Burroughs
B 7700
INFORMATION PROCESSING SYSTEMS

REFERENCE MANUAL

COPYRIGHT © 1973

AA 430046
Burroughs Corporation
Detroit, Michigan 48232

$7.00

Printed in U. S. America 1-73 1060233

Burroughs Corporation believes the information contained in this manual to be accurate and reliable, and much care has been taken in its preparation. However, the Corporation cannot accept any responsibility, financial or otherwise, for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Burroughs Corporation, Technical Information Organization, 200 W. Lancaster Avenue, Wayne, Pennsylvania 19087.

TABLE OF CONTENTS

CHAPTER I DESCRIPTION OF B 7700 SYSTEM 1-1
THE B 7700 SYSTEM 1-1
DISTINGUISHING FEATURES 1-2
SYSTEM CONFIGURATION 1-5
Maximum Configuration 1-8
Minimum Configuration 1-10
Typical Configuration With Full Fail-Soft Capabilities 1-10
CHAPTER II SYSTEM ARCHITECTURE 2-1
SECTION 1 DATA REPRESENTATION 2-1
GENERAL 2-1
INTERNAL CHARACTER CODES AND COLLATING SEQUENCES 2-1
NUMBERS AND NUMBERING SYSTEMS 2-1
Binary Notation 2-1
Hexadecimal and Octal Notation 2-3
NUMBER CONVERSION 2-3
Binary to Decimal Conversion 2-3
INTEGRAL 2-3
FRACTIONAL 2-3
Decimal to Binary Conversion 2-5
INTEGRAL 2-5
FRACTIONAL 2-6
Decimal to Octal Conversion 2-6
INTEGRAL 2-6
FRACTIONAL 2-6
Octal to Decimal Conversion 2-6
OCTADE 2-6
INTEGRAL 2-7
FRACTIONAL 2-9
Decimal to Hexadecimal Conversion 2-9
Hexadecimal to Decimal Conversion 2-9
OPERAND FORMATS 2-9
Numeric Operands 2-9
SINGLE PRECISION OPERANDS 2-9
DOUBLE PRECISION OPERANDS 2-10
NUMBER RANGES AND NORMALIZATION 2-10
Logical Operands 2-13
String Operands 2-15
SECTION 2 POLISH NOTATION AND STACK 2-18
GENERAL 2-18
POLISH NOTATION 2-18
General Rules for Generation of Polish String 2-18
Evaluating Polish String 2-21
Compilation Using Polish Notation 2-22
Program Code String 2-22
PROGRAM WORD 2-22
STACK CONCEPTS 2-22

TABLE OF CONTENTS (Cont)

General 2-24
BASE AND LIMIT OF STACK 2-24
BI-DIRECTIONAL DATA FLOW IN THE STACK 2-24
DOUBLE PRECISION STACK OPERATION 2-25
Hardware Implementation 2-25
DYNAMIC PROGRAM HISTORY 2-25
Addressing History 2-27
DIRECT ADDRESSING 2-27
RELATIVE-ADDRESSING 2-27
Stack History 2-29
SIMPLE STACK OPERATION 2-31
INTERRUPT HANDLING 2-32
MULTIPLE STACKS AND RE-ENTRANT CODE 2-35
Level Definition 2-35
Re-Entrance 2-35
Job-Splitting 2-35
Stack Descriptor 2-35
Stack Vector Descriptor 2-36
Presence Bit Interrupt 2-36
SECTION 3 PROCESSOR WORD FORMATS 2-36
GENERAL 2-36
WORDS FOR ADDRESSING OUTSIDE OF THE STACK 2-38
Presence Bit 2-38
Index Bit 2-39
Invalid Index 2-39
Valid Index 2-39
Read-Only Bit 2-39
Copy Bit 2-39
Data Descriptor 2-39
String Descriptor 2-40
Segment Descriptors 2-40
WORDS FOR ADDRESSING WITHIN STACKS 2-40
Program Control Word 2-40
Indirect Reference Word 2-40
Stuffed Indirect Reference Word 2-40
WORDS FOR STORING STACK HISTORY 2-46
Mark Stack Control Word 2-46
Return Control Word 2-46
Top of Stack Control Word 2-48
WORDS USED AS SPECIAL PARAMETERS 2-48
Step Index Word 2-48
Occurs Index Word 2-48
Time of Day Function Word 2-48
SECTION 4 INPUT/OUTPUT SUBSYSTEM MAP STRUCTURE 2-53
INTRODUCTION 2-53
QUEUE-DRIVEN I/O 2-53

TABLE OF CONTENTS (Cont)

ERROR HANDLING 2-54
DEFERMENT OF PATH BINDING 2-55
I/O SUBSYSTEM MAP 2-56
Commands and Requests 2-57
Map Integrity 2-57
Home Address Words 2-58
Unit Table 2-58
I/O Queue Head and Tail Words 2-58
Status Queue Headers 2-58
Input/Output Control Block 2-58
CHAPTER III CENTRAL PROCESSOR MODULE 3-1
SECTION I FUNCTIONAL DESCRIPTION OF THE CENTRAL PROCESSOR MODULE 3-1
OPERATIONAL CONCEPTS OF THE CENTRAL PROCESSOR MODULE. 3-3
Use Of Division Overlap And Local Buffering. 3-3
Memory Overlap 3-5
PROGRAM BUFFER 3-6
Buffer Word Format 3-6
Reading From The Program Buffer 3-6
Writing Into The Program Buffer 3-9
Branching 3-10
Edit Mode Operation 3-10
Vector Mode Operation 3-11
PROGRAM BARREL 3-11
Selection Gating Stage 3-11
Barrel Shift Stage 3-11
Syllable Parity Checking 3-12
Processing Of Lt48 Operator 3-13
PROGRAM CONTROL UNIT 3-13
Instruction Decode Register 3-13
REGISTERS ASSOCIATED WITH THE IDR 3-13
IDR DECODES 3-15
Instruction Execute Register 3-16
Preprocessing Of Value Call And Name Call Operators 3-17
ADDRESS UNIT 3-19
EXECUTION UNIT 3-23
General 3-23
Input Queues 3-23
OPERATOR QUEUE 3-23
THE EU DATA QUEUE 3-24
EU LOOK-AHEAD DATA QUEUE 3-25
THE K AND L QUEUES 3-25
THE PIR AND PSR QUEUE 3-26
THE LEXIC LEVEL QUEUE 3-26
Major EU Data Circuits 3-26
ADDER 3-27
SHIFT MECHANISM 3-27

TABLE OF CONTENTS (Cont)

LOCAL MEMORY 3-29
DATA REGISTERS 3-30
DATA TRANSFER BUSES 3-30
Major EU Control Circuits 3-31
Basic EU Operation 3-33
EU Error Detection Methods 3-35
EU Interrupt Processing 3-35
STACK BUFFER 3-36
Stack Buffer Function 3-37
Stack Buffer Operation 3-37
FETCHING THE TOP ITEM FROM THE STACK BUFFER 3-38
ADDING A NEW ITEM TO THE CONTENTS OF THE STACK BUFFER 3-38
FETCHING AND STORING OF LOCAL INFORMATION 3-41
STACK BUFFER PURGE OPERATION 3-42
ASSOCIATIVE MEMORY 3-42
Write Operation 3-42
Read Operation 3-44
STORAGE UNIT 3-46
Storage Input Register 3-46
Storage Unit Operations Queue 3-47
Storage Output Register 3-47
Storage Unit Data Queue 3-47
Memory Address Register 3-47
Storage Unit Control Logic 3-49
COMMUNICATIONS UNIT 3-49
FAULT CONTROL LOGIC 3-52
General Discussion Of Interrupts 3-54
General Discussion Of Fail Register 3-54
SECTION 2 INTERRUPTS 3-55
INTRODUCTION 3-55
HARDWARE INTERRUPT SYSTEM 3-55
CPM STATES AND MODES 3-55
Control State 3-56
Normal State 3-56
FAULT CONTROL LOGIC 3-57
Fault Condition Register 3-57
Fault Mask Register 3-57
Interrupt Identification 3-59
CPM Fail Register 3-60
Control Mode Register 3-61
MCM Fail Register 3-61
IOM Fail Register 3-64
INTERRUPT PROCESSING 3-64
Interrupt Processing In Normal Mode 3-64
Interrupt Processing In CM1 3-66
Interrupt Processing In CM2 3-66

TABLE OF CONTENTS (Cont)

Interrupt Processing In CM3 3-68
Control Mode Advancement 3-68
ALARM INTERRUPTS (FIRST PRIORITY) 3-73
SYLLABLE DEPENDENT INTERRUPTS (SECOND PRIORITY) 3-73
SPECIAL INTERRUPTS (THIRD PRIORITY) 3-73
EXTERNAL INTERRUPTS (FOURTH PRIORITY) 3-73
Memory Related Interrupts 3-74
IRRECOVERABLE MEMORY RELATED ERRORS 3-75
MEMORY FAIL 2 3-76
Interrupt Descriptions 3-77
ALARM INTERRUPTS 3-77
SYLLABLE DEPENDENT INTERRUPTS 3-81
SPECIAL INTERRUPTS 3-88
SECTION 3 OPERATORS 3-91
INTRODUCTION 3-91
GROUPING OF OPERATORS 3-94
PRIMARY MODE OPERATORS 3-105
Arithmetic Operators 3-105
ADD (ADD) (P)80 3-105
SUBTRACT (SUBT) (P)81 3-106
MULTIPLY (MULT) (P)82 3-106
EXTENDED MULTIPLY (MULX) (P)8F 3-106
DIVIDE (DIVD) (P)83 3-106
INTEGER DIVIDE (IDIV) (P)84 3-106
REMAINDER DIVIDE (RDIV) (P)85 3-106
INTEGERIZE, TRUNCATED (NTIA) (P)86 3-106
INTEGERIZE, ROUNDED (NTGR) (P)87 3-106
INTEGERIZE ROUNDED, DOUBLE PRECISION (NTGD) (V)87 3-107
Bit Operators 3-107
BIT SET (BSET) (P)96 3-107
DYNAMIC BIT SET (DBST) (P)97 3-107
BIT RESET (BRST) (P)9E 3-107
DYNAMIC BIT RESET (DBRS) (P)9F 3-107
CHANGE SIGN BIT (CHSN) (P)8E 3-107
COUNT BINARY ONE'S (CBON) (V)BB 3-107
LEADING ONE TEST (LOG2) (V)8B 3-107
Branch Operators 3-108
BRANCH UNCONDITIONAL (BRUN) (P)A2 3-108
BRANCH ON TRUE (BRTR) (P)A1 3-108
BRANCH ON FALSE (BRFL) (P)A0 3-108
DYNAMIC BRANCH UNCONDITIONAL (DBUN) (P)AA 3-108
DYNAMIC BRANCH TRUE (DBTR) (P)A9 3-108
DYNAMIC BRANCH FALSE (DBFL) (P)A8 3-109
STEP AND BRANCH (STBR) (P)A4 3-109
Compare Operators 3-109
COMPARE CHARACTERS GREATER, DESTRUCTIVE (CGTD) (P)F2 3-110

TABLE OF CONTENTS (Cont)

COMPARE CHARACTERS GREATER, UPDATE (CGTU) (P)FA 3-110
COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE (CGED) (P)F1 3-110
COMPARE CHARACTERS GREATER OR EQUAL, UPDATE (CGEU) (P)F9 3-110
COMPARE CHARACTERS EQUAL, DESTRUCTIVE (CEQD) (P)F4 3-110
COMPARE CHARACTERS EQUAL, UPDATE (CEQU) (P)FC 3-110
COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE (CLED) (P)F3 3-111
COMPARE CHARACTERS LESS OR EQUAL, UPDATE (CLEU) (P)FB 3-111
COMPARE CHARACTERS LESS, DESTRUCTIVE (CLSD) (P)F0 3-111
COMPARE CHARACTERS LESS, UPDATE (CLSU) (P)F8 3-111
COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE (CNED) (P)F5 3-111
COMPARE CHARACTERS NOT EQUAL, UPDATE (CNEU) (P)FD 3-111
Enter Edit Mode Operators 3-111
TABLE ENTER EDIT, DESTRUCTIVE (TEED) (P)D0 3-111
TABLE ENTER EDIT, UPDATE (TEEU) (P)D8 3-112
EXECUTE SINGLE MICRO, DESTRUCTIVE (EXSD) (P)D2 3-112
EXECUTE SINGLE MICRO, UPDATE (EXSU) (P)DA 3-112
EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE (EXPU) (P)DD 3-112
Enter Vector Mode Operators 3-112
SINGLE-WORD VECTOR MODE (VMOS) (P)E7 3-117
MULTIPLE-WORD VECTOR MODE (VMOM) (P)EF 3-117
Index And Load Operators 3-117
INDEX (INDX) 3-117
INDEX AND LOAD NAME (NXLN) (P)A5 3-118
INDEX AND LOAD VALUE (NXLV) (P)AD 3-118
LOAD (LOAD) (P)BD 3-118
LOAD TRANSPARENT (LODT) (V)BC 3-119
Input Convert Operators 3-119
INPUT CONVERT, DESTRUCTIVE (ICVD) (P)CA 3-119
INPUT, CONVERT, UPDATE (ICVU) (P)CB 3-119
Literal Call Operators 3-119
LIT CALL ZERO (ZERO) (P)B0 3-119
LIT CALL ONE (ONE) (P)B1 3-120
LIT CALL 8 BITS (LT8) (P)B2 3-120
LIT CALL 16 BITS (LT 16) (P)B3 3-120
LIT CALL 48 BITS (LT48) (P)BE 3-120
MAKE PROGRAM CONTROL WORD (MPCW) (P)BF 3-120
Logical Operators 3-120
LOGICAL AND (LAND) (P)90 3-120
LOGICAL OR (LOR) 3-120
LOGICAL NEGATE (LNOT) (P)92 3-120
LOGICAL EQUIVALENCE (LEQV) (P)93 3-120
Pack Operators 3-121
PACK, DESTRUCTIVE (PACD) (P)D1 3-121
PACK, UPDATE (PACU) (P)D9 3-122
Relational Operators 3-122
GREATER THAN (GRTR) (P)8A 3-122

TABLE OF CONTENTS (Cont)

GREATER THAN OR EQUAL (GREQ) (P)89 3-122
EQUAL (EQUL) (P)8C 3-122
LESS THAN OR EQUAL (LSEQ) (P)8B 3-122
LESS THAN (LESS) (P)88 3-123
NOT EQUAL (NEQL) (P)8D 3-123
LOGICAL EQUAL (SAME) (P)94 3-123
Scale Operators 3-123
SCALE LEFT (SCLF) (P)C0 3-123
DYNAMIC SCALE LEFT (DSLF) (P)C1 3-123
SCALE RIGHT SAVE (SCRS) (P)C4 3-124
DYNAMIC SCALE RIGHT SAVE (DSRS) (P)C5 3-124
SCALE RIGHT TRUNCATE (SCRT) (P)C2 3-124
DYNAMIC SCALE RIGHT TRUNCATE (DSRT) (P)C3 3-124
SCALE RIGHT ROUNDED (SCRR) (P)C8 3-124
DYNAMIC SCALE RIGHT ROUNDED (DSRR) (P)C9 3-124
SCALE RIGHT FINAL (SCRF) (P)C6 3-124
DYNAMIC SCALE RIGHT FINAL (DSRF) (P)C7 3-124
Stack Operators 3-125
EXCHANGE (EXCH) (P)B6 3-125
ROTATE STACK DOWN (RSDN) (V)B7 3-125
ROTATE STACK UP (RSUP) (V)B6 3-125
DUPLICATE TOP-OF-STACK (DUPL) (P)B7 3-125
DELETE TOP-OF-STACK (DLET) (P)B5 3-125
PUSH DOWN STACK REGISTERS (PUSH) (P)B4 3-125
STORE DESTRUCTIVE (STOD) (P)B8 3-125
STORE NON-DESCTRUCTIVE (STON) (P)B9 3-125
OVERWRITE DESTRUCTIVE (OVRD) (P)BA 3-125
OVERWRITE NON-DESTRUCTIVE (OVRN) (P)BB 3-125
READ WITH LOCK (RDLK) (V)BA 3-125
String Operators 3-125
STRING ISOLATE (SISO) (P)D5 3-128
String Transfer Operators 3-128
TRANSFER WORDS, DESTRUCTIVE (TWSD) (P)D3 3-128
TRANSFER WORDS, UPDATE (TWSU) (P)DB 3-129
TRANSFER WORDS, OVERWRITE DESTRUCTIVE (TWOD) (P)D4 3-129
TRANSFER WORDS, OVERWRITE UPDATE (TWOU) (P)DC 3-129
TRANSFER WHILE GREATER, DESTRUCTIVE (TGTD) (P)E2 3-129
TRANSFER WHILE GREATER, UPDATE (TGTU) (P)EA 3-130
TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE (TGED) (P)E1 3-130
TRANSFER WHILE GREATER OR EQUAL, UPDATE (TGEU) (P)E9 3-130
TRANSFER WHILE EQUAL, DESTRUCTIVE (TEQD) (P)E4" 3-130
TRANSFER WHILE EQUAL, UPDATE (TEQU) (P)EC 3-130
TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE (TLED) (P)E3 3-130
TRANSFER WHILE LESS OR EQUAL, UPDATE (TLEU) (P)EB 3-130
TRANSFER WHILE LESS, DESTRUCTIVE (TLSD) (P)E0 3-131
TRANSFER WHILE LESS, UPDATE (TLSU) (P)E8 3-131
TRANSFER WHILE NOT EQUAL, DESTRUCTIVE (TNED) (P)E5 3-131

TABLE OF CONTENTS (Cont)

TRANSFER WHILE NOT EQUAL, UPDATE (TNEU) (P)ED 3-131
TRANSFER WHILE TRUE, DESTRUCTIVE (TWTD) (V)D3 3-131
TRANSFER WHILE TRUE, UPDATE (TWTU) (V)DB 3-132
TRANSFER WHILE FALSE, DESTRUCTIVE (TWFD) (V)D2 3-132
TRANSFER WHILE FALSE, UPDATE (TWFU) (V)DA 3-132
TRANSFER UNCONDITIONAL, DESTRUCTIVE (TUND) (P)E6 3-132
TRANSFER UNCONDITIONAL, UPDATE (TUNU) (P)EE 3-132
Subroutine Operators 3-132
MARK STACK (MKST) (P)AE 3-132
INSERT MARK STACK (IMKS) (P)CF 3-133
NAME CALL (NAMC) (P)40 THRU (P)7F 3-133
VALUE CALL (VALC) (P)00 THRU (P)3F 3-133
EVALUATE DESCRIPTOR (EVAL) (P)AC 3-136
ENTER (ENTR) (P)AB 3-136
EXIT (EXIT (P)A3 3-136
RETURN (RETN) (P)A7 3-139
Transfer Operators 3-139
FIELD TRANSFER (FLTR) (P)98 3-139
DYNAMIC FIELD TRANSFER (DFTR) (P)99 3-139
FIELD ISOLATE (ISOL) (P)9A 3-139
DYNAMIC FIELD ISOLATE (DISO) (P)9B 3-142
FIELD INSERT (INSR) (P)9C 3-142
DYNAMIC FIELD INSERT (DINS) (P)9D 3-142
Type-Transfer Operators 3-142
SET TO SINGLE-PRECISION, TRUNCATED (SNGT) (P)CC 3-142
SET TO SINGLE-PRECISION, ROUNDED (SNGL) (P)CD 3-142
SET TO DOUBLE-PRECISION (XTND) (P)CE 3-142
SET TWO SINGLES TO A DOUBLE (JOIN) (V)42 3-142
Miscellaneous Primary Mode Operators 3-142
ESCAPE TO 16-BIT INSTRUCTION (VARI) (P)95 3-143
READ AND CLEAR OVERFLOW FLIP-FLOP (ROFF) (P)D7 3-143
READ TRUE FALSE FLIP-FLOP (RTFF) (P)DE 3-143
SET EXTERNAL SIGN (SXSN) (P)D6 3-143
STUFF ENVIRONMENT (STFF) (P)AF 3-143
Universal Operators 3-143
CONDITIONAL HALT (HALT) (U)DF 3-143
INVALID OPERATOR (NVLD) (U)FF 3-143
NO OPERATION (NOOP) (U)FE 3-143
VARIANT MODE OPERATORS 3-148
Scan Operators 3-148
SCAN IN (SCNI) (V)4A 3-148
Scan While Operators 3-148
SCAN WHILE GREATER, DESTRUCTIVE (SGTD) (V)F2 3-148
SCAN WHILE GREATER, UPDATE (SGTU) (V)FA 3-148
SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE (SGED) (V)FL 3-149
SCAN WHILE GREATER OR EQUAL, UPDATE (SGEU) (V)F9 3-149
SCAN WHILE EQUAL, DESTRUCTIVE (SEQD) (V)F4 3-149

TABLE OF CONTENTS (Cont)

SCAN WHILE EQUAL, UPDATE (SEQU) (V)FC 3-149
SCAN WHILE LESS OR EQUAL, DESTRUCTIVE (SLED) (V)F3 3-149
SCAN WHILE LESS OR EQUAL, UPDATE (SLEU) (V)FB 3-149
SCAN WHILE LESS, DESTRUCTIVE (SLSD) (V)F0 3-149
SCAN WHILE LESS, UPDATE (SLSU) (V)F8 3-149
SCAN WHILE NOT EQUAL, DESTRUCTIVE (SNED) (V)F5 3-149
SCAN WHILE NOT EQUAL, UPDATE (SNEU) (V)FD 3-149
SCAN WHILE TRUE, DESTRUCTIVE (SWTD) (V)D5 3-150
SCAN WHILE TRUE, UPDATE (SWTU) (V)DD 3-150
SCAN WHILE FALSE, DESTRUCTIVE (SWFD) (V)D4 3-150
SCAN WHILE FALSE, UPDATE (SWFU) (V)DC 3-150
Tag Field Operators 3-150
SET TAG FIELD (STAG) (V)B4 3-150
READ TAG FIELD (RTAG) (V)B5 3-150
SET INTERVAL TIMER (SINT) (V)45 (CONTROL STATE OPERATOR) 3-151
READ PROCESSOR IDENTIFICATION (WHOI) (V)4E 3-151
ENABLE EXTERNAL INTERRUPTS (EEXI) (V)46 3-151
DISABLE EXTERNAL INTERRUPTS (DEXI) (V)47 3-151
IDLE UNTIL INTERRUPT (IDLE) (V)44 3-151
READ PROCESSOR REGISTER (RPRR) (V)B8 3-151
SET PROCESSOR REGISTER (SPRR) (V)B9 3-152
Unpack Operators 3-154
UNPACK ABSOLUTE, DESTRUCTUVE (UABD) (V)D1 3-154
UNPACK ABSOLUTE, UPDATE (UABU) (V)D9 3-154
UNPACK SIGNED, DESTRUCTIVE (USND) (V)D0 3-154
UNPACK SIGNED, UPDATE (USNU) (V)D8 3-155
LINKED LIST LOOKUP (LLLU) (V)BD 3-155
MASKED SEARCH FOR EQUAL (SRCH) (V)BE 3-155
MOVE TO STACK (MVST) (V)AF 3-156
OCCURS INDEX (OCRX) (V)85 3-157
TRANSLATE (TRNS) (V)D7 3-157
Operators Exclusive To The B 7700 3-158
SET MEMORY INHIBITS (SINH) (V)A8 (CONTROL STATE OP) 3-158
SET MEMORY LIMITS (SLMT) (V)AA (CONTROL STATE OP) 3-158
FETCH MEMORY FAIL REGISTER (FMFR) (V)AC (CONTROL STATE OP) 3-158
IGNORE PARITY (IGPR) (V)48 (CONTROL STATE OP) 3-158
PAUSE UNTIL INTERRUPT (PAUS) (V)84 3-158
INTERRUPT CHANNEL N (INCN) (V)8F 3-159
STOP (STOP) (V)BF 3-159
EDIT MODE OPERATORS 3-161
Insert Operators 3-161
INSERT UNCONDITIONAL (INSU) (E)DC 3-161
INSERT CONDITIONAL (INSC) (E)DD 3-161
INSERT DISPLAY SIGN (INSG) (E)D9 3-161
INSERT OVERPUNCH (INOP) (E)D8 3-161
Move Operators 3-162
MOVE WITH INSERT (MINS) (E)D0 3-162

TABLE OF CONTENTS (Cont)

MOVE WITH FLOAT (MFLT) (E)D1 3-162
MOVE CHARACTERS (MCHR) (E)D7 3-162
MOVE NUMERIC UNCONDITIONAL (MVNU) (E)D6 3-163
Skip Operators 3-163
SKIP FORWARD SOURCE CHARACTERS (SFSC) (E)D2 3-163
SKIP REVERSE SOURCE CHARACTERS (SRSC) (E)D3 3-163
SKIP FORWARD DESTINATION CHARACTERS (SFDC) (E)DA 3-163
SKIP REVERSE DESTINATION CHARACTERS (SRDC) (E)DB 3-163
RESET FLOAT (RSTF) (E)D4 3-163
END FLOAT (ENDF) (E)D5 3-163
END EDIT (ENDE) (E)DE 3-163
Universal Operators 3-163
NO OPERATION (NOOP) (U)FE 3-163
CONDITIONAL HALT (HALT) (U)DF 3-163
INVALID OPERATOR (NVLD) (U)FF 3-163
VECTOR MODE OPERATORS 3-166
Vector Branch (VEBR) (Z)EE 3-167
Vector Exit (VXIT) (Z)E6 3-167
Vector Stack Operators 3-167
LOAD A (LDA) (Z)E0 3-167
LOAD B (LDB) (Z)E2 3-167
LOAD C (LDC) (Z)E4 3-167
LOAD A - INCREMENT (LDAI) (Z)E1 3-168
LOAD B - INCREMENT (LDBI) (Z)E3 3-168
LOAD C - INCREMENT (LDCI) (Z)E5 3-168
STORE A (STA) (Z)E8 3-168
STORE B (STB) (Z)EA 3-168
STORE C (STC) (Z)EC 3-168
STORE A - INCREMENT (STAI) (Z)E9 3-168
STORE B - INCREMENT (STBI) (Z)EB 3-168
STORE C - INCREMENT (STCI) (Z)ED 3-168
DOUBLE LOAD A (DLA) (Z)F0 3-168
DOUBLE LOAD B (DLB) (Z)F2 3-168
DOUBLE LOAD C (DLC) (Z)F4 3-168
DOUBLE LOAD A - INCREMENT (DLAI) (Z)F1 3-169
DOUBLE LOAD B - INCREMENT (DLBI) (Z)F3 3-169
DOUBLE LOAD C - INCREMENT (DLCI) (Z)F5 3-169
DOUBLE STORE A (DSA) (Z)F8 3-169
DOUBLE STORE B (DSB) (Z)FA 3-169
DOUBLE STORE C (DSC) (Z)FC 3-169
DOUBLE STORE A - INCREMENT (DSAI) (Z)F9 3-169
DOUBLE STORE B - INCREMENT (DSBI) (Z)FB 3-169
DOUBLE STORE C - INCREMENT (DSCI) (Z)FD 3-169
Vector Fetch And Store Operators 3-169
VECTOR FETCH (FTCH) (Z)00 THRU (Z)3F 3-170
VECTOR STORE (STOR) (Z)40 THRU (Z)7F 3-170

TABLE OF CONTENTS (Cont)

CHAPTER IV INPUT/OUTPUT SUBSYSTEM 4-1
SECTION 1 GENERAL DESCRIPTION AND OPERATION OF INPUT/OUTPUT MODULE.. 4-1
PRELIMINARY 4-1
BASIC IOM CONFIGURATION 4-1
Control Word Flow 4-1
Data Flow 4-2
IOM/Peripheral Interface Configuration 4-2
PERIPHERAL CONTROL INTERFACE (PCI) 4-2
DISK FILE INTERFACE (DFI) 4-6
SCAN INTERFACE (SCI) 4-7
DATA COMMUNICATIONS PROCESSOR INTERFACE (DCI) 4-7
IOM/Main Memory and IOM/CPM Interface Configurations 4-7
IOM/MCM INTERFACE 4-8
IOM/CPM INTERFACE 4-8
IOM OPERATIONAL CHARACTERISTICS 4-8
IOM Job Map 4-10
HOME ADDRESS WORD 4-11
UNIT TABLE WORD 4-11
IOQ HEAD (IOQH) AND IOQ TAIL (IOQT) TABLES AND WORDS 4-13
I/O CONTROL BLOCKS 4-14
FAIL I/O CONTROL BLOCKS 4-15
STATUS QUEUE 4-15
IOM HOME (HA) COMMANDS 4-16
Start I/O (Home Code 0001) 4-17
Set Channel Busy/Set Channel Reserved (Home Code 0010) 4-17
Reset Channel Busy/Reset Channel Reserved (Home Code 0011) 4-18
Load Address Commands 4-18
DFO/DCP Scan-Out Commands (Home Code 1000) 4-18
DFO SCAN-OUT COMMANDS 4-18
DCP Scan-Out Commands 4-19
DFO Scan-In Commands (Home Code 1001) 4-20
Synchronous I/O Command (Home Code 1010) 4-24
Interrogate Peripheral Status Command (Home Code 1011) 4-24
Inhibit IOM Command (Home Code 1100) 4-25
Activate IOM Command (Home Code 1101) 4-25
Load DFO Flags Command (Home Code 1110) 4-26
AUTOMATIC SERVICE OF DISK JOBS FOR UNITS UNDER DFO CONTROL 4-26
AUTOMATIC DISK-PACK OPERATION 4-28
CODE TRANSLATIONS 4-28
EBCDIC-BCL Exceptions 4-32
IOM-GENERATED INTERRUPTS 4-33
SECTION 2 FUNCTIONAL OPERATION OF INPUT/OUTPUT MODULE SUBSECTIONS .. 4-33
GENERAL 4-33
FUNCTIONAL OPERATION OF TRANSLATOR 4-33
Job Service Initiation 4-34

TABLE OF CONTENTS (Cont)

Job-Service Termination 4-37
Exchange Ring Walk 4-37
Automatic Service of Disk Jobs Via DFO Units 4-37
Disk-Pack Control 4-38
Fail Mode of Operation 4-39
FUNCTIONAL OPERATION OF MIU 4-39
Functional Components of the MIU 4-42
FUNCTIONAL OPERATION OF PCI 4-43
Channel Initiation Operation 4-43
Channel Service Operation 4-45
Memory Operation 4-45
Channel Termination Operation 4-46
Functional Components of the PCI 4-46
FUNCTIONAL OPERATION OF DFI 4-47
Channel Initiation Operation 4-47
Channel Service Operation 4-49
Channel Termination Operation 4-49
Functional Components of the DFI 4-49
FUNCTIONAL OPERATION OF SCI 4-50
Scan Interface 4-52
DCP Scan Interface 4-52
SECTION 3 PERIPHERALS AND CONTROL WORD FORMATS 4-54
IOCW 4-54
RESULT DESCRIPTOR (NORMAL) 4-56
HARD LOAD RESULT DESCRIPTOR 4-60
SYNCHRONOUS I/O RESULT DESCRIPTOR 4-62
FAIL RESULT DESCRIPTOR 4-64
CARD PUNCH 4-67
Card Punch Control 4-67
CDL Word Format 4-67
IOCW Information 4-67
Result Descriptor Format 4-68
Operations 4-69
CARD PUNCH BCL (OP 23) 4-69
CARD PUNCH BINARY (OP 24) 4-69
CARD PUNCH EBCDIC (OP 25) 4-70
TEST (OP 99) 4-70
PUNCH CHECK ERROR 4-70
CARD READER CONTROL 4-71
CDL Word Format 4-71
IOCW Information 4-72
Result Descriptor Format 4-72
Operations 4-74
READ ALPHA (OP 20) 4-74
READ BINARY (OP 21) 4-74
READ EBCDIC (OP 22) 4-74
TEST (OP 99) 4-74

TABLE OF CONTENTS (Cont)

DISK FILE SUBSYSTEM 4-75
Disk File Exchanges 4-75
Exchange Modularity 4-75
CDL Word Format 4-76
IOCW Information 4-77
Result Descriptor Format 4-78
Operations 4-79
READ (OP 51) 4-79
CHECK (OP 52) 4-79
WRITE (OP 50) 4-79
Error Termination 4-80
DISK-PACK DRIVE MEMORY SYSTEM 4-81
CDL Word - General Format 4-81
CDL Word Format - Write 4-83
CDL Word Format - Read 4-84
CDL Word Format - Initialize 4-85
CDL Word Format - Verify 4-86
CDL Word Format - Relocate 4-88
CDL Word Format - Test and On-line Test 4-89
IOCW Information 4-89
Result Descriptor Format 4-90
Operators 4-92
WRITE (OP CODE 50) 4-92
READ (OP CODE 51) 4-92
INITIALIZE (OP CODE 56) 4-92
VERIFY (OP CODE 57) 4-92
RELOCATE (OP CODE 58) 4-93
TEST (OP CODE 99) 4-93
Exception Conditions 4-93
SINGLE BIT ERROR CORRECTION (SBE) 4-93
DISK-PACK DRIVE BUSY (TIME-OUT) (DRB) 4-93
SPEED ERROR (1081) 4-93
ADDRESS POSITION ERROR (VERIFY) (1101) 4-93
TRANSMISSION PARITY ERROR (1181) 4-94
SEEK TIME-OUT (1801) 4-94
SEEK ERROR (DSK) 4-94
SECTOR TIME-OUT (0901) 4-94
WRITE-LOCKOUT (0501) 4-94
FIRST ACTION (0301) 4-94
MEMORY INTERFACE PARITY ERROR (0081) 4-94
CONTROL CLEARED (0089) 4-94
General Information 4-95
Multisector-Per-Track (Standard) Format 4-96
Spare Sectors 4-96
LINE PRINTER 4-103
Printer Control 4-103

TABLE OF CONTENTS (Cont)

CDL Word Format 4-103
IOCW Information 4-103
Result Descriptor Format 4-104
Operation 4-105
PRINT (OP 10) 4-105
SKIP/SPARE (OP 11) 4-106
TEST (OP 99) 4-106
Error Termination 4-106
BUFFERED PRINTER CONTROL NO. 2 4-106
MAGNETIC TAPE 4-107
Tape Subsystem 4-107
TAPE EXCHANGES, FREE-STANDING UNITS 4-107
Magnetic Tape Exchange No. 2 4-107
Magnetic Tape Exchange No. 1 4-107
TAPE EXCHANGE, CLUSTER UNITS 4-107
CDL Word Format 4-108
IOCW Information 4-109
Result Descriptor Format 4-110
Operations 4-112
REWIND (OP 01) 4-112
READ OP 02 (FORWARD) OR OP 03 (REVERSE) 4-112
ERASE (OP 04) 4-112
WRITE (OP 06) 4-112
WRITE TAPE MARK (OP 06) 4-112
SPACE (OP 08 (FORWARD): OP 09 (REVERSE)) 4-112
TEST (OP 99) 4-112
BCL ALPHA OPERATION (7-TRACK TAPE WITH EVEN PARITY) 4-112
Exception Conditions 4-113
CRC CORRECTION (9-TRACK, 800 BPI ONLY) 4-113
PAPER TAPE PUNCH 4-114
Paper Tape Punch Control 4-114
CDL Word Format 4-114
IOCW Information 4-114
Result Descriptor Format 4-115
Operations 4-116
WRITE BCL (OP 48) 4-116
WRITE BINARY (OP 48) 4-116
PUNCH LEADER (OP 49) 4-116
TEST (OP 99) 4-116
Exception Conditions 4-117
PAPER TAPE READER 4-118
Paper Tape Reader Control 4-118
CDL Word Format 4-118
IOCW Information 4-119
Result Descriptor Format 4-119
Operations 4-120

TABLE OF CONTENTS (Cont)

READ BCL (OP 40) 4-120
READ BINARY (OP 40) 4-121
FORWARD SPACE (OP 41) 4-121
BACKWARD SPACE (OP 43) 4-121
REWIND (OP 47) 4-121
TEST (OP 99) 4-121
Exception Conditions 4-121
SINGLE LINE CONTROL 4-122
CDL Word Format 4-122
IOCW Information 4-123
Result Descriptor Format 4-123
Operation 4-124
READ (OP 32) 4-124
WRITE (OP 34) 4-125
TEST (OP 99) 4-125
CHAPTER V MEMORY SUBSYSTEM 5-1
SECTION 1 GENERAL DESCRIPTION OF THE MEMORY SUBSYSTEM 5-1
INTRODUCTION 5-1
Memory Capacity 5-1
MINIMUM MEMORY SIZE 5-1
MAXIMUM MEMORY SIZE 5-1
MEMORY CONFIGURATION 5-3
RECONFIGURATION 5-3
ADDRESS ALLOCATION 5-4
SUBSYSTEM ALLOCATION 5-4
Clock Rate and Access Times 5-4
M.S.U. INTERLACING 5-4
MULTIPLE-WORD TRANSFER (PHASING) 5-4
Word Size 5-5
REQUESTOR WORDS 5-5
MEMORY WORDS 5-6
Interface Signals 5-6
Memory Addressing 5-6
SECTION 2 MEMORY CONTROL MODULE (MCM) 5-10
GENERAL DESCRIPTION OF MCM 5-10
MCM Logic Functions 5-10
PRIORITY-RESOLUTION LOGIC 5-10
DATA-TRANSFER-AND-CONTROL LOGIC 5-10
ERROR-DETECTION LOGIC 5-10
MCM Communications 5-10
MCM FUNCTIONAL DETAILS 5-13
Requestor Interface 5-13
INFORMATION AND SIGNALS BETWEEN MCM AND REQUESTOR 5-13
Priority Resolver 5-16
Input Register 5-17
Control Word Register 5-17

