithmetic

The instructions in this subgroup perform logical operations between pairs of elements that compose source vectors from CM. After completing the required operation, the instructions store the results in a destination vector that returns to CM.

48jkiD	Logical Vector Sum, V(Ak) replaced by V(Aj) OR V(Ai)	IORV
49jkiD	Logical Vector Difference, V(Ak) replaced by V(Aj) exclusive-OR V(Ai)	XORV
4AjkiD	Logical Vector Product, V(Ak) replaced by V(Aj) AND V(Ai)	ANDV

These instructions perform the indicated logical operation on the first element from V(Aj) and V(Ai) and store the result as the first element of V(Ak). This operation repeats for successive elements until the required number of operations has been performed.

Integer/Floating-Point Vector Conversion

The instructions in this subgroup perform conversions on successive element that compose a source vector from CM. After completing the required operation, the instructions store the results in a destination vector that returns to CM.

4BjkiD	Convert Vector, floating-point V(Ak) formed from integer V(Aj)	CNIFV
4CjkiD	Convert Vector, integer V(Ak) formed from floating-point V(Aj)	CNFIV

These instructions perform the indicated conversion on the first element from V(Aj) and store the result as the first element of V(Ak). This operation repeats for successive elements until the required number of conversions has been performed.

Floating-Point Vector Arithmetic

The instructions in this subgroup perform arithmetic operations on pairs of floating-point operands that compose source vectors from CM. After completing the required operation, the instructions store the results in a destination vector that returns to CM.

40jkiD	Floating-Point Vector Sum, V(Ak) replaced by V(Aj) plus V(Ai)	ADDFV
41jkiD	Floating-Point Vector Difference, V(Ak) replaced by V(Aj) minus V(Ai)	SUBFV
42jkiD	Floating-Point Vector Product, V(Ak) replaced by V(Aj) times V(Ai)	MULFV
43jkiD	Floating-Point Vector Quotient, V(Ak) replaced by V(Aj) divided by V(Ai)	DIVFV

These instructions perform the indicated arithmetic operations on the first element from V(Aj) and V(Ai) and store the result as the first element of V(Ak). This operation repeats for successive elements until the required number of operations has been performed.

Special Purpose Vector Instructions

The instructions in this subgroup perform various manipulative operations on source vectors from CM.

4DjkiD	Shift Vector Circular, V(Ak) replaced by V(Ai), direction and count per V(Aj)	SHFV

This instruction performs a circular shift on the first element from V(Ai) as directed by the first element of V(Aj) and stores the result as the first element of V(Ak). This operation repeats for successive elements until the required number of operations has been performed.

The shift count for each element in V(Ai) is taken from the rightmost 8 bits of the corresponding element of V(Aj). The sign bit in the leftmost position of the 8-bit shift count determines the shift direction. A positive shift count (sign bit = 0) left-shifts the instruction; a negative shift count (sign bit = 1) right-shifts the instruction. Shifts may be from 0 through 63 bits left and from 1 through 64 bits right. (A shift count of 0 causes the associated instruction to transfer the initial element of V(Ai) to the corresponding element in V(Ak) with no shift performed.)

When vector broadcast of V(Aj) is selected and j=0, the X0 contents interpret as all zeros.

54jkiD Merge Vector, V(Ak) partially replaced by V(Aj) per mask V(Ai) — MRGV

This instruction replaces the first element of V(Ak) with the first element of V(Aj) if bit 0 is set in the first element of V(Ai). If bit 0 is clear, the first element of V(Ak) is left unchanged. This operation repeats for successive elements until the required number of operations has been performed.

55jkiD Gather Vector, V(Ak) replaced by gathered V(Aj) with interval Xi — GTHV

This instruction forms the contiguous vector V(Ak) by gathering elements from V(Aj) at interval Xi (refer to figure II-1-2). This instruction obtains the first element from V(Aj) and stores it as the first element of V(Ak). The second element to be stored in V(Ak) is taken from the address formed by adding the rightmost 32 bits of Xi, left-shifted 3 places with zero-fill, to the rightmost 32 bits of the previous address. The n^{th} element of V(Ak) is replaced by V(Ak) whose address is (Aj)+8*(n-1)(Xi). Execution does not alter the Xi contents.

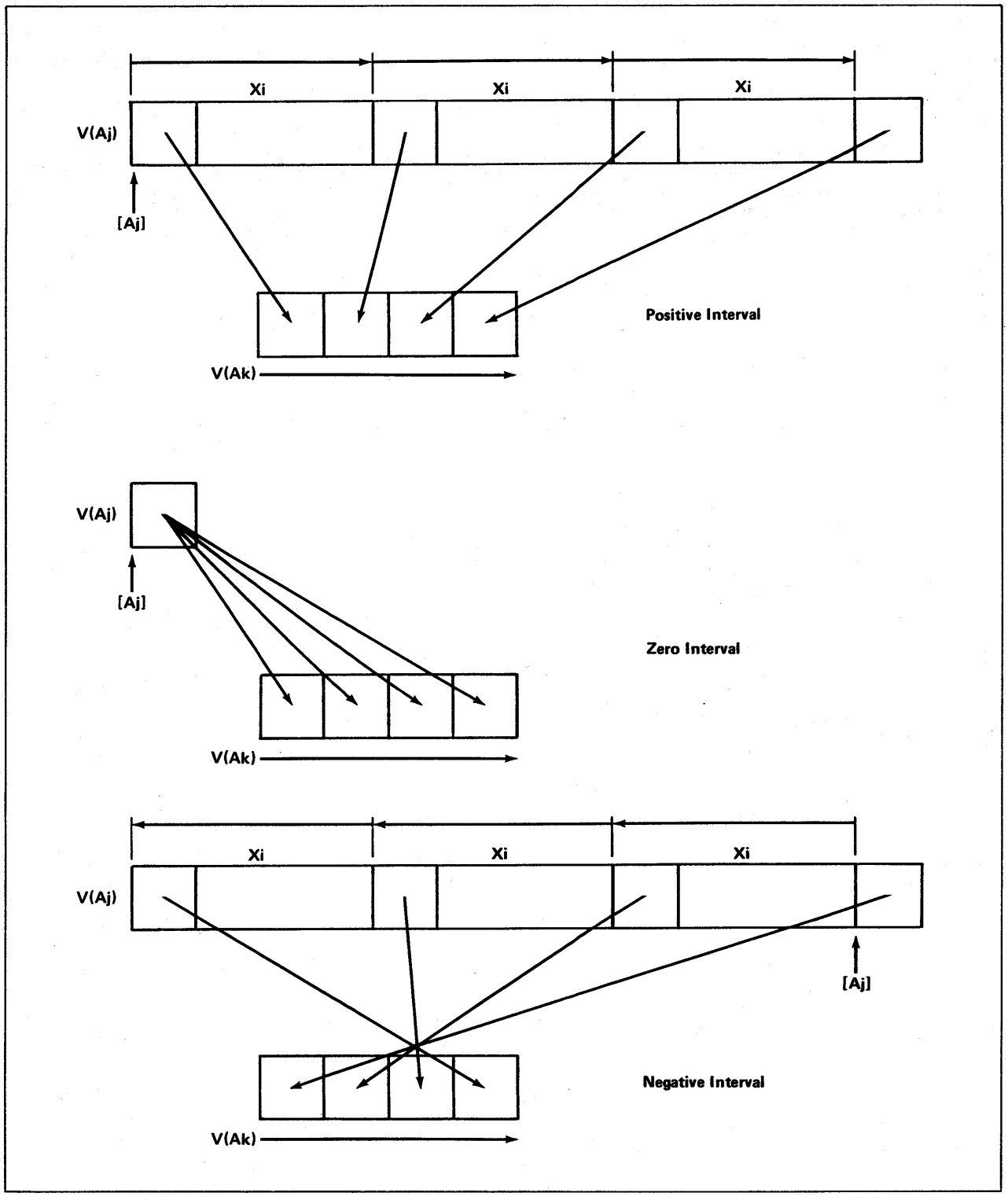


Figure II-1-2. Gather Instruction

56jkiD Scatter Vector, V(Ak) replaced by scattered V(Aj) with interval Xi — SCTV

This instruction scatters the contiguous V(Aj) elements in V(Ak) at interval Xi (refer to figure II-1-3). This instruction obtains the first element from V(Aj) and stores it as the first element of V(Ak). The second contiguous element from V(Aj) is stored into V(Ak) at the address formed by adding the rightmost 32 bits of Xi, left-shifted 3 places with zero-fill, to the rightmost 32 bits of A(k). Successive elements from V(Aj) are stored into the addresses formed by adding the rightmost 32 bits of Xi, left-shifted 3 places with zero-fill, to the rightmost 32 bits of the previous address. The n^{th} element of V(Aj) is stored into V(Ak) at (Aj)+8*(n-1)(Xi). Execution does not alter the Xi contents.

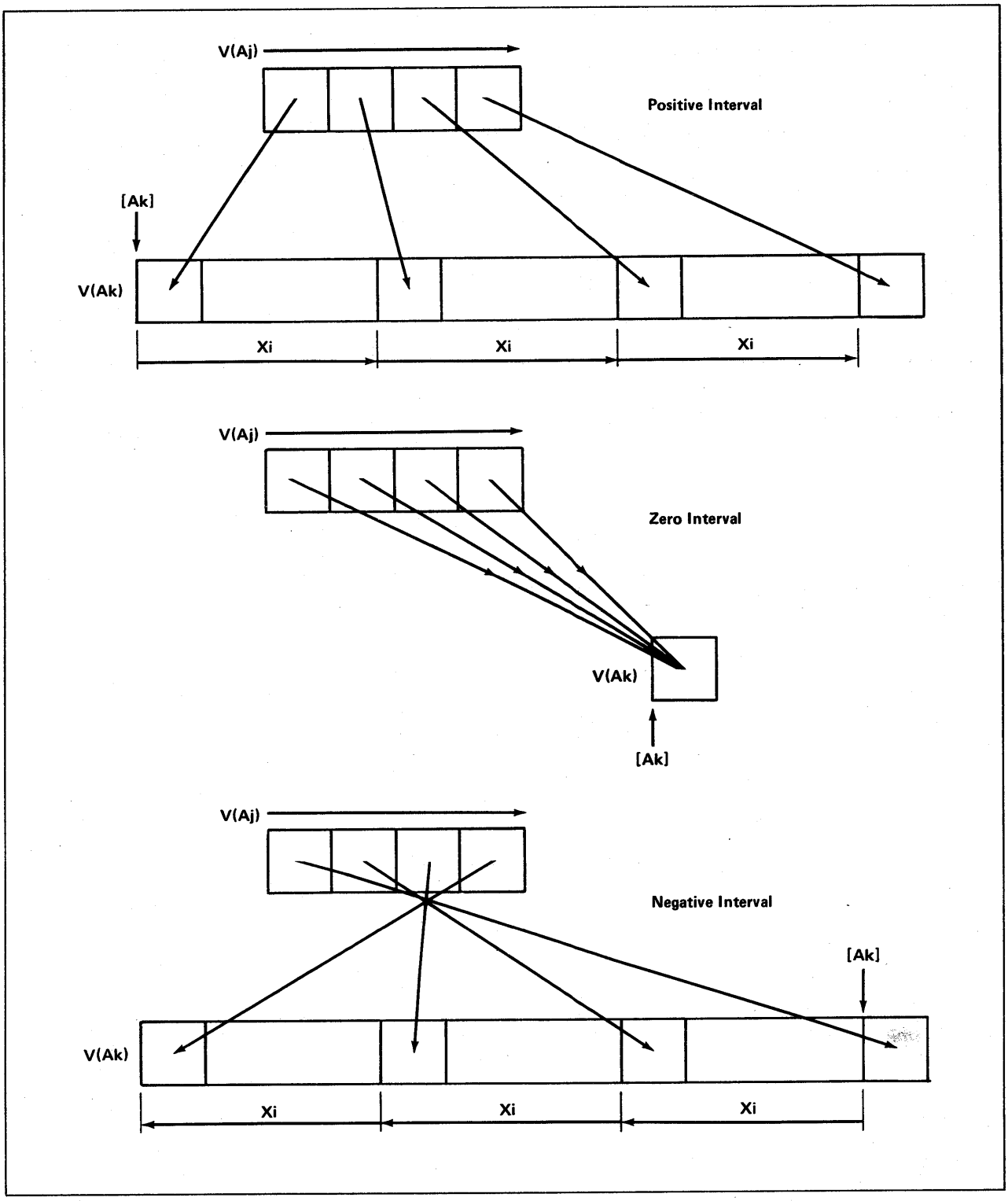


Figure II-1-3. Scatter Instruction

57jkiD Floating-Point Vector Summation, Xk replaced by summation of elements in V(Ai) SUMFV

This instruction adds together all the elements in V(Ai) and stores the sum in Xk. The individual add operations which together form this instruction are single-precision sums and may be performed in any order.

SYSTEM INSTRUCTION DESCRIPTIONS

The system instructions consist of 27 operation codes in 5 classes. The classes are based on the characteristics of the code segment from which the instructions are accessed, or the CP mode in which the instructions may operate. The classes are as follows:

- Nonprivileged.
- Local privileged.
- Global privileged.
- Virtual State.
- Virtual State monitor mode.
- Mixed mode.

Local and global privileged instructions execute only when the XP field of the associated segment descriptor designates the appropriate privilege (with the CP in any mode). Virtual State monitor mode instructions execute only when the CP is in Virtual State monitor mode. Mixed mode instruction parameters within the instruction determine their privilege/mode requirements. Refer to Access Protection in section 2 of this volume for more information.

NONPRIVILEGED SYSTEM INSTRUCTIONS

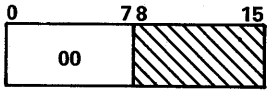
The instructions in this subgroup are listed in table II-1-19. In some cases, a portion of the instruction word is unused, as indicated in the instruction format. Instruction execution is not affected by these unused bits, but it is recommended these bits be zeros.

Table II-1-19. Nonprivileged Instructions

Opcode	Format	Instruction	Mnemonic
00	jk	Program error	HALT
01	jk	Scope loop synchronization	SYNC
02	jk	Exchange	EXCHANGE
04	jk	Return	RETURN
06	jk	Pop	POP
08	jk	Copy free running counter	CPYTX
14	jk	Test and set bit	LBSET
16	jk	Test and set page	TPAGE
B0	jkQ	Call relative	CALLREL
B4	jkQ	Compare swap	CMPXA
B5	jkQ	Call indirect	CALLSEG
BE,BF	jkQ	Reserved opcodes	--
C0-C7	SjkiD	Execute algorithm	EXECUTE,S

Program Error

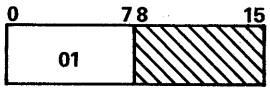
00jk Program Error HALT



This instruction causes an instruction specification error with the corresponding program interrupt or halt.

Scope Loop Sync

01jk Scope Loop Sync SYNC

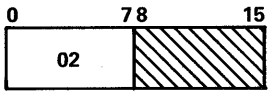


For the model 855, set CP breakpoint register 32 to 101_{16} by performing the CMSE command ER2,32 = 101_{16}. The instruction triggers at TP 44 at location 3A1-C1C.

This instruction is a no-operation within the CP. The instruction generates a pulse to a test point for oscilloscope synchronization.

Exchange

02jk Exchange EXCHANGE



This instruction exchanges the current process registers (formatted as an exchange package) with another set stored in CM, and does the following:

- When executed with CP in Virtual State monitor mode, the processor switches from monitor to job mode.
- When executed in Virtual State job mode, the processor switches from job to monitor mode and the system call bit sets in the monitor condition register (MCR 10).

In either case, the P register stored in the outgoing exchange package points to the next instruction that would have executed if the exchange had not occurred.

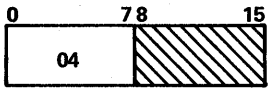
This instruction can cause the following exception conditions:

- Environment specification error.
- System call.

Refer to CP Modes of Operation in section 2 of this volume for further information.

Return

04jk Return RETURN



This instruction requires the following register assignments:

(A0)	Dynamic space pointer (DSP).
(A1)	Current stack frame (CSF) pointer.
(A2)	Previous save area (PSA) pointer.
In exchange package	Top of stack (TOS) pointer for current ring of execution.
In exchange package	TOS pointer for previous ring of execution.

This instruction reestablishes the stack frame and environment of the previous procedure (which must be executing in an equal or less privileged ring as the current procedure). This operation does not load MCR or UCR. The instruction executes as follows:

1. Update the TOS pointer by storing the CSF pointer from A1 into the TOS pointer for the current ring of execution. This has the effect of cancelling the current stack frame.
2. Load the environment from the previous save area (as defined by PSA pointer in A2 and the stack frame descriptor in PSA) as follows:
    - P register (all fields).
    - VMID (CP state switch may take place).
    - CFF and OCF.
    - User mask register.
    - A0 through At (per SFSA descriptor).
    - Xs through Xt (per SFSA descriptor).
3. Set the RN field of each A register loaded from SFSA equal to the largest of the following:
    - A(RN) from SFSA.
    - Initial A2(RN).
    - R1 of SDE for initial A2.
4. If the final P(RN) does not equal the initial P(RN), set any A(RN) not loaded from PSA in step 2 (and less than the final P(RN)) equal to the final P(RN).
5. Update TOS pointer in the exchange package.

6. Clear trap enable delay.

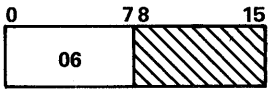
7. If any A(RN) loaded from PSA in step 2 is zero, set MCR 60, with interrupt or halt. When this happens, the instruction execution completes and UTP is unaltered.

This instruction can cause the following exception conditions:

- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Environment specification error.
- Page table search without find.
- Outward call/inward return.
- Critical frame flag.
- Debug.

Pop

06jk Pop POP



This instruction requires the following register assignments:

(A0)	Dynamic space pointer (DSP).
(A1)	Current stack frame (CSF) pointer.
(A2)	Previous save area (PSA) pointer.
In exchange package	Top of stack (TOS) pointer for current ring of execution.

This instruction moves the CSF, PSA, and TOS pointers to eliminate the stack frame without changing the P-counter. This instruction reestablishes the stack frame (but not the environment) of the previous procedure, which must be in the same ring of execution as the current procedure. The stack frame is reestablished as follows:

1. Obtain the stack frame descriptor from the PSA (SFSA for the previous procedure) using the PSA pointer in A2.

2. Update the CSF pointer by loading A1 with word 2 from the PSA. Set A1 ring number equal to P ring number.

3. Update the PSA pointer by loading A2 with word 3 from the PSA. Set A2 ring number equal to the largest of: a) the A2 initial ring number, b) the A2 ring number from PSA, or c) the R1 field of the segment descriptor associated with the PSA.

4. Load the critical frame flag (CFF) and the on-condition flag from the PSA.

5. Update the TOS pointer by storing the CSF pointer from final A1 into the TOS pointer for the current ring of execution. This has the effect of cancelling the current stack frame.

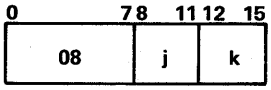
6. If any A1(RN) or A2(RN) loaded from PSA in step 2 is zero, set MCR60, with interrupt or halt. Instruction execution completes and UTP is unaltered.

This instruction may cause the following exception conditions:

- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Environment specification error.
- Page table search without find.
- Inter-ring pop.
- Critical frame flag.
- Debug.

Copy Free Running Counter

08jk Copy Free Running Counter to Xk at XjR CPYTX Xk,Xj

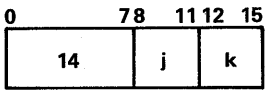


This instruction copies the free running counter in CM into Xk (the free-running counter consists of either 64 bits of counter, or 48 bits of counter which are right-justified with zero-fill in the leftmost 16 bits). XjR bits 32 and 34 through 63 are zeros. XjR bit 33 specifies which processor port the instruction uses to read the counter as follows:

Bit 33	Port Selected
0	Local processor port to CM.
1	External processor port to CM of another system.

Test and Set Bit

14jk Load Bit to XkR from Aj Bit Indexed by XOR and Set Bit in CM — LBSET Xk,Aj,X0



This instruction transfers one bit from CM into XkR bit position 63 and clears Xk bits 0 through 62. The instruction also sets that bit in CM without changing any other bits in CM.

The instruction addresses the CM byte containing the bit by adding bits 32 through 60 of XOR (right-shifted 3 positions, end-off, with sign extension on the left), to bits 32 through 63 of Aj. The instruction uses XOR bits 61 through 63 to locate the bit position within the addressed byte. Values 0 through 7 for these 3 bits select corresponding bits 0 through 7 from the addressed byte.

No other CM accesses (from any port) to the CM byte containing that bit are permitted from the start of the read access until the end of the write access (when the instruction sets the bit in CM).

The system performs a serialization function before and after instruction execution. The CP delays instruction execution until all previous CM accesses by previous instructions complete, and delays execution of the next instruction until all CM accesses from this instruction complete.

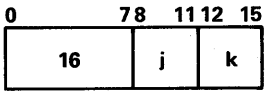
To establish operand access validity, the instruction uses read- and write-type CM accesses. The read access bypasses cache memory. Termination of the write access purges the associated cache entry.

This instruction may cause the following exception conditions:

- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Page table search without find.
- Debug.

Test and Set Page

16jk Test page (Aj) and set XkR — TPAGE Xk,Aj



This instruction tests CM for the presence of the page (corresponding to the PVA in Aj) in the system page table with its valid bit set in the associated page descriptor. If the tested page is in CM, the used bit in the associated page descriptor sets, and the real memory address translated from the PVA from Aj transfers to XkR. If the tested page is not in CM, the instruction sets XkR bit 32 and clears XkR bits 33 through 63.

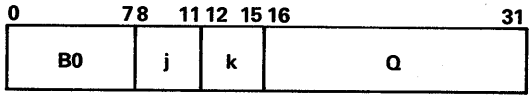
This instruction may cause the following exception conditions:

- Address specification error.
- Invalid segment/ring number zero.

Call Relative

B0jkQ Call to P displaced by 8*Q, binding section pointer per Aj, arguments per Ak

CALLREL Aj,Ak,Q



Register assignments are as follows:

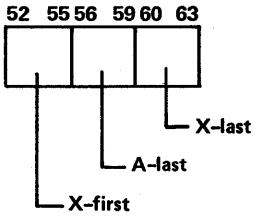
(A0)	Dynamic space pointer (DSP).
(A1)	Current stack frame (CSF) pointer.
(A2)	Previous save area (PSA) pointer.
(A4)	Argument pointer.
In exchange package	Top of stack (TOS) pointer per RN in P.

This instruction saves the current procedure (caller) environment and calls another procedure (callee) within the same segment as the caller. The RN and SEG fields of P remain unaltered.

The caller's environment is saved by storing designated process and processor registers into a stack frame save area (SFSA) generated on top of the current stack frame. The DSP in A0, rounded to the next available full word address, is the PVA of the first word in this SFSA. The instruction saves some CP registers in the SFSA unconditionally. These registers are as follows:

- P register.
- Stack frame descriptor.
- User mask.
- Virtual machine identifier.
- Register A0.

The caller specifies other registers saved. A0 is always the first register saved, and X0R specifies other A and X registers to be saved. X0R has the following format:



X0 Bits	Registers Saved
52-55	First X register.
56-59	Last A register.
60-63	Last X register.

The call instruction does not store any X register if the value of X-last exceeds X-first.

After storing the registers in the SFSA, the instruction executes as follows:

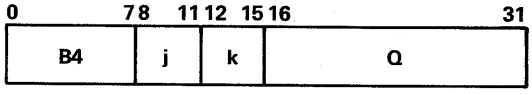
1. Modify dynamic space pointer (DSP) in A0 by adding 8 times the number of SFSA words to the BN in A0.
2. Update the top of stack (TOS) pointer in the exchange package by storing the modified DSP into the exchange package TOS entry corresponding to the current ring of execution, as determined by the RN in P. This creates a new stack frame.
3. Form the target address by adding 8 times Q to the BN in P. Bits 61 through 63 of P are forced to zero.
4. Establish the stack frame of the callee by loading A0, A1, and A2 from the PSA (SFSA of the callee).
5. Copy Aj to A3 and Ak to A4 to reflect parameter changes required to transfer control to the callee.

This instruction may cause the following exception conditions.

- Instruction specification.
- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Page table search without find.
- Debug.

Compare Swap

B4jkQ Compare Xk to (Aj), if locked, branch to P displaced by 2*Q, if unlocked, load/store (Aj), result to X1R CMPXA Xk,Aj,X0,Q



If the leftmost 32 bits of a 64-bit word in CM location Aj are all ones (Aj locked), the instruction takes a branch exit. The target address forms by adding the value 2 times Q (sign-extended) to the BN field of the PVA in P.

If the above condition is absent, the instruction compares the Xk 64-bit word with the word in CM location Aj (64-bit integer compare). If the two words are equal, the instruction stores X0 in location Aj and clears X1R. If the two words are unequal, the instruction loads the word in CM location Aj into Xk and sets X1R as follows (in either case, the instruction takes a normal exit):

Results of Compare	Action Taken
Xk = (Aj)	Store X0 at (Aj), clear X1R.
Xk > (Aj)	Load (Aj) into Xk, clear X1R bits 32 and 34 through 63, set X1R bit 33.
Xk < (Aj)	Load (Aj) into Xk, clear X1R bits 34 through 63, set X1R bits 32 and 33.

Within a given CP, execution of this instruction delays until all previous CM accesses complete. Execution of all subsequent instructions delays until all CM accesses due to this instruction complete. In dual CP systems, if a second CP executes a compare swap instruction while the other CP is processing one, the second CP reads the 64-bit word in location Aj, finds the leftmost 32 bits all ones (locked), and branch-exits. The hardware, however, does not inhibit other instruction codes issued from the other CP (or any PP instructions) from accessing and altering location Aj.

The read access bypasses cache, and the write access purges the associated cache entry.

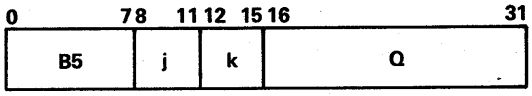
For the debug scan, Aj is both a read and a write address, whereas P+2Q is a branch target address only when the branch occurs.

This instruction may cause the following exception conditions:

- Instruction specification.
- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Page table search without find.
- Debug.

Call Indirect

B5jkQ Call per (Aj Displaced by 8*Q), arguments per Ak CALLSEG Aj,Ak,Q



The instruction uses the following assigned registers:

Register	Description
(A0)	Dynamic space pointer (DSP).
(A1)	Current stack frame pointer.
(A2)	Previous save area pointer.
(A3)	Binding section pointer.
(A4)	Argument pointer.
In exchange package	Top of stack (TOS) pointer for the caller's ring of execution.
In exchange package	TOS pointer for the callee's ring of execution.
In CM	Code base pointer (CBP) addressed by A3+8*Q.

This instruction saves the current procedure (caller) environment and calls another procedure (callee) indirectly. The callee must be executing within the same or in a higher privileged ring as the caller. The indirect target address is listed in the CBP addressed by (Aj displaced by 8 times sign-extended Q). The instruction saves the environment (as specified by XOR) in the SFSA generated on top of the current stack frame. For details, refer to the call relative instruction described in this section.

The instruction executes as follows:

1. Add 8 times Q to the BN field from register Aj to form the PVA of a CBP from a binding section segment (which contains the target PVA).
2. Round DSP upward as follows: Add 7 to A0, then force A0 bits 61 through 63 to zero.
3. Store environment in SFSA, per XOR.
4. Copy P bits 0 through 31 to XOR (callers ID).
5. Modify DSP in A0 by adding 8 times the number of SFSA words to A0 bits 32 through 63.
6. Adjust TOS pointer in the exchange package by storing this modified DSP in the TOS entry for the current ring of execution, as determined by the RN field in P.

7. Load P key with segment descriptor lock for callee.

8. If P ring number is less than callee segment descriptor R2 (inter-ring call), go to step 12.

9. Set P ring number equal to callee segment descriptor R2.

10. Load P SEG and BN fields with code base pointer SEG and BN fields.

11. If CBP VMID = 1 (call is to CYBER 170 State), go to step 17.

12. If internal procedure (code base pointer EPF = 0), go to step 16.

13. Load A3 with new binding section pointer, setting RN equal to the larger of the RN in CBP and the new RN in P register.

14. If trap operation, go to step 17; if call instruction, copy Ak to A4 (pass parameters). When k is 0 through 3, the final contents of A4 is with respect to which A register is copied.

15. Copy (A0) to A2 (DSP from step 2 to PSA pointer).

16. Clear on-condition flag.

17. Load A1 with top of stack pointer from exchange package per final P ring number, and clear the critical frame flag.

18. Set dynamic space pointer in A0 equal to current stack frame pointer in A1.

19. Copy VMID from CBP to the VMID register.

NOTE

The trap interrupt operation unconditionally includes all the above steps except items 10, 11, and 14.

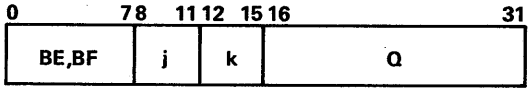
This instruction may cause the following exception conditions:

- Instruction specification.
- Address specification error.
- Invalid segment/ring number zero.
- Access violation.
- Environment specification error.
- Page table search without find.
- Outward call/inward return.
- Debug.

Reserved Operation Codes

BEjkQ Reserved for user

BFjkQ Reserved for user



These two instructions are reserved for the user for software simulation of operations that executive state does not provide through trap interrupts. These operation codes will not be used in future hardware extensions.

When not implemented, these instructions cause the unimplemented instruction exception, with interrupt or halt.

Execute Algorithm

CSjkiD Execute Algorithm per S EXECUTE,s j,k,i,D

This instruction is reserved for future expansion.

When not implemented, this instruction causes the unimplemented instruction exception, with interrupt or halt.

LOCAL PRIVILEGED SYSTEM INSTRUCTIONS

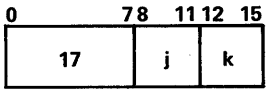
Instructions in this subgroup (table II-1-20) can execute only from segments which (by the associated segment descriptor) have either local or global privilege. If a local privileged instruction is fetched from a segment without either local or global privilege, the CP detects a privileged instruction fault, inhibits execution, and initiates the corresponding program interrupt or halt.

Table II-1-20. Local Privileged Instructions

Opcode	Format	Instruction	Mnemonic
17	jk	Load page table index	LPAGE

Load Page Table Index

17jk Load Page Table Index per Xj to XkR and set X1R — LPAGE Xk,Xj,Xi



This instruction searches the page table in CM for the presence of a page, returns the final search index value to XkR, and sets X1R to indicate the search results. The SVA in Xj defines the required page table entry.

The SVA determines the starting point in the page table search. The search continues until the corresponding page descriptor is found, a continue bit equal to zero is detected, or 32 entries have been searched. The validity bit is ignored.

When the page is found, the page table index associated with that entry transfers to XkR, the number of entries searched transfers to X1R bits 33 through 63 (right-justified with zeros extended), and X1R bit 32 sets to indicate the find.

When a page corresponding to the SVA in Xj is not found, the page table index value of the last entry tested transfers to XkR, the number of entries searched transfers to X1R bits 33 through 63 (right-justified with zeros extended), and X1R bit 32 clears.

If the instruction's k field equals 1, register X1R loads with the result-indication bit and the number of entries searched instead of the index value.

This instruction can cause the following exception conditions:

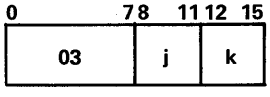
- Privileged instruction fault.
- Address specification error.

GLOBAL PRIVILEGED SYSTEM INSTRUCTION

The processor interrupt instruction can execute only from segments which (by the associated segment descriptor) have global privilege. If this instruction is fetched from a segment without global privilege, the CP detects a privileged instruction fault, inhibits execution, and initiates the corresponding program interrupt or halt.

Processor Interrupt

03jk Processor Interrupt per Xk — INTRUPT Xk,j



This instruction sends an external interrupt to one or more CPs (including the executing CP) through their CM ports. The interrupting CP sends Xk to CM. CM then sends an external interrupt to the processor(s) connected to the ports whose numbers correspond to the bits set in Xk as follows:

Xk Bit	Port Number
60	3
61	2
62	1
63	0

Bits 0 through 59 are not used to send interrupts and are ignored by the CM, but have correct parity. When two ports of the same memory connect to the interrupting CP, the state of Xk bit 33 selects the port the CP uses to send Xk to CM along with the interrupt. (Xk bit 33 thus overrides RMA bit 33 for memory port selection).

State of Bit 33	Memory Port Used
Clear	0
Set	1

The system delays this instruction's execution until all previous CM accesses by the interrupting CP complete. If a CP sends an interrupt to itself, this instruction completes executing before the interrupt is taken.

This instruction can cause the privileged instruction fault exception condition.

MONITOR MODE INSTRUCTIONS

Instructions in this subgroup can execute only with the processor in executive monitor mode. Otherwise, the CP detects an instruction specification error, inhibits instruction execution, and initiates the corresponding program interrupt. Refer to Mixed Mode Instructions in the following description.

MIXED MODE INSTRUCTIONS

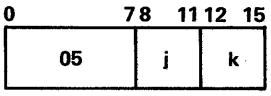
The execution of instructions in this subgroup (table II-1-21) depends on an instruction parameter. The parameter value determines whether the instruction is executable from nonprivileged, local-privileged, or global-privileged segments, or whether the CP must be in Virtual State monitor mode.

Table II-1-21. Mixed Mode Instructions

Opcode	Format	Instruction	Mnemonic
05	jk	Purge buffer	PURGE
0E	jk	Copy from state register	CPYSX
0F	jk	Copy to state register	CPYXS
9F	jk	Branch on condition register	BRCR

Purge Buffer

05jk Purge buffer k of entry per Xj PURGE Xj,k



This instruction invalidates entries in the cache (models 835, 845, 855, and 990), map, or instruction buffer, selectable as follows:

- All entries in cache (models 835, 845, 855, and 990), map or instruction buffer.
- All entries for a given segment in cache (models 835, 845, 855, and 990) or map.
- All entries for a given page in cache (models 835, 845, 855, and 990) or map.
- All entries for a given 512-byte block in cache (models 835, 845, 855, and 990).

Xj contains the required address information as either the system virtual address (SVA) or the process virtual address (PVA). The k value determines the buffer to be purged, the range of entries to be purged, and the addressing type used, as follows:

Value of k	Description
k = 0	Purge all cache entries in a 512-byte block defined by SVA in Xj.
k = 1	Purge all cache entries in ASID defined by SVA in Xj.
k = 2	Purge all cache entries.
k = 3	Purge all cache entries in 512-byte block defined by PVA in Xj.
k = 4 through 7	Purge all cache entries in SEG defined by PVA in Xj.
k = 8	Purge all map entries in page associated with page table entry defined by SVA in Xj. (Page size is determined from page size mask register.)
k = 9	Purge all map entries pertaining to page table entries included in segment defined by SVA in Xj.
k = A	Purge all map entries pertaining to page table entry defined by PVA in Xj. Page size mask register specifies number of bytes in page.
k = B	Purge all map entries pertaining to segment table entry defined by PVA in Xj, and to all page table entries included within that segment.
k = C through F	Purge all map entries, ignore Xj.

If k = 0, 1, 2, or 8 through F, this instruction is a local privileged instruction. It is a nonprivileged instruction for all other k values.

The system performs a serialization function before this instruction begins execution, and again when execution completes. The system delays instruction execution until all previous accesses to CM by this processor complete, and delays the fetch or execution of subsequent instructions until all CM accesses for this instruction complete.

This instruction may cause the following exception conditions:

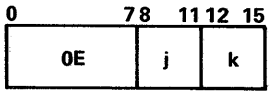
- Privileged instruction fault.
- Address specification error (k = 0, 1, 8, or 9).
- Invalid segments.

Copy to/from State Buffer

These instructions copy certain state registers to and from X registers.

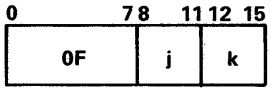
If a copy instruction reads a nonexistent register or any register restricted to MCU access only, the system clears all 64 bits of Xk. A copy instruction used to write a nonexistent register, or any register restricted to read only or MCU access only, results in a no-operation.

0Ejk Copy to Xk from State Register per Xj CPYSX Xk,Xj



This instruction copies the state register addressed by Xj into Xk.

0Fjk Copy to state register from Xk per Xj CPYXS Xk,Xj



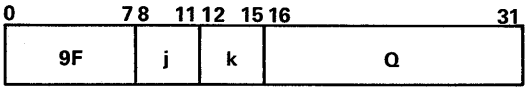
This instruction copies Xk into the state register addressed by Xj. Refer to tables I-2-1 and I-2-2 in volume 1.

These instructions can cause the following exception conditions:

- Instruction specification error.
- Privileged instruction fault (CPYXS only).

Branch on Condition Register

9FjkQ Branch to P Displaced by 2*Q and Alter Condition Register per jk BRCR j,k,Q



This instruction tests the state of a bit in the monitor or user condition register (MCR or UCR), as selected by the instruction j and k fields. The j field selects the bit within the register, and the k field selects the register, branch condition, and bit alteration. When the branch condition is satisfied, the target address forms by adding 2 times Q (sign-extended) to the BN in P. The instruction depends on k as follows:

Value of k	Description
k = 0 or 8	If bit j of MCR is set, clear bit and branch.
k = 1 or 9	If bit j of MCR is clear, set bit and branch.
k = 2 or A	If bit j of MCR is set, branch.
k = 3 or B	If bit j of MCR is clear, branch.
k = 4 or C	If bit j of UCR is set, clear bit and branch.
k = 5 or D	If bit j of UCR is clear, set bit and branch.

Value of k	Description
k = 6 or E	If bit j of UCR is set, branch.
k = 7 or F	If bit j of UCR is clear, branch.

When the k field is 0, 8, 1, or 9, this instruction executes in Virtual State monitor mode only. If the processor is not in monitor mode with execution restricted to that mode, the CP detects an instruction specification error, inhibits instruction execution, and initiates the corresponding program interrupt or halt.

This instruction can cause the following exception conditions:

- Instruction specification error.
- Debug.

PERIPHERAL PROCESSOR INSTRUCTION DESCRIPTIONS

The peripheral processor (PP) instruction set comprises the following eight subgroups:

- Load and store.
- Arithmetic.
- Logical.
- Replace.
- Branch.
- Central memory (CM) access.
- Input/Output (I/O).
- Other IOU.

The Virtual State PP instruction set includes the CYBER 170 State PP instructions as a subset. The instruction set uses a 7-bit operation code (opcode) which includes the CYBER 170 State 6-bit operation code. Extensions to the instruction set allow programs to manipulate 16-bit IOU words, 64-bit CM words (as both 12- and 16-bit bytes), and to reference 28-bit CM addresses.

PP INSTRUCTION FORMATS

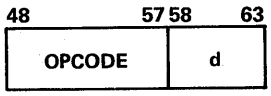
Figure II-1-4 shows and describes Virtual State PP instruction formats. PP instructions are 16 or 32 bits long. In instruction descriptions, the opcode is represented by four or five octal digits. The fifth digit, when used, indicates the state of bit 58/42 (zero or one) in I/O instructions.

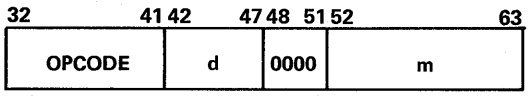
PP DATA FORMAT

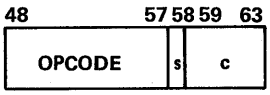
Figure II-1-5 shows PP data formats, the packing of PP data into CM words, and the unpacking of CM words into PP words.

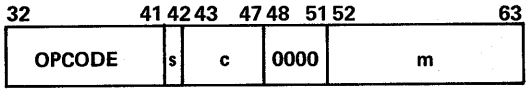
PP RELOCATION REGISTER FORMAT

Figure II-1-5 shows the PP relocation register format. This register is loaded/stored by instructions 24 and 25 (load/store R register).









Term	Description
OPCODE	Instruction operation code; bits 49 through 51 or 33 through 35 of which are zeros.
d	Operand, direct/indirect address, shift count.
m	Operand, direct address or I/O function code.
dm	Operand.
s	I/O instruction subcode.
c	I/O instruction channel number.
A	Eighteen-bit arithmetic register.
P	Twelve-bit program address counter.
R	Twenty-eight-bit relocation register for central memory addressing (bits 58 through 63 of R are appended zeros).
( )	Quantity in brackets is a direct address (used when required for clarity).
(( ))	Quantity in brackets is an indirect address (used when required for clarity).

Figure II-1-4. PP Instruction Formats and Nomenclature

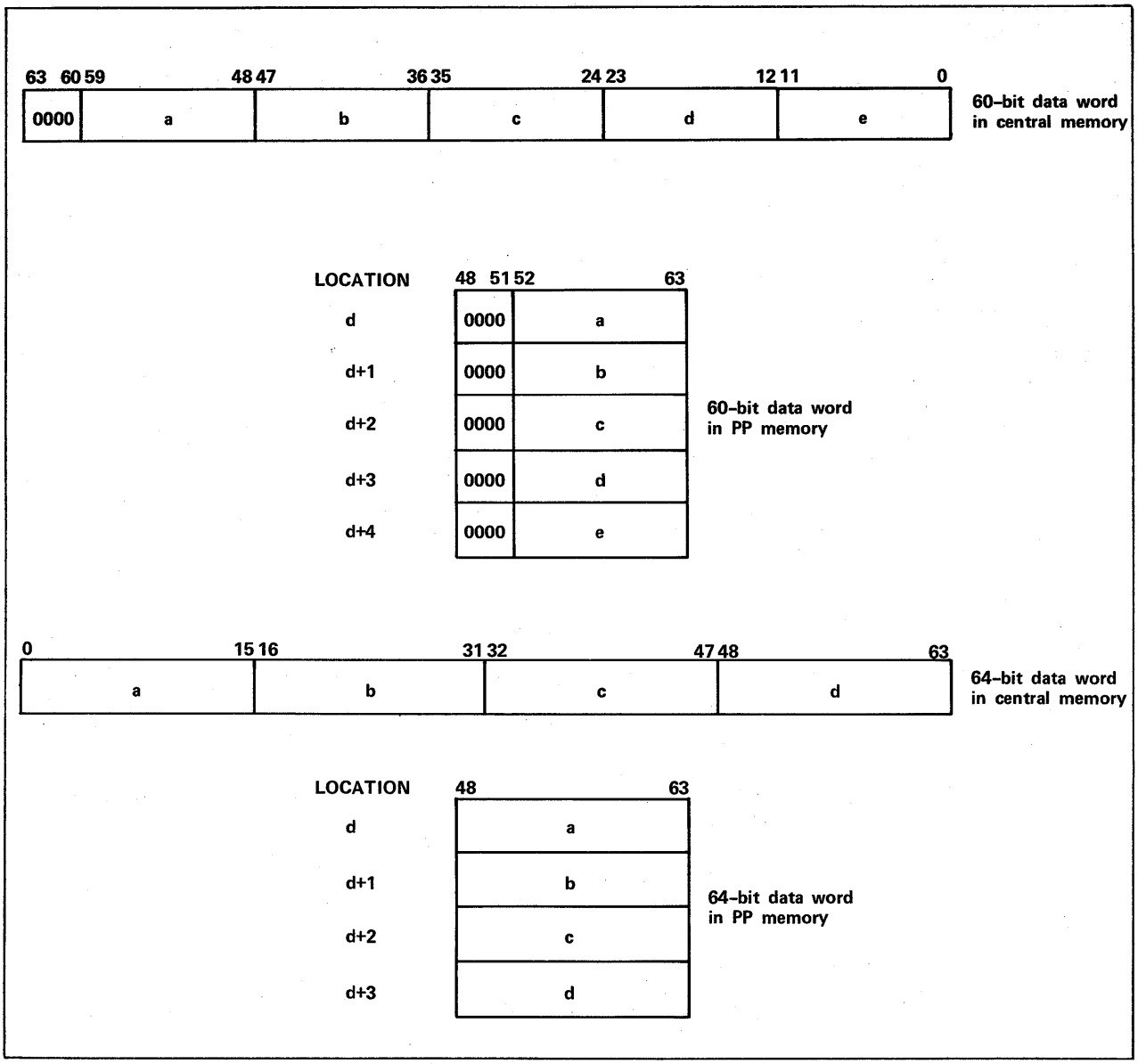


Figure II-1-5. PP Data Format

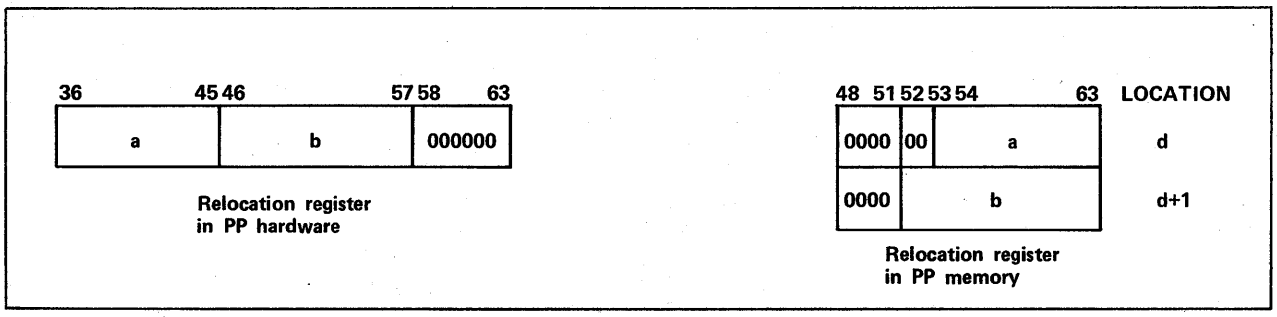


Figure II-1-6. PP Relocation Register Format

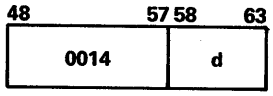
PP LOAD AND STORE INSTRUCTIONS

Load and store instructions (table II-1-22) transfer 6-, 12-, 16-, and 18-bit quantities between the PP A register and the PP memory.

Table II-1-22. PP Load and Store Instructions

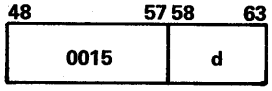
Opcode	Instruction
0014	Load d
0015	Load complement d
0020	Load dm
0030	Load (d)
1030	Load (d) long
0034	Store (d)
1034	Store (d) long
0040	Load ((d))
1040	Load ((d)) long
0044	Store ((d))
1044	Store ((d)) long
0050	Load (m+(d))
1050	Load (m+(d)) long
0054	Store (m+(d))
1054	Store (m+(d)) long

0014d Load d LDN d



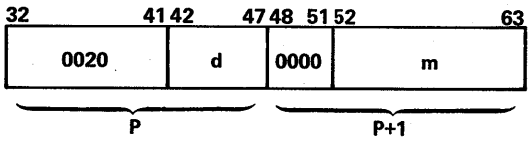
This instruction clears the A register and loads the rightmost 6 bits of A with a copy of the 6-bit positive integer in the d field. The leftmost 12 bits of A are zeros.

0015d Load Complement d LCN d



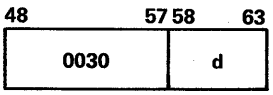
This instruction clears the A register and loads the rightmost 6 bits of A with a one's complement copy of the d field. The leftmost 12 bits of A are ones.

0020dm Load dm LDC dm



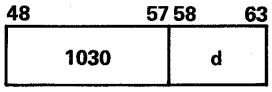
This instruction clears the A register and loads A with an 18-bit operand consisting of d as the leftmost 6 bits and m as the rightmost 12 bits.

0030d Load (d) LDD d



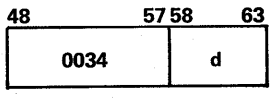
This instruction clears the A register and loads A with the rightmost 12 bits of the positive integer from PP memory location d. The leftmost 6 bits of A are zeros.

1030d Load (d) long LDDL d



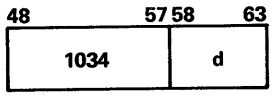
This instruction clears the A register and loads A with the 16-bit positive integer from PP memory location d. The leftmost 2 bits of A are zeros.

0034d Store (d) STD d



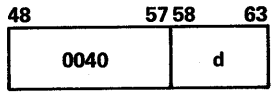
This instruction stores the quantity contained in the A register rightmost 12 bits in location d, and clears the leftmost 4 bits of location d. The operation does not alter the contents of A.

1034d Store (d) Long STDL d



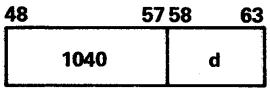
This instruction stores the A register rightmost 16 bits in location d. The operation does not alter the contents of A.

0040d Load ((d)) LDI d



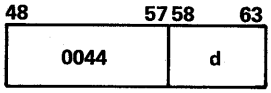
This instruction clears the A register and loads A with the rightmost 12 bits of an operand obtained by indirect addressing. The leftmost 6 bits of A are zeros. To perform indirect addressing, the IOU reads a word from PP memory location d and uses it as the operand address.

1040d Load ((d)) Long LDIL d



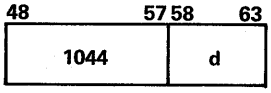
This instruction clears the A register and loads A with a 16-bit operand obtained by indirect addressing. Refer to instruction 0040. The leftmost 2 bits of A are zeros.

0044d Store ((d)) STI d



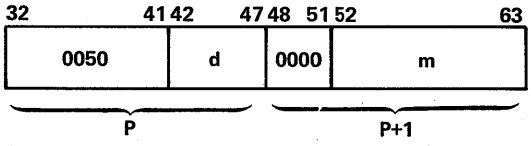
This instruction stores the A register rightmost 12 bits in the location specified by the location d contents. The leftmost 4 bits of ((d)) are zeros and the A register contents are unaltered.

1044d Store ((d)) Long STI d



This instruction stores the A register rightmost 16 bits at a location obtained by indirect addressing. Refer to instruction 0044.

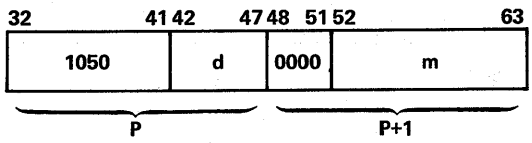
0050dm Load (m+(d)) LDM m,d



This instruction clears the A register and loads A with the rightmost 12 bits of an operand read from PP memory using indexed direct addressing. The leftmost 6 bits of A are zeros.

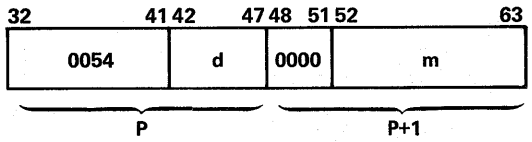
To accomplish indexed direct addressing, the IOU adds an index value to a base address to form the operand address. The m field contains the base address and the d field specifies the PP memory location containing the index value. If d equals 0, the m field base address is the operand address.

1050dm Load (m+(d)) Long LDML m,d



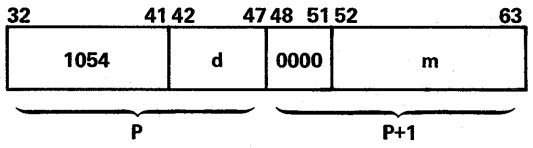
This instruction clears the A register and loads A with a 16-bit operand read from PP memory using indexed direct addressing. Refer to instruction 0050. The leftmost 2 bits of A are zeros.

0054dm Store (m+(d)) STM m,d



This instruction stores the A register rightmost 12 bits in the location determined by indexed direct addressing. Refer to instruction 0050. Bits 48 through 51 of (m+(d)) clear.

1054dm Store (m+(d)) Long STML m,d



This instruction stores the A register rightmost 16 bits in the location determined by indexed direct addressing. Refer to instruction 0050.

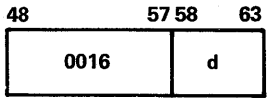
PP ARITHMETIC INSTRUCTIONS

The PP arithmetic instructions (table II-1-23) perform integer arithmetic using the PP A register contents as one operand, with the other operand specified by the instruction. The result replaces the original contents of A. The IOU considers the operands as one's complement integers and performs the arithmetic in one's complement.

Table II-1-23. PP Arithmetic Instructions

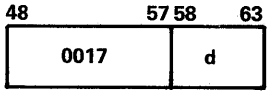
Opcode	Instruction
0016	Add d
0017	Subtract d
0021	Add dm
0031	Add (d)
1031	Add (d) long
0032	Subtract (d)
1032	Subtract (d) long
0041	Add ((d))
1041	Add ((d)) long
0042	Subtract ((d))
1042	Subtract ((d)) long
0051	Add (m+(d))
1051	Add (m+(d)) long
0052	Subtract (m+(d))
1052	Subtract (m+(d)) long

0016d Add d ADN d



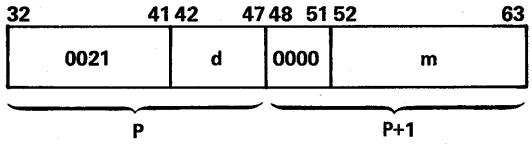
This instruction adds d (treated as a 6-bit positive quantity) to the A register contents.

0017d Subtract d SBN d



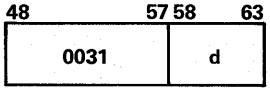
This instruction subtracts d (treated as a 6-bit positive quantity) from the A register contents.

0021dm Add dm ADC dm



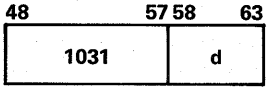
This instruction adds an 18-bit value comprised of the 6-bit d field and the m field rightmost 12 bits to the A register operand. The d field becomes the rightmost 6 bits and m becomes the rightmost 12 bits of the value.

0031d Add (d) ADD d



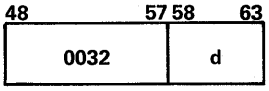
This instruction adds the rightmost 12 bits of the operand in location d to the A register contents.

1031d Add (d) Long ADDL d



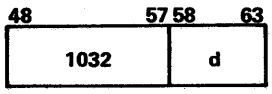
This instruction adds the 16-bit operand in location d to the A register contents.

0032d Subtract (d) SBD d



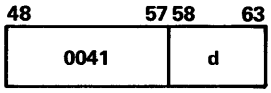
This instruction subtracts the 12-bit operand in location d from the A register contents.

1032d Subtract (d) Long SBDL d



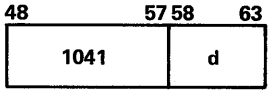
This instruction subtracts the 16-bit operand in location d from the A register contents.

0041d Add ((d)) ADI d



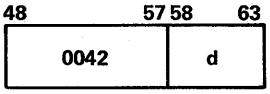
This instruction reads a 12-bit operand from PP memory (PPM) using indirect addressing, and adds the rightmost 12 bits to the A register contents. To perform indirect addressing, the PP reads a word from PPM location d and uses it as the operand address.

1041d Add ((d)) Long AIDL d



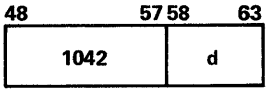
This instruction reads a 16-bit operand from PP memory (PPM) using indirect addressing, and adds the operand to the A register contents. To perform indirect addressing, the PP reads a word from PPM location d and uses it as the operand address.

0042d Subtract ((d)) SBIL d



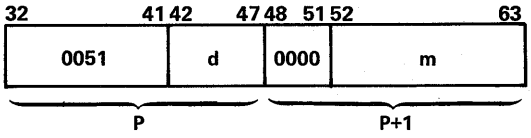
This instruction reads a 12-bit operand from PP memory (PPM) using indirect addressing, and subtracts the operand from the A register contents. To perform indirect addressing, the PP reads a word from PPM location d and uses it as the operand address.

1042d Subtract ((d)) Long SBIL d



This instruction reads a 16-bit operand from PP memory (PPM) using indirect addressing, and subtracts the operand from the A register contents. To perform indirect addressing, the PP reads a word from PPM location d and uses it as the operand address.

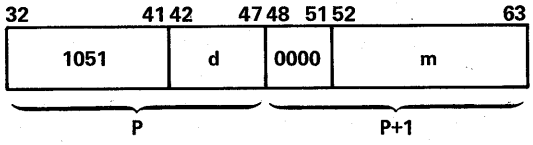
0051dm Add (m+(d)) ADM m,d



This instruction reads an operand from PP memory (PPM) using indexed direct addressing, and adds the rightmost 12 bits to the A register contents.

To accomplish indexed direct addressing, the PP adds an index value to a base address to form the operand address. The m field contains the base address and the d field specifies the PPM location containing the index value. If d equals 0, the m field base address is the operand address.

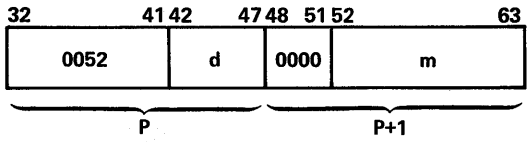
1051dm Add (m+(d)) Long ADML m,d



This instruction reads a 16-bit operand from PP memory (PPM) using indexed direct addressing and adds the operand to the A register contents.

To accomplish indexed direct addressing, the PP adds an index value to a base address to form the operand address. The m field contains the base address and the d field specifies the PPM location containing the index value. If d equals 0, the m field base address is the operand address.

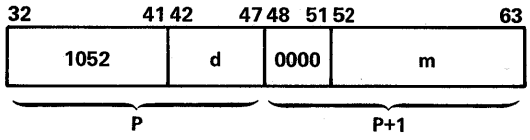
0052dm Subtract (m+(d)) SBM m,d



This instruction reads an operand from PP memory (PPM) using indexed direct addressing and subtracts the rightmost 12 bits of the operand from the A register contents.

To accomplish indexed direct addressing, the PP adds an index value to a base address to form the operand address. The m field contains the base address and the d field specifies the PPM location containing the index value. If d equals 0, the m field base address is the operand address.

1052dm Subtract (m+(d)) Long SBML m,d



This instruction reads a 16-bit operand from PP memory (PPM) using indexed direct addressing, and subtracts this operand from the A register contents.

To accomplish indexed direct addressing, the PP adds an index value to a base address to form the operand address. The m field contains the base address and the d field specifies the PPM location containing the index value. If d equals 0, the m field base address is the operand address.