TABLE OF CONTENTS (Cont)

MCM Operational Characteristics 5-17
DEFINITION OF MCM OPERATIONS 5-21
Requestor-Inhibit Register 5-22
Memory Address Limits Register 5-22
Memory Buffer Register 5-24
Failure Interrupt Signals 5-24
FAIL 1 INTERRUPT SIGNAL 5-24
FAIL 2 INTERRUPT SIGNAL 5-26
FAIL S INTERRUPT SIGNAL 5-26
Fail Register 5-27
Error Detection and Correction 5-27
Output Register 5-32
MSU Control 5-32
MCM-MSU Control and Data Interface 5-32
Master Clock and System Distribution 5-32
SECTION 3 MEMORY STORAGE UNIT (MSU) 5-33
GENERAL DESCRIPTION OF MSU 5-33
MSU FUNCTIONAL DESCRIPTION 5-33
MEMORY ADDRESS REGISTER (MAR) 5-34
MEMORY WRITE REGISTER (MWR) 5-34
MEMORY READ REGISTER (MRR) 5-34
Memory Interlacing and Phasing 5-35
CHAPTER VI FAIL SOFT AND MAINTENANCE 6-1
DETECTION, REPORTING, AND CORRECTION OF ERRORS 6-2
Detection and Reporting of Errors in the CPM 6-2
Detection and Reporting of Errors in the ICM 6-3
Detection and Correction of Single-Bit Errors in Memory 6-3
Detection and Reporting of Two-Bit Errors in Memory 6-3
INDEPENDENCE OF MODULES 6-3
Distributed Switching Interlock 6-4
Distributed Fail-Soft Power Subsystem 6-4
IDENTIFICATION AND REPAIR OF FAULTS 6-5
Confidence and Diagnostic Programs 6-5
Maintenance Diagnostic Unit 6-5
DATA RECOVERY 6-6
CHAPTER VII CONTROLS AND INDICATORS 7-1
SECTION 1 OPERATING CONTROLS 7-1
INTRODUCTION 7-1
CONSOLE CONTROL PANEL 7-1
SUPERVISORY CONSOLE 7-3
KEYBOARD CONTROL KEYS 7-3
SUPERVISORY CONSOLE OPERATION 7-5
COLD START/HALT LOAD SELECTION CARD 7-6
SECTION 2 CENTRAL PROCESSOR MODULE PANELS 7-8
DISPLAY ORGANIZATION 7-8
PANEL 2 INDICATORS (LEFT-HAND PANEL) 7-8

TABLE OF CONTENTS (Cont)

EU DATA SECTION 7-8
Row 1 - Row 21 Display Selection 7-8
Row 1 7-8
Row 21 7-8
Row 2 - Row 22 Display Selection 7-14
Row 2 7-14
Row 22 7-14
Row 3 - Row 23 Display Selection 7-14
Row 3 7-14
Row 23 7-15
Row 4 7-15
Row 5 - Row 25 Display Selection 7-16
Row 5 7-16
Row 25 7-17
EU CONTROL SECTION 7-17
Row 6 - Row 26 Display Selection 7-17
Row 6 7-17
Row 26 7-18
Row 7 7-19
Row 8 7-21
Row 9 7-22
Row 10 7-23
Row 11 7-24
Row 12 7-25
MISCELLANEOUS SECTION 7-27
Row 13 7-27
Row 14 - Row 34 Display Selection 7-28
Row 14 7-28
Row 34 7-28
Row 15 7-29
PANEL 2 SWITCHES (LEFT-HAND PANEL) 7-29
PANEL 1 INDICATORS (RIGHT-HAND PANEL) 7-30
COMMUNICATIONS SECTION 7-30
Row 1 - Row 21 Display Selection 7-30
Row 1 7-30
Row 21 7-30
Row 2 7-30
Row 3 7-31
STORAGE SECTION 7-32
Row 4 7-32
Row 5 7-32
ADDER SECTION 7-33
Row 6 7-33
STACK SECTION 7-35
Row 7 - Row 27 Display Selection 7-35
Row 7 7-35
Row 27 7-35

TABLE OF CONTENTS (Cont)

Row 8 - Row 28 Display Selection 7-35
Row 8 7-35
PROGRAM CONTROL UNIT SECTION 7-36
Row 9 - Row 29 Display Selection 7-36
Row 9 7-36
Row 29 7-37
Row 10 7-37
Row 11 7-38
Row 12 7-38
Row 13 7-40
Row 14 7-41
Row 15 7-41
PANEL 1 SWITCHES (RIGHT-HAND) PANEL 7-42
PANEL OPERATIONS 7-43
SETTING/RESETTING OF INDICATORS 7-43
LAMP TEST 7-44
STORAGE TEST 7-45
PROGRAM BUFFER READ 7-47
PROGRAM BUFFER WRITE 7-50
STACK BUFFER READ 7-52
STACK BUFFER WRITE 7-53
DISPLAY BUFFER-ADDRESSIBLE REGISTERS 7-54
DISPLAY BUFFER READ 7-55
DISPLAY BUFFER WRITE 7-57
EU LOCAL STORAGE - ADDRESSING 7-58
EU LOCAL STORAGE READ 7-59
EU LOCAL STORAGE WRITE 7-60
ASSOCIATIVE MEMORY READ 7-62
ASSOCIATIVE MEMORY WRITE 7-64
ASSOCIATIVE ADDRESS MEMORY READ 7-66
ASSOCIATIVE ADDRESS MEMORY WRITE 7-68
SECTION 3 INPUT/OUTPUT MODULE PANELS 7-71
SECTION 4 MEMORY CONTROL MODULE PANEL 7-77
ROWS 1 THROUGH 12 7-77
Row 1 7-77
Row 3 7-80
Row 4 7-80
Row 5 7-80
Row 6 7-80
Row 7 7-80
Row 9 7-80
Row 10 7-80
Row 11 7-81
Row 12 7-82
SWITCHES AND INDICATORS 7-83
MCM PANEL OPERATIONS 7-85
Lamp Test 7-85

TABLE OF CONTENTS (Cont)

Storage Test 7-85
Register Loading 7-85
Single Word Operations (Read or Write) 7-86
Conditional Halt Operation 7-86
SECTION 5 MAINTENANCE DIAGNOSTIC UNIT PANELS 7-88
CHAPTER VIII FUNCTIONS OF THE MASTER CONTROL PROGRAM (MCP) 8-1
SYSTEM STARTUP 8-3
HARDWARE INTERRUPTS 8-3
STORAGE CONTROL 8-5
CONTROL OF JOBS 8-5
Scheduling and Initiation of Jobs 8-7
Execution of Jobs 8-7
Software Interrupts and Events 8-9
Termination of Jobs 8-10
INPUT/OUTPUT CONTROL 8-10
File Handling 8-10
Transfer of Data 8-11
Data Communications 8-11
COMMUNICATION BETWEEN USER AND MCP 8-12
APPENDIX A ORDER OF MAGNITUDE CHART A-1
APPENDIX B HEXADECIMAL ADDITION TABLE B-1
APPENDIX C HEXADECIMAL TO DECIMAL CONVERSION TABLES C-1
APPENDIX D DECIMAL-HEXADECIMAL CONVERSION TABLE D-1
APPENDIX E COLLATING INFORMATION E-1
APPENDIX F DATA REPRESENTATION F-1
APPENDIX G WORD FORMATS G-1
APPENDIX H PROCESSOR OPERATORS, BY MNEMONICS H-1
APPENDIX I PROCESSOR OPERATORS, BY NAME I-1
APPENDIX J PROCESSOR OPERATORS, BY HEXADECIMAL CODE J-1
APPENDIX K IOM WORD FORMATS K-1

LIST OF ILLUSTRATIONS

Figure I-1. B 7700 Exchange..... 1-4

Figure I-2. Maximum Configuration of the B 7700 System..... 1-9

Figure II-1-1. Word Structure..... 2-2

Figure II-1-2. Number Base Graphic Characters..... 2-2

Figure II-1-3. Binary Integers..... 2-2

Figure II-1-4. Binary to Hexadecimal and Octal Conversion 2-4

Figure II-1-5. Relationship of Octal, Decimal and Hexadecimal Numbers 2-5

Figure II-1-6. Binary to Decimal Conversion 2-5

Figure II-1-7. Decimal to Binary Conversion 2-7

Figure II-1-8. Binary to Octal Conversion 2-7

Figure II-1-9. Powers of 8 2-8

Figure II-1-10. Octal to Decimal Conversion 2-8

Figure II-1-11. Single Precision Operand 2-11

Figure II-1-12. Order of Magnitude Chart 2-12

Figure II-1-13. Double Precision Operand 2-13

Figure II-1-14. Logical Operand 2-15

Figure II-1-15. String Operands 2-16

Figure II-1-16. Use of String Operand to Store a Signal Number (-4259) 2-17

Figure II-2-1. Polish Notation Flow Chart 2-20

Figure II-2-2. Evaluation of Polish String BC+7xA= 2-21

Figure II-2-3. Program Word 2-23

Figure II-2-4. Top of Stack and Stack Bounds Registers..... 2-24

Figure II-2-5. Stack Buffer and Stack Memory Area..... 2-26

LIST OF ILLUSTRATIONS (Cont)

Figure II-2-6. ALGOL Program With Lexicographical Structure and Related Stack Structure..... 2-28

Figure II-2-7. More Advanced ALGOL Program..... 2-30

Figure II-2-8. Addressing Environment Tree of ALGOL Program..... 2-30

Figure II-2-9. Stack History List..... 2-31

Figure II-2-10. Stack Cut Back on Procedure Exit..... 2-32

Figure II-2-11. Stack Operation..... 2-33

Figure II-2-12. Multiple Linked Stacks..... 2-36

Figure II-3-1. Basic Word Format..... 2-37

Figure II-3-2. Data Descriptor..... 2-41

Figure II-3-3. String Descriptor..... 2-42

Figure II-3-4. Segment Descriptor..... 2-43

Figure II-3-5. Program Control Word..... 2-44

Figure II-3-6. Indirect Reference Word..... 2-45

Figure II-3-7. Stuffed Indirect Reference Word..... 2-46

Figure II-3-8. Mark Stack Control Word..... 2-47

Figure II-3-9. Return Control Word..... 2-49

Figure II-3-10. Top of Stack Control Word..... 2-50

Figure II-3-11. Step Index Word..... 2-51

Figure II-3-12. Occurs Index Word..... 2-51

Figure II-3-13. Time of Day Function Word..... 2-52

Figure III-1-1. Simplified Block Diagram of Central Processor Module..... 3-2

Figure III-1-2. Division Overlap 3-4

Figure III-1-3. Memory Overlap 3-7

LIST OF ILLUSTRATIONS (Cont)

Figure III-1-4. Program Buffer Arrangement 3-8

Figure III-1-5. Program Buffer Unit 3-9

Figure III-1-6. Program Barrel 3-12

Figure III-1-7. Program Control Unit 3-14

Figure III-1-8. Address Couple Bit Assignment 3-18

Figure III-1-9. Address Unit 3-21

Figure III-1-10. Execution Unit, Major Data Circuits 3-28

Figure III-1-11. Execution Unit, Major Control Circuits 3-32

Figure III-1-12. Stack Buffer and Stack Memory Area 3-39

Figure III-1-13. Stack Buffer Operation 3-40

Figure III-1-14. Local Buffer System 3-43

Figure III-1-15. Associative Memory Storage Area 3-44

Figure III-1-16. Associative Memory 3-45

Figure III-1-17. Storage Unit 3-47

Figure III-1-18. Communications Unit 3-51

Figure III-1-19. Fault Control Logic 3-53

Figure III-2-1. CPM Fail Register 3-62

Figure III-2-2. MCM Fail Register 3-63

Figure III-2-3. IOM Fail Register 3-65

Figure III-2-4. Stack Format 3-67

Figure III-2-5. Stack Format Prior To Calling Interrupt Procedure While In CM1 (Move Stack Operation) 3-68

Figure III-2-6. Interrupt Reporting 3-70

Figure III-2-7. Stack Format Before Reentering Interrupt Procedur to Report Stack Overflow 3-71

LIST OF ILLUSTRATIONS (Cont)

Figure III-2-8. Stack Format After Reentering Interrupt Procedure and Reporting Stack Overflow 3-72

Figure III-2-9. Presence Bit Interrupt Chart 3-85

Figure III-3-1. Format of Program Buffer Word Storage 3-91

Figure III-3-2. Program Buffer Word Format 3-93

Figure III-3-3. B 7700 CPM Program Operator Hexadecimal Code Assignments 3-95

Figure III-3-4. Step Index Word Format 3-109

Figure III-3-5. Flow Chart of Value Call Operator 3-134

Figure III-3-6. Flow Chart of Value Call Operator 3-135

Figure III-3-7. Flow Chart of Evaluate Descriptor Operator 3-137

Figure III-3-8. Flow Chart of Enter Operator 3-138

Figure III-3-9. Flow Chart of Exit Operator 3-140

Figure III-3-10. Flow Chart of Return Operator 3-141

Figure III-3-11. Flow Chart of Stuff Environment Operator 3-144

Figure III-3-12. Time of Day Word Format 3-149

Figure III-3-13. Occurs Index (OCRX) Format.....3-157

Figure IV-1-1. IOM Basic Block Diagram..... 4-1

Figure IV-1-2. Typical Data-Transfer Classifications and Related IOM Subsections..... 4-3

Figure IV-1-3. Typical IOM/Peripheral Configuration..... 4-4

Figure IV-1-4. Typical IOM/Main Memory and IOM/CPM Interface Configurations..... 4-9

Figure IV-1-5. IOM Job Map..... 4-12

Figure IV-1-6. HA Word Content, Start I/O Command..... 4-17

LIST OF ILLUSTRATIONS (Cont)

Figure IV-1-7. HA Word Content, Set/Reset Channel Busy/Channel Reserved Commands..... 4-19

Figure IV-1-8. HA Word Content, Load Address Commands (HA, UT, IOQH, SQH)..... 4-20

Figure IV-1-9. HA Word Content, DFO Scan-Out Commands (Clear the Stack, Store Control Word Request)..... 4-21

Figure IV-1-10. HA Word Content, DCP Scan-Out Commands (Initialize, Halt, Set Attention)..... 4-22

Figure IV-1-11. HA Word Content, DFO Scan-In Commands (Queued Control Word, Top of Stack, Report)..... 4-23

Figure IV-1-12. HA Word Content, Synchronous I/O Command..... 4-24

Figure IV-1-13. HA Word Content, Interrogate Peripheral Status Command..... 4-25

Figure IV-1-14. HA Word Content, Inhibit IOM, Activate IOM Commands..... 4-26

Figure IV-1-15. HA Word Content, Load DFO Flags Command..... 4-26

Figure IV-2-1. Translator Component Interface..... 4-35

Figure IV-2-2. Memory Interface Unit..... 4-41

Figure IV-2-3. Peripheral Control Interface..... 4-44

Figure IV-2-4. Disk File Interface..... 4-48

Figure IV-2-5. Scan Bus Interface..... 4-51

Figure IV-2-6. DCI Unit..... 4-53

Figure IV-3-1. Disk-Pack Recording Surfaces..... 4-95

Figure IV-3-2. Standard Format (Sheet 1)..... 4-100

Figure IV-3-2. Standard Format (Sheet 2)..... 4-101

Figure IV-3-3. Single Sector For Track Format..... 4-102

Figure IV-3-4. Single Line Control Configuration..... 4-125

LIST OF ILLUSTRATIONS (Cont)

Figure IV-3-5. Read Message Format TC500 and B 9352 4-126

Figure IV-3-6. Write Message Format TC500 and B 9352 4-127

Figure V-1-1. B 7700 Memory Subsystem Modularity Diagram..... 5-2

Figure V-1-2. Data Word Transfer Between Requestor and Memory.... 5-7

Figure V-1-3. Requestor-MCU-MSU Interface..... 5-8

Figure V-2-1. Memory Control Module Block Diagram..... 5-11

Figure V-2-2. MCM Control Word Format (2 Sheets) 5-18

Figure V-2-3. Load Requestor Inhibit Word Format..... 5-23

Figure V-2-4. Memory Address Limits Word Format..... 5-25

Figure V-2-5. MCM Fail Word Format (3 Sheets) 5-28

Figure V-2-6. Bit Positions for Error Detection and Correction ... 5-31

Figure V-3-1. Interlacing MSU's 5-35

Figure VII-1-1. Operators Control Console..... 7-2

Figure VII-1-2. Console Control Panel..... 7-3

Figure VII-1-3. Supervisory Console..... 7-4

Figure VII-1-4. Keyboard Format..... 7-4

Figure VII-1-5. Cold Start/Halt Load Selection Card..... 7-6

Figure VII-2-1. Central Processor Module..... 7-9

Figure VII-2-2. Panel 2 (Left-Hand Panel) of CPM..... 7-11

Figure VII-2-3. Panel 1 (Right-Hand Panel) of CPM..... 7-13

Figure VII-3-1. Overall View of IOM..... 7-72

Figure VII-3-2. Left-Hand Panel of IOM..... 7-73

Figure VII-3-3. Right-Hand Panel of IOM..... 7-75

LIST OF ILLUSTRATIONS (Cont)

Figure VII-4-1. Overall View of MCM..... 7-77

Figure VII-4-2. Panel of MCM..... 7-79

Figure VII-5-1. Maintenance Diagnostic Unit..... 7-89

Figure VII-5-2. System Diagnostic Panel..... 7-90

Figure VII-5-3. Card Test Panel..... 7-91

Figure VII-5-4. Maintenance Panel..... 7-93

LIST OF TABLES

Table I-1. Central Components of the B 7700 System 1-5
Table II-1-1. Sign Configurations of String Operands..... 2-18
Table II-2-1. Description of Stack Operation..... 2-34
Table III-2-1. B 7700 Interrupt Bit Assignments 3-58
Table III-2-2. CPM Handling of Memory Related Errors 3-74
Table III-2-3. Processor Internal Interrupts 3-80
Table IV-1-1. PCI/PCC Channels..... 4-5
Table IV-1-2. DFI/PCC Channels..... 4-7
Table IV-1-3. IOM HA Operations and Corresponding Home Codes..... 4-16
Table IV-1-4. General Translation Specification Codes..... 4-28
Table IV-1-5. Device Related Code Translations..... 4-29
Table IV-2-1. Unit Table and Active Channel Coded Decisions..... 4-36
Table IV-3-1. Disk-Pack Subsystem Characteristics..... 4-97
Table IV-3-2. Disk-Pack File Addresses (Burroughs Multi-Sector Format)..... 4-98
Table IV-3-3. Disk-Pack File Addresses (Single Sector/Track Format)..... 4-99
Table V-1-1. B 7700 Memory Module Combinations..... 5-3
Table V-1-2. B 7700 Memory System Read Access Time..... 5-5
Table V-1-3. B 7700 Memory System Effective Read Access Time(s).. 5-5
Table V-2-1. MCM Typical Times of Operation..... 5-20
Table V-2-2. Legal Operation Codes for the MCM..... 5-20

INTRODUCTION

This system reference manual presents the technical details about the general architecture, the components, the subsystems, and the software executive program (or master control program) of the Burroughs B 7700 Information Processing System--the most advanced, the largest, and the most powerful member of the Burroughs family of 700 systems. For a more general coverage of the hardware and software characteristics of the system, the reader should refer to the system characteristics manual.

The eight chapters of this reference manual are as follows.

Chapter I, Description of the B 7700 System: an introduction to the idea of the interaction of independently operating computing, input/output, and memory modules through an exchange and a presentation of the range of configurations of the system.

Chapter II, System Architecture: discussions of data representation, Polish notation and stack concepts, processor control words, and the concepts of the input/output subsystem map.

Chapter III, Central Processor Module: a functional description of the operation of the central processor module, an explanation of hardware interrupts, and a brief description of each program operator.

Chapter IV, Input/Output Subsystem: a general description of the operation of the input/output module, functional descriptions of the subsections of the input/output module, and detailed descriptions of the control words and descriptors associated with each type of peripheral device that may be included in the system.

Chapter V, Memory Subsystem: a general description of the memory subsystem and details about both the memory control module and the memory storage unit.

Chapter VI, Fail Soft and Maintenance: a summary of the features of the system that help to maximize its availability to the user.

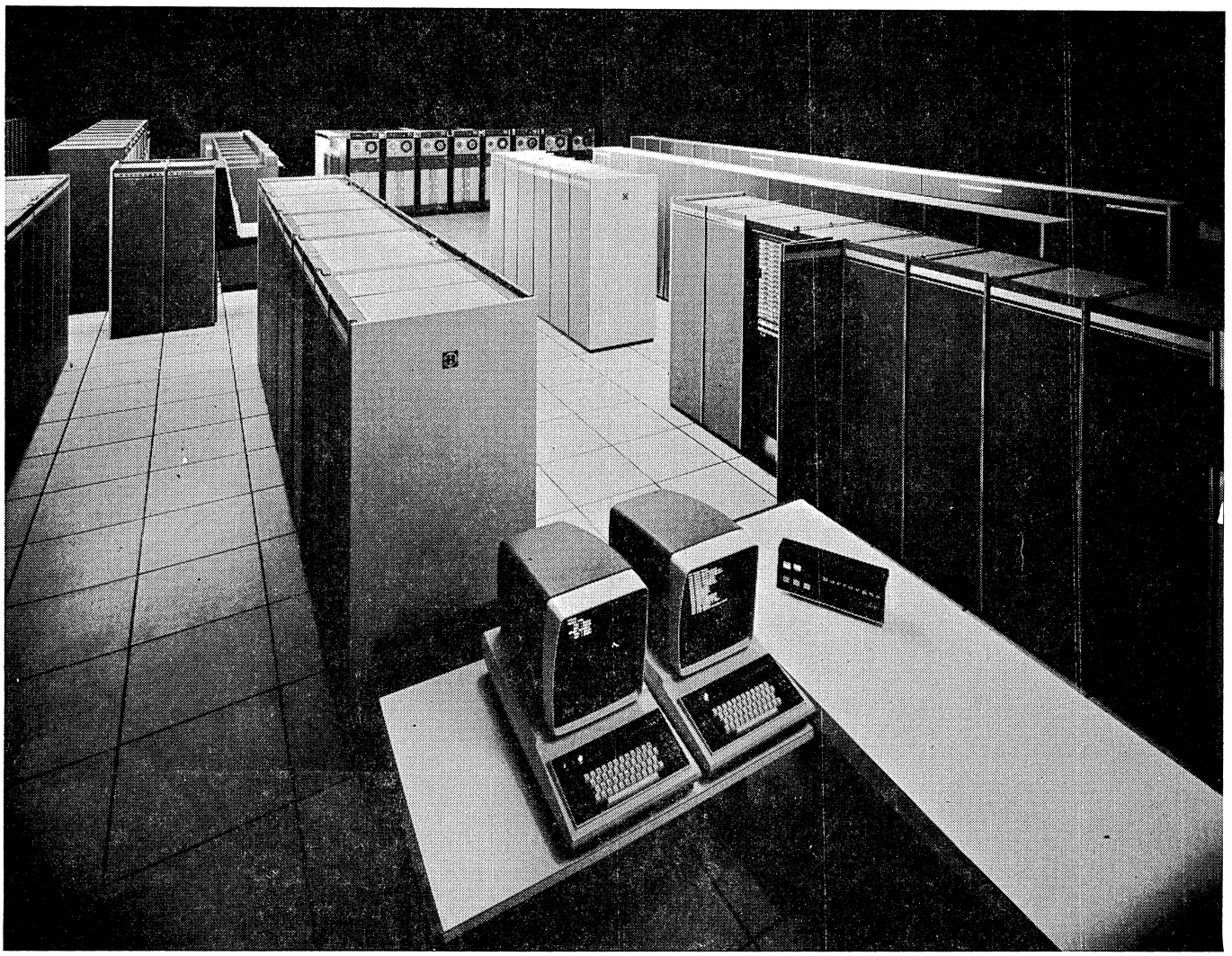
Chapter VII, Controls and Indicators: detailed descriptions of the functions and uses of the controls and indicators of the central components of the system.

Chapter VIII, Functions of the Master Control Program (MCP): an introduction to the master control program--the key to the effective, balanced use of system resources.

The term "software" as used in this manual applies to that category of Burroughs Program Products defined as "Systems Software".

Other categories of Burroughs Program Products are:

Application Program Products
Program Product Development Aids
Program Product Conversion Aids



CHAPTER I
DESCRIPTION OF B 7700 SYSTEM

THE B 7700 SYSTEM

The Burroughs B 7700 Information Processing System is a large-scale, truly general-purpose, balanced, flexible, multiprogramming and multiprocessing computing system that is suitable for such diverse applications as time sharing, scientific problem solving, and business data processing. Carrying forward ideas proven successful in the Burroughs B 5700 and B 6700 information processing systems, the B 7700 is, in fact, fully code compatible with the B 6700 and affords Burroughs users the opportunity for growth without reprogramming or recompiling. In other words, object code of users' programs that can be executed successfully on the B 6700 can be executed without modification on the B 7700, and object code that can be executed on the B 7700 can be executed without modification on the B 6700. Nevertheless, the B 7700 is designed to satisfy the increasingly complex data processing needs of the years to come. The system is able to handle complex data structures and sophisticated program structures dictated both by higher-level languages now in use and by the requirements of advanced problems, is able to manage efficiently the massive on-line and archival storage requirements of large data bases, and is able to accommodate vast networks of data communications devices.

A very fast, modular parallel processing system with exceptional versatility in configuration, the B 7700 can be tailored to the processing needs of a user by arranging central processor modules, input/output modules, and memory modules on an electronic grid, or exchange (figure I-1), in a variety of ways depending upon the exact needs of the user. If the high performance and adaptability of the B 7700 could be attributed to a single factor, it would be to the balance attained by means of the controlled interaction of independently operating computing, input/output, and memory modules through the exchange. Thus, the throughput of the system as a whole is maximized, and the performance of no single element of the system is maximized to the neglect or detriment of others.

The key to the efficient and balanced use of the system is the Burroughs master control program (MCP)- a unique executive software operating system that automatically makes optimum use of all system resources. It is this operating system that makes multiprogramming and multiprocessing both functional and practical by dynamically controlling system resources and by scheduling jobs in the multiprogramming mix. In use, the master control program allocates system resources to meet the needs of the programs introduced into the computer. It continually and automatically reassigns resources, starts jobs, and monitors their performance.

Further implications of the modularity and flexibility of the system are its expandability (a capacity to add hardware modules without reprogramming) and its increased reliability (and thus increased availability to the user). This reliability is achieved by the use of fail-

soft techniques that (in addition to providing for error detection and error correction, redundancy of data paths, and independence and redundancy of power supplies) exclude faulty modules from the system and permit processing to continue (again, without reprogramming) even with a temporarily reduced configuration.

Even though it is very large and immensely complicated, the B 7700 is, nevertheless, comprehensible to the persons who use it: programming is done only in higher-level, problem-oriented languages (COBOL, ALGOL, FORTRAN, PL/I, and ESPOL); the control language used in entering jobs into the system is a simple, free-form, English-like language; and the messages that pass between the system and the operator are brief, clear, and easy to learn.

DISTINGUISHING FEATURES

Although the balanced use of the principal components of the system as a whole under the control and coordination of the master control program is the key to the high throughput of the B 7700, the high performance of the system is in large part achieved by improving the speed of execution of instructions, by reducing or masking the overhead associated with references to memory, by freeing the central processor from concern with input/output operations, and by employing fail-soft measures that minimize system degradation.

Because system main-frame hardware has been designed and built strictly according to stringent circuit and wiring rules and proven design and packaging techniques and because its processing elements incorporate monolithic integrated circuits, the B 7700 system performs consistently at high operating frequencies: the central processor module at a clock rate of 16 megahertz and the remainder of the system at 8 megahertz.

By combining the following features with the high internal operating frequencies, the performance of the system is further enhanced.

1. The parallel and independent operation of the three main sections (program, execution, and storage) of the central processor module. This parallelism (coupled with the high clock rate) makes possible the speeding up of arithmetic computations and data manipulations and the overlapping of these computations and manipulations with memory references.

2. A special high-speed integrated circuit memory (program, stack, and associative data buffers). This high-speed local memory permits multiword transfers between the central processor and main memory and makes possible the anticipation of the need for program and data words. Hence, the time spent waiting for the completion of transfers to and from memory is reduced and at times virtually eliminated.

3. The four-way interleaving of addresses in main memory and the capability for phased multiword transfers of information to and from memory in groups of up to four words.

Consequently, memory access times for each user of memory are reduced, and memory is thus made more accessible to all users.

4. The asynchronous performance of input/output operations by the input/output module independent of the central processor, which is therefore freed to do other useful work.

The three goals of the fail-soft features of the B 7700 are to keep the system running 100 percent of the time, to minimize system degradation, and to provide the user with tools for performing his own data recovery. These goals are achieved by the artful combination of hardware and software throughout the system.

The first goal--to keep running--is achieved as follows.

1. By the high reliability of system hardware.

2. By the incorporation of error detection circuits throughout the system.

3. By single-bit error correction of errors in memory.

4. By recording errors for software analysis.

5. By modular design, by use of separate power supplies and redundant regulators for each module, and by use of redundant buses.

6. By the ability of the master control program to reconfigure the modules of the system to temporarily exclude a faulty one.

In short, the detection and reporting of errors is done by hardware, analysis of errors is done by software, and the reconfiguration of the system is done dynamically by software. Because of the modularity of power supplies and the use of redundant regulated supplies for critical voltages, the impact of a malfunctioning dc supply is minimized and does not result in a catastrophic failure.

The second goal--to minimize system degradation--is achieved by providing diagnostic programs and equipment for rapidly identifying and repairing faults and for reestablishing confidence in a repaired module before it is returned to the user's system. The diagnostic programs of the B 7700 system identify a faulty module. By the use of the maintenance diagnostic unit, a fault in any main-frame module or in a disk file optimizer is narrowed to a single clock period and to a flip-flop and its associated logical circuits. Finally, by the use of the card tester on the maintenance diagnostic unit, the faulty integrated circuit chip is identified.

The third goal--to provide the user with tools for performing his own data recovery--is achieved by the use of such features as installation allocated disk, protected disk files, duplicated disk files, and fault statements in the higher-level programming languages used on the system.

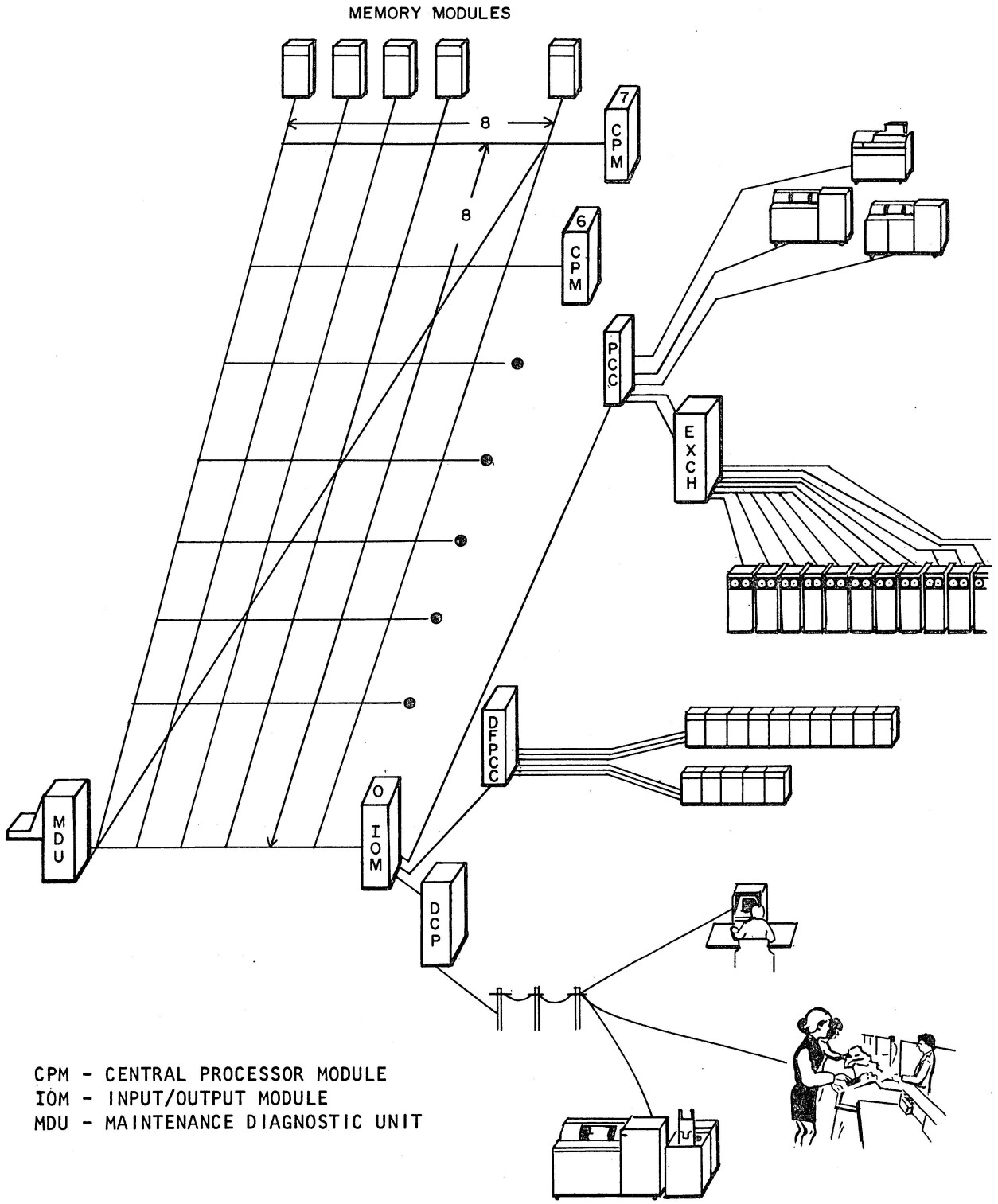


Figure I-1. B 7700 Exchange

Installation allocated disk allows the user to specify the physical allocation of his critical disk files in order to facilitate the maintenance and reconstruction of these files. Protected disk files allow a user to gain access to the last portion of valid data written in a file before an unexpected system halt. The use of duplicated disk files is to avoid the problem of fatal disk file errors. The master control program maintains more than one copy of each disk file row, and, if access cannot be gained to a record, an attempt is made to gain access to a copy of the record. By the use of fault statements, the user can stipulate the actions to be taken by his programs in case certain errors occur.

SYSTEM CONFIGURATION

Physically, the components of the B 7700 system fall into three categories, as follows:

1. Central components of the B 7700 system--the central processor module, the input/output module, the memory module, the maintenance diagnostic unit, and the operator's console (see table I-1).

2. Standard Burroughs cabinets that contain peripheral controls and exchanges, the disk file optimizer, the data communications processor, and ac power supplies.

3. Standard peripheral devices that are joined to the central system by means of standard Burroughs peripheral controls, adapters, and exchanges and standard remote devices that are joined to the central system by means of line adapters and the data communications processor.

The arrangement of these components into a system and the size of the system depend on the application and workload of the user. In the following paragraphs, the range of configurations of the B 7700--the maximum configuration, the minimum configuration, and the typical configuration with full fail-soft capabilities--is described.

Table I-1. Central Components of the B 7700 System

Style Number	Name	Description
B 7700	Basic system	One 16-megahertz, parallel processing central processor module One asynchronously-operating input/output module containing four multiword channels for disk file controls, 20 word channels for peripheral controls, four word channels

Table I-1. Central Components of the B 7700 System (Cont'd)

Style Number	Name	Description
		for data communications processors, and one disk file optimizer adapter
		Processor / memory exchange
		Operator's console and control
		One maintenance diagnostic unit
B 7701	Additional central processor module	See above.
B 7780	Additional input/output module	See above.
B 7780-1	Multiword channels	Four multiword channels for disk file controls and one disk file optimizer adapter
B 7001-2	Memory module	786,432 eight-bit bytes (131,072 words) of core memory storage
		1.5-microsecond cycle time
		Two-way interleaving that permits two-word transfers to and from memory
B 7001-4	Memory module	1,572,864 eight-bit bytes (262,144 words) of core memory storage
		1.5-microsecond cycle time
		Four-way interleaving that permits four-word transfers to and from memory

Table I-1. Central Components of the B 7700 System (Cont'd)

Style Number	Name	Description
B 7341	Additional operator's display control	Controls a maximum of eight operator's display terminals
B 9342-1	Additional operator's display terminal	
B 9951-7	Console display stand	Low, without work table
B 9951-8	Console display stand	High, without work table
B 9951-9	Console display stand work table	Right or left

1 byte = 2 digits
1 digit = 4 bits.
1 word = in data signal 6 bytes. (6×2×4 = 48 bits).