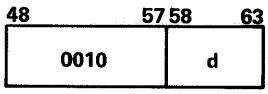
PP LOGICAL INSTRUCTIONS

The logical instructions (table II-1-24) perform operations with one operand as the PP A register contents, and the other as specified by the instruction. The result replaces the original contents of A.

Table II-1-24. PP Logical Instructions

Opcode	Instruction
0010	Shift d
0011	Logical difference d
0012	Logical product d
0013	Selective clear d
0022	Logical product dm
1022	Logical product (d) long
0023	Logical difference dm
1023	Logical product ((d)) long
1024	Logical product (m+(d)) long
0033	Logical difference (d)
1033	Logical difference (d) long
0043	Logical difference ((d))
1043	Logical difference ((d)) long
0053	Logical difference (m+(d))
1053	Logical difference (m+(d)) long

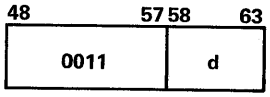
0010d Shift A by d SHN d



This instruction shifts the A register operand in the direction and by the number of places specified by the d field value. If d is in the range 00 through 37 (a positive value), the shift is left-circular, d positions. A circular shift means that a bit shifted out of the highest-order position moves into the lowest-order position.

If d is in the range 40 through 77 (a negative value), the shift is to the right, end-off. Thus, d equal to 06 causes a left shift of 6 places; d equal to 71 causes a right shift of 6 places.

0011d Logical Difference d LMN d



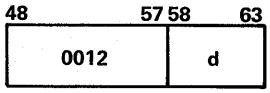
This instruction forms the logical difference between the d field contents and the rightmost 6 bits of the A register operand. The operation does not alter the most significant 12 bits of A.

The logical difference (exclusive OR) results from a bit-for-bit logical comparison of the two binary quantities, as illustrated by the following example:

Operand 1	0011
Operand 2	0101
Result	0110

This comparison is equivalent to complementing the first operand bits corresponding to the second operand bits that are ones.

0012d Logical Product A and d LPN d



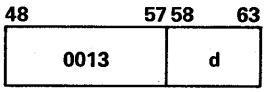
This instruction forms the logical product of the d field contents and the rightmost 6 bits of the A register operand. The leftmost 12 bits of A are zeros.

The logical product results from a bit-for-bit logical comparison of the two binary quantities, as illustrated by the following example:

Operand 1	0011
Operand 2	0101
Result	0001

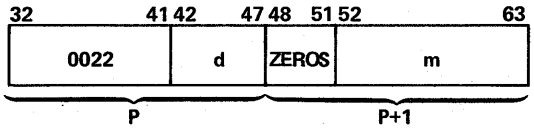
The individual result bits are ones if the corresponding bits in the first and second operand are ones.

0013d Selective Clear d SCN d



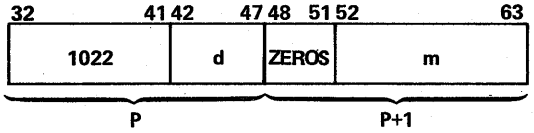
This instruction clears each of the rightmost 6 bits of the A register operand if the corresponding d field bit is a 1. The operation does not alter the leftmost 12 bits of A.

0022dm Logic Product dm LPC dm



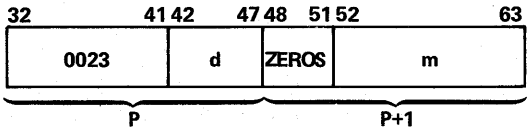
This instruction forms the bit-for-bit logical product of the A register operand and an 18-bit operand consisting of d as the leftmost 6 bits and m as the rightmost 12 bits. Refer to instruction 0012 for an example of the logical product operation.

1022d Logical Product (d) Long LPDL d



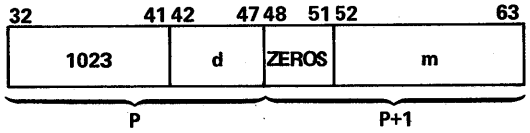
This instruction forms the bit-for-bit logical product of the A register operand and the 16-bit quantity from location d. The result replaces the original contents of A; bits 46 and 47 of A clear. Refer to instruction 0012 for an example of the logical product operation.

0023dm Logical difference dm LMC dm



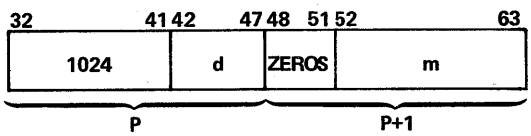
This instruction replaces the A register contents with the bit-for-bit logical difference between the A register operand and the 18-bit quantity dm. Refer to instruction 0011 for an example of the logical difference operation.

1023d Logical Product ((d)) Long LPIL d



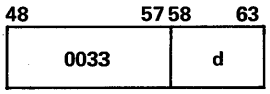
This instruction forms the logical product of a 16-bit operand read from storage (using indirect addressing) and the original contents of A. Bits 46 and 47 of A clear. The d field contents specify the PP memory location containing the operand address.

1024dm Logical Product (m+(d)) Long LPMLm d



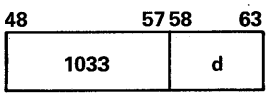
This instruction replaces the A register contents with the logical product of the A register rightmost 16 bits and a 16-bit operand read by indexed direct addressing. Bits 46 and 47 of A clear. Indexed direct addressing uses the m field contents as a base address. The d field specifies a PP memory location containing the index value which adds to the base address to form the operand address.

0033d Logical Difference (d) LMD d



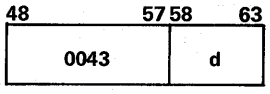
This instruction replaces the A register contents with the logical difference between the A register rightmost 12 bits and the rightmost 12 bits of the operand in the location specified by the d field. The operation does not alter the leftmost 6 bits of A.

1033d Logical Difference (d) Long LMDL d



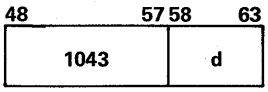
This instruction replaces the A register contents with the logical difference between the A register rightmost 16 bits and the operand in the location specified by the d field. The operation does not alter the most significant 2 bits of A.

0043d Logical Difference ((d)) LMI d



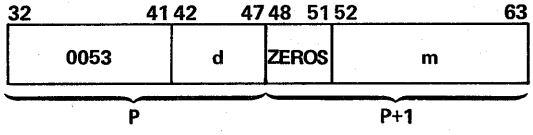
This instruction replaces the A register contents with the logical difference between the A register rightmost 12 bits and the rightmost 12 bits of an operand read by indirect addressing. The operation does not alter the leftmost 6 bits of A. Refer to instruction 0011 for an example of the logical difference operation. The d field contents specify the PP memory location containing the operand address.

1043d Logical Difference ((d)) Long LMIL d



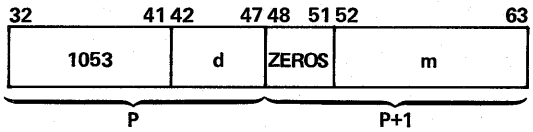
This instruction replaces the A register contents with the logical difference between the A register rightmost 16 bits and an operand read by indirect addressing. The operation does not alter the leftmost 2 bits of A. Refer to instruction 0011 for an example of the logical difference operation. The d field contents specify the PP memory location containing the operand address.

0053dm Logical Difference (m+(d)) LMM m,d



This instruction replaces the A register contents with the logical difference between the A register rightmost 12 bits and the rightmost 12 bits of an operand read by indexed direct addressing. The operation does not alter the most significant 6 bits of A. Refer to instruction 0011 for an example of the logical difference operation. Indexed direct addressing uses the m field contents as a base address. The d field specifies the PP memory location containing the index value which the PP adds to the base address to form the operand address.

1053dm Logical Difference (m+(d)) Long LMML m,d



This instruction replaces the A register contents with the logical difference between the A register rightmost 16 bits and an operand read by indexed direct addressing. The operation does not alter the leftmost 2 bits of A. Refer to instruction 0011 for an example of the logical difference operation. Refer to instruction 0053 for a description of indexed direct addressing.

PP REPLACE INSTRUCTIONS

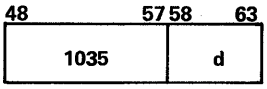
The replace instructions (table II-1-25) perform integer arithmetic with one operand as the contents of A and the other as specified by the instruction. The result replaces the original contents of A and the contents of the other operand's location. The result stored in location d is either the rightmost 12 bits (for the normal instructions) or the rightmost 16 bits (for the long instructions) of the A register. Therefore, since A contains 18 bits, the value remaining in A cannot equal the value stored in PP memory location d.

The PP considers the operands as one's complement integers and performs one's complement arithmetic.

Table II-1-25. PP Replace Instructions

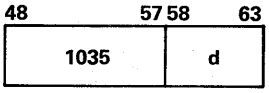
Opcode	Instruction
0035	Replace add (d)
1035	Replace add (d) long
0036	Replace add one (d)
1036	Replace add one (d) long
0037	Replace subtract one (d)
1037	Replace subtract one (d) long
0045	Replace add ((d))
1045	Replace add ((d)) long
0046	Replace add one ((d))
1046	Replace add one ((d)) long
0047	Replace subtract one ((d))
1047	Replace subtract one ((d)) long
0055	Replace add (m+(d))
1055	Replace add (m+(d)) long
0056	Replace add one (m+(d))
1056	Replace add one (m+(d)) long
0057	Replace subtract one (m+(d))
1057	Replace subtract one (m+(d)) long

0035d Replace Add (d) RAD d



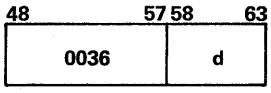
This instruction adds the rightmost 12 bits of the location d contents to the A register, and stores the rightmost 12 bits of A, zero-extended, at location d. The result also remains in A at the end of the operation, with the original contents purged.

1035d Replace Add (d) Long RADL d



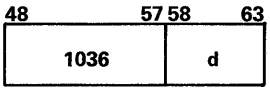
This instruction replaces the operand in the PP memory (PPM) location d with the sum of the PPM location d operand plus the A register rightmost 16 bits. Refer to instruction 0035.

0036d Replace Add One (d) AOD d



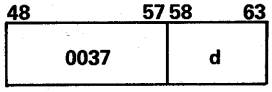
This instruction clears the A register, loads A with the location d rightmost 12 bits, and adds one to A. The instruction then stores the rightmost 12 bits of A, zero-extended, at location d. The result remains in the A register at the end of the operation, with the original contents purged.

1036d Replace Add One (d) Long AODL d



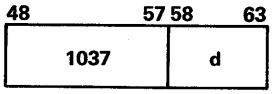
This instruction replaces the operand in PP memory (PPM) location d with the sum of the original operand value plus 1. Refer to instruction 0036.

0037d Replace Subtract One (d) SOD d



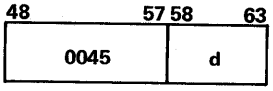
This instruction replaces the rightmost 12 bits of the operand in PP memory (PPM) location d with the original operand value minus 1. Refer to instruction 0036.

1037d Replace Subtract One (d) Long SODL d



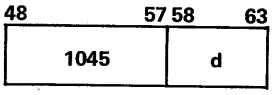
This instruction replaces the operand in PP memory (PPM) location d with the difference of the original operand value minus 1. Refer to instruction 0036.

0045d Replace Add ((d)) RAI d



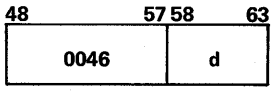
This instruction replaces the rightmost 12 bits of the operand at the address specified by the PP memory (PPM) location d contents with the sum of the original operand value plus the A register rightmost 12 bits. Refer to instruction 0035.

1045d Replace Add ((d)) Long RAIL d



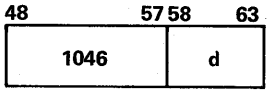
This instruction replaces the 16-bit operand at the address specified by the PP memory (PPM) location d contents with the sum of the original operand value plus the A register contents. Refer to instruction 0035.

0046d Replace Add One ((d)) AOI d



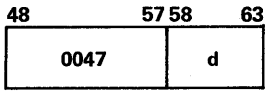
This instruction replaces the rightmost 12 bits of the operand at the address specified by the PP memory (PPM) location d contents with the sum of the original operand value plus 1. Refer to instruction 0036.

1046d Replace Add One ((d)) Long AOIL d



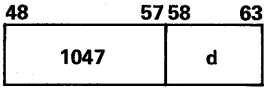
This instruction replaces the operand at the address specified by the PP memory (PPM) location d contents with the sum of the original operand value plus 1. Refer to instruction 0036.

0047d Replace Subtract One ((d)) SOI d



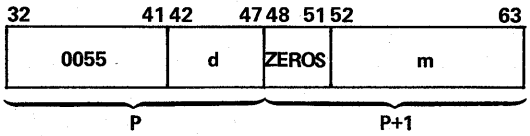
This instruction replaces the rightmost 12 bits of the operand at the address specified by the PP memory (PPM) location d contents with the original operand value minus 1. Refer to instruction 0036.

1047d Replace Subtract One ((d)) SOIL d



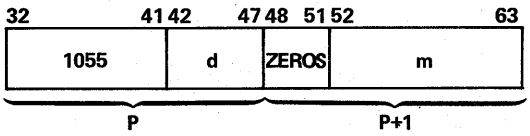
This instruction replaces the operand at the address specified by the PP memory (PPM) location d contents with the original operand value minus 1. Refer to instruction 0036.

0055dm Replace Add (m+(d)) RAM m,d



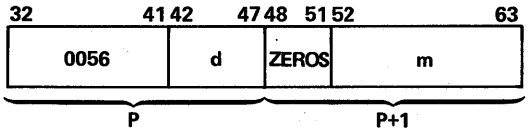
This instruction reads the rightmost 12 bits of an operand from PP memory (PPM) using indexed direct addressing, and adds the operand to the the A register contents. The sum's rightmost 12 bits replace the original PPM operand. The result remains in A at the end of the operation, with the original contents of A purged. Indexed direct addressing uses the m field contents as a base address. The d field specifies the PPM location containing the index value which the PP adds to the base address to form the operand address.

1055dm Replace Add (m+(d)) Long RAML m,d



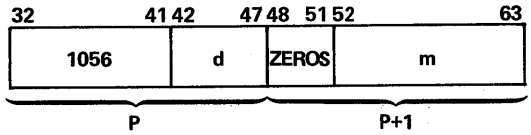
This instruction reads a 16-bit operand from PP memory (PPM) using indexed direct addressing, adds the operand to the A register contents, and replaces the original PPM operand with the sum's rightmost 16 bits. The result remains in A at the end of the operation. Refer to instruction 0055.

0056dm Replace Add One (m+(d)) AOM m,d



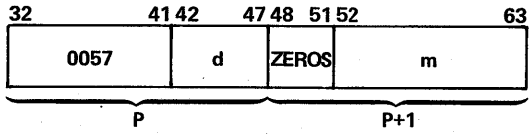
This instruction adds one to the rightmost 12 bits of a PP memory (PPM) operand read using indexed direct addressing, and replaces the original PPM operand with the sum's rightmost 12 bits. Refer to instruction 0055.

1056dm Replace Add One (m+(d)) Long AOML m,d



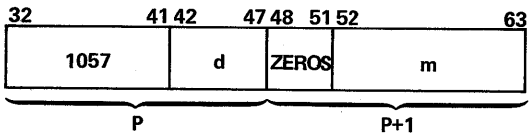
This instruction adds one to a 16-bit PP memory (PPM) operand read using indexed direct addressing, and replaces the original PPM operand with the sum's rightmost 16 bits. Refer to instruction 0055.

0057dm Replace Subtract One (m+(d)) SOM m,d



This instruction subtracts one from the rightmost 12 bits of a PP memory (PPM) operand read using indexed direct addressing, and replaces the original PPM operand with the rightmost 12 bits of the difference. Refer to instructions 0036, 0055.

1057dm Replace Subtract One (m+(d)) Long SOML m,d



This instruction subtracts one from a 16-bit PP memory (PPM) operand read using indexed direct addressing, and replaces the original PPM operand with the rightmost 16 bits of the difference. Refer to instructions 0036, 0055.

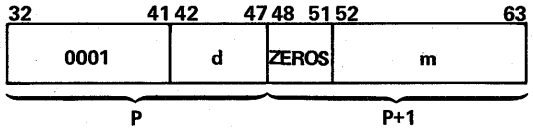
PP BRANCH INSTRUCTIONS

The branch instructions (table II-1-26) allow departure from sequential instruction execution.

Table II-1-26. PP Branch Instructions

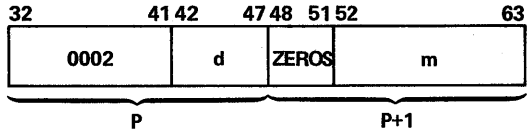
Opcode	Instruction
0001	Long jump to m+(d)
0002	Return jump to m+(d)
0003	Unconditional jump d
0004	Zero jump d
0005	Nonzero jump d
0006	Plus jump d
0007	Minus jump d

0001dm Long Jump to m+(d) LJM m,d



The long jump instruction branches to the address formed by adding the m field rightmost 12 bits to the location d rightmost 12 bits. The result is the first word's address in the new instruction sequence. If d equals zero, the m field contents is the jump address.

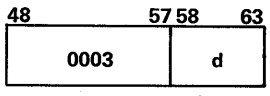
0002dm Return Jump to m+(d) RJM m,d



This instruction stores the current program address plus two (P+2) in the address formed from m+(d). (If the d field is zero, the m field contents is the address.) The instruction then branches to location m+(d)+1.

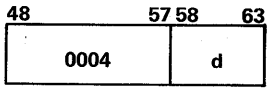
This instruction interrupts the current program sequence and jumps to a subroutine while providing the means for a return to the original program. The entry point address to the subroutine forms from m+(d). The called subroutine must have a common exit point in the form of a long-jump-to-m instruction (operation code 0001) preceding the entry point (location m+(d)-1). The return jump instruction stores the current program address plus two (P+2) in the first word of the subroutine (location m+(d)). This word is the second half of the long jump instruction (its m field) and is the return address to the original program sequence. The instruction then branches to m+(d)+1. When the subroutine completes, the long jump instruction at location m+(d)-1 executes and performs a branch to the return address in the original program sequence.

0003d Unconditional Jump d UJN d



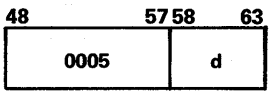
This instruction causes an unconditional branch to any address up to 31 (decimal) locations forward or backward from the current program address. If d is positive (01 through 37 octal), the jump is forward. If d is negative (40 through 76 octal), the jump is backward.

0004d Zero Jump d ZJN d



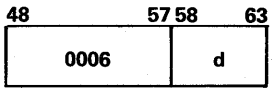
If the contents of A is zero, this instruction causes an unconditional branch to any address up to 31 (decimal) locations forward or backward from the current program address. The instruction adds the d value to the current program address. If d is positive (01 through 37 octal), the jump is forward. If d is negative (40 through 76 octal), the jump is backward.

0005d Nonzero Jump d NJN d



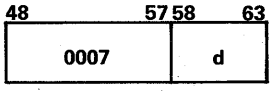
If the contents of A is nonzero, this instruction causes an unconditional branch to any address up to 31 (decimal) locations forward or backward from the current program address. If d is positive (01 through 37 octal), the jump is forward. If d is negative (40 through 76 octal) the jump is backward.

0006d Plus Jump d PJN d



If the contents of A is positive, this instruction causes an unconditional branch to any address up to 31 (decimal) locations forward or backward from the current program address. The instruction adds the d value to the current program address. If d is positive (01 through 37 octal), the jump is forward. If d is negative (40 through 76 octal), the jump is backward.

0007d Minus Jump d MJN d



If the contents of A is negative, this instruction causes an unconditional branch to any address up to 31 (decimal) locations forward or backward from the current program address. The instruction adds the d value to the current program address. If d is positive (01 through 37 octal), the jump is forward. If d is negative (40 through 76 octal), the jump is backward.

PP CENTRAL MEMORY ACCESS INSTRUCTIONS

The CM access instructions (table II-1-27) provide the capability to read and write CM words to and from PP memory. The PPs have read access to all CM storage locations, while the OS bounds register controls write and exchange accesses. (Refer to IOU Maintenance Registers in section 2 of volume 1.) The IOU performs CM addressing with real memory word addresses. To address all locations in the larger CM sizes available, the IOU uses address relocation to modify the CM address in the A register of the PP. Refer to figure II-1-7. If bit 46 in A is a 1 during a PP central memory read or write instruction, the IOU adds the R register contents to A register bits 47 through 63 to produce the CM address. If bit 46 of A is 0, the IOU does not perform address relocation but uses the A address. The R register contains an absolute 64-word starting boundary within CM. When relocation is desired, an absolute CM address forms by concatenating six 0s to the rightmost end of the R contents, and adding bits 47 through 63 of A.

Table II-1-27. PP Central Memory Access Instructions

Opcode	Instruction
0024	Load R register
0025	Store R register
0060	Central read from (A) to d
1060	Central read from (A) to d long
0061	Central read (d) words from (A) to m
1061	Central read (d) words from (A) to m long
0062	Central write to (A) from d
1062	Central write to (A) from d long
0063	Central write (d) words to (A) from m
1063	Central write (d) words to (A) from m long
1000	Central read and set lock from d to (A)
1001	Central read and clear lock from d to (A)

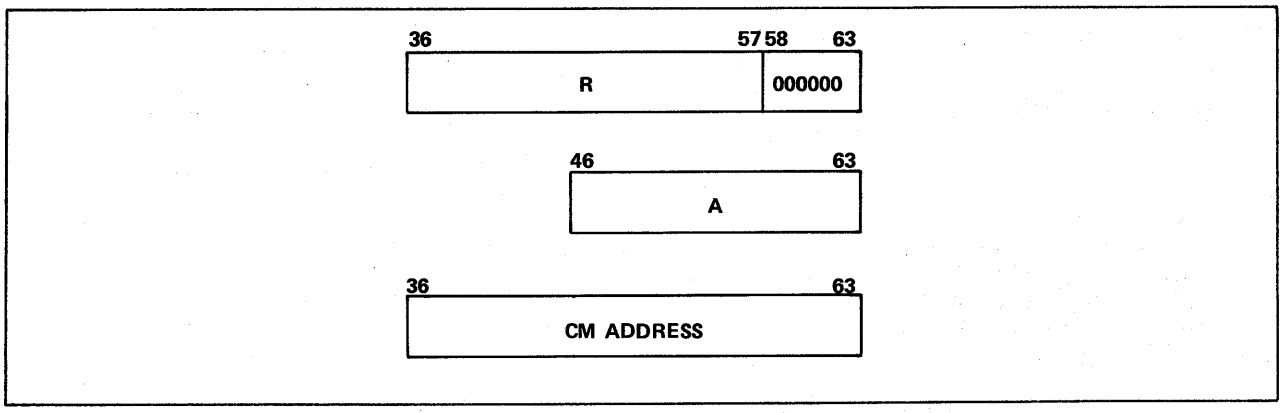
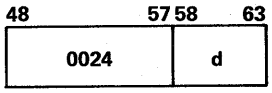


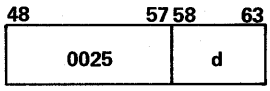
Figure II-1-7. Relocation Address Formation

0024d Load R Register LRD d



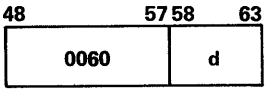
See figure II-1-6. This instruction loads the 22-bit R register from PP memory (PPM) locations d and d+1. If the instruction d field is not zero, the instruction loads bits 47 through 57 of R from bits 52 through 63 of location (d)+1. Bits 36 through 45 of R are loaded from bits 54 through 63 of (d). If the d field is zero, this instruction is a pass.

0025d Store R Register SRD d



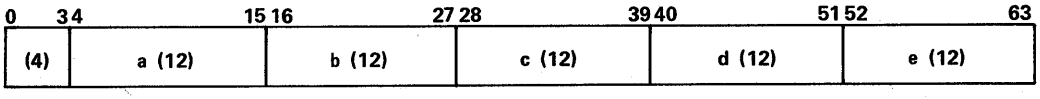
See figure II-1-3. This instruction stores the 22-bit R register contents into PP memory locations d and d+1. If d is nonzero, the instruction stores bits 46 through 57 of R in bit positions 52 through 63 of (d)+1, and bits 36 through 45 of R store in bit positions 54 through 63 of (d). Bits 48 through 51 of (d)+1 and 48 through 53 of (d) clear. If the d field is zero, this instruction is a pass.

0060d Central Read from (A) to d CRD d

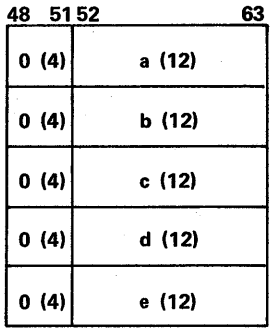


This instruction transfers the rightmost 60 bits of one CM word to the rightmost 12 bits of 5 consecutive PP memory (PPM) words. The IOU discards the leftmost 4 bits of the CM word and disassembles the remaining 60 bits from left to right into five 12-bit bytes. The following illustrations show this unpacking. R+A specifies the CM word address (refer to figure II-1-7); d specifies the first PPM word address.

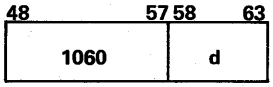
The CM word is as follows:



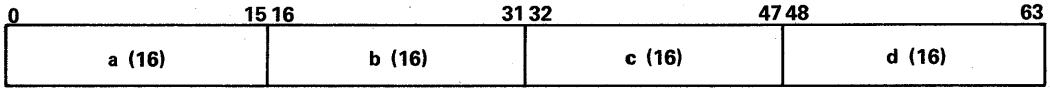
The PPM words formed by unpacking one CM word are as follows:



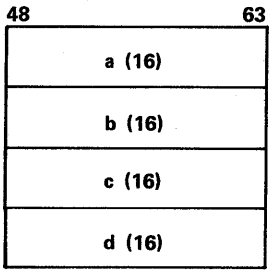
1060d Central Read from (A) to d Long CRDL d



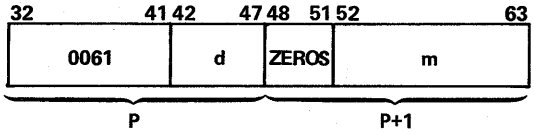
This instruction transfers one CM word to four consecutive PP memory (PPM) words. The IOU disassembles the CM word from the left. The following illustrations show this unpacking. R+A specifies the CM word address (refer to figure II-1-7); d specifies the first PPM word address. The CM word is as follows:



The PP memory words formed by unpacking one CM word are as follows:



0061dm Central Read (d) Words from (A) to m CRM m,d

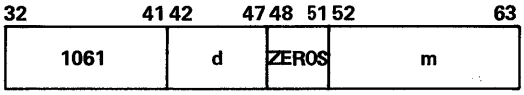


This instruction transfers the rightmost 60 bits of consecutive CM words to the rightmost 12 bits of consecutive PP memory (PPM) words. The PP discards the leftmost 4 bits of each CM word and disassembles the remaining 60 bits from the left into five 12-bit bytes. Refer to instruction 0060 for an illustration of unpacking. R+A specifies the first CM word address (refer to figure II-1-7), m specifies the first PPM word address, and location d specifies the number of CM words transferred. Upon completion, A contains the nonrelocated portion of the CM address (plus one) for the last word transferred.

NOTE

If the value of the rightmost 17 bits of A exceeds $(2^{17})-1$, the leftmost bit toggles, switching the operation from direct address to relocation address mode. If the last word transferred is from a relative address of 377776_8 and relocation is in effect, the PP clears the A register and the value returned to A may not point to the last word transferred plus 1. Also, when bit 17 switches to 0, addressing switches to the direct addressing mode.

1061dm Central Read (d) Words from (A) to m Long CRML m,d

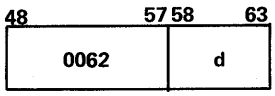


This instruction transfers consecutive CM words to consecutive PP memory (PPM) words. The IOU disassembles each CM word from the left. Refer to the 1060 instruction for an illustration of this unpacking. R+A specifies the first CM word address (refer to figure II-1-7), m specifies the first PP memory word address, and location d specifies the number of CM words transferred. On completion, A contains the nonrelocated portion of the CM address (plus one) for the last word transferred.

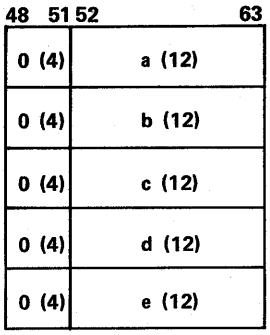
NOTE

If the value of the rightmost 17 bits of A exceeds $(2^{17})-1$, the leftmost bit toggles, switching the operation from direct address to relocation address mode. If the last word transferred is from a relative address of 377776_8 and relocation is in effect, the PP clears the A register and the value returned to A may not point to the last word transferred plus 1. Also, when bit 17 switches to zero, addressing switches to the direct addressing mode.

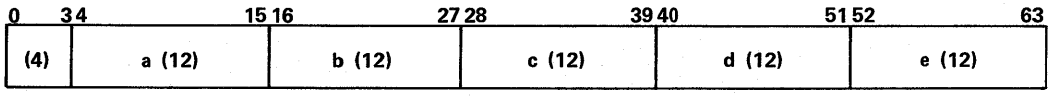
0062d Central Write to (A) from d CWD d



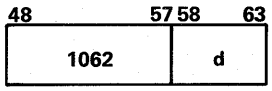
This instruction transfers the rightmost 12 bits of 5 consecutive PP memory (PPM) words to the rightmost 60 bits of one CM word. (The IOU ignores the leftmost 4 bits of the words.) These words are assembled from left to right, as shown in the following illustration. R+A specifies the CM word address (see figure II-1-7); d specifies the first PPM word address. The IOU verifies each CM address against the OS bounds address. PPM words are as follows:



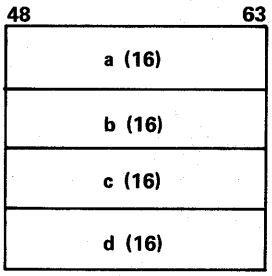
The CM word formed by packing five 12-bit PP memory words is as follows:



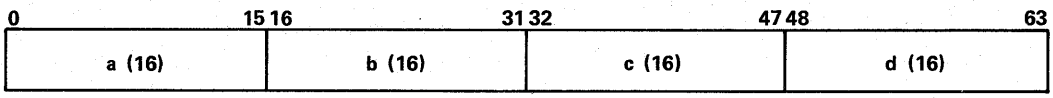
1062d Central Write to A from d Long CWDL d



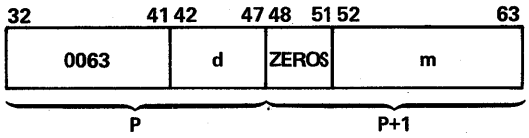
This instruction transfers four consecutive PP memory (PPM) words to one CM word, as shown in the following illustration. R+A specifies the CM word address (see figure II-1-7); d specifies the first PPM word address. The IOU verifies each CM address against the OS bounds address. The PPM words are as follows:



The CM word formed by packing four 16-bit PPM words is as follows:



0063dm Central Write (d) Words to (A) from m CWM m,d

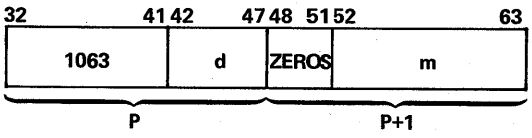


This instruction transfers the rightmost 12 bits of consecutive PP memory (PPM) words to the rightmost 60 bits of consecutive CM words. Refer to the instruction 0062 for an illustration of this packing. R+A specifies the first CM word address (see figure II-1-7), m specifies the first PPM word address, and d specifies the number of CM words transferred. The IOU verifies the CM address against the OS bounds address. On completion, A contains the nonrelocated portion of the CM address (plus 1) for the last word transferred.

NOTE

If the value of the rightmost 17 bits of A exceeds $(2^{17})-1$, the leftmost bit toggles, switching the operation from direct address to relocation address mode. If the last word transferred is from a relative address of 377776_8 and relocation is in effect, the PP clears the A register and the value returned to A may not point to the last word transferred plus one. Also, when bit 17 switches to 0, addressing switches to the direct addressing mode.

1063dm Central Write (d) Words to (A) from m Long CWML m,d

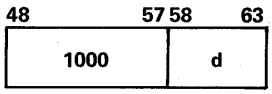


This instruction transfers consecutive PP memory (PPM) words to consecutive CM words. The IOU packs four PPM words (from the left) into each CM word. R+A specifies the first CM word address (see figure II-1-7), m specifies the first PPM word address, and d specifies the number of CM words transferred. The IOU verifies each CM address against the OS bounds address. On completion, A contains the nonrelocated portion of the CM address (plus 1), for the last word transferred.

NOTE

If the value of the rightmost 17 bits of A exceeds $(2^{17})-1$, the leftmost bit toggles, switching the operation from direct address to relocation address mode. If the last word transferred is from a relative address of 377776_8 and relocation is in effect, the PP clears the A register and the value returned to A may not point to the last word transferred plus 1. Also, when bit 17 switches to 0, addressing switches to the direct addressing mode.

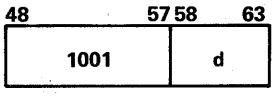
1000d Central Read and Set Lock from d to (A) RDSL d



This instruction performs a logical OR function between four consecutive PP memory (PPM) words and one CM word. The original CM word contents replace the four PPM words; the logical OR result replaces the original CM word. Refer to the instructions 1060 and 1062 for packing and unpacking of the words. R+A specifies the CM word address (see figure II-1-7); d is the first PPM word address. The IOU verifies each CM address against the OS bounds address.

The instruction does not start execution until the IOU completes all previous CM accesses. The CM does not permit any other port to access the CM word from the start of the read until the end of the write. The instruction delays subsequent instruction execution by the IOU until all CM accesses for the instruction complete.

1001d Central Read and Clear Lock from d to (A) RDCL d



This instruction performs a logical AND function between four consecutive PP memory (PPM) words and one CM word. The original CM word contents replace the four PPM words; the logical AND result replaces the original CM word. See the 1060 and 1062 instructions for packing and unpacking of the words. R+A specifies the CM word address (see figure II-1-7); d is the first PPM word address. The IOU verifies each CM address against the OS bounds address. The instruction does not start execution until all previous CM accesses by the IOU complete. The CM does not permit any other port to access the CM word from the start of the read until the end of the write. The instruction delays execution of subsequent instructions by the IOU until all CM accesses for the instruction complete.

PP INPUT/OUTPUT INSTRUCTIONS

The 26 instructions (in table II-1-28) direct activity on the I/O channels. They select an external device and transfer data to or from that device. The instructions also determine whether a channel or external device is available and ready to transfer data. The preparatory steps ensure that the channels carry out an orderly data transfer. Each external device has a set of external function codes the PP uses to establish operation modes, and to start and stop data transfer. The devices can also detect certain errors which are indicated to the controlling PP.

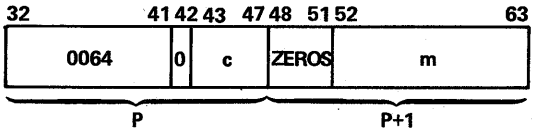
Table II-1-28. PP Input/Output Instructions (Sheet 1 of 2)

Opcode	Instruction
00640	Jump to m if channel c active
00641	Test and set channel c flag
1064X	Jump to m if channel c flag set
00650	Jump to m if channel c inactive
00651	Clear channel c flag
1065X	Jump to m if channel c flag clear
00660	Jump to m if channel c full
00661	Test and clear channel c error flag set
00670	Jump to m if channel c empty
00671	Test and clear channel c error flag clear
00700	Input to A from channel c when active
00701	Input to A from channel c if active
00710	Input A words to m from channel c
10710	Input A words to m from channel c packed
00720	Output from A to channel c when active
00721	Output from A on channel c if active
00730	Output A words from m on channel c
10730	Output A words from m on channel c packed
00740	Activate channel c
00741	Unconditionally activate channel c
00750	Deactivate channel c

Table II-1-28. PP Input/Output Instructions (Sheet 2 of 2)

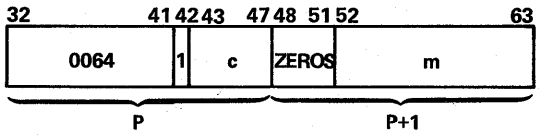
Opcode	Instruction
00751	Unconditionally deactivate channel c
00760	Function A on channel c when inactive
00761	Function A on channel c if inactive
00770	Function m on channel c when inactive
00771	Function m on channel c if inactive

00640cm Jump to m if Channel c Active AJM m,c



This instruction branches to the location specified by m if channel c is active.

00641cm Test and Set Channel c Flag SCF m,40B+c)

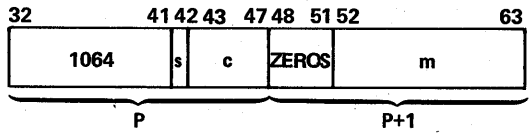


This instruction branches to the location specified by m if the channel c flag is set. Otherwise, it sets the channel flag and exits. Setting m to P+2 unconditionally sets the channel flag.

NOTE

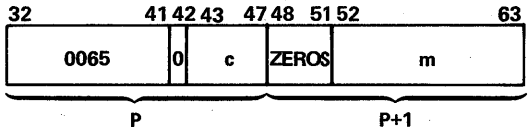
A conflict condition may occur when two or more PPs in the same time slot attempt to simultaneously execute a 00641 instruction on the same channel. Only the maintenance channel (17_8) resolves this condition by letting the PP in the lowest physical barrel see the true status of the flag. The flag appears set to the other PPs in conflict and the PPs take the branch.

1064Xcm Jump to m if Channel c Flag Set FSJM m,c



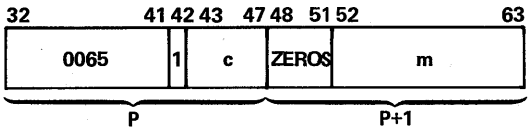
This instruction branches to the location specified by m if the channel c flag is set.

00650cm Jump to m if Channel c Inactive IJM m,c



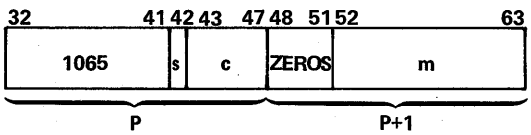
This instruction branches to the location specified by m if channel c is inactive.

00651cm Clear Channel c Flag CCF c



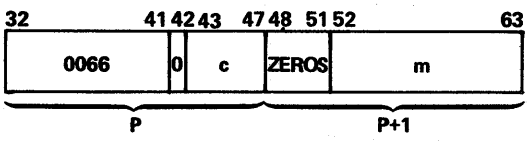
This instruction clears the channel c flag, and requires but does not use the m field.

1065Xcm Jump to m if Channel c Flag Clear FCJM m,c



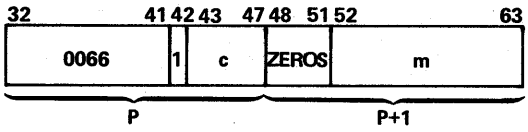
This instruction branches to the location specified by m if the channel c flag is clear.

00660cm Jump to m if Channel c Full FJM m,c



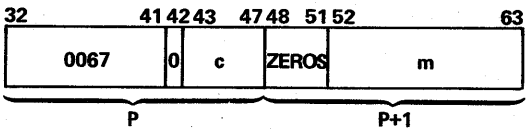
This instruction branches to the location specified by m if channel c is full.

00661cm Test and Clear Channel c Error Flag Set SFM m,40B+c



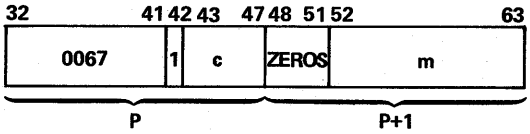
If the channel c error flag is set, this instruction branches to the location specified by m and clears the error flag.

00670cm Jump to m if Channel c Empty EJM m,c



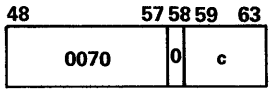
This instruction branches to the location specified by m if channel c is empty.

00671cm Test and Clear Channel c Error Flag Clear CFM m,40B+c



If the channel c error flag is clear, this instruction branches to the location specified by m. If the error flag is set, this instruction clears it.

00700c Input to A from Channel c When Active IAN c

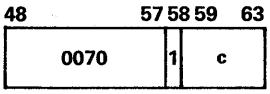


This instruction transfers one word from channel c to the low-order 16 bits of A. The high-order 2 bits of A are zeros. The instruction waits for the channel to become active and full.

NOTE

If the channel uses a 12-bit external interface, the high-order 6 bits of A are 0s. If it uses an 8-bit external interface, the high-order 10 bits of A are 0s.

00701c Input to A from Channel c if Active IAN 40B+c

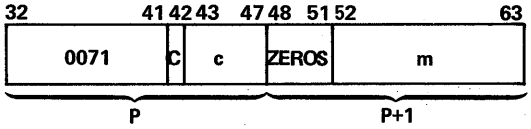


This instruction transfers one channel c word to the low-order 16 bits of A. The high-order 2 bits of A are zeros. If the channel is inactive or becomes inactive before becoming full, no transfer occurs and the instruction exits with A = 0.

NOTE

If the channel uses a 12-bit external interface, the high-order 6 bits of A are 0s. If it uses an 8-bit external interface, the high-order 10 bits of A are 0s. If the addressed channel is not connected, the instruction exits with A = 177777_8.

0071Ccm Input A Words to m from Channel c IAM m,c



This instruction transfers successive words from channel c to PP memory (PPM). The m field specifies the first PPM word address; (A) specifies the number of words transferred.

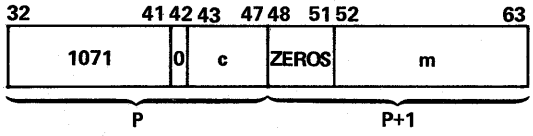
The transfer completes when either A = 0 or the channel becomes inactive. If an inactive channel caused termination, the instruction clears the next PPM word, and A contains the difference between its initial value and the number of channel words actually transferred.

No transfer takes place if the instruction executes with the channel initially inactive. The instruction exits with A unchanged and the PPM word specified by m sets to 0. However, if the addressed channel is not connected, the instruction exits with A = 177777_8.

NOTE

If the channel uses a 12-bit external interface, the high-order 4 bits of the PP memory word are 0s. If it uses an 8-bit external interface, the high-order 8 bits of the PP memory word are 0s.

10710cm Input A Words to m from Channel c Packed — IAPM m,c



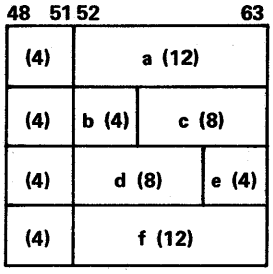
This instruction transfers the low-order 12 bits of successive channel c words to consecutive PP memory (PPM) words. During this transfer, the IOU packs four 12-bit words (48 bits) into three PPM words. (Refer to the following paragraphs.) The high-order 4 bits of the channel words are ignored. The m field specifies the first PPM word address; A specifies the number of channel words transferred.

A complete transfer depends on the channel word count being a multiple of 4. If the channel word count is not a multiple of 4, the IOU fills the bits left over when A is counted to zero (these bits copy the corresponding bits on the channel). When the channel is inactive or empty, these bits are zeros and, hence, the fill is with zeros. When, however, the external device and the PP have different word counts, or for some other reason the channel bits are nonzero, the fill is not zero-fill.

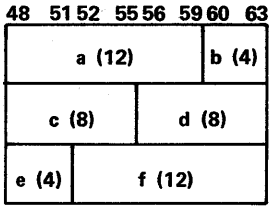
The instruction exits when A is zero or when the channel becomes inactive. If an inactive channel causes termination, the leftover bits from the previous channel word will fill up to the next four-channel-word boundary as described above.

No transfer takes place if the instruction executes with the channel initially inactive. The instruction exits with A unchanged, and the next three PPM words specified by m, m+1, and m+2 fill as described above.

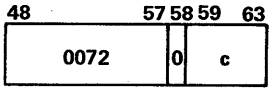
This instruction allows 16-bit PPM words to be read from 12-bit external devices. The channel words are as follows:



The PPM words are as follows.



00720c Output from A on Channel c when Active OAN c

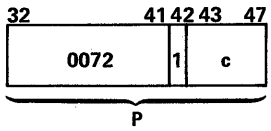


This instruction transfers one word from the A register low-order 12 bits to channel c. The instruction waits for an active and empty channel before executing.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits.

00721c Output from A on Channel c if Active OAN 40B+c

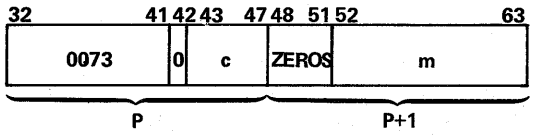


This instruction transfers the A register low-order 16 bits to channel c. If the channel is inactive, no transfer occurs and the instruction exits. The operation does not alter the contents of A.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits.

00730cm Output A words from m on Channel c OAM m,c

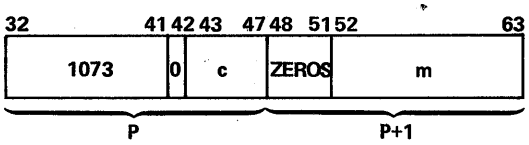


This instruction transfers the contents of successive PP memory (PPM) words as successive words on channel c. The m field specifies the first PPM word address; A specifies the number of words to be transferred. The transfer completes when either A = 0 or the channel becomes inactive. If an inactive channel caused termination, A contains the difference between its initial value and the number of words transferred on the channel. If the instruction executes with the channel initially inactive, no transfer occurs and the instruction exits with A unchanged.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits.

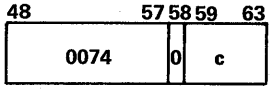
10730cm Output A Words from m on Channel c Packed — OAPM m,c



This instruction transfers consecutive PP memory (PPM) words as the low-order 12 bits of successive words on channel c. During the transfer, processing occurs such that the contents of three PPM words result in four channel words. The high-order 4 bits of the 16-bit channel words set to zeros. This packing is illustrated in the 1071 instruction. The m field specifies the first PP word address; A specifies the number of channel words to be transferred. The transfer completes when either A = 0 or the channel becomes inactive. If an inactive channel caused termination, A contains the difference between its initial value and the number of words actually transferred on the channel.

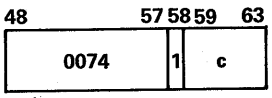
If the instruction executes with the channel initially inactive, no transfer occurs and the instruction exits with A unchanged.

00740c Activate channel c — ACN c



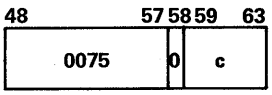
This instruction sets channel c active to prepare it for I/O transfer operations. If the channel is initially active, the instruction waits for the channel to become inactive before executing.

00741c Unconditionally Activate Channel c — ACN 40B+c



This instruction sets channel c active to prepare it for I/O transfer operations. The instruction executes regardless of the channel's active/inactive status.

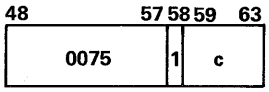
00750c Deactivate channel c — DCN c



This instruction sets channel c inactive to terminate I/O operations on the channel. If the channel is initially inactive, the instruction waits for the channel to become active before executing.

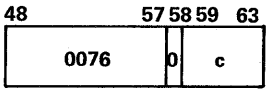
If this instruction executes after an output instruction without waiting for the channel to become empty, the last channel word transferred may be lost.

00751c Unconditionally Deactivate Channel c DCN 40B+c



This instruction sets channel c inactive to terminate I/O operations on the channel. If the channel is initially inactive, the instruction executes regardless of the channel's active/inactive state. If this instruction executes after an output instruction without waiting for the channel to become empty, the last channel word transferred may be lost.

00760c Function A on Channel c when Inactive FAN c

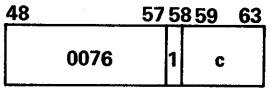


This instruction transfers the low-order 16 bits of A to channel c as a function code. If the channel is initially active, the instruction waits for the channel to become inactive before executing. The operation does not alter the contents of A.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits. Parity, however, is always calculated on the rightmost 16 bits of A when outputting a function word from A.

00761c Function A on Channel c if Inactive FAN 40B+c

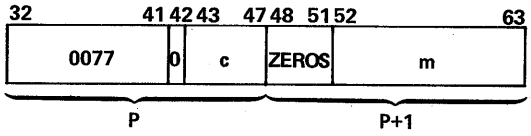


This instruction transfers the low-order 16 bits of A to channel c as a function code. If the channel is initially active, the IOU does not transfer the function on the channel, and the instruction exits.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits.

00770cm Function m on Channel c when Inactive FNC m,c

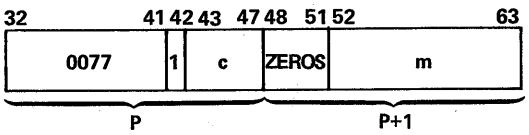


This instruction transfers the m field contents to channel c as a function code. If the channel is initially active, the instruction waits for the channel to become inactive before executing.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits.

00771cm Function m on Channel c if Inactive FNC m,40B+c



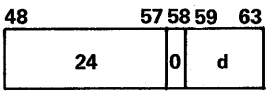
This instruction transfers the m field contents to channel c as a function code. If the channel is initially active, the IOU does not transfer the function on the channel and the instruction exits.

NOTE

If the channel uses a 12-bit interface, it does not transmit the channel word high-order 4 bits to the external device. Similarly, if it uses an 8-bit external interface, it does not transmit the channel word high-order 8 bits.

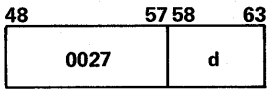
OTHER IOU INSTRUCTIONS

002400 Pass PSN
000000 Pass
002500 Pass



The pass instructions perform no operation.

0027d PP Keypoint KPT d



This instruction executes as a pass instruction, but allows test-point sensing of its execution by way of external monitoring equipment.

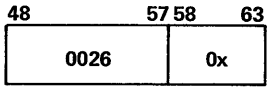
Exchange Jumps

The exchange jump instructions allow PP programs to request CYBER 170 State exchanges in the CP. The IOU transmits the exchange request to the CP designated for executing CYBER 170 State instructions. Bit 05 of the monitor condition register sets to indicate an outstanding IOU exchange request.

If an exchange request for any PP is outstanding, another exchange request from any other PP causes the second PP to wait until the outstanding exchange request completes.

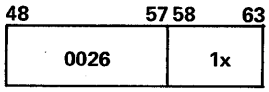
The d field value controls the action taken to process the exchange request in CYBER 170 State, as described in the following paragraphs.

00260x Exchange Jump EXN d



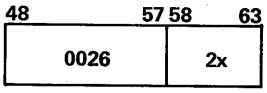
This is an unconditional exchange jump performed with the exchange package at address R+A (see figure II-1-4). The IOU verifies this address against the OS bounds address and, if in the prohibited region, the exchange does not occur. In this case, the OS bounds fault sets and, if the enable error stop bit is set in the environment control register, the PP is idled.

00261x Monitor Exchange Jump MXN d



This is a conditional exchange jump performed with the exchange package at address R+A (see figure II-1-7). The IOU verifies this address against the OS bounds address and, if in the prohibited region, the exchange does not occur. The OS bounds fault sets and, if the enable error stop is set in the environment control register, the PP is idled. If monitor flag is clear, the exchange jump occurs and the monitor flag sets. If the monitor flag is set before this instruction begins to execute, the exchange jump is not performed.

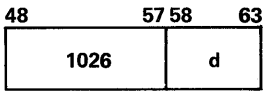
00262x Monitor Exchange Jump to MA MAN d



This is a conditional exchange jump performed with the exchange package at the address in the CP monitor address register. If the monitor flag is clear, the exchange jump occurs and the monitor flag sets. If the monitor flag is set before this instruction begins to execute, the exchange jump is not performed.

00263x Instruction executes as if d = 2x.

1026d Interrupt Processor INPN d



This instruction causes the IOU to transmit an interrupt for a CP on the memory port specified by d. This interrupt signal causes the external interrupt bit to set in the monitor condition register. Refer to CP Interrupts in section 2 of this volume.

PROGRAMMING INFORMATION 2

This section contains programming information for the CP, CM, IOU, system console, and two-port multiplexer.

CP EXCHANGE OPERATIONS

Figure II-2-1 shows CP modes of operation. Exchange operations switch the CP between monitor and job modes in both Virtual State and CYBER 170 State. Exchange operations may also switch states while switching modes. Refer to Interstate Programming Information in this section for a description of state-switching operations.

An exchange operation exchanges the process running in CP with another process and switches CP modes. The exchange stores the CP registers of the outgoing process in CM as an exchange package (refer to Exchange Package, figure II-2-2), and reads the registers of the incoming process from another exchange package in CM into the CP registers. Exchange operations are caused by the following:

- Execution of a Virtual State exchange instruction in the CP.
- Execution of the interrupt processor instruction (0026) in any PP.
- Hardware-detected fault or exception with the CP in Virtual State job mode. Such exchange operations are called exchange interrupts.

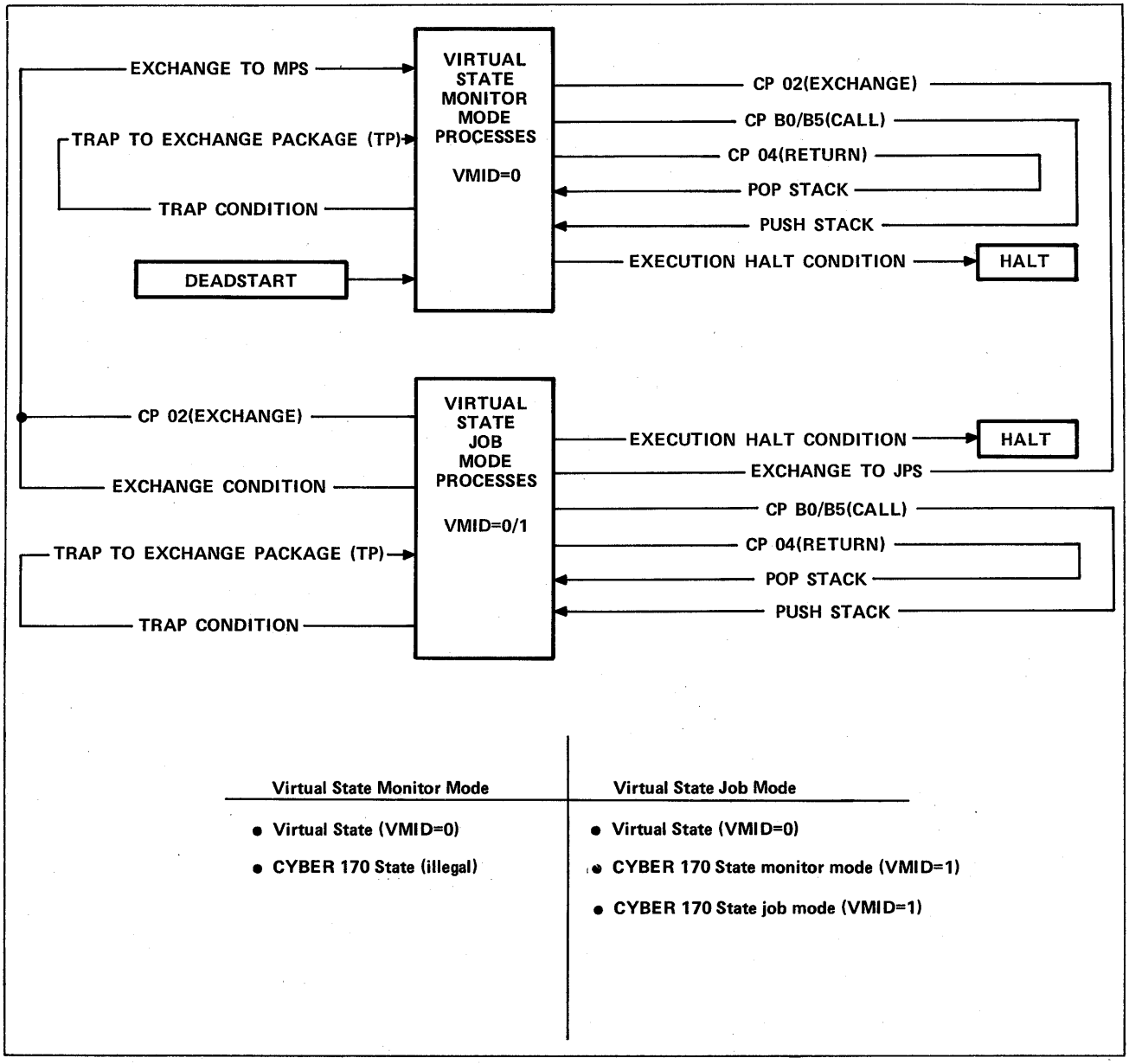


Figure II-2-1. CP Calls, Returns and Interrupts

VIRTUAL STATE JOB-TO-MONITOR EXCHANGE OPERATIONS

The CP performs an exchange from Virtual State job mode to Virtual State monitor mode as follows:

1. It stores the outgoing job process registers as an exchange package starting at the CM address in the job process state (JPS) pointer register.

2. It disables exchange interrupts.

3. It loads the incoming monitor process registers (from another exchange package in CM) into the CP registers, starting at the CM address in the monitor process state (MPS) pointer register.

Exchange interrupt conditions occurring with the CP in monitor mode do not cause an exchange interrupt, but may cause a trap interrupt. Refer to CP Interrupts in this section.

VIRTUAL STATE MONITOR-TO-JOB EXCHANGE OPERATIONS

The CP performs an exchange from Virtual State monitor mode to Virtual State job mode as follows:

1. It stores the outgoing monitor process registers as an exchange package starting at the CM address in the monitor process state (MPS) pointer register.

2. It enables exchange interrupts.

3. It loads the incoming job process registers (stored in CM as another exchange package into the CP registers) starting at the CM address in the job process state pointer register.

When a Virtual State monitor-to-job mode exchange operation sets MCR bit 55 (environment specification error), the CP completes the exchange and initiates a job-to-monitor mode (JPS) exchange in response to the error. Refer to Interrupts in this section.

EXCHANGE PACKAGES

Before initiating a Virtual State monitor-to-job exchange, the operating system specifies the process environment by composing in CM an exchange package for that process. When the process is suspended, hardware records system conditions into the same exchange package in CM, permitting process reactivation (in the absence of uncorrectable errors) without violating process integrity.

Each suspended process (including the monitor program) has one exchange package stored in CM. Each exchange package contains the process registers in fifty-two 64-bit words as shown in figure II-2-2. Refer to Process State Registers in this section for descriptions of the exchange package entries.

The CP uses the following types of exchange packages:

- For exchanges between Virtual State monitor and job modes (exclusively within Virtual State), the exchange package format is shown in figure II-2-2.
- For exchanges between Virtual State monitor and job modes (including switching between Virtual State and CYBER 170 State), the exchange package format is shown in figure II-2-27.
- For exchanges with CYBER 170 State monitor and job modes, the exchange package format is shown in figure II-2-28.
- For exchanges exclusively within CYBER 170 State, the exchange package format is shown in the appropriate CYBER 170 State hardware reference manual listed in the preface.

Exchange package addresses, used by hardware to locate exchange packages during exchange operations, are real memory addresses (RMAs) in hardware registers designated as follows:

- For exchanges to Virtual State monitor mode, the RMA is in the monitor process state (MPS) pointer register.
- For exchanges to Virtual State job mode, the RMA is in the job process state (JPS) pointer register.

Exchange operations do not copy an exchange package into cache memory.

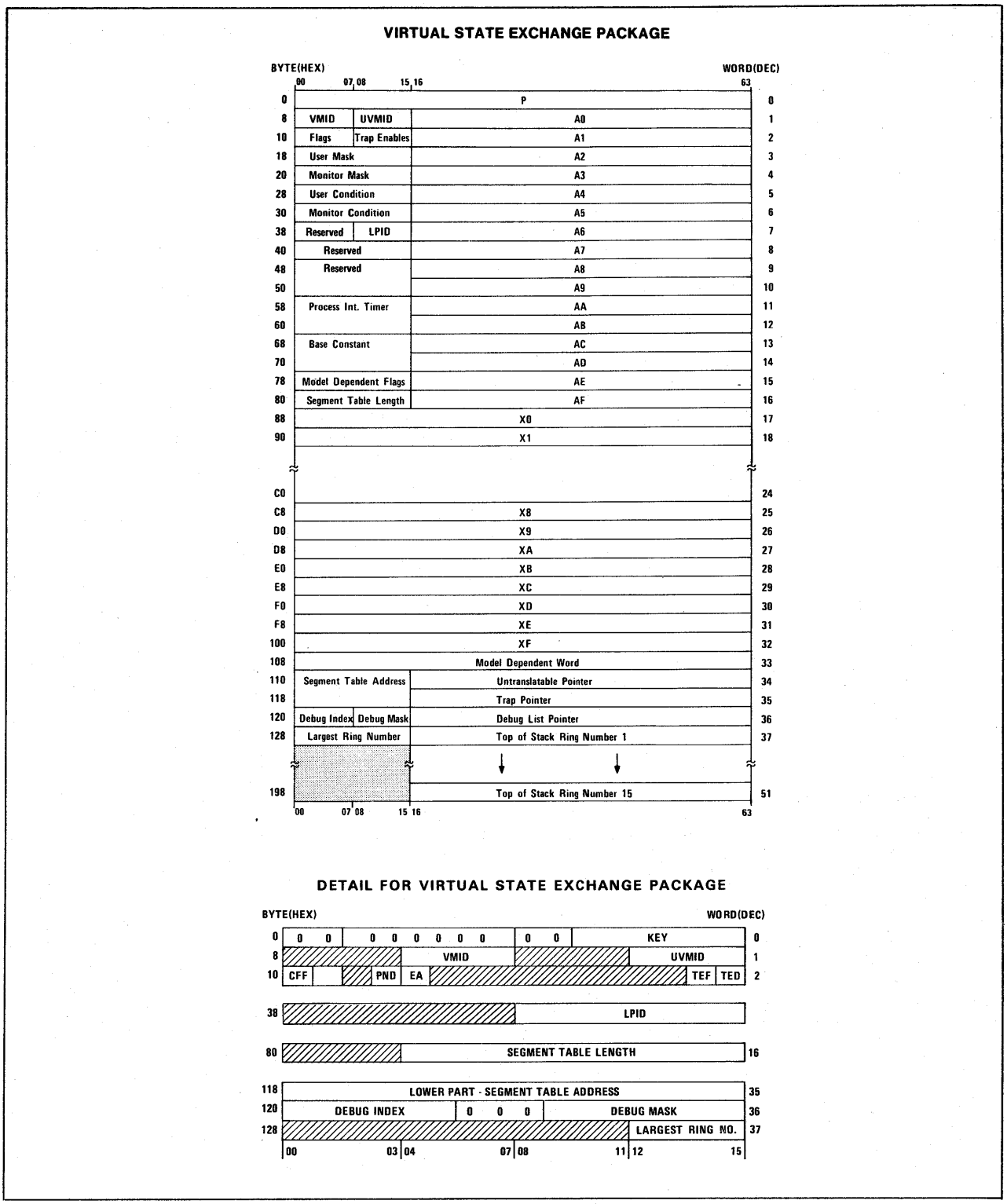


Figure II-2-2. Virtual State Exchange Package

CP REGISTERS

The CP registers comprise two classes: process state registers and processor state registers. This distinction arises because the state of the process, and the state of the processor, characterize CP operation. Both classes of registers are accessible to the CP and PPs. Bits within the registers number consecutively from left to right, with the rightmost bit always equal to 63.

PROCESS STATE REGISTERS

The process state registers relate to a specific Virtual State process executing in the CP. Various process state registers also support CYBER 170 State operation. The contents of the process state registers can be written into memory as a Virtual State exchange package for either a Virtual State process or a CYBER 170 State process (refer to figures II-2-2 and II-2-27).

The exchange package for each process contains the step-by-step operating register contents as directed by process execution. In addition, the exchange package holds other detailed process state information such that the CP may dynamically switch between exchange packages (processes) while preserving process integrity. When a process executes in the CP, its exchange package resides in the process state registers. When a process awaits execution, its exchange package resides in central memory.

Table II-2-1 lists the processor state registers and permissible access types for CP copy-to/from-state-register instructions and for maintenance channel (MCH) access.

CP Base Constant (BC) Register

The 32-bit BC register provides a means for communication with the operating system. The contents of this register do not directly affect hardware operation.

CP Debug Index (DI) Register

The 6-bit DI register is added to the debug list pointer (DLP) register to form the PVA of each word read from the debug list (DL). DI increments after each word is read. When a DL match occurs, DI+DLP points to the second word of the matched DL entry.

CP Debug List Pointer (DLP) Register

The 48-bit DLP register contains the PVA of the first debug list entry. DLP bits 61 through 63 must be zeros or an address specification error (MCR 52) occurs.

Table II-2-1. Process State Registers

Register Name	No. of Bits	Address	Access Type	
			Copy	MCH
Address (A0 through AF) (16 registers)	48	-	-	-
Base constant (BC)	32	47	R	R/W
Debug index (DI)	6	E4	R/W	R/W
Debug list pointer (DLP)	48	C5	R/L	R/W
Debug mask (DM)	7	E5	R/W	R/W
Flags				
Critical-frame flag (CFF)	1	E0,E1	R/W	R/W
On-condition flag (OCF)	1	E2,E3	R/W	R/W
Process-not-damaged (PND) flag	1	-	-	-
Largest ring number (LRN)	4	-	-	CM
Last processor identification (LPID)	8	-	-	-
Monitor condition (MCR)	16	43	R	R/W
Monitor mask (MMR)	16	60	R/W	R/W
Operand (X0 through XF) (16 registers)	64	-	-	-
Process interval timer (PIT)	32	C9	R/L	R/W
Program address (P)	64	40	R	R/W
Segment table address (STA)	32	45	R	R/W
Segment table length (STL)	12	45	R	R/W
Top-of-stack (TOS) pointer (15 registers)	48	-	-	CM
Trap enable (TE)	2	C0-C3	R/L	R/W
Trap pointer (TP)	48	C4	R/L	R/W
Untranslatable pointer (UTP)	48	44	R	R/W
Untranslatable virtual machine identifier (UVMID)	4	-	-	-
User condition (UCR)	16	43	R	R/W
User mask (UMR)	16	E6	R/W	R/W
Virtual machine identifier (VMID)	4	-	-	-

Notes: R: Unprivileged read
W: Unprivileged write
G: Globally-privileged write
CM: In central memory (indirectly accessible)

CP Debug Mask (DM) Register

The 7-bit DM register contains 2 flag bits and 5 mask bits. The flag bits control debug initiation and termination. A mask bit, when set, enables the corresponding debug interrupt. DM has the following bit assignments:

DM Bit	Description
9	End-of-list-seen flag
10	Debug scan-in-progress flag
11	Data-read mask
12	Data-write mask
13	Instruction-fetch mask
14	Branch target instruction-fetch mask
15	Call target instruction-fetch mask

CP Flag Register

The 4-bit flag register contains the flag bits described in the following paragraphs.

Critical-Frame Flag (CFF)

Software sets this flag to indicate that the stack frame in use (when the flag is set) requires special attention before this frame may be abandoned. Executing return/pop instructions with CFF set causes an interrupt (other enables-permitting). Call instructions and trap interrupts record CFF in the stack frame save area (SFSA) and proceed to clear CFF; return instructions restore the previous CFF condition.

On-Condition Flag (OCF)

Software sets this flag to assist the operating system in the handling of certain trap interrupts. Call instructions and trap interrupts record OCF in the SFSA and proceed to clear OCF; return instructions restore the previous OCF condition.

Process-Not-Damaged (PND) Flag

Hardware sets this flag in the outgoing exchange package after an uncorrectable error (MCR 48) occurs. PND indicates that the interrupted process is undamaged and may be restarted. Hardware ignores PND in an incoming exchange package.

CP Largest Ring Number (LRN) Register

The 4-bit LRN register contains the largest ring number for which there is a top-of-stack (TOS) entry in the associated TOS register. (In model 835, this register is not used as the TOS pointers are kept in CM.)

CP Last Processor Identification (LPID) Register

In dual-CP systems, the 8-bit LPID register identifies the last CP which executed the process defined by the exchange package.

CP Monitor Condition Register (MCR)

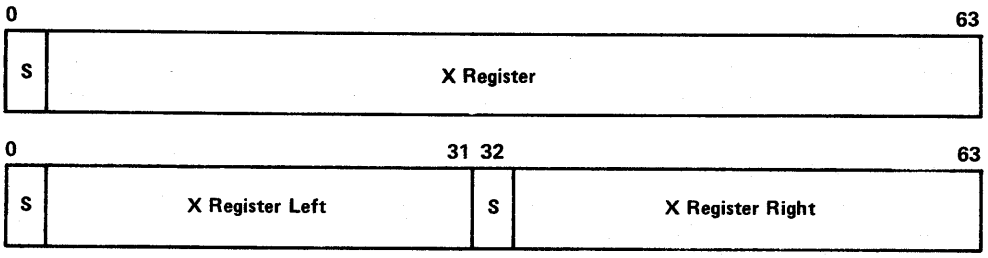
The 16-bit MCR register records system exception conditions which the operating system must resolve (for example, hardware errors, instruction specification errors, and access violations). Refer to CP Interrupts in this section for further information.

CP Monitor Mask Register

The 16-bit MMR register enables or masks certain software-specified conditions directly associated with the monitor condition register (MCR). An interrupt occurs when an MCR bit is set with the corresponding MMR bit set (other enables-permitting). Refer to CP Interrupts in this section for further information.

Operand X Registers

The sixteen 64-bit X registers, numbered X0 through XF, supply operands for arithmetic operations and data manipulation. Depending on the operation, the registers contain a logical quantity, a signed binary integer, or a signed FP number. CP instructions which only require 32 data bits access the X registers as X-left (bits 0 through 31) or X right (bits 32 through 63). The X-register formats are as follows (the S-field is the sign bit):



Store operations to X-left (XkL) do not alter X-right (XkR). Store operations to XkR do not alter XkL.

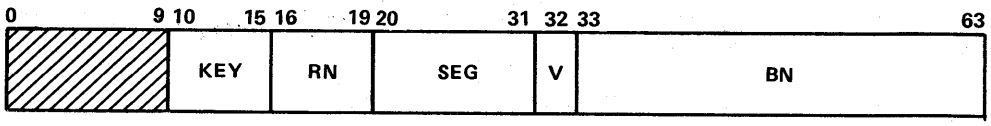
CP Process Interval Timer (PIT)

The 32-bit PIT register allows each process to track its own execution time. PIT is set either by a Copy-To-State-Register instruction with local privilege or from an incoming exchange package. The CP records PIT in an outgoing exchange package. PIT can be read via a Copy-From-State-Register instruction.

PIT decrements at a 1-microsecond rate without stopping. A trap interrupt occurs whenever the count equals zero (other enables-permitting).

CP Program Address (P) Register

The 64-bit P register contains the PVA of an instruction during the time the CP interprets and executes it. The P register also contains bits which define memory access protection. The P-register format is as follows:



Field	Name	Description
KEY	Key	Access permission attribute (refer to Access Protection under Virtual Memory Programming in this section).
RN	Ring Number	Access privilege indicator (refer to Access Protection under Virtual Memory Programming in this section).
SEG	Segment Number	Process segment number (refer to Access Protection under Virtual Memory Programming in this section).
V	Valid bit	Validity indicator (refer to Process Virtual Memory under Virtual Memory Programming in this section.

Field	Name	Description
BN	Byte Number	Byte displacement within the 231 bytes in a segment. Bit 32 in the final PVA used as a validity indicator and must be a zero or the PVA is rejected. Bit 32 in an A register may be a one, provided indexing or displacement changes this bit to a zero in the final PVA.

CP Segment Table Address (STA) Register

The 32-bit STA register contains the real memory address (RMA) of the first segment descriptor table entry (interpreted as a byte address). Hardware ignores the rightmost 3 bits of STA. Refer to Virtual Memory Programming in this section for further information.

CP Segment Table Length (STL) Register

The 12-bit STL register contains a count equal to one less than the number of 64-bit entries in the associated segment descriptor table. The virtual addressing mechanism uses this count to verify that segment table references are to an address within the segment table boundaries. Refer to Virtual Memory Programming in this section for further information.

CP Top-of-Stack (TOS) Pointer Register

The operating system has for each process fifteen 48-bit TOS pointers to guarantee access protection. The TOS pointers are located in words 37 through 51 of an exchange package in CM. Each TOS is associated with one of the 15 rings of access protection; hardware uses the pointers to switch stacks during call/return instructions. Each TOS is a PVA pointing to the top of its associated stack when this stack is not in active use. Refer to Stack Manipulating Operations in this section, and Access Protection under Virtual Memory Programming in this section, for further information.

CP Trap Enable (TE) Register

The 2-bit TE register contains information that determines how traps are enabled. The TE register bits are represented as follows:

- Trap-enable flip-flop (TEF)

  When set, TEF is one of the conditions which enable trap interrupts. TEF is normally set by a Copy-To-State-Register instruction and cleared by hardware when a trap interrupt occurs. TEF can also be cleared by another Copy-To-State-Register instruction.

- Trap-enable delay (TED) flip-flop

  The TED flip-flop disables a trap interrupt until after the next return instruction completes execution. TED is normally set by a Copy-To-State-Register instruction and cleared by a return instruction. TED can also be cleared by another Copy-To-State-Register instruction.

CP Trap Pointer (TP) Register

The 48-bit TP register contains a PVA which is the indirect address of the entry point into the trap interrupt target procedure (refer to CP Interrupts in this section). The code base pointer (CBP) is the direct address to which TP points. An incoming exchange package loads the TP register. TP can also be written by a Copy-To-State-Register instruction with local privilege, and read by a Copy-From-State-Register instruction.

CP Untranslatable Pointer (UTP) Register

When an interrupt occurs because the CP cannot translate a PVA or SVA to an RMA, the 48-bit UTP register contains the untranslatable PVA or SVA. UTP is set to PVA when the following MCR conditions occur:

- MCR 52 sets (except during load-page or purge instructions).
- MCR 54 or 57 sets.
- MCR 60 sets due to invalid SDE or exceeded STL.
- MCR 60 sets due to code base pointer (CBP) RN-field of zero during a call-indirect instruction.

In the following cases, UTP may be set to either PVA or SVA:

- MCR 52 sets during load-page instruction.
- MCR 52 sets during purge instruction with k=0/1/8/9.

No other interrupt alters the UTP register.

CP Untranslatable Virtual Machine Identifier (UVMID) Register

Hardware sets the 4-bit UVMID register when an exchange operation, call-indirect instruction, or return instruction is interrupted due to an attempt to switch the CP to a state for which there is no set bit in the virtual machine capability list. In such a case, hardware sets the UVMID code to indicate which VMCL bit is missing. Values of 0 through 15 of UVMID correspond to VMCL bits 48 through 63 as follows:

UVMID	Missing VMCL Bit
0	48 (Virtual State)
1	49 (CYBER 170 State)
2 through FF	50 through 63 (reserved)

CP User Condition Register (UCR)

The 16-bit UCR register records conditions that normally do not require an exchange to monitor mode for corrective action. Each bit indicates detection of a particular error or exception condition in the CP (for example, divide fault, arithmetic overflow and underflow, or invalid BDP data). A trap interrupt occurs when a UCR bit sets with the trap enable flip-flop set, the corresponding user mask register (UMR) bit set, and trap-enable delay flip-flop clear. Refer to CP Interrupts in this section for further information.

CP User Mask Register (UMR)

The 16-bit UMR register enables or masks certain software-specified conditions directly associated with the user condition register (UCR). An interrupt occurs when a UCR bit is set with the corresponding UMR bit set (other enables-permitting). Refer to CP Interrupts in this section for further information.

CP Virtual Machine Identifier (VMID) Register

The 4-bit binary code in the VMID register identifies the virtual machine capability being used, as follows:

VMID	Virtual Machine Capability
0	Virtual State
1	CYBER 170 State
2 through FF	Reserved

CP PROCESSOR STATE REGISTERS

The processor state registers contain information about the state of the CP hardware, rather than a unique process. Included among this category of registers are the CP maintenance registers, which provide additional information about the condition of the CP hardware for diagnostic purposes. Other processor state registers contain such information as parameters of tables and pointers to exchange packages in CM.

Table II-2-2 lists the processor state registers and permissible access types for CP Copy-To/From-State-Register instructions and for maintenance channel (MCH) access.

Table II-2-2. Processor State Registers

Register Name	No. of Bits	Address	Access Type	
			Copy	MCH
Cache/map corrected error log (CCEL/MCEL)	64	92/93	R/G	R/W
Dependent environment control (DEC)	64	30	-	R/W
Element identifier (EID)	32	10	R	R
Job process state (JPS)	32	61	R/M	R/W
Model-dependent word (MDW)	64	51	R	R/W
Monitor process state (MPS)	32	41	R	R/W
Options installed (OI)	64	12	R	R
Page size mask (PSM)	7	4A	R	R/W
Page table address (PTA)	32	48	R	R/W
Page table length (PTL)	8	49	R	R/W
Processor fault status (PFS)	64	80-81	R/G	R/W
Processor identifier (PID)	8	11	R	R
Processor test mode (PTM)	64	A0	R/G	R/W
Status summary (SS)	6	00	-	R
System interval timer (SIT)	32	62	R/M	R/W
Virtual machine capability list (VMCL)	16	13	R	R

Notes: R: Unprivileged read
W: Unprivileged write
G: Globally-privileged write
M: Virtual State monitor mode write

CP Options Installed (OI) Register

The 64-bit OI register is a hard-wired register identifying the options installed in the CP. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the OI register.

CP Page Size Mask (PSM) Register

The 7-bit PSM register specifies the page size used in allocating real memory. Page sizes are selectable at system initialization time and range from 2K to 16K bytes. Refer to Virtual Memory Programming in this section for further information.

CP Page Table Address (PTA) Register

The 32-bit PTA rgister is a real-memory byte address pointing to the first page table entry. Refer to Virtual Memory Programming in this section for further information.

CP Page Table Length (PTL) Register

The 8-bit PTL register is a mask specifying the page table length. The page table ranges from 512K to 131K words in 512K-word increments. Refer to Virtual Memory Programming in this section for further information.

CP Processor Fault Status (PFS) Registers

The PFS registers record hardware-detected errors occurring within the CP. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the PFS registers.

CP Processor Identifier (PID) Register

The 8-bit PID register is a hard-wired register identifying each processor in the system. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the PID register.

CP Processor Test Mode (PTM) Register

The 64-bit PTM register provides a maintenance capability which forces faults for the purpose of testing hardware-fault sensing within the CP. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the PTM register.

CP Status Summary (SS) Register

The 6-bit SS register indicates CP status (similar registers exist in the CM and IOU). Aside from the Virtual State monitor mode bit, if any SS bit is set the SS bit also sets in the IOU status summary register. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

CP Cache/Map Corrected Error Log (CCEL/MCEL) Register

The CCEL/MCEL register contains model-dependent diagnostic information. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

CP Dependent Environment Control (DEC) Register

The 64-bit DEC register is a maintenance register which controls CP operating conditions. The model-independent bit is as follows:

Bit	Description
35	Disable corrected error to status summary. When set, disables the setting of corrected error (bit 62) in the CP status summary register.

Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent bit descriptions.

CP Element Identifier (EID) Register

The 32-bit EID register is a backpanel-wired register identifying each system hardware element. The EID bits are represented as follows:

EID Bits	Description
32 through 39	Element type
40 through 47	Model number
48 through 63	Serial number (hexadecimal)

Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the EID register.

CP Job Process State (JPS) Register

The 32-bit JPS register contains the real memory address (RMA) of the first word of a Virtual State job mode exchange package. Hardware aligns the JPS address with bits 32 through 63 of RMAs, and interprets the JPS address as zero, modulo 16. The CP ignores bit 32 and interprets bits 60 through 63 as zeros. System deadstart procedures load the initial JPS.

CP Model Dependent Word (MDW) Register

The MDW register (models 810, 815, 825, 830, and 835 only) contains the PVA of the instruction that caused the last branch to take place in the CP. The PVAs include those of call and return instructions.

CP Monitor Process (MPS) Register

The 32-bit MPS register contains the real memory address (RMA) of the first word of a Virtual State monitor mode exchange package. Hardware aligns the MPS address with bits 32 through 63 of RMAs, and interprets the MPS address as zero, modulo 16. The CP ignores bit 32 and interprets bits 60 through 63 as zeros. System deadstart procedures load the initial MPS.

Bit	Description
58	Virtual State monitor mode.
59	Short warning. Sets to warn the system of an imminent environmental failure (for example, system power failure, local 50-Hz/60-Hz power failure, or cooling unit fault).
60	Processor halted.
61	Uncorrectable error. Sets whenever the detected uncorrectable error (DUE) bit in MCR sets.
62	Corrected error. Sets after the CP corrects an error. The dependent environment control (DEC) register can be set to disable the recording of corrected errors.
63	Long environment warning. Sets to indicate an imminent failure condition (for example, high-temperature warning, blower fault, or low-temperature fault).

CP System Interval Timer (SIT) Register

The 32-bit SIT register is a timer which establishes maximum time intervals for process execution. The operating system first sets the timer to the desired value. The timer then decrements at a 1-microsecond rate until the count equals zero (the timer does not stop counting at zero; it decrements to all ones and continues decrementing). When enabled, the zero count causes an interrupt.

CP Virtual Machine Capability List (VMCL)

The 16-bit VMCL register is a backpanel-wired register indicating CP capabilities. The VMCL format is as follows:

VMCL Bit	Capability
48	Virtual State
49	CYBER 170 State
50 through 63	Reserved

CM REGISTERS

The CM contains maintenance registers which hold memory status and error information (refer to table II-2-3). CM maintenance registers are accessible through the maintenance channel (register 80 is also accessible through the memory ports).

Table II-2-3. CM Maintenance Registers

Register Name	No of Bits	Address	Access Type	
			Copy	MCH
Corrected error log (CEL)	64	A0	-	R/W
Element identifier (EID)	32	10	-	R
Environment control (EC)	64	20	-	R/W
Free-Running counter	48	B0	R	W
Options installed (OI)	32	12	-	R
Port bounds register	64	21	-	R/W
Status summary	6	00	-	R
Uncorrectable error log (UEL) 1	64	A4	-	R/W
Uncorrectable error log (UEL) 2	64	A8	-	R/W

NOTE: The free-running counter can be read from the CP by the Copy Free Running Counter (08) instruction.

CM CORRECTED ERROR LOG (CEL) REGISTER

The 64-bit CEL register contains details concerning the first corrected error since this register was last reset. The model-independent bits are represented as follows:

Bit	Description
0	Valid bit. Indicates that the CEL contains a valid entry. When this bit is set, further correctable errors are discarded.
1	Unlogged corrected error. Indicates that a correctable error occurred but could not be logged because the CEL already contained an entry.

The CEL register contains model-dependent information regarding the address, parity, and encoded number of the memory port associated with the error. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the CEL register.

CM ELEMENT IDENTIFIER (EID) REGISTER

The 32-bit EID register is set by switches on logic panels. EID identifies each system hardware element according to the following bit assignments:

Bits	Description
32 through 39	Element type
40 through 47	Model number
48 through 63	Serial number (hexadecimal)

Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the EID register.

CM ENVIRONMENT CONTROL (EC) REGISTER

The 64-bit EC register controls CM error-checking and interleaving. Refer to the Maintenance Register Codes Booklet listed in the preface for information contained in the EC register.

CM FREE-RUNNING COUNTER REGISTER

This counter register consists of either 64 counter bits, or 48 counter bits right-justified with zero-fill in the leftmost 16 bits. The counter increments at a 1-microsecond rate. Successive reads of the free-running counter guarantee different values.

The free-running counter can be written at any time through the maintenance channel. The CP can read the counter using the Copy Free Running Counter (08) instruction.

CM OPTIONS INSTALLED (OI) REGISTER

The 64-bit OI register identifies the memory configuration and is set by field-modifiable switches on logic panels. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the OI register.

CM PORT BOUNDS REGISTER

This 64-bit register controls the range of addresses accessible during a write operation through specified ports. For ports specified by bounds-register bits, write access is limited to an area between two real memory addresses (RMAs) in this register. Refer to Maintenance Channel Programming in this section for further information.

CM STATUS SUMMARY REGISTER

This 64-bit register provides information about the CM clock, error status, and physical environment condition. Refer to the Maintenance Register Codes Booklet listed in the preface for specific information contained in the CM status summary register.

CM UNCORRECTABLE ERROR LOG (UEL) REGISTERS

The two 64-bit UEL registers contain details of the first two uncorrected CM errors which occurred since the registers were last reset. The model-independent bits are represented as follows:

Bit	Description
0	Valid bit. Indicates that the UEL contains a valid entry. When this bit is set, further uncorrectable errors are discarded.
1	Unlogged uncorrectable error. Indicates that an uncorrectable error occurred but could not be logged because the UEL already contained an entry.

The UEL registers also contain information about the source of the error. This information includes the illegal function, memory bounds fault, and multiple-bit memory error. Refer to the Maintenance Register Codes Booklet listed in the preface for model-dependent information contained in the UEL registers.

IOU REGISTERS

The IOU contains maintenance registers which hold IOU status and error information (refer to table II-2-4). IOU registers are accessible through the maintenance channel.

Table II-2-4. IOU Maintenance Registers

Register Name	No. of Bits	Address	Access Type	
			Copy	MCH
Element identifier (EID)	32	10	-	R
Environment control (EC)	32	30	-	R/W
Fault status (FS) 1	64	80	-	R/W
Fault status (FS) 2	64	81	-	R/W
Fault status mask	64	18	-	R/W
Options installed (OI)	64	12	-	R
OS bounds	64	21	-	R/W
Status summary	6	00	-	R
Test mode	16	A0	-	R/W
Uncorrectable error log (UEL) 2	64	A8	-	R/W

IOU ELEMENT IDENTIFIER (EID) REGISTER

The 32-bit EID register is a backpanel-wired register identifying each system hardware element. The EID bits are represented as follows:

Bits	Description
32 through 39	Element type
40 through 47	Model number
48 through 63	Serial number (hexadecimal)

IOU ENVIRONMENT CONTROL (EC) REGISTER

The 64-bit EC register controls timing margins, test mode and deadstart, PP memory dumps, reconfiguration, and stop-on-error conditions for the IOU. It also selects PP internal registers for reading. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

IOU FAULT STATUS (FS) REGISTERS

The 64-bit FS registers indicate the presence of uncorrectable faults in the IOU, PP memories, I/O channels, or PP hardware. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

IOU FAULT STATUS MASK REGISTER

This 64-bit register controls IOU fault reporting to the IOU fault status (FS) registers. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

IOU OPTIONS INSTALLED (OI) REGISTER

The 64-bit OI register identifies the options installed in the IOU. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

IOU OS BOUNDS REGISTER

The 64-bit operating system (OS) bounds register divides the CM into an upper and a lower region for system protection. The OS bounds register contains a bit for each PP which indicates the region in CM into which the specified PP may initiate exchange operations or writes. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

IOU STATUS SUMMARY REGISTER

The status summary register indicates errors in the CP, CM and IOU. It also provides information about the PP-halt, error status, and physical environment conditions. Refer to the Maintenance Register Codes Booklet listed in the preface for further information.

IOU TEST MODE (TM) REGISTER

The 64-bit TM register forces faults in the IOU for testing of the fault sensing logic. Bits 48 through 63 of this register serve a dual role. With the Enable Test Mode Register bit set in the EC register, these bits are used to force test conditions (refer to the Maintenance Register Codes Booklet listed in the preface for further information). When the Enable Test Mode Register bit is clear, these read/write bits can be used by software as interlock/flag status bits.

CP CONDITION AND MASK REGISTERS

The CP contains a monitor condition register (MCR) and a user condition register (UCR), each of which records interrupt causes and displays flag conditions. Each condition register has a corresponding mask register controlling the action taken when a condition register bit sets. Table II-2-5 lists the bit assignments in the monitor condition and mask registers. Table II-2-6 lists the bit assignments in the user condition and mask registers. The significance of the individual bits is further described in Interrupt Conditions in this section. Section 2 in volume 1 also describes all CP registers.

In general, MCR and UCR bits may be altered by the following:

- CP hardware indicating a processor condition or external event.
- Branch-on-condition register instruction (9F).
- PP communication over the maintenance channel.
- Software with the condition register stored in an exchange package in CM.
- Trap interrupts which clear any condition register bits for which the corresponding mask register bit is set.

Monitor condition and user mask register bits may be altered by the following:

- Copy to state register instruction (0F).
- PP communication over the maintenance channel.
- Software with the mask register stored in an exchange package in CM.

Table II-2-5. Monitor Condition/Mask Register Bit Assignments

MCR bit	Exchange package bit	Type	Description	Associated MCR Bit: Set; Condition of Traps: Enabled; CP Mode of Operation: Job	Set; Enabled; Mon	Set; Disabled; Job	Set; Disabled; Mon	Clear; Any; Any
48	0	X X	Detected uncorrectable error.	Exch	Trap	Exch	Halt	Halt
49	1	X X	Not assigned.	Exch	Trap	Exch	Halt	Halt
50	2	E N	Short warning.	Exch	Trap	Exch	Stack	Stack
51	3	S T	Instruction specification error.	Exch	Trap	Exch	Halt	Halt
52	4	S T	Address specification error.	Exch	Trap	Exch	Halt	Halt
53	5	E N	CYBER 170 State exchange request.	Exch	Trap	Exch	Stack	Stack
54	6	S T	Access violation.	Exch	Trap	Exch	Halt	Halt
55	7	S 1	Environment specification error.	Exch	Trap	Exch	Halt	Halt
56	8	E N	External interrupt.	Exch	Trap	Exch	Stack	Stack
57	9	S T	Page table search without find.	Exch	Trap	Exch	Halt	Halt
58	10	E N	System call (status bit).	None	None	None	None	None
59	11	E N	System interval timer.	Exch	Trap	Exch	Stack	Stack
60	12	S 2	Invalid segment/ ring number zero.	Exch	Trap	Exch	Halt	Halt
61	13	S T	Outward call/ Inward return.	Exch	Trap	Exch	Halt	Halt
62	14	E N	Soft error.	Exch	Trap	Exch	Stack	Stack
63	15	S T	Trap exception (status bit).	None	None	None	None	None

Notes: Refer to Interrupt Conditions in this section for bit descriptions.

Stack	Test for opportunity to trap or exchange at each instruction fetch.
X	Either condition may happen.
E	Execution completed.
S	Execution suppressed.
N	P = PVA of next instruction.
T	P = PVA of this instruction.
1	For RN = 0 on Load A, Return or Pop, 1 = N; for RN = 0 on Call or Trap, 1 = T; for invalid segment, 1 = T.
2	For exchange operations, 2 = N; Call, Return or Pop, 2 = T.

Table II-2-6. User Condition/Mask Register Bit Assignments

UCR bit	Exchange package bit	Type	Description	Associated UCR Bit: Set; Condition of Traps: Enabled; CP Mode of Operation: Job	Associated UCR Bit: Set; Condition of Traps: Enabled; CP Mode of Operation: Mon	Associated UCR Bit: Set; Condition of Traps: Disabled; CP Mode of Operation: Job	Associated UCR Bit: Set; Condition of Traps: Disabled; CP Mode of Operation: Mon	Associated UCR Bit: Clear; Condition of Traps: Any; CP Mode of Operation: Any
48	0	S T	Privileged instruction fault.	Trap	Trap	Exch	Halt	-
49	1	S T	Unimplemented instruction.	Trap	Trap	Exch	Halt	-
50	2	E N	Free flag.	Trap	Trap	Stack	Stack	-
51	3	T	Process interval timer.	Trap	Trap	Stack	Stack	-
52	4	S T	Inter-ring pop.	Trap	Trap	Exch	Halt	-
53	5	T	Critical frame flag.	Trap	Trap	Exch	Halt	-
54	6		Reserved.	Trap	Trap	Stack	Stack	-
55	7	S N	Divide fault.	Trap	Trap	Stack	Stack	Stack
56	8	S T	Debug.	Trap	Trap	Stack	Stack	Stack
57	9	S T	Arithmetic overflow.	Trap	Trap	Stack	Stack	Stack
58	10	E T	Exponent overflow.	Trap	Trap	Stack	Stack	Stack
59	11	E T	Exponent underflow.	Trap	Trap	Stack	Stack	Stack
60	12	E T	FP loss of significance.	Trap	Trap	Stack	Stack	Stack
61	13	S T	FP indefinite.	Trap	Trap	Stack	Stack	Stack
62	14	S T	Arithmetic loss of significance.	Trap	Trap	Stack	Stack	Stack
63	15	E T	Invalid BDP data.	Trap	Trap	Stack	Stack	Stack

Notes: Refer to Interrupt Conditions in this section for bit descriptions.

Stack	Test for opportunity to trap or exchange at each instruction fetch.
E	Execution of instruction completed.
S	Execution of instruction suppressed.
N	P = PVA of next instruction.
T	P = PVA of this instruction.

CP CONDITION REGISTER BIT GROUPING

Refer to Interrupt Conditions in this section for bit descriptions. The four groups of condition register bits shown in table II-2-7 are a function of the characteristics of the event detected, and of the P register PVA at time of interrupt.

The PVA from the P register stored in the exchange package during exchange interrupts [or in the stack frame save area (SFSA) during trap interrupts] points to an instruction address dependent on the condition register bit group as follows:

Group	PVA in P Stored During Interrupt
1	Points to the instruction executing when the malfunction was detected. This instruction did not necessarily initiate the activity resulting in the malfunction.
2a,2b	Points to the instruction that would have been executed if the interrupt had not occurred. After executing an exchange or return to the interrupted procedure, processing continues (from the PVA stored in the exchange package) as though the interrupt had not occurred.
3	Points to the instruction causing the interrupt.

Table II-2-7. Interrupt Condition Groups

Group	Interrupt Condition	MCR/UCR Bit	Type	Occurrence	
				CY170	Virtual
Malfunctions Not Neccessarily Related to Current Instruction					
1	Detected uncorrectable error	MCR 48	Mon	X	X
Tested Between Instructions, Not Generated by Instructions					
2a	Short warning	MCR 50	Sys	X	X
	System interval timer	MCR 51	Sys	X	X
	Soft error	MCR 52	Sys	X	X
	External interrupt	MCR 59	Sys	X	X
	Free flag	UCR 50	User	X	X
	Process interval timer	UCR 51	User	X	X
	CYBER 170 State exchange request	MCR 53	Sys	X	X
Tested Between Instructions, Generated by Instructions					
2b	System call	MCR 58	Status		X
	Exponent overflow	UCR 58	User		X
	Exponent underflow	UCR 59	User		X
	FP loss-of-significance	UCR 60	User		X
	Invalid segment/RN zero (Note 1)	MCR 60	Mon	X	X
	Environment specification error (Note 2)	MCR 55	Mon	X	X
Tested Before Execution, Generated by Instruction					
3	Instruction specification error	MCR 62	Mon		X
	Address specification error	MCR 52	Mon	X	X
	Invalid segment/RN zero (Note 1)	MCR 60	Mon	X	X
	Access violation	MCR 54	Mon	X	X
	Environment specification error (Note 2)	MCR 55	Mon	X	X
	Page table search without find	MCR 57	Mon	X	X
	Outward call/inward return	MCR 61	Mon	X	X
	Trap exception	MCR 63	Status	X	X
	Privileged instruction fault	UCR 48	Mon	X	X
	Unimplemented instruction	UCR 49	Mon	X	X
	Inter-ring pop	UCR 52	Mon		X
	Critical frame flag	UCR 53	Mon	X	X
	Divide fault	UCR 55	User		X
	Debug	UCR 56	User		X
	Arithmetic overflow	UCR 57	User		X
	FP indefinite	UCR 61	User		X
	Arithmetic loss-of-significance	UCR 62	User		X
	Invalid BDP data	UCR 63	User		X

Notes: 1 MCR 60 set by load address, return, or pop instructions when RN = 0, is in group 2b.
MCR 60, set by call or trap instructions when RN = 0, or set by an invalid segment, is in group 3.
2 MCR 55 set by exchange operations is in group 2b.
MCR 55 set by call, return, or pop instructions is in group 3.

CP INTERRUPTS

Exchange interrupts and trap interrupts comprise the CP interrupt structure. The following paragraphs describe the characteristics of the two interrupt types.

EXCHANGE INTERRUPTS

An exchange interrupt causes an exchange operation as described in Exchange Operations in this section. Exchange interrupts switch the system from Virtual State job mode (Virtual State or CYBER 170 State environment) to Virtual State monitor mode. Exchange interrupts are disabled in Virtual State monitor mode.

Exchange interrupts initiate from conditions that set a bit in the monitor condition register (and in some cases in the user condition register). Exchange interrupts can only occur when the CP is in Virtual State job mode and the corresponding mask register bit is set. Refer to tables II-2-5 and II-2-6.

TRAP INTERRUPTS

A trap interrupt acts as an implicit call indirect (B0) instruction to an interrupt-handling procedure. Trap interrupts save the current stack frame save area (SFSA) environment, push the stack, and switch control to a software procedure for trap handling. Trap interrupts occur in response to the setting of UCR or MCR bits (as shown in tables II-2-5 and II-2-6) in the following environments:

- Within Virtual State monitor mode.
- Within Virtual State job mode.
- Upon switch from CYBER 170 state to Virtual State job mode (refer to State Switching Operations in this section).

The trap interrupt creates the maximum (33 word) SFSA descriptor and preserves the contents of the following in the associated SFSA: the P register, A and X registers, VMID, SFSA descriptor, monitor mask and condition registers, and user mask and condition registers. After the trap interrupt stores these registers in the SFSA, the CP clears the UCR or MCR bits causing the trap interrupt.

The trap interrupt target address is obtained by the CP using the PVA in the trap pointer register to access a code base pointer (CBP) in a system binding section. This CBP contains the PVA of the next instruction to be executed. The external procedure flag must be set in the CBP. Refer to CP Stack Manipulating Operations in this section.

A trap interrupt disables further trap interrupts. Software may reenable traps by either setting the trap enable delay flip-flop and executing the return (04) instruction, or by setting the trap enable flip-flop. The return instruction reestablishes the suspended environment but does not load the monitor/user condition registers from the SFSA into the CP.

If an exception condition arises during execution of a trap operation, the trap interrupt aborts and the following actions occur:

1. The trap exception bit (MCR 63) sets.
2. The appropriate UCR/MCR bit sets for the condition causing the trap to abort.

3. The trap enable flip-flop condition (set) is recorded in the exchange package stored for the interrupted procedure.

4. The exchange or halt performs as indicated in tables II-2-5 and II-2-6.

Virtual State job mode processes can control trap interrupts by setting bits in the user mask register. Bits set in the user mask register permit the trap interrupt when the corresponding UCR bit sets. Trap conditions occurring when traps are disabled have effect as listed in tables II-2-5 and II-2-6.

INTERRUPT CONDITIONS

The following paragraphs present the various conditions causing system interrupts. Each condition description includes a reference to the corresponding MCR or UCR bit that sets upon condition detection. Refer to Condition and Mask Registers in this section.

Access Violation (MCR 54)

This bit sets when the CP attempts a CM access without the required access permission. The CM access is blocked. The following conditions result in a CM access violation:

- Attempt to read a nonreadable segment.
- Attempt to read outside the read ring limit.
- Attempt to write into a nonwritable segment.
- Attempt to write outside the write ring limit.
- Attempt to execute from a nonexecutable segment.
- Attempt to execute from outside the execute ring bracket.
- Attempt to call indirect when the code base pointer is not in a binding section segment.
- Attempt to call indirect from a process outside the code base pointer call ring limit.
- Key/lock violation.

The PVA in P points to the instruction attempting the illegal access.

Address Specification Error (MCR 52)

This bit sets when the CP attempts to use an improper address. This includes:

- Data address with nonzero bits 61 through 63 generated by the following instructions:

Opcode	Instruction
04	Return.
B5	Call indirect.
80	Load multiple.
81	Store multiple.
82	Load word.
83	Store word.
B0	Call relative.
F4	Calculate subscript.

- Instruction address with nonzero bits 62 and 63 generated by unconditional branch instruction (2F).
- Any PVA with nonzero bit 32.

The following instructions may also detect an address specification error:

- Decimal arithmetic (70 to 75, E4, and E5).
- Move immediate data (F9).
- Convert from floating point to integer (3B).
- Test and set page (16).

The PVA in P points to the instruction specifying the incorrect address.

Arithmetic Loss-of-Significance (UCR 62)

This bit sets when significant digit(s) in the result are truncated or not stored in CM during execution of the following instructions:

- Decimal arithmetic (70 to 75, E4, and E5).
- Move immediate data (F9).
- Convert from floating point to integer (3B).

The PVA in P points to the instruction causing this condition.

Arithmetic Overflow (UCR 57)

This bit sets as a result of one of the following conditions:

- Integer sum instructions (10, 20, 24, 28, 8A, and 8B) when augend and addend have same signs but sum has opposite sign.

- Integer difference instructions (11, 21, 25, and 29) when minuend and subtrahend have opposite signs but difference sign is opposite of minuend sign.
- Half-word integer product instructions (22 and 8C) when most significant 32 bits of intermediate product are not equal to sign bit.
- Integer product instructions (26 and B2) when leftmost 64 bits of intermediate product are not equal to sign bit.
- Half-word integer quotient instruction (23) when -2^{31} is divided by -2^{0}.
- Integer quotient instruction (27) when -2^{64} is divided by -2^{0}.
- Decimal arithmetic instructions (70 to 73) when result length exceeds destination field length.
- Add immediate data instruction (FB) when source or destination field data descriptors specify invalid data type.

The PVA in P points to the instruction causing the arithmetic overflow condition.

Critical Frame Flag (UCR 53)

This bit sets when the CP attempts to execute a pop or return instruction from a critical stack frame. The PVA in P points to the pop or return instruction causing this interrupt.

Debug (UCR 56)

This bit sets when a debug match occurs. Refer to Debug in this section for a description of this condition. The PVA in P points to the instruction causing the debug interrupt.

Divide Fault (UCR 55)

This bit sets when the CP detects a divisor equal to zero during execution of one of the integer quotient instructions (23, 27, 33, 37, and 73). Also, for the floating point quotient instructions (33 and 37), the CP detects a divide fault if the divisor coefficient is a nonstandard value of zero, or is unnormalized and divisible into the dividend coefficient by a factor exceeding or equal to 2. The PVA in P points to the instruction causing the divide fault condition.

Environment Specification Error (MCR 55)

This bit sets when the CP detects an error in the environment during a call, return, or pop instruction, or during an exchange or trap operation. The PVA in P at the time of interrupt points to the instruction causing the error when the error results from any of the following conditions:

- A mismatch between VMCL and the VMID obtained from the code base pointer on a call indirect instruction (B5).
- A mismatch between VMCL and the VMID obtained from the stack frame save area (SFSA) on a return instruction (04).

- Initial A2 (previous save area pointer) not equal to A0 (dynamic space pointer) in the SFSA on a return (04) or pop (06) instruction.
- In the previous stack frame descriptor, the field value designating the last A register to be loaded is less than 2 on a return instruction (04).

When this error results from the following condition, the PVA in P at the time of interrupt points to the next instruction which would have executed:

- A mismatch between VMCL and the VMID obtained from the exchange package during an exchange operation.

When this error results from the following conditions, the PVA in P at the time of interrupt points to the instruction as defined under the condition causing the trap operation:

- A mismatch between VMCL and the VMID obtained from the code base pointer during a trap interrupt.
- External procedure flag not set in the code base pointer during a trap interrupt.
- The VMID from the code base pointer not equal to zero when executing a trap interrupt.

Exponent Overflow (UCR 58)

Exponent overflow occurs when an FP comparison or arithmetic instruction produces an exponent with an actual value between 2^{4096} and 2^{12187}. The PVA in P points to the next instruction that would have executed.

Exponent Underflow (UCR 59)

Exponent underflow occurs when an FP arithmetic instruction produces an intermediate exponent value between -2^{4096} and -2^{12287}. The PVA in P points to the next instruction that would have executed.

External Interrupt (MCR 56)

This bit sets as a result of a processor interrupt instruction (03) in the interrupted CP (or in another CP in a multiprocessor system), through the CM port the instruction specifies. At the time of interrupt, the PVA in P points to the next instruction that would have executed.

Floating-Point Indefinite (UCR 61)

This bit sets when an FP arithmetic instruction produces a final nonstandard indefinite result. The PVA in P points to the instruction causing the FP indefinite condition.

Floating-Point Loss-of-Significance (UCR 60)

This bit sets when an FP arithmetic instruction produces an intermediate result with an overflow bit and coefficient of all zeros. The PVA in P points to the next instruction that would have executed.

Free Flag (UCR 50)

The free flag is normally set by software in an exchange package in CM: this bit causes an immediate trap interrupt after an exchange operation loads this exchange package into the CP. Software conventions dictate the use of this flag; hardware does not set this bit. The PVA in P points to the next instruction that would have executed.

Instruction Specification Error (MCR 51)

This bit sets:

- During isolate/insert instructions (AC, AD, and AE) when the sum of the leftmost position designator plus the length designator exceeds 63.
- During business data processing (BDP) instructions when the length specified by the data descriptor L field exceeds the maximum length for applicable data type.
- If data type fields in source and/or destination data descriptors are invalid during the following BDP instructions:

Opcode	Instruction
70	Decimal sum.
71	Decimal difference.
72	Decimal product.
73	Decimal quotient.
74	Decimal compare.
75	Numeric move.
E4	Decimal scale.
E5	Decimal scale rounded.
ED	Edit.
F4	Calculate subscript and add.
F9	Move immediate data.
FA	Compare immediate data.
FB	Add immediate data.

- Execution of BDP calculate subscript and add instruction (FA) when PVA bits 61 through 63 (used to access the subscript range table (SRT)) do not equal 0.
- Execution of the program error instruction (00).
- Execution of the Copy-To-State-Register instruction (0F), or the Branch-on-Condition-Register instruction (9F), when execute access is restricted to Virtual State monitor mode with the CP not in this mode.
- Execution of a call instruction (B0/B5) when the number of the last A register field (At) in X0R bits 56 through 59) is less than 2.

The P register contains the PVA of the instruction with the error.

Inter-Ring Pop (UCR 52)

This bit sets from an attempt to pop a stack frame in one ring with a pop instruction (06) executing in a different ring. The pop instruction moves the CSF, PSA, and TOS pointers to eliminate the stack frame without changing the P-counter. The PVA in P points to the pop instruction attempting the inter-ring pop.

Invalid BDP Data (UCR 63)

This bit sets when the CP detects an invalid decimal digit during execution of the following instructions: BDP decimal numeric, calculate subscript and add, compare immediate data, move immediate data, edit, and convert floating point to integer. The PVA in P points to the instruction causing this condition.

Invalid Segment/Ring Number Zero (MCR 60)

This bit sets for the following reasons:

- A PVA was untranslatable into an RMA because the segment table length was exceeded or the segment descriptor was invalid. The PVA in P points to the instruction attempting the CM access.
- A call (B0 or B5) instruction attempted to execute with a code base pointer (CBP) ring number (RN) equal to zero. The PVA in P points to the instruction attempting the CM access.
- An A register was loaded with a PVA with RN equal to zero during a load A (80, 84, A0), return (04) or pop (06) instruction. The PVA in P points to the next instruction that would have executed.

Not Assigned (MCR 49)

When set explicitly by software, this bit causes an interrupt identical to the detected uncorrectable error (MCR 48).

Outward Call/Inward Return (MCR 61)

This bit sets when the CP attempts an outward call or an inward return. The PVA in P points to the instruction attempting the outward call or inward return.

Page Table Search Without Find (MCR 57)

This bit sets when a page table search does not locate the requested page table entry. The PVA in P points to the instruction attempting the CM access that resulted in this condition, except when this exception is caused by an instruction fetch directly after a branch exit. In this case, the PVA in P points to the branched-to instruction.

Privileged Instruction Fault (UCR 48)

This bit sets when:

- An attempt is made to execute a local privileged instruction from other than a locally-privileged or globally-privileged segment.
- An attempt is made to execute a globally-privileged instruction from other than a globally-privileged segment.
- A trap (017) instruction executes in CYBER 170 State.

The PVA in P points to the instruction causing the privileged instruction fault interrupt.

Process Interval Time (UCR 51)

This bit sets when the process interval timer decrements to zero. The PVA in P points to the next instruction that would have executed.

Detected Uncorrectable Error (MCR 48)

This bit sets when the CP detects an uncorrectable error condition in the processor, or on a processor-initiated memory reference. Typical examples are a parity error in data from memory that cannot be retried, an uncorrectable error in control storage, and CP errors that cannot be corrected or retried. A CM bounds violation also causes this exception.

The PVA in P does not necessarily point to the instruction causing the malfunction.

CYBER 170 State Exchange Request (MCR 53)

This bit sets when the CP receives the CYBER 170 State exchange request signal from the IOU, indicating that a PP in the IOU has executed one of the following instructions: exchange jump (00260X), monitor exchange jump (00261X), or monitor exchange jump MA (00262X). When the CP is in Virtual State, the operating system must switch the CP to CYBER 170 State job mode before the exchange can occur. The PVA in P points to the next instruction that would have executed.

Short Warning (MCR 50)

This bit sets when certain power distribution and warning system faults occur (refer to the appropriate power system manual listed in the preface). The PVA in P points to the next instruction that would have executed. This bit remains set until the condition returns to normal, at which time it clears.

Soft Error Log (MCR 62)

This bit sets to indicate error detection and correction by the hardware regarding the following:

- A corrected error in CM for the port used by this CP (also recorded in the CM corrected-error register).
- A corrected hardware malfunction in the CP.

The PVA in P points to the next instruction that would have executed.

System Interval Timer (MCR 59)

This bit sets when the system interval timer decrements to zero. The PVA in P points to the next instruction that would have executed.

Trap Exception (MCR 63)

This bit sets when the system detects a fault during a trap interrupt operation. In such a case, at least one other MCR bit indicates the cause of the trap exception. The PVA in P points to the PVA that would have been stored in the stack frame, word 0, if the trap had completed without any exception conditions.

Unimplemented Instruction (UCR 49)

This bit sets when an instruction not implemented in the CP attempts to execute. The instruction descriptions in section II-1 specify which instructions are model-dependent. The PVA in P points to the instruction causing the interrupt.

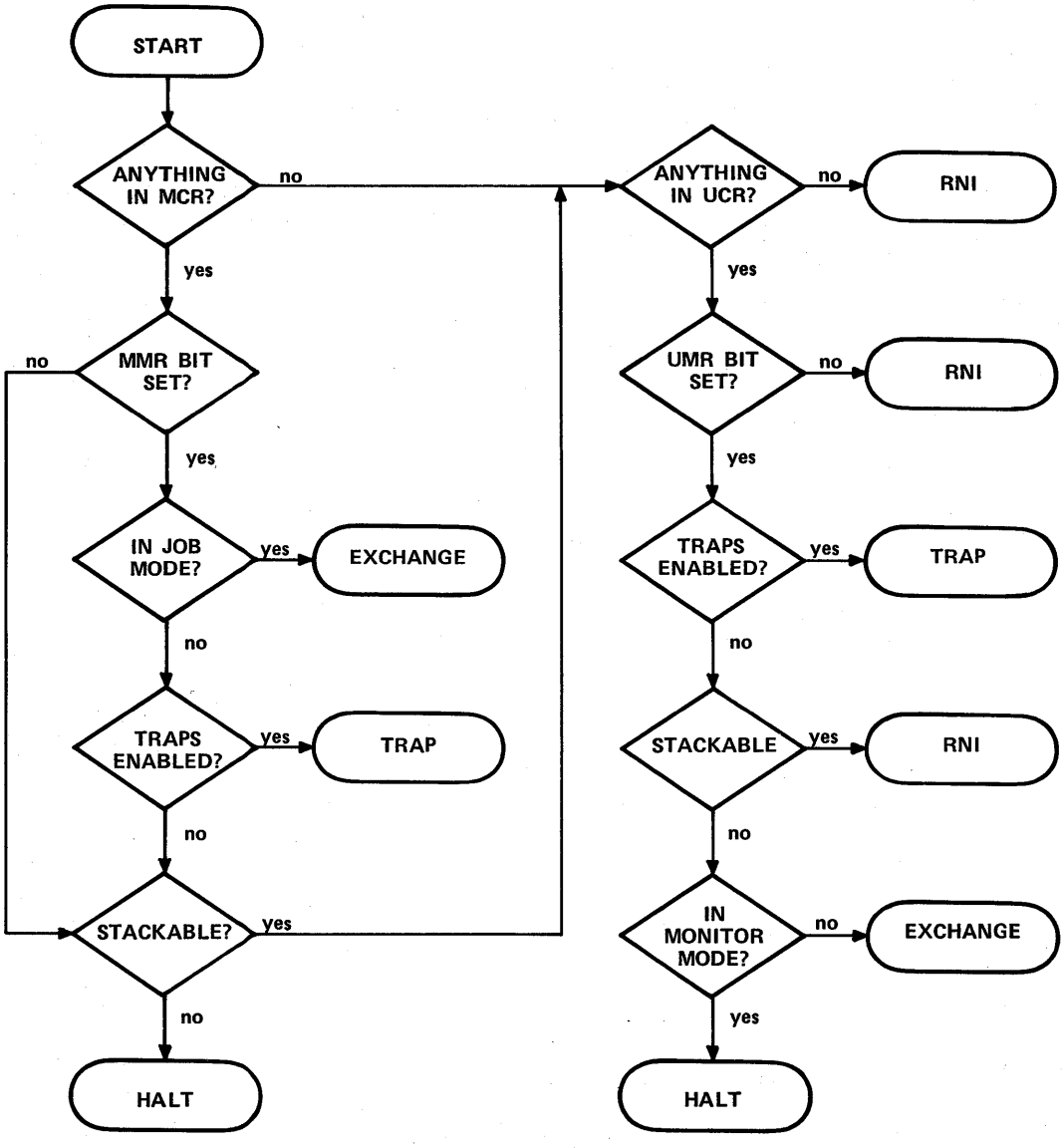
The CYBER 170 State compare/move instructions (464-467) also cause this interrupt.

MULTIPLE INTERRUPT CONDITIONS

Tables II-2-5 and II-2-6 list the interrupt action taken in various operating modes. When more than one bit sets in the MCR/UCR, the interrupts are processed with the following priority:

1. Halt (any halt condition present).
2. Exchange (no halt condition present, and any exchange condition present).
3. Trap (no halt or exchange condition present, and any trap condition present).
4. Stack (none of the above conditions present).

Figure II-2-3 is a flowchart showing the CP detecting an exception condition and taking action on it.



Notes: 1. RNI: Read next instruction.
2. Traps enabled implies: trap enable (TEF) set and trap enable delay (TED) clear.

Figure II-2-3. Interrupt Flowchart

FLAGS

Table II-2-8 indicates the state of the critical frame flag, on-condition flag, trap enable flip-flop, and trap enable delay flip-flop following the execution of the Virtual State call, return, pop, exchange, and trap operations.

Table II-2-8. Condition of Flags Following Call, Return, Pop, Exchange, and Trap Operations

Operations	Flags			
	CFF	OCF	TEF	TED
Call	C	C	A	A
Return	PS	PS	A	C
Pop	PS	PS	A	A
Exchange	XP	XP	XP	XP
Trap	C	C	C	A

Notes:

C	Cleared by operation.
A	As is (unchanged by operation).
PS	Loaded by operation from previous stack frame save area.
XP	Loaded by operation from exchange package.
CFF	Critical frame flag.
OCF	On-condition flag.
TEF	Trap enable flip-flop.
TED	Trap enable delay.

STACK MANIPULATING OPERATIONS

Each process has up to 15 stacks: one for each ring of execution privilege as defined by the RN field (bits 16 through 19) of the hardware P register. Hardware accesses these stacks to save/restore the process registers and operating conditions during trap interrupts, and during the following Virtual State instructions:

- Call indirect.
- Call relative.
- Return.
- Pop.
- Trap.

These 15 stacks are used as parts of a single stack divided solely to guarantee access protection. The buildup and reduction of the 15 stacks always occurs through the same locations (in opposite directions), switching from stack to stack only when the P register ring number changes.