Maximum Configuration

Figure I-2 illustrates the theoretical maximum configuration of the B 7700 system. As many as eight memory modules may be arranged on the exchange with a combined total of up to eight requestors of memory--central processor modules and input/output modules. Any single requestor of memory may address and gain access to the entire contents of high-speed main memory (1,048,576 words, or 6,291,456 eight-bit bytes). On the maintenance bus (which services the memory control modules, central processor modules, input/output modules, and disk file optimizers) one or two maintenance diagnostic units may be placed.

At rates of up to 6.75 million bytes per second, a single input/output module is capable of transferring data simultaneously between main memory and 28 peripheral controls (including eight high-speed controls) and between main memory and as many as four data communications processors. It is also capable of handling as many as four disk file optimizers (devices that are used in improving the rate of transfer of data between main memory and disk files). At present, the maximum number of high-speed, medium-speed, and low-speed peripheral devices that may be attached through controls and exchanges to a single input/output module or that may be included in the input/output subsystem of the B 7700 is 255. (Each card reader, pseudoreader, card punch, line printer, paper tape reader, paper tape punch, operator's display terminal, and free-standing magnetic tape unit; each station on a magnetic tape cluster; and each electronics unit in a disk file subsystem is considered a device.) By suitable cross-connection through exchanges, it is possible to establish pathways between disk files, disk packs, or magnetic tape units and more than one input/output module; hence, these peripheral devices can be shared by all of the input/output modules in the system.

Among the peripheral devices available are disk file and disk pack memory modules that constitute a virtual memory that in effect greatly expands the storage capacity of the main memory of the system; these modules, which are interfaced with the input/output module through controls are as follows:

1. Head-per-track disk file modules that are combined under the control of disk file optimizers to form optimized-access memory banks capable of storing from 450 million to 8 billion eight-bit bytes of information per input/output module and whose access time is as low as 2 to 6 milliseconds.

2. Head-per-track disk file modules that are combined (without the control of the optimizer) into randdom-access memory banks of from 15 million to 16 billion eight-bit bytes per input/output module and whose average access time is 23 or 40 milliseconds.

3. Disk pack memory modules that are combined into random-access memory banks with a capacity of from 121 million to many billions of eight-bit bytes of storage per input/

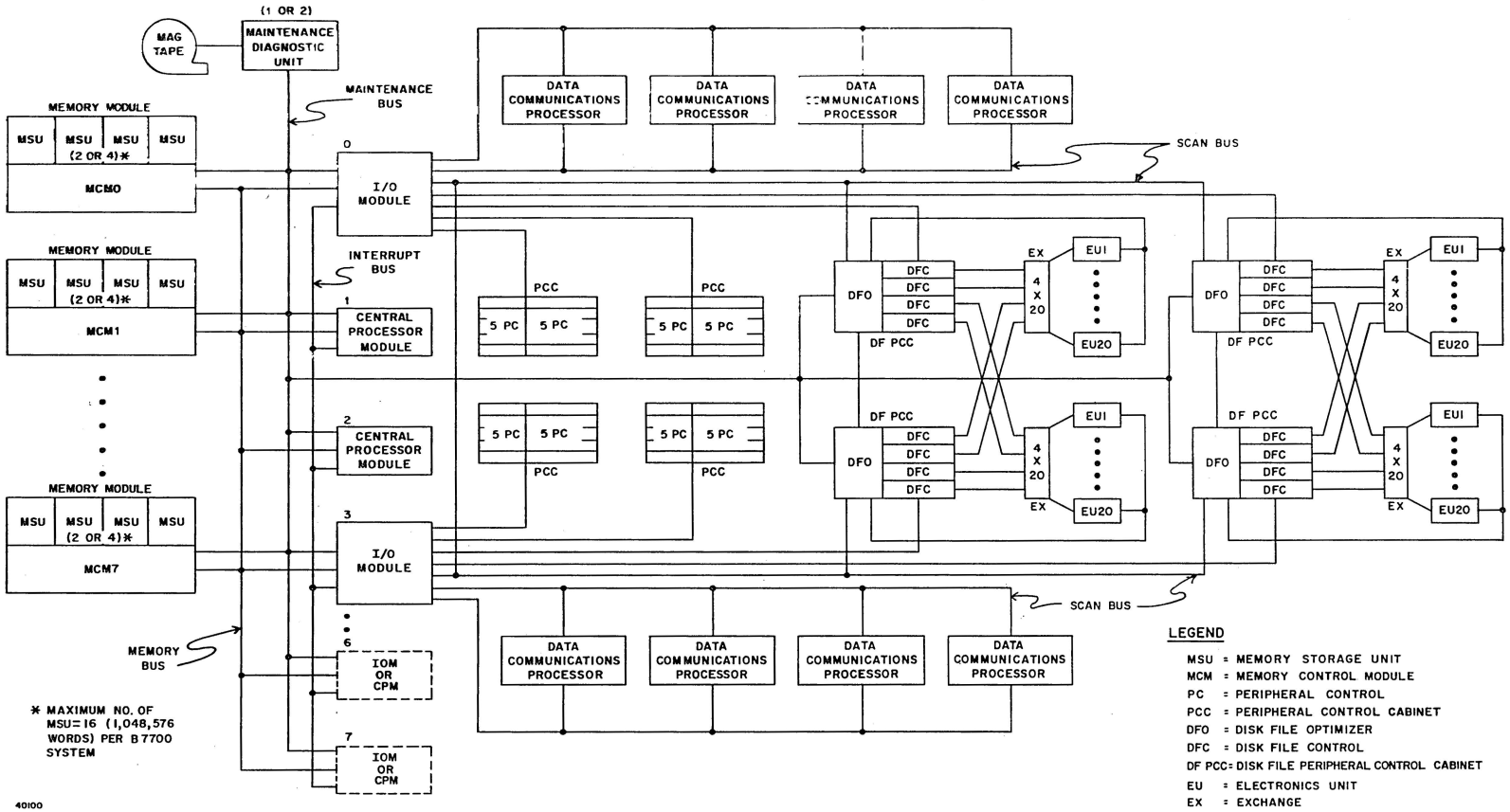


Figure I-2. Maximum Configuration of the B 7700 System

output module and whose average access time is 30 milliseconds.

Besides the 255 peripheral devices that may be included in the input/output subsystem, there is a vast network of remote terminals, remote controllers, and remote computers that can be accommodated by the up to 1024 remote lines serviced by the four programmable data communications processors that can be controlled by a single input/output module. Normally, each line handles a number of remote devices, and, naturally, systems that have more than one input/output module can have more than one data communications network. The maximum number of data communications processors that may be included in a B 7700 system is 28.

Minimum Configuration

The smallest possible B 7700 system is composed of the central components listed below.

Central Components	Quantity
Central processor module (CPM)	1
Input/output module (IOM)	1
Memory module	1
Memory control module (MCM)	1
Memory storage cabinet (MSC)	1
Memory storage unit (MSU)	2
Maintenance diagnostic unit (MDU) and its associated magnetic tape unit	1
Operator's console	1

Besides these central components, the minimum configuration must contain a disk file memory subsystem at least large enough to hold the master control program, a card reader, a line printer, a magnetic tape unit, peripheral controls, and ac power cabinets. In practice, other peripheral devices and their controls are used with this minimum configuration.

Naturally, this minimum system lacks the redundancy and power of larger configurations. First (lacking redundancy of main-frame modules) this configuration does not take full advantage of the fail-soft features possible with the B 7700 and second (because each memory control module controls but two storage units) two-word transfers, not four-word transfers, to and from memory are possible.

Typical Configuration With Full Fail-Soft Capabilities

The power, speed, flexibility, and reliability of which the B 7700 is capable are fully realized in a system that includes the following central components.

Central Components	Quantity
Central processor module (CPM)	2
Input/output module (IOM)	2
Memory module	4
Memory control module (MCM)	4
Memory storage cabinet (MSC)	8 (2 per MCM)
Memory storage unit (MSU)	16 (2 per MSC)
Maintenance diagnostic unit and its associated magnetic tape unit	1
Operator's console	2 (1 per IOM)

Besides these central components, this typical fail-soft configuration must contain two disk file memory subsystems (one for each input/output module) or a single disk file subsystem that is shared by means of exchanges by the two input/output modules, peripheral controls, and ac power cabinets. Naturally, a complement of peripheral devices and their controls and exchanges, data communications processors, and remote devices suited to the application and workload of the system is also needed.

A system of the proportions described above incorporates fully the fail-soft features of the B 7700 and takes complete advantage of its capability of handling four-word transfers of data to and from main memory.

CHAPTER II
SYSTEM ARCHITECTURE

SECTION 1
DATA REPRESENTATION

GENERAL

The basic information structure used in the B 7700 Information Processing System is the word. Each word contains 48 information bits, three tag bits, and one parity bit (see figure II-1 1). The information bits may be used to store character values, logical values, or numeric values. The tag bits are control bits which identify the type of information contained in the information field. The tag bits are inaccessible to normal state (user) programs. The parity bit is used to check for correct information transfer between the CPM or IOM and main memory.

INTERNAL CHARACTER CODES AND COLLATING SEQUENCES

Extended Binary Coded Decimal Interchange Code (EBCDIC) is the primary internal character code of the B 7700. EBCDIC is an eight-bit alphanumeric code containing four zone and four numeric bits. Other internal codes which may be used include the American Standard Code for Information Interchange (ASCII), and the Burroughs Common Language Code (BCL). ASCII is the primary data communication code; BCL is used to interface with peripheral units. Numeric EBCDIC and BCL codes may be packed into four-bit digits by internal commands which delete the zones and compress the numeric portion of the characters. In general, characters are collated according to their internal binary value. Character codes and collating sequences are provided in the appendices (see legend).

NUMBERS AND NUMBERING SYSTEMS

The B 7700 is a digital computer; that is, values are stored internally in binary digits (bits). Data display in registers and printed forms may be in octal or hexadecimal format. Generally, we think in terms of, and manually perform arithmetic with, decimal numbers. Thus, an understanding of all of these numbering systems is desirable.

The decimal system is based on the first ten digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and upon the powers of ten. Similarly, the binary system is based upon the first two digits, 0 and 1, and the powers of two. Two raised to the third power (2^3) is 8, the base of the octal system. Likewise, 2 raised to the fourth power (2^4) is 16, the base of the hexadecimal system. The set of digits for each number system is shown in figure II-1-2.

The digits 0 through 9 and the alphabetic characters A through F comprise the 16-character requirement for the hexadecimal numbering system. The letter A is assigned a value of 10, B equals 11, etc., to F which equals 15.

* tag field is major factor determining the meaning of the data field!

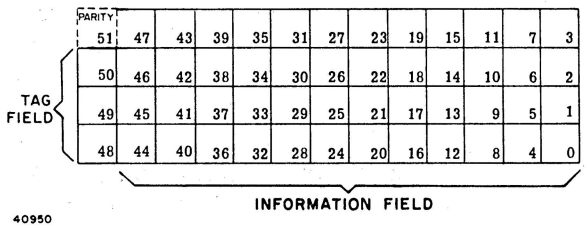


Figure II-1-1. Word Structure

DECIMAL	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
BINARY	0 1
OCTAL	0 1 2 3 4 5 6 7
DECIMAL	0 1 2 3 4 5 6 7 8 9
HEXADECIMAL	0 1 2 3 4 5 6 7 8 9 A B C D E F

40951

Figure II-1-2. Number Base Graphic Characters

Binary Notation

The decimal system is based upon the ten digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and upon the powers of ten. Similarly, the binary system is based upon the two digits, 0 and 1, and upon the powers of two. Every binary digit is represented by one flip-flop, thus a binary digit is one bit. A number will be represented internally as a series of bits either off or on. When a bit is on (1), its position determines the value. Consider an example of five bits.

The least significant bit, if on (1), has a value of 2^0, or 1; the next most significant bit to the left of the binary point has the value of 2^1, or 2; the third bit (count from right to left) has the value of 2^2, or 4; etc. In this manner, any integer can be represented in binary form. Figure II-1-3 illustrates some integers. Fractions in binary are much the same as integers. Here, though, the powers are negative powers with the first power to the right of the binary point having the value of 2^{-1}, or 1/2; the second bit has the value

$2^0 = 1$

0 = off bit

1 = on bit

value of position = $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

...0 0 0 0 1 = 0 +0 +0 +0 + 1 = decimal 1

...0 0 0 1 0 = 0 +0 +0 +2 + 0 = decimal 2

...0 0 0 1 1 = 0 +0 +0 +2 + 1 = decimal 3

.

.

.

...1 1 1 1 1 = $2^4+2^3+2^2+2^1+ 1$ = 1 = 16 + 8 + 4 + 2 + 1 = decimal 31

40952

Figure II-1-3. Binary Integers

of 2^{-2}, or 1/4; the third bit 2^{-3}, or 1/8; the fourth bit, 2^{-4}, or 1/16; etc. It is apparent that while some fractions are represented correctly, others can only be approximated. However, the degree of error is very small when a sufficient number of bits are used.

Hexadecimal and Octal Notation

Since binary words are cumbersome to display, the more efficient methods of Hexadecimal and Octal notation are employed. The hexadecimal representation of a binary word is obtained by dividing the bits into groups of four with each group assigned a successive power of 16. A binary-to-octal conversion is obtained by dividing the bits into groups of three and assigning successive powers of 8 to each group (figure II-1-4).

The relationship between octal, decimal and hexadecimal is shown in figure II-1-5 using the decimal number 1013_{10} (equivalent to 1765_8 and $3F5_{16}$ where the subscript 8, 10, or 16 indicates the base).

NUMBER CONVERSION

Binary to Decimal Conversion

INTEGRAL

This conversion is effected by adding together the value of each bit that is on. In this way, the binary number 11010011 would be equal to:

$$1x2^7+1x2^6+0x2^5+1x2^4+0x2^3+0x2^2+1x2^1+1x2^0=$$

$$1x2^7+1x2^6+0+1x2^4+0^{+0}+1x2^1+1x2^0=$$

$$128 + 64 + 16 + 2 + 1 \quad 211_{10}$$

A second method of effecting a binary to decimal conversion is the "double dabble" method. In this procedure, the high-order bit is doubled (multiplied by 2) and then added to the next lower-order bit. This sum is then doubled and again added to the next lower bit. This process is continued until the entire binary number has been expended (figure II-1-6A). The correct result is obtained after the low-order bit (units) has been added.

FRACTIONAL

The above process will work for integral numbers and for the integral part of fractional numbers, but it will not work for the fractional part of fractional numbers. To convert binary fractions to decimal fractions, division is used. As was previously stated, the bits to the right of the binary point have the decreasing values of 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, etc., or, as fractions 1/2, 1/4, 1/8, 1/16, etc., respectively. N1 is the binary digit in the first position to the right of the binary point, N2 is the digit in the second position to the right of the binary point, etc. Thus, a binary fraction would be expressed in the decimal equivalent as:

$$N1x2^{-1} + N2x2^{-2} + N3x2^{-3} = \frac{\frac{N3}{2} + N2}{\frac{2}{2}} + N1 = \text{decimal equivalent fraction}$$

This can be used to find the decimal equivalent of a binary fraction. In this process, the lowest order significant bit is taken as the integer 1 and divided by 2. The next higher-order bit is then added into the unit's position of the resulting quotient, and the division is repeated. This is repeated until the

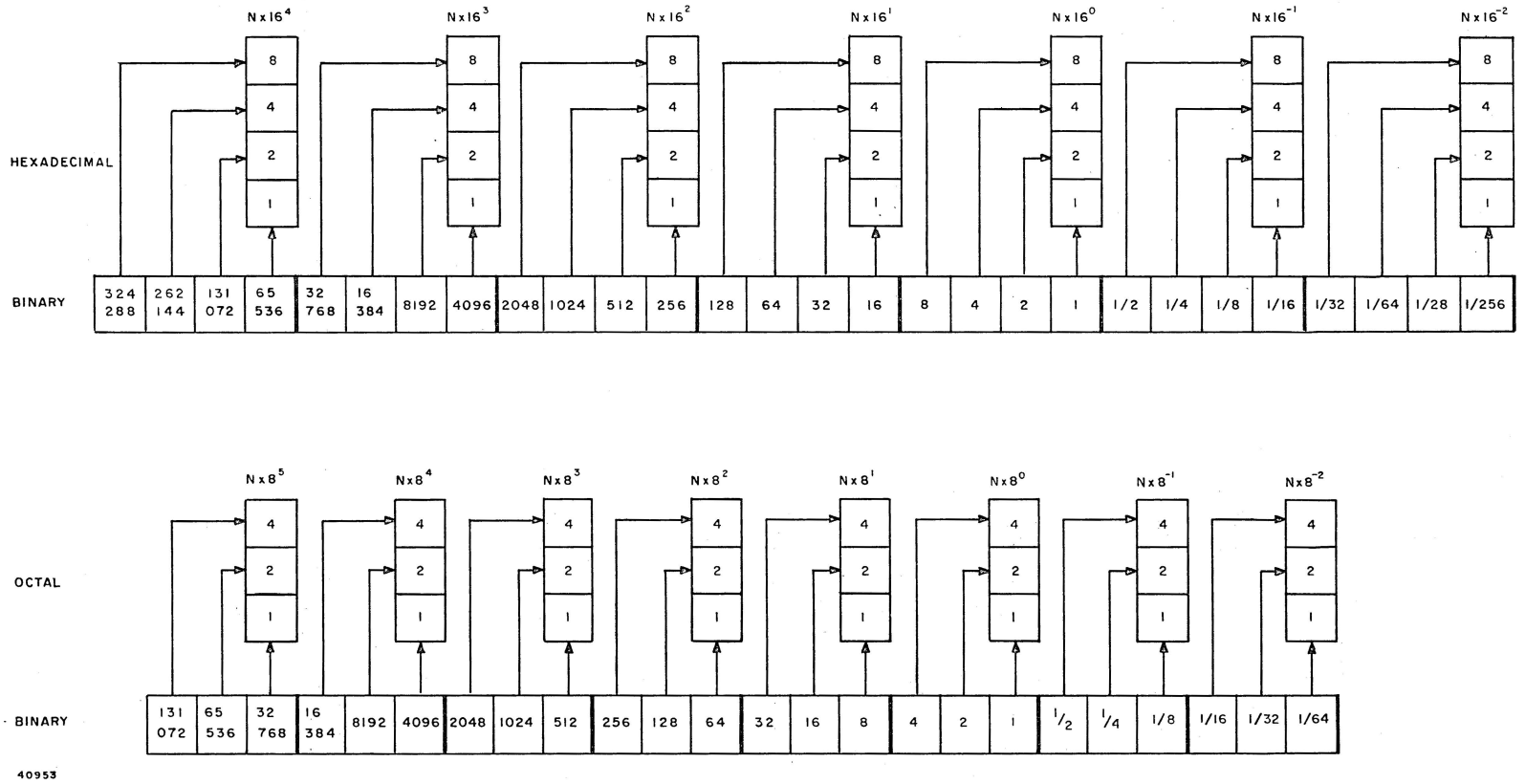


Figure II-1-4. Binary to Hexadecimal and Octal Conversion

binary point is reached. The result is complete when the bit to the immediate right of the binary point has been added into the unit's position and the result divided by 2. This process is shown in figure II-1-6B.

Decimal to Binary Conversion

INTEGRAL

This may be effected in several ways. If the powers of 2 are known, then the binary equivalent can be

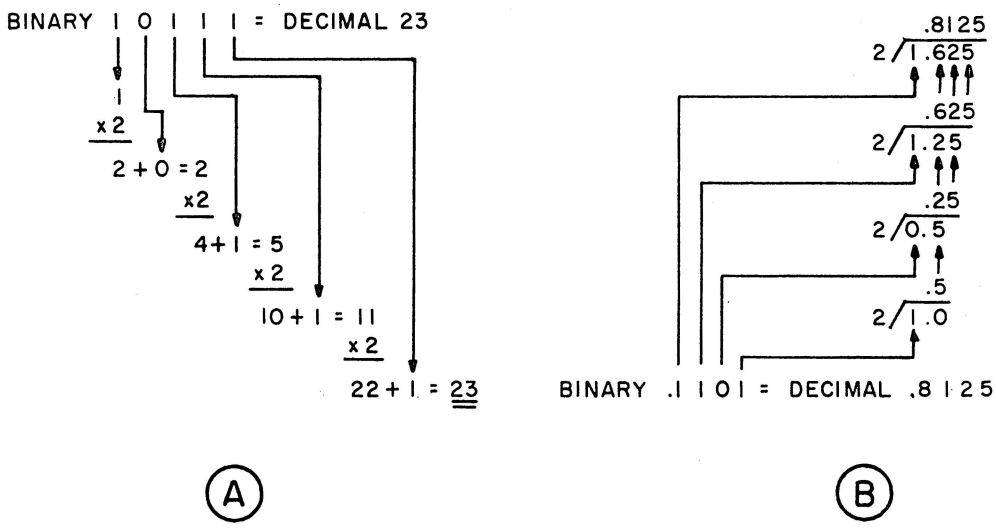
$1765_8 = 1 \times 8^3 + 7 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 =$
$1 \times 512 + 7 \times 64 + 6 \times 8 + 5 \times 1 =$
$512 + 448 + 48 + 5 = 1013_{10}$

$1013_{10} = 1 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 3 \times 1^0 =$
$1 \times 1000 + 0 \times 100 + 1 \times 10 + 3 \times 1 =$
$1000 + 0 + 10 + 3 = 1013_{10}$

$3F5_{16} = 0 \times 16^3 + 3 \times 16^2 + F \times 16^1 + 5 \times 16^0 =$
$0 \times 4096 + 3 \times 256 + F \times 16 + 5 \times 1 =$
$0 + 768 + 240 + 5 = 1013_{10}$

40954

Figure II-1-5. Relationship of Octal, Decimal, and Hexadecimal Numbers



40955

Figure II-1-6. Binary to Decimal Conversion

found by subtracting from the number the largest power of 2, which is smaller than the decimal number, and then recording a bit for that power of two. The largest power of 2, which is smaller than the result of the preceding subtraction, is then found, subtracted, and the corresponding binary bit recorded. In effect, this is the reverse of the first method of converting from binary to decimal.

A second method of conversion is accomplished by successive division. The decimal number to be converted is divided by 2 and the quotient and remainder are noted. The remainder will always be either 0 or 1. Then the quotient is divided by 2, resulting in another quotient and remainder. This is repeated until the quotient is 0. The remainder, resulting from the first division, is the low order bit; and the last remainder is the high order bit. This process is valid for the integral part of a number (figure II-1-7A).

FRACTIONAL (< 1)

The fractional part of a number may be converted in a method somewhat similar to the preceding method of division. The fraction is multiplied by 2 and, if the result is greater than 1, the 1 is recorded in the binary string as a 1 bit. If the product remains less than 1, the binary bit is 0. The fractional part of the product is carried down and again multiplied by 2. This is repeated until the fractional part is equal to 0, or the required degree of accuracy is attained. This process is shown in figure II-1-7B.

Decimal to Octal Conversion

INTEGRAL

When it is desirable to convert a decimal number to its octal form, the powers of eight may be used. Another method is to divide the number by eight. The remainder is the low-order octal digit. The quotient is then again divided by eight, and the remainder resulting is the next higher-order octal digit. This process is repeated until the quotient is zero. This method is used for the integral part of numbers (figure II-1-8A).

FRACTIONAL

When a fractional part of the number is to be converted, multiplication is used. Here the fraction is multiplied by eight and the integral portion formed is the first octal digit to the right of the octal point. This process is repeated until either the fraction is zero, or the desired degree of accuracy is attained. This is shown in figure II-1-8B.

Octal to Decimal Conversion

OCTADE

In octal to decimal or decimal to octal conversions, if the powers of 8 are known, then the procedure is much the same as the corresponding subtraction method of binary. The difference is the digital multiplier which will have a value of from 0 through 7 in octal. Each octal digit will be referred to as an octade. The values of the octades are shown figure II-1-9.

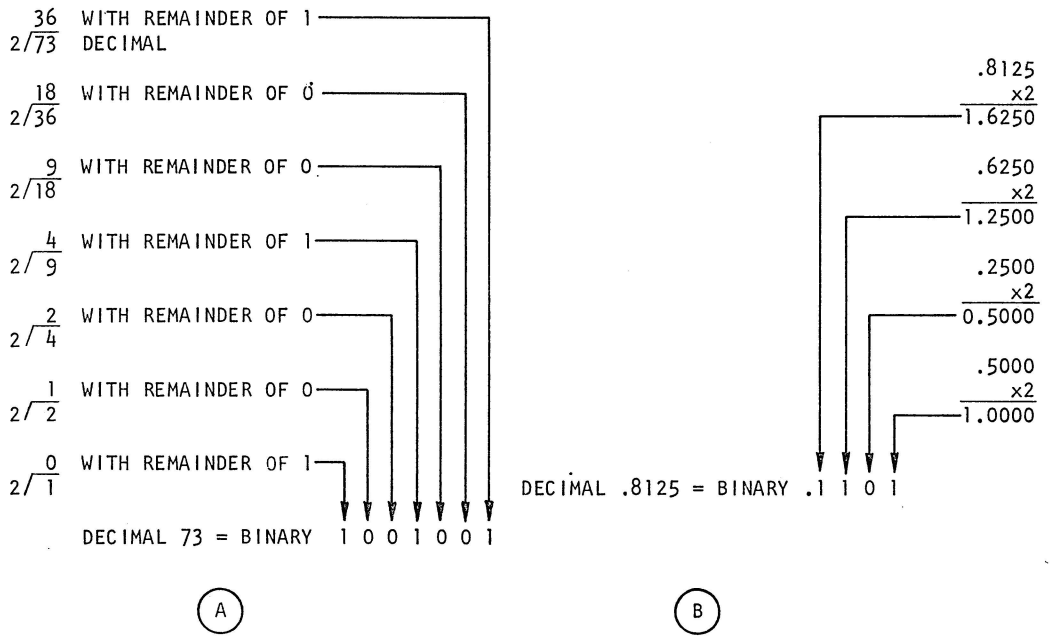


Figure II-1-7. Decimal to Binary Conversion

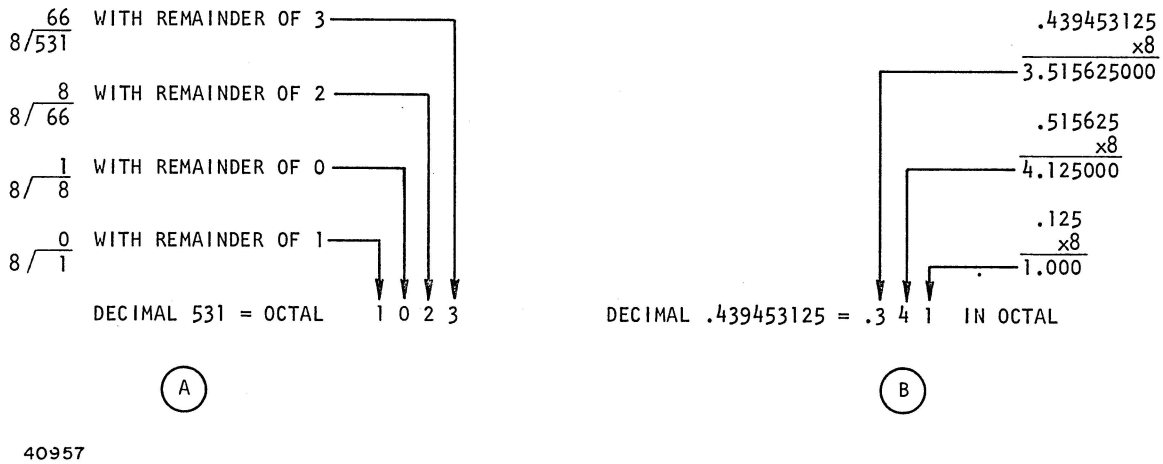


Figure II-1-8. Binary to Octal Conversion

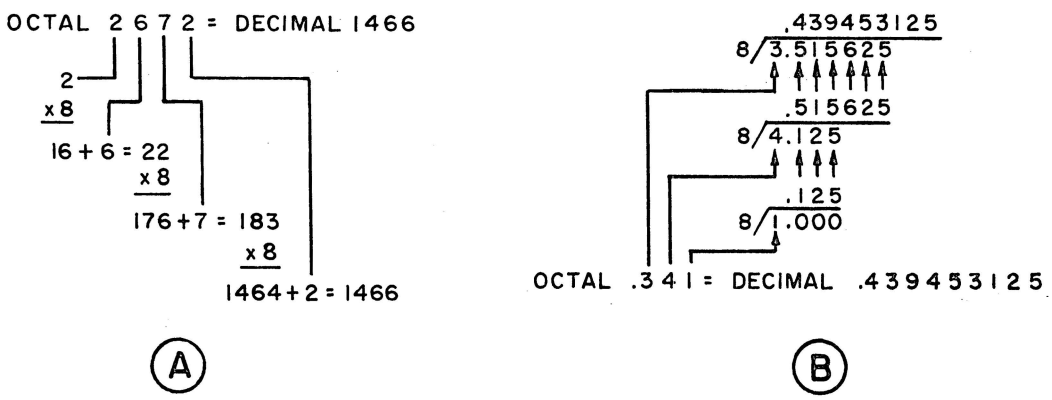
INTEGRAL

On the conversion from octal to decimal, a method very similar to "double dabble" may be used. Here the higher-order octade is multiplied by 8 and then added to the next lower octade. This sum is then multiplied by 8 and again added to the next lower octade. This is continued until the first octade to the left of the octal point is reached. After the unit's octade has been added, the result should be complete (figure II-1-10A).

8^n	n	8^{-n}
1	0	1.0
8	1	0.125
64	2	0.015625
512	3	0.001953125
4096	4	0.000244140625
32768	5	0.000030517578125
262144	6	0.000003814697265625
2097152	7	0.0000004768371582031 25
16777216	8	0.000000059604644775390625
134217728	9	0.000000007450580596923828125
1073741824	10	0.000000000931322574615478515625
8589934592	11	0.000000000116415321826934814453125
68719476736	12	0.000000000014551915228366851806640625
549755813888	13	0.000000000001818989493545856475830078125

40958

Figure II-1-9. Powers of 8



40959

Figure II-1-10. Octal to Decimal Conversion

FRACTIONAL

The above is valid for the integral part of a number, but for the fractional part of a number, the following must be used. The lowest order octade is considered to be an integer. As such, it is divided by 8. The next higher octade is then added to this quotient in the unit's position and the sum again divided by 8. This continues until the first octade to the right of the octal point has been added and the result divided by 8. See figure II-1-10B which is the implementation of:

$$\frac{\frac{N3}{8} + N2}{8} + N1 \over 8 = \text{decimal equivalent fraction}$$

Decimal to Hexadecimal Conversion

When it is desirable to convert an integral or a fractional decimal number to its hexadecimal form, the powers of 16 may be used. Methods similar to those used for conversion to octal representation may also be used, with the multiplication or division being by 16 rather than eight; however, such methods are very cumbersome. The simplest method is to convert the decimal number to a binary number as described earlier, and then convert the binary number to its hexadecimal representation (each four binary digits are used to form one hexadecimal digit).

Hexadecimal to Decimal Conversion

The simplest method for converting integral or fractional hexadecimal numbers to their decimal equivalent is to first convert the hexadecimal number to its binary equivalent (each hexadecimal digit is used to form four binary digits) and then convert the resultant binary number to its decimal representation as described earlier.

OPERAND FORMATS

Operands are the words of information that are worked with when processing. B 7700 data words are normally referred to as operands. An operand may be used to store numeric values (a numeric operand), logical values (a logical operand), or character values (a string operand). Most operands are one word in length, and are identified by a tag field of zero. Double precision operands, which are used to store numbers in which many significant digits of accuracy are needed, are two words in length and are identified by a tag field of two. Thus, the tag field of an operand indicates the size of the operand (one or two words).

Numeric Operands

Numeric operands are used to store numeric values (numbers) in floating point format. A numeric operand may be single or double precision.

When the tag bits of a memory word (bits 50, 49, 48) are 0 (000), they denote a single-precision operand. When the tag bits are 2 (010), i.e., bit 49 set, they denote a double precision operand.

SINGLE PRECISION OPERANDS

All numeric operands are expressed in floating point form, where each numeric operand has both a mantissa and an exponent. This form may be related to power of ten notation where 13297. is the mantissa and

-3, the exponent in a representation of the number 13.297 (13297. x 10^{-3}). The mantissa of a single precision operand is comprised of 39 bits which make up 13 octades. The mantissa of a single precision numeric operand is considered to be an integer and is treated as such; i.e., the binary point is considered to be to the right of the least significant octade. The exponent of the number is represented by 6 bits (bits 43 through 39) which form two octades. Bit number 45 is the sign of exponent. When bit 45 is off, the exponent is positive; when on, negative.

The structure of a single precision operand is shown in figure II-1-11. Because the exponent is an octal scale factor, the single precision operand is shown in both hexadecimal and octal representation.

Exponent Field

The exponent is a binary number which, with its sign, is an octal scale factor for the mantissa. That is, the binary point in the mantissa must be shifted left three binary places (the mantissa must be shifted right three binary places) for each increase by one in the value of the exponent. The exponent is used for automatic scaling of operands when arithmetic, comparison and integer operations are being performed. The range of the exponent is from +63 to -63 for single-precision operands.

Mantissa Field

The mantissa is the significant part of the operand. The magnitude of the operand is obtained by multiplying the value contained in the mantissa by eight raised to the value of the exponent sign and exponent as follows:

$$V = \pm M \times 8^{\pm E}$$

V = Value of number
±M = Mantissa with sign
±E = Exponent with sign

The order of number magnitude in the 39 bit mantissa, as decimal numbers and powers of base 16, 8, and 2 is shown in figure II-1-12.

DOUBLE PRECISION OPERANDS

Double precision operands are identified by a tag field of two, indicating that the operand is one of a pair of two words (figure II-1-13).

The first word of the double precision operand is identical to the single precision operand.

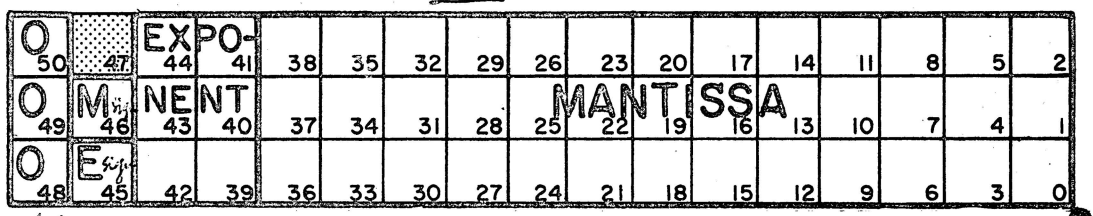
The integral part of the mantissa is contained in the mantissa field of the first word. The fractional part of the mantissa is contained in the mantissa extension field of the second word.

The 15-bit exponent of a double precision operand is formed by the concatenation of the exponent extension with the exponent. The exponent extension is more significant than the exponent.

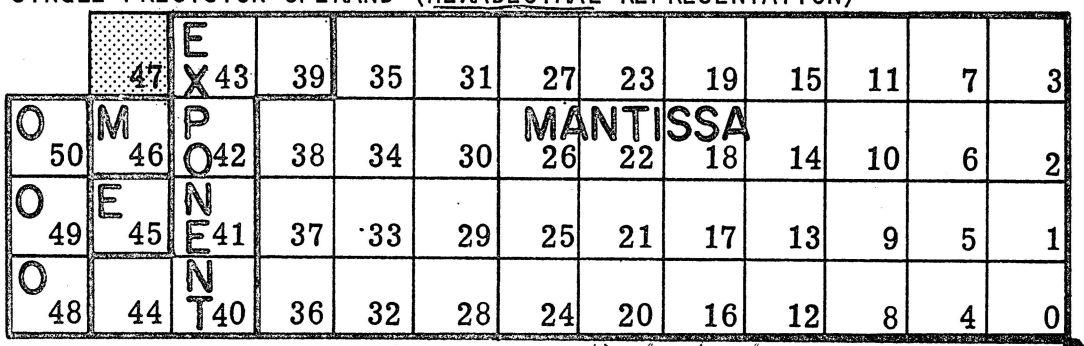
NUMBER RANGES AND NORMALIZATION

In order to add and subtract two numeric operands on the B 7700, the exponents of the two operands must be equal. The B 7700 equalizes the exponents of the two operands automatically; this equalization may require that one of the operands be "normalized". Normalization occurs if the exponent difference of the two operands is greater than the

SINGLE PRECISION OPERAND (OCTAL REPRESENTATION)



SINGLE PRECISION OPERAND (HEXADECIMAL REPRESENTATION)



TAG	50:13	000
	47:1	Not used
M	46:1	Sign of Mantissa.
		1 = Negative, 0 = Positive.
E	45:1	Sign of exponent.
		1 = Negative, 0 = Positive.
EXPONENT	44:6	Exponent.
MANTISSA	38:39	Mantissa.