The operating system allocates stack space to each process. One use of the critical frame flag (CFF) is to mark the first frame in each stack to indicate the beginning of the stack. The operating system may also check for a maximum allowable stack length when assigning a new page to the stack through the virtual memory demand paging mechanism.

The 15 stacks operate in conjunction with assigned registers A0 through A4 and 15 top-of-stack (TOS) pointers for the specific process. An exchange operation switches stacks by providing new A0 through A4 and new TOS pointers.

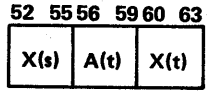
STACK FRAMES AND SAVE AREAS

A procedure may use its stack for storing its dynamic variables. At times it may call another procedure, which in turn may call another procedure, and so on. Also, at any time, a trap interrupt condition may initiate a call-type operation. Each time a call occurs, hardware saves the registers of the suspended part of the process (the caller) in the currently active stack, together with some status information. This leaves these registers free for use by the branched-to software (the callee). The area in which the registers are stored is the stack frame save area (SFSA). The SFSA combines with the previously stored dynamic variables (if any) to comprise a stack frame.

The CP hardware design provides that the string of successively-called procedures may include previously called procedures (recursive calls), provided code modification is not used.

Stack Frame Save Area Format

For call instructions, the programmer specifies the number of registers stored in SFSA (from 4 to 33) by way of a descriptor placed into X0, as shown in figure II-2-4. Trap interrupts always generate the maximum save area of 33 words. Figure II-2-5 shows the format of SFSA.



Field	Register Saved
X(s)	Starting (first) X register.
A(t)	Terminating (last) A register.
X(t)	Terminating (last) X register.

Notes

1. If X(s) exceeds X(t), nothing is stored.
2. First A register is always A0.
3. If A(t) is less than 2, an instruction specification error interrupt occurs.

Figure II-2-4. Format of X0 for Call Instructions

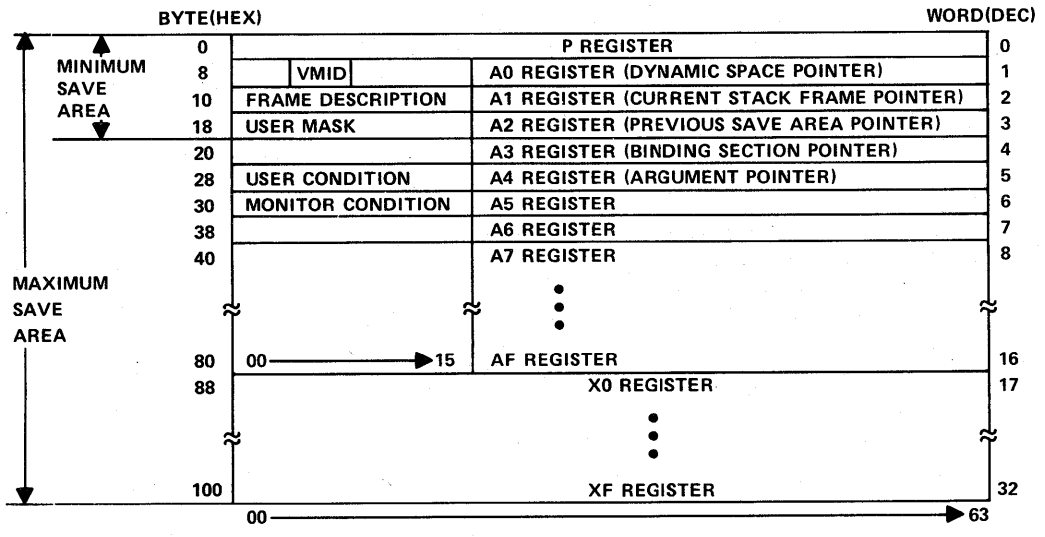


Figure II-2-5. Virtual State Stack Frame Save Area

Stack Frame Save Area Descriptor Field

The SFSA descriptor (figure II-2-6) is in word 2, bits 0 through 15 of the SFSA. It records the number of X and A registers saved in the SFSA, and also the state of the critical-frame flag (CFF), the on-condition flag (OCF), and the process-not-damaged (PND) flag when the SFSA is generated.

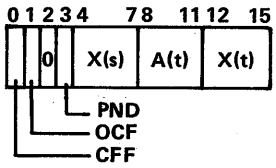
The CFF and the OCF are hardware register flags set/cleared by copy to/from state register instructions, and which may also be prerecorded in an exchange package or the SFSA. CFF set for the current stack frame inhibits instruction execution and causes an interrupt when encountered during a return or pop instruction.

Executing a call instruction or a trap interrupt stores the CFF and OCF in the SFSA descriptor generated for the current stack frame and clears these flags. Executing a return instruction loads these flags from the previous SFSA.

The PND flag indicates whether or not a process being executed was damaged and whether the process may be restarted. This flag is intended to allow recovery of monitor mode processes where possible.

The PND flag is valid when set during a Virtual State monitor mode trap operation caused by an uncorrectable error. In this case, the flag indicates that the executing process was undamaged and that it may be restarted. The PVA in P of the stack frame is the restart address for the process, but is not necessarily the address of the instruction which initiated the malfunction.

The default (clear) state of the PND flag interprets the process as damaged. The hardware ignores the flag when loading a stack frame.



Field	Description
CFF	Critical frame flag.
OCF	On-condition flag.
PND	Process not damaged.
X(s)	Starting (first) X register.
A(t)	Terminating (last) A register.
X(t)	Terminating (last) X register.

Note: If X(s) exceeds X(t), nothing is loaded.

Figure II-2-6. Stack Frame Save Area Descriptor

Virtual Machine Identifier (VMID) Field

A call instruction or a trap interrupt stores the virtual machine identifier in SFSA word 1, bits 4 through 7. A return instruction loads the VMID from the previous SFSA into the CP, with the exception that an attempt to load a VMID = 1 requires global privilege.

User Mask/Condition and Monitor Condition Fields

A trap interrupt (but not a call instruction) stores the UCR and MCR in words 5 and 6, respectively (bits 0 through 15), in the SFSA. The CP clears the condition register bit(s) causing the trap interrupt. The return instruction does not restore the condition registers from the SFSA to the CP.

ASSIGNED REGISTERS DURING STACK OPERATION

Stack manipulating operations change registers A0 through A4 and TOS pointers from the exchange package. For proper operation, the programmer must use registers A0 through A4 as designated.

Top-of-Stack Pointers

An exchange package contains 15 top-of-stack (TOS) fields which initially point to the next available vacant word in each stack, as set by the operating system. In subsequent use, TOS for the active stack points to the first word in the current stack frame. Hardware updates TOS when any stack frame is pushed or popped. Hardware uses TOS only when stacks switch.

The TOS pointers remain in the exchange package stored in CM and are accessed from there by hardware.

Dynamic Space Pointer (A0)

Register A0 has the role of dynamic space pointer (DSP), pointing to the first available vacant byte number in the active stack. Hardware updates DSP when a stack frame is pushed or popped. Software must update DSP when storing/removing process dynamic variables in the stack.

Current Stack Frame Pointer (A1)

Register A1 has the role of current stack frame pointer (CSF), pointing to the first word in the current stack frame. CSF updates when a stack frame is pushed or popped. The process must not reduce a stack frame below CSF, and must not change CSF.

Previous Save Area Pointer (A2)

Register A2 has the role of previous save area pointer (PSA), pointing to the first word of the previous save area (not necessarily in the currently active stack). PSA updates when a stack frame is pushed or popped. The process must not change PSA.

Binding Section Pointer (A3)

A binding section pointer (BSP) points to the first word in a list of indirect addresses (called code base pointers, or CBP) for use by call indirect instructions or other information determined by software conventions. During call indirect instructions to an external procedure (which has its own binding section), register A3 always provides the BSP of this external procedure. The call indirect instruction first uses the BSP in Aj to access the CBP containing the target address. When this CBP has its external procedure flag set, the word stored in CM immediately after this CBP loads into A3.

Argument Pointer (A4)

Register A4 has the role of argument pointer, used through software conventions. It copies from Ak during call indirect instructions. The process may use A4 as permitted through software conventions.

EXCEPTIONS DURING STACK OPERATIONS

When an exception causes a call instruction or trap interrupt to abort, the following may precede the abort:

- Dynamic space pointer (A0) may be rounded up.
- Portions of the environment may be stored into the save area on top of the current stack frame.

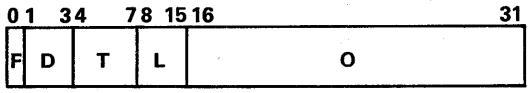
BUSINESS DATA PROCESSING PROGRAMMING

Business data processing (BDP) instructions operate on CM data fields which may be 1 through 8, 19, 38, or 256 bytes in length. BDP instructions utilize two forms of data fields in CM: 1) the source field, and 2) the destination field. The former modifies, replaces, or compares to the latter. These fields are independently designated by BDP data descriptors, described in the following paragraphs. The CP accommodates 9 types of packed and unpacked binary-coded decimal (BCD) data, plus alphanumeric, binary-unsigned, and binary-signed data types. In many cases the data types may be freely mixed as the hardware performs the necessary type translations. The CP also manipulates alphanumeric data fields.

BDP DATA DESCRIPTORS

The source and destination field data is described by one or two data descriptors obtained from the CM at locations immediately following a BDP instruction. The instructions using the format jk have two descriptors. The instructions using the format jkQ have either one or two descriptors.

As shown in figure II-2-7, each BDP data descriptor is a 32-bit half word describing the source or destination field data type, number of bytes, and relative memory location.



Field	Description
F	(1 bit) Function of the L field. Length retrieval information, as follows: F = 0 Length is obtained from the L field. F = 1 Length of the descriptor associated with Aj is obtained from X0R bits 55 through 63. Length of the descriptor associated with Ak is obtained from X1R bits 55 through 63. Other bits in X0R and X1R are not used.
D	(3 bits) Reserved.
T	(4 bits) Data type (refer to table II-2-5).
L	(8 bits) Length (in bytes) of the source or destination field (refer to table II-2-5). The maximum length is restricted according to the operand data type. When the maximum length is exceeded, an instruction specification error occurs, causing an interrupt or halt.
O	(16 bits) Offset. PVA of the leftmost byte of source or destination field is obtained by adding the sign-extended O field to the BN field of the base PVA in Aj or Ak, respectively.

Figure II-2-7. BDP Data Descriptor Format

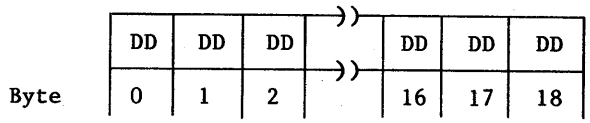
BDP DATA TYPES

The 12 data types listed in table II-2-9 are described in this subsection, including the permitted range of values for each data type with respect to digits (D), characters (C), signs (S), and maximum length (L).

Table II-2-9. BDP Operand Types and Field Lengths

T	Data Type	Maximum Length (Bytes)
0	Packed decimal, no sign.	19
1	Packed decimal, no sign, leading slack digit.	
2	Packed decimal, signed.	
3	Packed decimal, signed, leading slack digit.	
4	Unpacked decimal, unsigned.	38
5	Unpacked decimal, trailing sign combined Hollerith.	
6	Unpacked decimal, trailing sign separate.	
7	Unpacked decimal, leading sign combined Hollerith.	
8	Unpacked decimal, leading sign separate.	
9	Alphanumeric.	256
10	Binary, unsigned.	8
11	Binary, signed.	

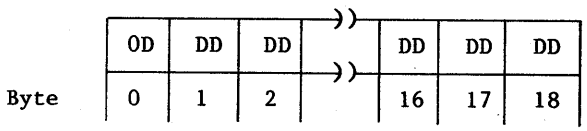
Data Type 0: Packed Decimal, Unsigned



D: Hex 0 through 9.
L: 19 bytes maximum.

This format corresponds to an even number of digits in the decimal number.

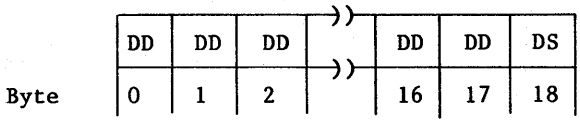
Data Type 1: Packed Decimal, Unsigned Slack Digit



O: Hex 0.
D: Hex 0 through 9.
L: 19 bytes maximum.

This format corresponds to an odd number of digits in the decimal number.

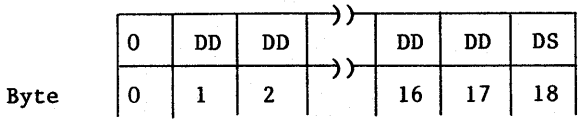
Data Type 2: Packed Decimal, Signed



D: Hex 0 through 9.
S: (Positive sign) hex A, B, C, E, or F (C preferred);
(Negative sign) hex D.
L: 19 bytes maximum.

This format corresponds to an odd number of digits in the decimal number.

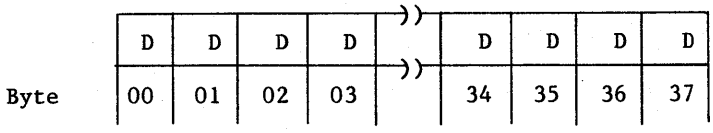
Data Type 3: Packed Decimal, Signed, Slack Digit



0: Hex 0.
D: Hex 0 through 9.
S: (Positive sign) hex A, B, C, E, or F (C preferred);
(Negative sign) hex D.
L: 19 bytes maximum.

This format corresponds to an even number of digits in the decimal number.

Data Type 4: Unpacked Decimal, Unsigned



D: ASCII characters 0 through 9 (represented by hex 30 through 39).
L: 38 bytes maximum.

Data Type 5: Unpacked Decimal, Trailing Sign Combined Hollerith

	D	D	D	D	⟩⟩	D	D	D	C
Byte	00	01	02	03	⟩⟩	34	35	36	37

In the following, the preferred characters and codes are underlined.

D: ASCII character 0 to 9 (represented by hex 30 through 39).
C: ASCII character decoded as follows:

ASCII 1 to 9 (hex 31 through 39) represents +1 through +9, or
ASCII A through I (hex 41 through 49) represents +1 through +9.
ASCII J through R (hex 4A through 4F and hex 50 through 52) represents -1 through -9.
ASCII {, [, 0, 8 (hex 7B, 3C, 30, 26) represents +0.
ASCII },], - (hex 7D, 21, 2D) represents -0.

L: 38 bytes maximum.

Data Type 6: Unpacked Decimal, Trailing Sign Separate

	D	D	D	⟩⟩	D	D	D	D	S
Byte	0	1	2	⟩⟩	33	34	35	36	37

D: ASCII character 0 through 9 (hex 30 through 39).
S: ASCII character + (hex 2B), positive sign;
ASCII character - (hex 2D), negative sign.
L: 38 bytes maximum.

Data Type 7: Unpacked Decimal, Leading Sign Combined Hollerith

	C	D	D	D	⟩⟩	D	D	D	D
Byte	0	1	2	3	⟩⟩	34	35	36	37

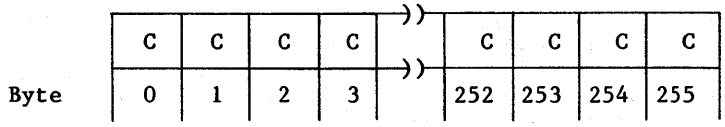
D: Same as data type 5.
C: Same as data type 5.
L: 38 bytes maximum.

Data Type 8: Unpacked Decimal, Leading Sign Separate

	S	D	D	D	⟩⟩	D	D	D	D
Byte	0	1	2	3	⟩⟩	34	35	36	37

S: Same as data type 6.
D: Same as data type 6.
L: 38 bytes maximum.

Data Type 9: Alphanumeric



C: Any ASCII character code.
L: 256 bytes maximum.

Data Type 10: Binary, Unsigned

L: 8 bytes maximum.

The L bytes of the type 10 data field contain the positive binary operand value. Negatively-signed data moved to a type 10 destination field is considered positive.

Data Type 11: Binary, Signed

L: 8 bytes maximum.

The L bytes of the type 11 data field contain the signed binary operand value. Negative values are represented in two's complement form.

Slack Digit

With data types 1 and 3, the slack digit value as read from CM is ignored and treated as zero. The slack digit value as written into CM is forced to zero and is not affected by any arithmetic overflow or arithmetic loss-of-significance that may occur.

UNDEFINED RESULTS

Overlap

BDP instruction execution produces undefined results whenever the source and destination fields overlap and the leftmost and rightmost byte positions do not coincide.

Invalid Data

As a rule, invalid BDP data causes undefined results to be stored in the destination field in CM only when the corresponding mask bit is clear or traps are disabled. An exception: the decimal-compare and numeric-move instructions always store undefined results in X1R when invalid BDP data is detected.

VECTOR PROGRAMMING

Vector operations are memory-to-memory operations; that is, the CP accesses one or two source vector streams from CM, repeats an operation on successive elements, and returns the results to CM in a destination vector stream. These operations occur without modifying the CP operating registers. Refer to Vector Instruction Descriptions in section 1 of this manual for detailed descriptions of the vector instructions.

Most vector instructions stream results at a one-clock-period rate. The source and destination vectors may be consecutive streams of:

- Floating-point operands (12-bit exponent plus sign bit, 48-bit coefficient plus sign bit).
- 64-bit integers.
- 64-bit elements (shift/compare/logical data).

Refer to table II-2-10 for the source- and destination-vector characteristics of each type of vector operation.

Table II-2-10. Vector Operations

Vector Operation	Source Vector A	Source Vector B	Destination Vector
Integer Arithmetic	64-bit integers	64-bit integers	64-bit integers
Integer Compare	64-bit integers	64-bit integers	64-bit elements
Logical Arithmetic	64-bit elements	64-bit elements	64-bit elements
Floating-Point Arithmetic	F.P. operands	F.P. operands	F.P. operands
Floating-Point Summation	-	F.P. operands	F.P. operand †
Shift Circular	64-bit elements	64-bit elements	64-bit elements
Merge	64-bit elements	64-bit elements	64-bit elements
Gather	Nonconsecutive 64-bit elements	interval ††	Consecutive 64-bit elements
Scatter	Consecutive 64-bit elements	interval ††	Nonconsecutive 64-bit elements

† Stored in an X register; not sent to CM.
†† Accessed from an X register.

VECTOR LENGTH (NUMBER OF OPERATIONS)

The D-field rightmost 10 bits, when nonzero, specify the length or number of operations to be performed (1 through 512). X1R specifies the length when the D-field rightmost 10 bits equal zeros, as follow:

- When X1R is positive and less than 512, this number provides the vector length.
- When X1R is positive and greater than 512, the vector length is 512.

An instruction specification error (MCR 51) occurs when X1R is negative or when the D-field rightmost 10 bits are greater than 512. When the D-field rightmost 10 bits and all 32 bits of X1R are zeros, no memory reference occurs but the instruction undergoes normal address-exception detection.

VECTOR PAGE SIZE

Vector operations require a page size of 4096 bytes (512 words) or larger. (A page size of less than 4096 bytes inhibits the vector instruction and results in an environment specification error, MCR 55.) Since a vector may be from 1 to 512 elements long and may overlap page boundaries, it may occupy at most two partially-filled pages. Exceptions to this are the gather and scatter instructions, which by nature may require up to 512 pages per vector.

VECTOR BROADCAST

Vector broadcast is an additional feature for use with all vector instructions except Floating-Point Summation. Vector broadcast generates a source vector by repeating a single 64-bit element contained in the Xj register in place of V(Aj) if the leftmost D-field bit is a one.

VECTOR INTERRUPTS

Vector instructions (other than gather and scatter) may not be interrupted after any results in the destination vector stream have been stored in CM. Instruction execution completes before a monitor mode routine can process the interrupt. A program interrupt occurring before any results are stored inhibits the instruction. For the gather and scatter instructions, interrupts may occur after results have been partially stored in CM.

Interrupting a vector instruction prevents the addressing section from making additional memory requests, and purges all unused operands assembled for execution.

VECTOR OVERLAP

Source and destination vectors for the same instruction may overlap only when the destination-vector starting address is less than or equal to the source-vector starting address. All other cases of source- and destination-vector overlap within a single instruction cause undefined results.

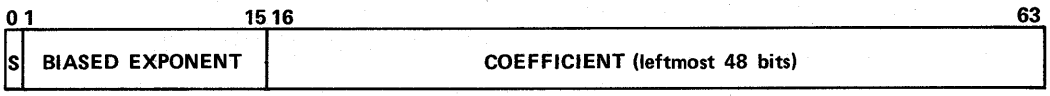
FLOATING-POINT PROGRAMMING

Floating-point (FP) arithmetic automatically maintains binary point placement during computations involving large numeric values or values within a widely varying range. This occurs by separating a number's significant digits from the number size to express the number as a fraction multiplied by a power of 2. Thus, each FP number contains two values as follows:

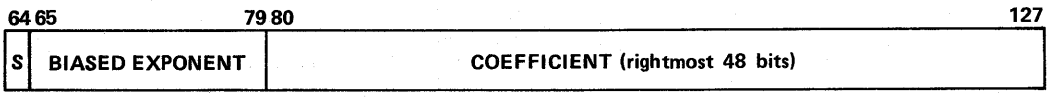
- Coefficient (fraction) represents the number's significant digits. The binary point of the coefficient is always directly left of its most significant bit.
- Exponent (characteristic) is a power of 2 by which the coefficient must be multiplied to obtain the whole FP number value.

FLOATING-POINT DATA FORMATS

FP data exists in 64- and 128-bit fixed-length formats (single precision and double precision), as shown in figure II-2-8.



Format of single precision FP number and of the leftmost part of double precision FP number



Format of rightmost part of double precision FP number

Bits	Description
0,64	Coefficient sign 0 Coefficient positive. 1 Coefficient negative. Double-precision FP sign bit 64 is set equal to bit 0 in results.
16-63	Coefficient magnitude (leftmost 48 bits).
80-127	Coefficient magnitude (rightmost 48 bits). The coefficient without the sign bit is an unsigned, exclusively positive fraction with the binary point directly left of bit 16. Double-precision FP number bits 64 through 79 are set equal to bits 0 through 15 in results.
1	Exponent sign, biased.
4-15, 65-79	Exponent value, two's complement. The exponent is a biased-signed two's complement integer. A bias of 16,384 (4000_{16}) adds to the exponent to allow encoding of exponent values from -4096 to +4095 inclusive (within the 15 exponent bits) as shown in table II-2-11.
1-3	Coded to indicate the following FP numbers 00X FP zero. 0X0 FP zero. 011 Standard FP number. 100 Standard FP number. 101 FP infinity. 110 FP infinity. 111 FP indefinite.

Figure II-2-8. Floating-Point Data Formats

Table II-2-11. Floating-Point Representation

Exponent With Coefficient Sign Hexadecimal	Actual Exponent (Binary)	Input Arguments	Term Used for Numbers in This Range
Positive Numbers (Coefficient sign = 0)			
7XXX	---	Indefinite	+IND
6FFF ↑ 5000	12278 ↑ 4096	Infinite	+∞
4FFF ↑ 4000	4095 ↑ 0	Standard	+N
3FFF ↑ 3000	-1 ↑ -4096		+Z3
2FFF ↑ 1000	-4097 ↑ -12288	Zero	+Z2
0XXX	---	Zero	+Z1
Negative Numbers (Coefficient sign = 1)			
8XXX		Zero	-Z1
9000 ↓ AFFF	-12288 ↓ -4096	Zero	-Z2
8000 ↓ 8FFF	-4096 ↓ -1	Standard	-Z3
C000 ↓ CFFF	0 ↓ 4095		-N
D000 ↓ EFFF	4096 ↓ 12287	Infinite	-∞
FXXX	---	Indefinite	-IND

STANDARD AND NONSTANDARD FP NUMBERS

Nonstandard FP numbers are FP numbers outside the capacity of standard FP numbers. Special exponent field codes identify the three nonstandard FP numbers: zero (±Z1,±Z2), infinity (±∞), and indefinite (±INDEF). Table II-2-11 lists hexadecimal exponent codes for nonstandard and standard FP numbers.

Floating-Point Zero

Nonstandard FP operands with bits 01 and 02, or 01 and 03 clear, are treated as if consisting of all zeros.

The nonstandard zero FP numbers are represented as ±Z1 or ±Z2 as shown in table II-2-11. The specific number in the +Z1 range which consists of all (64) zeros is termed +0. Thus, wherever ±Z1 is indicated, the +0 is also included since it is a member of +Z1.

The standard zero FP numbers are represented as ±Z3 as shown in table II-2-11.

Floating-Point Nonzero

Standard FP operands which have nonzero coefficients are represented as ±N in table II-2-11.

Floating-Point Infinite

Nonstandard FP operands with bit 01 set, and bit 02 not equal to bit 03, are treated as infinite values.

The nonstandard FP numbers in the Infinite range are represented as ±∞ as shown in table II-2-11. The specific number in the +∞ range consists of 5000---000. The specific number in the -∞ range consists of D000---000.

Floating-Point Indefinite

Nonstandard FP numbers with bits 01, 02, and 03 set are treated as indefinite values.

The nonstandard FP numbers in the Indefinite range are represented as ±IND as shown in table II-2-11. The specific number in the +IND range consists of 7000---000. The specific number in the -IND range consists of F000---000.

Double-Precision Nonstandard FP Numbers

When nonstandard results are generated, the rightmost part of a double-precision FP result is made identical to the leftmost part.

EXPONENT ARITHMETIC

When the operand exponent fields are added (as in FP multiplication) or subtracted (as in FP division), the exponent arithmetic performs algebraically in two's complement mode. Such operations take place as if the bias were removed.

Exponent underflow and overflow conditions are detected for all single-precision results, but only for the leftmost part of double-precision results.

NORMALIZATION

Normalized operands ensure the highest possible precision in the result. An FP number is normalized when the coefficient bit 16 is a 1.

Normalization takes place when intermediate results become final results. It occurs by left-shifting the 48-bit fraction until bit 16 is a 1, and by reducing the exponent value by the number of positions shifted. Numbers with zero fractions cannot be normalized and remain equal to zero.

When normalizing a double-precision FP number, the entire 96-bit fraction left-shifts until bit 16 is a 1, with a corresponding exponent value reduction.

If the coefficient of an intermediate result overflows, the fraction right-shifts 1 bit position and the exponent increases by 1. If the input operands for the FP multiply (32 and 36) and the FP divide (33 and 37) instructions are unnormalized, the result may be unnormalized.

FLOATING-POINT SUM AND DIFFERENCE

When two FP operands with unequal exponents are added or subtracted, the hardware aligns copies of these operands before the addition or subtraction performs, as follows: the fraction with the smaller exponent is right-shifted, end-off, by the number of bit positions equal to the difference between the exponents. The maximum shift is 48 positions (single-precision) or 96 positions (double-precision). After copies of the fractions have been aligned in this manner, they are added or subtracted. The result generated is 48 bits (single-precision) or 96 bits (double-precision).

When summing coefficients with like signs or subtracting coefficients with unlike signs, the result may overflow/underflow by 1 bit; this bit is saved. In such case, the 48- or 96-bit intermediate result fraction shifts right 1 position, end-off. The overflown bit inserts into the high-order position. The result's exponent increases by 1 to adjust for the right shift of the coefficient. The adjusted exponent and the 48-bit or 96-bit fraction and its sign bit are the final result.

FLOATING-POINT MULTIPLY

The signed exponents for the two input operands to be multiplied are algebraically added, with the result used as an intermediate exponent.

The multiplied fractions generate an intermediate product with 96 bits (single-precision) or 192 bits (double-precision). The correct sign bit is algebraically determined. If the high-order bit in the product is a 1, the product is already normalized and remains unchanged. If the high-order bit in the product is a zero, the entire 96- or 192-bit product left-shifts 1 bit position and the intermediate exponent decreases by 1. This one-position shift normalizes the product if the original input operands were normalized. The high-order 48 or 96 bits of the product or shifted product become an intermediate coefficient. If the final intermediate exponent indicates a standard FP number, the intermediate exponent and the intermediate coefficient with its sign bit are the final result.

FLOATING-POINT DIVIDE

During execution of an FP divide instruction, the divisor (Xj or XXj) exponent is subtracted from the dividend (Xk or XXk) exponent, and the signed result provides an intermediate exponent.

The dividend (Xk or XXk) fraction is divided by the divisor (Xj or XXj) fraction. The result's sign is determined algebraically from the operand signs.

If the fraction in Xj or XXj is initially unnormalized and can be divided into the fraction in Xk or XXk by a factor equal to or exceeding 2, a divide fault occurs, setting UCR bit 55, with an interrupt (when enabled).

If no error occurs, the intermediate quotient generated is 48 or 96 bits. When the divisor can be divided into the dividend by a factor equal to or exceeding 1, but less than 2, an overflow bit also generates. If the overflow bit is a zero, the sign bit and the 48- or 96-bit fraction require no further adjustment. When the overflow bit is a 1, the 48- or 96-bit fraction right-shifts 1 position, end-off, and the overflow bit inserts into the high-order bit position. In such a case, the exponent increases by 1 to compensate for the shift.

The intermediate exponent and the intermediate fraction (with its sign) then transfer as the final result to the Xk register.

FLOATING-POINT END CASES

Tables II-2-12 through II-2-20 list FP end cases. The nomenclature used is as follows:

N	Standard FP number: $(3000)_{16} \leq$ exponent $< (5000)_{16}$ nonzero, coefficient normalized or unnormalized.
0	Zero: sign bit followed by 63 zero bits.
Z1	Zero: FP numbers with exponents in the range $0000_{16} \leq$ exponent $< 1000_{16}$.
Z2	Underflow, zero: FP numbers with exponents in the range $1000_{16} \leq$ exponent $< 3000_{16}$.
Z3	Zero: An unnormalized FP number with a zero coefficient and a standard exponent. That is, $3000_{16} \leq$ exponent $< 5000_{16}$.
INF	FP numbers with exponents in the range $5000_{16} \leq$ exponent $< 7000_{16}$.
∞	Infinite: The nonstandard FP number sign, 5000 0000 0000 0000_{16}.
INDEF	FP numbers with exponents in the range $7000_{16} \leq$ exponent $\leq 7FFF_{16}$.
+IND	Indefinite: The nonstandard FP number 7000 0000 0000 0000_{16}.
INDC	A result of indefinite generated by the FP compare instruction. That is, a value for X1R = 8000 0000_{16}.
S	Algebraic sum of two FP numbers (excluding Z3).
D	Algebraic difference of two FP numbers (excluding Z3).
P	Algebraic product of two FP numbers (excluding Z3).
Q	Algebraic quotient of two FP numbers (excluding Z3).
DVF	Divide fault condition.
OVL	Exponent overflow, FP.
UNL	Exponent underflow, FP.
LOS	Loss of significance, FP.
IND	Indefinite, FP.

Table II-2-12. FP Compare Results

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+D, +Z2 -D, -Z2> +Z3 =	<	+D, +Z2< +Z3 =	<	<	>	<	Note 1
-N	>	+D, +Z2< -D, -Z2> +Z3 =	>	-D, -Z2> +Z3 =	>	>	<	
+Z3	-D, -Z2> +Z3 =	<	+Z3 =	<	<	>	<	
-Z3	>	+D, +Z2< +Z3 =	>	+Z3 =	>	>	<	
±Z1 ±Z2	>	<	>	<	=	>	<	
+INF	<	<	<	<	<	Note 1	<	
-INF	>	>	>	>	>	>	Note 1	
±INDEF	Note 1							

Note: 1. FP branch instructions perform normal exit and record FP indefinite (UCR 61). FP compare instructions set X1 to INDC and record FP indefinite (UCR 61), except when UMR 61 is set traps are enabled, in which case X1 is unaltered.

Table II-2-13. FP Sum Results, UM Clear

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+S +∞ OVL +0 UNL	±S +0 UNL +0 LOS	+S +0 UNL +0 LOS	+S +0 UNL +0 LOS	+N +0 UNL	+∞ OVL	+∞ OVL	+IND IND
-N		-S - OVL +0 UNL	-S +0 UNL +0 LOS	-S +0 UNL +0 LOS	-N +0 UNL	+∞ OVL	+∞ OVL	+IND IND
+Z3			+0 LOS	+0 LOS	+0 LOS	+∞ OVL	+∞ OVL	+IND IND
-Z3				+0 LOS	+0 LOS	+∞ OVL	+∞ OVL	+IND IND
±Z1 ±Z2			+0 LOS	+0 LOS	+0	+∞ OVL	+∞ OVL	+IND IND
+INF						+∞ OVL	+IND IND	+IND IND
+INF							+∞ OVL	+IND IND
±INDEF								+IND IND

Table II-2-14. FP Sum Results, UM Set

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+S +∞ OVL +Z2 UNL	±S ±Z2 UNL ±Z3 LOS	+S +Z2 UNL +Z3 LOS	+S +Z2 UNL +Z3 LOS	+N +Z2 UNL	+∞ OVL	-∞ OVL	+IND IND
-N		-S -∞ OVL +Z2 UNL	-S -Z2 UNL +Z3 LOS	-S -Z2 UNL ±Z3 LOS	-N -Z2 UNL	+∞ OVL +∞ OVL	-∞ OVL +∞ OVL	+IND IND
+Z3			+Z3 LOS	+Z3 LOS	+Z3 LOS	+∞ OVL	+∞ OVL	+IND IND
-Z3				+Z3 LOS	+Z3 LOS	+∞ OVL	+∞ OVL	+IND IND
±Z1 ±Z2			+Z3 LOS	+Z3 LOS	+0	+∞ OVL	+∞ OVL	+IND IND
+INF						+∞ OVL	+IND IND	+IND IND
+INF							-∞ OVL	+IND IND
±INDEF								+IND IND

Note: This chart is for traps disabled. For traps enabled, replace +IND with Xk.

Table II-2-15. FP Difference Results, UM Clear

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	-D +0 UNL +0 LOS	+D +∞ OVL +0 UNL	+D +0 UNL +0 LOS	+D +0 UNL +0 LOS	+N +0 UNL	-∞ OVL	+∞ OVL	+IND IND
-N	-D -∞ OVL +0 UNL	+D +0 UNL +0 LOS	-D +0 UNL +0 LOS	-D +0 UNL +0 LOS	-N +0 UNL	-∞ OVL	+∞ OVL	+IND IND
+Z3	-D +0 UNL +0 LOS	+D +0 UNL +0 LOS	+0 LOS	+0 LOS	+0 LOS	+∞ OVL	+∞ OVL	+IND IND
-Z3	-D +0 UNL +0 LOS	+D +0 UNL +0 LOS	+0 LOS	+0 LOS	+0 LOS	+∞ OVL	+∞ OVL	+IND IND
±Z1 ±Z2	-N +0 UNL	+N +0 UNL	+0 LOS	+0 LOS	+0	+∞ OVL	+∞ OVL	+IND IND
+INF	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+IND IND	+∞ OVL	+IND IND
-INF	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	+IND IND	+IND IND
±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND

Table II-2-16. FP Difference Results, UM Set

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+D ±Z2 UNL +Z3 LOS	+D +∞ OVL +Z2 UNL	+D +Z2 UNL +Z3 LOS	+D +Z2 UNL +Z3 LOS	+N +Z2 UNL	-∞ OVL	+∞ OVL	+IND IND
-N	-D -∞ OVL -Z2 UNL	+D ±Z2 UNL +Z3 LOS	-D -Z2 UNL +Z3 LOS	-D -Z2 UNL +Z3 LOS	-N -Z2 UNL	-∞ OVL	+∞ OVL	+IND IND
+Z3	-D -Z2 UNL +Z3 LOS	+D +Z2 UNL +Z3 LOS	+Z3 LOS	+Z3 LOS	+Z3 LOS	+∞ OVL	+∞ OVL	+IND IND
-Z3	-D +Z2 UNL +Z3 LOS	+D +Z3 UNL +Z3 LOS	+Z3 LOS	+Z3 LOS	+Z3 LOS	+∞ OVL	+∞ OVL	+IND IND
±Z1 ±Z2	-N -Z2 UNL	+N +Z2 UNL	+Z3 LOS	+Z3 LOS	+0	+∞ OVL	+∞ OVL	+IND IND
+INF	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+IND IND	+∞ OVL	+IND IND
-INF	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	+IND IND	+IND IND
±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND

Note: This chart is for traps disabled. For traps enabled, replace +IND with Xk.

Table II-2-17. FP Product Results, UM Clear

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+P +∞ OVL +0 UNL +Z3	-P - OVL +0 UNL +Z3	+∞ OVL +0 UNL +Z3	+∞ OVL +0 UNL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
-N		+P +∞ OVL +0 UNL +Z3	-∞ OVL +0 UNL +Z3	+∞ OVL +0 UNL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
+Z3			+∞ OVL +0 UNL +Z3	-∞ OVL +0 UNL +Z3	+0	+∞ OVL	+∞ OVL	+IND IND
-Z3				+∞ OVL +0 UNL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
±Z1 ±Z2			+0	+0	+0	+IND IND	+IND IND	+IND IND
+INF						+∞ OVL	-∞ OVL	+IND IND
-INF							+∞ OVL	+IND IND
±INDEF								+IND IND

Table II-2-18. FP Product Results, UM Set

	Standard Numbers				Nonstandard Numbers			
Xj / Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+P +∞ OVL +Z2 UNL +Z3	-P -∞ OVL +Z2 UNL +Z3	+P OVL +Z2 UNL +Z3	-P OVL +Z2 UNL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
-N		+P +∞ OVL +Z2 UNL +Z3	-P OVL -Z2 UNL +Z3	+P OVL +Z2 UNL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
+Z3			+P OVL +Z2 UNL +Z3	-P OVL -Z2 UNL +Z3	+0	+∞ OVL	-∞ OVL	+IND IND
-Z3				+P OVL +Z2 UNL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
±Z1 ±Z2			+0	+0	+0	+IND IND	+IND IND	+IND IND
+INF						+∞ OVL	-∞ OVL	+IND IND
-INF							+∞ OVL	+IND IND
±INDEF								+IND IND

Note: This chart is for traps disabled. For traps enabled, replace +IND with Xk.

Table II-2-19. FP Quotient Results, UM Clear

	Standard Numbers				Nonstandard Numbers			
Xk \ Xj	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+Q +∞ OVL +0 OVL +Z3 Xk DVF	-Q -∞ OVL +0 OVL +Z3 Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
-N	-Q +∞ OVL +0 UNL +Z3 Xk DVF	+Q +∞ OVL +0 UNL +Z3 Xk DVF	-P OVL Xk DVF	+P OVL Xk DVF	Xk DVF	+0	+0	+IND IND
+Z3	+∞ OVL +0 UNL +Z3	+∞ OVL +0 UNL +Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
-Z3	+∞ OVL +0 UNL +Z3	+∞ OVL +0 UNL +Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
±Z1 ±Z2	+0	+0	+0	+0	Xk DVF	+0	+0	+IND IND
+INF	+∞ OVL	+∞ OVL	+∞ OVF	-∞ OVF	Xk DVF	+IND IND	+IND IND	+IND IND
-INF	+∞ OVL	+∞ OVL	-∞ OVF	+∞ OVF	Xk DVF	+IND IND	+IND IND	+IND IND
±INDEF	+IND IND	+IND IND	IND IND	IND IND	Xk DVF	+IND IND	+IND IND	+IND IND

Table II-2-20. FP Quotient Results, UM Set

	Standard Numbers				Nonstandard Numbers			
Xj Xk	+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
+N	+0 + +∞ OVL +Z2 UNL +Z3 Xk DVF	-0 -∞ OVL -Z2 UNL +Z3 Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
-N	-0 -∞ OVL +Z2 UNL +Z3 Xk DVF	+0 + +∞ OVL +Z2 UNL +Z3 Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
+Z3	+Q OVL +Z2 UNL +Z3	-Q OVL +Z2 UNL +Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
-Z3	-Q OVL -Z2 UNL +Z3	+Q OVL +Z2 UNL +Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
±Z1 ±Z2	+0	+0	+0	+0	Xk DVF	+0	+0	+IND IND
+INF	+∞ OVL	-∞ OVL	+∞ OVF	-∞ OVF	+∞ OVF	+IND IND	+IND IND	+IND IND
-INF	-∞ OVL	+∞ OVL	-∞ OVF	+∞ OVF	-∞ OVF	+IND IND	+IND IND	+IND IND
±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND
Note: This chart is for traps disabled. For traps enabled, replace +IND with Xk.								

PROGRAM MONITORING

The CP provides a debug feature to aid the debugging of new Virtual State programs.

The debug feature causes a debug trap interrupt (when enabled), when a CM access of a given type into a given PVA range occurs. The user can specify up to 32 access ranges and 5 access types for simultaneous debugging.

DEBUG

When enabled, the debug feature tests all executive state memory accesses by scanning a list of up to 32 entries of selected access type and PVA range combinations. When a match is found, UCR bit 56 sets and, if enabled, a trap interrupt occurs.

A debug address range may span an entire process virtual segment or any contiguous byte field within the segment. Any or all of the following access types can be selected:

- Data read.
- Data write.
- Instruction fetch (excluding target instruction fetch).
- Branch or return target instruction fetch (excluding call target instruction fetch).
- Call target instruction fetch.

Debug is enabled by setting UMR bit 56 and the trap enable flip-flop. Debug is controlled by the following:

- Debug list - lists access types and PVA ranges.
- Debug list pointer register - PVA of first entry in debug list.
- Debug mask register - enables/disables any or all access types.
- Debug index registers - record number of debug list entries scanned during debug of a single instruction.

Debug List

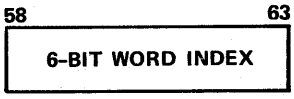
The debug list has up to 32 double-word entries (refer to figure II-2-9), each one aligned at a word boundary. The end of list (debug code bit 5) is interpreted after all other bits in the same debug code have been interpreted and acted upon. Any or all access types can be selected for debug by the debug code. Access types are defined by the type of access privilege required (read, write, or execute).

Debug List Pointer Register

This register is a process register containing a PVA that points to the first debug list entry.

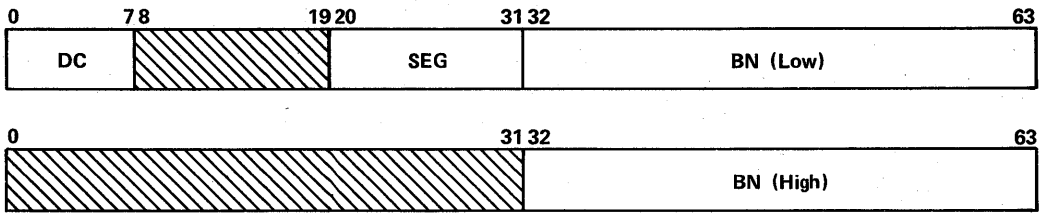
Debug Index Register

This register is a process register which increments as the debug scan proceeds. It contains a 6-bit word index that is added to the debug list pointer register contents. This generates the debug list entry addresses during a debug scan initiated by each instruction accessing virtual memory. The debug index register format is as follows:



Debug Mask Register

This register (figure II-2-10) is a process register which activates the access types selected in the debug code field of a debug list entry. Each access type selected in the debug code is activated for debugging only if the corresponding debug mask bit is also set, as shown in figure II-2-10.



Field	Description
BN	Byte numbers of the first and last bytes in the contiguous byte field in memory to which the debug code applies.
SEG	Process segment number to which the debug code applies.
DC	Debug code (listed below).

DC Bit	Operations Triggering Scan
0	Data read. Activates debug scan on all CM read accesses.
1	Data write. Activates debug scan on all CM write accesses.
2	Instruction fetch. Activates debug scan on all CM execute accesses, after instruction fetch.
3	Branch or return target instruction fetch, if branch occurs. PVA bracket applies to the PVA branched to. Call (B5/B0) target addresses are excluded. Branch address generated during compare and swap instruction (B4) is excluded.
4	Call (B5/B0) target instruction fetch. Address bracket applies to address of the called procedure. For the call indirect (B5) instruction, this is the code base pointer address; for the call relative (B0) instruction, this is the address in the P register plus displacement.
5	End of list. Denotes the last debug list entry.

Figure II-2-9. Debug List Entry

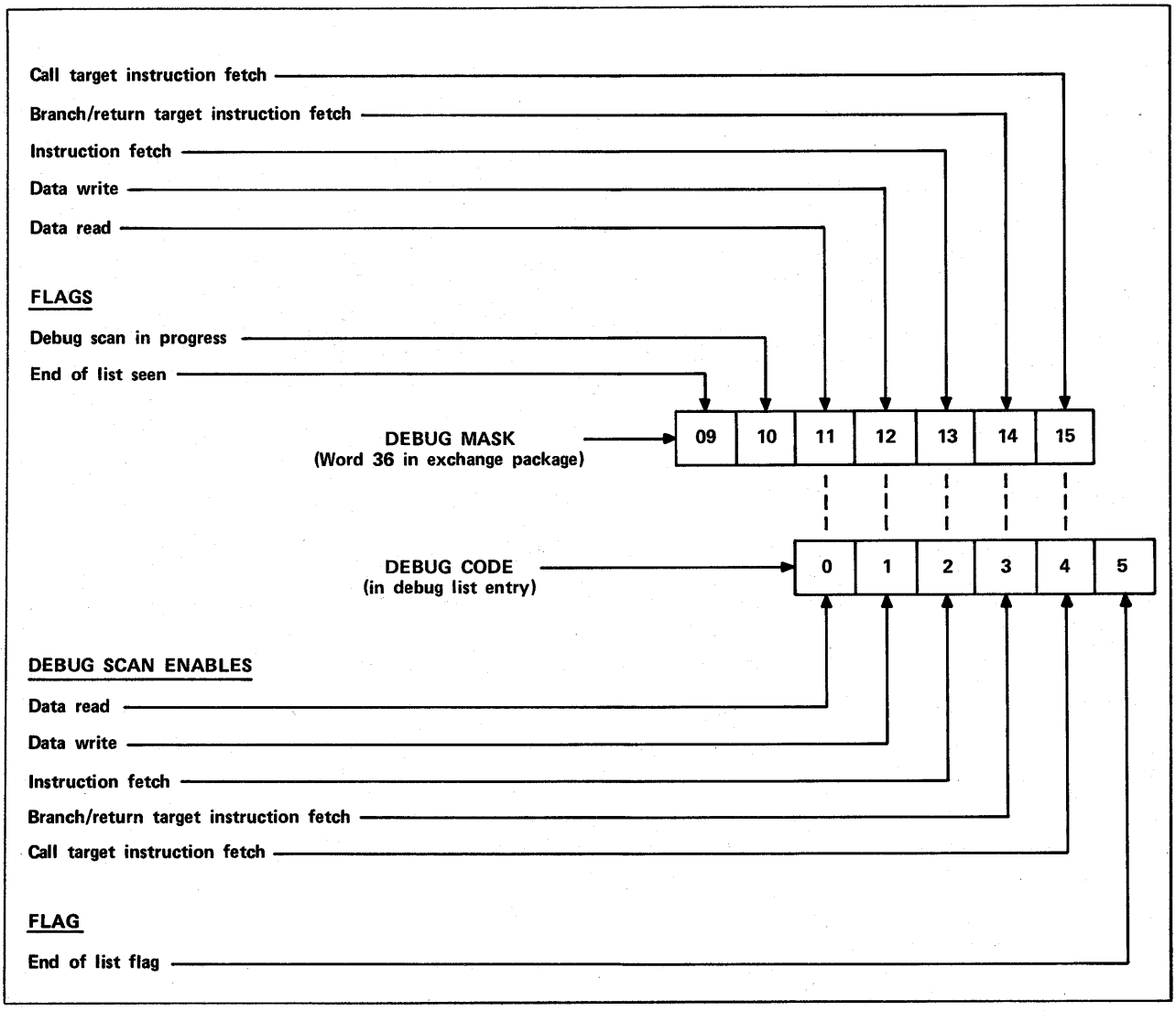


Figure II-2-10. Debug Condition Select

Enabling Debug

The debug operation may be enabled by any of the following:

- Exchange to a Virtual State process where traps are enabled and UMR 56 is set.
- Return (04) to a Virtual State process where the return operation enables traps and UMR 56 is set in the user mask register being loaded.
- Set UMR 56 by way of Copy-To-State-Register instruction when traps are enabled. The debug flags and index must be zero prior to execution of the Copy-To-State-Register instruction or an undefined initial debug scan follows the instruction.
- Enable traps by way of Copy-To-State-Register instruction when UMR 56 is set. The debug flags and index must be zero prior to execution of the Copy-To-State-Register instruction or an undefined initial debug scan follows the Copy-To-State-Register instruction.

Debug Scan Operation

Debug conditions apply to specific instructions as described in table II-2-21. BN(low) and BN(high) are matched against the address of the leftmost byte of a piece of information only, whether it is a word, half word, byte string, or 16-bit instruction. The match is as follows:

$$BN(low) \leq Address \leq BN(high)$$

If BN(low) exceeds BN(high) in any debug list entry, the scan proceeds to the next double-word entry. If either BN(low) or BN(high) bit 32 is set, the comparison results are undefined.

The CP starts the debug list scan (after instruction fetch but before instruction execution) if all of the following conditions exist:

- Traps are enabled.
- Debug mask bit in user mask register is set (UMR 56).
- One or more bits in debug mask register are applicable to the type of access. Refer to table II-2-19.
- End-of-list flag in debug mask register is clear.

When the debug scan initiates, the debug scan-in-progress flag is clear and the debug index register is zero. The debug scan locates a debug list entry by adding the debug list pointer and debug list index registers. The debug scan proceeds (not necessarily in the exact order given) as follows:

1. Set debug scan-in-process flag in debug mask register.
2. Read first half of debug list entry. If end-of-list code, set end-of-list-seen flag in debug mask register.
3. Add one to debug index register and read second half of debug list entry.

4. Set UCR 56 (triggering a trap interrupt) if a debug list match is found as follows:

   - Accessed PVA is within address bracket of debug list entry.
   - One or more debug code bits of debug list entry match the type of access, with the corresponding bit set in the debug mask register.

5. If step 4 triggered a debug interrupt, proceed to step 8.

6. If end-of-list flag is set, or 32 entries have been read, proceed to step 8.

7. Add one to debug index register and repeat from step 2.

8. Clear debug index register, debug scan-in-process flag, and end-of-list-seen flag to complete the debug scan.

9. Execute the instruction triggering the scan.

Debug list scanning prior to instruction execution includes all instruction results except the following (which may occur before the debug scan completes):

1. Setting the page-used bit, either explicitly as in test and set page instruction (16), or implicitly as with any instruction.

2. Setting of condition register bits.

3. Rounding of A0 on call instructions.

4. Storing the environment into SFSA on call instructions (a debug trap also stores the environment into SFSA).

The exception testing and debug scan are not constrained to occur in any given sequence relative to each other. Two or more matches within the same entry produce only one trap. The traps due to execution testing may occur concurrently with a debug trap (several bits set in MCR and/or UCR) or separately, either before or after the debug scan.

Interrupts During Debug Scan

The debug index and flags provide the means for properly initiating, resuming, and terminating debug scan operations, particularly when an instruction's execution has been inhibited by one or more interrupts. These interrupts may either be trap or exchange interrupts.

Exchange interrupts cause the flags and debug index register to be stored in the exchange package, for example, to allow resumption of a partially completed debug entry list scan.

On trap interrupts, the processor retains the flags and index register to allow proper completion of the debug scan upon return from the trap interrupt.

When enabling traps during the processing of a debug trap interrupt, software must not reenable debug to prevent loss of integrity of the interrupted debug scan.

Debug-Software Interaction, Debug Enabled

The following items describe interactions with the debug facility that are available with debug enabled:

- Debug mask bits 11 through 15 of debug mask register may be set or cleared by way of a Copy-To-State-Register instruction; the new bits will be in effect for the debug scan on the instruction following the copy instruction.
- Any copy to the debug flags or index must clear both flags and the index or the following debug scan is undefined.
- UMR 56 may be cleared or traps disabled by way of a Copy-To-State-Register instruction with no scan performed on the instruction following the copy instruction.
- A return instruction disables debug by loading a user mask register with bit 56 clear, or by entering CYBER 170 State. In such a case, no scan is performed on the next instruction.

Debug-Software Interaction, Debug Disabled

When debug is disabled after a debug match (after which an exchange or trap interrupt occurs), the scan-in-progress flag sets and, if applicable, the end-of-list flag sets. In this case, the following software action may be taken through a Copy-To-State-Register instruction after a trap interrupt, or through altering the exchange package in CM after an exchange interrupt:

- Any of the debug mask register bits 11 through 15 may be set or cleared. The new mask bits affect the first debug scan when debug is reenabled for this process.
- Debug flags and index may be cleared to reinitiate the debug scan from the beginning when debug is reenabled.
- Debug index may be modified by multiples of 2 as the final value is greater than or equal to 1 and less than or equal to 61.
- End-of-list-seen flag may be set to terminate the current debug scan when debug is reenabled. The scan-in-progress flag may (but need not) be altered when setting this flag.
- The end-of-list-seen flag may be cleared and the scan-in-progress flag set to continue a scan that terminated. The debug index may also be modified by multiples of 2 as long as the final value is greater than or equal to 1 and less than or equal to 61.

When a debug match is absent, the debug may also be disabled by any of the following:

- Trap interrupt.
- Exchange interrupt.
- Copy-To-State-Register instruction which clears UMR 56 or disables traps.
- Call to CYBER 170 State.
- Return which clears UMR 56.
- Return to CYBER 170 State.

When a debug match is absent, and the debug is disabled by any of the six methods described in the preceding paragraph, only the following software actions may be taken:

- Any of the debug mask register bits 11 through 15 may be set or cleared. The new mask bits affect the first debug scan when debug is enabled for this process.
- The debug flags and index may be cleared to reinitiate the full debug scan when debug is enabled.

Table II-2-21. Debug Conditions (Sheet 1 of 5)

Opcode	Mnemonic	Instruction	Debug Condition
		Bit 0: Data Read	
D0-D8	LBYTS,S	Load bytes, immediate	LO $\leq$ Aj+Xi+D $\leq$ HI
A2	LXI	Load word, indexed	LO $\leq$ Aj+8*Xi+8*D $\leq$ HI
82	LX	Load word	LO $\leq$ Aj+8*Q $\leq$ HI
A4	LBYT,X0	Load bytes	LO $\leq$ Aj+Xi+D $\leq$ HI
88	LBIT	Load bit	LO $\leq$ Aj+Q+X0/8 $\leq$ HI
A0	LAI	Load address, indexed	LO $\leq$ Aj+Xi+D $\leq$ HI
84	LA	Load address	LO $\leq$ Aj+Q $\leq$ HI
80	LMULT	Load multiple	LO $\leq$ Aj+8*Q $\leq$ HI
70	ADDN,Aj,X0	Decimal sum	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI
71	SUBN,Aj,X0	Decimal difference	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI
72	MULN,Aj,X0	Decimal product	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI
73	DIVN,Aj,X0	Decimal quotient	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI
E4	SCIN,Aj,X0	Decimal scale	LO $\leq$ Aj+01 $\leq$ HI
E5	SCLR,Aj,X0	Decimal scale, rounded	LO $\leq$ Aj+01 $\leq$ HI
74	CMPN,Aj,X0	Decimal compare	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI
77	CMPB,Aj,X0	Byte compare	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI
E9	CMPC,Aj,X0	Byte compare, collated	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ak+02 $\leq$ HI LO $\leq$ Ai+D $\leq$ HI
F3	SCNB,X0	Byte scan while nonmember	LO $\leq$ Ak+01 $\leq$ HI LO $\leq$ Ai+D $\leq$ HI

Table II-2-21. Debug Conditions (Sheet 2 of 5)

Opcode	Mnemonic	Instruction	Debug Condition
		Bit 0: Data Read Continued	
E8	TRANB,Aj,X0	Byte translate	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ai+D $\leq$ HI
76	MOVB,Aj,X0	Move bytes	LO $\leq$ Aj+01 $\leq$ HI
ED	EDIT,Aj,X0	Edit	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ai+D $\leq$ HI
75	MOVN,Aj,X0	Numeric move	LO $\leq$ Aj+01 $\leq$ HI
F4	CALDF,Aj,X0	Calculate subscript and add	LO $\leq$ Aj+01 $\leq$ HI LO $\leq$ Ai+D $\leq$ HI
B5	CALLSEG	Call indirect	LO $\leq$ Aj+8*Q $\leq$ HI
04	RETURN	Return	LO $\leq$ A2 $\leq$ HI
06	POP	Pop	LO $\leq$ A2 $\leq$ HI
14	LBSET	Test and set bit	LO $\leq$ Aj+X0/8 $\leq$ HI
B4	CMPXA	Compare swap	LO $\leq$ Aj $\leq$ HI
FA	CMPI,Xi,D	Compare immediate data	LO $\leq$ Ak+01 $\leq$ HI
FB	ADDI,Xi,D	Add immediate data	LO $\leq$ Ak+01
		Bit 1: Data Write	
D8-DF	SBYTS,S	Store bytes, immediate	LO $\leq$ Aj+Xi+D $\leq$ HI
A3	SXI	Store word, indexed	LO $\leq$ Aj+8*Xi+8*D $\leq$ HI
83	SX	Store word	LO $\leq$ Aj+8*Q $\leq$ HI
A5	SBYT,X0	Store bytes	LO $\leq$ Aj+Xi+D $\leq$ HI
89	SBIT	Store bit	LO $\leq$ Aj+Q+X0/8 $\leq$ HI
A1	SAI	Store address, indexed	LO $\leq$ Aj+Xi+D $\leq$ HI
85	SA	Store address	LO $\leq$ Aj+Q $\leq$ HI
81	SMULT	Store multiple	LO $\leq$ Aj+8*Q $\leq$ HI
70	ADDN,Aj,X0	Decimal sum	LO $\leq$ Ak+02 $\leq$ HI

Table II-2-21. Debug Conditions (Sheet 3 of 5)

Opcode	Mnemonic	Instruction	Debug Condition
Bit 1: Data Write Continued			
71	SUBN,Aj,X0	Decimal difference	LO $\leq$ Ak+02 $\leq$ HI
72	MULN,Aj,X0	Decimal product	LO $\leq$ Ak+02 $\leq$ HI
73	DIVN,Aj,X0	Decimal quotient	LO $\leq$ Ak+02 $\leq$ HI
E4	SCLN,Aj,X0	Decimal scale	LO $\leq$ Ak+02 $\leq$ HI
E5	SCLR,Aj,X0	Decimal scale, rounded	LO $\leq$ Ak+02 $\leq$ HI
EB	TRANB,Aj,X0	Byte translate	LO $\leq$ Ak+02 $\leq$ HI
76	MOVB,Aj,X0	Move bytes	LO $\leq$ Ak+02 $\leq$ HI
ED	EDIT,Aj,X0	Edit	LO $\leq$ Ak+02 $\leq$ HI
75	MOVN,Aj,X0	Numeric move	LO $\leq$ Ak+02 $\leq$ HI
B5	CALLSEG	Call indirect	LO $\leq$ AQ+7,mod 8 $\leq$ HI
B0	CALLREL	Call relative	LO $\leq$ AQ+7,mod 8 $\leq$ HI
14	LBSET	Test and set bit	LO $\leq$ Aj+X0/8 $\leq$ HI
B4	CMPXA	Compare swap	LO $\leq$ Aj $\leq$ HI
F9	MOVI,Xi,C	Move immediate data	LO $\leq$ Ak+01 $\leq$ HI
FB	ADDI,Xi,D	Add immediate data	LO $\leq$ Ak+01 $\leq$ HI
Bit 2: Instruction Fetch			

All instruction fetches initiate a debug trap interrupt when the PVA accessed is within an address bracket on the debug list. However:

- The load bytes relative (D0-D7) reference to P+Q is not detected.
- Unimplemented instruction, program error, and execute algorithm is not necessarily detected.
- The descriptors for BDP instructions are not detected.
- The test is applied for each instruction rather than for each instruction word.

Table II-2-21. Debug Conditions (Sheet 4 of 5)

Opcode	Mnemonic	Instruction	Debug Condition
Bit 3: Branch Target Instruction			
94	BRXEQ	Branch on equal	LO ≤ P+2*Q ≤ HI
95	BRXNE	Branch on not equal	LO ≤ P+2*Q ≤ HI
96	BRXGT	Branch on greater than	LO ≤ P+2*Q ≤ HI
97	BRXGE	Branch on greater than or equal	LO ≤ P+2*Q ≤ HI
90	BRREQ	Branch on half word equal	LO ≤ P+2*Q ≤ HI
91	BRRNE	Branch on half word not equal	LO ≤ P+2*Q ≤ HI
92	BRRGT	Branch on half word greater than	LO ≤ P+2*Q ≤ HI
93	BRRGE	Branch on half word greater than or equal	LO ≤ P+2*Q ≤ HI
9C	BRINC	Branch and increment	LO ≤ P+2*Q ≤ HI
9D	BRSEG	Branch on segments unequal	LO ≤ P+2*Q ≤ HI
2E	BRREL	Branch relative	LO ≤ P+2*Xk ≤ HI
2F	BRDIR	Branch intersegment	LO ≤ Aj+2*Xk ≤ HI
98	BRFEQ	FP branch on equal	LO ≤ P+2*Q ≤ HI
99	BRFNE	FP branch on not equal	LO ≤ P+2*Q ≤ HI
9A	BRFGT	FP branch on greater than	LO ≤ P+2*Q ≤ HI
9B	BRFGE	FP branch on greater than or equal	LO ≤ P+2*Q ≤ HI
9E	BROVR	FP branch on overflow	LO ≤ P+2*Q ≤ HI
9E	BRUND	FP branch on underflow	LO ≤ P+2*Q ≤ HI
9E	BRINF	FP branch on indefinite	LO ≤ P+2*Q ≤ HI
04	RETURN	Return	LO ≤ FINAL P ≤ HI
9F	BRCR	Branch on condition register	LO ≤ P+2*Q ≤ HI

Table II-2-21. Debug Conditions (Sheet 5 of 5)

Opcode	Mnemonic	Instruction	Debug Condition
Bit 4: Call Target Instruction Fetch			
B5	CALLSEG	Call indirect	LO $\leq$ CBP $\leq$ HI
B0	CALLREL	Call relative	LO $\leq$ P+8*Q,mod 8 $\leq$ HI

VIRTUAL AND CENTRAL MEMORY PROGRAMMING

Figure II-2-11 shows how a process virtual address (PVA) converts to a system virtual address (SVA), and then to a real memory address (RMA). The operating system provides a segment descriptor table and a system page table to make the conversion possible.

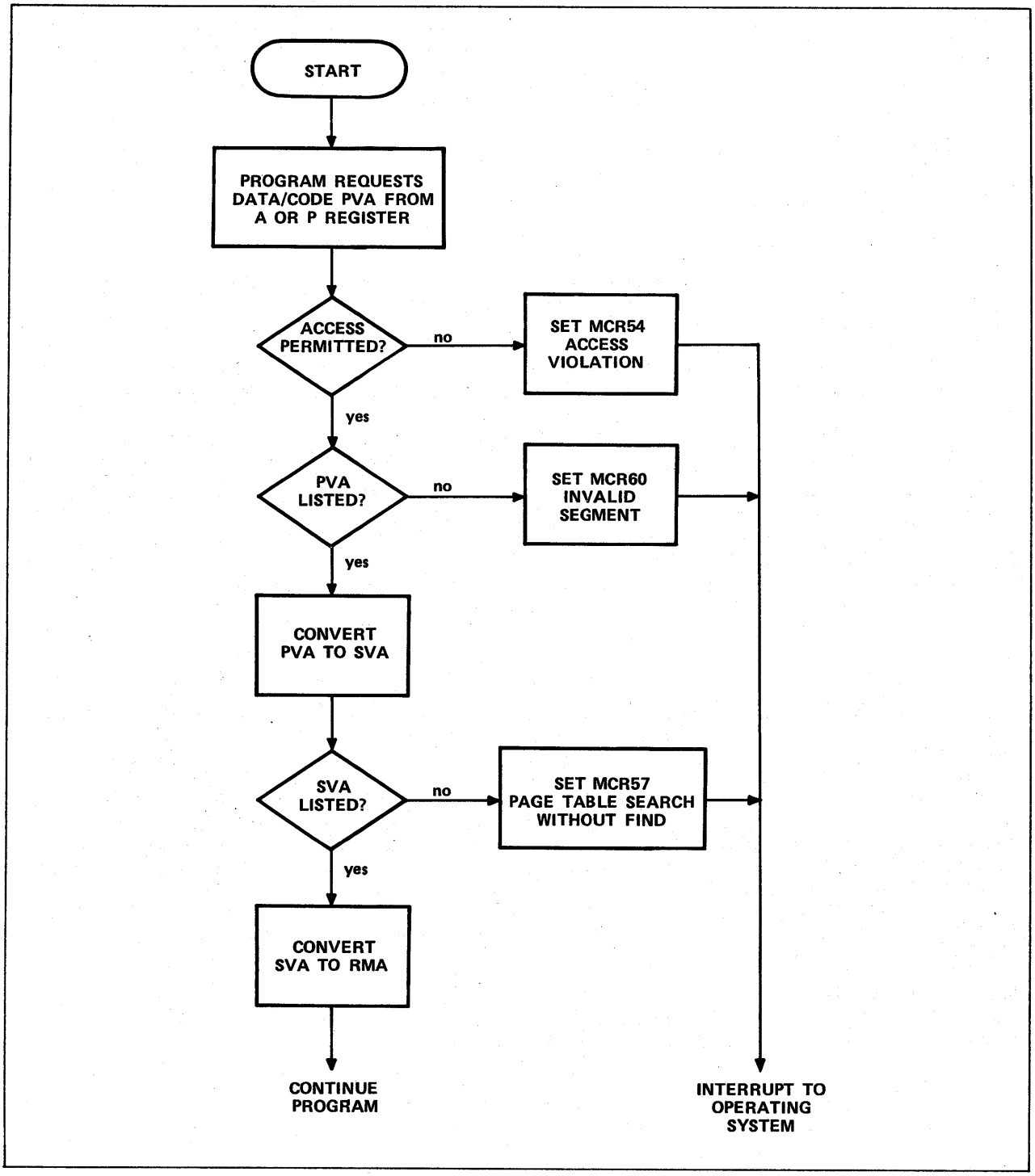
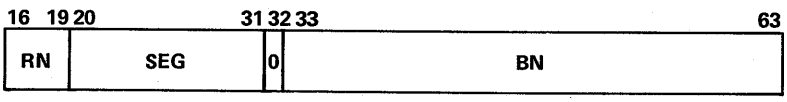


Figure II-2-11. Central Memory Addressing from CP

PROCESS VIRTUAL MEMORY

To the user, memory is a set of segments in process virtual memory space. Each segment is a contiguous byte string of $2^{31}-1$ bytes. A maximum of 4096 segments may exist for each process at any one time. During process execution, the CP presents process virtual addresses (PVAs) for hardware translation, first to system virtual addresses (SVAs), and then to real memory addresses (RMAs) where the requested data resides in CM. The PVA is the address seen by the user and used by executing code. The PVA may identify a P register address or a CM operand address. The PVA format is shown in figure II-2-12.



Field	Name	Description
RN	Ring Number	Access privilege indicator that selectes 1 of 16 rings. Refer to Access Protection in this section for further information.
SEG	Segment	Process segment number. The same shared segment may be addressed by different segment numbers.
BN	Byte number	Byte location within the $2^{31}-1$ bytes in a segment. Bit 32 in the final PVA used is a validity indicator and must be a zero or the PVA is rejected. Bit 32 in an A register may be a one provided indexing or displacement changes this bit to a zero in the final PVA.

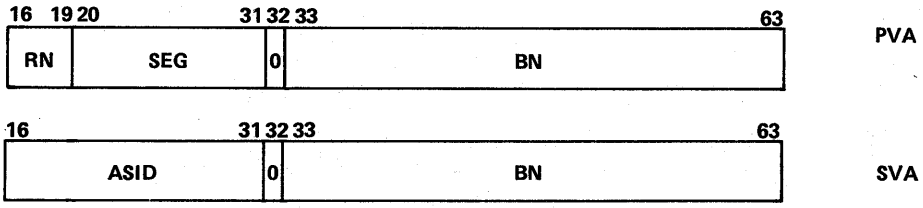
Figure II-2-12. Process Virtual Address (PVA) Format

SYSTEM VIRTUAL MEMORY

The operating system sees virtual memory as a set of system-wide active segments that totals all of the process segments. An active segment is the virtual memory division for uniquely identifying system data. System virtual addresses (SVAs) address the active segments. Figure II-2-13 shows how hardware translates a PVA to an SVA.

In the translation, an active segment identifier (ASID) replaces the RN and SEG fields of the PVA. All segments active in the system are assigned unique ASIDs to ensure unique system data. The 16-bit ASID field in the SVA defines a total of 65,536 active segments that may simultaneously exist in the system. The ASID resides in the segment descriptor table that the operating system maintains for each process. Any number of ASIDs can be listed in various process segment descriptor tables against the same segment. This forms the basis for code sharing.

During the translation, hardware also verifies permission to access the segment containing the SVA. This occurs by way of the access protection attributes listed in the segment descriptor table for that segment.



Field	Name	Description
ASID	Active segment identifier	Global name that uniquely identifies any active segment in the system.
BN	Byte number	Byte location within a segment.

Figure II-2-13. System Virtual Address (SVA) Format

REAL MEMORY

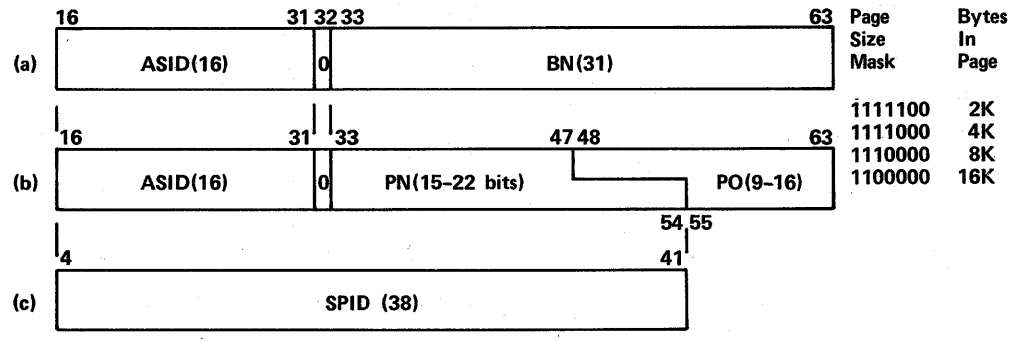
The operating system sees real memory as pages in CM or external mass storage. It maintains the necessary tables to identify, address, and retrieve the stored information when an executing process requests it. A page ranges in size from 2K bytes to 16K bytes, as selected at system initialization via the page size mask register.

CM corresponds with virtual memory by page frames. Page frames are the same size as pages, and provide the means for swapping pages between CM and external mass storage. A page frame may be empty, contain new information being created, or contain a copy of a page from external mass storage that is being examined, modified, or executed.

The operating system manages CM by way of demand paging. In demand paging, the CP retrieves the requested page(s) from external mass storage to CM as needed for process execution. This frees the operating system from having to collect in CM all the pages necessary to complete process execution. In operation, the CP executes a process until a page fault occurs, whereby the CP switches execution to another process while it retrieves the page from external mass storage.

The CP always retrieves a copy of a page, and not the page itself, from external mass storage. Pages in such storage remain unaltered unless or until the CP overwrites the new page back to its location in external mass storage. (Hardware keeps track of pages in CM which are still true copies of pages in external mass storage.) The CP writes pages back to external mass storage to make room for new pages. The operating system identifies candidate pages for transfer by way of an algorithm which determines the least-used and least-recently-used pages.

Hardware uses the system page table in CM (described under Address Tables later in this section) to convert SVAs to RMAs and to complete address translation. In the SVA-to-RMA conversion, hardware uses a hashing algorithm to replace the ASID and PN fields of the SVA with a system/page identifier (SPID) [refer to figure II-2-14]. The SPID resides in the single system page table that the operating system maintains for all processes.

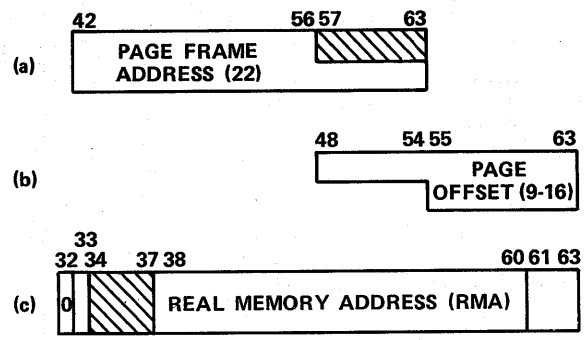


Field	Name	Description
ASID	Active segment identifier	Global name that uniquely identifies any segment active in the system.
BN	Byte number	Byte location within a segment.
PN	Page number	Part of BN which is hashed with the ASID to form the SPID. Its length varies from 15 to 22 bits, depending on system page size.
PO	Page offset	Byte location within a page. Length varies from 9 to 16 bits, depending on system page size.
SPID	Segment/ page identifier	Identifies page in a particular global segment for conversion to RMA.

Figure II-2-14. Segment/Page Identifier (SPID) Format

Once obtained, the SPID provides an index into a list of coincident hashed entries in the system page table. The CP linearly searches up to 32 coincident entries in comparing the true SPID (maintained by the operating system) to the system page table entries. The search continues until a valid page with the requested entry is found, or until 32 entries have been searched. If the requested entry is not found among the 32 entries, a page fault occurs and the CP retrieves the page and accompanying entry from external mass storage.

A successful search of the system page table results in identifying the desired page and the corresponding page frame address listed with the SPID. The page frame address lists the destination of the page in real memory. The page offset (carried directly from the SVA) identifies the memory word containing the requested byte. The rightmost 3 bits select the requested byte within the word. The RMA, formed by the page frame address and the page offset, has a format as shown in figure II-2-15. The conversion of the virtual byte number (BN) to the page number and page offset is shown in figure II-2-16.



Bits	Description
(a)42-63	Page frame address from system page table. Where the page offset has significant bits, the page frame address rightmost bits must be zeros in the system page table.
(b)47-63	Page offset from SVA, unchanged from PVA. Where the page frame address has corresponding significant bits, the page offset leftmost bits are set to zero by the page size mask.
(c)32	Must be zero.
33	Selects the port when CP has two memory ports.
34,35	Reserved.
36,37	Not used.
38-60	CM word address.
61-63	Byte address within a CM word.

Figure II-2-15. Real Memory Address (RMA) Format

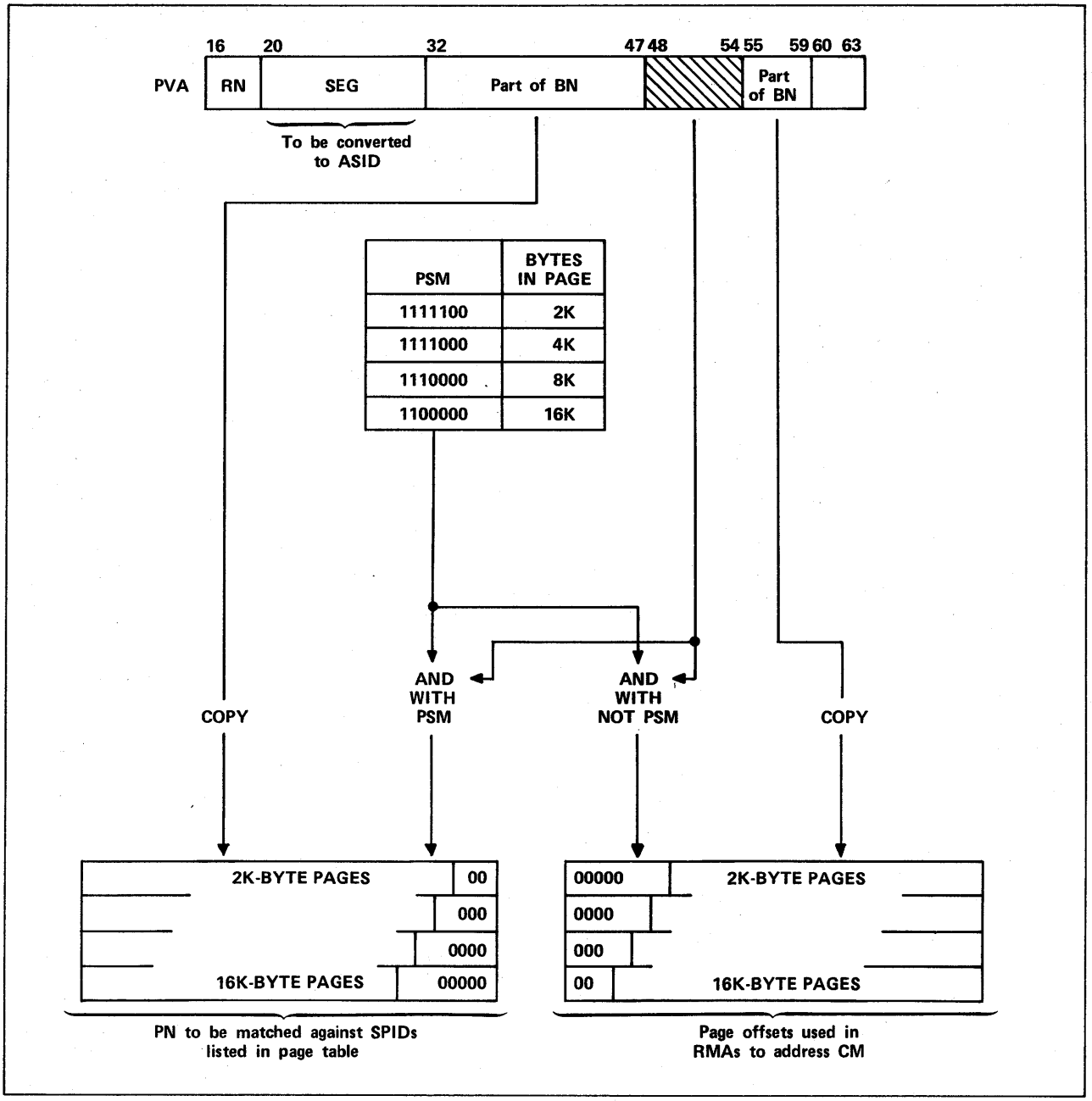


Figure II-2-16. Virtual BN-to-Page Number/Page Offset Conversion

ADDRESS TABLES

The PVA-to-SVA-to-RMA translation with access protection depends upon the operating system keeping the following tables in CM:

- A segment descriptor table (SDT) for each process. Hardware accesses the SDT for converting PVAs to SVAs and for enforcing access protection.
- A system page table (SPT) common for all processes. Hardware accesses the SPT for converting SVAs to RMAs and for keeping track of modified/used pages.

The PVA-to-RMA Conversion with reference to the address tables is shown in figure II-2-17.

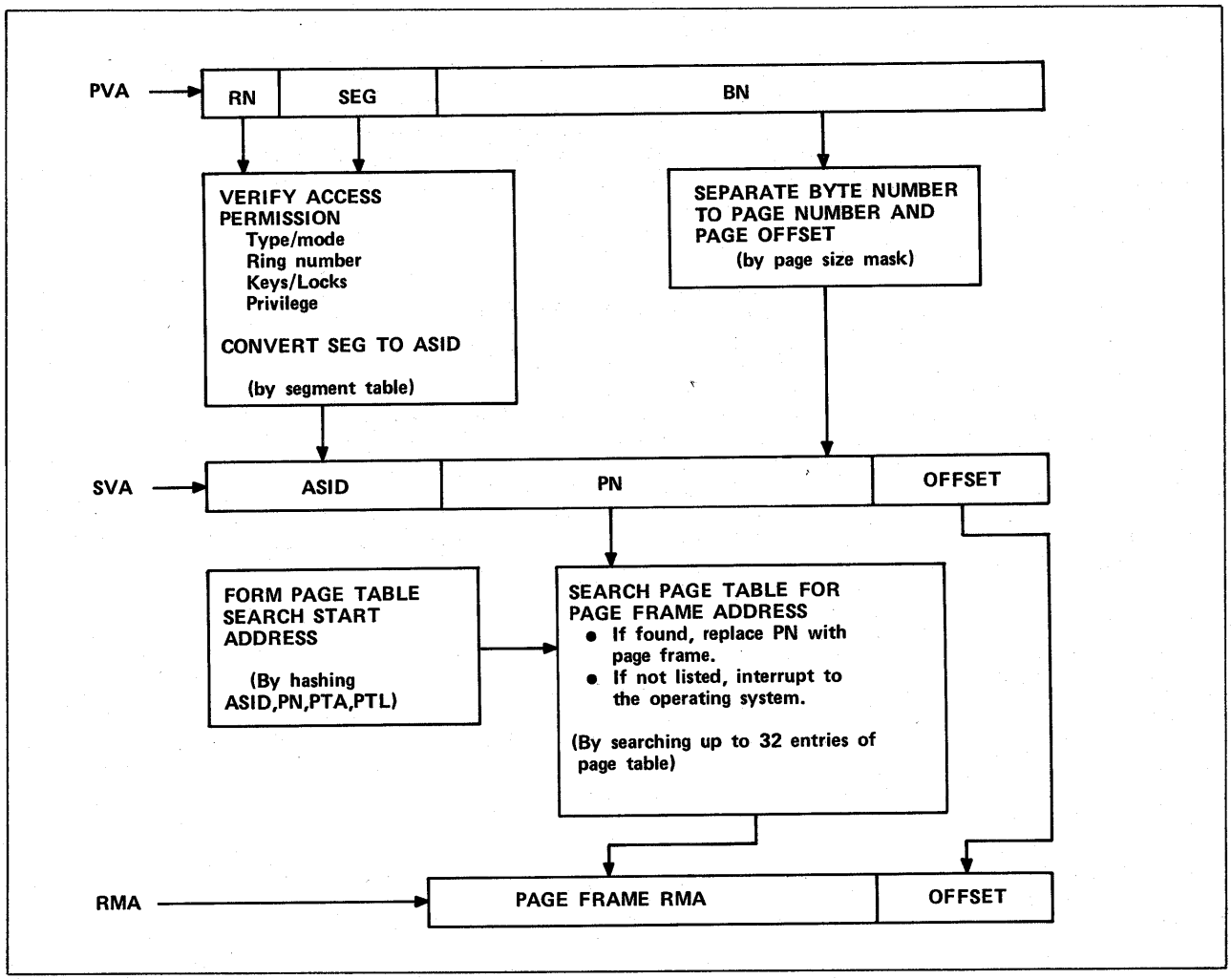


Figure II-2-17. PVA-to-RMA Conversion

Each process obtaining call-indirect and trap-interrupt target addresses requires binding section segments. Hardware accesses the binding segments during call-indirect instructions and trap interrupts to obtain entry point addresses (code base pointers) and additional access protection information.

Segment Descriptor Table

A segment descriptor table (SDT) for each process lists the process segments against the active system segments and provides access protection information. The SDT starts at a word boundary and is defined by the following fields in an incoming exchange package:

- Segment table address (STA): An RMA pointing to the first SDT entry.
- Segment table length (STL): The number of SDT entries, minus 1.

For each process segment, the SDT maintains an entry listing the corresponding ASID and access attributes, and whether or not the segment is a cache bypass segment. Refer to figure II-2-18 for the SDT entry format. The CP reads cache bypass segments directly from CM without copying the accessed information to cache memory. The CP purges the relevant cache entry when it writes cache bypass segments. The CP does not update or purge cache when it reads or writes CM using an RMA stored in hardware such as MPS or JPS.

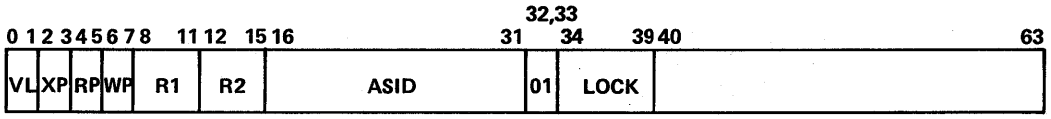
The SDT entry (SDE) lists the ASID corresponding to each segment number. The segment numbers index the table, and combine with the STA to provide the required SDT address. Invalid SDEs can be marked as such; thus, process segment numbers need not form a consecutive set (although this is often the case).

An interrupt condition occurs when:

- Hardware attempts to use a segment number for which there is no valid SDE.
- Hardware attempts to use a segment number which exceeds the segment table length.

Each SDE also lists the segment access protection attributes as established by software convention. (Refer to Access Protection in this section for further information.)

The information in CM which is subject to change without cache update or purge must be kept in cache bypass segments. For example, Virtual State exchange operations to MPS and JPS use RMAs; thus, the CP does not update cache data because cache is organized by SVAs. (CYBER 170 State exchange operations treat target addresses as PVAs; thus, cache is updated.)



Field	Name	Description
VL	Validity	Entry validity indicator 00 Invalid entry. 01 Reserved. 10 Valid entry, regular segment. 11 Valid entry, cache bypass segment.
XP	Execute permission	Permissible access type indicator 00 Nonexecutable segment. 01 Nonprivileged executable segment. 10 Locally privileged executable segment. 11 Globally privileged executable segment.
RP	Read permission	Permissible access type indicator 00 Nonreadable segment. 01 Read controlled by key/lock. 10 Read not controlled by key/lock. 11 Binding section segment, read not controlled by key/lock (may be read as if RP = 10).
WP	Write permission	Permissible access type indicator 00 Nonwritable segment. 01 Write controlled by key/lock. 10 Write not controlled by key/lock. 11 (Reserved).
R1	Ring 1	Ring requirement indicator • For execute access, RN of PVA used may not be less than R1 of the accessed segment's SDE. • For write access, RN of PVA used may not exceed R1 of the accessed segment's SDE.
R2	Ring 2	Ring requirement indicator • For execute access, RN of PVA used may not exceed R2 of the accessed segment's SDE. • For read access, RN of PVA used may not exceed R2 of the accessed segment's SDE.
ASID	Active segment identifier	A global segment name which uniquely identifies each active segment in the system
LOCK	Lock	One of 64 locks. A zero value indicates a no-lock condition.

Figure II-2-18. Segment Descriptor Table Entry Format

System Page Table

The system has a system page table which lists via an SPID each virtual page resident in CM against the physical address of the assigned page frame. This table is defined by the following:

- Page table address (PTA). This is an RMA pointing to the first page table entry, which must be zero, modulo the page table length.
- Page table length (PTL). The page table length in modulo 512 words is as follows.

PTL	Words in Page	Bytes in Page
0000 0000	512	4 096
0000 0001	1 024	8 192
0000 0011	2 048	16 384
0000 0111	4 096	32 768
0000 1111	8 192	65 536
0001 1111	16 338	131 072
0011 1111	32 768	262 144
0111 1111	65 536	524 288
1111 1111	131 072	1 048 576

- Page size mask (PSM). The page size in modulo 512 bytes is as follows.

PSM	Words in Page	Bytes in Page
111 1100	256	2 048
111 1000	512	4 096
111 0000	1 024	8 192
110 0000	2 048	16 384

Page Table Search

Hardware converts SVAs to RMAs by searching up to 32 page table entries for a SPID matching the SVA (ASID-PN) being converted. Figure II-2-19 depicts how hardware obtains a starting address for the 32-entry linear search by combining information from the ASID, page table length, and virtual page number. The technique is called hashing. The page table search is necessary because many SVAs hash to the same page table search starting address.

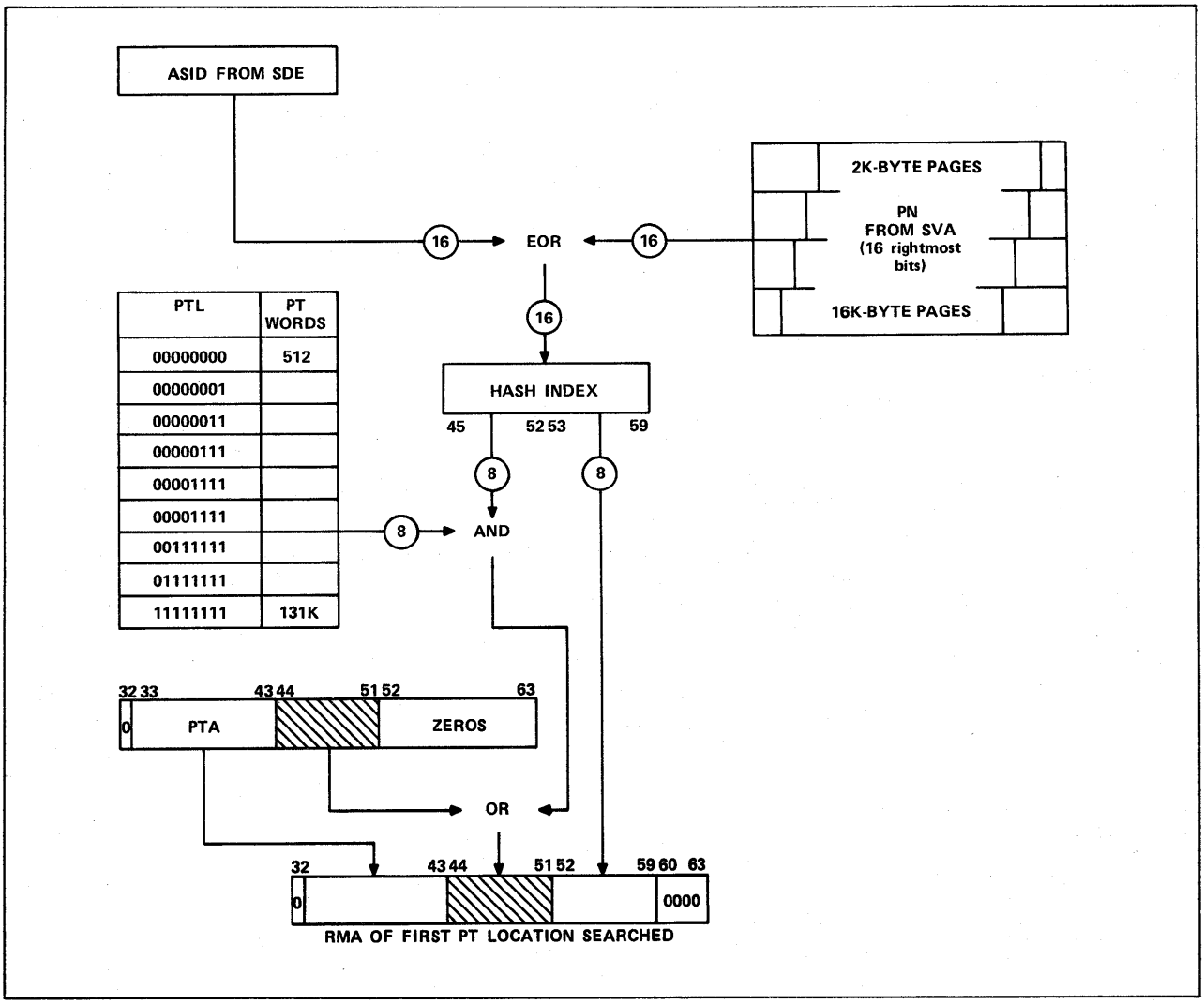


Figure II-2-19. Page Table Search, Start RMA Formation

Page Table Entries

Page table entries (refer to PTE in figure II-2-20) list all pages residing in CM by listing the SPID against the allocated CM page frame (physical) address. The operating system ensures that only one copy of any active page exists in CM at a time. The PTEs described also contain 4 control bits.

PTE Control Fields

The PTE contains 4 control bits: valid (V), continue (C), used (U), and modified (M). Hardware decodes and translates these fields during the page table search as follows:

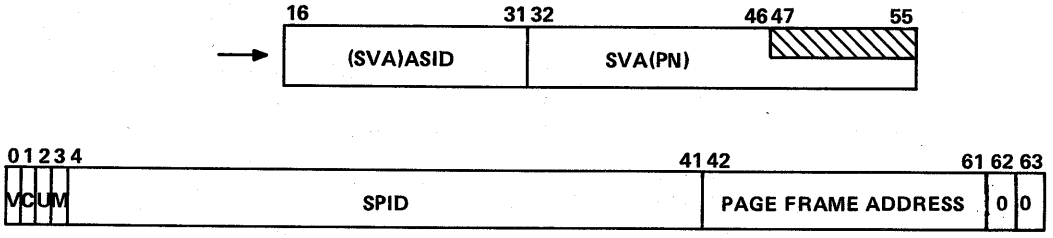
- The valid (V) bit, when set, causes the PTE under examination to be tested for a SVA-RMA match. When clear, that PTE is ignored.
- The continue (C) bit, when set, causes the hardware search to continue with the next PTE. When clear, the hardware search may be halted after testing the current PTE.
- The used (U) bit is set by hardware whenever a PTE is used for address translation. This bit is cleared by software only.
- The modified (M) bit is set by hardware whenever a PTE is used for a write access to indicate that the page has been modified.

PTE Segment/Page Identifier Field

The Segment/Page Identifier (SPID) is the PTE field tested for a match against the 38-bit ASID-PN combination from the SVA. When the page size exceeds 512 bytes, the PN is less than 22 bits. In this case, the unused rightmost bits are zeros to obtain proper alignment.

PTE Page Frame RMA Field

This 22-bit field is the page frame physical starting address. When the page size exceeds 512 bytes, the rightmost address bits must be zeros to obtain proper alignment.



Field	Name	Description
ASID		Active segment identifies part of SVA.
SVA(PN)		Page number part of SVA.
V	Valid	(bit 00) Coded as follows: V = 1 PTE tested. V = 0 PTE ignored.
C	Continue	(bit 01) Coded as follows: C = 1 Search continued. C = 0 Search aborted.
U	Used	(bit 02) Cleared by software only. Indicates whether a page was accessed by the CPU: U = 1 An SVA-PTE match occurred. U = 0 No SVA-PTE match occurred.
M	Modified	(bit 03) Cleared by software only and coded as follows: M = 1 This PTE has been used for a write access since it was entered. M = 0 This PTE has not been used for a write access since it was entered.
SPID	Segment/page Identifier	(38-bits) Identical to the ASID-PN field combination from SVA. Refer to figure II-2-14.
PAGE FRAME ADDRESS		Real memory starting address of the page frame. When the page size exceeds 2K bytes, the rightmost bits of this field are zeros (corresponding to the unused rightmost bits in SPID and SVA-PN).

Figure II-2-20. Page Table Entry Format

Listing of Pages in Page Table

To continuously guarantee the same data for all processes, the operating system ensures that only one copy of a page resides in CM at a time. Therefore, each page has the same page frame address and is listed against a unique ASID in the page table. Also, the operating system lists pages in the page table so that a page table search always results in finding the requested page if it is in CM.

Due to the vast number of possible virtual pages, special techniques are used to list pages in the page table. The system page table length is, typically, 2 to 4 times the number of available page frames, and the 16-bit ASID numbers are assigned nonsequentially to facilitate page listing.

If a vacant spot is not found when attemting to list a page, the operating system takes further action such as cancelling a page to make room for the new page, changing the ASID number, or rearranging CM and enlarging the page table size.

PROCESS BINDING SECTION

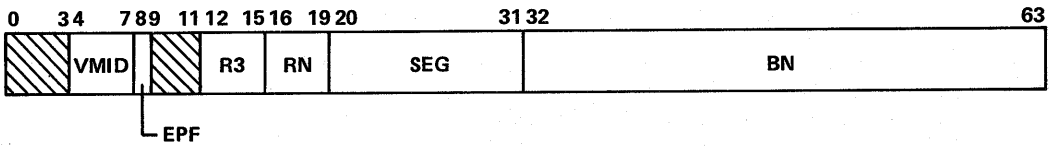
Binding sections bind different segments into one process. Each process has at least one binding section for use by hardware during the call indirect (CALLSEG,B5) instruction and during trap interrupts.

A binding section used for such purposes resides in a segment for which the SDE(RP) = 11; such a segment represents a binding segment and lists entry points (code base pointers, CBP) into called code segments. The 64-bit CBPs (figure II-2-21) reside at a word boundary and are addressed by PVAs as follows:

- For use by CALLSEG:
  The relevant CBP is addressed by (Aj+8*Q) from the instruction.

- For use by trap interrupts:
  The relevant CBP is addressed by the trap pointer (TP) field from the exchange package.

A called procedure may have, and a trap interrupt target procedure always has, its own binding section. In such a case, the external procedure flag (EPF) sets in the relevant CBP. After a CBP with its EPF set is accessed, the word stored following that CBP loads into A3 as the new binding section pointer.



Field	Name	Description
VMID	Virtual machine identifier	Specifies the state of the CP after a call indirect instruction or after a trap interrupt, as follows: VMID = 0000 Virtual State. VMID = 0001 CYBER 170 State.
EPF	External procedure flag	When EPF = 1, hardware loads the word stored following the CBP as a new binding section pointer into register A3. When the CBP is used for trap interrupts, this bit must be a 1.
R3	Call limit ring number	Highest ring number from which a call indirect instruction to the listed target PVA may be issued. Initial P(RN) and Aj(RN) must both not exceed R3.
RN	CBP ring number	When zero, causes a ring-number zero-exception condition, setting MCR bit 60. When nonzero, prevents this exception with no other effects.
SEG, BN	Segment Number, Byte number	Part of PVA of entry point into branched-to code.

Figure II-2-21. Code Base Pointer Format

ACCESS PROTECTION

Access protection is by way of segments, based on the following elements:

- Process segment table
  The process segment table defines the process address space. A process may access only the segments listed in its segment descriptor table. Refer to Segment Descriptor Table in this section.

- Ring structure
  All PVAs used have a 4-bit ring number (RN) field specifying an access privilege from 1 to 15. (A lower RN in a PVA indicates a higher PVA access privilege.) All SDEs and CBPs list access privilege requirements. (A lower RN listed indicates a higher privilege requirement.)

- Type of access
  SDE control fields permit the type of access attempted (read, write, or execute). Refer to figure II-2-16.

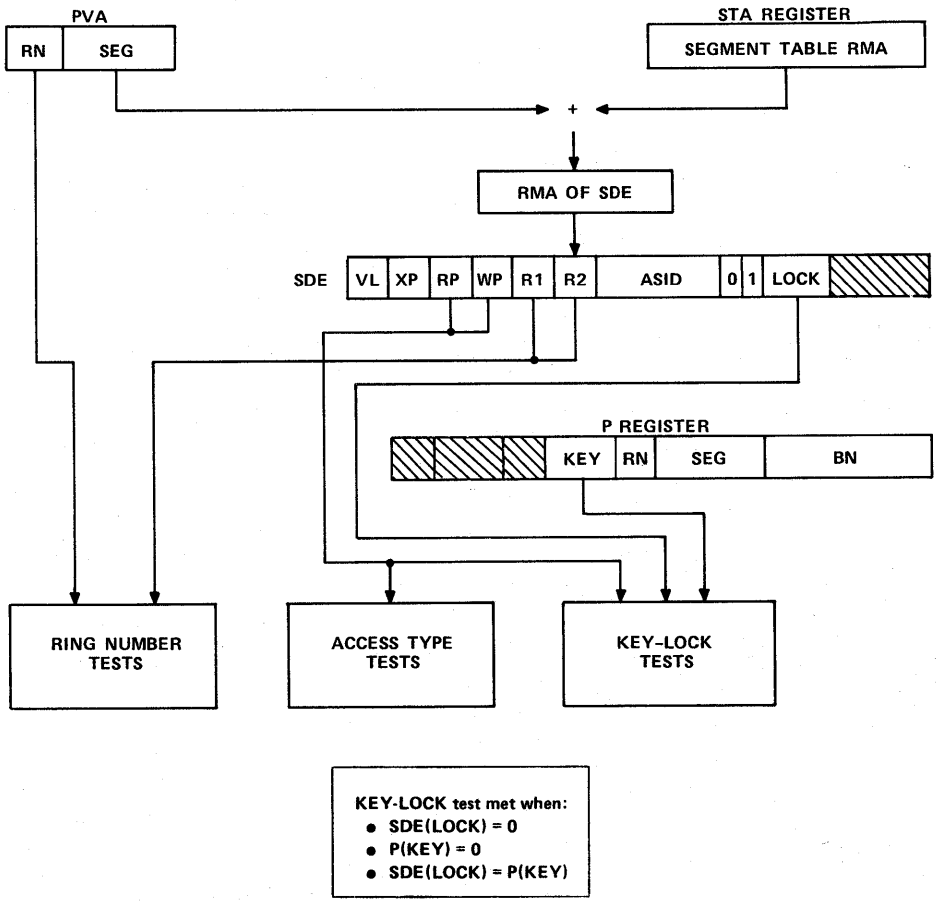
- Execute access privilege
  Some system instructions require special codes in SDE execute permission control fields. (A local/global execution privilege requirement.)

- Execute access mode
  Some system instructions may execute only in Virtual State monitor mode.

- Keys and locks
  SDE control fields may further specify that a 6-bit key number in the P register must equal a 6-bit lock number in the SDE.

- Code base pointers
  Call indirect instructions and trap interrupts use additional access protection attributes in CBPs, which must be listed in binding section segments.

Figure II-2-22 and II-2-23 illustrate access protection as it pertains to the PVA-to-SVA conversion for the read/write and execute cases, respectively.

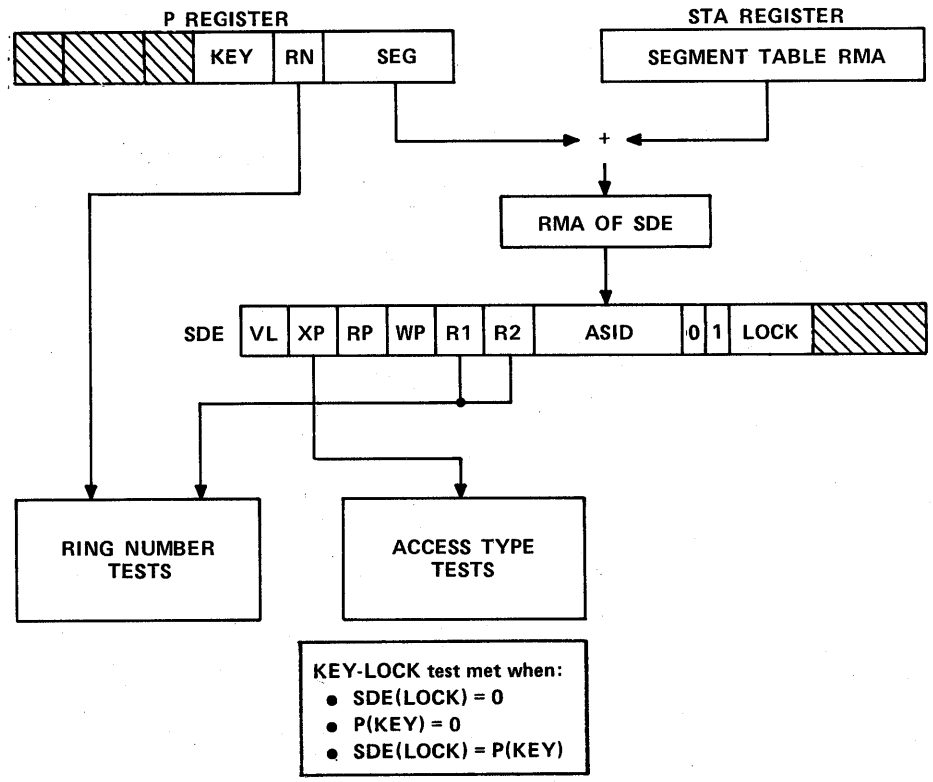


General access requirements:
PVA(SEG) $\leq$ STL, SDE(VL = 10/11) SDE ADRS BIT 32 = 0

Read access requirements:
PVA(RN) $\leq$ SDE(R2)
SDE(RP) = 10/11
SDE(RP) = 01, KEY - LOCK test met

Write access requirements:
PVA(RN) $\leq$ SDE(R1)
SDE(WP) = 10
SDE(WP) = 01, KEY - LOCK test met

Figure II-2-22. PVA-to-SVA Conversion, Read/Write



General access requirements:
PVA(SEG) $\leq$ STL, SDE(VL) = 10/11 SDE ADRS BIT 32 = 0.

Execute access requirements:

- SDE(R1) $\leq$ P(RN) $\leq$ SDE(R2).
- When segment supplies unprivileged instructions only, SDE(XP) = 01.
- When segment supplies unprivileged and local privileged instructions, SDE(XP) = 10.
- When segment supplies unprivileged, local privileged and global privileged instructions, SDE(XP) = 11.

Figure II-2-23. PVA-to-SVA Conversion, Execute

Ring Structure

The ring hierarchy controls read, write, and execute accesses to a segment, as follows:

- All PVAs have a 4-bit ring number (RN) field specifying an access privilege from 1 to 15.
- R1 and R2 fields in SDEs, and the R3 field in CBPs, specify (PVA)RN requirements for access.

Ring Voting

During certain conditions, the RN loaded into an A register is the largest of the following:

- The RN initially in the A register.
- The RN accessed in memory.
- The R1 of the SDE used to access the RN in memory.

Such conditions and the resulting RN loaded are as follows:

- Load AK instructions (A0, 84) load a new Ak(RN) which is the largest of the following:
  1. Initial Aj(RN).
  2. SDE(R1) addressed by initial Aj(SEG).
  3. Aj(RN) from memory.
- Load multiple instruction (80) loads new A(RN)s which are the largest of the following:
  1. Initial Aj(RN).
  2. SDE(RN) addressed by initial Aj(SEG).
  3. The relevant A(RN) from memory.
- Call indirect instruction (B5), when the caller's CBP(EPF) = 1, loads a new A3(RN) which is the largest of the following:
  1. New CBP(RN) from (Aj+8*Q+8).
  2. New P(RN).
- Return instruction (04) loads new A(RN)s which are the largest of the following:
  1. Initial A2(RN).
  2. SDE(RN) addressed by initial A2(SEG).
  3. A(RN) from SFSA in memory.

  When the return instruction loads new P(RN), any A(RN) not specified for loading by the SFSA descriptor are set to the largest of the following:
  1. Initial A(RN) of the relevant A register.
  2. New P(RN).

Effect of RN = 0

RN = 0 can serve as a flag to the operating system to link segments on a demand basis. RN = 0 causes an interrupt condition (setting MCR bit 60), when detected as follows:

- Call indirect instruction (B5) and CBP(RN) = 0.
- Return instruction (04) and an A(RN) in memory (specified for loading by SFSA descriptor) is zero.
- Pop instruction (06) and A1(RN) or A2(RN) in SFSA is zero.
- Load Ak instructions (80, 84) and new Ak(RN) in memory are zero.
- Load multiple instruction (80) and any A(RN) accessed in memory is zero.
- Trap interrupt and CBP(RN) is zero.

No test occurs for RN zero when the A registers are loaded by an exchange operation, or when an A register serves as an address source to access memory.

RN for Read/Write Access

PVAs used for read/write are in A registers and must have RN as follows:

- For a write access, the A(RN) used may not exceed the accessed segment's SDE(R1):

  $A(RN) \leq SDE(R1)$
  (write limit test).

- For a read access, the A(RN) used may not exceed the accessed segment's SDE(R2):

  $A(RN) \leq SDE(R2)$
  (read limit test).

RN for Execute Access

The RN for execute access are verified only for operations having the capability to switch segments. The PVA used may originate from an A register, from a CBP, or from the P register. PVA(RN) requirements are as follows:

- For accessing a branch intersegment target instruction, initial P(RN) and Aj(RN) must fall within the execute bracket given by target segment's SDE(R1) and SDE(R2):

  $SDE(R1) \leq P(RN) \leq SDE(R2)$
  (prevents branches to outside the execute ring bracket of the target segment).
  $SDE(R1) \leq Aj(RN) \leq SDE(R2)$
  (prevents branches to outside the execute ring bracket of the target segment).

- For access to a call indirect target instruction, initial P(RN) must not exceed R3 of CBP used, and also not less than R1 of SDE for target instruction. Refer to figure II-2-24. Also, the initial Aj(RN) must not exceed R3 of CBP used:

  $P(RN) \leq CBP(R3)$
  (a limit set by the operating system).
  $P(RN) \geq SDE(R1)$
  (prevents outward calls).
  $Aj(RN) \leq CBP(R3)$
  (a limit set by the operating system).

- For accessing a trap interrupt target instruction, initial P(RN) must not exceed R3 of CBP used, and also not less than R1 of SDE for target instruction. Also, the trap pointer (TP) ring number must not exceed R3 of CBP used:

  $P(RN) \leq CBP(R3)$
  (a limit set by operating system).
  $P(RN) \geq SDE(R1)$
  (prevents outward trap interrupts).
  $TP(RN) \leq CBP(R3)$
  (a limit set by operating system).

- For access to a return target instruction, P(RN) loaded from SFSA may not be less than initial previous save area pointer A2(RN):

  Initial $A2(RN) \leq$ final P(RN)
  (prevents user from setting P(RN) in SFSA for an inward return as user cannot diminish A2(RN)).

RN Effect on Pop Instruction

A pop instruction loads A1, A2, CFF, and OCF from SFSA, and updates TOS pointer. It does not alter P or A0, and must not alter the ring number in which the stack resides. The entire initial A2 must equal A0 in SFSA, and the initial A2(RN) must equal P(RN). The tests are:

Initial A2 = A0 in SFSA

Initial A2(RN) = P(RN)
(which equals A0(RN) by the previous test).

Effect of RN Violations

Ring number violations and the detection of RN = 0 have effect as follows:

Operation	Violation	Effect
Write	Aj(RN) > SDE(R1)	Access violation (MCR 54)
Read	Aj(RN) > SDE(R2)	Access violation (MCR 54)
Branch Intersegment	SDE(R1) > P(RN) > SDE(R2)	Access violation (MCR 54)
Call Indirect	P(RN) > CBP(R3)	Access violation (MCR 54)
	Aj(RN) > CBP(R3)	Access violation (MCR 54)
	P(RN) < SDE(R1)	Outward call (MCR 61)
	CBP(RN) = 0	RN zero (MCR 60)
Return	A2(RN) > P(RN) in SFSA	Inward return (MCR 61)
	Any A(RN) from SFSA = 0	RN zero (MCR 60)
Pop	Initial A2 ≠ A0 in SFSA	Environment specification error (MCR 55)
	Initial A2(RN) ≠ P(RN)	Interring pop (UCR 52)
	A1(RN) from SFSA = 0	RN zero (MCR 60)
	A2(RN) from SFSA = 0	RN zero (MCR 60)
Load A	Accessed A(RN) = 0	RN zero (MCR 60)
Trap interrupt	P(RN) > CBP(R3)	Access violation (MCR 54)
	TP(RN) > CBP(R3)	Access violation (MCR 54)
	P(RN) < SDE(R1)	Outward call (MCR 61)
	CBP(RN) = 0	RN zero (MCR 60)

Execute Access Privilege/Mode

Execution of some system instructions may occur only in a Virtual State monitor process, or require global/local privilege. An instruction is globally privileged when it is fetched from a segment for which the SDE(XP) is 11. An instruction is locally privileged when it is fetched from a segment for which the SDE(XP) is 10. An instruction which has global privilege also has local privilege. In some cases the requirements are dependent on an instruction parameter. Such system instructions are listed in table II-2-22.

Table II-2-22. System Instruction Privilege and Mode

Instruction	Privilege Required	Mode Required	Effect of Violation
Interrupt processor(03)	Global	Any	UCR 48
Copy to state register			
60-7F	None	Monitor	MCR 51
80-BF	Global	Any	UCR 48
C0-DF	Local/global	Any	UCR 48
Return(04) with SFSA(VMID) = 1	Global		MCR 55
		Job	Undefined operation
Load page table index(17)	Local/global	Any	UCR 48
Branch, alter condition register(9F)			
k = 0/1/8/9	None	Monitor	MCR 51
Purge buffer k (05)			
k = 0/1/2/8-F	Local/global	Any	UCR 48

Notes: UCR 48 Privileged instruction fault.
MCR 55 Environment specification error.
MCR 51 Instruction specification error.

Keys/Locks

The key/lock access protection mechanism is independent of other access protection facilities, and consists of the following:

- Each SDE has a 6-bit field specifying a lock.
- The P register has a 6-bit field specifying a key.

Unlike the ring mechanism, the key-lock mechanism is not hierarchical. For access permission, a specified lock requires an exact match with the key tested, except as follows:

- A zero lock is a no-lock condition accessible by any key.
- A zero key is a master key opening all locks.

Key/lock tests perform under the following conditions:

- For read access when the accessed segment's SDE(RP) = 01.
- For write access when the accessed segments SDE(WP) = 01.

Any segment, except a binding segment used during call indirect access, can be locked as follows:

- When LOCK = 0, the segment is unlocked.
- When LOCK ≠ 0, the segment is locked.

Read or write access to a segment is permitted only when the P register providing the PVA of the instruction attempting the access has a KEY as follows:

- To access a locked segment, P(KEY) must exactly equal the accessed segment's SDE(LOCK), P(KEY) must be a master key, or the segment's SDE(LOCK) must be a no-lock.

Call indirect/relative instructions always set the final P(KEY) equal to the accessed segment's SDE(LOCK). The call indirect access requirements are shown in figure II-2-24.

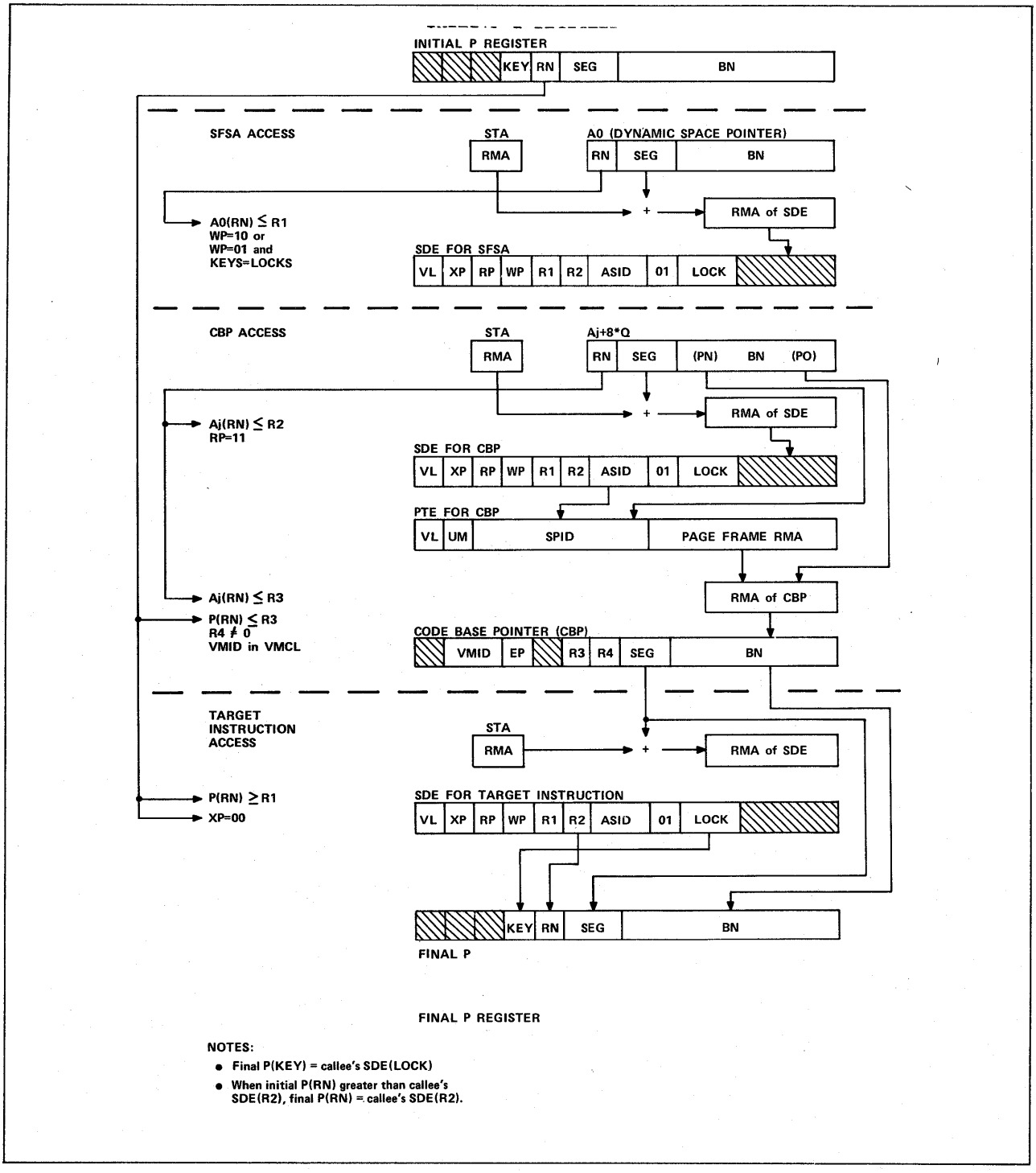


Figure II-2-24. Call Indirect Access Requirements

INTERSTATE PROGRAMMING

The CYBER 170 State environment is a substate of Virtual State job mode, within which the CP uses CYBER 170 State compatible instructions, exchanges, and data formats.