40960

Figure II-1-11. Single Precision Operand

REGISTER BIT SET	DECIMAL	DECIMAL RECIPROCAL	HEX.	OCTAL	BINARY
0	1	1.0	16^0	8^0	2^0
1	2	0.5			
2	4	0.25			
3	8	0.125		8^1	2^3
4	16	0.0625	16^1		
5	32	0.03125			
6	64	0.015625		8^2	2^6
7	128	0.0078125			
8	256	0.00390625	16^2		
9	512	0.001953125		8^3	2^9
10	1024	0.0009765625			
11	2048	0.00048828125			
12	4096	0.000244140625	16^3	8^4	2^{12}
13	8192	0.0001220703125			
14	16384	0.00006103515625			
15	32768	0.000030517578125		8^5	2^{15}
16	65536	0.0000152587890625	16^4		
17	131072	0.00000762939453125			
18	262144	0.000003814697265625		8^6	2^{18}
19	524288	0.0000019073486328125			
20	1048576	0.00000095367431640625	16^5		
21	2097152	0.00000047683715820312 5		8^7	2^{21}
22	4194304	0.00000023841857910156 25			
23	8388608	0.00000011920928955078125			
24	16777216	0.00000005960464477539062 5	16^6	8^8	2^{24}
25	33554432	0.00000002980232238769531 25			
26	67108864	0.00000001490116119384765625			
27	134217728	0.00000000745058059692382812 5		8^9	2^{27}
28	268435456	0.00000000372529029846191406 25	16^7		
29	536870912	0.00000000186264514923095703125			
30	1073741824	0.00000000093132257461547851562 5		8^{10}	2^{30}
31	2147483648	0.00000000046566128730773925781 25			
32	4294967296	0.00000000023283064365386962890625	16^8		
33	8589934592	0.00000000011641532182693481445 3125		8^{11}	2^{33}
34	17179869184	0.00000000005820766091346740722656 25			
35	34359738368	0.00000000002910383045673703613281 25			
36	68719476736	0.00000000001455191522836685180664 0625	16^9	8^{12}	2^{36}
37	137438953472	0.00000000000727595761418342590332 03125			
38	274877906944	0.00000000000363797880709171295166 015625			
*	549755813887				
39	549755813888	0.00000000000181898940354585647583 0078125		8^{13}	2^{39}

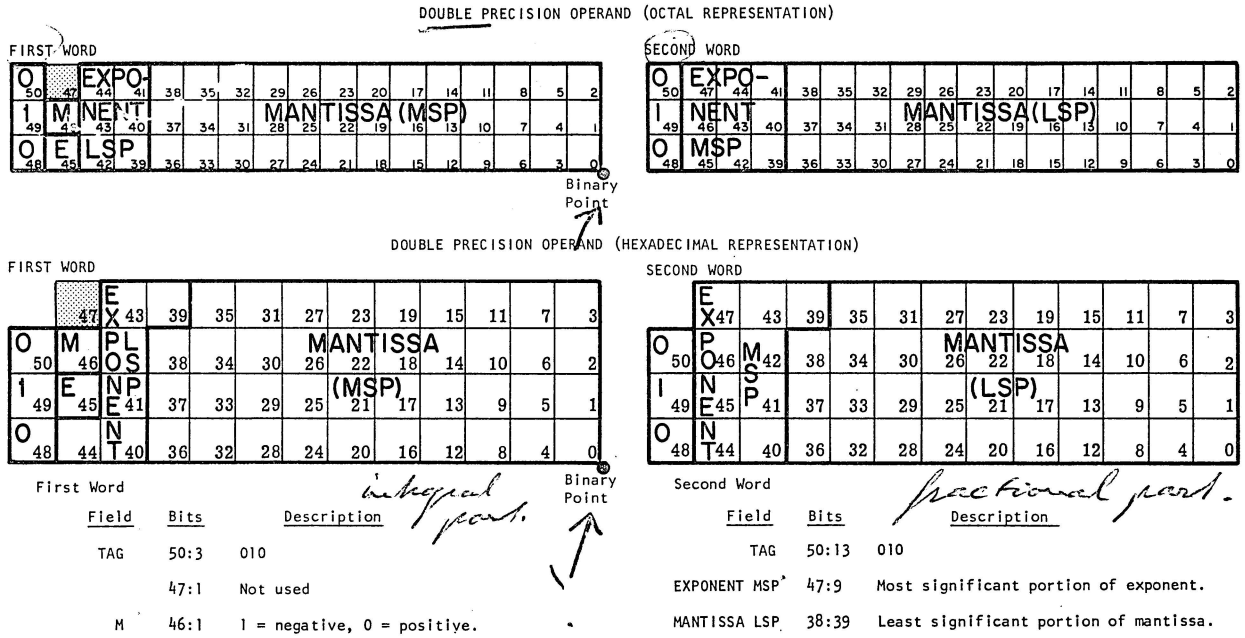
* FIRST 39 BITS SET. (MAXIMUM INTEGER VALUE ALLOWED).

40961

Figure II-1-12. Order of Magnitude Chart

number of leading zero (octal) digits in the mantissa of the operand with the larger exponent. In such cases, the larger operand is normalized, and the mantissa of the smaller operand is then shifted right until the exponents are equal.

A normalized number is a number which has the smallest exponent with which the number can be expressed without losing the most significant digit of the number. A number is normalized by shifting the mantissa to the left, (moving the binary



First Word

Field	Bits	Description
TAG	50:3	010
	47:1	Not used
M	46:1	1 = negative, 0 = positive.
E	45:1	Sign of exponent. 1 = negative, 0 = positive.
EXPONENT LSP	44:6	Least significant portion of exponent.
MANTISSA MSP	38:39	Most significant portion of mantissa.

Second Word

Field	Bits	Description
TAG	50:13	010
EXPONENT MSP	47:9	Most significant portion of exponent.
MANTISSA LSP	38:39	Least significant portion of mantissa.

40962

Figure II-1-13. Double Precision Operand

point right) in three-bit increments until the number of leading zeroes in the mantissa is less than three. For each three-bit shift to the left (of the mantissa), the exponent is decreased by one.

Because of the use of normalization, the range of numbers which are useable on the B 7700 includes both normalized and unnormalized numbers. In general, one may think of normalized numbers as being those which the system may use for arithmetic, and, in general, one may think of unnormalized numbers as being those which the system may store.

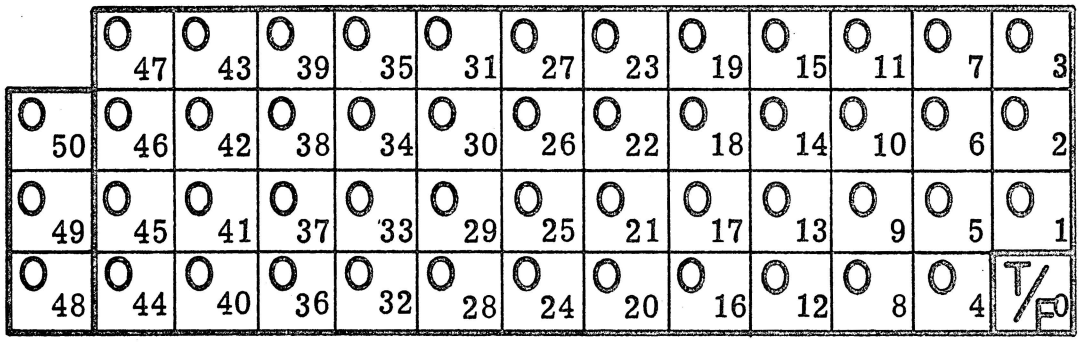
Logical Operands

Logical operands (figure II-1-14) have one of two values: true (on) or false (off). Logical values are the result of Boolean operations or relational operations. Relational operators generate a logical value as the result of an algebraic comparison of two arithmetic expressions. Bit 0 contains the logical value. Relational operators set bit 0, where conditional operators use bit 0 for the decision.

The largest and smallest numbers representable in normalized and unnormalized numbers are:

The largest single precision integer	549755813887_{10}	decimal
or	$8^{13}-1$	
	0007777777777777	octal
The largest single precision number	4.31359146673^{68}	decimal
or	$(8^{13}-1) \times 8^{63}$	
	0777777777777777	octal
The largest double precision integer	$3022314549036572936765 43$	decimal
or	$8^{26}-1$	
(first word)	0157777777777777	octal
(second word)	0007777777777777	
The largest double precision number	$1.9488283820502807911244629603$	decimal
or	$(1-8^{-26}) \times 8^{32780}$	
(first word)	0777777777777777	octal
(second word)	7777777777777777	
The smallest positive unnormalized single precision number	1.27447352891^{-57}	
or	8^{-63}	decimal
	1770000000000001	octal
The smallest positive normalized single precision number	$8.75811540 20^{-47}$	
or	8^{-51}	decimal
	1771000000000000	octal
The smallest positive normalized double precision number	$1.9385458571375858335564 0^{29581}$	
or	8^{-32755}	decimal
(first word)	1771000000000000	octal
(second word)	7770000000000000	

The number sets are symmetrical with respect to zero. The negative number corresponding to any valid positive number may also be expressed. From the ranges above, one can see that a single precision integer must always have an exponent of zero.



Field	Bits	Description
TAG	50:3	000
	47:47	All zeroes.
T/F	0:1	True/false bit.
		1 = True, 0 = False

40963

Figure II-1-14. Logical Operand

NOTE

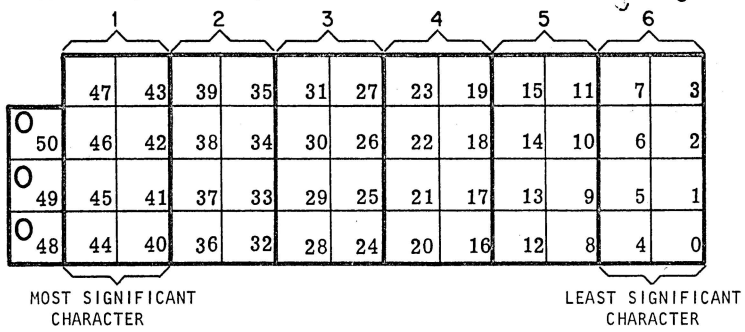
Logical operators (LAND, LOR, LNOT, and LEQV) cause a logical operation to be performed on each bit of the two operands and the results of these operations (48 single precision values or 96 double precision values) are left in the top-of-stack operand. Logical operators may operate on logical, string, or numeric operands.

String Operands

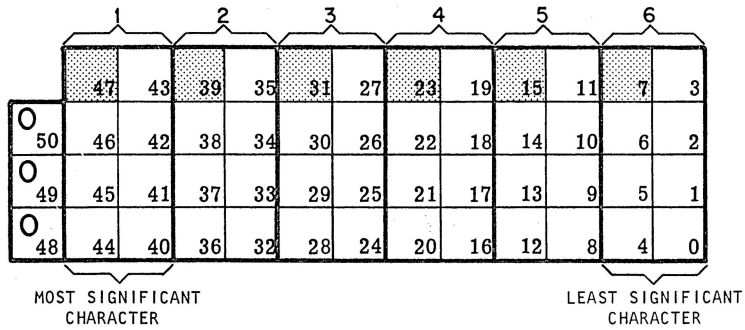
A string operand is a single word operand (identified by a tag of zero) which is used to store characters. Character representation may be 8-bit (EBCDIC), 7-bit (USASCII), 6-bit (BCL), or 4-bit (packed BCD) characters. Generally, a string of characters is stored in one or more string operands in memory as an array or table. Such arrays or tables are addressed by means of string descriptors. The format of string operands for storage of 8-bit, 7-bit, 6-bit, and 4-bit characters is shown in figure II-1-15.

String operands may also be used to store signed numeric characters in 8-bit, 6-bit, and 4-bit formats. Each string operand can store one signed numeric number consisting of six 8-bit characters, eight 6-bit characters, or 11 4-bit characters. Eight-bit and 6-bit characters are divided into a zone portion and a number portion. The number portion consists of the four least signifi-

8-BIT BYTES (EBCDIC CODE)

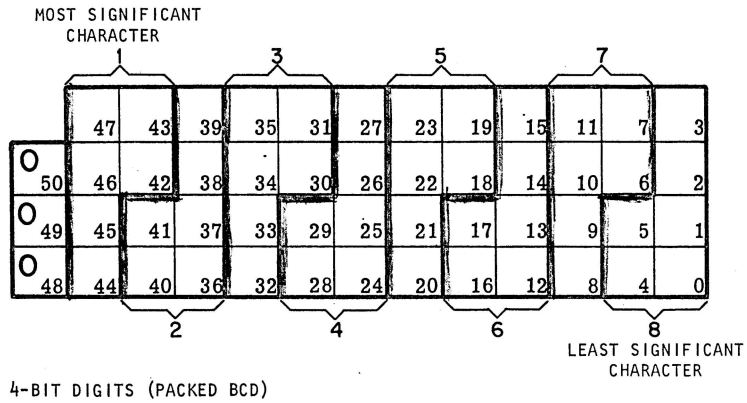


7-BIT CHARACTERS (USASC11 CODE)

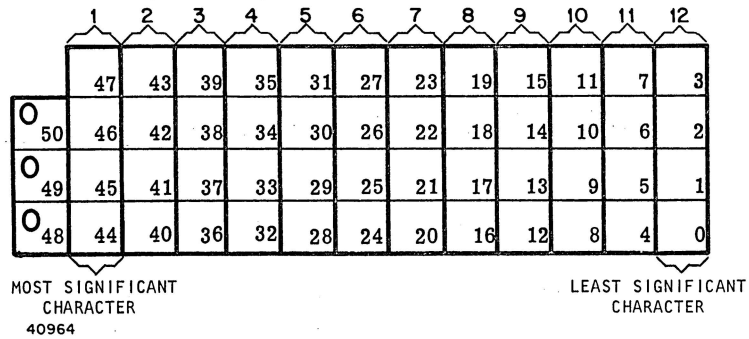


(BITS 47,39,31,23,15,AND 7 ARE NOT USED)

6-BIT CHARACTERS (BCL CODE)



4-BIT DIGITS (PACKED BCD)



40964

Figure II-1-15. String Operands

cant bits of each character; the remaining bits form the zone. When 8-bit or 6-bit signed numeric characters are stored in a string operand, the sign of the characters is stored in the zone bits of the least significant character. When 4-bit signed numeric characters are stored in a string operand, the sign of the characters is stored as the most significant character of the operand. Table II-1-1 shows the bit configurations for negative and positive signs in 8-bit, 6-bit, and 4-bit formats. Figure II-1-16 illustrates the manner in which a signed number (-4259) is stored in 8-bit, 6-bit, and 4-bit code.

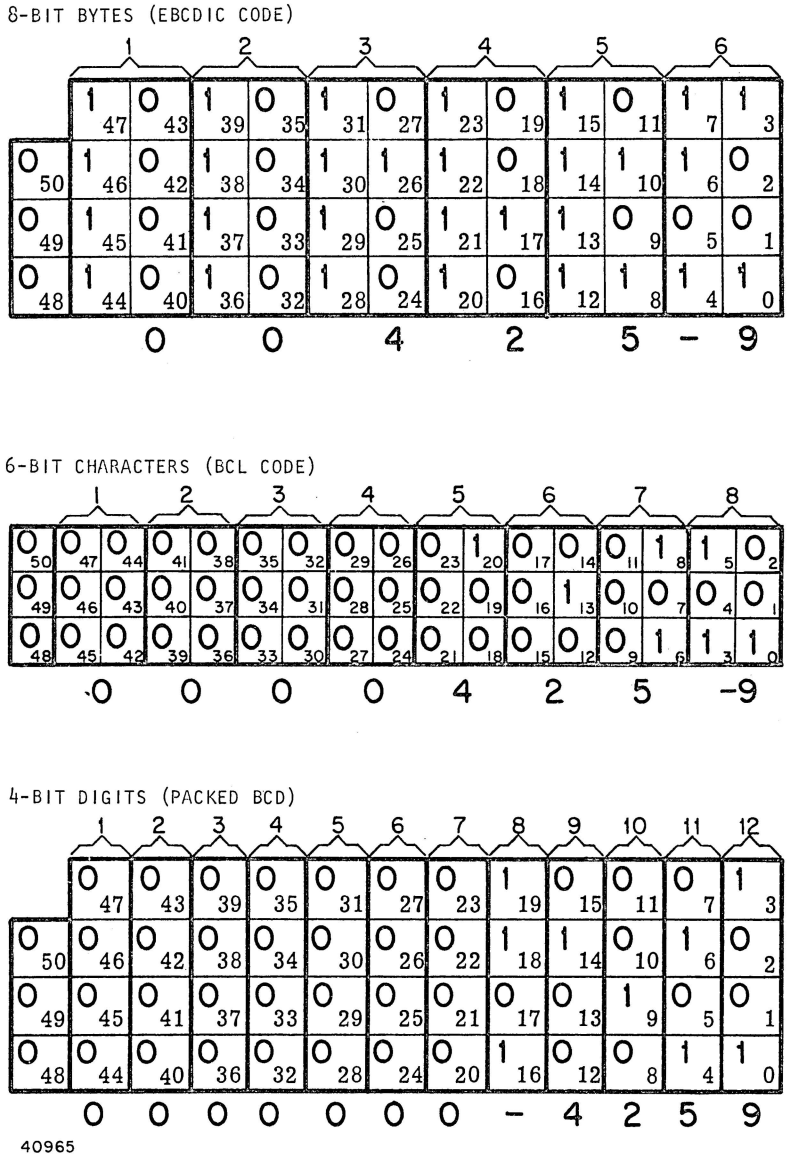


Figure II-1-16. Use of String Operand to Store a Signal Number (-4259)

Table II-1-1. Sign Configurations Of String Operands

Size	Sign Location	Negative	Positive
8-bit	Zone, least significant character	1101	Any bit configuration other than the negative bit configurations
6-bit	Zone, least significant character	10	Any bit configuration other than the negative bit configurations
4-bit	Most significant digit	1101	Any bit configuration other than the negative bit configurations

SECTION 2
POLISH NOTATION AND STACK

GENERAL

To facilitate the understanding of the B 7700 stack concept, a method of mathematical notation known as Polish notation must be understood. A problem that exists with most forms of mathematical notation is clarifying the boundaries of specific terms. This has been eliminated with the use of parentheses, brackets, and braces. However, with a complex equation, it becomes necessary to duplicate the use of the few types of delimiters that exist. It might be noted that it is common to encounter mathematical equations such as Y = 5Z + 7/2Z and Y = (5Z + 7)/2Z. Two equations express different functions of Z, but one could easily be used when the other was intended. From this it can be seen that an error in notation can change the whole problem, because the parentheses have definite meaning.

Polish notation is an arithmetical or logical notational system using only operands and operators arranged in a sequence or string which eliminates the necessity of factor boundaries. The B 7700 compilers translate source statements to Polish strings, and convert these Polish strings to a series of machine instructions (program operators).

POLISH NOTATION

The essential difference between Polish notation and conventional notation is that operators are written to the right of a pair of operands instead of between them. For example, the conventional B + C would be written B C + in Polish notation. Looking at the example, A = 7 (B + C), it could be written as follows:

B C + 7 x A =

Any expression written in Polish notation is called a Polish string. In order to fully understand this concept, the rule for evaluating a Polish string should be known.

General Rules For Generation of Polish String

Figure II-2-1 is a flow chart for generation of a Polish string. In

general, the rules for generation of a Polish string may be stated as follows. If the source of expression is:

Name	Action
A Variable	Place variable in string being built and examine next symbol.
An Operator	
-Separator	Place in delimiter list and examine next symbol.
-Arithmetic or Boolean operator and last entered delimiter list symbol was:	Place operator in the delimiter list and examine next source symbol.
a. an operator of lower priority.	
b. a left bracket "[" or paren "(".	
c. a separator.	
d. nothing (delimiter list empty).	
-An Arithmetic or Boolean operator and last entered delimiter list symbol was: an operator of priority equal to or greater than the symbol in the source	Remove the operator from the delimiter list and place in the string being built. Then compare the next symbol in the delimiter list against the source expression symbol.
-A right bracket "]" or parenthesis ")"	Pull out from delimiter list or until corresponding left bracket or parenthesis.

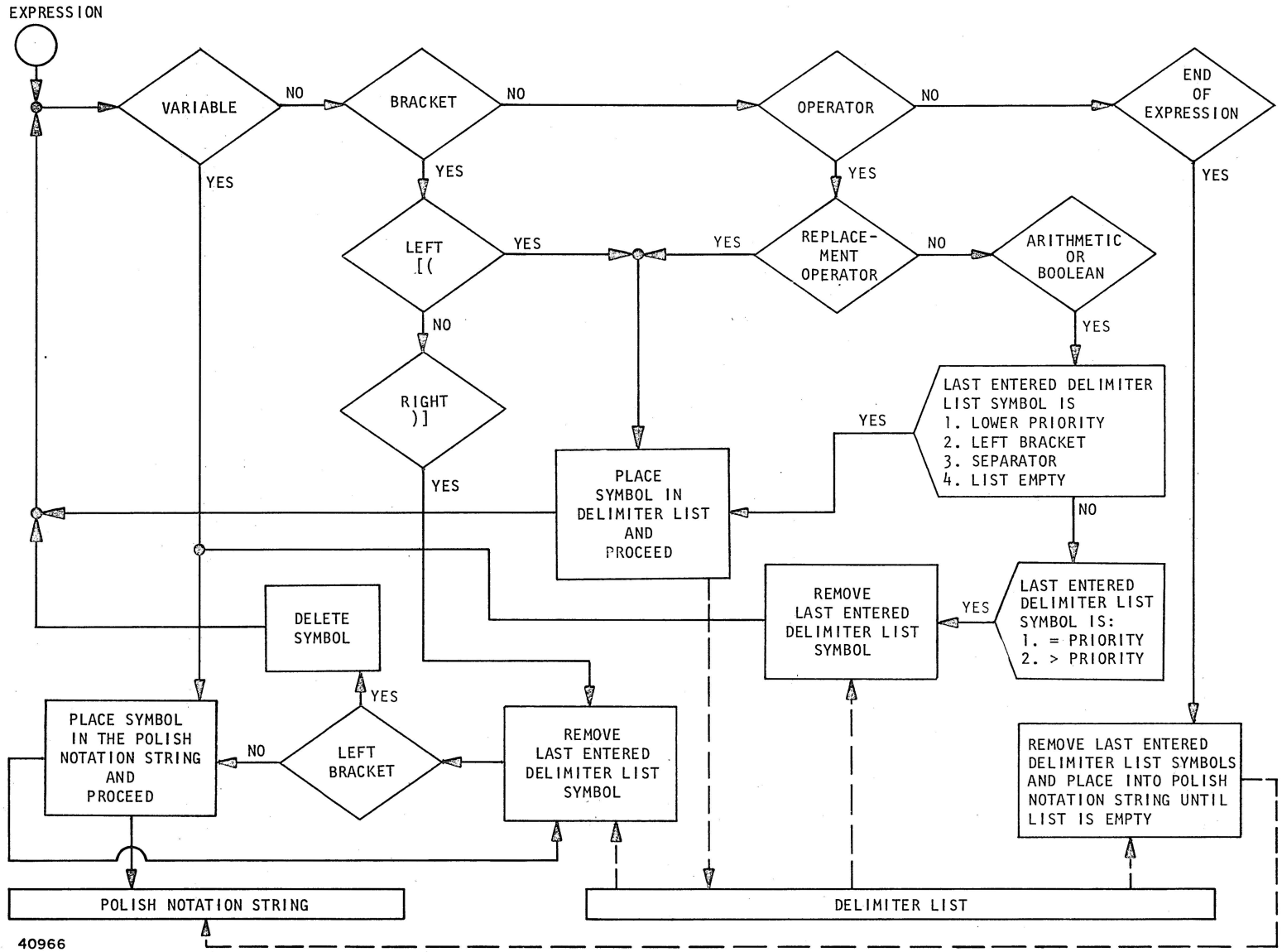


Figure II-2-1. Polish Notation Flow Chart

Evaluating Polish String

The following procedure may be used to evaluate a Polish string.

a. Scan the string from left to right.

b. Remember the operands and the order in which they occur.

c. When an operator is encountered do the following:

1) Take the two operands which are last in order.

2) Operate upon them according to the type of operator encountered.

3) Eliminate these two operands from further consideration.

4) Remember the result of (2) and consider it as the last operand in order.

Following this rule through the Polish string step by step, B C + 7 x A = would result in A assuming the value 7 (B + C) (figure II-2-2).

NOTE

Because replacement operators vary depending upon the language used, ←, =, and := may be used interchangeably in discussing Polish strings.

Step	Symbol Being Examined	Symbol Type	Operands Being Remembered and Their Order of Occurrence (1 or 2) Before Operation	Operation Taking Place	Results of Operation
a	B	Operand			
b	C	Operand	1 B		
c	+	Add Operator	2 C 1 B	B + C	(B + C)
d	7	Operand	1 (B + C)		
e	x	Multiply Operator	2 7 1 (B + C)	7 x (B + C)	7 x (B + C)
f	A	Operand	1 7(B + C)		
g	=	Replace Operator	2A 1 7 (B + C)	A←7(B + C)	A = 7(B + C)

40967

Figure II-2-2. Evaluation of Polish String BC+7xA=

Compilation Using Polish Notation

Polish notation is used as the base for the B 7700 ALGOL compilation algorithm. An ALGOL arithmetic or Boolean expression or assignment statement may be translated to Polish notation in much the same way as the arithmetic (or algebraic) expression that already has been considered. In compiler translation, the source expression is examined one symbol at a time with a left to right scan and is combined into logical entities. As each logical entity is examined, a specific procedure is followed so that the Polish notation expression is constructed in its finalized form with one scan of the source expression.

Program Code String

When the program is compiled, the computational part of the source program will be converted into a machine language string of instructions. An example of this is the source language plus sign (+) which will be directly replaced by the machine language ADD instruction. The machine language string, resembling a Polish notation string, will be referred to as the program code string. This code string will be divided into two or more variable-sized segments, according to the structure of the program. Program segments are normally stored on disk files. When a program is executed, program segments are made present in memory as needed. Because such program segments cannot be modified a single copy of a program segment in memory may be used for several concurrent executions of the same program; thus, the program code string is often described as "re-entrant".

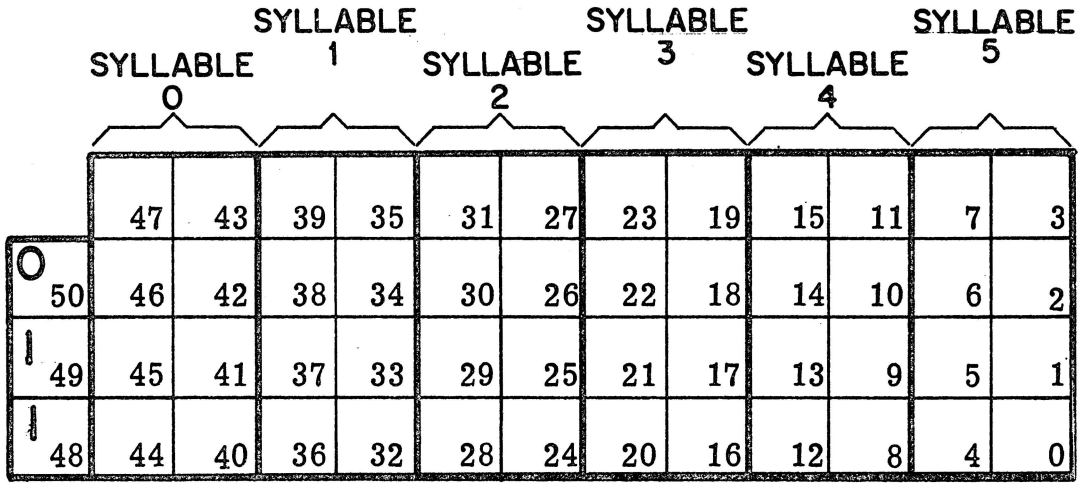
As mentioned earlier, a program code string may be divided into two or more program segments. For each program segment, there is a single segment descriptor, which defines the length and location of the program segment. The segment descriptors are stored in a special stack known as the segment dictionary. Thus, each job is associated not only with one job stack, but also with one segment dictionary stack. (In addition, the MCP has its own stack and segment dictionary.) Within the job stack, a Program Control Word is provided for each point of entry into a segment of code. The PCW provides an index, not only into the segment dictionary to locate the proper segment descriptor, but also into the program segment itself to locate the proper program word and syllable. The formats of the segment descriptor and the PCW are described in detail in section 3 of this chapter.

PROGRAM WORD

Each program word contains six 8-bit syllables. The two most significant bits of a syllable indicate whether a syllable begins a Value Call, Name Call, or other operator. A value in these two bits of 00 indicate a Value Call operator, a value of 01 indicates a Name Call operator. The format of a program word is shown in figure II-2-3.

STACK CONCEPTS

The constants and variables of a program are assigned locations within the "stack" of the program when it is compiled. The stack can be thought of as analogous to a physical stack where the last item placed on the stack is the top of the stack. When items are removed (one



Field	Bits	Description
Tag	50:3	Tag field. Value of three indicates that this word is non-modifiable (except by Overwrite operators).
	47:8	Syllable 0
	39:8	Syllable 1
	31:8	Syllable 2
	23:8	Syllable 3
	15:8	Syllable 4
	7:8	Syllable 5

40968

Figure II-2-3. Program Word

at a time) from the stack, the item on the top of the stack is the first item to be removed. The item at the bottom of the stack remains at the bottom of the stack until all other items have been removed from the stack. The stack not only provides an easily manageable means for keeping a dynamic history of the program as it is being processed, but also lends itself to the use of program code strings based on Polish notation.

General

When a job is activated, two top-of-stack locations (A and B) are linked to the job's stack (figure II-2-4). This linkage is established by the stack-pointer register (S), which contains the memory address of the last word placed in the stack. The two top-of-stack locations (A and B) extend the stack to provide quick access for data manipulation.

Data are brought into the stack through the top-of-stack locations in such a manner that the last operand placed into the stack is the first to be extracted. Total capacity of the top-of-stack locations (A and B) is two operands. Loading a third operand into the top-of-stack locations causes the first operand to be pushed from the top-of-stack locations into the stack. The stack-pointer register (S) is incremented by 1 before a word is placed into the stack and is decremented by 1 after a word is withdrawn from the stack and placed in the Top-of-Stack locations. As a result, the S register continually points to the last word placed into the job's stack.

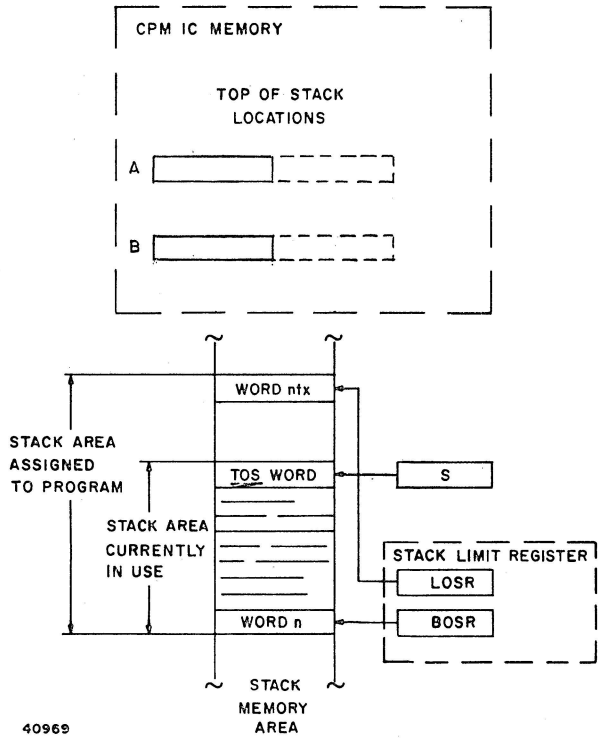


Figure II-2-4. Top of Stack and Stack Bounds Registers

BASE AND LIMIT OF STACK

A job's stack is bounded, for memory protection, by two registers: the Base-of-Stack register (BOSR) and the Limit-of-Stack register (LOSR). The contents of BOSR define the base of the stack, and the contents of LOSR define the upper limit of the stack. The job is interrupted if the S register is set to the value, contained in either LOSR or BOSR.

BI-DIRECTIONAL DATA FLOW IN THE STACK

The contents of the top-of-stack locations are maintained automatically by the processor to meet the requirements of the current operator. If the current operator requires data transfer into the stack, the top-of-stack locations receive the incoming data, and the surplus con-

tents, if any, of the top-of-stack locations, are pushed into the stack. Words are brought out of the stack into the top-of-stack locations. These words are used by operators which require the presence of data in the top-of-stack locations. These operators, however, do not explicitly move data into the stack.

DOUBLE PRECISION STACK OPERATION

Each top-of-stack location (A and B) can accommodate two memory words. For single precision operations, location A will contain one single precision operand and location B will contain the other single precision operand. However, calling a double precision operand into either top-of-stack location (A or B) will cause both halves of the double precision operand to be loaded into the A or B location. The first word is loaded into the top-of-stack location and its tag bits are checked. If the value of the tag bits indicates double precision, the second half of the operand is loaded into the second half of the top-of-stack location. Double precision operands revert to single words when they are pushed down into the stack (the most significant half of the operand is pushed down first). The process is reversed when a double precision operand is returned from the stack to the top-of-stack locations. That is, the least significant half of the double precision operand is popped up first and the tag is discovered to have a value of two, causing the most significant half of the operand to also be popped into the top-of-stack.

Hardware Implementation

The B 7700 stack implementation includes a 32-word stack buffer, which permits a portion of an active stack to be contained in IC memory locations within the CPM. This stack buffer (see figure II-2-5) may contain information which has not yet been written to core memory, as well as copies of words which are resident in core memory. The stack buffer permits a portion of the stack to be held local within the CPM, to provide quick access for stack manipulation by the execution unit of the CPM.

In addition to the portion of the stack held local in the stack buffer, certain other data from the stack may be contained in a local memory within the CPM. This local memory, the associative memory, is used to capture data fetched by program unit look ahead which is not resident in the stack buffer.

Although an active stack may be contained partly in the stack buffer within the CPM and partly in core memory, the stack buffer is purged whenever the stack becomes inactive (when a move-to-stack operation takes place). This purging of the stack buffer causes the unique data within the stack buffer to be copied to core memory. Thus, for practical purposes, this section discusses the stack as if it exists solely within core memory. A detailed description of the stack buffer and the associative memory may be found in chapter III.

DYNAMIC PROGRAM HISTORY

One very important aspect of the B 7700 is the retention of the dynamic history for the program being processed. Two lists of program history are maintained in the B 7700 stack, the addressing environment list and the stack history list.