This subsection contains some general aspects of the CYBER 170 State and a detailed description of interaction between the CYBER 170 State and Virtual State. The CYBER 170 State is described in detail in the appropriate computer system (CYBER 170 State) hardware reference manual listed in the preface.

The general characteristics of the CYBER 170 State are as follows:

- The CYBER 170 State user sees the P register as a CYBER 170 State register. Hardware, however, treats the P register the same as in Virtual State and performs the PVA-to-SVA-to-RMA conversion with access protection.
- The CYBER 170 State user sees CM addresses as CYBER 170 State addresses. Hardware, however, treats these as PVAs and fills in the missing RN and SEG fields from P register data.
- The entire CYBER 170 State memory exists in one system virtual segment with an ASID of $FFFF_{16}$.
- Virtual pages into which PPs write 60-bit words map 1:1 into real memory.
- Virtual State procedures execute the CYBER 170 State compare/move instructions (464 through 467) by way of trap interrupts.
- Virtual State procedures handle some CYBER 170 State exception conditions, such as system or job timer interrupts, hardware errors, or conditions which halt model 173. For the latter case, refer to the CYBER 170 Model 173 hardware reference manual listed in the preface.

OPERATION IN CYBER 170 STATE

Virtual State programs establish the CYBER 170 State environment and provide recovery facilities for all hardware and some software errors occurring in CYBER 170 State. The general operation is as follows:

- Virtual State programs build the page table, segment table, and exchange packages for the CYBER 170 State environment.
- A Virtual State program switches the CP to CYBER 170 State through one of the following:

  An exchange from Virtual State monitor mode to Virtual State job mode, CYBER 170 State monitor/job mode.

  A call/return within Virtual State job mode, from Virtual State to CYBER 170 State monitor/job mode.

- The CP runs in CYBER 170 State, with exchanges between CYBER 170 State monitor and job modes, until one of the following occurs:

  An exchange interrupt to Virtual State monitor mode.

  A trap interrupt within Virtual State job mode, from CYBER 170 State to Virtual State.

- Virtual State programs determine the cause of the exchange or trap interrupt and take appropriate action.

MEMORY ADDRESSING IN CYBER 170 STATE

The CM space allocated to CYBER 170 State is addressed as a single segment. An ASID of $FFFF_{16}$ is globally reserved for this segment. For models 835, 845, and 855, hardware uses this ASID value for cache invalidation, described under Cache Invalidation in CYBER 170 State in this section.

Hardware treats all CM addresses supplied by CYBER 170 State programs as PVAs and converts these first to SVAs, and then to RMAs. The P register provides the key, ring number, and segment number for all CM accesses. The PVA-to-RMA conversion includes instruction fetch, load/store, extended memory transfers, and CYBER 170 State exchanges to MA, Bj+K, or R+A.

The operating system supplies the P register KEY, RN, and SEG fields to make the PVA-to-RMA conversion possible; these cannot be changed by CYBER 170 State programs. The entire P register, including the KEY, RN, and SEG fields, loads as follows:

- During Virtual State monitor mode-to-CYBER 170 State monitor/job mode exchanges from the exchange package at (JPS). The exchange package used is specially formatted for such an exchange.

- During calls from Virtual State job mode to CYBER 170 State job/monitor mode, from the code base pointer.

- During returns from Virtual State job mode to CYBER 170 State job/monitor mode, from the stack frame save area.

CACHE INVALIDATION IN CYBER 170 STATE (MODELS 835, 840, 845, 850, 855, AND 860 ONLY)

When a PP writes a 60-bit word into CM, cache memory is updated by cancelling any copy of that word in cache memory. The RMA used to write CM is also supplied to cache memory. Hardware treats this RMA as a SVA with an ASID of $FFFF_{16}$.

To fulfill the requirements of cache invalidation addressing, the operating system assigns an ASID of $FFFF_{16}$ for the virtual memory segment used as CYBER 170 State memory, and must also map the virtual pages into which PPs write 60-bit words, 1:1, into real memory. In such a case, the RMA supplied is numerically equal to the SVA information necessary to locate the addressed word in cache memory. The operating system maps CYBER 170 State address 0 into BN 0 of system virtual segment $FFFF_{16}$.

Cache is not purged when PPs write 64-bit words into CM. Therefore, it is possible to have other than 1:1 mapping of CM for CYBER 170 State PP-write-to-CM instructions in conjunction with software cache invalidation.

STATE-SWITCHING OPERATIONS

Figure II-2-25 depicts the state-switching operations. The CP switches states when its virtual machine identifier (VMID) changes states through the following:

- Exchanges between Virtual State monitor mode and CYBER 170 State. The new VMID loads from the exchange package.
- Calls within Virtual State job mode, from Virtual State to CYBER 170 State. The new VMID loads from the code base pointer.
- Returns within Virtual State job mode, from Virtual State to CYBER 170 State. The new VMID loads from the stack frame save area.
- Trap interrupts within Virtual State job mode, from CYBER 170 State to Virtual State. The new VMID is loads from the code base pointer addressed by the trap pointer.

Virtual State Monitor Mode-to-CYBER 170 State Exchange

The interstate exchange (02) instruction executes the same as a Virtual State exchange instruction, except the incoming exchange package is formatted as shown in figure II-2-26.

CYBER 170 State-to-Virtual State Monitor Mode Exchange

An exchange from a CYBER 170 State process to Virtual State monitor mode is through an interstate exchange interrupt, which executes the same as a Virtual State exchange interrupt, except the outgoing interstate exchange package is formatted as shown in figure II-2-27. Such an interrupt initiates as follows:

- By conditions which set a bit in the monitor or user condition register (when enabled).
- By software conditions which cause a model 173 CP to halt. Such conditions set the exit mode halt flag in the interstate exchange package at (JPS). Refer to the CYBER 170 Model 173 hardware reference manual listed in the preface.
- By a Central Exchange Jump (013) instruction when the CYBER 170 State monitor flag is set.

This exchange does not update the CYBER 170 State exchange package. Virtual State monitor mode software may, however, perform the update. Refer to the operating systems manual listed in the preface for further information.

Exchanges Within CYBER 170 State

Within the CYBER 170 State, CYBER 170 State-compatible exchange jumps and interrupts occur between the CYBER 170 State monitor and job modes. Refer to the appropriate computer system (CYBER 170 State) hardware reference manual listed in the preface. Such exchanges use the exchange package format shown in figure II-2-27. The interstate exchange package is not altered by this exchange.

Call from Virtual State to CYBER 170 State

The interstate call indirect instruction (B5) executes identical to a Virtual State call indirect instruction, except the stack frame save area (SFSA) has the format shown in figure II-2-28. The operating system arranges the Virtual State A and X registers so the required parameters pass to the CYBER 170 State procedure (the call instruction leaves these registers unchanged in hardware). Figure II-2-28 also shows the register formats in hardware. The call instruction does not store the user and monitor condition registers in SFSA.

The BN of P supplied by the code base pointer is the P + RAC of the called procedure target address; BN bits 32 through 39 and 61 through 63 must be zeros.

Trap Interrupt from CYBER 170 State to Virtual State

The interstate trap interrupt causes the same action as the Virtual State trap interrupt described in this section, except the SFSA has the format shown in figure II-2-28. This trap interrupt occurs when traps are enabled and a bit sets in the user condition register as listed in table II-2-20.

Bits 0 through 3 of the X register fields in SFSA are undefined. The VMID and the P register ring number load from the code base pointer addressed by the trap pointer.

The BN of P stored in SFSA is the P + RAC of the interrupted procedure; BN bits 32 through 39 and 63 are zeros, and bits 61 and 62 denote the parcel address within the instruction word.

Return from Virtual State to CYBER 170 State

The interstate return instruction (04) executes the same as the Virtual State return instruction, except the SFSA from which the CYBER 170 State registers are restored is formatted as shown in figure II-2-28. The return instruction with VMID = 1 in SFSA is a global-privileged instruction.

The BN of P loaded from SFSA must be P+RAC into the CYBER 170 State process, where BN bits 32 through 39 and 63 are zeros, and bits 61 and 62 denote the parcel where execution starts.

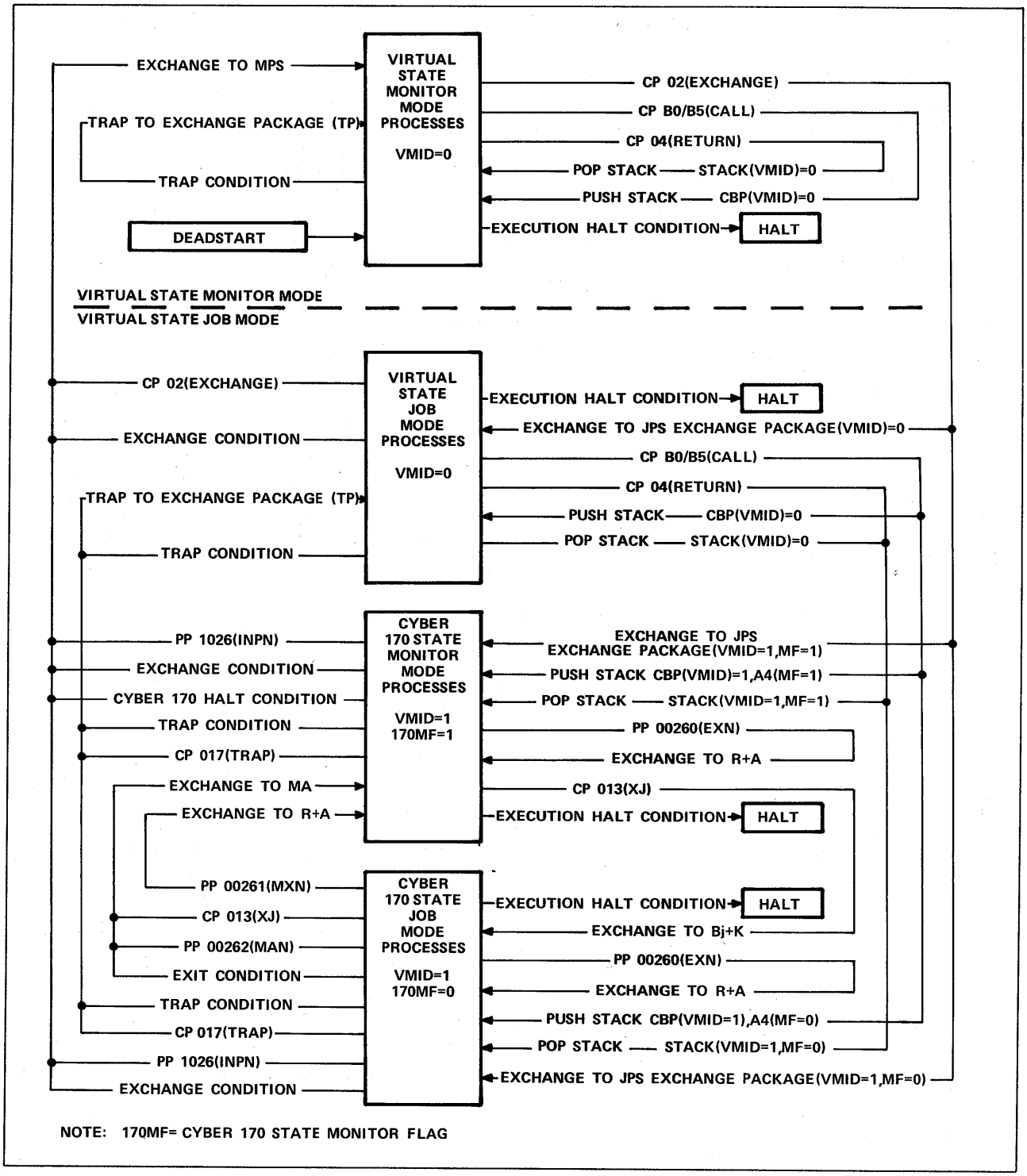


Figure II-2-25. Interstate Calls, Returns and Interrupts

EXCHANGE PACKAGES USED IN CYBER 170 STATE

The CP uses two types of exchange packages with CYBER 170 State operations:

- Exchange package for interstate exchanges.
- Exchange package for exchanges within CYBER 170 State.

Before Virtual State monitor mode initiates an exchange to CYBER 170 State monitor/job mode, the following exchange packages must be ready in CM:

- An interstate exchange package for exchanging from Virtual State monitor mode to CYBER 170 State monitor/job mode.
- An interstate exchange package for exchanging back to Virtual State monitor mode.
- A CYBER 170 State exchange package for a possible immediate exchange to CYBER 170 State monitor mode (when exchanging from Virtual State monitor mode to CYBER 170 State job mode).

Interstate Exchange Package

The job process state (JPS) pointer locates the first word in this exchange package. The exchange package format is shown in figure II-2-26. The format is a modified format of the exchange package used within Virtual State. The CYBER 170 State A, B, X, and other registers reside in locations occupied by the Virtual State A and X registers in the Virtual State exchange package. Therefore, after a trap interrupt, the Virtual State program entered has access to these CYBER 170 State registers by accessing the registers as if they were Virtual State registers.

The following paragraphs describe the modified fields of the exchange package. For other fields, refer to Exchange Package in this section.

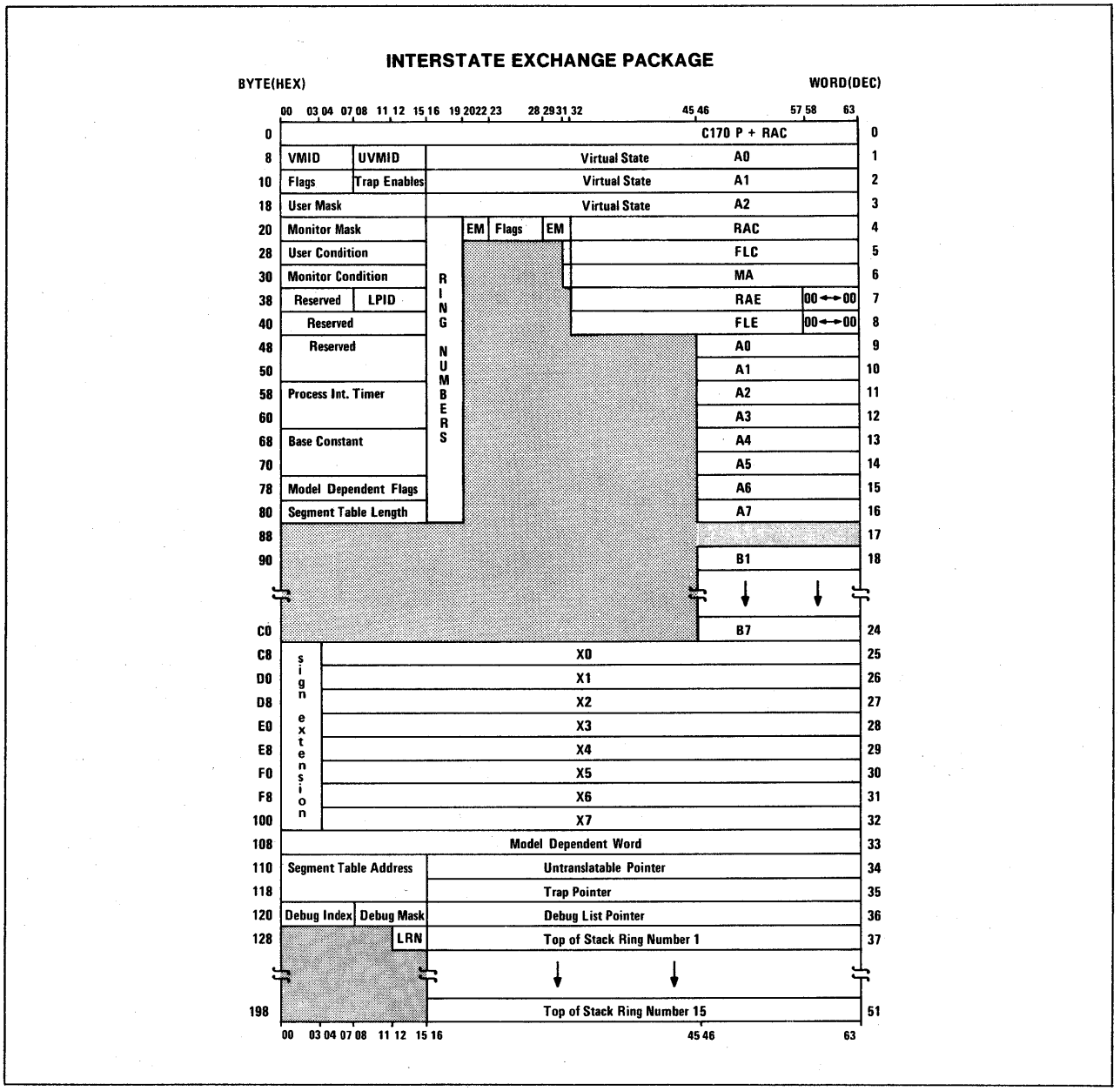


Figure II-2-26. Interstate Exchange Package

Program Address (P) Register

The P field (word 1, bits 0 through 63) contains the Virtual State P register, which supplies the PVA for the CYBER 170 State where instruction execution is to begin or resume. The KEY field contains the process key for access protection. The BN field of the PVA interprets as the CYBER 170 State P register plus RAC, appended with the parcel address. The P register part of the parcel address must not exceed 18 bits (because exchanges within CYBER 170 State truncate the P register to 18 bits). The user must also ensure that the CYBER 170 State P register does not count past 18 bits, which is possible. The format is as follows:

Bits	Description
00-01	Not used.
02-07	Not used.
08-09	Not used.
10-15	Key (KEY).
16-19	Ring number (RN).
20-31	Process segment (SEG).
32-43	Must be zero.
43-60	P + RAC (word address).
61-62	Parcel address.
63	Must be zero.

CP operation is out of range or undefined when CYBER 170 State P + RAC > 7777777_8, or when P > 777777_8.

Stack Pointers

The 48-bit A0, A1, and A2 fields (words 1 through 3, bits 16 through 63) are Virtual State A registers. These registers function as stack pointers after a trap interrupt from CYBER 170 State to Virtual State.

EM Register

The EM register contains the exit mode selection bits for use in CYBER 170 State:

Interstate Exchange Package			CYBER 170 State Exchange Package	
Word	Bit	Description	Word	Bit
4	20	Hardware error (not used).	3	59
4	21	Hardware error (not used).	3	58
4	22	Hardware error (not used).	3	57
4	29	Indefinite operand.	3	50
4	30	Infinite operand.	3	49
4	31	Address out of range.	3	48

NOTE

Hardware errors in CYBER 170 State cause an exchange to Virtual State monitor mode. Bits 20 through 22 function as software flags preserved during exchanges.

Flags

The flag formats are as follows:

Virtual State Exchange Package			CYBER 170 State Exchange Package	
Word	Bit	Description	Word	Bit
4	23	UEM enable flag.	4	56
4	24	Expanded addressing select flag.	4	55
4	25	Enhanced block copy flag.	4	54
4	26	Software flag.	4	53
4	27	Instruction stack purge flag.	4	52
4	28	Software flag.	4	51
5	31	CYBER 170 State monitor flag.	-	-
6	31	Exit mode halt flag.	-	-

Unified Extended Memory (UEM) Enable Flag

If set, this flag enables the CM block copy and single-word copy instructions (011, 012, 014, and 015) to access CM.

Expanded Addressing Select Flag

If set, slects expanded addressing mode, which provides addressing up to 24 bits in a 30-bit format for data transfer between CM and UEM. If clear, selects standard addressing mode, which provides addressing up to 21 bits in a 24-bit format for data transfer between CM and UEM.

Enhanced Block Copy Flag

If set, CYBER 170 State block copy instructions (011, 012) use XO bits 30 through 50 rather than AO to determine the CM address.

Software Flag (Word 4, Bit 28)

This is a reserved flag described in software documentation.

Instruction Stack Purge Flag

If set, this flag causes instruction stack purges as described under Code Modification in CYBER 170 State in this section.

Software Flag (Word 4, Bit 26)

This is a reserved flag described in software documentation.

CYBER 170 State Monitor Flag

If set in an incoming exchange package, this flag indicates that the CYBER 170 State process is to start (or resume) execution in CYBER 170 State monitor mode. If set in an outgoing exchange package, this flag indicates that the interrupted process was executing in CYBER 170 State monitor mode.

Exit Mode Halt Flag

If set, this flag indicates a software error that would halt a model 173 CP. This bit does not set for hardware errors that would have halted a model 173 CP. Refer to the hardware reference manual (listed in the preface) that describes the CYBER 170 Model 173.

RAC Register

The 32-bit RAC field (word 4, bits 31 through 63) contains the 21-bit CYBER 170 State reference address for CM addressing. CP operation is undefined when RAC > $7\ 777\ 777_8$ or RAC + FLC > $7\ 777\ 777_8$.

FLC Register

The 32-bit FLC field (word 5, bits 31 through 63) contains the 21-bit CYBER 170 State field length for CM addressing. CP operation is undefined when FLC > $7\ 777\ 777_8$ or RAC + FLC > $7\ 777\ 777_8$.

Monitor Address (MA) Register

The 32-bit MA field (word 6, bits 31 through 63) contains the 18-bit CYBER 170 State MA register pointing to the CYBER 170 State exchange package starting address used when executing the following instructions with the CP in CYBER 170 State job mode:

- CYBER 170 State central exchange jump (013).
- PP monitor exchange jump to MA (262x).

CP operation is undefined when MA > $777\ 777_8$.

Address (A) Registers

The eight 18-bit A fields (words 9 through 16, bits 46 through 63) are CYBER 170 State CM address (A) registers.

RAE Register

The 32-bit RAE field (word 7, bits 31 through 63) contains the 21-bit CYBER 170 State RAE register supplying the reference address for UEM addressing (instructions 011, 012, 014, 015). CP operation is undefined when RAE > 7777777_8 or when the RAE rightmost 6 bits are nonzero.

FLE Register

The 32-bit FLE field (word 8, bits 31 through 63) contains the 24-bit CYBER 170 State FLE register supplying the field size for UEM addressing (instructions 011, 012, 014, and 015). CP operation is undefined when FLE > 77777777_8 or when the FLE rightmost 6 bits are nonzero.

Virtual State Ring Numbers

The operating system supplies the A register Virtual State ring numbers (words 04 through 16, bits 16 through 19) for use after an interrupt to Virtual State. These must not be altered in CYBER 170 State.

Index (B) Registers

The 18-bit CYBER 170 State B registers 1 through 7 (words 18 through 24, bits 46 through 63) are used primarily as indexing registers. Register B0 is not included in the exchange package; this register always contains all zeros.

Operand (X) Registers

Words 25 through 32, bits 04 through 63 contain the CYBER 170 State operand (X) registers. Hardware sign-extends the X registers in an outgoing exchange package to 64 bits. The operating system sign-extends the X registers in an incoming exchange package to 64 bits.

CYBER 170 STATE EXCHANGE PACKAGE

The CYBER 170 State exchange package (figure II-2-27) resides within the CYBER 170 State process segment in memory. During exchange operations between CYBER 170 State monitor and job modes, the current CP CYBER 170 State registers store into an outgoing CYBER 170 State exchange package and reload from the incoming CYBER 170 State exchange package. These two exchange packages store into the same CM locations and, thus, the outgoing exchange package replaces the incoming exchange package in CM. Refer to the appropriate computer system (CYBER 170 State) hardware reference manual listed in the preface for further information.

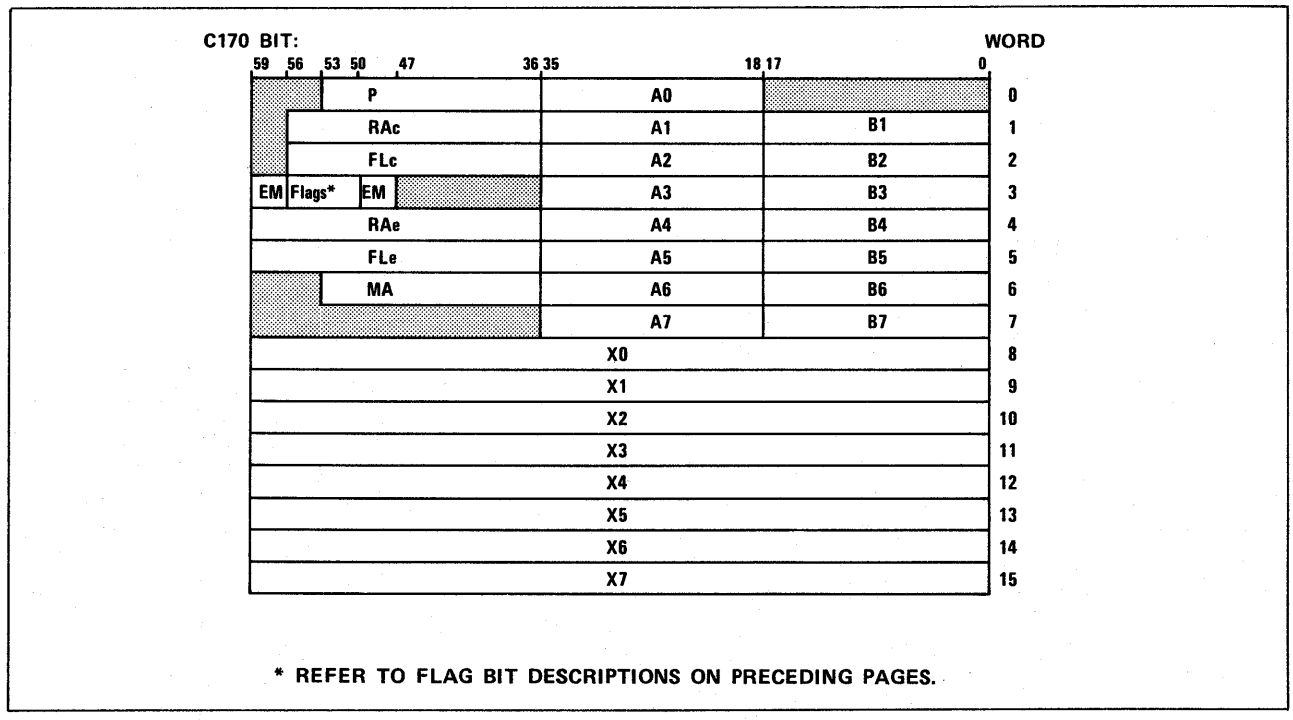
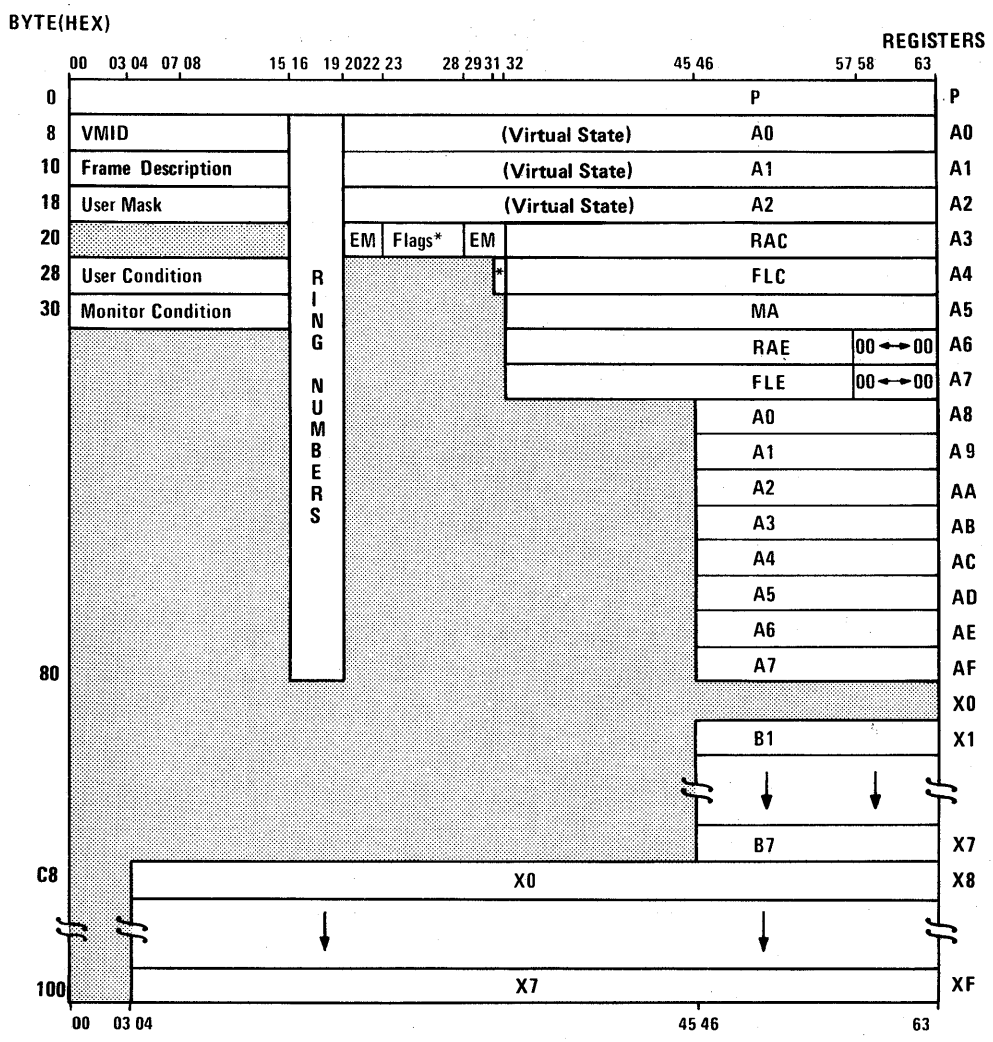


Figure II-2-27. CYBER 170 State Exchange Package

INTERSTATE STACK FRAME SAVE AREA

Interstate calls/returns and trap interrupts use a SFSA with the format shown in figure II-2-29. For the description of modified fields, refer to the corresponding fields in the Interstate Exchange Package in this section, except as follows:

- The user and monitor condition registers are stored only during trap interrupts.
- The user and monitor condition registers are not loaded by a return instruction to CYBER 170 State.
- X register field bits 0 through 3 are undefined in the SFSA after a call, return, or trap interrupt. Hardware does not sign-extend these fields.



* REFER TO FLAG BIT DESCRIPTIONS ON PRECEDING PAGES.

Figure II-2-28. Interstate Stack Frame Save Area

CODE MODIFICATION IN CYBER 170 STATE

In model 173, the CYBER 170 State return jump (010), extended memory read or write (011, 012), exchange jump (013), or long jump (02) instructions purge the instruction buffer. This is also the case in models 810 through 860.

Additionally, when the operating system sets the CYBER 170 State instruction stack purge flag in the interstate exchange package or SFSA, the conditional jump instructions (03 through 07) and store instructions (50 through 57 with i = 6 or 7) also purge the instruction stack. In such case, the modified code always executes, even with the previous code in the same instruction word as the modifying code.

With the instruction stack purge flag clear, execution of unmodified code in the instruction buffer may occur but can never be guaranteed, since an exchange interrupt may clear all instructions in the buffer at any time, including instructions in the same instruction word as the code-modifying instruction.

DEBUG/PERFORMANCE MONITORING

The CYBER 170 State environment does not support the Virtual State debug feature. Other information related to performance monitoring may be collected through the maintenance channel.

EXCEPTION HANDLING IN CYBER 170 STATE

During execution of CYBER 170 State programs, certain hardware or software errors cause an exchange to Virtual State monitor mode, or a trap interrupt to Virtual State, as described in the following paragraphs. Such exceptions do not set exit condition bits or store the P register at location RAC. Instead, RAC+P stores in the outgoing exchange package or in SFSA.

SOFTWARE EXCEPTION CONDITIONS

Table II-2-23 lists CYBER 170 State exception conditions. In general, software errors occurring in CYBER 170 State job mode with the corresponding exit mode selected cause an exchange to CYBER 170 State monitor mode. Corresponding software errors occurring in CYBER 170 State monitor mode which result in an exchange from CYBER 170 State to Virtual State monitor mode with no MCR bits set and the CYBER 170 State halt flag in the outgoing exchange package set are as follows:

- Illegal instruction in CYBER 170 State monitor mode.
- Read or write address out of range (with exit mode selected for this error) in CYBER 170 State monitor mode.
- Instruction fetch address or branch target address out of range in CYBER 170 State monitor mode.
- Infinite or indefinite value detected (with exit mode selected for this error) in CYBER 170 State monitor mode.
- 00 instruction in CYBER 170 State monitor mode.

Exceptions that may only occur immediately after entering or leaving the CYBER 170 State environment, and which cause an exchange from CYBER 170 State to Virtual State monitor mode, with an MCR bit set and the CYBER 170 State halt flag in the outgoing exchange package clear, are as follows:

Bit	Description
MCR 55	Environment specification error.
MCR 60	Invalid segment/ring number zero.
MCR 52	Address specification error.
MCR 54	Access violation.
MCR 61	Outward call/inward return.
MCR 63	Trap exception.

Table II-2-23. CYBER 170 State Exceptions (Sheet 1 of 2)

MCR/ UCR Bit	Job Process State Exchange Package Bit	Error	CYBER 170 State		Response
			Mode	Exit Mode	
MCR 48	-	Hard errors: Parity Double SECDED CM port bounds.	Any	Any	When address error, inhibit write to that address. Complete current instruction; exchange to (MPS); P = RAC + next fetch address (not necessarily related to error).
MCR 50	-	Short warning.	Any	Any	Complete current instruction; exchange to (MPS); P = RAC + next fetch address (not necessarily related to condition).
MCR 53	-	CYBER 170 State exchange request (PP).			
MCR 56	-	External interrupt (CP).			
MCR 59	-	System interval timer.			
MCR 62	-	Soft error.			
MCR 55	-	No CYBER 170 State bit in VMCL on entering CYBER 170 State.	Any	Any	Inhibit execution of target instruction; exchange to (MPS); P = RAC + target fetch address.
MCR 60	-	RN = 0 on return to CYBER 170 State.			
MCR 52	-	Address specification error when CYBER 170 State entered.	Any	Any	Inhibit execution of target instruction; exchange to (MPS); P = RAC + target fetch address.
MCR 54	-	Access violation when CYBER 170 State entered.			
MCR 60	-	Invalid segment when CYBER 170 State entered.			
MCR 60	-	RN = 0 on call to CYBER 170 State or trap from CYBER 170 State.			
MCR 61	-	Outward call/inward return to CYBER 170 State.			
MCR 63	-	Trap exception on trap from CYBER 170 State.			

Table II-2-23. CYBER 170 State Exceptions (Sheet 2 of 2)

MCR/ UCR Bit	Job Process State Exchange Package Bit	Error	CYBER 170 State		Response
			Mode	Exit Mode	
MCR 57	-	Page fault without FLC or FLE violation.	Any	Any	Interrupt execution; exchange to (MPS); P = RAC + this/next fetch address.
UCR 48	-	Trap to executing instruction (017).	Any	Any	Trap to TP in exchange package; P = RAC + next fetch address.
UCR 49	-	Unimplemented instruction fetched.	Any	Any	Inhibit execution; trap to TP in exchange package; P = RAC + this fetch address.
UCR 50	-	Free flag in incoming exchange package.	Any	Any	Inhibit next instruction; trap to TP in exchange package; P = RAC + next fetch address.
UCR 51	-	Process interval timer.			
UCR 53	-	Critical frame on return to CYBER 170 State.	Any	Any	Inhibit execution of target instruction; trap to TP in exchange package; P = RAC + target fetch address.
	29	FP indefinite. FP infinite.	Mon	Sel	Exchange to (MPS); P = RAC + this/next fetch address.
	30	FP indefinite. FP infinite.	Mon	Sel	Exchange to (M,PS); P = RAC + this/next fetch address.
-	31	FLC violation, incremental read or write.	Mon	Sel	Interrupt execution; X or CM = unchanged; A = read address less RAC; exchange to (MPS); P = RAC + this/next fetch address.
-	31	FLE violation, block transfer instruction 011, 012.	Mon	Sel	Execute as pass; exchange to (MPS); P = RAC + next fetch address.
-	31	FLE violation, single-word transfer instruction 014, 015.	Mon	Sel	Execute as pass; exchange to (MPS); P = RAC + this/next fetch address.

The following exception causes an exchange from CYBER 170 State to Virtual State monitor mode with an MCR bit set and the CYBER 170 State halt flag in the outgoing exchange package clear:

Bit	Description
MCR 57	Page table search without find.

Exceptions that may occur only immediately after entering or leaving the CYBER 170 State environment, and which cause a trap interrupt from CYBER 170 State to Virtual State, with a UCR bit set, are as follows:

Bit	Description
UCR 50	Free flag set in incoming exchange package.
UCR 53	Return with critical frame flag set.

Exceptions causing a trap interrupt from CYBER 170 State to Virtual State, with a UCR bit set, are as follows:

Bit	Description
UCR 48	Trap to Virtual State (017) instruction in the CYBER 170 State instruction set.
UCR 49	Unimplemented instructions.

A trap interrupt attempted with trap interrupts disabled causes an exchange or stack operation as described in table II-2-6.

Address Errors

An address error does not change the destination CM location or register. Read/write address errors occurring in CYBER 170 State monitor mode initiate an exchange to Virtual State monitor mode only with the corresponding exit mode selected. Instruction fetch address errors and target instruction fetch address errors in CYBER 170 State monitor mode always initiate the exchange.

A page-table-search-without-find fault (page fault) in CYBER 170 State (MCR bit 57) causes an exchange to Virtual State monitor mode only in the absence of a simultaneous FLC or FLE violation. When the exchange does occur, the interrupted process is restartable only when the page fault occurred during execution of a CYBER 170 State UEM transfer instruction (011, 012, 014, 015).

When the page fault causes the exchange, the CP places the PVA causing the page fault into the untranslatable pointer register, including the SEG and RN fields of the PVA.

The CP tests each CM reference for an address specification error, invalid segment, or access violation. When an interrupt occurs because of these conditions, the interrupted CYBER 170 State process may not be restartable.

Illegal Instructions

The following CYBER 170 State instructions, when executed in CYBER 170 State monitor mode, cause an exchange to Virtual State monitor mode with the CYBER 170 State halt flag set in the exchange package:

- Any 30-bit instruction at parcel 3.
- Instructions 011 through 013 at parcel 1, 2, or 3.
- Instruction 016.
- Instructions 011, 012, 014, and 015 with certain parameters as described below.

Extended Memory Transfer Exceptions

Instructions 011 and 012 in CYBER 170 State monitor mode, with exit mode selected, cause exceptions with the following priority:

	Exception	Response
1.	Not in parcel 0	Illegal instruction.
2.	UEM enable flag clear (exchange package)	Illegal instruction.
3.	FLC violation	Address range error.
4.	Negative blocklength	Address range error.
5.	FLE violation	Address range error.
6.	Zero blocklength	Fetch next instruction.

Instructions 014 and 015 in CYBER 170 State monitor mode, with exit mode selected, cause exceptions with the following priority:

	Exception	Response
1.	UEM enable flag clear (exchange package)	Illegal instruction.
2.	FLE violation	Address range error.
3.	Xk (CYBER 170 State) bit 21/22/23 set	Address range error.

HARDWARE EXCEPTIONS IN CYBER 170 STATE

Table II-2-20 lists CYBER 170 State exception conditions. In general, hardware exceptions cause an exchange to Virtual State monitor mode or a trap interrupt to Virtual State job mode.

Hardware exceptions causing an exchange from CYBER 170 State to Virtual State monitor mode are as follows:

Bit	Description
MCR48	Detected uncorrectable error.
MCR50	Short warning.
MCR56	External interrupt.
MCR59	System interval timer.
MCR62	Soft error.

Hardware exceptions causing a trap interrupt to Virtual State job mode are as follows:

Bit	Description
UCR53	Process interval timer.

IOU PERIPHERAL PROCESSOR PROGRAMMING

The PPs may access all CM storage locations. One CM word or a block of CM words can transfer from a peripheral processor memory (PPM) to CM or from CM to a PPM. Data from external devices is read into a PPM, and with additional instructions, transfers to CM. Conversely, data is transferred from CM to a PPM and then transfers by way of additional instructions to external devices.

CENTRAL MEMORY ADDRESSING BY PPs

Addresses sent to CM from PPs are real memory addresses. PPs address CM using either absolute or relocation addressing. Every PP can read all CM locations without restriction. Every PP has write access to CM as determined by the OS bounds register in the IOU. The port bounds register in CM may also be set to limit write access from IOU.

Absolute and Relocation Addressing

If A register bit 46 is a zero, bits 47 through 63 of A specify an absolute CM address 0 through 377777_8. If bit 46 of A is a 1, bits 47 through 63 of A are added to the 28-bit relocation register R to specify an absolute CM address 0 through 1777777777_8. If bit 46 of A changes during a transfer, the addressing mode changes accordingly. Figure II-1-7 shows how a relocation address forms.

Instructions 0024/0025 load/store the relocation register. The leftmost 7 bits of R represent (unused) extra addressing capacity. The rightmost 6 bits of R are appended zeros. Figure II-1-6 shows how R stores in PP memory.

OS Bounds Test

The OS bounds test restricts write access from selected PPs to an upper or a lower region in CM. The PP instructions for which the OS bounds test performs are as follows:

Exchange jumps (00260, 00261, 00262).

Central write (0062, 1062).

Central write (d) words (0063, 1063).

Central read and set/clear lock (1000/1001).

PP Central Memory Read

Instructions which read CM data into PPM are as follows:

- 60-bit CM words to five 12-bit PP words

  Central read from A to d (0060).

  Central read (d) words from (A) to m (0061).

- 64-bit CM words to four 16-bit PP words

  Central read from A to d long (1060).

  Central read (d) words from (A) to m long (1061).

  Central read and set lock from d to (A) (1000).

  Central read and clear lock from d to (A) (1000).

It is possible, by way of block read (0061, 1061) to read up to 4095 CM words, over-writing PP memory cyclically. Hardware, however, uses PPM location 0 to hold the program counter during block transfers. Refer to instructions 0061 and 1061 in section II-1.

PP Central Memory Write

Instructions which write PPM data from into CM are as follows:

- Five 12-bit PP words to 60-bit CM words

  Central write to (A) from d (0062).

  Central write (d) words to (A) from m (0063).

- Four 16-bit PP words to 64-bit CM words

  Central write to (A) from d long (1062).

  Central write (d) words to (A) from m long (1063).

It is possible by way of block write (0063, 1063) to write up to 4095 CM words, repeating PP memory cyclically. Hardware, however, uses PPM location 0 to hold the program counter during block transfers. Refer to instructions 1062 and 1063 in section II-1.

PP MEMORY ADDRESSING BY PPs

PP instructions use 6-bit/18-bit direct operands or obtain the operand from PP memory using direct, indirect, or indexed addressing.

Direct 6-Bit Operand

PP instructions of this type are no-address instructions. They have the format OPCODEd. The d-field provides a 6-bit direct operand zero-extended to 18 bits in calculations.

Direct 18-Bit Operand

PP instructions of this type are constant address instructions. They have the format OPCODEdm. The combined d and m field provides an 18-bit operand.

Direct 6-Bit Address

PP instructions of this type are direct address instructions. They have the format OPCODEd. The d field provides a 6-bit direct address, accessing PPM locations 0 to 77_8.

Direct 12-Bit Address

PP instructions of this type are indexed direct address instructions, with zero index. They have the format OPCODEdm, d = 0. The m field provides a 12-bit direct address, accessing PP memory locations 0 through 7777_8.

Indexed 12-Bit Address

PP instructions of this type are indexed direct address instructions. They have the format OPCODEdm, d = 0. The m field provides a 12-bit direct address (base address). The d field specifies a PP memory location from 0 to 77_8, the contents of which is a 12-bit index. The indexed direct address forms by adding the index to the base address as signed one's complement numbers, ignoring overflow. When m + (d) = 7777 the result sets to 0000, except in the addition 7777 + 7777 = 7777.

Indirect 6-Bit Address

PP instructions of this type are indirect address instructions. They have the format OPCODEd. The 6-bit d field addresses PP locations 0 through 77_8. The 12 rightmost bits of the addressed location provide an address to access PP memory locations 0 through 7777_8.

CHANNEL INPUT/OUTPUT OPERATIONS

All PPs may access all external devices through internal and external interfaces. Each internal interface contains a data register and channel control flags. The internal interfaces connect to external interfaces communicating with the external devices.

CHANNEL FLAGS

Channel operation is controlled by the channel flags, which are set/reset by PP instructions and by signals from the external devices. The channel flags are as follows:

- Channel active/inactive flag.
- Register full/empty flag.
- Channel (marker) flag.
- Error flag.

The active flag and the full flag control the channel input/output transfers. The status of these two flags determines the channel active, inactive, full, or empty. The marker flag is for software use, and the error flag indicates transmission parity errors.

Channel Active Flag

A PP sets the active flag to indicate a reserved channel (channel active). The PP or the external device clears the active flag to indicate a free channel (channel inactive). Devices connected through the CYBER 170 State 12-bit channels may also set this flag to request attention.

A PP sets the active flag by the activate instruction (0074) or function instructions (0076, 0077). A PP clears the active flag with the deactivate instruction (0075). Normally, external devices clear the active flag in response to a function instruction, or when they have no more data to send. A PP senses the active flag state using the jump on active/inactive instructions (00640 and 00650).

Register-Full Flag

A register is full when it contains a function or data word for an external device, or when it contains a similar word received from the external device. The register is empty after the word is read. The flag turns on or off as the register changes states. A channel can only be full when it is active.

When set, the register-full flag signals the destination that data is available, and signals the sender that no more immediate data can be sent. When clear, full flag signals the destination to wait for the next data word, and signals the sender to send another word. During block transfers, the register-full flag sets once for each word written into the register.

The register-full flag also clears when the channel goes inactive for any reason. The PPs can sense the flag state using the jump-on-full/empty instructions (00660/00670).

Channel (Marker) Flag

This flag is used by software as a marker and does not affect hardware operation. The flag provides dual PP I/O driver programs with a synchronization mechanism. The flag is inaccessible to external devices.

The marker flag is set/cleared by the channel flag instructions (00641/00751). A PP can sense the marker flag state using the jump-on-set/clear instructions (1064/1065).

Priority conflicts exist when PPs in the same time slot use this flag. Hardware resolves the marker flag priority conflicts for the maintenance channel 17_8. For other channels, the problem is resolved by software interlocks kept in CM, or by not assigning PPs in the same time slot to the same channel. Any five consecutively-numbered PPs are not in the same time slot.

Error Flag

This flag indicates a data parity error on a channel transfer. The IOU interface sets the error flag when it detects a data parity error on input data. External devices connected through 16-bit channels can also set this flag when detecting an output data parity error. When this flag sets in any internal interface, the channel parity error bit also sets in the IOU fault status register. PP instructions 00661 and 00671 clear and sense the error flag.

PROGRAMMING FOR CHANNEL INPUT/OUTPUT

Data transfers to/from external devices are controlled by PP instructions 0064 through 0077. The same instruction set services 8-, 12-, and 16-bit channels. The assignment of PPs, transfer priorities, and resolution of conflicts is a software responsibility.

The channel marker flag and/or software interlocks in central memory provide for channel parity and reservation. Proceed as follows after resolving conflicts:

	Action	Typical Instruction
1.	Clear error flag.	Jump if error flag set, and clear flag (00661).
2.	Verify channel availability.	Jump if active (00640).
3.	Verify device availability:	
	Request device to send status.	Function m (00770).
	Wait until device responds.	Jump if active (00640).
	Activate channel.	Activate (00740).
	Read device status.	Input to A (00700).
	Verify error status.	Jump if error flag set (00661).
	Analyze device status.	Logical product (0012), zero jump (0004).

4. Prepare for input/output:

Enter number of words to A.	Load d (0014).
Verify channel inactive.	Jump if active (00640).
Prepare device for read/write.	Function m (00770).
Wait until device responds.	Jump if active (00640).

5. Read/write data:

Activate channel.	Activate (00740).
Read/write data.	Input/output A words (0071/0073).
If write, loop until empty.	Jump if full (00660).
Disconnect Channel.	Deactivate (00750).
Verify inactive status.	Jump if active (00640).

6. Verify transfer integrity:

Verify A words were transferred (note 1).	Nonzero jump (0005).
Verify error status.	Jump if error flag set (00661).
Verify inactive status.	Jump if active (00640).
Request device. to send status.	Function m (00770).
Wait until the device responds.	Jump if active (00640).
Activate channel.	Activate (00740).
Read device status.	Input to A (00700).
Verify error status.	Jump if error flag set (00661).
Analyze device status.	Logical product (0012), nonzero jump (0005)
Verify inactive status.	Jump if active(00640).

NOTE

If A = original value, no words were transferred. If A is not equal to 0, device or another PP ended transfer.

INTER-PP COMMUNICATIONS

Any PP can communicate with any other PP using any channel (except the real-time clock) by omitting the conditioning of that channel's external devices for a data transfer. Both single-word and block transfers can be used.

Either the sending PP or the receiving PP can activate the channel used, after which the sending PP outputs data into the data register and the receiving PP inputs data from the same register.

The transfer rate is one word every 250 nanoseconds, except when the transfer is between PPs in different barrels but the same time slot. In such a case, the transfer rate is one word every 500 nanoseconds. PPs using the same time slots are as follows:

Models 810, 815 and 825, and 830

Slot	PP Number	
1	0	10
2	1	11
3	2	12
4	3	13
5	4	14
6	5	15
7	6	16
8	7	17
9	8	18
10	9	19

Models 835, 840, 845, 850, 855, 860, and 990

Slot	PP Number
1	0, 5, 20, 25
2	1, 6, 21, 26
3	2, 7, 22, 27
4	3, 10, 23, 30
5	4, 11, 24, 31

Software must resolve priority and reservation problems arising in inter-PP communications.

PP PROGRAM TIMING CONSIDERATION

Some external equipment requires timing considerations in issuing a function, activate, or input instruction. Refer to the applicable external equipment reference manual. Such timing considerations may be required, for example, to ensure that the equipment attains a proper speed before data is sent (required by some magnetic tape equipment). Also, equipment terminating a data transfer by resetting the active flag often requires timing considerations in issuing the next function instruction.

CACHE INVALIDATION

When a PP executes 60-bit central write instructions, the IOU sends cache memory invalidation requests to the CP. The CP responds by purging the cache memory of any former copies of the words stored in CM. Such invalidation requests are sent during the following central write instructions:

- Central write A to d (0062), with every 60-bit word.
- Central write (d) words from m to (A) (0063), when an address with bits 62 and 63 set is sent to CM, and with the last word written.

NOTE

Cache is not invalidated during the execution of instructions 1001, 1002, 1062, and 1063.

ERROR DETECTION AND RECOVERY

The IOU and each PP have fault detection and reporting hardware. The IOU generates and checks parity on all data transferred between PP and PP memory, CM and IOU, and IOU and external devices.

PP Hardware Errors

When a PP hardware error occurs with the enable error stop bit set in the IOU environment control register, the PP with the fault halts (idles). In this case, another PP may perform error detection and logging. When one PP halts from error detection, the remaining PPs are affected only when a PP is waiting for the halted PP to perform some action.

Error reporting from any PP with a fault can be disabled by setting the relevant bit in the IOU fault status mask register. This is normally done when removing a PP from service, and restores normal error reporting from other PPs through the summary status byte.

Channel Parity Errors

The output register 16 bits are checked for parity whenever the register is full. When a parity error is detected, the following takes place:

- Channel error flag in the channel concerned sets.
- Fault status register bit for this channel sets.
- Uncorrected error bit in IOU status summary register sets.

Error reporting from any channel with a fault can be disabled by setting the relevant bit in the IOU fault status mask register. This is normally done when removing a channel from service, and restores normal error reporting from other PPs through the summary status byte.

Parity Errors on Output Data

The IOU sends a data or function word to the channel with parity calculated on all 16 bits of the channel output register. In case of 8- and 12-bit channels, software must ensure that the missing bits in the output register are zeros. This ensures correct channel parity after the unused bits discard.

Software must verify the integrity of a 12-bit channel output data transfer by requesting a status word from the device concerned. When a device detects a parity error on a 12-bit function word output, it does not send any response and the channel remains active and full.

Devices connected through 16-bit channels or the maintenance channel respond to a data word parity error detected at the device by resetting the channel full flag and setting the error flag. The channel remains active and execution of the current output instruction continues. These devices respond to a function word parity error by resetting the active flag and setting the error flag.

The 12-bit channel contains a switch to disable parity checking.

Parity Errors on Input Data

For all channels, the IOU sets the channel error flag whenever it detects a parity error on input data. The IOU regenerates correct parity before storing the data into PP memory.

Timeout

The maintenance channel interface provides a 100-microsecond timeout counter to ensure that the PP dealing with that channel continues operations when the maintenance channel does not respond to a data transfer command. The timeout interval starts when the maintenance channel goes active or full, and resets when the channel goes inactive or empty. If the IOU receives no response by the end of the timeout interval, it clears the channel active flag.

Function word output does not activate the timeout counter. This allows software to recover from a maintenance access control malfunction.

To allow inter-PP communications without timeout, the timeout is disabled when the maintenance channel interface is deselected from channel 17_8 using connect codes 8 through F.

INITIALIZATION

System initialization begins with the IOU, which requires no external hardware or software aid to initialize itself. After the operator presses the deadstart button (CC545) or presses the CTRL G key (CC634B, model 990 only), a storage device in the IOU provides initialization programs and data for further action.

After the IOU has self-initialized, any or all of the following operations may be performed by way of the system console and deadstart options:

- Load CP control memory.
- Initialize CM.
- Dump CM.
- Run CP quick look test.
- Begin maintenance system load.
- Begin operating system load.

SYSTEM CONSOLE PROGRAMMING (CHANNEL 10_8)

KEYBOARD

A PP must transmit a one-word function code (7020_8) to request data from the system console keyboard. The code prepares the display controller for an input operation. The PP then activates the input channel and receives one character from the keyboard. This character enters as the lower 6 bits of the word, and the upper bits clear. There is no status report by the keyboard. Table II-2-24 lists the keyboard character codes.

DATA DISPLAY (CC545)

Data is displayed within an 203.2 mm (8-in) area of a cathode-ray tube (CRT). The display can be alphanumeric (character mode) or graphic (dot mode). There are 262144 dot locations arranged in a 512-by-512 format. Each dot position is determined by the intersection of X and Y coordinates. The lower left corner dot is octal address X = 6000 and Y = 7000, and the upper right corner dot is octal address X = 6777 and Y = 7777. (Model 990 also uses a CC634B system console. Refer to the hardware reference manual listed in the preface for information regarding this terminal.)

Character Mode

Large, medium, and small characters are provided in character mode. Large characters are arranged in a 32-by-32 dot format with 16 characters per line. Medium characters are arranged in a 16-by-16 dot format with 32 characters per line. Small characters are arranged in an 8-by-8 dot format with 64 characters per line. Table II-2-25 lists the display character codes.

Dot Mode

In dot mode, display dots are positioned by the X and Y coordinates. The X coordinates position the dots horizontally. The Y coordinates position the dots vertically and unblank the CRT for each dot. Horizontal lines form from a series of X and Y coordinates. Vertical lines form from a single X coordinate and a series of Y coordinates.

Table II-2-24. Keyboard Character Codes

Character	Code	Character	Code
No data	00	0	33
A	01	1	34
B	02	2	35
C	03	3	36
D	04	4	37
E	05	5	40
F	06	6	41
G	07	7	42
H	10	8	43
I	11	9	44
J	12	+	45
K	13	-	46
L	14	*	47
M	15	/	50
N	16	(	51
O	17	)	52
P	20	Left blank key	53
Q	21	=	54
R	22	Right blank key	55
S	23	,	56
T	24	.	57
U	25	Carriage return	60
V	26	Backspace	61
W	27	Space	62
X	30		
Y	31		
Z	32		

Table II-2-25. Display Character Codes

Character	Code	Character	Code
No data	00	0	33
A	01	1	34
B	02	2	35
C	03	3	36
D	04	4	37
E	05	5	40
F	06	6	41
G	07	7	42
H	10	8	43
I	11	9	44
J	12	+	45
K	13	-	46
L	14	*	47
M	15	/	50
N	16	(	51
O	17	)	52
P	20	Space	53
Q	21	=	54
R	22	Space	55
S	23	,	56
T	24	.	57
U	25		
V	26		
W	27		
X	30		
Y	31		
Z	32		

Codes

A single function word transmits to select the presentation, mode, and character size (character mode only). Figure II-2-29 illustrates the function word format. The word following the function word specifies the starting coordinates for the display (for either mode). Figure II-2-30 illustrates the coordinate data word. Figure II-2-31 illustrates the character word.

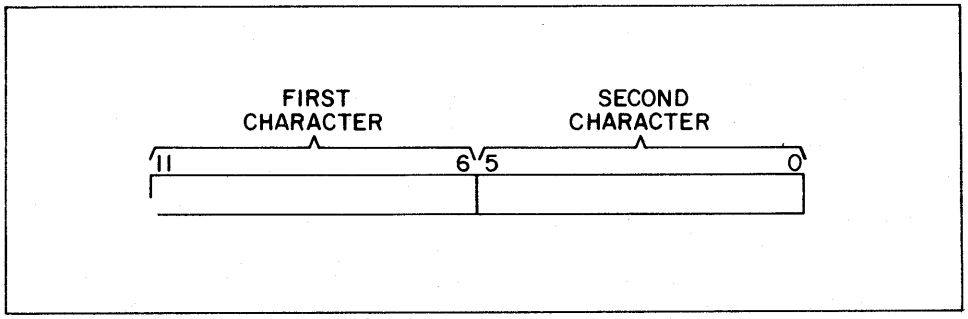


Figure II-2-29. Display Station Output Function Code

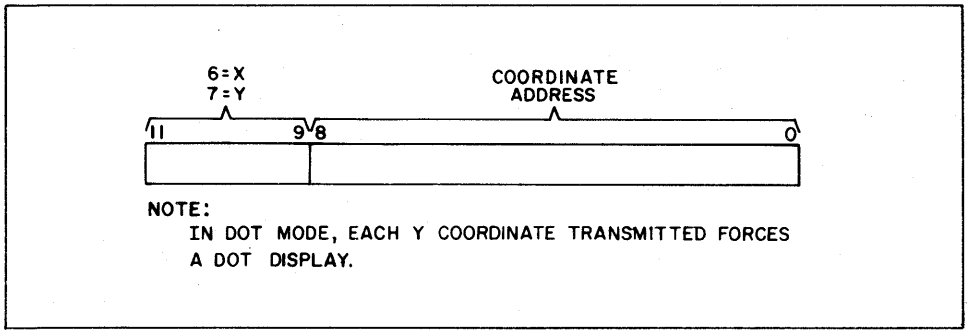


Figure II-2-30. Coordinate Data Word

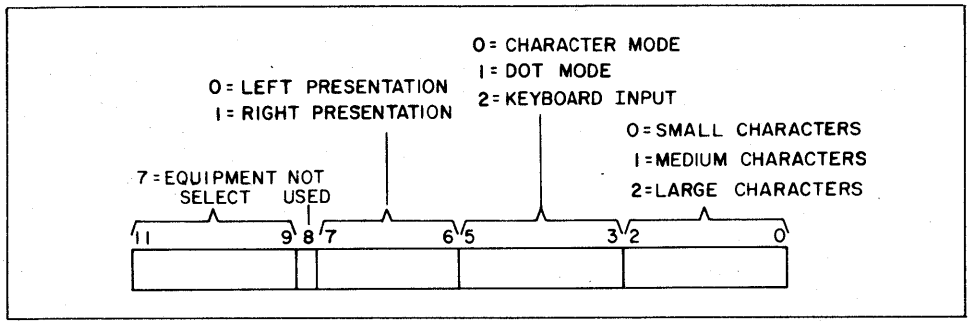


Figure II-2-31. Character Data Word

The controller regulates character spacing on the line once the display operation starts. A new Y coordinate data word must be sent to start each line. If a new Y coordinate is not specified, data is written on the line specified by the active Y coordinate word, and information already on that line is overwritten. Character sizes can be mixed by sending a new function word and coordinate word for each size change. Spacing on a line can be varied by sending a coordinate word for the character which is to be spaced differently.

PROGRAMMING EXAMPLE

The following programming example (figure II-2-32) requests an input of one line of data from the system console, and displays this data on the CRT as it is being typed.

PROGRAM TIMING CONSIDERATION

When performing an output operation, the computer must wait for a channel empty condition at the end of the output to prevent loss of coordinates or data. A full jump at the end of the output ensures the channel empty and acceptance of the last output word by the display controller before disconnecting from the channel.

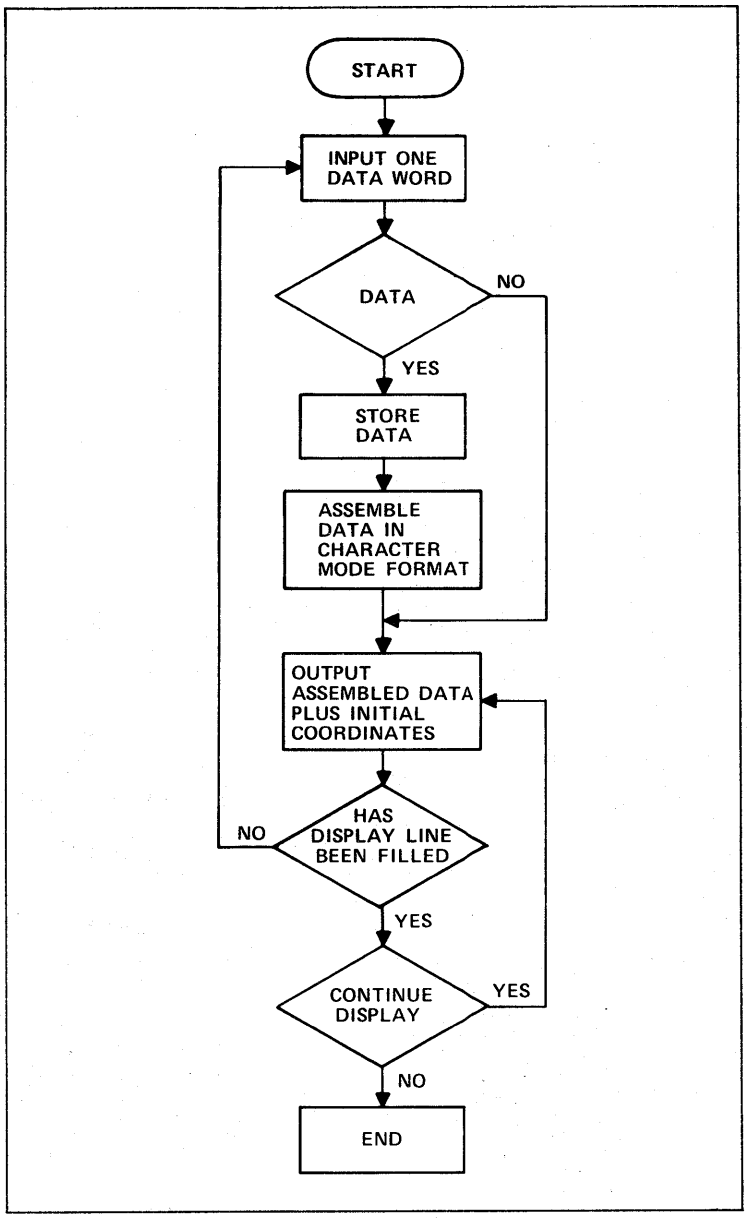


Figure II-2-32. Receive and Display Program Flowchart

REAL-TIME CLOCK PROGRAMMING

Channel 14_8 is reserved for the real-time clock. This channel is always active and full, and may be read at any time. The real-time clock is a 12-bit free-running counter incrementing at a 1-megahertz rate from 0 to 4095_{10}.

IOU DEDICATED CHANNELS

Figure II-2-33 illustrates the IOU dedicated channels for models 810, 815, 825, 830, and 990. Figure II-2-34 illustrates the IOU dedicated channels for models 835, 840, 845, 850, 855, and 860.

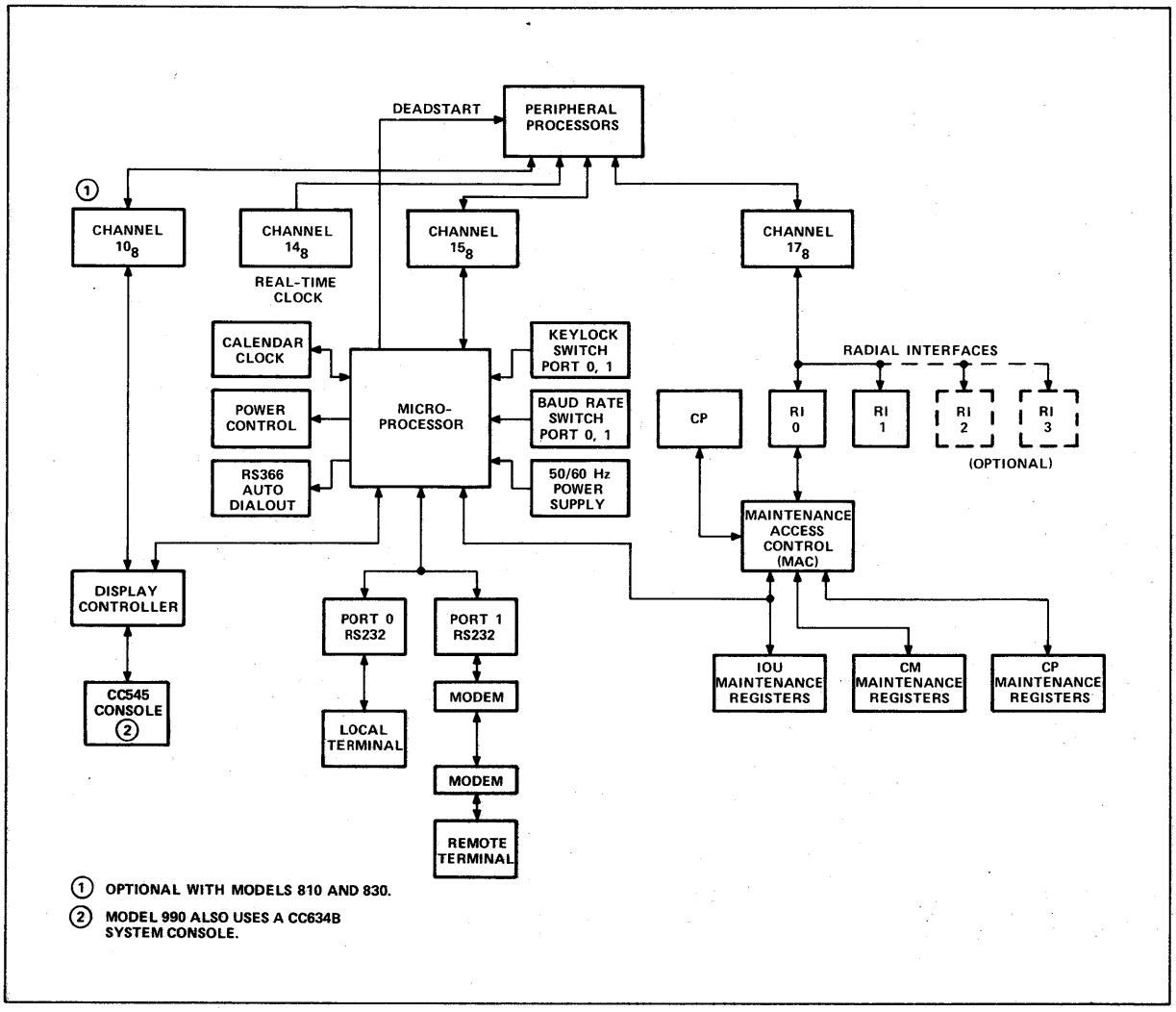


Figure II-2-33. IOU Dedicated Channels, Models 810, 815, 825, 830, and 990

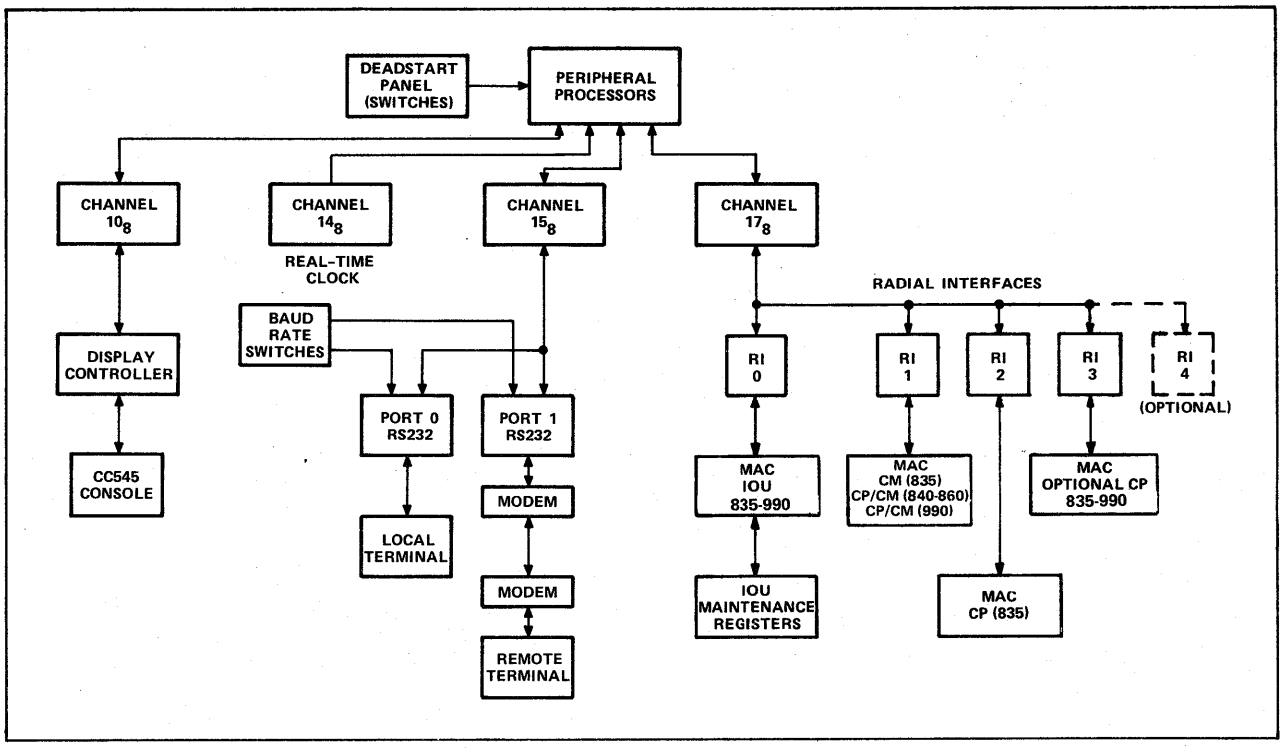
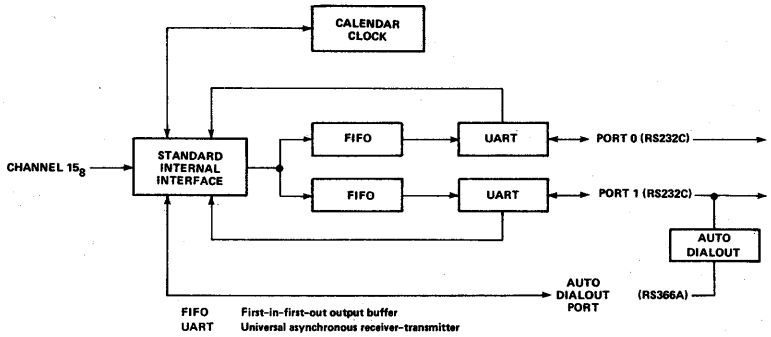


Figure II-2-34. IOU Dedicated Channels, Models 835, 840, 845, 850, 855, and 860

TWO-PORT MULTIPLEXER PROGRAMMING

Channel 15_8 provides serial communications capability with two external devices through the two-port multiplexer. One port is reserved for maintenance use. With both ports deselected, this channel can also be used for PP-to-PP communications. The arrangement is as shown below.



The two-port multiplexer can communicate with all devices which use EIA standard RS-232 serial asynchronous interface at baud rates of 110, 300, 600, 1200, 2400, 4800, or 9600. (Additionally, a baud rate of 19 200 can be used with the models 815 and 825.) The baud rate for each port is independently selectable by switches on the two-port multiplexer PCB.

The multiplexer ports can accommodate data with odd/even parity, 5 to 8 bits per character, and 1 or 2 stop bits. The format is set by issuing appropriate function codes.

The following table lists the devices which the two-port multiplexer can use to display the system deadstart settings:

		IOU (model 810,815)	IOU (model 825)	IOU (model 830)	IOU (models 835-860)	IOU (model 990)
TWO-PORT MUX DISPLAY	CC 545 SYSTEM CONSOLE		X	X		X
	CC634B SYSTEM CONSOLE					X
	CDC 752/722 TERMINAL		X	X		
	CDC 721 TERMINAL	X		X		
SWITCH PANEL DISPLAY					X	

The models 810, 815, 825, 830, and 990 IOU multiplexer supports the following special features:

- Remote deadstart.
- Calendar clock.
- Internal port-baud-rate selection.

The models 810 and 830 IOU multiplexer additionally supports the following special features:

- Auto dial-out.
- Remote power control.

FUNCTION WORDS

The two-port multiplexer uses the channel 15_8 rightmost 12 bits as a function word from the PP. The function word specifies the following:

Octal Code	Description
7XXX	Terminal select.
6XXX	Terminal deselect.
1XXX	Calendar clock/auto dial-out operations.
00XX	Read summary status.
01XX	Read terminal data.
02XX	Output to first in, first out (FIFO) buffer.
03XX	Set operation mode to terminal.
04XX	Set/clear terminal control signal DATA TERMINAL READY (DTR).
05XX	Set/clear terminal control signal REQUEST TO SEND (RTS).
06XX	Not used.
07XX	Master clear selected port.

Terminal Select (7XXX)

This code selects the terminal to which the function codes and data transmissions apply:

Code	Description
7000	Select port 0 (future use).
7001	Select port 1 (maintenance use).

Terminal Deselect (6XXX)

This code deselects the two-port multiplexer from channel 15_8. When deselected, channel 15_8 can be used for 16-bit PP-to-PP communications. Inter-PP communications over channel 15_8 should be used with caution since the transfer rate is variable (5 microseconds/word through 1 millisecond/word, with 100 microseconds/word typical).

Calendar Clock/Auto Dial-Out (1XXX)

This code can select several functions which pertain mostly to the calendar clock and auto dial-out functions. These particular functions involve a data transfer from the microprocessor memory to the PPs.

Code	Description
1X02	Read Deadstart Port/Terminal Type. Identifies the port which initiated the last deadstart operation. The multiplexer stores the terminal type and port number of the logged-in deadstart device.
1X03	Set Port Baud Rate. Sets the baud rate of the port which is currently selected.
1X04	Read Calendar Clock. Reads the calendar clock after the multiplexer selects either port 0 or port 1.
1X05	Write Calendar Clock. Writes the calendar clock after the multiplexer selects either port 0 or port 1.
1X06	Write Auto Dial-Out Data (models 810 and 830 only). Prepares the multiplexer for an auto dial-out write operation. Port 1 must be selected before this function is issued.

Bit	Bit Description
56 through 59	First number
60 through 63	Second number

Code	Description
1X07	Read Auto Dial-Out Status (models 810 and 830 only). Requests the multiplexer for an auto dial-out status operation. Port 1 must be selected before this function is issued.

Bit	Bit Description
52 through 59	First number
60	Abandon call
61	Call origination status
62	Data line occupied
63	Power indication

Code	Description
1X10	Abandon Call (models 810 and 830 only). Requests that the multiplexer abandon the call currently being attempted.

Read Summary Status (00XX)

This code prepares the channel for status input from the selected terminal. A one word input must follow to read the 12-bit status response, which is as follows:

Bit	Description
52 through 58	Not used.
59	Output buffer not full.
60	Input ready.
61	Data carrier detect or carrier on.
62	Data set ready.
63	Ring indication.

PP Read Terminal Data (01XX)

This code prepares the channel for data input from the selected terminal. Channel 15_8 must be activated and one word data input instructions must follow to read in the terminal data. The 12-bit data word has the following format:

Bit	Description
52	Data set ready. Indicates that the DATA SET READY (DSR) signal is active.
53	Data set ready and data carrier detector. Indicates that both DATA SET READY (DSR) and DATA CARRIER DETECTOR (DCD) signals are active.
54	Overrun. Indicates that the previously received character was not read by the PP before the present character over-wrote the previous character.
55	Framing or parity error. Indicates that the received character does not have a valid stop bit (framing error) or that the received character parity does not agree with the selected parity (parity error).
56 through 63	Data character.

PP Write Output Buffer (02XX)

This code prepares the multiplexer for an output operation to the 64-character output buffer memory. The channel 15_8 active flag must be set before an output operation can proceed.

When an output operation fills the buffer completely and no more locations are available, the multiplexer arbitrarily resets the channel active flag.

Set Operation Mode to Terminal (03XX)

This code sets data terminal operation mode as follows.

Code Bit	Description
58	Enable loop back (models 815 and 825 only). When set, this bit enables a round-trip data path from channel 15_8 to the selected RS-232 port and back to channel 15_8. The RS-232 interface does not transmit data externally in this mode.
59	No parity. When set, this bit eliminates parity bit from transmitted and received character. In such a case, stop bit(s) immediately follow the last data bit.
60	Number of stop bits. Selects number of stop bits (1 or 2) which follow immediately after parity bit:

Bit 60	Description
Clear	1 stop bit.
Set	2 stop bits.

61 through 62	Number of bits per character. Select 5, 6, or 7 bits per character:

Code	Bits/Character
00	5
01	6
10	7
11	8

63	Odd/even parity select. Selects type of parity appended immediately after data bits. Also determines the parity checked on input. When set, selects even parity.

Set/Clear Data Terminal Ready (DTR) (04XX)

This code conditions the data terminal to send or discontinue the DATA TERMINAL READY (DTR) control signal as follows:

Code Bit 63	Action Taken
Set	DTR set active.
Clear	DTR set inactive.

Set/Clear Request to Send (RTS) (05XX)

This code sets or clears the terminal control signal REQUEST TO SEND (RTS) as follows:

Code Bit 63	Action Taken
Set	RTS set active.
Clear	RTS set inactive.

Master Clear (07XX)

This code master clears the selected port, including any buffer-stored data. The DTR and RTS terminal control signals are not affected.

PROGRAMMING CONSIDERATIONS

Channel 15_8 communicates one at a time with the terminals connected to the external interface. To establish communications between a PP and the terminal, the following takes place:

1. PP issues a coded function word to select the terminal.
2. Multiplexer responds by resetting the channel active flag to acknowledge receipt of the function code.

The multiplexer now routes all data to the selected terminal; other function words and data input/output follow.

Data Output

The multiplexer can buffer-store a maximum of 64 characters per port. After 64 characters are stored in the buffer, the multiplexer resets the channel active flag on the last output word. The multiplexer terminates an output transfer when it receives an inactive signal from the channel.

The multiplexer does not permit output to a full buffer. Whenever the output buffer is full and the multiplexer decodes a function code 02XX (PP write output buffer), the multiplexer resets the channel active flag.

Data Input

The multiplexer does not buffer-store input data from the terminal. When the PP does not input the previous data before the new data arrives, a lost data condition (overrun) exists.

Request to Send and Data Terminal Ready

Request to send and data terminal ready signals are automatically brought up by the hardware under the following conditions (regardless of the software RTS and DTR bits):

- Data in the universal asynchronous receiver-transmitter (UART) output register.
- Data in the FIFO output buffer register.

When no data is in the FIFO or UART, the software bit determines RTS and DTR.

MAINTENANCE CHANNEL PROGRAMMING

Any PP in the IOU can be programmed to perform any or all of the following operations on the CP, CM, and IOU through the 8-bit maintenance channel (MCH):

- Initializing registers, controls, and memories.
- Monitoring and recording error information.
- Verifying error detection and correction hardware.

The PP performing such operations is often called the maintenance control unit. The MCH consists of the MCH interface on channel 17_8, a maintenance access control in CP, CM, and IOU, and two sets of interconnecting cables.

The MCH interface contains a selector that connects the MCH to one of up to seven isolated sets of cables. The IOU is element 0 and its maintenance access control connects internally to the selector. The CP and the CM (models 810, 815, 825, 830, and 835) are assigned arbitrary element numbers depending on the connector used at the MCH interface. (The models 845 and 855 CM shares a common cable and element number with the CP.)

The CP and the CM (models 810, 815, 825, 830, and 835) connect to the IOU by separate cables and gates. This arrangement results in a radial connection that allows the CP or the CM (models 810, 815, 825, 830, and 835) to be shut down or removed without affecting communication with the other unit.

MCH FUNCTION WORDS

The MCH function word consists of the connect, opcode, and type fields used as described below. Table II-2-26 describes the MCH function word bit assignments.

The connect field specifies the unit to which the MCH is connected [CP, CM (models 810, 815, 825, 830, and 835), or IOU], controlling selection within the IOU only. The unit remains connected until another connect code selects a different unit. Connect codes 10_8 to 17_8 leave the MCH unconnected; in this state the interface can be used for PP-to-PP communications without timeout restrictions.

The OPCODE field controls the unit selected by the connect code; preparing the unit for a coming read/write/echo operation; or causing the unit to halt, start, clear, or deadstart.

The use of the TYPE field depends on the connected unit. With the CP the connected unit, type codes 1 to A_{16} (models 810 - 835) or 1 through 7 (models 840 through 860) specify the CP register connected. Also, for the CP, type code 0 specifies that the internal address of the CP register to be connected is specified in a control word sent as two data words immediately following the function word. With the IOU the connected unit, type codes 0 to 7 specify the starting byte number for read/write operations (all models except 990). For model 990, the TYPE field must be set to all zeroes. For the models 810, 815, 825, 830, and 835, the CM ignores the type code. For the models 845 and 855, type code A selects access to CM.

MCH CONTROL WORDS

Some function words must be followed by a 16-bit control word specifying the internal address of the register to be connected. The control word must issue as two 8-bit data words (sometimes called address bytes). This is accomplished by outputting two 16-bit words from PP memory where each word's rightmost 8 bits comprise the 16-bit control word. Such control words are required after the following:

1. Function words to CP (models 810, 815, 825, 830, and 835) with opcodes 4/5 (read/write) and typecode 0.
2. Function words to CP (models 840, 845, 850, 855, 860, and 990) with opcodes 4/5.
3. Function words to CM and IOU with opcodes 4/5.
4. Function words to CP, CM, and IOU with opcode 8 (echo).

MCH Programming for Halt/Start (Opcode 0/1)

These operations consist of the function word output. A halt opcode halts the processor without damaging the executing process, including the integrity of the halted processor's interunit communications such as CYBER 170 State exchange request communication, central memory communications, and the process state. If the process is restarted without performing any other MCH operations, or after performing read/write with precautions as described in the Operating Systems Manual, the process continues undamaged.

MCH Clear LED (Opcode 3)

This operation clears all LEDs associated with pak errors and is intended, however not required, for use at system initialization. For maintenance reasons, this operation can also clear LEDs without initializing and master clearing.

MCH Programming for Read/Write (Opcode 4/5)

Refer to Programming for PP Data Input/Output in this section for a more complete procedure. In general terms, proceed as follows:

1. Issue function with opcode 4/5.
2. Output data word (leftmost half of control word).
3. Verify error flag clear.
4. Output data word (rightmost half of control word).
5. Verify error flag clear.
6. Input/output required number of data words.
7. Verify error flag clear.

Reading a nonexistent register returns all zeros. Writing to a read-only register, or to a nonexistent register, does not alter any register. Most registers are read/written as 64-bit (8-byte) registers, requiring the input/output of eight MCH data words. Most registers physically smaller than eight bytes are right-justified with zero-fill. Reading a status summary register is an exception in that the status information repeats in each byte.

The IOU may disconnect the MCH without affecting subsequent MCH operations after the following:

- Reading one to eight bytes from any register.
- Writing one byte to a corrected error log register.
- Writing one byte to a uncorrectable error log register.

The following MCH operations on CP registers can be performed with the CP running or halted (when reading or writing registers which may change while being accessed, the CP should be halted to avoid erroneous results):

- Read CP status summary register.
- Read CP fault status register.
- Read CP corrected error log registers.
- Read CP options installed.
- Read CP equipment ID register.
- Read/write CP dependent environmental control register.
- Read/write test mode control registers.
- Clear errors.

To read/write other CP registers, the CP must be running since microcode accesses these registers. Refer to table II-2-26. When reading or writing registers which may change while being accessed, precautions must be taken as described in the operating systems manual listed in the preface.

MCH Programming for Master Clear/Clear Errors (Opcode 6/7)

These operations consist of a single function word output. The master clear immediately and arbitrarily clears the connected unit without regard to possible information loss. The clear-errors operation clears the connected unit error indicators. The unit concerned should be halted to avoid loss of possible (next) error reporting while the errors are cleared.

MCH Echo (Opcode 8)

This operation checks the data path between the MCH and the IOU MAC. Following the operation MCH is activated and two bytes are sent to IOU MAC. IOU ignores the first byte and latches the second byte in the Address Holding Register, in any data pattern. MCH is deactivated after the second byte is accepted in IOU MAC and the channel is activated followed by an input sequence. IOU MAC sends data (contents of Address Holding Register) upon receiving the Active signal and subsequent Empty signals. There is no restriction on the number of data words read.

MCH Programming for Read IOU Summary Status (Opcode C, IOU Only)

This operation is an alternative, faster means of reading the IOU summary status register. In general terms, proceed as follows:

1. Issue function with opcode c.
2. Input summary status byte.

Table II-2-26. MCH Function Word Bit Assignments (Sheet 1 of 2)

Field	Code (Hex)	Description
MCH Function Word to IOU		
CONNECT (bits 8-11)	0	Connect IOU maintenance registers.
	8-F	PP-to-PP communications.
OPCODE	4	Prepare for read (control word required).
	5	Prepare for write (control word required).
	6	Master clear ADU and R barrel.
	7	Clear fault status registers.
	8	Echo.
	C	Read IOU summary status (reads one byte, control word not required).
TYPE (bits 0-3)	0-7	IOU registers are read circularly (byte 0 follows byte 7) from the byte specified by the TYPE field.
MCH Function Word to CM (Models 810, 815, 825, 830, and 835 Only)		
CONNECT (bits 8-11)	1	Connect CM maintenance register.
OPCODE	4	Prepare for read (control word required).
	5	Prepare for write (control word required).
	6	Master clear.
	7	Clear fault status register.
	8	Echo.

Table II-2-26. MCH Function Word Bit Assignments (Sheet 2 of 2)

Field	Code (Hex) (Hexadecimal)	Description
MCH Function Word to CP		
CONNECT (bits 8-11)	2	Connect CP maintenance registers.
OPCODE	0	Halt processor.
	1	Start processor.
	4	Prepare for read (control word required).
	5	Prepare for write (control word required).
	6	Master clear.
	7	Clear errors.
	8	Echo.
TYPE (Models 810, 815, 825, 830, and 835	0	Control word required.
	1	Read/write cache data buffer.
	2	Read/write map segment files.
	3	Read maintenance scan.
	4	Read map page files.
	5	Read/write register file A.
	6	Read/write register file B.
	7	Write maintenance scan limit.
	8	Read/write control store.
	A	Read AD register.
TYPE (Models 840, 845, 850, 855, and 860)	0	CP process state register.
	1	Control store micrand data.
	3	Maintenance access control reference (ROM).
	4	Soft control memories.
	5	BDP control memories.
	6	Instruction fetch decode memories.
	7	Register file.
	A	CMC maintenance registers.
TYPE (Model 990)	0	CP process state register.
	1	Control store micrand data.
	2	Maintenance access control (echo) function.
	3	Reserved for future use.
	4	Soft control memories.
	5	BDP control memories.
	6	Operand cache.
	7	Register file.
	8	Load and store section control memories.
	9	Error processing network.
	A	CMC maintenance registers.
	B	Maintenance access control extended echo.
	C-F	Reserved for future use.

GLOSSARY A

A register	Address register
ASCII	American Standard Code for Information Interchange
ASID	Active segment identifier
BC	Base constant
BDP	Business data processing
BN	Byte number
BS	Binding section
CBP	Code base pointer
CEJ/MEJ	Central exchange jump/monitor exchange jump
CEL	Corrected error log
CEM	Configuration environment monitor
CF	Critical frame pointer
CFF	Critical frame flag
CM	Central memory
CP	Central processing unit
CSF	Current stack frame pointer
DEC	Model-dependent environment control
DI	Debug index
DLP	Debug list pointer
DM	Debug mask
DMR	Debug mask register
DSC	Display station controller
DSP	Dynamic space pointer
DSR	Data set ready
DTR	Data terminal ready
EBCDIC	Expanded binary coded decimal interchange code
ECL	Emitter-coupled logic
ECM	Extended central memory
ECS	Extended core storage
EIA	Electronics Industries Association
EID	Element identifier
EPF	External procedure flag
ES	End suppression toggle (BDP edit instruction)
FIFO	First-in, first-out
FL	Field length
FLC	Central memory field length register
FLE	Extended core storage field length register
FP	Floating-point
G/L	Global/local
IC	Integrated circuit
ILH	Instruction look-ahead
I/O	Input/output
IOU	Input/output unit
JPS	Job process state pointer
KEY	Key
LOCK	Lock
LPID	Last processor identification
LRN	Largest ring number
LSI	Large-scale integration
MA	Monitor address
MAC	Maintenance access control

MCH	Maintenance channel
MCR	Monitor condition register
MDF	Model-dependent flags
MDW	Model-dependent word
MF	Monitor flag
MMR	Monitor mask register
MOP	Micro-operator (BDP edit instruction)
MOS	Metal-oxide-semiconductor
MPS	Monitor process state pointer
NOS	Network Operating System
NS	Negative sign toggle
OCF	On-condition flag
OI	Options installed
ON	Occurrence number
OP CODE	Operation code
P register	Program address register
PFA	Page frame address
PFS	Processor fault status
PID	Processor identifier
PIT	Process interval timer
PMF	Performance monitoring flag
PN	Page number
PND	Process-not-damaged flag
PO	Page offset
PP	Peripheral processor
PPM	Peripheral processor memory
PSA	Previous save area pointer
PSF	Previous stack frame
PSM	Page size mask
PTA	Page table address
PTE	Page table entry
PTL	Page table length
PTM	Processor test mode
PVA	Process virtual address
RAC	Central memory reference address register
RAE	Extended core storage reference address register
RAM	Random-access-memory
RMA	Real memory address
RN	Ring number
ROM	Read-only memory
RP	Read permission (segment descriptor field)
RTS	Request to send
SCT	Special characters table (BDP edit intruction)
SDE	Segment descriptor table entries
SDT	Segment descriptor table
SECDED	Single error correction/double error detection
SEG	Process segment number
SFSA	Stack frame save area
SIT	System interval timer
SM	The symbol (BDP edit instruction)
SN	Negative sign (BDP edit instruction)
SPID	Segment page identifier
SPT	System page table
SRT	Subscript range table
SS	Status summary
STA	Segment table address
STL	Segment table length
SV	Specification value

SVA	System virtual address
TED	Trap-enable delay
TEF	Trap-enable flip-flop
TOS	Top of stack
UART	Universal asynchronous receiver-transmitter
UCR	User condition register
UEL	Uncorrected error log
UMR	User mask register
UTP	Untranslatable pointer
UVMID	Untranslatable virtual machine identifier
VC	Search control code (page descriptor field)
VL	Segment validation (segment descriptor field)
VMCL	Virtual machine capability list
VMID	Virtual machine identifier
WP	Write access control (segment descriptor field)
XP	Execute access control (segment descriptor field)
ZF	Zero field toggle (BDP edit instruction)

EDIT EXAMPLES B

This appendix contains edit examples for the BDP edit instruction (ED).

NOTE

For examples in this appendix, the destination field is assumed to have the same length and decimal point position as the source field, except for the differences necessitated by insertion characters.

Edit Masks 1 through 25.

These edit masks are used in the examples given in the following pages.

Mask Number	COBOL Picture	Edit Mask (Hexadecimal with insertion characters *, $, 0, /, b, C and R shown as alphanumerics)
1	$ZZ,ZZ9.99	08 96 72 C4 72 01 95 02
2	$ZZ,ZZZ.99	07 96 72 C4 73 95 02
3	$ZZ,ZZZ.ZZ	08 96 72 C4 73 95 02 FA
4	-ZZZZ9.99	06 B3 74 01 95 02
5	ZZZZ9.99+	07 74 01 95 02 52 98
6	ZZ.999,99	06 72 C5 03 94 02 (Decimal point is comma)
7	$$$$.99CR	0B 61 $ 73 80 95 02 62 C R B8
8	$$$,$$$.$$	0A 61 $ 72 C4 73 80 95 02 FA
9	$$$$99,99CR	0C 61 $ 73 80 02 94 02 62 C R B8
10	$$$,$$9.99	DA 61 $ 72 C4 72 80 01 95 02
11	$99.99	05 96 02 95 02
12	$**,**9.99	0A 96 D1 * 72 C4 72 01 95 02
13	$**,***.**BCR	11 96 D1 * 72 C4 73 95 02 63 b C R B8 F7 95 E5
14	$**,***.**	0C 96 D1 * 72 C4 73 95 02 F7 95 E2
15	**,***,**+	0D D1 * 72 C4 73 95 02 52 98 F6 95 E3
16	--99999,99	07 50 71 80 05 94 02
17	----.99	06 50 73 80 95 02
18	++++99	05 52 73 80 02

Mask Number	COBOL Picture	Edit Mask (Hexadecimal with insertion characters *, $, 0, /, b, C and R shown as alphanumerics)
19	00999.00	09 42 0 0 03 95 42 0 0
20	99,999	05 02 C4 03 F6 (blank when zero)
21	XX/XX/XX	08 12 41 / 12 41 / 12
22	BBB99.99-	09 43 b b b 02 95 02 B3
23	999.00	06 03 95 42 0 0
24	999.BB	06 03 95 42 b b
25	9B9B9	06 01 91 01 91 01 or 08 01 41 b 01 41 b 01

Edit Examples Using Edit Masks 1 through 8.

Example	Source Field	Mask Used	Destination Field
1	00000.00	1	$bbbbb0.00
2	00000.01	1	$bbbbb0.01
3	000000.10	1	$bbbbb0.10
4	00001.00	1	$bbbbb1.00
5	00010.00	1	$bbbb10.00
6	00100.00	1	$bbb100.00
7	01000.00	1	$b1,000.00
8	10000.00	1	$10,000.00
9	00000.00	2	$bbbbbb.00
10	00000.00	3	bbbbbbbbbb
11	00000.01	3	$bbbbbb.01
12	00001.00	3	$bbbbb1.00
13	10000.00	3	$10,000.00
14	-00000.00	4	-bbbb0.00
15	+00000.00	4	bbbbb0.00
16	-12345.67	5	12345.67-
17	+00012.34	5	bbb12.34+
18	00000.00	6	bbb000,00
19	01000.00	6	b1.000,00
20	-123.45	7	$123.45CR
21	-023.45	7	b$23.45CR
22	003.45	7	bb$3.45bb
23	000.45	7	bbb$.45bb
24	00000.00	8	bbbbbbbbbb
25	00000.01	8	bbbbbb$.01

Edit Examples Using Edit Masks 9 through 16.

Example	Source Field	Mask Used	Destination Field
26	00001.00	8	bbbbbb$.10
27	00001.00	8	bbbbb$1.00
28	00010.00	8	bbbb$10.00
29	00100.00	8	bbb$100.00
30	01000.00	8	b$1,000.00
31	10000.00	8	$10,000.00
32	-0000000	9	bbb$00,00CR
33	0010000	9	bb$100,00bb
34	0100000	9	b$1000,00bb
35	-1000000	9	$10000,00CR
36	00000.00	10	bbbbb$0.00
37	10000.00	10	$10,000.00
38	00.00	11	$00.00
39	12.34	11	$12.34
40	00000.00	12	$*****0.00
41	00000.01	12	$*****0.01
42	00000.10	12	$*****0.10
43	00001.00	12	$*****1.00
44	00010.00	12	$****10.00
45	00100.00	12	$***100.00
46	01000.00	12	$*1,000.00
47	10000.00	12	$10.000.00
48	00000.00	13	*******.*****
49	-00000.01	13	$**,***.01bCR
50	00000.01	13	$**,***.01bbb
51	-00000.00	13	*******.*****
52	00000.00	14	*******.**
53	-00000.01	14	$******.01
54	00000.00	15	******.***
55	-00000.00	15	******.***
56	12345.67	15	12,345.67+
57	-12345.67	15	12,345.67-
58	-00000000	16	b-00000,00
59	-12345678	16	-123456,78
60	00000000	16	bb00000,00
61	12345678	16	b123456,78

Edit Examples Using Edit Masks 17 through 25.

Example	Source Field	Mask Used	Destination Field
62	-000.00	17	bbb-.00
63	000.00	17	bbbb.00
64	-001.00	17	bb-1.00
65	010.00	17	bb10.00
66	-100.00	17	-100.00
67	00000	18	bbb+00
68	-00000	18	bbb-00
69	00012	18	bbb+12
70	-00123	18	bb-123
71	01234	18	b+1234
72	-12345	18	-12345
73	000	19	00000.00
74	-123	19	00123.00
75	123	19	00123.00
76	00000	20	bbbbbb
77	00001	20	00,001
78	HHMMSS	21	HH/MM/SS
79	-00.00	22	bbb00.00-
80	00.00	22	bbb00.00b
81	12.34	22	bbb12.34b
82	000	23	000.00
83	-123	23	123.00
84	123	23	123.00
85	000	24	000.bb
86	-123	24	123.bb
87	123	24	123.bb
88	000	25	0b0b0
89	123	25	1b2b3

Edit Mask 26.

```
COBOL Picture: $ $ $ $ , $ $ $ , $ $ $ , $ $ $ , $ $ $ , $ $ $
    Edit Mask: 11 61 $ 73 C4 73 C4 73 C4 73 80 95 03 94 03 FF E9
```

Example Number 90 Using Edit Mask Number 26.

```
Source Field:       0 0 0 0 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0

Destination Field: b b b b b b b b b b b b b b b b b b b b b b
```

Example Number 91 Using Edit Mask Number 26.

```
Source Field:       1 2 3 4 5 6 7 8 9 0 1 2 . 6 5 4 3 2 1

Destination Field: $ 1 2 3 , 4 5 6 , 7 8 9 , 0 1 2 . 6 5 4 , 3 2 1
```

INTERFACE INFORMATION C

INTERFACES

This appendix contains signal description and sequencing information for models 810 through 990 input/output channel interfaces. The following interfaces are available.

- External interface, 12-bit
- Maintenance channel interface
- Two-port multiplexer interface

TWELVE-BIT EXTERNAL INTERFACE

The 12-bit external interface uses bidirectional, synchronous communication to transmit data between bits 52 through 63 of the channel data register and a number of CDC CYBER 170 external devices. The transmission is over separate input and output coaxial cables using an AC transmission scheme. In addition to 13 data signals (12 data, 1 parity), the cables also transmit eight control signals from a PP to an external device, and four from an external device to a PP. Maximum cable length between repeaters is 21 meters (70 feet).

MAINTENANCE CHANNEL INTERFACE

The maintenance channel interface (channel 17_8) uses unidirectional, asynchronous communication. It transmits only 9 data bits (8 data, 1 parity) in each direction. The data transfers between bits 56 through 63 of the channel data register and the external device.

TWO-PORT MULTIPLEXER INTERFACE

The two-port multiplexer interface is an EIA standard RS-232 serial interface. Refer to this standard for more information.

SIGNALS

The following signals are described below.

TWELVE-BIT CHANNEL CONTROL SIGNALS

The 12-bit channel uses the following control signals:

Signal	Description
Active Pulse	Sent by a data sending device to a data receiving device to begin a data transmission. Normally sent by a PP to an external device. An external device can send this signal to a PP only on the 12-bit channel.
Inactive Pulse	Sent from either a data sending or data receiving device to signify end of a data transmission; clears active and full flags.
Full Pulse	Sent from a data sending device to a data receiving device with transmitted data. The full pulse directs the receiving device to sample the data signals.
Empty Pulse	Sent by a data receiving device to a data sending device to acknowledge receipt of a full pulse and associated data. This pulse signals the sender to transmit more data.
Function Pulse	Sent by IOU to an external device to indicate that the associated data signals are control signals.
Master Clear	Sent by IOU to all external devices on the I/O channel. It indicates to those devices that all activity is to cease and initial conditions are to be restored.
10-Megahertz Clock	Consists of a pulse sent every 100 nanoseconds. This clock synchronizes all external devices to IOU.
1-Megahertz Clock	A pulse every 1 microsecond; sent by the IOU to an external device.

MAINTENANCE CHANNEL SIGNALS

The maintenance channel uses the following control signals.

Control Signals

Signal	Description
Active	Sent from IOU to external device to indicate start of data transmission.
Inactive	Sent from either data sending or data receiving device to indicate end of transmission.
Ready	Sent from data sending device to data receiving device with transmitted data; instructs receiving device to sample the data lines. The receiving device then sends a ready pulse back to acknowledge receipt of the data and to indicate it is ready for more data.
Function	Sent only to an external device to indicate that signals on data lines are control signals.
Error	Sent by external device to IOU to indicate that the device detected an error.

Signals and Cables

Table II-C-1 lists the signals used by the maintenance channel interface.

Table II-C-1. Maintenance Channel Signals (Sheet 1 of 2)

Signal Name	Connector Pins
Data out bit 2^0 (unidirectional)	A1/A2
Data out bit 2^1 (unidirectional)	A3/A4
Data out bit 2^2 (unidirectional)	A5/A6
Data out bit 2^3 (unidirectional)	A7/A8
Data out bit 2^4 (unidirectional)	A9/A10
Data out bit 2^5 (unidirectional)	B1/B2
Data out bit 2^6 (unidirectional)	B3/B4
Data out bit 2^7 (unidirectional)	B5/B6
Data out parity (unidirectional)	B7/B8

Table II-C-1. Maintenance Channel Signals (Sheet 2 of 2)

Signal Name	Connector Pins
Data in bit 2^0 (unidirectional)	C1/C2
Data in bit 2^1 (unidirectional)	C3/C4
Data in bit 2^2 (unidirectional)	C5/C6
Data in bit 2^3 (unidirectional)	C7/C8
Data in bit 2^4 (unidirectional)	C9/C10
Data in bit 2^5 (unidirectional)	D1/D2
Data in bit 2^6 (unidirectional)	D3/D4
Data in bit 2^7 (unidirectional)	D5/D6
Data in parity (unidirectional)	D7/D8
Function out	E1/E2
Ready out	E3/E4
Spare	E5/E6
Active out	E7/E8
Inactive out	E9/E10
Ready in	F1/F2
Spare	F3/F4
Inactive in	F5/F6
Summary status in	F7/F8
Exchange accept in	D9/D10
Error in	B9/B10

DATA SIGNALS

A data sending device transmits the data signals to a data receiving device along with the associated full pulses. The IOU also transmits a function code over the data lines to an external device, with a function pulse.

PP AND CHANNEL INTERACTION

Channel transmissions are controlled by the active and full flags. When a PP executes an I/O instruction, the state of these flags is altered and control signal(s) sent to the external devices. When the devices send control signals to the IOU, the state of these flags is again altered.

ACTIVE FLAG

When the PP sets the active bit with a 00740 or 00741 instruction, the IOU sends an active pulse on the external channel. The IOU sends an inactive pulse when the PP clears the active bit. An active pulse sent by an external device sets the active bit; an inactive pulse sent by an external device clears the active bit.

FULL FLAG

When a PP sets the full bit using a 00720, 00721, 0073, or 1073 instruction, the IOU sends a full pulse and data pulses for the data contained in the channel data register to an external device. When a PP clears the full bit with a 00700, 00701, or 1071 instruction, the system sends an empty pulse.

When an external device sends a full pulse, the associated data pulses set the channel data register, and the system sets the full bit. When an external device sends an empty pulse, the IOU clears the full bit.

FUNCTION INSTRUCTIONS

When a function instruction (00760, 00761, 00770, 00771) executes, the IOU sets active and full bits, writes a word into the channel data register, and transmits the word from the data register to an external device. The IOU transmits a function pulse to the external device to indicate that the word is a control signal rather than data. The external device sends an inactive pulse to acknowledge the receipt of the function, thereby setting the active bit and the full bit to zero.

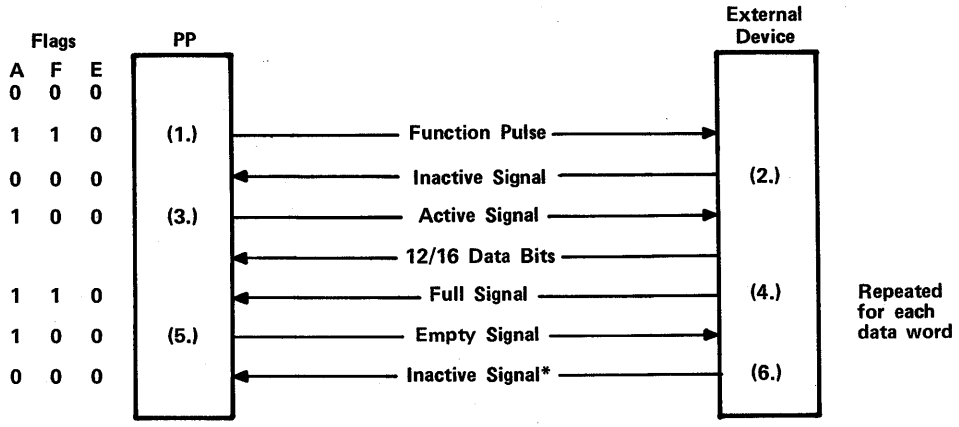
EXTERNAL CHANNEL INPUT/OUTPUT SEQUENCES

Tables II-C-2 (Data Input Sequence), and II-C-3 (Data Output Sequence) show the sequences followed by the channels during data input and output over an external interface.

Similarily, tables II-C-4 (MCH Input Sequence), and II-C-5 (MCH Output Sequence) show input and output sequences for the maintenance channel (17_8).

Data Sequences Timing is shown in figure II-C-1.

Table II-C-2. Data Input Sequence



*The inactive signal is normally sent from the external device to the IOU. However, in certain cases the IOU may deactivate the channel. This is determined by the external device and the function being executed.

Key:

A, F, E are the active, full and error flags

1. PP executes a function instruction which sets the active and full flags in the internal interface, places a word in the channel register, and sends a function pulse.

2. The external device acknowledges the acceptance of the function by sending an inactive signal which clears the active flag, the full flag, and the channel register.

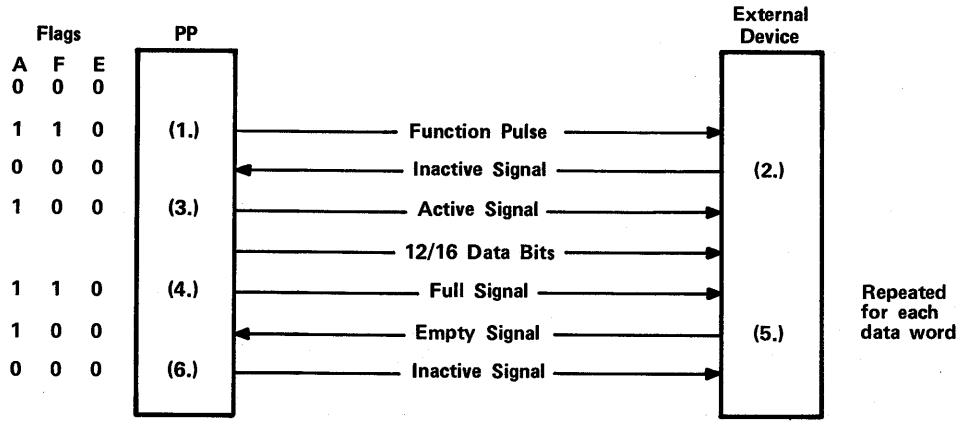
3. PP sets the active flag to indicate that data flow may start.

4. The external device sends a 12-bit word (plus parity) to the channel register, with a full signal which sets the full flag.

5. PP stores the data word in PPM and clears the full flag which, in turn, sends an empty signal to the external device.

6. Steps 4 and 5 repeat until the device completes the data transfer. Then the external device clears its active condition and sends an inactive signal to the PP, which clears the channel active flag.

Table II-C-3. Data Output Sequence



Key:

A, F, E are the active, full, and error flags.

1. PP executes a function instruction which sets the active and full bits in the internal interface, places a word into the channel register, and sends the function pulse.

2. The external device acknowledges the acceptance of the function by sending an inactive signal. This, in turn, clears the active flag, the full flag, and the channel register.

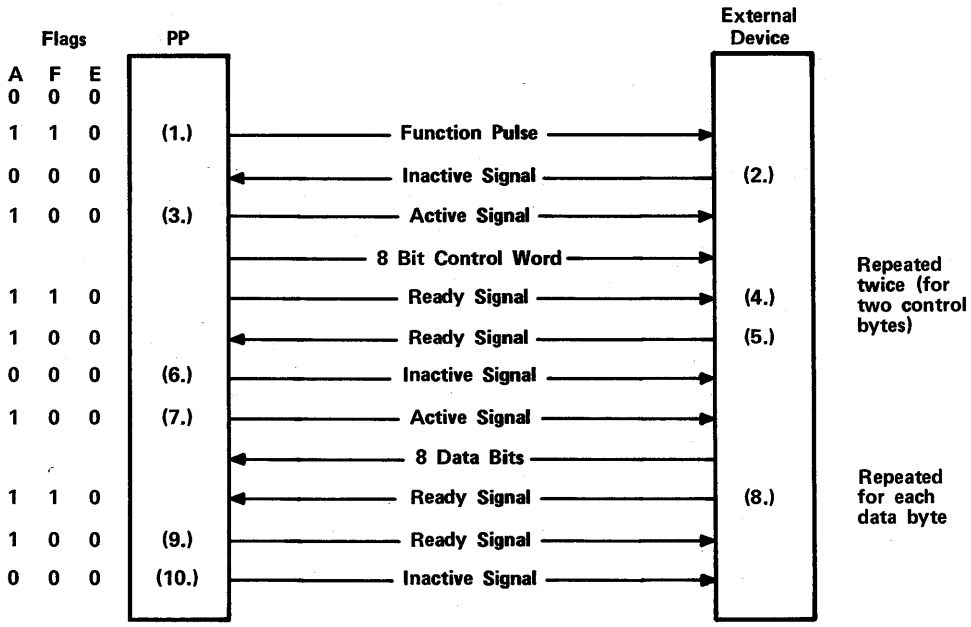
3. PP sets the active flag to indicate that data flow is about to start.

4. PP places a 12-bit data word (plus parity) into the channel register, which sets the full flag and sends the full signal.

5. The external device accepts the data word and sends an empty signal which clears the channel register and the full flag.

6. Steps 4 and 5 repeat until the PP has sent all the data to complete the data transfer. Then the PP clears the channel active flag, which turns off the external device with an inactive signal.

Table II-C-4. MCH Input Sequence



Key:

A, F, E are the active, full and error flags.

1. PP executes a function instruction which sets the active and full flags in the internal interface, places a word in the channel register and sends a function pulse.

2. The external device acknowledges the acceptance of the function by sending an inactive signal. This, in turn, clears the active flag, the full flag, and the channel register.

3. The PP sets the active flag to indicate that control word data flow is about to start.

4. The PP places a control byte into the channel register, which sets the full flag and sends the ready signal.

5. The external device accepts the control byte and sends the ready signal, which clears the channel register and the full flag.

6. Steps 4 and 5 are repeated for a second control byte. The two control bytes contain the upper and lower portions of the address of the data to be read.

7. The PP ensures that the channel is empty and then deactivates the channel, which clears the active flag.

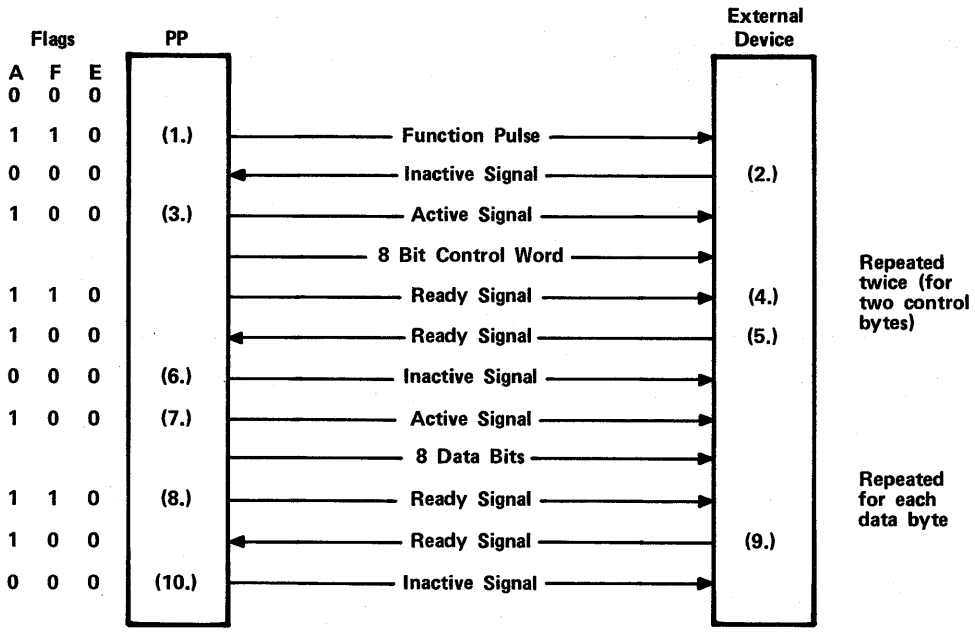
8. The PP sets the active flag to indicate that data flow may start.

9. The external device sends an 8-bit byte to the channel register with a ready signal which, in turn, sets the full flag.

10. The PP stores the data word and clears the full flag which, in turn, sends the ready signal to the external device.

11. Steps 8 and 9 repeat until the data transfer is complete. The PP deactivates the channel, which turns off the external device with an inactive signal.

Table II-C-5. MCH Output Sequence



Key:

A, F, E are the active, full and error flags.

1. PP executes a function instruction which sets the active and full bits in the internal interface, places a word in the channel register and sends the function pulse.

2. The external device acknowledges the acceptance of the function by sending an inactive signal. This, in turn, clears the active flag, the full flag, and the channel register.

3. The PP sets the active flag to indicate that control word data flow is about to start.

4. The PP places a control byte into the channel register, which sets the ready flag and sends the full signal.

5. The external device accepts the control byte and sends the ready signal which clears the channel register and the full flag.

6. The PP ensures that the channel is empty and then deactivates the channel, which clears the active flag.

7. The PP sets the active flag to indicate that data flow is about to start.

8. The PP places an 8-bit byte into the channel register, which sets the full flag and sends the ready signal.

9. The external device accepts the data byte and sends the ready signal, which clears the channel register and the full flag.