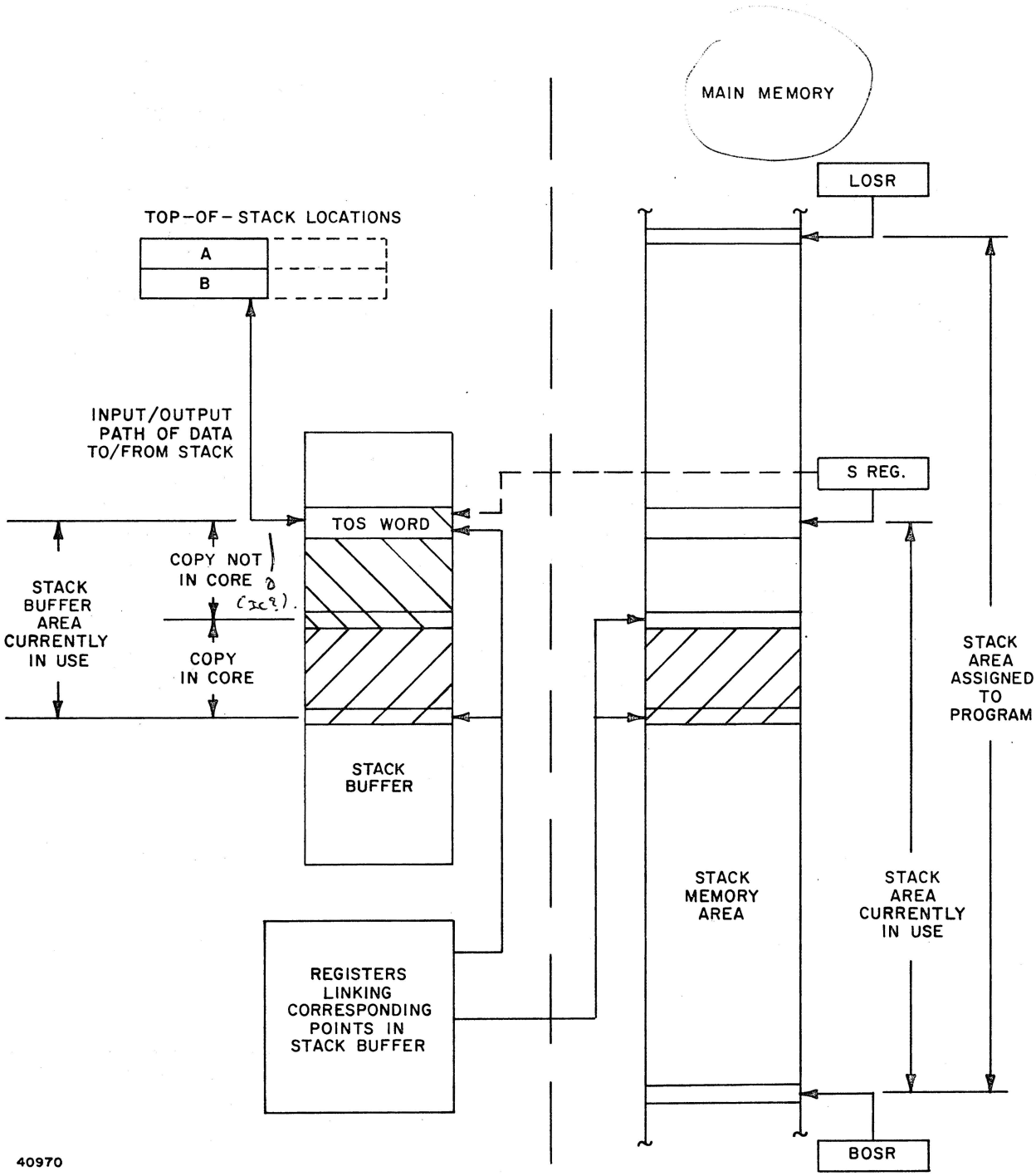


Figure II-2-5. Stack Buffer and Stack Memory Area

Both of these lists are dynamic, varying as the job proceeds along different program paths with varying sets of data. The two lists grow and contract in accordance with the procedural depth of the program. Both of these lists are generated automatically by the B 7700 hardware. Before further stack discussion can be considered, addressing history and stack history must be discussed.

Addressing History

The B 7700 CPM provides two methods for addressing data. Direct addressing is provided by descriptors, which contain the address (core or disk) of the data. Descriptors are used to address data which are located outside of the stack area of the job. Relative addressing is provided by the Indirect Reference Word (IRW) and the Stuffed Indirect Reference Word (SIRW). The IRW and SIRW address components are both relative address components. The IRW addresses within the immediate environment of the job relative to one of 32 CPM display registers. The SIRW addresses beyond the immediate environment of the current procedure, the addressing being relative to the base of the job's stack. Addressing across stacks is accomplished with an SIRW.

DIRECT ADDRESSING

In general, the descriptor describes and locates data associated with a given job. String descriptors and data descriptors are used to fetch data to the stack or to store data from the stack into an array located outside the stack area of the job. The address contained in one of these descriptors is the absolute address of an array in either system main memory or in the backup disk file, as indicated by the setting of a single bit called the presence bit. Another bit, called the double-precision bit, is used to identify the referenced data as single precision or double precision. The formats of string and data descriptors, and detailed discussions of each, are presented in section 3 of this chapter.

RELATIVE-ADDRESSING

Analyzing the structure of an ALGOL program results in a better understanding of the relative-addressing procedures used in the B 7700 stack. The addressing environment of an ALGOL procedure is established automatically as the program is structured by the programmer and is referred to as the lexicographical ordering of the procedural blocks. At compile time, the lexicographical ordering is used to form address couples. An address couple consists of two items:

1. The lexicographical addressing level (LL) of the variable,

2. An index value (I) used to locate the specific variable within its addressing level.

The lexicographical ordering of the program remains static as the program is executed, thereby allowing variables to be referenced via address couples as the program is executed.

The lexicographical structure of a very simple ALGOL program is illustrated in figure II-2-6. When executed, this program would call procedure C (LL=3) from the outer block of the program (LL=2), and, in turn, procedure C would call proce-

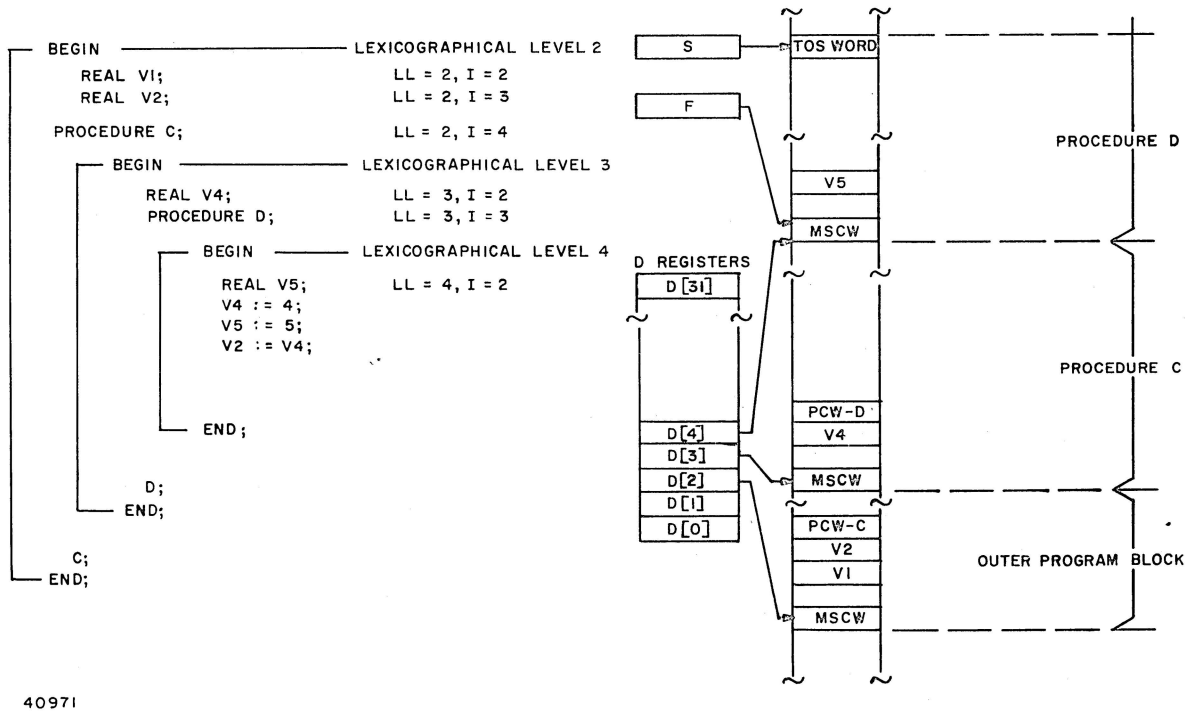


Figure II-2-6. ALGOL Program With Lexicographical Structure and Related Stack Structure

dure D (LL=4). The stack structure is illustrated as it would exist as procedure D was being executed. It can be seen that, as the outer block of the program was entered, and again as each procedure was entered, a Mark Stack Control Word (MSCW) was placed in the stack. The MSCW (described in detail in section 3 of this chapter) denotes the base of each lexicographical addressing level.

Display Registers

Each MSCW provides a point in the stack relative to which the variables for the associated addressing level may be referenced. The B 7700 CPM contains 32 display registers (D[0] through D[31]). As shown, the base of each addressing level is addressed by one of these registers. The local variables of the outer block or of the procedures are addressed relative to the D registers. The D registers are updated at each procedure entry or exit.

Absolute Address Conversion

Each variable is indirectly addressed by an address couple containing a lexicographical level and an index value. The address couple is converted into an absolute memory address when the variable is referenced. The lexicographical level portion of the address couple selects the D register which contains the absolute memory address of the MSCW for the environment (lexicographical level) in which

the variable is located. The index value of the address couple is added to the contents of the D register to generate the absolute memory address of the desired variable.

Addressing Environment

Thus far we have considered a very simple program in which each procedure has a different lexicographical addressing level. Generally, however, many procedures of a program may have the same lexicographical addressing level; however, no two procedures of a program may have the same addressing environment. Consider the more advanced exemplary program shown in figure II-2-7.

This program consists of an outer block (LL=2), two procedures which have a lexicographical addressing level of three (procedures A and C), and two procedures which have a lexicographical level of four (procedures B and D). The addressing environment of the program is maintained automatically by linking the MSCWs together in accordance with the lexicographical structure of the program. This linkage is composed of the stack number (STACK NO.) and displacement (DISP) fields of the MSCW, and is inserted into the MSCW when the procedure is entered. A tree-structured addressing environment list is formed by linking the MSCW to the MSCW at the preceding lexicographical level to the procedure being entered. This tree-structured list indicates the addressing environment of the procedures.

Comparing the addressing tree in figure II-2-8 with the exemplary program, one can see that when procedure B is being executed, the addressing environment includes only the variables in procedures B and A and the outer block; variables declared in procedure C and D are not addressable by procedure B. Thus, one can see that the address couples assigned to the variables in a program need not be unique. This is true because if there is no procedure which can address both of any two variables, then the two variables may have identical address couples. This addressing scheme is practical because two variables which have the same address couples will be contained within two different addressing environments.

Addressing Environment List

There is a unique set of MSCWs which the D registers must address during the execution of any particular procedure. The D registers must be changed, upon procedure entry or exit, to address the correct MSCWs. The process of changing the D registers is referred to as display update. The list of MSCWs which the D registers address is the addressing environment list, and the areas of the stack which can be addressed relative to the settings of the D registers are the addressing environment.

Stack History

The B 7700 stack provides an easily manageable means for keeping line control information (program history) necessary for procedure entry and exit. The stack history list is a list of Mark Stack Control Words, linked together by their DF fields (figure II-2-9).

An MSCW is inserted into the stack as a procedure is entered and is removed as that procedure is exited. Therefore, the stack history list grows and contracts with the procedural depth of the program. Mark

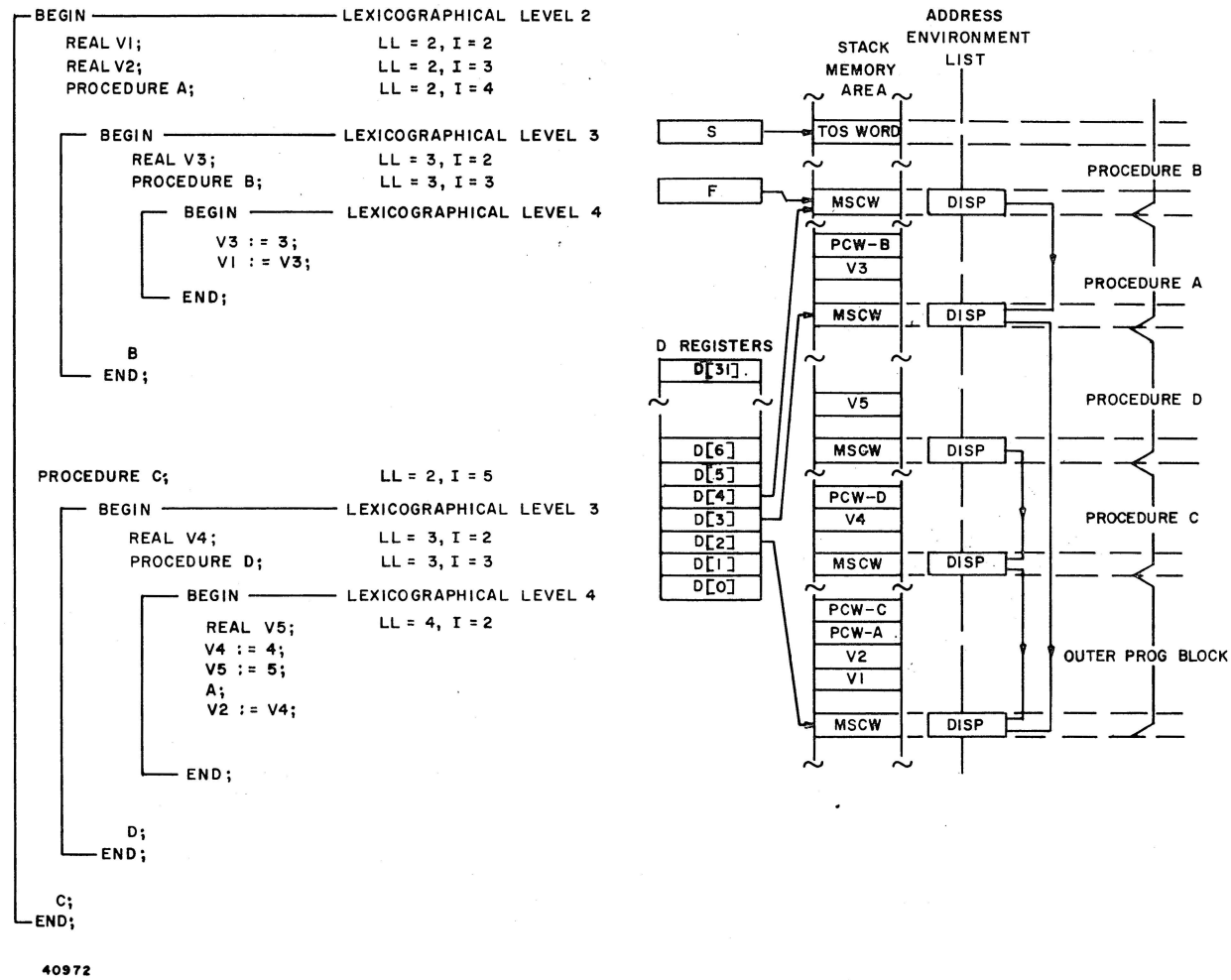


Figure II-2-7. More Advanced ALGOL Program

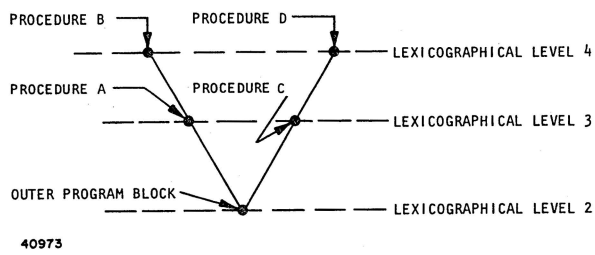


Figure II-2-8. Addressing Environment Tree of ALGOL Program

Stack Control Words identify the portion of the stack related to each procedure. When the procedure is entered, its parameters and local variables are entered in the stack following the MSCW. When the procedure is executed its parameters and local variables are referenced by addressing relative to the MSCW.

Each MSCW is linked to the prior MSCW through the contents of its DF field in order to identify the point in the stack where the prior procedure began. When a procedure is ex-

ited, its portion of the stack is discarded. This action is achieved by setting the stack-pointer register (S) to address the memory location preceding the most recent MSCW (figure II-2-10). This topmost MSCW, addressed by another register (F), is deleted from the stack-history list by changing F to address the prior MSCW, placing this MSCW at the head of the stack history.

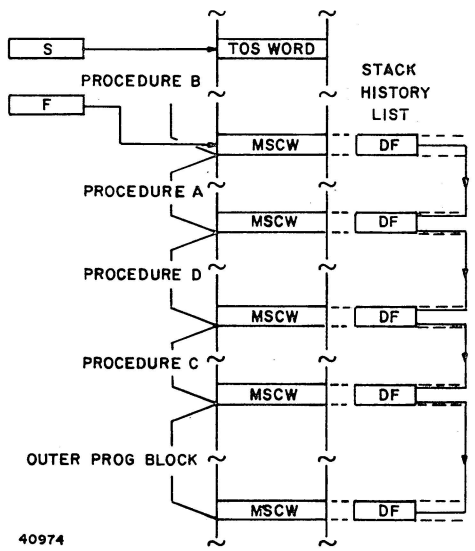


Figure II-2-9. Stack History List

SIMPLE STACK OPERATION

All program information must be in the system before it can be used. Input areas are allocated for information entering the system and output areas are set aside for information exiting the system; array and table areas are also allocated to store certain types of data. Thus data is stored in several different areas: the input/output areas, data tables (arrays), and the stack. Since all work is done in the top-of-stack locations, all information or data is transferred to the top-of-stack locations and the stack itself.

At this point, an ALGOL assignment statement and the Polish notation equivalent will be related to the stack concept of operation. The example is Z:=Y + 2x(W+V), where := means "is replaced by." In terms of a computer program, this assignment statement indicates that the value resulting from the evaluation of the arithmetic expression is to be stored in the location representing the variable Z.

When Z:=Y + 2x(W+V) is translated to Polish notation, the result is ZY2WV+ x +:=. Each element of the example expression causes a certain type of syllable to be included in the machine language program when the source problem is compiled. The following is a detailed description of each element of the example, the type of syllable compiled, and the resulting operation (see figure II-2-11 and table II-2-1).

In the example statement, Z is to be the recipient of a value, the address of Z must be placed into the stack just prior to the store command. This is accomplished by a Name Call (NAMC) syllable which places an Indirect Reference Word (IRW) in the stack. The IRW contains the address of Z in the form of an "address couple" that references the memory location reserved in the stack for the variable Z.

Since Y is to be added to a quantity, Y is brought into the top of the stack as an operand. This is accomplished with a Value Call (VALC) syllable that references Y. The value 2 is then brought to the

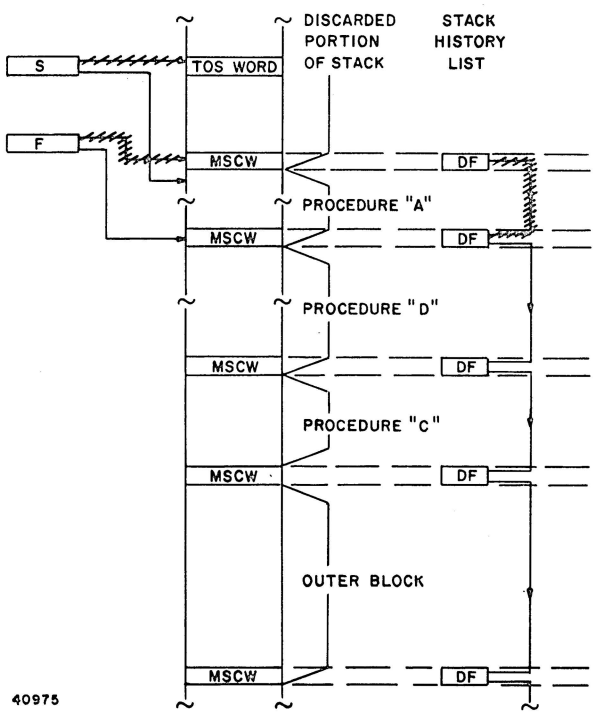


Figure II-2-10. Stack Cut Back on Procedure Exit

stack, with an eight-bit literal syllable (LT8). Since W and V are to be added, the respective variables are brought to the stack with Value Call syllables. The ADD operator adds the two top operands and places the sum in the top of stack. This example assumes, for simplicity, single-precision operands not requiring use of additional top-of-stack locations which are used in double-precision operations.

The multiply operator (MULT) is the next symbol encountered in the Polish string; when executed, it places the product "2x(W+V)" in the top of the stack. The next symbol, ADD, when executed, leaves the final result "Y+2x(W+V)" in the top of the stack.

The store syllable (STOD) completes the execution of the statement Z:=Y + 2x(W+V). The store operation examines the two top-of-stack operands looking for an IRW or Data Descriptor. In this example, the IRW addresses the location where the computed value of Z is to be stored. The stack is empty at the completion of this statement.

Thus, the Polish string ZY2WV + x+:= is used to produce the following code string:

```
NAMC  (Z)
VALC  (Y)
LT8   2
VALC  (W)
ADD
MULT
ADD
STOD
```

When this code string is executed on the B 7700, the value of the expression Y+2x(W+V) is stored in the stack location reserved for the variable Z.

INTERRUPT HANDLING

In the B 7700, hardware interrupts are treated as hardware-originated procedure calls. When the hardware detects an interrupt condition, the CPM causes an MSCW to be placed in the stack, then places in the stack an IRW addressing the interrupt handling procedure, places two parameters in the stack to identify and describe the interrupt condition, and then causes the interrupt handling procedure to be entered. When the interrupt handling procedure is entered, the D registers are updated

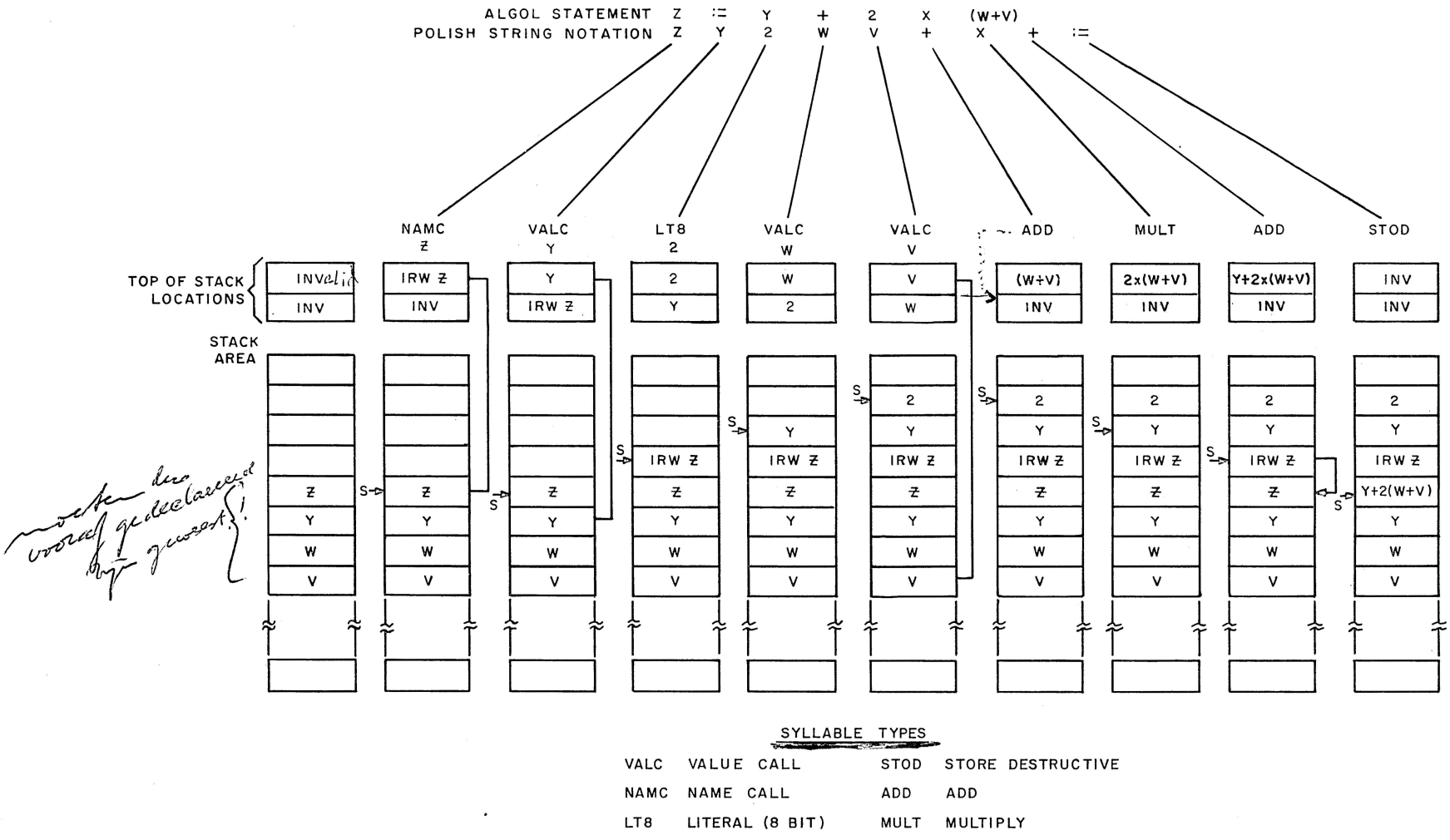


Figure II-2-11. Stack Operation

Table II-2-1. Description of Stack Operation

Execution Sequence	Polish Notation Element	Syllable Type Compiled	Function of Syllable During Running of the Program
0	-	-	Stack location of program variables illustrated.
1	Z	Name call for Z.	Build an indirect reference word that contains the address of Z and place it in the top of the stack.
2	Y	Value call for Y.	Place the value of Y in the top of the stack.
3	2	Literal 2.	Place a 2 in the top of the stack.
4	W	Value call for W.	Place the value of W in the top of the stack.
5	V	Value call for V.	Place the value of V in the top of the stack.
6	+	Operator add.	Add the two top words in the stack and place the result in the A location as the top of the stack operand.
7	x	Operator multiply.	Multiply the two top-of-stack operands. The product is left in the A location as the top of the stack operand.
8	+	Operator add.	Add the two top words in the stack and leave the result in the A location as the top of the stack operand.
9	:=	Operator store destructive.	Store an item into memory. The address in which to store is indicated by an indirect reference word or a data descriptor. The address can be above or below the item stored.

to make all legitimate variables addressable. Similarly, upon return from the interrupt handling procedure, the D registers are again updated to make all of the variables of the former procedure addressable again. A detailed description of interrupt handling is provided in chapter III.

MULTIPLE STACKS AND RE-ENTRANT CODE

The B 7700 stack mechanism provides a facility for handling several active stacks, which are organized in a tree structure. The trunk of this tree structure is a stack containing MCP global quantities.

Level Definition

As the MCP is requested to run an execution of a program, a level-1 branch of the stack is created. This level-1 branch is a separate stack which contains only the descriptors pointing to the executable code and the read-only data segments for the program. Emerging from this level-1 branch is a level-2 branch, containing the variables and data for this job. Starting from the job's stack and tracing downward through the tree structure, one finds first the stack containing the variables and data for the job (at level 2), the segment descriptor to be executed (at level 1), and the MCP's stack at the trunk (level 0).

Re-Entrance

A subsequent request to run another execution of an already-running program requires that only a level-2 branch be established. This level-2 stack branch emerges from the level-1 stack of the already-running program. Thus two jobs which are different executions of the same program have a common node, at level-1, describing the executable code. It is in this way that program code is re-entrant and shared. This results simply from the proper tree-structured organization of the various stacks within the machine. All programs within the system are re-entrant, including all user programs as well as the compilers and the MCP.

Job-Splitting

The B 7700 stack mechanism also provides the facility for a single job to split itself into two independent jobs. A common use of this facility occurs when there is a point in a job where two relatively large independent processes must be performed. This splitting can be used to make full use of a multiprocessor configuration, or to reduce elapsed time by multiprograming the independent processes.

A split of this type establishes a new limb of the tree-structured stack, with the two independent jobs sharing that part of the stack which was created before the split was requested. The process is recursively defined and can happen repeatedly at any level.

Stack Descriptor

Stack branches are located by an array of descriptors, the stack vector array (figure II-2-12). There is a data descriptor in this array for every stack branch. This data descriptor, the stack descriptor, describes the length of the memory area assigned to a stack branch and its location in either main memory or disk.

A stack number is assigned to each stack branch. The stack number is the index value of the stack descriptor in the stack vector array.

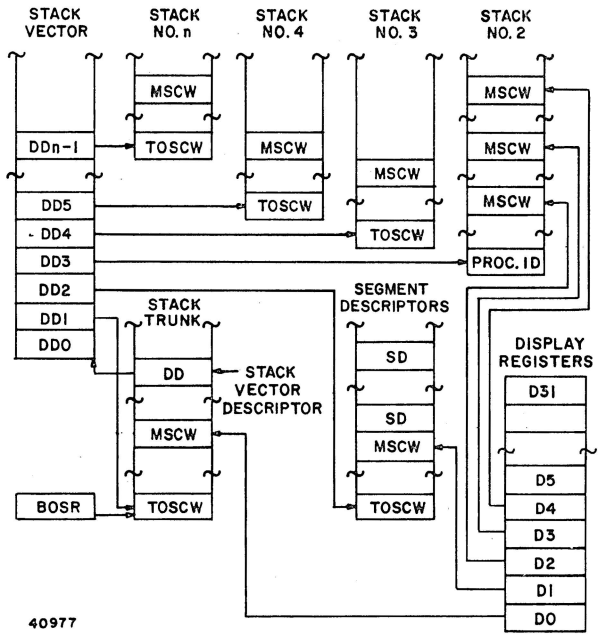


Figure II-2-12. Multiple Linked Stacks

Stack Vector Descriptor

The array size of the stack vector and its location in memory is described by the stack vector descriptor, located in a reserved position of the trunk of the stack (figure II-2-12). All references to stack branches are made through the stack vector descriptor, indexed by the stack number.

Presence Bit Interrupt

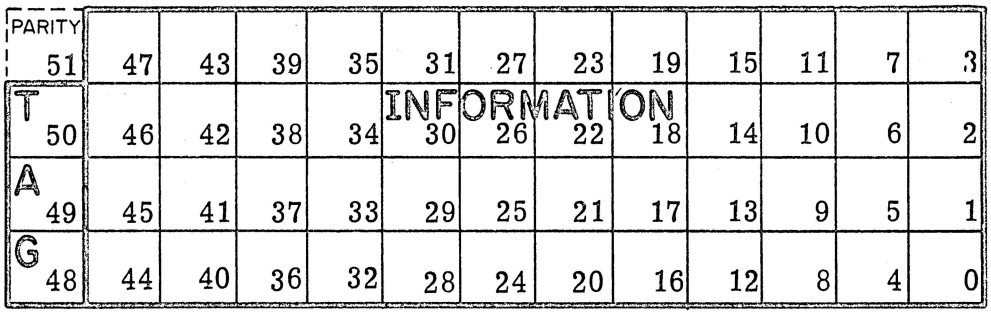
A Presence Bit Interrupt results when an addressed stack is not present in memory. This Presence Bit Interrupt facility permits stack overlays and recalls under dynamic conditions. Idle or inactive stacks may be moved from main memory to disk as the need arises and, when a stack is subsequently referenced, a Presence Bit Interrupt is generated to cause the MCP to recall the nonpresent stack from disk.

SECTION 3
PROCESSOR WORD FORMATS

GENERAL

The basic information structure of the B 7700 is the word. As transferred between CPMs or IOMs and core memory, a word consists of 52 bits (see figure II-3-1), and is considered in three parts: a parity bit, which is used to maintain overall parity for the word being transferred; a 3-bit tag field, which indicates the type of information contained within the word, and a 48-bit information field, which contains the actual information.

The tag field not only serves to identify the type of information contained in the word but also can be thought of as an extension of the operator being executed against the word. For example, because the tag field indicates to the arithmetic unit whether the operation involves single precision or double precision operands, a single instruction (ADD) serves both types of operations. In similar fashion, if the sum obtained was a double precision number (requiring two memory words of storage), and the receiving



Field	Bits	Description
Parity	50:1	Parity bit. Odd parity for the 51 bit word.
Tag	50:3	Value of this field indicates the usage of the information field, as described below.
INFORMATION	47:48	Use of this field depends on the value of the tag field.

Tag Value	Information Field Usage
0	Single Precision Operand, Logical Operand, String Operand, Occurs Index Word, Time of Day Function Word
1	Indirect Reference Word, Stuffed Indirect Reference Word
2	Double Precision Operand
3	Mark Stack Control Word, Return Control Word, Top of Stack Control Word, Program Word, Segment Descriptor
4	Step Index Word
5	Data Descriptor, String Descriptor
6	Uninitialized Operand
7	Program Control Word

40978

Figure II-3-1. Basic Word Format

memory word indicates that a single precision operand was resident there, the CPM will round the sum to single precision and then store it.

The tag field also prevents the user from writing over program code or read-only data areas, and prevents him from reading (as data) program code, processor control words, and uninitialized operands.

Consider the bit assignments for the tag field, as illustrated. One can see that words which have bit 48 set, such as IRW's, SIRW's, Segment Descriptors, MSCW's, RCW's, Data Descriptors, and Program Control Words, should not be alterable by the user. The CPM will not allow such words to be modified except by use of the overwrite operators. Words that are used for stack control, such as code, segment descriptors, MSCW's, RCW's, and PCW's, have bits 49 and 48 of the tag field set. The CPM will not allow such words to be interpreted as operands or to be copied.

The information field may be used to store data (logical operands, string operands, numeric operands), to store program code (program word), to address data or code outside of the stack (data descriptor, string descriptor, segment descriptor), to address within stacks (indirect reference word, stuffed indirect reference word, program control word), to store information regarding stack history (mark stack control word, return control word, top of stack control word), or to provide a parameter for use with certain operators (step index word, occurs index word, read time of day function word). Operands are described in section 1, and program words are described in section 2 of this chapter. The remainder of these processor control words are described in this section.

WORDS FOR ADDRESSING OUTSIDE OF THE STACK

There are three types of descriptors which are used for addressing data or code which is not resident in the stack. The type of descriptor is directly related to the data or code being referenced. Thus, a segment descriptor will always address a segment of program code (contained in program words), a string descriptor will always address a string of characters (contained in string operands), and a data descriptor will address an array of logical or numeric operands.

The ADDRESS field in each of these descriptors is 20 bits in length; this field contains the absolute address of an array in either system main memory or in the backup disk file, as indicated by setting of the Presence bit (P). The referenced data is in main memory when the presence bit is set.

Presence Bit

A Presence Bit Interrupt occurs when the job references data by means of a descriptor in which the P-bit is equal to 0; i.e., the data is located in a disk file, rather than in main memory. The Master Control Program (MCP) recognizes the Presence Bit Interrupt and transfers data from disk file storage to main memory. After the data transfer to main memory is completed, the MCP marks the descriptor present by set-

ting the P-bit to 1, and places the new main memory address into the address field of the descriptor. The interrupted job is then reactivated.

Index Bit

A Data Descriptor describes either an entire array of data words, or a particular element within an array of data words. If the descriptor describes the entire array, the Index bit (I-bit) in the descriptor is 0, indicating that the descriptor has not yet been indexed. The length field of the descriptor defines the length of the data array.

Invalid Index

A particular element of an array is described by indexing an array descriptor. Memory protection is ensured during indexing operations by performing a comparison between the length field of the descriptor and the index value. An Invalid Index Interrupt results if the index value exceeds the length of the memory area defined by the descriptor, or if the index is less than 0.

Valid Index

If the index value is valid, the length field of the descriptor is replaced by the index value, and the I-bit in the descriptor is set to 1 to indicate that indexing has taken place. The address and index fields are added together to generate the absolute machine address whenever an indexed Data Descriptor in which the P-bit is set is used to fetch or store data.

The Double-Precision bit (D) is used to identify the referenced data as single-or double-precision and directly affects the indexing operation. The D-bit equal to 1 signifies double-precision and causes the index value to be doubled before indexing.

Read-Only Bit

The Read-Only bit (R) specifies that the memory area described by the Data Descriptor is read-only area. If the R-bit of a descriptor is set to 1, and the area referenced by that descriptor is used for storage purposes, an interrupt results.

Copy Bit

The Copy bit (C) identifies a descriptor as a copy of a master descriptor and is related to the presence-bit action. The copy bit links multiple copies of an absent descriptor (i.e., the presence bit is off) to the one master descriptor. The copy bit mechanism is invoked when a copy is made in the stack. If it is a copy of the original, absent descriptor, the processor sets the copy bit to 1 and inserts the address of the master descriptor into the address field. Thus, multiple copies of absent data descriptors are all linked back to the master descriptor.

Data Descriptor

Data descriptors refer to data areas, including input/output buffer areas. The data descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in operands is contained in the

length field of the descriptor. Data descriptors may directly reference any memory word address from 0 through 1,048,576. The structure of the data descriptor is illustrated in figure II-3-2.

String Descriptor

String descriptors refer to strings of 4-bit digits, 6-bit or 7-bit characters, or 8-bit bytes. The string descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in characters is contained in the length field of the descriptor. The structure of the String Descriptor is illustrated in figure II-3-3.

Segment Descriptors

Segment descriptors refer to areas of program code. The descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in program words is contained in the length field of the descriptor. The structure of the segment descriptor is illustrated in figure II-3-4.

WORDS FOR ADDRESSING WITHIN STACKS

There are three types of words which are used for addressing data or descriptors which are resident within a stack. 1 A Program Control Word is used, at the time of procedure entry, to locate a segment descriptor (and the proper word and syllable of code) for the procedure. 2 An Indirect Reference Word is used to address within the current addressing environment of a procedure. 3 A Stuffed Indirect Reference Word is used to address outside the current addressing environment of a procedure.

1 Program Control Word

The Program Control Word (PCW), and the MSCW are used during entry into a procedure. The organization of the PCW is illustrated in figure II-3-5 and contains the following:

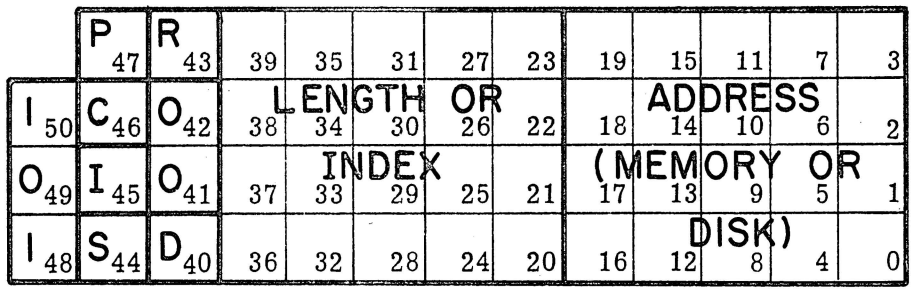
2 Indirect Reference Word

Referencing a variable within the current addressing environment of a procedure is accomplished through the address couple in the Indirect Reference Word (IRW). References are relative to the D register specified by the address couple. The format of the IRW is shown in figure II-3-6.

3 Stuffed Indirect Reference Word

Reference to variables outside the current environment is accomplished by a Stuffed Indirect Reference Word. This addressing is relative to the base of the stack in which the variable is located.

The SIRW contains the stack number, the location (DISP) of the MSCW, and the index to the variable relative to the MSCW. The absolute memory location of the variable is formed by adding the contents of DISP and index to the base address of the referenced stack from the stack descriptor. The contents of the SIRW (with the exception of index) are dynamic and are accumulated as the program is executed. The stack number and DISP fields are entered into the SIRW by the Stuff Environment (STFF) operator. The bit format of the SIRW is shown in figure II-3-7.

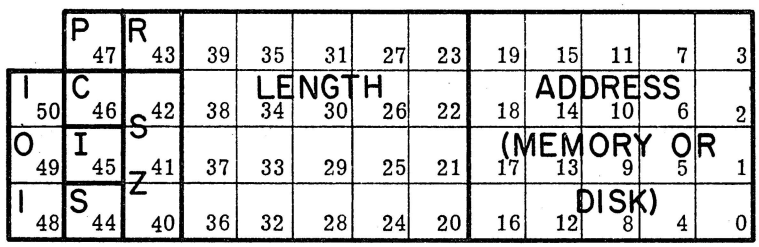


Field	Bits	Description
Tag	50:3	Tag field. Value of five.
P	47:1	Presence bit. Indicates the presence or absence of data in main memory. A 0 causes a presence bit interrupt whenever the descriptor is used by a processor to obtain non-present data. A 1 indicates that the data described is in main memory.
C	46:1	Copy bit. A 0 indicates that this is the original descriptor for the particular data area. A 1 indicates that this descriptor is a copy of the original descriptor.
I	45:1	Indexed bit. A 0 indicates that an indexing operation is required before the descriptor may be used to obtain data. A 1 indicates that indexing has already taken place and the index value is stored in bit positions 39:20 (Length or Index).
S	44:1	Segmented bit. A 0 indicates that the data is not segmented. A 1 indicates that the data is divided into segments.
R	43:1	Read-only bit. A 0 indicates that the data may be referenced for reading or writing. A 1 indicates that the area cannot be used for data storage.
	42:2	Size field, must be 0 to indicate a data descriptor.
D	40:1	Double-precision bit. A 0 indicates single-precision operands, a 1 indicates double-precision operands.
LENGTH OR INDEX	39:20	This field contains either the length (in operands) of the memory area (if bit 45 = 0) or an index value (if bit 45 = 1). If bit 45 equals 0, the descriptor has not been indexed. This field is used for size checking during the indexing operation. If bit 45 equals 1, the descriptor has been indexed. For a double-precision operation, the index is doubled after index size checking, and the result is stored in the index field.
ADDRESS (MEMORY OR DISK)	19:20	This field contains either a main memory or disk address. If the presence bit (bit 47) equals 1, this field contains the memory address of data. If the presence bit equals 0 and the copy bit (bit 46) equals 0, this field contains the disk address of the data. If the presence bit equals 0 and the copy bit equals 1, this field contains the memory address of the original descriptor.

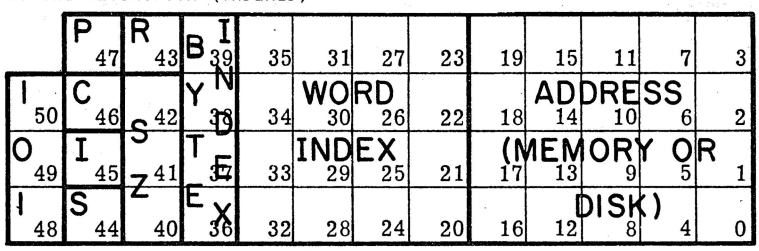
40979

Figure II-3-2. Data Descriptor

STRING DESCRIPTOR (NON-INDEXED)



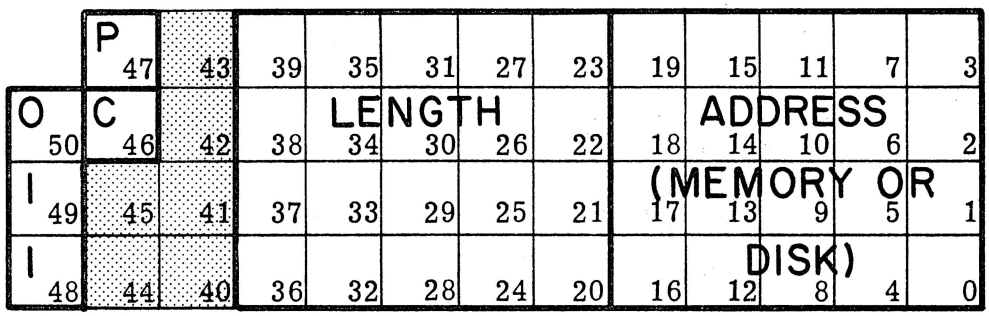
STRING DESCRIPTOR (INDEXED)



Field	Bits	Description
Tag	50:3	Tag field. Value of five.
P	47:1	Presence bit. A 0 causes a presence bit interrupt if the descriptor is used to access data. A 1 indicates the data is present in main memory.
C	46:1	Copy bit. A 0 indicates that this is the original descriptor for the particular data area. A 1 indicates that this descriptor is a copy of the original descriptor.
I	45:1	Indexed bit. A 0 indicates indexing is required. A 1 indicates that indexing has taken place and the word and character index are in the WORD INDEX and BYTE INDEX fields.
S	44:1	Segmented bit. A 0 indicates that the data area is not segmented. A 1 indicates that the data is segmented.
R	43:1	Read only bit. A 0 indicates that the data may be referenced for reading or writing. A 1 indicates that the data can be read only.
SZ	42:3	Size field. 100 indicates character size of 8-bit bytes, 101 indicates 7-bit ASCII characters, 011 indicates 6-bit characters, and 010 indicates 4-bit digits.
LENGTH	39:20	Bits 39:20, contain either the length of the memory area (bit 45=0) or an index value (bit 45=1). When bit 45 equals 0, this field contains the length of the area in digits, characters or bytes.
BYTE INDEX	39:4	Byte index (Bit 45=1).
WORD INDEX	35:16	Word Index (Bit 45=1).
ADDRESS (MEMORY OR DISK)	19:20	This field contains either a main memory or a disk address. If the presence bit (bit 47) is 1, the field contains a memory address of the data. If both the presence bit and the copy bit (bit 46) are equal to 0, the field contains the disk address of the non-present data. If the presence bit is 0 and the copy bit is 1, the field contains the memory address of the original descriptor.

40980

Figure II-3-3. String Descriptor

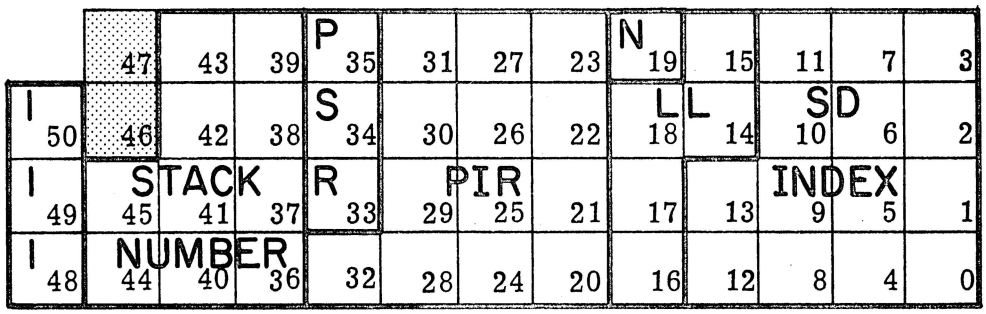


Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
P	47:1	Presence bit. A 0 indicates that the segment is absent from main memory.
C	46:1	Copy bit. A 0 indicates that this is the original segment descriptor. A 1 indicates that this is a copy of the original segment descriptor.
	45:4	Not used. Unused bits may be either 0 or 1.
LENGTH	39:20	The length of the program segment in words.
ADDRESS (MEMORY OR DISK)	19:20	This field contains either the main memory address or the disk file address. If the presence bit (bit 47 equals 1, the field contains the main memory address of the program segment. If both the presence bit and the copy bit (bit 46) equal 0, the field contains the disk address of the non-present program segment. If the presence bit equals 0 and the copy bit equals 1, the field contains the absolute memory address of the original program segment descriptor.

40981

Figure II-3-4. Segment Descriptor

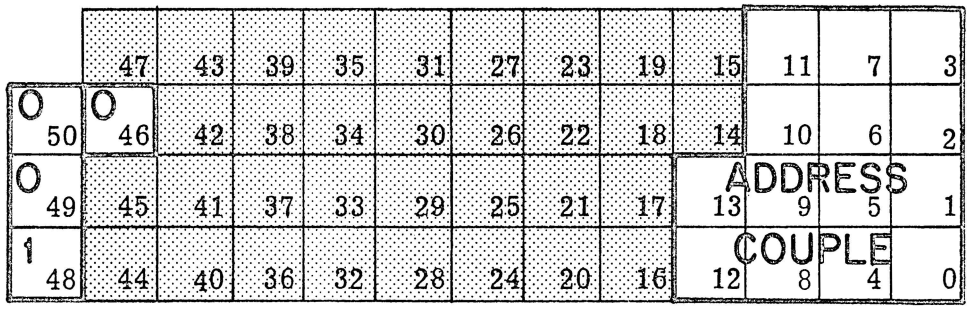
Refer to areas of program code.



Field	Bits	Description
Tag	50:3	Tag field. Value equals seven.
	47:2	Not used.
STACK NUMBER	45:10	The number of the stack which contains the PCW.
PSR	35:3	The program syllable (0-5) within the word located by PIR.
PIR	32:13	Index to the Program Base Register. Locates a word within the code segment.
N	19:1	Normal state (0) or control state (1).
LL	18:5	The level of the procedure being entered.
SD INDEX	13:14	The segment descriptor index. Bits 12 through 0 specify the value to be added to the address located by either D-register 0 or 1. When bit 13 equals 0, D-register 0 is selected; when bit 13 equals 1, D-register 1 is selected. Locates a segment descriptor.

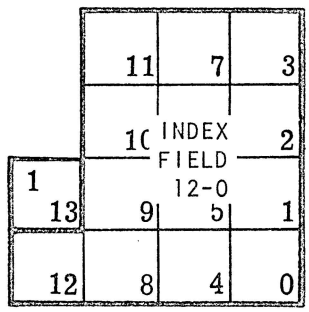
40982

Figure II-3-5. Program Control Word

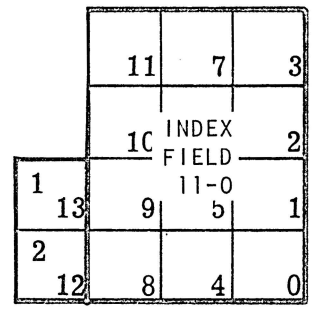


Field	Bits	Description
Tag	50:3	Tag field. Value equals one.
	47:1	Not used
	46:1	Environment bit. Must equal zero for an IRW. (1 = SIRW).
	45:32	Not used.
Address Couple	13:14	SELECTS D REGISTER (ACCORDING TO CURRENT PROGRAM LEVEL AS INDICATED BY rLL) AND PROVIDES INDEX VALUE (SEE BELOW).

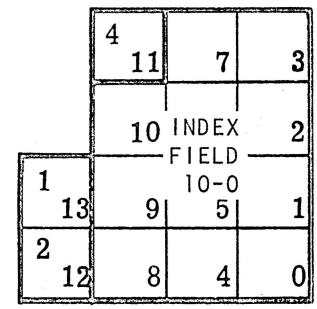
PROGRAM LEVEL 0-1



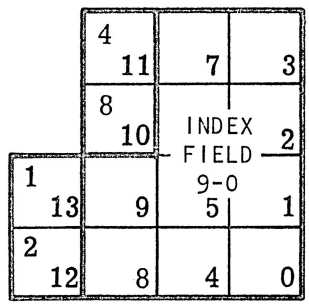
PROGRAM LEVEL 0-3



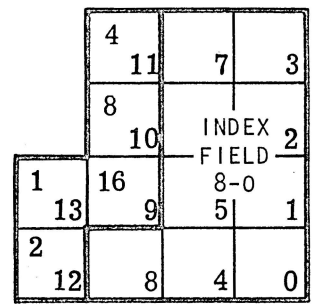
PROGRAM LEVEL 0-7



PROGRAM LEVEL 0-15



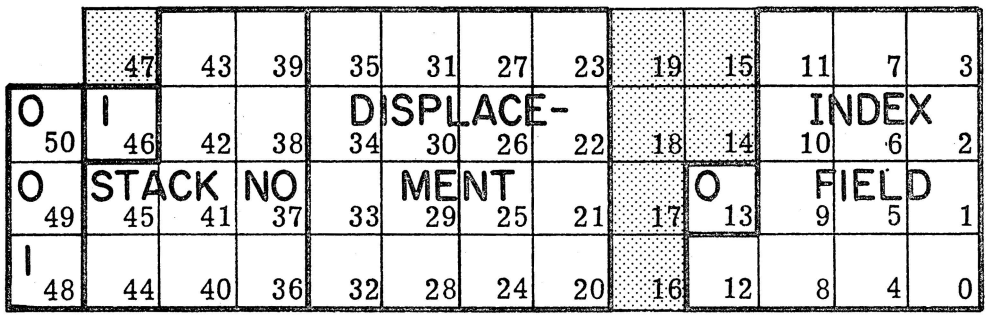
PROGRAM LEVEL 0-31



NOTE: THE BIT ORDER OF THE LL FIELD IS INVERTED.

40983

Figure II-3-6. Indirect Reference Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals one.
	47:1	Not used.
	46:1	Environment bit. Must be a one (0=IRW).
STACK NO.	45:10	The number of the stack containing the referenced word.
DISPLACEMENT	35:16	This number, added to the stack base address, addresses an MSCW.
	19:6	Not used.
	13:1	Must be 0.
INDEX FIELD	12:13	This number, added to the address of the MSCW, addresses the referenced word.

40984

Figure II-3-7. Stuffed Indirect Reference Word

WORDS FOR STORING STACK HISTORY

Certain words can be thought of as words used for storing stack history. These words, used for procedure entry and exit, as well as for storing the stack state for inactive stacks, include the Mark Stack Control Word, the Return Control Word, and the Top Of Stack Control Word.

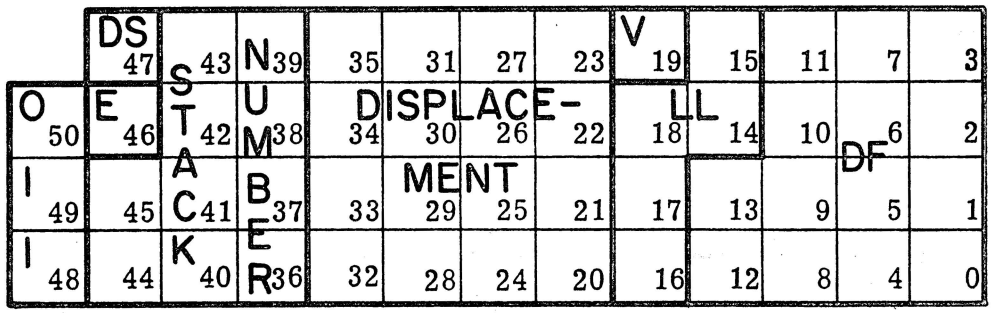
Mark Stack Control Word

The Mark Stack Control Word (MSCW), together with the Return Control Word (RCW), provides a linking mechanism for the history of previous control-register settings through the stack.

The MSCW is placed in the stack by the Mark Stack operator. The MSCW is organized as illustrated in figure II-3-8.

Return Control Word

The Return Control Word (RCW) and the MSCW are used for subroutine



Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
DS	47:1	Different-stack bit. A 0 indicates that the stack-number field refers to the current stack. A 1 indicates that the stack-number field refers to a different stack.
E	46:1	Environment bit. A 0 indicates an inactive MSCW, generated directly by the Mark Stack operator. The procedure entry has not been performed. A 1 denotes an active MSCW generated upon entry into a procedure, at which time the environment fields (stack number, displacement, value, and LL fields) are stored into the MSCW.
STACK NUMBER	45:10	Stack-number field. Contains the number of the stack from which the PCW was obtained at procedure-entry.
DISPLACEMENT	35:16	Displacement field. When added to the stack base address, locates the MSCW of the prior lexicographic level.
V	19:1	Value bit. A 0 indicates that the MSCW was generated during any operation that will be restarted from the beginning. A 1 indicates that the operator must continue after the Exit or Return which refers to this MSCW (e.g., an accidental entry by a Value Call).
LL	18:5	LL field. Denotes the lexicographical level at which the program will run when the procedure is entered.
DF	13:14	Denotes the stack history. This field is used to locate, in the stack, the preceding MSCW (i.e., the previous "F" register setting).

40985

Figure II-3-8. Mark Stack Control Word

handling. The Return Control Word stores the environment to which the subroutine will return. The organization of the RCW is illustrated in figure II-3-9.

Top Of Stack Control Word

The Top Of Stack Control Word (TOSCW) contains all information needed to restore the operating environment when a stack (or process) is activated. When a stack is active, the first word of the stack is a single precision operand containing the processor ID (a number, 0 through 7). When the stack is made inactive, the processor ID is changed to a TOSCW, containing the status of various processor flip-flops necessary to restore the stack's environment when it is again activated. The TOSCW is created by the Move Stack (MVST) operator. The TOSCW is illustrated in figure II-3-10.

WORDS USED AS SPECIAL PARAMETERS

Certain control words are used only as a parameter to a single operator. Among these are the Step Index Word, used with the Step and Branch operator; the Occurs Index Word, used with the Occurs Index operator; and the Read Time Of Day Function Word, used with the Scan In operator.

Step Index Word

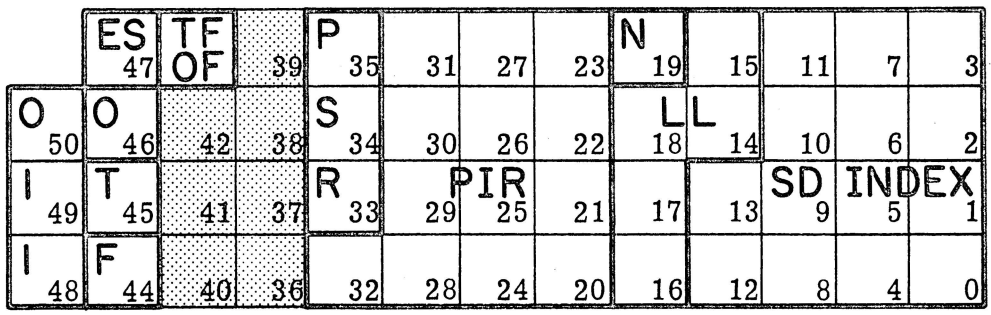
The Step Index Word (SIW) is used as a parameter to the Step and Branch operator, to increase the efficiency of this operator in iteration loops. When the Step and Branch operator is invoked, the SIW addressed by the IRW in the top of stack location is located. The increment field is added to the current value field. If the current value field is then greater than the final value field, PIR and PSR are set from the next two syllables in the program code string and the branch is made. If the current value field is not greater than the final value field, PIR and PSR are advanced three syllables, the SIW is replaced in memory, and the iteration loop continues. The format of the SIW is illustrated in figure II-3-11.

Occurs Index Word

The Occurs Index Word (OIW) is used to index a field within an array. COBOL permits arrays to be constructed of a series of fields of a specified character size (through use of the OCCURS clause). This series of fields may not necessarily begin at a word boundary, because the array may be one of several items subordinated under a group item. The OCRX operator, together with an OIW in the A location and an index value in the B location, is used to calculate a new index value which is left in the top of the stack. The original index value is an integer which indicates the relative position of the desired field within the array. The new index value is the displacement (in characters) of the desired field from the first character of the array. The character size (specified in a descriptor) and the index value (left in the top of stack) can then be used to address the desired field. The format of the OIW is shown in figure II-3-12.

Time of Day Function Word

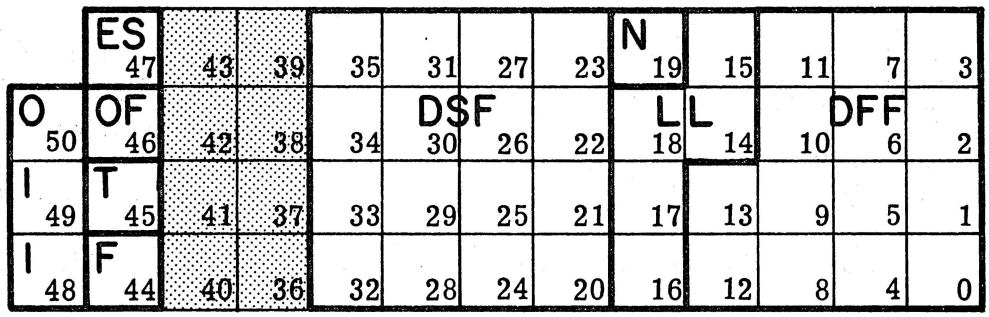
This word is used as a parameter to the Scan In operator, to specify that the time of day is to be scanned in. The format of the Time of Day Function Word is shown in figure II-3-13.



Field	Bits	Description
Tag	50:3	Tag field. Value of three.
ES	47:1	External Sign flip-flop.
O	46:1	Overflow flip-flop.
T	45:1	True/False flip-flop.
F	44:1	Float flip-flop.
TFOF	43:1	True/False Flip-flop occupied flip-flop.
	42:7	Not used.
PSR	35:3	Program syllable of the operator to be executed after return from the subroutine.
PIR	32:13	PIR setting of the operator to be executed next in the calling routine.
N	19:1	Normal state (0) or control state (1) procedure.
LL	18:5	Level of the calling procedure when the RCW was generated (at procedure entry).
SD INDEX	13:14	Segment descriptor index. Bits 12 through 0 specify the value to be added to the address located by either D-register 0 or 1. When bit 13 = 0, D-register$_0$ is selected; when bit 13 = 1, D register 1 is selected.

40986

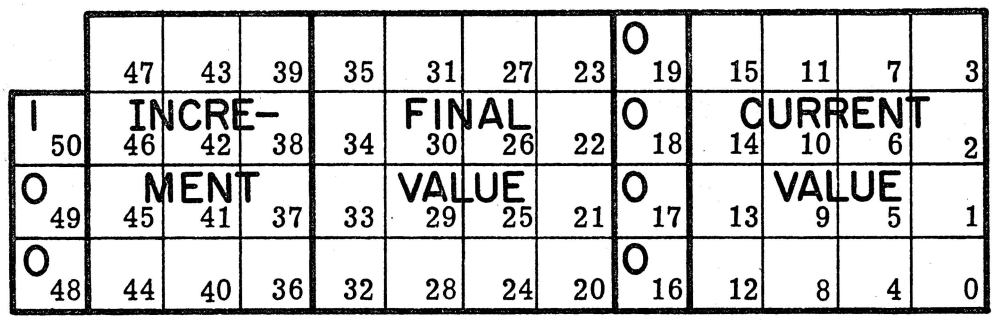
Figure II-3-9. Return Control Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals three.
ES	47:1	External sign flip-flop.
OF	46:1	Overflow flip-flop.
T	45:1	True/False flip-flop.
F	44:1	Float flip-flop.
	43:8	Not used.
DSF	35:16	Delta S-register field. The value of S-register displacement above BOSR.
N	19:1	Normal-control state flip-flop. 0 = normal; 1 = control state.
LL	18:5	Lexicographic level.
DFF	13:14	Delta F-register field. The value of F-register displacement below the S-register.

40987

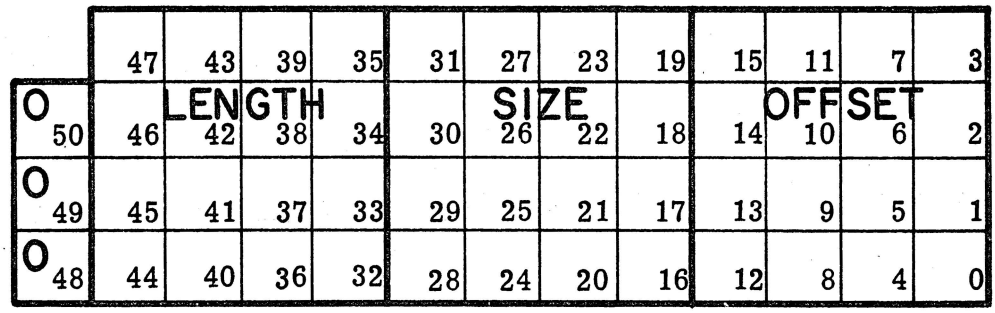
Figure II-3-10. Top of Stack Control Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals four.
INCREMENT	47:12	Increment: value to be added to current value field.
FINAL VALUE	35:16	Final value: value used to terminate the iteration loop.
	19:4	Must be 0 for SIW.
CURRENT VALUE	15:16	Current value or count. The branch is made if this field is greater than the final value field.

40988

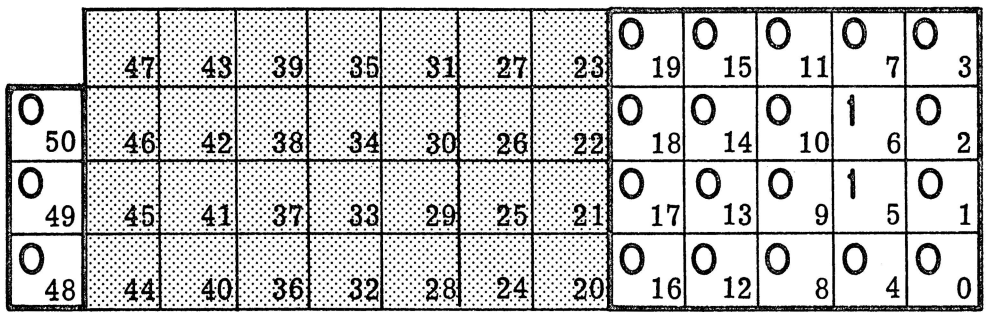
Figure II-3-11. Step Index Word



Field	Bits	Description
Tag	50:3	Tag field. Value equals zero.
LENGTH	47:16	The length, in characters, of each field in the array.
SIZE	31:16	The size, in fields, of the array.
OFFSET	15:16	The number of characters preceding the first field of the array.

40989

Figure II-3-12. Occurs Index Word



Field	Bits	Description
Tag	50:3	Tag field. Value equal zero.
	47:28	Not used.
	19:13	Must equal zero.
	6:2	Must equal three.
	4:5	Must equal zero.

Figure II-3-13. Time of Day Function Word

SECTION 4
INPUT/OUTPUT SUBSYSTEM MAP STRUCTURE

INTRODUCTION

The B 7700 Input/Output Modules (IOM) operate in parallel with the Central Processor Modules (CPM). The purpose of the IOM is to control all data transfers between main memory and peripheral devices, or between two peripheral devices, so that the CPM is released from I/O operations at the earliest possible moment. In brief, the IOM controls not only the selection of I/O requests from lists of such requests in main memory, but also the selection of optimum paths to the desired devices, the initiation of requests on the appropriate device, the transfer of data as specified by the requests, and the construction of a list of completed requests in main memory. The CPM, on the other hand, builds the I/O request, places it in the appropriate list in main memory, notifies the IOM of the presence of the request (if this is the only request for the device), and then is free to continue with other processing. Routinely, the CPM checks memory for the presence of completed I/O requests and processes the completed requests.

Each IOM is, in effect, a separate computer with its own local memory, logic, arithmetic, and communication capabilities. This independent processing capability permits the IOM to perform routine input-output tasks without interrupting the CPM. Thus the IOM can control transfers of data between peripheral storage devices and main memory or other storage devices without direct supervision of the CPM. In fact, parallelism within the IOM permits it to initiate, service, and terminate data transfers for several users while the CPM is processing data for yet another user.

QUEUE-DRIVEN I/O

To allow the IOMs to properly select paths to the devices and to service I/O requests, certain structures are created by software when the system is initialized. These structures, which provide a mechanism to allow the CPMs to queue I/O requests, allow each IOM to be aware of the requests, of the devices it can service, and of the order of priority of devices served by an exchange. These structures are referred to as the I/O Subsystem Map, and hence this type of I/O is often referred to as "map" I/O or "queue-driven" I/O. Because the use of the map allows the IOM to process many I/O operations in parallel, independent of CPM, I/O performed using the map is also known as asynchronous I/O. The IOM may also operate synchronously to process one I/O request at a time; however, such synchronous operation is used only for special applications such as system initialization and is not further discussed in this section.

The operation of asynchronous I/O is illustrated in simplified form in figure II-4-1. When the I/O subsystem map is initialized the CPM places information about each peripheral device and the paths to it into a table in memory. During operation, the I/O subsystem map is accessed by both the CPM and IOM as I/O requests are built (by the CPM) and processed (by the IOM). In essence, the CPM builds I/O requests and places them in queues of such

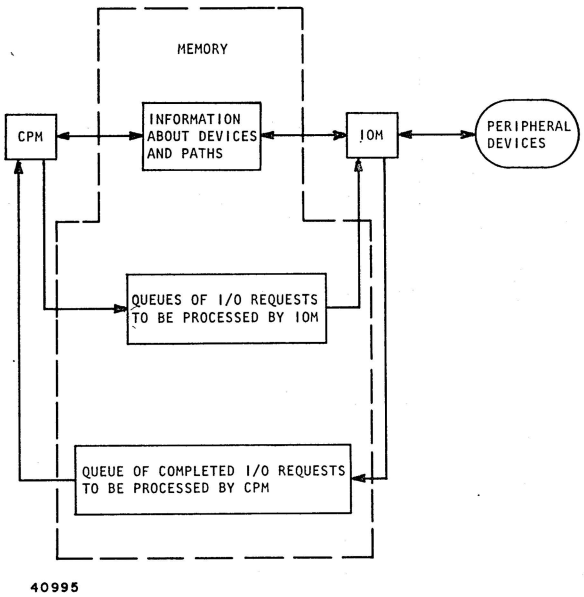


Figure II-4-1. Asynchronous I/O Operation, Simplified Block Diagram

requests in main memory. Each request specifies the desired I/O operation and the device on which the operation is to be performed. The IOM extracts requests from these queues on a first-in first-out basis, processes each request, and places the completed requests into a queue in main memory. Periodically, the CPM extracts the completed requests from the queue in main memory and takes the necessary action to check them. Once the IOM is notified (by the CPM) of the presence of an I/O request in one of the input queues, all requests in that queue will be processed by the IOM independent of CPM actions until the queue becomes empty. The CPM may place additional requests into a queue while the IOM is processing a request from the queue. Thus, once the IOM starts processing a queue, the CPM may process other programs, queue new I/O requests, and perform computations; effectively masking out the IOM transfer times.

ERROR HANDLING

From time to time conditions may arise which prevent I/O operations from being accomplished successfully. A printer may run out of paper, a card punch may be out of cards, or a device may for some reason not be ready. The design of the I/O subsystem map allows the IOMs to continue to process requests for other devices even though an error is detected on a particular device. When the error is recognized by an IOM, processing of further requests for the particular device is suspended, the I/O request is marked as containing an error, and that marked request is linked into the queue of completed requests. The CPM is not interrupted to handle the error; however, when the CPM does process the queue of completed requests it will recognize and process the error. When the error has been processed, the CPM can again cause the IOM to process requests for the device on which the error was detected.

If such a strategy were to be applied to the handling of all input/output errors a catastrophic situation might arise. If, say, the IOM itself were the source of the error, it is conceivable that it could then process all (or many) I/O requests erroneously. However, in such cases the B 7700 IOM stops all processing of I/O requests (for all devices) and immediately interrupts the CPM. In short, I/O errors associated with a particular device cause processing of further requests for the device to be halted but allow the processing of requests for other devices by the IOM to continue. I/O errors which can be associated only with an IOM and not with a particular device cause the IOM involved to stop all

processing of requests (other IOMs are not affected) and causes the system to be interrupted so that the IOM error may be processed. Provision is also made to allow the software to request that the system be interrupted when a particular I/O request is completed.

DEFERMENT OF PATH BINDING

The I/O subsystem map allows the IOM to select the transfer path for a device as the path becomes available. This dynamic path selection is logically similar to the call routing of a long distance telephone network -- the route of the call is selected based on the locations of the correspondents and the available paths. On certain conventional systems this binding may occur before the programmer even begins to code and may involve dictation of the identity of the correspondents (perhaps a physical tape number). In the B 7700 system the user need only be concerned about the type of device to be used (card reader, magnetic tape, disk file, etc.); the MCP will associate the logical file with a physical device when the program is executed, and the IOM, when it initiates each transfer of data, will select an available transfer path to the device.

Maximum I/O throughput can be realized only if the binding of the data path between an IOM and a device is delayed until the device is ready to initiate the job. As shown in figure II-4-2, if device #4 is to be initiated, the path required to connect CPM #1 with device #4 involves selecting between two IOM's and between two channels within each IOM. (The peripheral controls have been excluded from this figure because they do not affect the concepts being described. For purposes of this discussion the peripheral controls may be thought of simply as extensions of the IOMs.) If the path to device #4 were to be preselected programmatically, a situation could develop in which the device is free but the preselected path is not. Thus, execution of the request would be unnecessarily delayed if in fact an alternate path to the device was available.

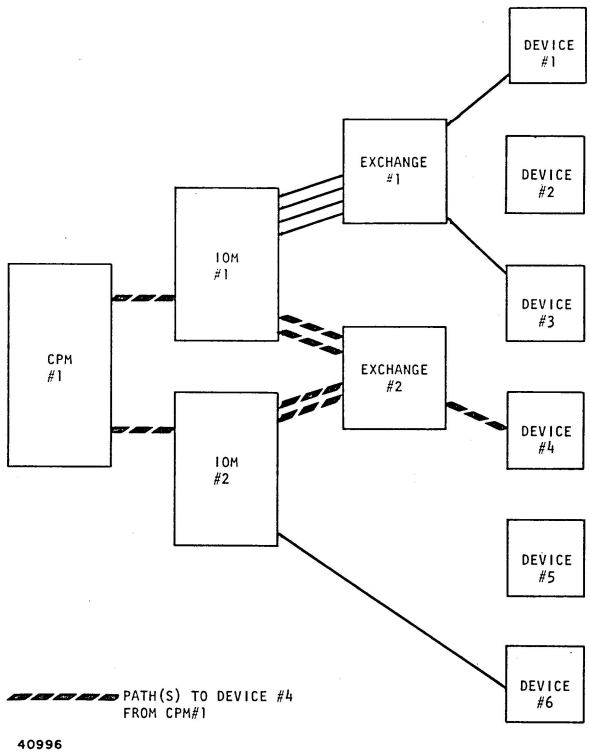


Figure II-4-2. Data Transfer Path Selection

To delay binding the path programmatically generally would require that the CPM which initiated the job be involved in the operation until the request is actually initiated on the specified device. The I/O subsystem map, however, allows the IOM's to manage selection and bind-

ing of paths, allowing the CPM's to be free to do other processing. Thus, because the IOM processes I/O requests without CPM intervention, and because the IOM selects data paths at the time of execution, the total system time required to accomplish an I/O operation is limited to the amount of time required for a CPM to build an I/O request and place it in memory.

I/O SUBSYSTEM MAP

As shown in figure II-4-3, the I/O subsystem map is made up of four major software structures in main memory. These four software structures are addressed by registers within the IOM: the Home Address words are addressed by the HA register; the Unit Table is addressed by the UT register; the Queue Head and Queue Tail words table is addressed by the QH register; and the Status Queue Header is addressed by the SQ register. The IOM uses the Queue Head word for the appropriate device to locate the I/O request. Thus, the IOM can locate any element of the map as necessary. Of course, since the map is constructed by the

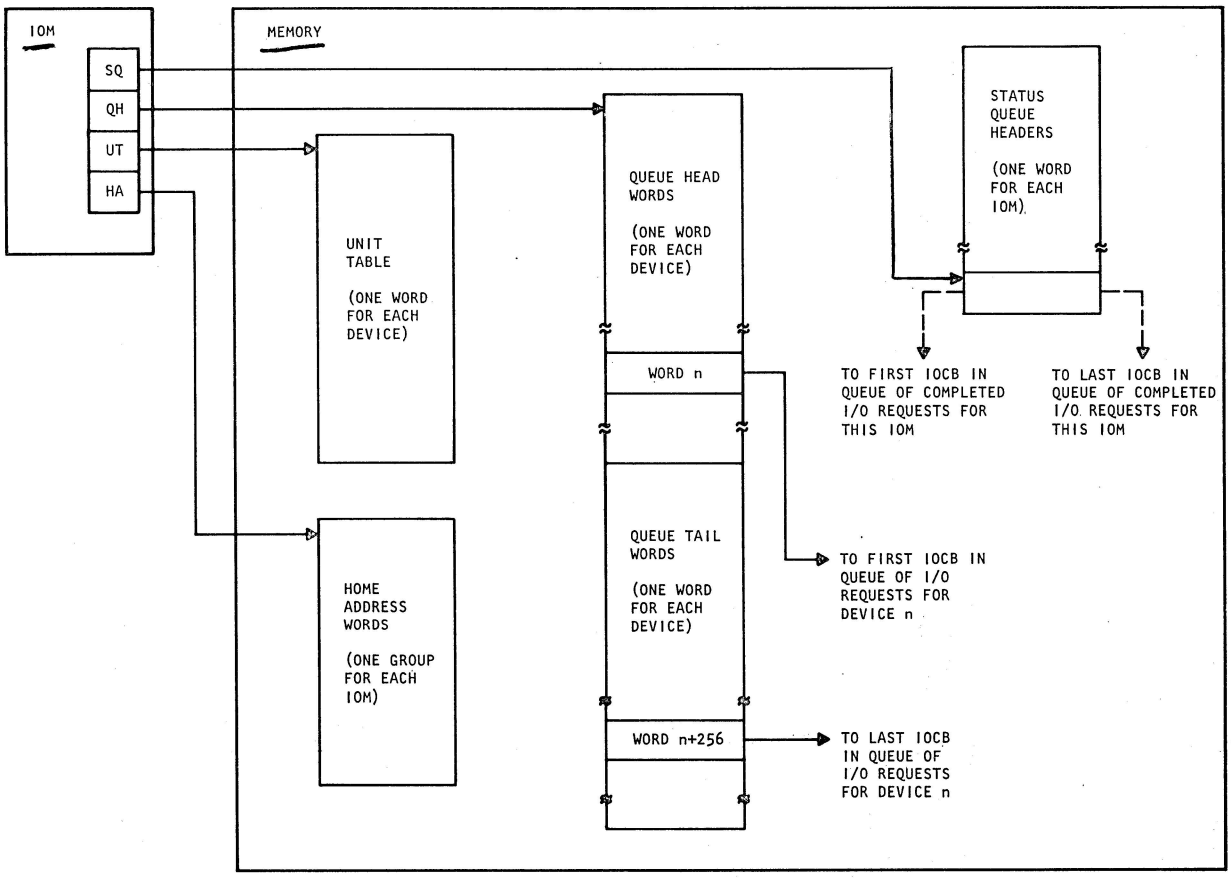


Figure II-4-3. I/O Subsystem Map, Simplified Block Diagram

MCP, it too is aware of the location of each element of the map.

I/O Commands and Requests

Before further discussion of the I/O subsystem map can take place, the difference between an I/O command and an I/O request must be made clear. An I/O command is an order to an IOM which can cause one operation or many operations for one device to be initiated by the IOM. Although there are special I/O commands which control but a single I/O operation, the I/O command most often associated with asynchronous I/O is the Start IO command, which causes the IOM to process I/O requests from a queue of such requests until the queue is empty. Each I/O request contains information describing a single input or output operation that is used not only by the IOM but also by the peripheral control and even the device itself. Each I/O request is made up of several words and is known as an I/O Control Block (IOCB). The IOCB is discussed in detail later in this section; I/O commands are described in Chapter IV.

Map Integrity

Because the I/O subsystem map may be accessed and modified by all CPM's and IOM's in the I/O subsystem, the integrity of the map is protected by special lock bits and lock words. This system of locks prevents conflicts between the IOM's and CPM's which use and modify the map. As shown in figure II-4-4, the system consists of three types of locks; a lock bit and a lock word for each group of Home Address words, and a lock bit for each Unit Table word.

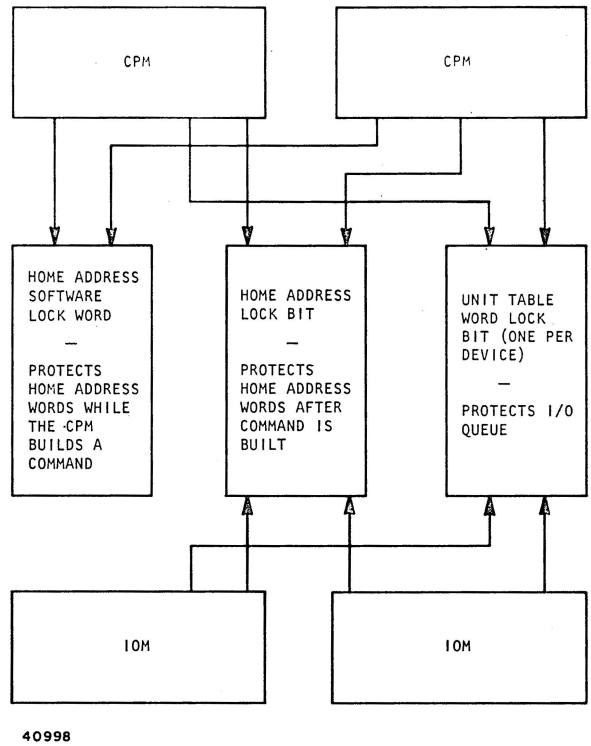


Figure II-4-4. I/O Subsystem Map Protection

The software lock word prevents two or more CPM's from attempting to build I/O commands in the Home Address words simultaneously. This word must be unlocked before a CPM can access the Home Address words; the CPM will immediately lock this word when it gains access.

The Home Address words lock bit prevents a command from being altered once it has been placed in the Home Address words for execution. This word also ensures that the command will be executed only by one IOM. The CPM locks this bit when a command is placed in the Home Address words. Once this bit is locked the Home Address words can only be accessed by an IOM. The IOM which first responds to the command un-

locks this bit so that no other IOM will attempt to execute the command.

The lock bit in each Unit Table word protects the IO queues so that access to an I/O queue is not granted to more than one IOM or CPM at a time. The I/O queue can only be accessed when the lock bit is unlocked. Each IOM or CPM locks the bit when it is using the I/O queue and unlocks the bit when it is finished.

Home Address Words

For each IOM there exists a unique set of Home Address words in memory. The basic purpose of the Home Address words is to provide a location into which CPM's can store an I/O command until an IOM is ready to execute the command. The most generally used command is Start IO, which is used to initiate the processing of a queue of I/O requests for a device by an IOM. Other commands allow the IOM to perform special functions, such as loading into the IOM the addresses of the structures in the I/O map or performing synchronous I/O operations. Other words in the Home Address words are used as software lock words and, in certain cases, to store result descriptors for completed I/O operations.

Unit Table

For each device in the I/O subsystem there is one word in the Unit Table. This word is used both by the MCP and the IOM, and contains a lock bit which prevents conflicts of interest. This word indicates the path or paths to the unit, and provides other information needed by the IOM.

I/O Queue Head and Tail Words

For each device in the I/O subsystem there is one Queue Head word and one Queue Tail word. These words contain the address of the first IOCB and the last IOCB, respectively, in the queue of I/O requests for the unit. If there are no IOCB's to be processed for the unit, these words will be empty.

Status Queue Headers

For each IOM there is a Status Queue Header. Fields in the Status Queue Header contain the addresses of the first and last IOCB in a queue of completed IOCB's. Thus, the Status Queue Header allows each IOM to maintain a single queue of completed I/O requests. Periodically, the MCP checks these completed requests.

Input/Output Control Block

An Input/Output Control Block (IOCB) contains the information needed by the IOM to perform one I/O operation on a device. I/O Control Blocks (see figure II-4-5) contain information needed to link queues of IOCB's together, to describe the I/O operation to be performed, to locate the data buffer to be used for the operation, and in the case of completed IOCB's, to store the result descriptor describing the completed operation. A generalized illustration of an IOCB is shown in figure II-4-5.

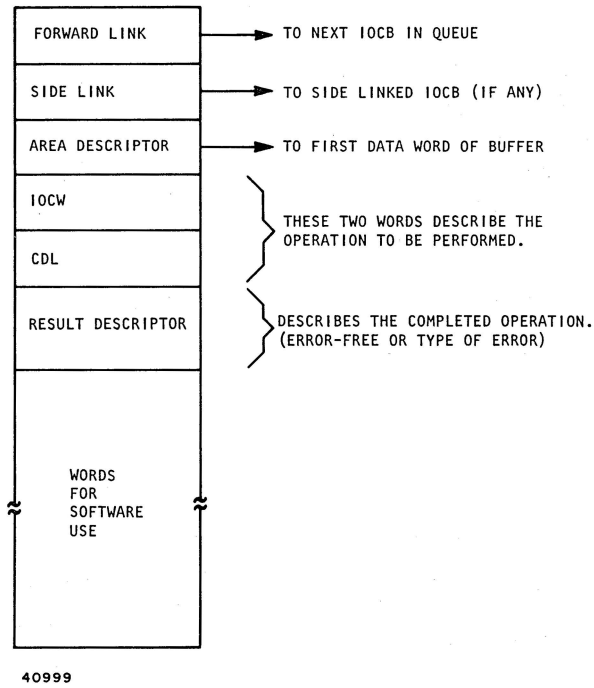


Figure II-4-5. IOCB Format, Simplified

CHAPTER III
CENTRAL PROCESSOR MODULE

SECTION 1
FUNCTIONAL DESCRIPTION OF THE CENTRAL PROCESSOR MODULE

The B 7700 Central Processor Module, which is a highly parallel machine and which is completely program compatible with the B 6700 Processing System, consists of three major functional sections that are operationally independent:

a. The program section, which performs instruction decoding operations of object code strings and address calculations of absolute addresses.

b. The execution section, which performs all arithmetic and logical data manipulation operations.

c. The storage section, which performs all storage related functions.

Figure III-1-1 is a simplified block diagram showing the general interconnections and data flow between the three sections. Communications between the sections is established by operations queues.

The program section consists of the program buffer and barrel, program control unit, the fault control logic and the address unit. The program section is responsible for extracting each instruction from the program code string and initiating processing of the instructions, and for the update of the program index and program syllable registers. The program section also controls and responds to the fault interrupt system. The primary responsibility of the program section is to separate the object code string into operations which are then placed in the appropriate queues for execution section. A few instructions are executed entirely by the program section, such as an unconditional branch, and others are executed in part, such as the address calculation portion of Value Call.

The execution section consists of the execution unit and the execution unit input queues. The execution section is responsible for all data and control manipulations. The execution section performs all arithmetic and logical operations as well as stack related control functions. The execution section is driven in an orderly manner from a first in first out list of operations placed in its operator queue by the program section.

The storage section consists of the storage unit, the stack buffer unit, the associative memory and the communications unit. The storage section is responsible for all storage related functions. Some of the storage section's duties are implied such as maintaining the stack buffer, but most operations are explicit in that they result directly from the processing of program code. Implicit operations for the storage section are placed in the input queue of the storage unit by the program section or in the storage output register by the execution

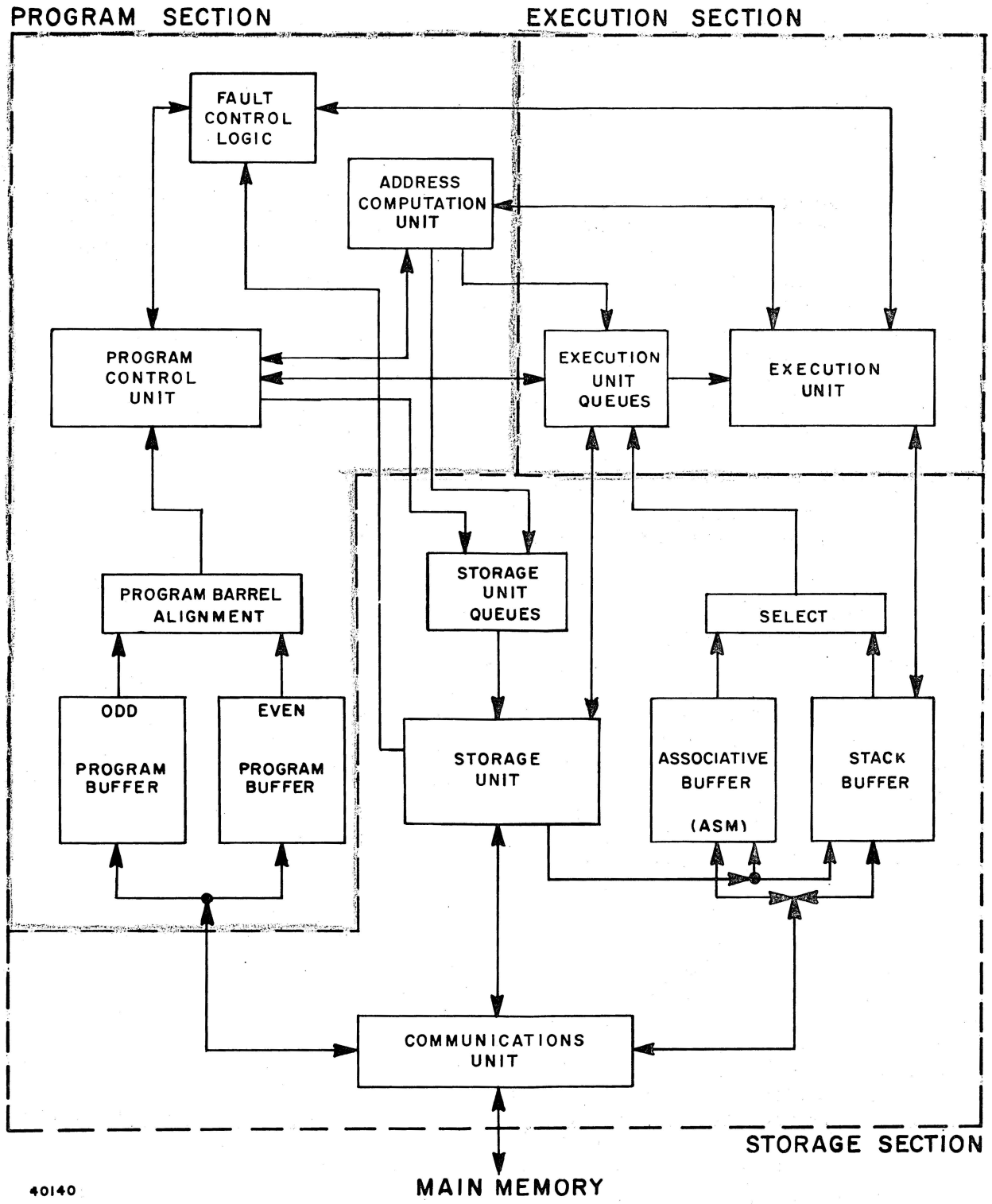


Figure III-1-1. Simplified Block Diagram of Central Processor Module

section. It is the responsibility of the storage section to determine if an address reference points to local storage or to main memory, in which case, a main memory cycle is necessary.

These major sections are subdivided into units which operate relatively independently. The operation of each of the units of the CPM is described separately. First some of the basic operational concepts of the CPM are presented to aid in understanding the physical and conceptual design of the Central Processor Module.

In general, the program operators in the program code string are fetched from memory in multi-word segments and placed in the program buffer. The operators are extracted one at a time by the program control unit and each is separated into one or more micro operators, which are queued for processing by the execution unit. The program control unit determines what data will be required for execution of the micro operators and requests this data from the storage unit. For literal values, which are contained in the code string, the program control unit extracts the data and forwards it directly to the execution unit. Therefore, as the execution unit processes the micro operators, the required data is usually instantly available, allowing the execution unit to perform the required processing without delay. Results derived by the execution unit may either be stored in one of the local memory areas or may be sent through the storage unit and the communications unit to main memory. By using this pipeline technique, relatively high-speed processing has been achieved without compromising equipment reliability.

To further increase processing speed, extensive use has been made of buffer memory areas contained within the processor. These local memory areas are used to store program code, a portion of the active program stack, and frequently referenced variables. The following example shows how these local memory areas increase processing speed by eliminating many memory references.

OPERATIONAL CONCEPTS OF THE CENTRAL PROCESSOR MODULE

The B 7700 Central Processor Module (CPM) is designed as a pipeline processing unit; therefore each processing station may be operating simultaneously on a different task. As any instruction is passed through the processing pipeline, successive operations are performed by the various processing stations until the instruction is fully executed.

Use of Division Overlap and Local Buffering

Figure III-1-2 shows a simple statement along with its compiler generated code and traces each operator as it is encountered in each of the major processing units.

The sequence begins as the first Value Call arrives at the program control unit. This unit selects the appropriate display register and calculates the absolute address.

The address and operator are placed in the input queue of the storage unit. At the same time, the operator is placed in the input queue of the execution unit. The execution unit then begins to wait for the return of the value. Next, the program control unit processes the Name Call. Detection of the Name Call

STATEMENT

A [I] : = B [I]

COMPILER CODE

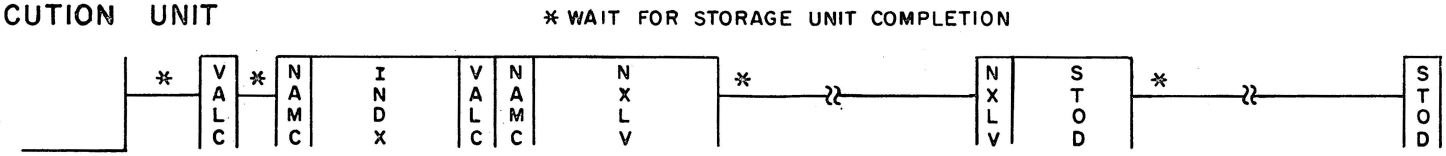
index load value.

VALC; NAMC; INDX; VALC; NAMC; NXLV; STOD;

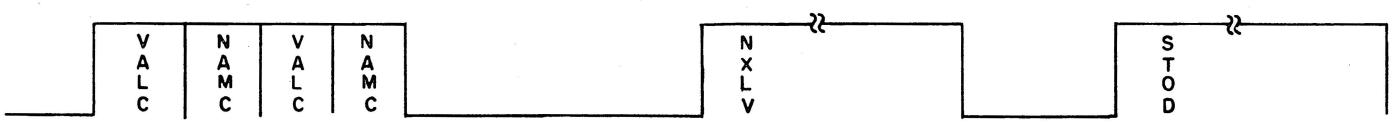
PROGRAM UNIT



EXECUTION UNIT



STORAGE UNIT



40141

Figure III-1-2. Division Overlap

alerts the program control unit to look at the next instruction to set the context of the Name Call.

The appearance of the Index allows the Name Call to be concatenated with the Index operator. The address calculation is performed and the address and operator are placed in the storage unit's queue. A micro operator indicating a concatenated Name Call is placed in the execution unit queue. The Index operator is now placed in the execution queue.

The second Value Call is then processed by the program unit and the address and operator are placed in the queues. During this time, the storage unit has been busy with the first Value Call. The program control unit now has the second Name Call and has finished its concatenation investigation and subsequence address calculation. It then places the address and operator in the storage unit queue and passes another pseudo operator to the execution unit. The storage unit just prior to this has completed the first Value Call, which was found in the stack-buffer area of the processor, and has passed the operand to the waiting execution unit. The storage unit now goes to the next item in its queue, the first Name Call reference. The execution unit investigates the control information of the first Value Call and places it in the top-of-stack location. The execution unit is now waiting for the Name Call reference which will be found local in the stack buffer and thus will be transferred by the storage unit. At the same time, the program control unit has placed the Index Load Value in the execution unit queue. The storage unit finds the first Name Call reference to be local in the stack buffer and places it in the execution unit queue. The execution unit now begins the Index function and the storage unit goes on to the second Value Call and then on to the second Name Call.

The execution unit subsequently accepts both of these calls through its queue and begins the computation involved with the Index Load Value instruction. When the execution unit supplies to the storage unit the address for the fetch, the memory reference is initiated and the execution unit holds until the return of the value. Upon return of the value, the execution unit places the data and the descriptor in the storage unit. These units will remain in sync until the store is completed. The program control unit has as this point proceeded to the Branch instruction, but prior to this, a temporary hold was placed on the pipeline since the Index Load Value operator has caused the execution unit queue to go full. The hold was released as soon as a slot became available and the program control unit went on to the branch. The branch point is calculated and presented to the program buffer control for local test. If the branch point is within the portion of program code held in the program buffer, the local pointers are readjusted and processing continues. If the code at the branch address is not available in the program buffer, then a main memory reference is initiated by the storage control unit.

Memory Overlap

In the preceding example, all reference data were found in the local

buffers. However, the pipeline processing technique is efficient even when none of the required data is found local. This efficiency is illustrated by the following example. As shown in figure III-1-3, the program control unit progresses without interruption through the entire sequence of code. All three Value Calls are found to require main memory fetches. The execution unit expends much of its time waiting for the first two Value Calls to be transferred into its queue by the storage unit. The third Value Call, however, completely masks the multiply time. Although average time was used for multiply in the diagram, a maximum multiply would still conclude before the third Value Call arrived at the execution unit queue. The program control unit again took advantage of concatenating the Name Call and thus reduced the time necessary for the execution unit's portion of the store. The Branch instruction was again completed long before the execution unit reached this point in the program.

A. PROGRAM BUFFER

The program buffer provides local storage for up to 32 words of the executing program's object code. The algorithm for loading the buffer is based on anticipation rather than waiting until all code in the buffer has been processed, so that full advantage is taken of the natural idle time on the main memory bus. Because an average of 3.5 instructions (operators) are contained in each program word, program loops are often entirely contained within the buffer. Therefore, in many cases, branching may take place without a main memory reference for the new program word. (A branch which may be made with no main memory reference is referred to as a "local branch".)

In addition, by fetching the code for the program buffer in multi-word blocks, the number of memory accesses required is significantly reduced.

As shown in figure III-1-4, the IC-memory storage area of the program buffer is arranged in four blocks, with eight 60-bit words in each block. Each eight-word block is further divided into odd and even segments. For example in block 0, words 0, 2, 4, and 6 constitute the even segment, and words 1, 3, 5, and 7 constitute the odd segment.

Buffer Word Format

When fetched into the CPM, each word of object code consists of six eight-bit syllables, a tag of 011 (which identifies the word as containing object code), and an odd parity bit. Because operators vary in length and because they are packed in main memory without regard for memory word boundaries, to ensure operator integrity, separate parity bits are generated on each syllable of the program code prior to entry of the word into the buffer. The parity is checked as the syllables are used.

The 60-bit program buffer word consists of six eight-bit syllables of code, six syllable parity bits, three tag bits, an overall parity bit, and two error-check bits.

Reading from the Program Buffer

To read a location in any of the local memory areas in the processor, it is necessary only to enter the

STATEMENT

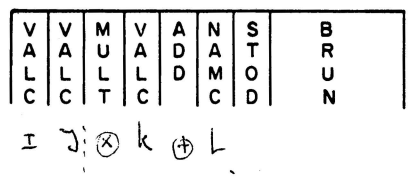
L:=I⊗J +K

L kIJ=+X

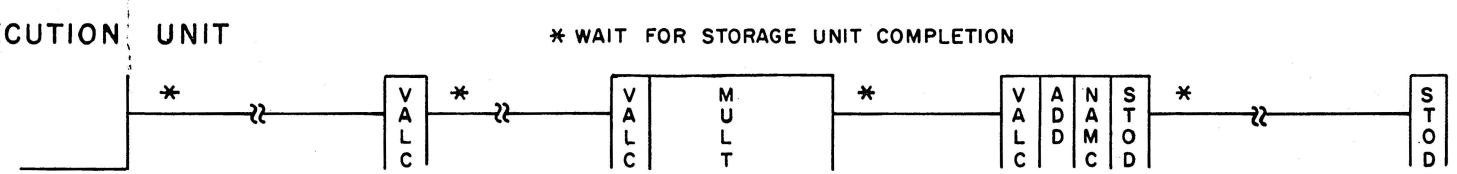
COMPILER CODE

VALC; VALC; MULT; VALC; ADD; NAMC; STOD;

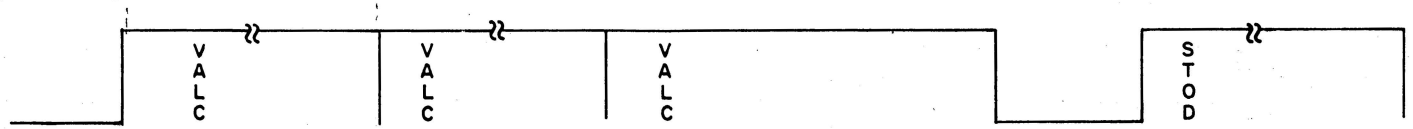
PROGRAM UNIT



EXECUTION UNIT



STORAGE UNIT



40142

Figure III-1-3. Memory Overlap

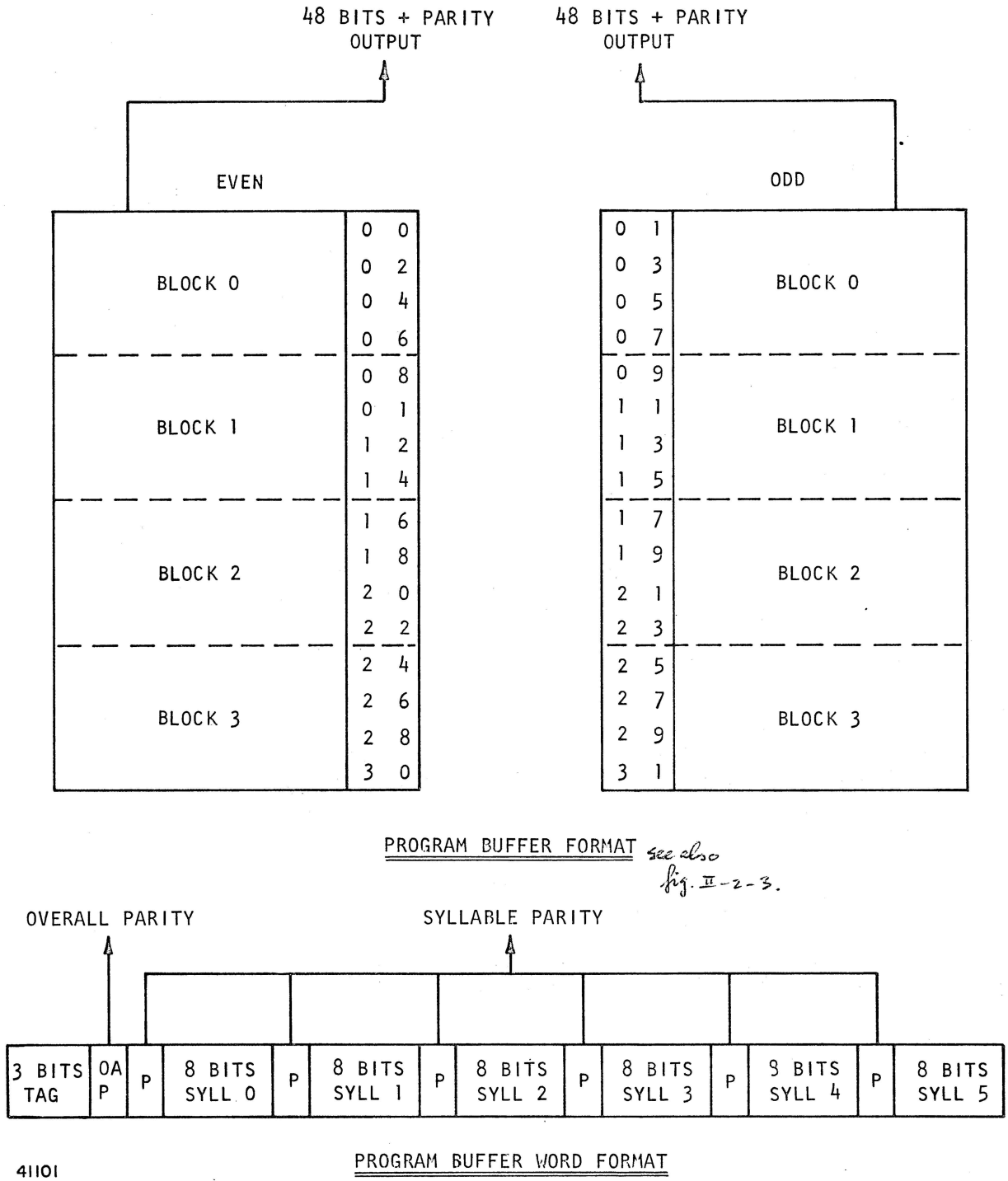


Figure III-1-4. Program Buffer Arrangement

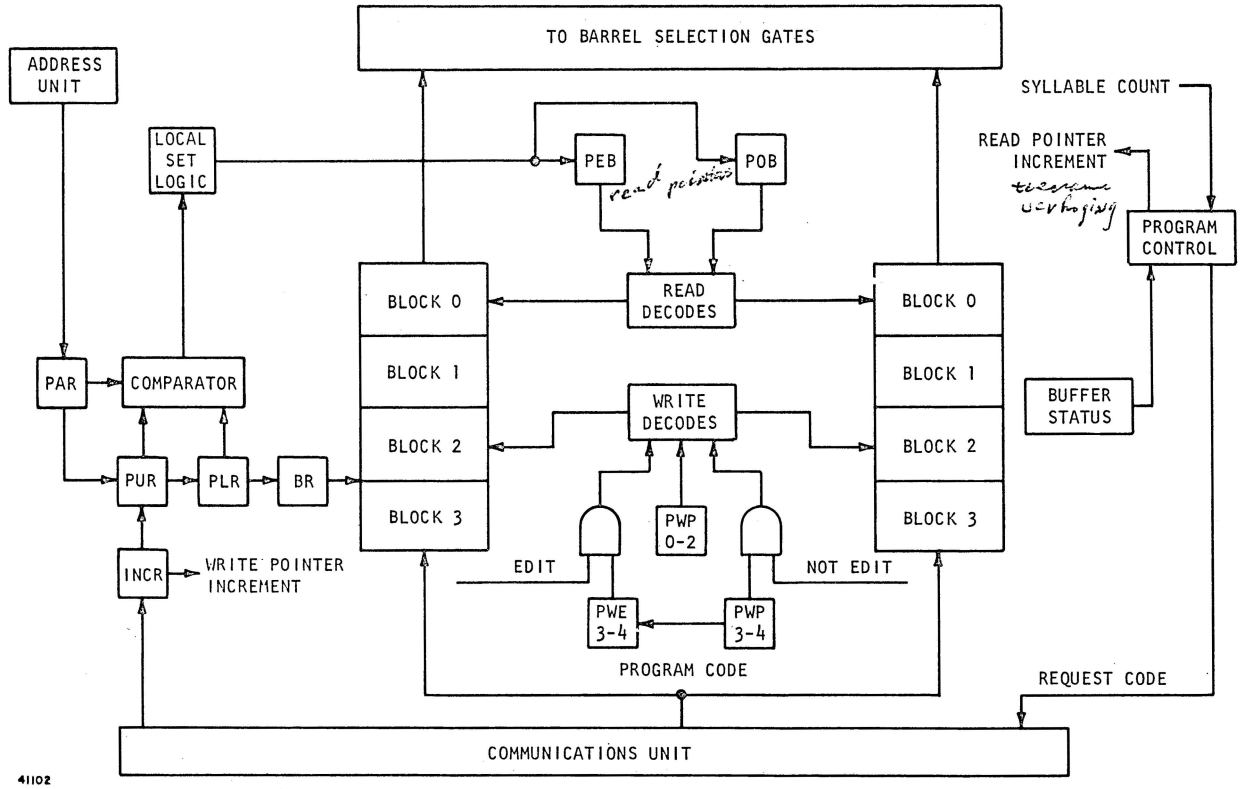


Figure III-1-5. Program Buffer Unit

address into the appropriate read pointer register. (Refer to figure III-1-5.) The decoding logic then selects the addressed word and gates the contents of the location to the storage area output as long as the read pointer contents remain unchanged. Thus the contents of the addressed location are always available on the output, and the output can be used as needed.

In the program buffer, two words, one from an odd segment and the other from an even segment, are addressed simultaneously. This is accomplished by using two read pointers. The program odd buffer (POB) is the read pointer for the odd segments, and the program even buffer (PEB) is the read pointer for the even segments.

The two words read simultaneously from the buffer storage come from consecutive addresses in main memory. One of the words contains the beginning of or all of the next operator to be preprocessed, and the other word is the word which was fetched from the next-higher memory address. The second word may or may not contain syllables of the desired operator as the operators are of variable length and are not restricted to memory word boundaries.

Writing into the Program Buffer

When a word of code is ready for entry into the buffer storage, it is written into the buffer word location (word 0 thru 31) pointed at at by the five-bit program write pointer (PWP). (See figure III-1-5.)

The two high-order bits are decoded to select an eight-word block, and the three low-order bits are decoded to select a word location within the block. The actual write cycle is initiated by triggering a single-shot circuit. Each time a valid data word is written into the buffer storage, the count in the program write pointer is increased by 1.

The absolute memory address of the next word to be placed in the program buffer is held in the program upper register (PUR). This count is advanced each time a word of code is received from main memory.

The program lower register (PLR) contains the absolute memory address of the first word of the block that has been in the program buffer for the longest time. When the CPM is started, the buffer is filled from word 0 thru 31. Thereafter, each word fetched normally overwrites the oldest word resident in the buffer.

When the processing of the last two words of program code in the buffer begins, the program-buffer logic requests the communications unit to fetch the next eight words of code from main memory. The PLR contents are then increased by eight.

The branch pointer (BR) identifies with two bits which block of the buffer contains the word whose address is in the PLR. The BR is counted up each time the PLR address is changed.

Branching

Whenever a branch is executed, the branch address is entered into the program address register (PAR). Then the PAR contents are compared with the contents of the PU and PL to determine if the branch is local. If the branch address is between the PU and PL addresses, the code is in the program buffer.

For local branches, the read pointers are updated by determining the offset of the branch address from the PLR setting. This offset, together with the BR contents, can then be used to provide the block and word address of the code in the buffer.

If the branch is not local, the program buffer is declared empty, and the address in PAR is transferred to the PUR, then to the communications unit as the address for the fetch.

Edit Mode Operation

The operation of the program buffer is altered during table edit mode. When the processor executes a Table Enter Edit operator, it in effect branches to a block of code referred to as an edit table. An edit table consists of a series of special operators used to edit data. One pass is made thru the table each time the table is used. The last operator in the table is an End Edit operator. At the completion of the table pass, the processor returns to the operator following the Table Enter Edit operator in the program code string.

Some of the program code string is maintained in the buffer during the table pass to facilitate an orderly return after the edit mode operators have been processed. Therefore, during edit mode, two blocks of the buffer are used for edit operators and two blocks are assigned for keeping a portion of the program code. The program write edit

pointer (PWE) flip-flops replace the two high-order bits of the PWP during edit mode operation. These PWE bits are then counted in a manner that allows only the two blocks assigned to the edit operators to be loaded during edit mode. The two high-order bits of PWP remain unchanged and are used to reestablish the write pointer address on completion of edit mode.

On entry into edit mode, the PL address is adjusted to account for the blocks of program code which will be overwritten by edit operators, then the updated PL address is saved. On exit from edit mode, the PL address is reentered into PLR. Then the PL address, together with the block (BK) count, is used to set up the PU address. (The block counter shows how many blocks of program code were in the buffer prior to entry into edit mode.)

Vector Mode Operation

Another change in operation occurs when the processor encounters vector mode operators. Vector mode operators facilitate the repetitive execution of an operator or a group of operators on all items in an array or a group of arrays. When the vector mode operators appear in a single word of code, the logic simply forces an automatic one-word branch backwards until all items have been processed. When the vector mode operators extend beyond the boundaries of a program word, advantage is taken of the automatic local-branch-point detection.

PROGRAM BARREL

The program barrel, shown in figure III-1-6, is a shifting mechanism used for aligning and extracting the program operators from the two words of code read from the program buffer. The program barrel consists of one selection stage and one shift stage. The selection stage receives as an input the two program words which are being read from the program buffer.

Selection Gating Stage

The selection gates align the two words read from the program buffer so that the word in which the beginning (most-significant portion) of the next operator appears is placed in the more-significant word position, and the other word in the less-significant word position. When the odd-even flip-flop (OEF) is set, the odd word is placed as more significant. When OEF is reset, the even word is placed as more significant. The OEF flip-flop is complemented each time the last syllable of a word is processed.

Only the eight most-significant syllables of the two words are provided as an output from the selection gates. The four least-significant syllables of the word in the less-significant position are not required for decoding the operator and are stripped off at this point. The eight output syllables include at least the most-significant syllable of the next operator to be processed and the two syllables which follow the most-significant syllable. All operators can be decoded from this information.

Barrel Shift Stage

The shift stage is used to extract the most-significant syllable of the next operator and the two following syllables. These three syllables are then forwarded as the input to

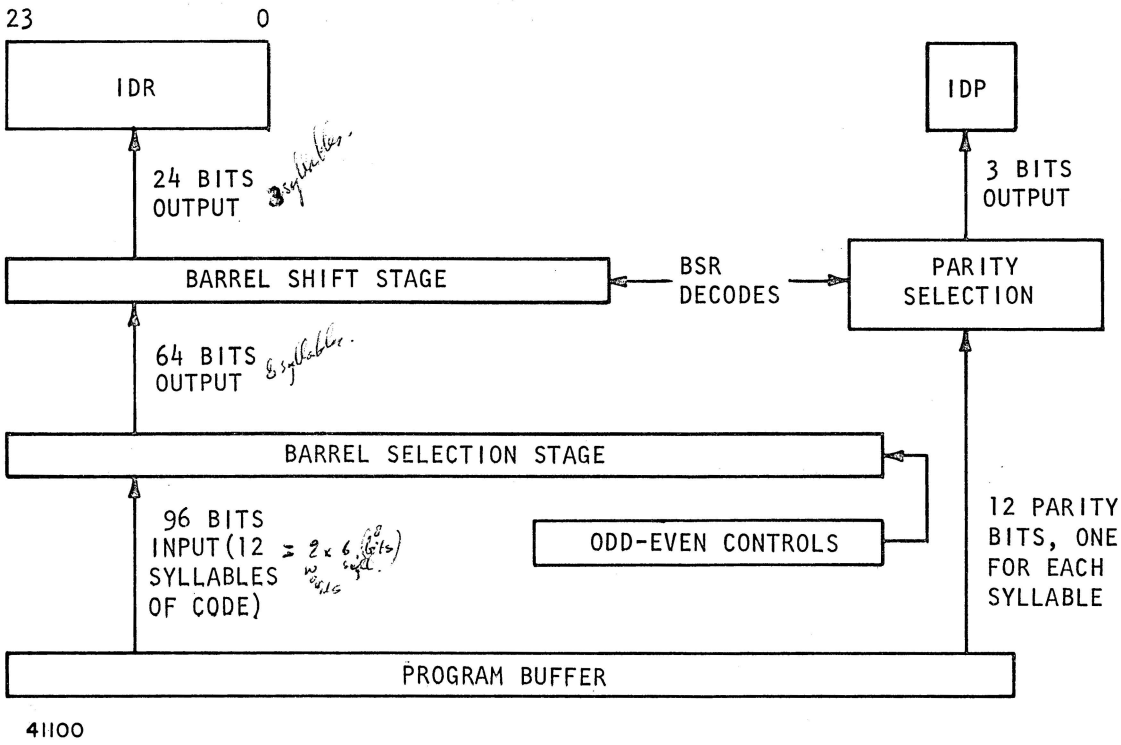


Figure III-1-6. Program Barrel

the instruction decode register (IDR). In effect, the three syllables are left justified and then extracted. The three syllables required are selected by decoding the contents of the barrel select register (BSR). The contents of BSR provide the syllable position of the beginning of the operator being extracted. The BSR count is increased by one when the 24 bits are extracted. As soon as the operator has been decoded, the BSR count is advanced further if the operator contains more than one syllable. Each time a BSR count cycle (from 0 thru 5) is completed, the end of a program word has been reached. Then the OEF flip-flop is complemented. When a branch occurs, both the BSR and OEF are force-loaded to properly identify the location of the new code in the program buffer. The BSR is located in the program control unit and is updated under control of that unit.

Syllable Parity Checking

Separate gating is provided to extract from the program barrel the syllable parity bits associated with the three selected syllables of program code (see figure III-1-6). The 12 parity bits from the two words read from the program buffer are applied to the parity gating, which uses the OEF and BSR contents to select the three desired bits. These parity bits are loaded into the IDR parity register (IDP), at the same time as the operator syllables are loaded into the IDR.

Processing of LT48 Operator

Most of the operators consist of three or less syllables. However the LT48 operator has a one-syllable operator code in one program code word and a 48-bit literal value contained in the following program word. The execution of this operator consists of placing the 48-bit literal value on the top of the stack. Because only 24 bits are output from the program barrel, the 48-bit literal value is not available at the barrel output. However the LT48 operator code is passed thru the barrel, and the 48-bit literal value is available at the program buffer output. Therefore the value is passed to the execution unit directly from the program buffer output. This is accomplished by applying both the odd-word and even-word outputs from the program buffer to the execution unit's data input register (the EWR), then complementing OEF and using the result to select the literal value for loading into the EWR. Then, so that the next two words of program code are selected during the following operation, the BSR is reset to 0 and OEF is complemented once more.

PROGRAM CONTROL UNIT

The primary tasks of the program control unit (PCU) are to decode the program operators and to partition the object code into a series of micro operators which are placed in the appropriate queues for execution. Figure III-1-7 is a simplified block diagram of the PCU and shows the important operational flow and control interconnections.

Two major registers in the PCU are directly in the operator pipeline and constitute two of the processing stations in the pipeline; these are the instruction decode register (IDR) and the instruction execute register (IER). The IDR is the preprocessing or "look-ahead" station of the PCU. The IER holds the operator when it is being divided into micro operators and is the major execute register of the PCU.

Instruction Decode Register

As a "look-ahead" station, the IDR is used to decode the program operator and to set up conditions for PCU execution of the operator in the IER. The operator remains in the IDR until the PCU processing of the preceding operator is complete, then the operator in the IDR is passed on to the IER and the following operator is loaded into the IDR. The 24-bit IDR is loaded with the output of the program barrel and, by its decodes, in conjunction with the IER decodes, controls the output of the program barrel. The processing functions of the IDR include the initiation of stack adjustments to provide the proper configuration of operands in the execution unit at the start of each operator, requesting access to other units within the CPM when the operator decode indicates that communication with other units is required, examination of the operator following each Name Call operator to determine if the Name Call and the following operator can be concatenated, and initiation of the appropriate timing sequence for IER execution of the operator.

REGISTERS ASSOCIATED WITH THE IDR

The control registers associated with the IDR are the instruction decode parity (IDP) register, the program index next (PN) register, the next syllable (NS) register, and the barrel select (BS) regis-

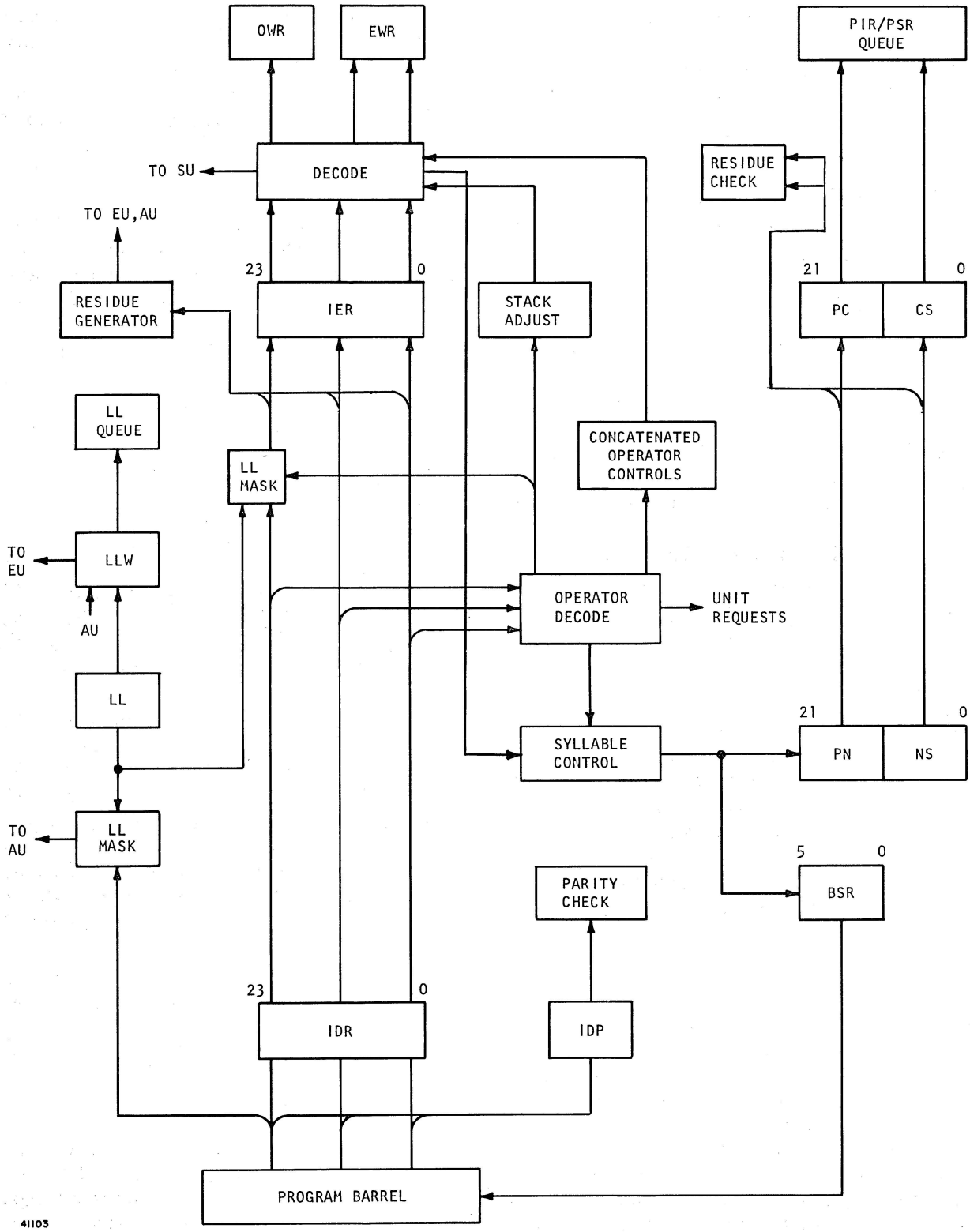


Figure III-1-7. Program Control Unit

ter. The IDP register contains the three syllable parity bits associated with the syllables contained in the IDR and is used in checking IDR parity. The PN and NS registers contain the program index and syllable counts for the operator contained in the IDR. The program index value, when added to the contents of the program base register (PBR), provides the absolute main memory address of the program operator. The syllable count identifies the starting syllable position in the memory word of that operator.

The barrel select register, which identifies the syllable position of the first syllable of the next operator to be placed in the IDR, is updated in accordance with the decodes from the IDR and the IER. The contents of the barrel select register controls the output of the program barrel as previously described.

IDR DECODES

Decoding of the operator in the IDR provides two types of decode signals: control decodes, which are used for updating the contents of the registers associated with the IDR, and operational decodes, which are used in the preprocessing of the operator in the IDR.

Control Decodes

When processing of a new segment of program code string begins, the contents of the program base register, which is maintained in the address memory area of the address unit, are updated to provide the base address for the code segment being executed. The memory address of each operator in the segment is maintained as an index to this base. This program index and the associated syllable count are passed along with the operator in the CPM pipeline until the execution of the operator is complete. If some interrupt is encountered in the pipeline, the memory address of the operator is thus available for re-execution or error reporting purposes. The program index and syllable counts are established in the PN and NS registers. The contents of these registers are updated by IDR and IER decodes as part of the preprocessing of each operator. The syllable count in the NS register is updated along with the BS register contents. Each time an NS register cycle, which is a count from 0 thru 5, is completed, the PN count is increased by 1.

The control decode signals from the IDR are also used to update the contents of the BS register. The BS register contents are then used to control the alignment of the next operator coming out of the program barrel. The contents of the BS register are upcounted by 1 each time the IDR is loaded. For monosyllabic operators, no additional update is required. For multi-syllable operators which will be held in the IER for only one machine cycle, additional updating occurs at the beginning of the next machine cycle. However, for those operators which will be held in the IER for several cycles, additional updating occurs at various times, but always quickly enough so that the next operator may be in the IDR for at least one machine cycle before transfer to the IER.

Operational Decodes

The operational decodes of the IDR contents are used primarily for stack adjustments, for concatenation investigation, and for gaining access to other units as required.

To facilitate the issuing of micro-operators which provide for the proper operand configuration in the execution unit at the beginning of each program operator, the PCU must predict what top-of-stack operands will be left in the execution unit at the completion of each operator. This prediction is maintained in the stack-A-operand (SKA) and stack-B-operand (SKB) flip-flops in the PCU. When an operator is in the IDR, the control logic determines what the initial operand configuration for that operator must be. If the contents of SKA and SKB indicate that the top-of-stack operands in the EU will not be in the proper configuration, then IDR operational decodes are passed to the IER which inserts stack-adjustment micro operators as required.

The purpose of the NAMC operator is to place an IRW on the top of the stack. However, if the operator following the NAMC requires that the address couple in the IRW be evaluated to derive the memory address, the NAMC operator is concatenated with the following operator, so that the address couple in the code string can be converted directly to an address. The concatenation occurs whenever NAMC is followed by an Enter operator, any index operator, or any store operator. The IDR operational decodes are used to detect when concatenation may occur.

Because the NAMC operator contains two syllables, the third syllable in the IDR with the NAMC operator is the operator code of the next operator in the code string. When concatenation is possible, the operational-decode signals set a control flip-flop to denote the action.

The IDR decodes are also used for access requests. If the decoding of the operator in the IDR shows that a fetch or store operation is required for execution of the operator, the IDR requests use of the storage unit, so that when the operator is in the IER, the PCU may queue a request for the required operation. In a similar manner, if data is to be provided directly by the PCU to the execution unit along with the micro operators, or if variant information is to be loaded into the K and L queues, the IDR decode signals request use of the execution write register (EWR) for the PCU.

Instruction Execute Register

The 24-bit instruction execute register is the main PCU processing register. The IER is loaded from the IDR each time the IER completes the PCU processing sequence of the current operator. As the next operator is loaded into the IER and decoded, the IER issues any required stack-adjustment micro operators to the execution unit. Then the IER issues the required micro operator sequence to the EU. All micro operators issued are placed in the OW register in the PCU. The contents of the OWR are then written into the execution unit operator queue or, if the execution unit is waiting for work, are passed directly to the execution unit. The micro-operator codes issued to the execution unit are eight bits in length. Simple program operators may require only one micro operator, while complex program operators require a series of micro-operators to complete the operator functions. In many cases, the operator code of the program operator is issued directly to the execution unit as a micro operator.

The primary timing control signals for PCU processing are developed by a down-counter, which is preset to the proper configuration from IDR decodes. During processing, the IER decodes issue a micro-operator on each machine cycle. Issued along with each micro operator are variant codes and, in the case of literals, data. Variant information and data provided by the PCU are loaded into the EWR and then queued for EU use. Often the variant information supplied to the execution unit is taken directly from the second and third syllables of multi-syllable operators. When a micro-operator requires a fetch or store of data for execution, the IER decodes cause the address to be queued for storage unit action.

When the last micro-operator of a program operator sequence is issued by the IER, the PCU sets a special bit in the OW register which informs the execution unit that this micro operator completes a program operator sequence.

Many of the micro operators issued by the PCU can be executed in only one machine cycle, others require multiple cycles for execution. When micro operators requiring several machine cycles are executed, the execution unit operator queue may become full. In such cases, PCU operation is suspended until space is available.

As previously stated, the program index and syllable information remains with the operator throughout the execution of the operator. To accomplish this, the contents of the PN and NS registers are transferred into the PC (program current) and CS (current syllable) registers at the same time as the associated program operator is transferred from the IDR to the IER. Then, each time a micro operator is issued, the contents of the PC and CS registers are queued along with the micro operator for execution unit use.

Preprocessing of Value Call and Name Call Operators

The Value Call and Name Call operators are the most frequently used operators in the B 7700 operator set. Therefore, special processing features are provided to facilitate one-cycle execution of these operators.

The two-syllable Value Call instruction (VALC) requires that the 14-bit address couple in the instruction be evaluated to provide an absolute address from which data are fetched and placed on the top of the stack for EU use.

The two-syllable Name Call instruction (NAMC) indicates that the address couple in the instruction may be used to form an IRW, which is then placed on top of the stack. However, if the NAMC operator is followed by an operator which would require evaluation of the address couple to derive an absolute address, then the NAMC is concatenated with the following operator and the address couple is evaluated immediately. NAMC is concatenated when the next operator in the program code string is any of the following: ENTR, INDX, NXLN, NXLV, STOD, STON, OVRD, and OVRN. If a NAMC cannot be concatenated, an IRW containing the address couple is placed on the top of the stack for EU use.

The 14-bit address couple in the NAMC and VALC instructions consists of a lexicographic-level field (LL)

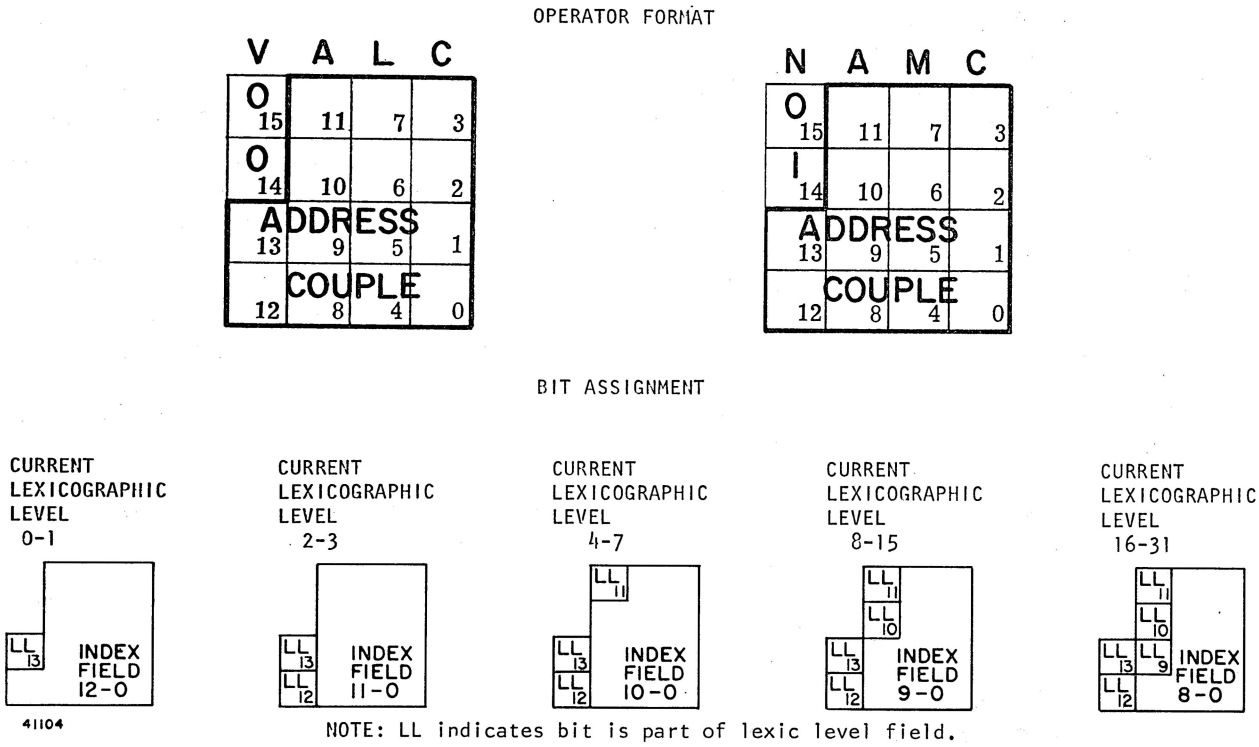


Figure III-1-8. Address Couple Bit Assignment

and an index field (I). As shown in figure III-1-8, the length of each of these fields varies with the current lexic level of the active program. The LL field ranges from one to five bits in length and contains only as many bits as are required to define the current lexic level. The remaining bits are the index field. (The bits of the LL field are in inverse order so that the least-significant bit of the field is located in the most-significant bit position of the address couple.)

To facilitate preprocessing of VALC and NAMC, the PCU presupposes that every operator in the code string is either a NAMC which can be concatenated or a VALC. Therefore, as each operator is loaded into the IDR, it is assumed to contain an address couple which must be converted into an absolute address. If subsequent decoding reveals that no address-couple conversion is required, or that there is no address couple in the operator, the conversion is terminated with no loss in processing speed.

When any operator is transferred from the program barrel to the IDR, bits 21:5 of the barrel output, which contain all possible bits of the LL field of the address couple (if the operator is a VALC or a NAMC), are gated with a lexic-level mask in addition to being entered intact into the IDR. The lexic-level mask gates the LL field bits of the address couple and inhibits any bits which are part of the index field. The lexic-level mask is set

up by decoding the current program lexicographic level, which is maintained in the lexic level (LL) register. The output of the LL mask, then, is the bits of the LL field if the operator is a NAMC or a VALC. In any event, these bits are loaded into the five LS bits of the program read pointer in the address unit. If no request for use of the AU is pending, the contents of the display register addressed by PRP are read out of the address memory area and entered into the display read register. The LL field, now in PRP, is also written into the lexic level write register (LLW) for possible later use.

By this time, the decoding of the operator, now in the IDR, is complete. If the operator is either a VALC or NAMC, it contains an address couple and the LL field is not transferred to the IER with the rest of the operator, but is stripped out of the operator code by use of the lexic-level mask. Therefore when a NAMC or VALC operator is transferred into the IER, bits 20:13 of the IER (the largest possible index field) contain only index bits from the address couple. These bits are now applied directly to the address-adder selection gates in the address unit for use as one of the inputs to the adder. The other input is the base address now contained in the display read register. Therefore, the adder output is the absolute memory address described by the address couple.

For a Value Call, the output of the address adder is gated into the input register of the storage unit, along with the PCU request for a fetch and the EU-data-queue address reserved for the requested data. This information is then queued for storage-unit processing. The same operation is performed for a concatenated Name Call which requires a fetch operation. (NAMC concatenated with an index-type or Enter operator requires a fetch operation.) For a name call-store combination, the address and a request for a store are queued for storage-unit action, but the store is not performed until the EU supplies the data to be stored.

For an unconcatenated Name Call, the output of the address adder is ignored, and an IRW is built in the EWR. The address couple for the IRW is formed by combining the index value in the IER with the LL field, which was saved for this purpose in the LLW register.

ADDRESS UNIT

The address unit (AU) contains the logic necessary for the calculation of absolute addresses, both directly as in Value Call and indirectly as in pointer update for string operators. As shown in figure III-1-9, the functional parts of the address unit are the display write (DW) register, the display read (DR) register, the address adder, the address-storage area, and the read and write pointers (PRP, ERP, and DWP) for the AU local storage. The address unit is not directly in the processing pipeline and is therefore not queue driven. It is an autonomous unit only to the extent that a write cycle into the address storage area need only be initiated and not completely controlled by the initiating unit.

The local address-storage area comprises 48 locations, each location having 20 bits for storage of an absolute address and two bits for storage of address residue. The lo-

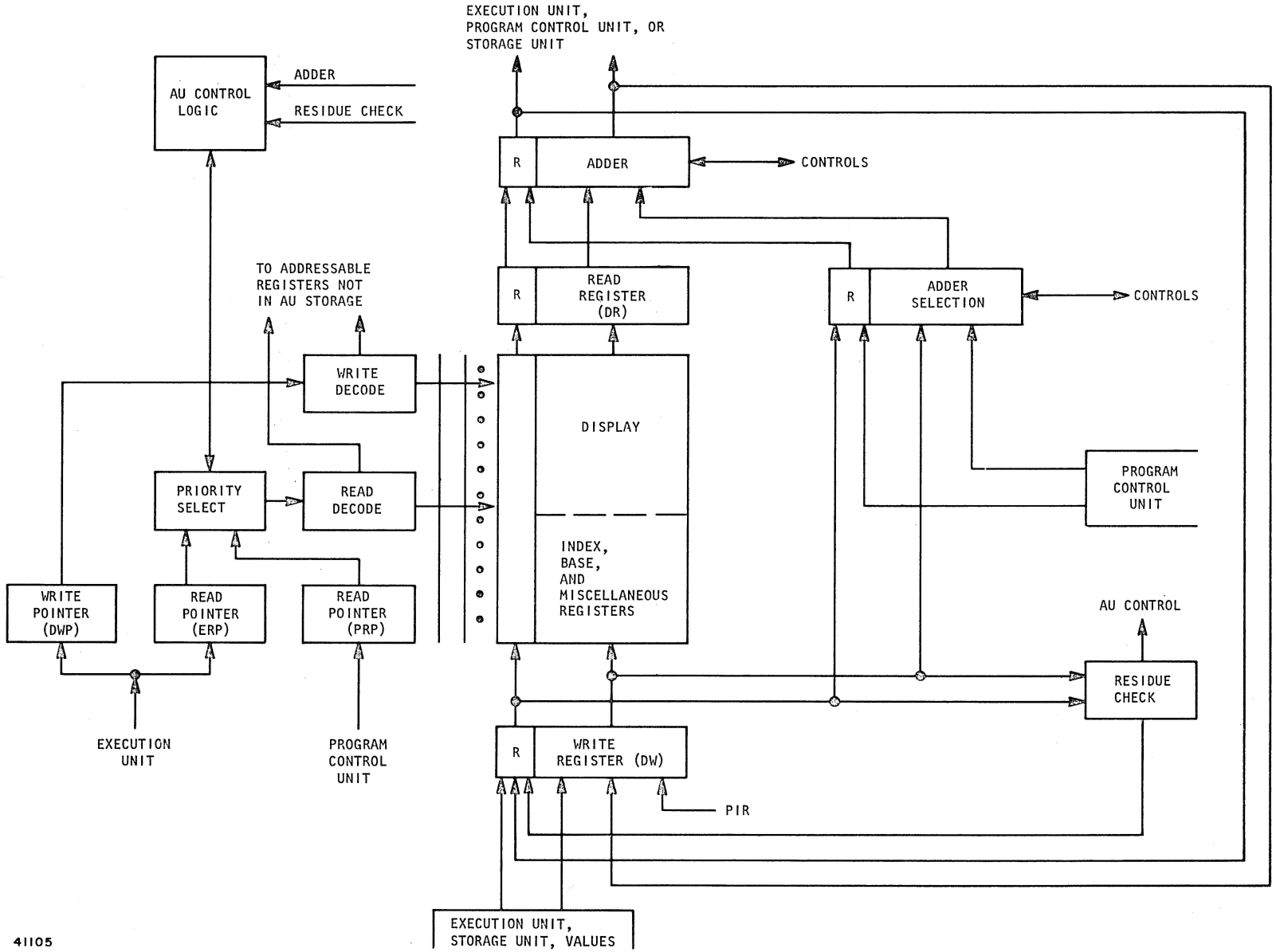
cal storage contains the 32 display registers used in relative addressing within the active program stack. Each display register in use contains the absolute address of the MSCW for a different lexicographical level. The display registers are numbered in order from D0 thru D31; the D0 register contains the MSCW address for lexicographic level 0, the D1 register contains the MSCW address for lexicographic level 1, etc. To address within any level, the appropriate D register contents are read, and then the displacement of the desired item from the MSCW IS added to the MSCW address to provide an absolute address for the required item.

The remaining 16 locations of the storage area provide storage for certain index, base and miscellaneous registers. The registers maintained in these locations are as follows:

Register Mnemonic	Register Name
SIR	Source Index Register
DIR	Destination Index Register
TIR	Table Index Register
BOSR	Base of Stack Register
BUF	Scratch Pad Location
PBR	Program Base Register
SBR	Source Base Register
DBR	Destination Base Register
TBR	Table Base Register
SNR	Stack Number Register
PDR	Program Dictionary Register
TEMP	Scratch pad location
ADZ	Alternate D0 register
APIR	Alternate Program Index Register
ALL1	All 1's Register
-	Spare location

All of the contents of the address storage are addressable by the SPRR and RPRR operators. (A total of 64 registers are addressable by these operators, 48 of which are in the AU address storage area. Those registers which are addressable but which are not contained in the AU address storage are also addressed by use of the read and write decode circuits of the address unit.)

The display write (DW) register is used to buffer information being written into the address-storage area so that the controlling logic can release immediately instead of waiting for the storage cycle to complete. Residue of the address being written is checked when the address is contained in the DW Register. The DW register is also used as one of the two adder inputs. In addition, it serves as an accumulator when more than one adder cycle is required. For example, in string processing, two adder cycles are required when adding the index, a constant, and a base to derive an address. All writing into the storage area is controlled by the EU, but the DW register may be loaded by the EU, the SU, or PCU when its contents are to be used as



41105

Figure III-1-9. Address Unit

an adder input. The DW register contents are written into the storage only when a write decode selects a storage location and a write strobe is generated by the AU control logic.

The DW register can be loaded in true or complement form so the adder may be used for addition or subtraction. Subtraction is used when performing limit comparisons.

The display read (DR) register is the output register of the address-storage area. The DR register contents are always an input to the address adder, although when the adder is used only to generate residue, the DR may contain only 0 bits. The DR register is always loaded with any information read from the address-storage area. When a limit comparison is required prior to a write cycle, the DR can be used to buffer the contents of the limit register so that the write cycle may be initiated while the add cycle for the limit comparison is in progress.

For write operations, the hexadecimal address of the addressable register is entered directly into the six-bit display write pointer. The write-decode circuitry then selects the appropriate storage location and the write cycle commences. For addressable registers not contained in the address-storage area, the decode logic selects the appropriate data paths to facilitate updating the addressable register.

There are two read pointers, the PRP which is used exclusively by the PCU, and the ERP which is used exclusively by the EU. Because either the EU or the PCU may request a read operation by loading the appropriate read pointer, priority-selection logic is required to resolve access conflicts. In general the PCU has priority, but must release the AU after one cycle. The EU has priority when the PCU is in a hold condition and can maintain control of the AU until it has finished any required operations. Access to the AU is granted by the control logic. Like the write pointer, the read pointers are loaded with any address of an addressable register. If the address is in AU local storage, a read cycle will ensure, and the contents of the addressed register will be subsequently loaded into the DR register. When an addressable register not in local storage is read, the contents of the register are gated to the execution unit by the read-decode logic.

The two-input address adder adds the contents of the DR register to the contents of either the DW register or a value (such as a displacement for Value Call) inserted directly through the selection logic from the PCU.

The adder output may be routed to the EWR in the EU; to the F, S, or LOSR in the stack buffer; to the MAR or SIR in the storage unit; to the PA or PIR in the PCU, or to the DW register for use in a subsequent adder cycle.

The AU control logic monitors the output of the adder to detect adder overflow, all 0 bit results, or the results of adder comparison.

The AU control logic provides the timing and control signals necessary for operation of the AU. The basic operations performed by the AU include a read cycle, a quick-write cycle, and a read-add-write cycle.

The read operation is performed in two machine cycles. During the first cycle, the contents of the addressed location are read, and during the second cycle the information is routed thru the adder, combined with any other input selected to the adder, and the result is routed to the appropriate destination. For example, in address calculation for Value Call, the appropriate D register is read, the displacement is added to the contents of the D register, and the result is sent to the SIR to be queued for SU action.

The quick-write cycle is used by the EU to update information in an addressable register. The two-cycle operation consists of a load cycle and a write cycle.

The read-add-write cycle is used to modify the contents of a address-storage location when the modification is based on the current contents of the location. The operation is performed in three cycles.

EXECUTION UNIT

General

The execution unit (EU) is the final stop in the processing pipeline. The great majority of instructions are not completed until the execution unit has processed the instruction. The execution unit is the only unit in the processor which operates on value data. It also has some control word formation and address calculation responsibilities. This unit includes storage for the two top of stack operands, A and B, and may temporarily store parts of character strings on which it is operating. The pipeline processing technique is implemented further in the EU. There exists within the EU, three distinct processing stations: an operator level, a command level, and a store level.

Input Queues

The execution unit like the program control unit is queue driven. All operations and operator associated data are placed into the queues of the execution unit by the program control unit. The value data inputs are supplied by the storage unit or the communications unit. Since the queues are implemented by memory chips, simultaneous read and write can take place. The purpose of each of the various EU input queues is described in the following paragraphs.

OPERATOR QUEUE

The operator queue (QQ) is the storage area for micro operators pending EU processing. The use of the queue allows the PCU and the EU to operate independently. Information to be written into the operator queue is placed in the order-code-write register (OWR). The PCU loads the OWR, then initiates the write cycle. The micro operator and control information in the OWR are then written into the address selected by the contents of the operator-queue write pointer (OQW). Following the write, the OQW count is advanced. The operator queue contains eight 12-bit locations; each location is loaded with the following information:

a. Bits 0 thru 7 contain the micro operator code.

b. Bit 8, when set, indicates that the A operand location will contain data at the start of the micro operator execution.

c. Bit 9, when set, indicates that the B operand location will contain data at the start of the micro-operator execution.

d. Bit 10 is the report bit, which when set, directs the EU to notify the PCU when execution of the micro-operator is complete.

e. Bit 11 is set if this micro operator is the last in a program operator sequence.

The operator queue is read only by the EU. The contents of the operator-queue read pointer (OQR) provide the queue address for the read operation. The read address is updated when the EU is executing the last routine of the previous micro operator if the queue is not empty, no hold condition exists, and the EU is not generating a micro operator. When the address is updated, the new information becomes immediately available at the output of operator queue. When the output of the operator queue is to be used, the micro-operator information is entered into the EU operator registers.

THE EU DATA QUEUE

The EU data queue (DQ) is an eight-word IC local storage area organized as four two-word groups. The purpose of the data queue is to buffer data generated by the PCU preprocessing (look-ahead) logic. Literals, Value Calls, and concatenated operators cause space to be reserved, and, in the case of literals, data to be entered. Each of the four group locations has a validity bit associated with it. When the data from a look-ahead fetch becomes available, the storage unit places it in the reserved data-queue location, and sets the associated group validity bit. The second word of each group is loaded with the second-half (least-significant portion) of double-precision operands, or, in the case of a non-present data descriptor, with the absolute address from which the descriptor was fetched. For single-precision words other than non-present data descriptors, the second word location in the group is left empty.

Writing into the data queue is performed by the PCU, the storage unit, or the communications unit. Reading from the data queue is performed by the EU only.

Data Queue Write Operation

The input register for the EU data queue is the EU write register (EWR). When a write cycle is initiated, the contents of EWR are entered into the address selected by the write pointer (DQW) and flip-flop WLSQ. The data group to be loaded is selected by decoding the contents of DQW, and WLSQ selects which of the two word locations in the group is to be loaded. (The first word of the two word group is loaded when WLSQ is reset.)

The PCU keeps track of the next group to be filled by use of the data queue assignment (DA) counter. When the PCU encounters a literal value, it loads the value into EWR, transfers the DA count into DQW, initiates the write cycle, and then advances the count in DA. When an operator requires data not contained in the program code string, the PCU posts in the storage unit operations

queue a request for the data together with the main memory address of the data and the contents of the DA counter. Then the PCU advances the DA count, thus reserving space for the requested data. When the storage unit or the communications unit obtains the requested data, the word is entered through the EWR into the reserved data-queue location.

In order that the EU may know when a group location has been loaded and whether one or two words have been written into the group location, there are separate data-queue-valid (DQV) and queue-single-precision (QSP) flip-flops associated with each data-queue group location.

Data Queue Read Operation

The EU reads the information from the data queue in the same order in which the PCU reserved the group locations. Read addressing for the data queue is accomplished by decoding the contents of the data-queue-read (DQR) pointer, which selects the group to be read, and of flip-flop RLSQ, which identifies which of the two words in the group is to be read. The contents of the word location selected by DQR and RLSQ are available on the queue output lines. To use the information, the EU simply gates the information onto the T bus. The EU will not attempt to use the information until the associated DQV bit is set.

EU LOOK-AHEAD DATA QUEUE

The control information provided with each group written into the EU data queue is also entered into an EU look-ahead data queue (EL queue). This control information includes a copy of bits 45 thru 50 of the data being entered into the data queue and a bit which indicates if the associated data-queue word is in integer form.

The look-ahead information is addressed by the data-queue read pointer, but the address decoding is such that the look-ahead information is available at the EL queue output when the preceding data-queue group is being addressed. In certain cases, the EU uses this information to determine how to generate residue on the data of the preceding group. (Residue may be generated either on the entire word or separately on exponent and mantissa.) The look-ahead information is also used to determine what commands should be called to process the associated operands.

THE K AND L QUEUES

The K queue (KQ) and the L queue (LQ), which are used in conjunction with the operator queue, contain coded variant information for use in execution of the related micro-operator. The K and L queues each contain eight 10-bit locations. The operator variant codes are eight bits in length, and in the other two bits of each location residue is maintained on the variant code. Usually the variant information supplied with a micro operator is taken directly from the second or third syllable of the program operator in the IER. Such variables could be used to identify the length of a selected field or a bit location in a word. In other cases, the variant information is generated by the PCU to alter or further define the related micro operator.

Addressing of the K and L queues for both read and write operations is accomplished by use of the operator queue read and write pointers. Therefore, each time an entry is made into the operator queue, an entry is also made into the K and L queues although