10. Steps 8 and 9 repeat a sufficient number of times to complete the data transfers. The PP deactivates the channel, which turns off the external device with an inactive signal.

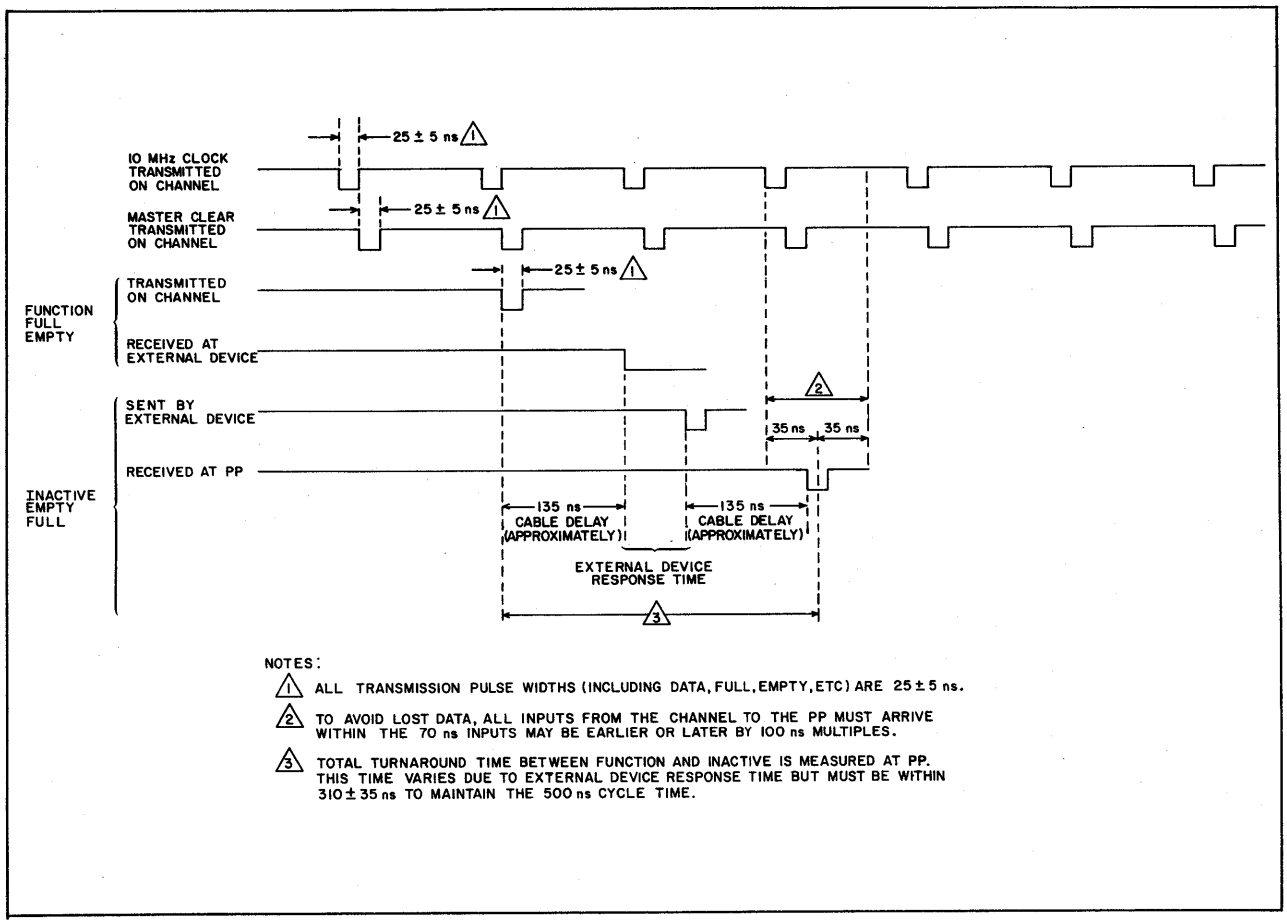


Figure II-C-1. Data Sequences Timing

INSTRUCTION INDEX

D

This appendix lists the central processor and peripheral processor instructions, in both opcode and mnemonic sequences.

CP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 1 of 4)

Opcode	Mnemonic	Instruction	Page
00	HALT	Program error	II-1-72
01	SYNC	Scope loop synchronization	II-1-73
02	EXCHANGE	Exchange	II-1-73
03	INTRUPT	Processor interrupt	II-1-84
04	RETURN	Return	II-1-74
05	PURGE	Purge buffer	II-1-86
06	POP	Pop	II-1-75
08	CPYMX	Copy free running counter	II-1-76
09	CPYAA	Copy address, A to A	II-1-21
0A	CPYXA	Copy address, X to A	II-1-21
0B	CPYAX	Copy address, A to X	II-1-21
0C	CPYRR	Copy half-word	II-1-21
0D	CPYXX	Copy full-word	II-1-22
0E	CPYSX	Copy from state register	II-1-88
0F	CPYXS	Copy to state register	II-1-88
10	INCX	Integer sum, immediate	II-1-13
11	DECX	Integer difference, immediate	II-1-14
14	LBSET	Test and set bit	II-1-77
16	TPAGE	Test and set page	II-1-77
17	LPAGE	Load page table index	II-1-84
18	IORX	Logical sum	II-1-28
19	XORX	Logical difference	II-1-28
1A	ANDX	Logical product	II-1-28
1B	NOTX	Logical complement	II-1-28
1C	INHX	Logical inhibit	II-1-29
1E	MARK	Mark to Boolean	II-1-30
1F	ENTZ	Enter zeros	II-1-24
1F	ENTO	Enter ones	II-1-24
1F	ENTS	Enter signs	II-1-24
20	ADDR	Half-word integer sum	II-1-12
21	SUBR	Half-word integer difference	II-1-14
22	MULR	Half-word integer product	II-1-13
23	DIVR	Half-word integer quotient	II-1-15
24	ADDX	Integer sum	II-1-13
25	SUBX	Integer difference	II-1-14

CP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 2 of 4)

Opcode	Mnemonic	Instruction	Page
26	MULX	Integer product	II-1-15
27	DIVX	Integer quotient	II-1-16
28	INCR	Half-word integer sum, immediate	II-1-12
29	DECR	Half-word integer difference, immediate	II-1-14
2A	ADDAX	Address increment, indexed	II-1-22
2C	CMPR	Half-word integer compare	II-1-16
2D	CMPX	Integer compare	II-1-16
2E	BRREL	Branch relative	II-1-17
2F	BRDIR	Inter-segment branch	II-1-18
30	ADDF	FP sum	II-1-55
31	SUBF	FP difference	II-1-55
32	MULF	FP product	II-1-56
33	DIVF	FP quotient	II-1-57
34	ADDD	Double-precision FP sum	II-1-55
35	SUBD	Double-precision FP difference	II-1-55
36	MULD	Double-precision FP product	II-1-56
37	DIVD	Double-precision FP quotient	II-1-57
39	ENTX	Enter X1, immediate logical	II-1-24
3A	CONI	Convert from integer to FP	II-1-53
3B	CONF	Convert from FP to integer	II-1-53
3C	CMPF	FP compare	II-1-60
3D	ENTP	Enter immediate, positive	II-1-24
3E	ENTN	Enter immediate, negative	II-1-24
3F	ENTL	Enter immediate, logical	II-1-24
40	ADDFV	FP vector sum	II-1-65
41	SUBFV	FP vector difference	II-1-65
42	MULFV	FP vector product	II-1-65
43	DIVFV	FP vector quotient	II-1-65
44	ADDXV	Integer vector sum	II-1-63
45	SUBXV	Integer vector difference	II-1-63
48	IORV	Logical vector sum	II-1-64
49	XORV	Logical vector difference	II-1-64
4A	ANDV	Logical vector product	II-1-64
4B	CNIFV	Convert vector from integer to FP	II-1-64
4C	CNFIV	Convert vector from FP to integer	II-1-64
4D	SHFV	Shift vector circular	II-1-65
50	CMPEQV	Integer vector compare, =	II-1-63
51	CMPLEV	Integer vector compare, $\leq$	II-1-63
52	CMPGEV	Integer vector compare, $\geq$	II-1-63
53	CMPNEV	Integer vector compare, $\neq$	II-1-63
54	MRGV	Merge vector	II-1-66
55	GTHV	Gather vector	II-1-66
56	SCTV	Scatter vector	II-1-68
57	SUMFV	FP vector summation	II-1-70
70	ADDN	Decimal sum	II-1-34

CP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 3 of 4)

Opcode	Mnemonic	Instruction	Page
71	SUBN	Decimal difference	II-1-34
72	MULN	Decimal product	II-1-34
73	DIVN	Decimal quotient	II-1-34
74	CMPN	Decimal compare	II-1-35
75	MOVN	Numeric move	II-1-36
76	MOVB	Move bytes	II-1-40
77	CMPB	Byte compare	II-1-39
80	LMULT	Load multiple	II-1-6
81	SMULT	Store multiple	II-1-6
82	LX	Load word	II-1-7
83	SX	Store word	II-1-7
84	LA	Load address	II-1-8
85	SA	Store address	II-1-8
86	LBYTP,j	Load bytes, relative	II-1-9
87	ENTC	Enter X1, signed immediate	II-1-25
88	LBIT	Load bit	II-1-10
89	SBIT	Store bit	II-1-10
8A	ADDRQ	Half-word integer sum, signed immediate	II-1-13
8B	ADDXQ	Integer sum, signed immediate	II-1-13
8C	MULRQ	Half-word integer product, signed immediate	II-1-15
8D	ENTE	Enter, signed immediate	II-1-25
8E	ADDAQ	Address increment, signed immediate	II-1-23
8F	ADDPXQ	Address relative	II-1-23
90	BRREQ	Branch on half-word equal	II-1-18
91	BRRNE	Branch on half-word not equal	II-1-18
92	BRRGT	Branch on half-word greater than	II-1-18
93	BRRGE	Branch on half-word greater than or equal	II-1-18
94	BRXEQ	Branch on equal	II-1-19
95	BRXNE	Branch on not equal	II-1-19
96	BRXGT	Branch on greater than	II-1-19
97	BRXGE	Branch on greater than or equal	II-1-19
98	BRFEQ	FP branch on equal	II-1-59
99	BRFNE	FP branch on not equal	II-1-59
9A	BRFGT	FP branch on greater than	II-1-59
9B	BRFGE	FP branch on greater than or equal	II-1-59
9C	BRINC	Branch and increment	II-1-19
9D	BRSEG	Branch on segments unequal	II-1-20
9E	BROVR	FP branch on overflow	II-1-59
9E	BRUND	FP branch on underflow	II-1-59
9E	BRINF	FP branch on indefinite	II-1-59
9F	BRCR	Branch on condition register	II-1-88
A0	LAI	Load address, indexed	II-1-8
A1	SAI	Store address, indexed	II-1-8
A2	LXI	Load word, indexed	II-1-7
A3	SXI	Store word, indexed	II-1-7

CP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 4 of 4)

Opcode	Mnemonic	Instruction	Page
A4	LBYT	Load bytes	II-1-9
A5	SBYT	Store bytes	II-1-9
A7	ADDAD	Address increment, modulo	II-1-23
A8	SHFC	Shift word, circular	II-1-26
A9	SHFX	Shift word, end-off	II-1-27
AA	SHFR	Shift half-word, end-off	II-1-27
AC	ISOM	Isolate bit mask	II-1-30
AD	ISOB	Isolate	II-1-30
AE	INSB	Insert	II-1-30
B0	CALLREL	Call relative	II-1-78
B2	MULXQ	Integer product, signed immediate	II-1-15
B3	ENTA	Enter X0, signed immediate	II-1-25
B4	CMPPXA	Compare swap	II-1-80
B5	CALLSEG	Call indirect	II-1-81
BE	xxxx	Reserved for user	II-1-83
BF	xxxx	Reserved for user	II-1-83
C0-7	EXECUTE,S	Execute algorithm	II-1-83
D0-7	LBYTS,S	Load bytes, immediate	II-1-9
D8-F	SBYTS,S	Store bytes, immediate	II-1-9
E4	SCLN	Decimal scale	II-1-37
E5	SCLR	Decimal scale, rounded	II-1-37
E9	CMPC	Byte compare, collated	II-1-39
EB	TRANB	Byte translate	II-1-40
ED	EDIT	Edit	II-1-41
F3	SCNB	Byte scan while nonmember	II-1-48
F4	CALDF	Calculate subscript and add	II-1-49
F9	MCVI	Move immediate data	II-1-50
FA	CMPI	Compare immediate data	II-1-51
FB	ADDI	Add immediate data	II-1-52

CP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 1 of 4)

Mnemonic	Opcode	Instruction	Page
ADDAD	A7	Address increment, modulo	II-1-23
ADDAQ	8E	Address increment, signed immediate	II-1-23
ADDAX	2A	Address increment, indexed	II-1-22
ADDD	34	Double-precision FP sum	II-1-55
ADDF	30	FP sum	II-1-55
ADDFV	40	FP vector sum	
ADDI	FB	Add immediate data	II-1-52
ADDN	70	Decimal sum	II-1-34
ADDPXQ	8F	Address relative	II-1-23
ADDR	20	Half-word integer sum	II-1-12
ADDRQ	8A	Half-word integer sum, signed immediate	II-1-13
ADDX	24	Integer sum	II-1-13
ADDXQ	8B	Integer sum, signed immediate	II-1-13
ADDXV	44	Integer vector sum	
ANDX	1A	Logical product	II-1-28
ANDV	4A	Logical vector product	
BRCR	9F	Branch on condition register	II-1-88
BRDIR	2F	Inter-segment branch	II-1-18
BRFEQ	98	FP branch on equal	II-1-59
BRFGE	9B	FP branch on greater than or equal	II-1-59
BRFGT	9A	FP branch on greater than	II-1-59
BRFNE	99	FP branch on not equal	II-1-59
BRINC	9C	Branch and increment	II-1-19
BRINF	9E	FP branch on indefinite	II-1-59
BROVR	9E	FP branch on overflow	II-1-59
BRREL	2E	Branch relative	II-1-17
BRREQ	90	Branch on half-word equal	II-1-18
BRRGE	93	Branch on half-word greater than or equal	II-1-18
BRRGT	92	Branch on half-word greater than	II-1-18
BRRNE	91	Branch on half-word not equal	II-1-18
BRSEG	9D	Branch on segments unequal	II-1-20
BRUND	9E	FP branch on underflow	II-1-59
BRXEQ	94	Branch on equal	II-1-19
BRXGE	97	Branch on greater than or equal	II-1-19
BRXGT	96	Branch on greater than	II-1-19
BRXNE	95	Branch on not equal	II-1-19
CALDF	F4	Calculate subscript and add	II-1-49
CALLREL	B0	Call relative	II-1-78
CALLSEG	B5	Call indirect	II-1-81
CMPB	77	Byte compare	II-1-39
CMPC	E9	Byte compare, collated	II-1-39
CMPEQV	50	Integer vector compare, =	II-1-63
CMPF	3C	FP compare	II-1-60
CMPGEV	52	Integer vector compare, $\geq$	II-1-63
CMPI	FA	Compare immediate data	II-1-51

CP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 2 of 4)

Mnemonic	Opcode	Instruction	Page
CMPLEV	51	Integer vector compare, $\leq$	II-1-63
CMPN	74	Decimal compare	II-1-38
CMPNEV	53	Integer vector compare, $\neq$	II-1-63
CMPPXA	B4	Compare swap	II-1-80
CMPR	2C	Half-word integer compare	II-1-16
CMPX	2D	Integer compare	II-1-16
CNFIV	4C	Convert vector from FP to integer	II-1-64
CNIFV	4B	Convert vector from integer to FP	II-1-64
CONF	3B	Convert from FP to integer	II-1-53
CONI	3A	Convert from integer to FP	II-1-53
CPYAA	09	Copy address, A to A	II-1-21
CPYAX	0B	Copy address, A to X	II-1-21
CPYMX	08	Copy free running counter	II-1-76
CPYRR	0C	Copy half-word	II-1-21
CPYSX	0E	Copy from state register	II-1-88
CPYXA	0A	Copy address, X to A	II-1-21
CPYXS	0F	Copy to state register	II-1-88
CPYXX	0D	Copy full-word	II-1-22
DECR	29	Half-word integer difference, immediate	II-1-14
DECX	11	Integer difference, immediate	II-1-14
DIVD	37	Double-precision FP quotient	II-1-57
DIVF	33	FP quotient	II-1-57
DIVFV	43	FP vector quotient	II-1-65
DIVN	73	Decimal quotient	II-1-34
DIVR	23	Half-word integer quotient	II-1-15
DIVX	27	Integer quotient	II-1-16
EDIT	ED	Edit	II-1-41
ENTA	B3	Enter X0, signed immediate	II-1-25
ENTC	87	Enter X1, signed immediate	II-1-25
ENTE	8D	Enter signed immediate	II-1-25
ENTL	3F	Enter, immediate logical	II-1-24
ENTN	3E	Enter, immediate negative	II-1-24
ENTO	1F	Enter ones	II-1-24
ENTP	3D	Enter, immediate positive	II-1-24
ENTS	1F	Enter signs	II-1-24
ENTX	39	Enter X1, immediate logical	II-1-24
ENTZ	1F	Enter zeros	II-1-24
EXCHANGE	02	Exchange	II-1-73
EXECUTE,S	C0-7	Execute alogrithm	II-1-83
GTHV	55	Gather vector	II-1-66
HALT	00	Program error	II-1-72
INCR	28	Half-word integer sum, immediate	II-1-12
INCX	10	Integer sum, immediate	II-1-13
INHX	1C	Logical inhibit	II-1-29
INSB	AE	Insert	II-1-30
INTRUPT	03	Processor interrupt	II-1-84

CP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 3 of 4)

Mnemonic	Opcode	Instruction	Page
IORV	48	Logical vector sum	II-1-64
IORX	18	Logical sum	II-1-28
ISOB	AD	Isolate	II-1-30
ISOM	AC	Isolate bit mask	II-1-30
LA	84	Load address	II-1-8
LAI	A0	Load address, indexed	II-1-8
LBIT	88	Load bit	II-1-10
LBSET	14	Test and set bit	II-1-77
LBYT	A4	Load bytes	II-1-9
LBYTP,j	86	Load bytes, relative	II-1-9
LBYTS,S	D0-7	Load bytes, immediate	II-1-9
LMULT	80	Load multiple	II-1-6
LPAGE	17	Load page table index	II-1-84
LX	82	Load word	II-1-7
LXI	A2	Load word, indexed	II-1-7
MARK	1E	Mark to Boolean	II-1-30
MCVI	F9	Move immediate data	II-1-50
MOVB	76	Move bytes	II-1-40
MOVN	75	Numeric move	II-1-36
MRGV	54	Merge vector	II-1-66
MULD	36	Double-precision FP product	II-1-56
MULF	32	FP product	II-1-56
MULFV	42	FP vector product	II-1-65
MULN	72	Decimal product	II-1-34
MULR	22	Half-word integer product	II-1-13
MULRQ	8C	Half-word integer product, signed immediate	II-1-15
MULX	26	Integer product	II-1-15
MULXQ	B2	Integer product, signed immediate	II-1-15
NOTX	1B	Logical complement	II-1-28
POP	06	Pop	II-1-75
PURGE	05	Purge buffer	II-1-86
RETURN	04	Return	II-1-74
SA	85	Store address	II-1-8
SAI	A1	Store address, indexed	II-1-10
SBIT	89	Store bit	II-1-11
SBYT	A5	Store bytes	II-1-9
SBYTS,S	D8-F	Store bytes, immediate	II-1-9
SCLN	E4	Decimal scale	II-1-37
SCLR	E5	Decimal scale rounded	II-1-37
SCNB	F3	Byte scan while nonmember	II-1-48
SCTV	56	Scatter vector	II-1-68
SHFC	A8	Shift word, circular	II-1-26
SHFR	AA	Shift half-word, end-off	II-1-27

CP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 4 of 4)

Mnemonic	Opcode	Instruction	Page
SHFV	4D	Shift vector circular	II-1-65
SHFX	A9	Shift word, end-off	II-1-27
SMULT	81	Store multiple	II-1-6
SUBD	35	Double-precision FP difference	II-1-55
SUBF	31	FP difference	II-1-55
SUBFV	41	FP vector difference	II-1-65
SUBN	71	Decimal difference	II-1-34
SUBR	21	Half-word integer difference	II-1-14
SUBX	25	Integer difference	II-1-14
SUBFV	45	Integer vector difference	II-1-63
SUMFV	57	FP vector summation	II-1-70
SX	83	Store word	II-1-7
SXI	A3	Store word, indexed	II-1-7
SYNC	01	Scope loop synchronization	II-1-73
TPAGE	16	Test and set page	II-1-77
TRANB	EB	Byte translate	II-1-40
XORV	49	Logical vector difference	II-1-64
XORX	19	Logical difference	II-1-28
xxxx	BE	Reserved for user	II-1-83
xxxx	BF	Reserved for user	II-1-83

PP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 1 of 4)

Opcode	Mnemonic	Instruction	Page
0000	-	Pass	II-1-135
0001dm	LJM m,d	Long jump to m+(d)	II-1-113
0002dm	RJM m,d	Return jump to m+(d)	II-1-114
0003d	UJN d	Unconditional jump d	II-1-114
0004d	ZJN d	Zero jump d	II-1-114
0005d	NJN d	Nonzero jump d	II-1-115
0006d	PJN d	Plus jump d	II-1-115
0007d	MJN d	Minus jump d	II-1-115
0010d	SHN d	Shift A by d	II-1-104
0011d	LMN d	Logical difference d	II-1-104
0012d	LPN d	Logical product d	II-1-104
0013d	SCN d	Selective clear d	II-1-105
0014d	LDN d	Load d	II-1-94
0015d	LCH d	Load complement d	II-1-94
0016d	ADN d	Add d	II-1-98
0017d	SBN d	Subtract d	II-1-98
0020dm	LDC m,d	Load dm	II-1-94
0021dm	ADC m,d	Add dm	II-1-99
0022dm	LPC m,d	Logical product dm	II-1-105
0023dm	LMC m,d	Logical difference dm	II-1-105
002400	PSN	Pass	II-1-135
0024d	LRD d	Load R	II-1-117
002500	-	Pass	II-1-135
0025d	SRD d	Store R	II-1-117
00260X	EXN	Exchange jump	II-1-136
00261X	MXN	Monitor exchange jump	II-1-136
00262X	MAN	Monitor exchange jump MA	II-1-136
0027X	KPT	Keypoint	II-1-135
0030d	LDD d	Load (d)	II-1-94
0031d	ADD d	Add (d)	II-1-99
0032d	SBD d	Subtract (d)	II-1-99
0033d	LMD d	Logical difference (d)	II-1-106
0034d	STD d	Store (d)	II-1-95
0035d	RAD d	Replace add (d)	II-1-109
0036d	AOD d	Replace add one (d)	II-1-109
0037d	SOD d	Replace subtract one (d)	II-1-109
0040d	LDI d	Load ((d))	II-1-95
0041d	ADI d	Add ((d))	II-1-100
0042d	SBI d	Subtract ((d))	II-1-100
0043d	LMI d	Logical difference ((d))	II-1-107

PP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 2 of 4)

Opcode	Mnemonic	Instruction	Page
0044d	STI d	Store ((d))	II-1-96
0045d	RAI d	Replace add ((d))	II-1-110
0046d	AOI d	Replace add one ((d))	II-1-110
0047d	SOI d	Replace subtract one ((d))	II-1-111
0050dm	LDM m,d	Load (m+(d))	II-1-96
0051dm	ADM m,d	Add (m+(d))	II-1-101
0052dm	SBM m,d	Subtract (m+(d))	II-1-102
0053dm	LMM m,d	Logical difference (m+(d))	II-1-107
0054dm	STM m,d	Store (m+(d))	II-1-96
0055dm	RAM m,d	Replace add (m+(d))	II-1-111
0056dm	AOM m,d	Replace add one (m+(d))	II-1-112
0057dm	SOM m,d	Replace subtact one (m+(d))	II-1-112
0060d	CRD d	Central read from (A) to d	II-1-118
0061dm	CRM m,d	Central read (d) words from (A) to m	II-1-119
0062d	CWD d	Central write to (A) from d	II-1-120
0063dm	CWM m,d	Central write (d) words to (A) from m	II-1-122
00640cm	AJM m,c	Jump to m if channel C active	II-1-125
00641cm	SCF m,c	Test to m and set channel C flag	II-1-125
00650cm	IJM m,c	Jump to m if channel C inactive	II-1-126
00651cm	CCF c	Clear channel C flag	II-1-126
00660cm	FJM m,c	Jump to m if channel C full	II-1-127
00661cm	SFM m,c	Jump to m if channel C error flag set	II-1-127
00670cm	EJM m,c	Jump to m if channel C empty	II-1-127
00671cm	CFM m,c	Jump to m if channel C error flag clear	II-1-127
00700c	IAN c	Input to A from channel C when active	II-1-128
00701c	IAN 40B+c	Input to A from channel C if active	II-1-128
0071Xcm	IAM m,c	Input A words to m from channel C	II-1-128
00720c	OAN	Output from A on channel C when active	II-1-130
00721c	OAN 40B+c	Output from A on channel C if active	II-1-131
0073xcm	OAM m+c	Output A words from m on channel C	II-1-131
00740c	ACN c	Activate channel C	II-1-132
00741c	ACN 40B+c	Unconditionally activate channel C	II-1-132
00750c	DCN c	Deactivate channel C	II-1-132
00751c	DCN 40B+c	Unconditionally deactivate channel C	II-1-133
00760c	FAN c	Function a on channel C when inactive	II-1-133
00761c	FAN 40B+c	Function a on channel C if inactive	II-1-134
00770cm	FNC m,c	Function m on channel C when inactive	II-1-134
00771cm	FNC m,40B+c	Function m on channel C if inactive	II-1-135
1000d	RDSL	Central read and set lock from d to (A)	II-1-123
1001d	RDCL	Central read and clear lock from d to (A)	II-1-123

PP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 3 of 4)

Opcode	Mnemonic	Instruction	Page
1002	-	Pass	II-1-135
1003	-	Pass	II-1-135
1004	-	Pass	II-1-135
1005	-	Pass	II-1-135
1006	-	Pass	II-1-135
1007	-	Pass	II-1-135
1010	-	Pass	II-1-135
1011	-	Pass	II-1-135
1012	-	Pass	II-1-135
1013	-	Pass	II-1-135
1014	-	Pass	II-1-135
1015	-	Pass	II-1-135
1016	-	Pass	II-1-135
1017	-	Pass	II-1-135
1020	-	Pass	II-1-135
1021	-	Pass	II-1-135
1022d	LPDL d	Logical product (d) long	II-1-105
1023d	LPIL d	Logical product ((d)) long	II-1-106
1024dm	LPML m,d	Logical product (m+(d) long	II-1-106
1025	-	Pass	II-1-135
1026d	INPN d	Interrupt processor	II-1-137
1027	-	Pass	II-1-135
1030d	LDDL d	Load (d) long	II-1-95
1031d	ADDL d	Add (d) long	II-1-99
1032d	SBDL d	Subtract (d) long	II-1-100
1033d	LMDL d	Logical difference (d) long	II-1-106
1034d	STDL d	Store (d) long	II-1-95
1035d	RADL d	Replace add (d) long	II-1-109
1036d	AODL d	Replace add one (d) long	II-1-109
1037d	SODL d	Replace subtract one (d) long	II-1-110
1040d	LDIL d	Load ((d)) long	II-1-96
1041d	ADIL d	Add ((d)) long	II-1-100
1042d	SBIL d	Subtract ((d)) long	II-1-101
1043d	LMIL d	Logical difference ((d)) long	II-1-107
1044d	STIL d	Store ((d)) long	II-1-96
1045d	RAIL d	Replace add ((d)) long	II-1-110
1046d	AOIL d	Replace add one ((d)) long	II-1-110
1047d	SOIL d	Replace subtract one ((d)) long	II-1-111
1050dm	LDML m,d	Load (m+(d)) long	II-1-97
1051dm	ADML m,d	Add (m+(d)) long	II-1-101

PP INSTRUCTIONS - OPCODE SEQUENCE (Sheet 4 of 4)

Opcode	Mnemonic	Instruction	Page
1052dm	SBML m,d	Subtract (m+(d)) long	II-1-102
1053dm	LMML m,d	Logical difference (m+(d)) long	II-1-107
1054dm	STML m,d	Store (m+(d)) long	II-1-97
1055dm	RAML m,d	Replace add (m+(d)) long	II-1-111
1056dm	AOML m,d	Replace add one (m+(d)) long	II-1-112
1057dm	SOML m,d	Replace subtract one (m+(d)) long	II-1-112
1060d	CRDL d	Central read from (A) to d long	II-1-118
1061dm	CRML m,d	Central read (d) words from (A) to m long	II-1-120
1062d	CWDL d	Central write to (A) from d long	II-1-121
1063dm	CWML m,d	Central write (d) words to (A) from m long	II-1-122
1064cm	TSJM m,c	Jump if channel C flag set	II-1-126
1065xcm	FCJM m,c	Jump if channel C flag clear	II-1-126
1066	-	Pass	II-1-135
1067	-	Pass	II-1-135
1070	-	Pass	II-1-135
1071xcm	IAPM m,c	Input A words to m from channel C packed	II-1-129
1072	-	Pass	II-1-135
1073xcm	OAPM m,c	Output A words from m on channel C packed	II-1-132
1074	-	Pass	II-1-135
1075	-	Pass	II-1-135
1076	-	Pass	II-1-135
1077	-	Pass	II-1-135

PP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 1 of 4)

Mnemonic	Opcode	Instruction	Page
-	0000	Pass	II-1-135
-	002500	Pass	II-1-135
-	1002	Pass	II-1-135
-	1003	Pass	II-1-135
-	1004	Pass	II-1-135
-	1005	Pass	II-1-135
-	1006	Pass	II-1-135
-	1007	Pass	II-1-135
-	1010	Pass	II-1-135
-	1011	Pass	II-1-135
-	1012	Pass	II-1-135
-	1013	Pass	II-1-135
-	1014	Pass	II-1-135
-	1015	Pass	II-1-135
-	1016	Pass	II-1-135
-	1017	Pass	II-1-135
-	1020	Pass	II-1-135
-	1021	Pass	II-1-135
-	1025	Pass	II-1-135
-	1027	Pass	II-1-135
-	1066	Pass	II-1-135
-	1067	Pass	II-1-135
-	1070	Pass	II-1-135
-	1072	Pass	II-1-135
-	1074	Pass	II-1-135
-	1075	Pass	II-1-135
-	1076	Pass	II-1-135
-	1077	Pass	II-1-135
ACN 40B+c	00741c	Unconditionally activate channel C	II-1-132
ACN c	00740c	Activate channel C	II-1-132
ADC m,d	0021dm	Add dm	II-1-99
ADD d	0031d	Add (d)	II-1-99
ADDL d	1031d	Add (d) long	II-1-99
ADI d	0041d	Add ((d))	II-1-100
ADIL d	1041d	Add ((d)) long	II-1-100
ADM m,d	0051dm	Add (m+(d))	II-1-101
AOM m,d	0056dm	Replace add one (m+(d))	II-1-112
ADML m,d	1051dm	Add (m+(d)) long	II-1-100
AOML m,d	1056dm	Replace add one (m+(d)) long	II-1-112
LCN d	0015d	Load complement d	II-1-94

PP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 2 of 4)

Mnemonic	Opcode	Instruction	Page
ADN d	0016d	Add d	II-1-98
AJM m,c	00640cm	Jump to m if channel C active	II-1-125
AOD d	0036d	Replace add one (d)	II-1-109
AODL d	1036d	Replace add one (d) long	II-1-109
AOI d	0046d	Replace add one ((d)) long	II-1-110
AOIL d	1046d	Replace add one ((d)) long	II-1-110
CCF c	00651cm	Clear channel C flag	II-1-126
CFM m,c	00671cm	Jump to m if channel C error flag clear	II-1-127
CRD d	0060d	Central read from (A) to d	II-1-118
CRDL d	1060d	Central read from (A) to d long	II-1-118
CRM m,d	0061dm	Central read (d) words from (A) to m	II-1-119
CRML m,d	1061dm	Central read (d) words from (A) to m long	II-1-120
CWD d	0062d	Central write to (A) from d	II-1-120
CWDL d	1062d	Central write to (A) from d long	II-1-121
CWM m,d	0063dm	Central write (d) words to (A) from m	II-1-122
CWML m,d	1063dm	Central write (d) words to (A) from m long	II-1-122
DCN 40B+c	00751c	Unconditionally deactivate channel C	II-1-133
DCN c	00750c	Deactivate channel C	II-1-132
EJM m,c	00670cm	Jump if channel C empty	II-1-127
EXN	00260X	Exchange jump	II-1-136
FAN 40B+c	00761c	Function A on channel C if inactive	II-1-134
FAN c	00760c	Function A on channel C when inactive	II-1-133
FCJM m,c	1065xcm	Jump if channel C flag clear	II-1-126
FJM m,c	00660cm	Jump if channel C full	II-1-127
FNC m,40B+c	00771cm	Function m on channel C if inactive	II-1-135
FNC m,c	00770cm	Function m on to channel C when active	II-1-134
IAM m,c	0071Xcm	Input A words from channel C to m	II-1-128
IAN 40B+c	00701c	Input To A from channel C if active	II-1-128
IAN c	00700c	Input To A from channel C when active	II-1-128
IAPM m,c	1071xcm	Input A words to m from channel C packed	II-1-129
IJM m,c	00650cm	Jump if channel C inactive	II-1-126
KPT	0027X	Keypoint	II-1-135
LDC m,d	0020dm	Load dm	II-1-94
LDD d	0030d	Load (d)	II-1-94
LDDL d	1030d	Load (d) long	II-1-95
LDI d	0040d	Load ((d))	II-1-95
LDIL d	1040d	Load ((d)) long	II-1-95

PP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 3 of 4)

Mnemonic	Opcode	Instruction	Page
LDM m,d	0050dm	Load (m+(d))	II-1-96
LDML m,d	1050dm	Load (m+(d)) long	II-1-97
LDN d	0014d	Load d	II-1-94
LJM m,d	0001dm	Long jump to m+(d)	II-1-113
LMD d	0033d	Logical difference (d)	II-1-106
LMDL d	1033d	Logical difference (d) long	II-1-106
LMI d	0043d	Logical difference ((d))	II-1-107
LMIL d	1043d	Logical difference ((d)) long	II-1-107
LMM m,d	0053dm	Logical difference (m+(d))	II-1-107
LMML m,d	1053dm	Logical difference (m+(d)) long	II-1-107
LMN d	0011d	Logical difference d	II-1-104
LPC m,d	0022dm	Logical product dm	II-1-105
LPDL d	1022d	Logical product (d) long	II-1-105
LPIL d	1023d	Logical product ((d)) long	II-1-106
LPML m,d	1024dm	Logical product (m+(d)) long	II-1-106
LPN d	0012d	Logical product d	II-1-104
LRN d	0024d	Load R	II-1-117
MAN	00262X	Monitor exchange jump MA	II-1-136
MJN d	0007d	Minus jump d	II-1-115
MXN	00261X	Monitor exchange jump	II-1-136
NJN d	0005d	Nonzero jump d	II-1-115
OAM m+c	0073xcm	Output (A) words from m on channel C	II-1-131
OAN	00720c	Output from (A) on channel C when active	II-1-130
OAN 40B+c	00721c	Output from A on channel C if active	II-1-131
OAPM m,c	1073xcm	Output A words from m on channel C packed	II-1-132
PJN d	0006d	Plus jump d	II-1-115
LMC m,d	0023dm	Logical difference dm	II-1-105
PSN	002400	Pass	II-1-135
RAD d	0035d	Replace add (d)	II-1-109
RADL d	1035d	Replace add (d) long	II-1-109
RAI d	0045d	Replace add ((d))	II-1-110
RAIL d	1045d	Replace add ((d)) long	II-1-110
RAM m,d	0055dm	Replace add (m+(d))	II-1-111
RAML m,d	1055dm	Replace add (m+(d)) long	II-1-111
RDCL	1001d	Central read and clear lock from d to (A)	II-1-123
RDSL	1000d	Central read and set lock from d to (A)	II-1-123
RJM m,d	0002dm	Return jump m+(d)	II-1-114
SBD d	0032d	Subtract (d)	II-1-99
SBI d	0042d	Subtract ((d))	II-1-100
SBDL d	1032d	Subtract (d) long	II-1-100

PP INSTRUCTIONS - MNEMONIC SEQUENCE (Sheet 4 of 4)

Mnemonic	Opcode	Instruction	Page
SBIL d	1042d	Subtract ((d)) long	II-1-101
SBM m,d	0052dm	Subtract (m+(d))	II-1-102
SBML m,d	1052dm	Subtract (m+(d)) long	II-1-102
SBN d	0017d	Subtract d	II-1-98
SCF m,c	00641cm	Test and set channel C flag	II-1-125
SFM m,c	00661cm	Jump to m if channel C error flag set	II-1-127
SOD d	0037d	Replace subtract one (d)	II-1-109
SODL d	1037d	Replace subtract one (d) long	II-1-110
SOI d	0047d	Replace subtract one ((d))	II-1-111
SOIL d	1047d	Replace subtract one ((d)) long	II-1-111
SOM m,d	0057dm	Replace subtract one (m+(d))	II-1-112
SOML m,d	1057dm	Replace subtract one (m+(d)) long	II-1-112
SRD d	0025d	Store R	II-1-117
STD d	0034d	Store (d)	II-1-95
STDL d	1034d	Store (d) long	II-1-95
STI d	0044d	Store ((d))	II-1-96
STIL d	1044d	Store ((d)) long	II-1-96
STM m,d	0054dm	Store (m+(d))	II-1-96
STML m,d	1054dm	Store (m+(d)) long	II-1-97
TSJM m,c	1064cm	Jump if channel C flag set	II-1-126
UJN d	0003d	Unconditional jump d	II-1-114
ZJN d	0004d	Zero jump d	II-1-114

INDEX

Address arithmetic instructions, CP II-1-22
Address translation, see Virtual and Central Memory Programming
Arithmetic instructions, floating-point II-1-54
Arithmetic instructions, PP II-1-97

BDP byte instructions II-1-38
BDP data descriptors II-2-40
BDP data types
    Slack digit II-2-45
    Type 0: packed decimal, unsigned II-2-42
    Type 1: packed decimal, unsigned slack digit II-2-42
    Type 10: binary, unsigned II-2-45
    Type 11: binary, signed II-2-45
    Type 2: packed decimal, signed II-2-43
    Type 3: packed decimal, signed, slack digit II-2-43
    Type 4: unpacked decimal, unsigned II-2-43
    Type 5: unpacked decimal, trailing sign combined Hollerith II-2-44
    Type 6: unpacked decimal, trailing sign separate II-2-44
    Type 7: unpacked decimal, leading sign combined Hollerith II-2-44
    Type 8: unpacked decimal, leading sign separate II-2-44
    Type 9: alphanumeric II-2-45
BDP instruction descriptions II-1-31
BDP instruction nomenclature II-1-32
BDP numeric instructions II-1-32
BDP operand types and field lengths II-2-42
BDP subscript and immediate data instructions II-1-48
BDP undefined results
    Invalid data II-2-45
    Overlap II-2-45
Branch instructions, CP II-1-17
Branch instructions, floating-point II-1-58
Branch instructions, PP II-1-113
Business data processing (BDP) programming II-2-40

Byte instructions, BDP II-1-38
Central memory access instructions, PP II-1-116
Central memory programming, see Virtual and Central Memory Programming
Character data word, display station II-2-133
Character mode word, display station II-2-130
Clock, real-time, programming II-2-136
CM registers
    Corrected error log (CEL) II-2-16
    Element identifier (EID) II-2-17
    Environment control (EC) II-2-17
    Free-running counter II-2-17
    Options installed (OI) II-2-17
    Port bounds II-2-17
    Status summary (SS) II-2-17
    Uncorrectable error log (UEL) II-2-18
Codes, display station II-2-133
Condition and mask registers, CP II-2-20
Conversion instructions, floating-point II-1-53
Coordinate data word, display station II-2-133
Copy instructions, CP II-1-20
CP address arithmetic instructions II-1-22
CP branch instructions II-1-17
CP condition and mask registers II-2-20
CP condition register bit grouping II-2-23
CP copy instructions II-1-20
CP exchange operations II-2-1
CP general instructions II-1-4
CP instruction description nomenclature II-1-2
CP instruction formats II-1-1
CP instruction index
    Mnemonic sequence II-D-5
    Opcode sequence II-D-1
CP integer arithmetic instructions II-1-11
CP interrupts II-1-3
    Exchange II-2-25
    Trap II-2-25
    Conditions, see Interrupt Conditions
CP load and store instructions II-1-5
CP logical instructions II-1-27
CP mark to Boolean instruction II-1-30
CP register bit string instructions II-1-29

CP registers
    Address A II-2-6
    Base constant (BC) II-2-6
    Cache/map corrected error log (CCEL/MCEL) II-2-14
    Debug index (DI) II-2-6
    Debug list pointer (DLP) II-2-6
    Debug mask (DM) II-2-7
    Dependent environment control (DEC) II-2-14
    Element identifier (EID) II-2-14
    Flag register
        Critical-frame flag (CFF) II-2-8
        On-condition flag (OCF) II-2-8
        Process-not-damaged (PND) flag II-2-8
    Job process state (JPS) II-2-14
    Largest ring number (LRN) II-2-8
    Last processor identification (LPID) II-2-8
    Model dependent word (MDW) II-2-14
    Monitor condition (MCR) II-2-8
    Monitor mask (MMR) II-2-8
    Monitor process state (MPS) II-2-15
    Operand X II-2-9
    Options installed (OI) II-2-13
    Page size mask (PSM) II-2-13
    Page table address (PTA) II-2-13
    Page table length (PTL) II-2-13
    Process interval timer (PIT) II-2-9
    Processor fault status (PFS) II-2-13
    Processor identifier (PID) II-2-13
    Processor test mode (PTM) II-2-13
    Program address (P) II-2-9
    Segment table address (STA) II-2-10
    Segment table length (STL) II-2-10
    Status summary (SS) II-2-13
    System interval timer (SIT) II-2-15
    Top-of-stack (TOS) pointer II-2-10
    Trap enable (TE)
        Trap-enable delay flip-flop (TEF) II-2-10
        Trap-enable flip-flop (TEF) II-2-10
    Trap pointer (TP) II-2-10
    Untranslatable pointer (UTP) II-2-11
    Untranslatable virtual machine identifier (UVMID) II-2-11
    User condition (UCR) II-2-11
    User mask (UMR) II-2-11
    Virtual machine capability list (VMCL) II-2-15
    Virtual machine identifier (VMID) II-2-12
CP shift instructions II-1-26
CYBER 170 State exchange package II-2-113

Data descriptors, BDP II-2-40
Data format, PP II-1-90
Data formats, FP II-2-49
Data sequences timing II-C-11
Data types, BDP, see BDP data types
Debug, see Program Monitoring
Dedicated channels, IOU II-2-136
Difference, FP II-2-52
Display station programming
    Character data word II-2-133
    Character mode II-2-130
    Codes II-2-133
    Coordinate data word II-2-133
    Data display II-2-130
    Display character codes II-2-132
    Display station output function code II-2-133
    Dot mode II-2-130
    Keyboard II-2-130
    Keyboard character codes II-2-131
    Program timing consideration II-2-134
    Programming example II-2-134
    Receive and display program flowchart II-2-135
Divide, FP II-2-53
Double-precision nonstandard FP numbers II-2-51
Double-precision register designators II-1-53

Edit examples
    Using edit masks 1 through 8 II-B-2
    Using edit masks 17 through 25 II-B-4
    Using edit masks 9 through 16 II-B-3
    Edit mask 26 II-B-5
    Edit masks 1 through 25 II-B-1
End cases, floating-point, see Floating-Point End Cases
Exceptions during stack operations II-2-40
Exchange interrupt II-2-25
Exchange jumps II-1-136
Exchange operations, CP II-2-1
Exchange packages II-2-3
    CYBER 170 State exchange package II-2-113
    Interstate exchange package II-2-108
    Virtual State exchange package II-2-5
Exponent arithmetic, FP II-2-51

Flags II-2-35
Floating-point arithmetic instructions II-1-54
Floating-point branch instructions II-1-58
Floating-point conversion instructions II-1-53
Floating-point (FP) end cases
    FP compare results II-2-55
    FP difference results, UM clear II-2-58

FP difference results, UM set II-2-59
FP product results, UM clear II-2-60
FP product results, UM set II-2-61
FP quotient results, UM clear II-2-62
FP quotient results, UM set II-2-63
FP sum results, UM clear II-2-56
FP sum results, UM set II-2-57
Floating-point (FP) programming
Exponent arithmetic II-2-51
FP data formats II-2-49
FP divide II-2-53
FP double-precision nonstandard numbers II-2-51
FP indefinite numbers II-2-51
FP infinite numbers II-2-50
FP multiply II-2-53
FP nonzero numbers II-2-50
FP standard and nonstandard numbers II-2-50
FP sum and difference II-2-52
FP zero numbers II-2-50
Normalization II-2-51
Representation II-2-50
Floating-point instruction descriptions II-1-52
Format
BDP data descriptor II-2-41
FP data II-2-49
Integer II-1-12
Of XO for call instructions II-2-37
Page table entry II-2-89
PP data II-1-90
PP instructions II-1-90
PP relocation register II-1-91
Process virtual address (PVA) II-2-78
Real memory address (RMA) II-2-81
Segment/page identifier (SPID) II-2-80
System virtual address (SVA) II-2-79
Vector instruction II-1-61

General instructions, CP II-1-4
Global privileged system instructions II-1-84
Glossary II-A-1

Immediate data instruction, BDP II-1-48
Indefinite numbers, FP II-2-51
Infinite numbers, FP II-2-50
Input/output instructions, PP II-1-124
Instruction description nomenclature, CP II-1-2
Instruction descriptions, BDP II-1-31
Instruction descriptions, floating-point II-1-52
Instruction descriptions, peripheral processor II-1-90
Instruction descriptions, system II-1-71
Instruction descriptions, vector II-1-60
Instruction format, vector II-1-61
Instruction formats, PP II-1-90
Instruction index
CP instructions - mnemonic sequence II-D-5
CP instructions - opcode sequence II-D-1
PP instructions - mnemonic sequence II-D-13
PP instructions - opcode sequence II-D-9
Instruction nomenclature, BDP II-1-32
Instructions, mixed mode II-1-86
Instructions, monitor mode II-1-85
Integer arithmetic instructions, CP II-1-11
Integer format II-1-12
Interface information
Active flag II-C-5
Control signals II-C-3
Data input sequences II-C-6
Data output sequences II-C-7
Data sequences timing II-C-11
Data signals II-C-5
External channel I/O sequences II-C-5
Full flag II-C-5
Function instructions II-C-5
Interfaces II-C-1
Maintenance channel interface II-C-1
Maintenance channel signals II-C-3
MCH input sequence II-C-8
MCH output sequence II-C-9
PP and channel interaction II-C-5
Signals II-C-2
Signals and cables II-C-3
Two-port multiplexer interface II-C-1
12-bit channel control signals II-C-2
12-bit external interface II-C-1
Interrupt conditions
Access violation (MCR 54) II-2-26
Address specification error (MCR 52) II-2-27
Arithmetic loss of significance (UCR 62) II-2-27
Arithmetic overflow (UCR 57) II-2-27
Critical frame flag (UCR 53) II-2-28
CYBER 170 State exchange request (MCR 53) II-2-32
Debug (UCR 56) II-2-28
Detected uncorrectable error (MCR 48) II-2-32
Divide fault (UCR 55) II-2-28
Environment specification error (MCR 55) II-2-28
Exponent overflow (UCR 58) II-2-29

Exponent underflow (UCR 59) II-2-29
External interrupt (MCR 56) II-2-29
FP indefinite (UCR 61) II-2-29
FP loss of significance (UCR 60) II-2-30
Free flag (UCR 50) II-2-30
Instruction specification error (MCR 51) II-2-30
Inter-ring pop (UCR 52) II-2-31
Invalid BDP data (UCR 63) II-2-31
Invalid segment/ring number zero (MCR 60) II-2-31
Multiple II-2-33
Not assigned (MCR 49) II-2-31
Outward call/inward return (MCR 61) II-2-31
Page table search without find (MCR 57) II-2-32
Privileged instruction fault (UCR 48) II-2-32
Process interval time (UCR 51) II-2-32
Short warning (MCR 50) II-2-32
Soft error log (MCR 62) II-2-33
System interval timer (MCR 59) II-2-33
Trap exception (MCR 63) II-2-33
Unimplemented instruction (UCR 49) II-2-33
Interrupt flowchart II-2-34
Interrupts, CP II-1-3; II-2-25
Interrupts, multiple II-2-33
Interstate exchange package II-2-108
Interstate programming
Address errors II-2-119
Cache invalidation in CYBER 170 State II-2-103
Call from Virtual State to CYBER 170 State II-2-105
Calls, returns, and interrupts II-2-106
Code modification in CYBER 170 State II-2-115
CYBER 170 State exchange package II-2-112
CYBER 170 State-to-Virtual State monitor mode exchange II-2-104
Debug/performance monitoring II-2-115
Exception handling in CYBER 170 State II-2-115
Exchange packages used in CYBER 170 State II-2-107
Exchanges within CYBER 170 State II-2-104
Extended memory transfer exceptions II-2-120
Hardware exceptions in CYBER 170 State II-2-121
Illegal instructions II-2-120
Interstate calls, returns, and interrupts II-2-106
Interstate exchange package II-2-107
Interstate stack frame save area (SFSA) II-2-113
Memory addressing in CYBER 170 State II-2-103
Operation in CYBER 170 State II-2-102
Return from Virtual State to CYBER 170 State II-2-105
Software exception conditions II-2-115
State-switching operations II-2-104
Trap interrupt from CYBER 170 State to Virtual State II-2-105
Virtual State monitor mode-to-CYBER 170 State exchange II-2-104
IOU dedicated channels II-2-136
IOU pass instructions II-1-135
IOU peripheral processor (PP) programming
Absolute and relocation addressing II-2-121
Cache invalidation II-2-127
Central memory addressing by PPs II-2-121
Channel active flag II-2-124
Channel flags II-2-124
Channel input/output (I/O) operations II-2-124
Channel marker flag II-2-125
Channel parity errors II-2-128
Direct 12-bit address II-2-123
Direct 18-bit operand II-2-123
Direct 6-bit address II-2-123
Direct 6-bit operand II-2-123
Error detection and recovery II-2-128
Error flag II-2-125
Indexed 12-bit address II-2-123
Indirect 6-bit address II-2-123
Initialization II-2-129
Inter-PP communications II-2-126
Operating system (OS) bounds test II-2-122
Parity errors on input data II-2-129
Parity errors on output data II-2-128
PP central memory read II-2-122
PP central memory write II-2-122
PP hardware errors II-2-128
PP memory addressing by PPs II-2-123
PP program timing consideration II-2-127
Programming for channel input/output II-2-125
Register-full flag II-2-124
Timeout II-2-129
IOU registers
Element identifier (EID) II-2-19
Environment control (EC) II-2-19
Fault status (FS) II-2-19
Fault status mask II-2-19
Operating system (OS) bounds II-2-19

Options installed (OI) II-2-19
Status summary (SS) II-2-20
Test mode (TM) II-2-20

Job-to-monitor exchange operations, Virtual State II-2-3

Keyboard character codes, display station II-2-131
Keyboard, display station II-2-130

Load and store instructions, CP II-1-5
Load and store instructions, PP II-1-93
Local privileged system instructions II-1-83
Logical instructions, CP II-1-27
Logical instructions, PP II-1-103

Maintenance channel programming
For halt/stop (opcode 0/1) II-2-146
For master clear/clear errors (opcode 6/7) II-2-146
For read IOU summary status (opcode C, IOU only) II-2-147
For read/write (opcode 4/5) II-2-146
MCH control words II-2-146
MCH function word bit assignments II-2-147
MCH functions words II-2-144
Maintenance registers
CM II-2-16
IOU II-2-18
Mark to Boolean instruction, CP II-1-30
Mixed mode instructions II-1-86
Monitor condition/mask register bit assignments II-2-21
Monitor mode instructions II-1-85
Monitor-to-job exchange operations, Virtual State II-2-3
Multiple interrupt conditions II-2-34
Multiply, FP II-2-53
Nonprivileged system instructions II-1-72
Nonstandard numbers, FP II-2-50
Nonzero numbers, FP II-2-50
Normalization, FP II-2-51
Numeric instructions, BDP II-1-32

Page table entry format II-2-89
Page table search, start RMA formation II-2-87
Page, virtual memory, see Virtual and Central Memory Programming
Pass instructions, IOU II-1-135
Peripheral processor instruction descriptions II-1-90
PP arithmetic instructions II-1-97
PP branch instructions II-1-113
PP central memory access instructions II-1-116
PP data format II-1-90
PP input/output instructions II-1-124
PP instruction formats II-1-90
PP instruction index
Mnemonic sequence II-D-13
Opcode sequence II-D-9
PP load and store instructions II-1-93
PP logical instructions II-1-103
PP relocation register format II-1-91
PP replace instructions II-1-108
Process state registers, CP II-2-6
Processor state registers, CP II-2-12
Process virtual address (PVA) format II-2-78
Program monitoring
Debug II-2-64
Debug condition select II-2-67
Debug conditions II-2-72
Debug index register II-2-65
Debug list II-2-64
Debug list entry II-2-66
Debug list pointer register II-2-65
Debug mask register II-2-66
Debug scan operation II-2-68
Debug-software interaction, debug disabled II-2-70
Debug-software interaction, debug enabled II-2-70
Enabling debug II-2-68
Interrupts during debug scan II-2-69
Programming information II-2-1

Real-time clock programming II-2-136
Real memory address (RMA) format II-2-81
Register bit string instructions, CP II-1-29
Register designators, double-precision II-1-53
Registers
CM, see CM Registers
CP, see CP Registers
IOU, see IOU Registers
Stack operations, see Stack Operations, Assigned Registers
Relocation register format, PP II-1-91
Replace instructions, PP II-1-108

Segment/page identifier (SPID)
format II-2-80
SFSA descriptor II-2-38
SFSA descriptor field II-2-37
Shift instructions, CP II-1-26
Stack frame save area (SFSA) format II-2-36
Stack frames and save areas II-2-36
Stack manipulating operations II-2-36
Stack operations, assigned registers
Argument pointer (A4) II-2-40
Binding section pointer (A3) II-2-40
Current stack frame pointer
(A1) II-2-39
Dynamic space pointer (A0) II-2-39
Previous save area pointer (A2) II-2-39
Top-of-stack pointers II-2-39
Stack operations, exceptions during II-2-40
Standard numbers, FP II-2-50
Subscript and immediate data instructions,
BDP II-1-48
Sum, FP II-2-52
System instruction descriptions II-1-71
System instruction privilege and
mode II-2-99
System instructions, global
privileged II-1-84
System instructions, local
privileged II-1-83
System instructions, nonprivileged II-1-72
System publication index 7/8
System virtual address (SVA) II-2-79

Trap interrupt II-2-25
Two-port multiplexer programming
Calendar clock/auto dial-out
(1XXX) II-2-140
Data input II-2-144
Data output II-2-143
Function words II-2-139
Master clear (07XX) II-2-143
PP read terminal data (01XX) II-2-141
PP write output buffer (02XX) II-2-142
Programming considerations II-2-143
Read status summary (00XX) II-2-141
Set/clear data terminal ready
(DTR) (04XX) II-2-143
Set/clear request to send (RTS)
(05XX) II-2-143
Set operation mode to terminal
(03XX) II-2-142
Terminal deselect (6XXX) II-2-139
Terminal select (7XXX) II-2-139

Undefined results, BDP II-2-45
User condition/mask register bit
assignments II-2-22
User mask/condition and monitor condition
fields II-2-39

Vector instruction descriptions II-1-60
Vector instruction format II-1-61
Vector operations II-2-47
Vector programming
Vector broadcast II-2-48
Vector interrupts II-2-48
Vector length (number of
operations) II-2-47
Vector overlap II-2-48
Vector page size II-2-48
Virtual and central memory programming
Access protection II-2-91
Address tables II-2-83
BN-to-page number/page offset
conversion II-2-82
Call indirect access
requirements II-2-101
CM addressing from CP II-2-77
Code base pointer format II-2-91
Effect of RN violations II-2-98
Effect of RN = 0 II-2-96
Execute access privilege/mode II-2-98
Keys/locks II-2-99
Listing of pages in page table II-2-90
Page table entries (PTE) II-2-87
Page table entry format II-2-89
Page table search II-2-86
Page table search, start RMA
formation II-2-87
Process binding section II-2-90
Process virtual address (PVA)
format II-2-78
Process virtual memory II-2-78
PTE control fields II-2-88
PTE page frame RMA field II-2-89
PTE segment/page identifier
field II-2-88
PVA-to-RMA conversion II-2-83
PVA-to-SVA conversion, execute II-2-94
PVA-to-SVA conversion, read/
write II-2-93
Real memory II-2-79
Real memory address (RMA)
format II-2-81
Ring structure II-2-95
Ring voting II-2-95
RN effect on pop instruction II-2-97

RN for execute access II-2-96
RN for read/write access II-2-96
Segment descriptor table entry format II-2-85
Segment descriptor table (SDT) II-2-84
Segment/page identifier (SPID) format II-2-80
System instruction privilege and mode II-2-99
System page table (SPT) II-2-86
System virtual address (SVA) format II-2-79
System virtual memory II-2-78
Virtual machine identifier (VMID) field II-2-38
Virtual State exchange package II-2-5
Virtual State job-to-monitor exchange operations II-2-3
Virtual State monitor-to-job exchange operations II-2-3

COMMENT SHEET

MANUAL TITLE: CDC CYBER 170 Computer Systems Models 815, 825, 835, 845, and 855; CDC CYBER 180 Computer Systems Models 810, 830, 835, 840, 845, 850, 855, 860, and 990 Virtual State Volume II Hardware Reference Manual

PUBLICATION NO.: 60458890 **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 O, MINNEAPOLIS, MINN 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA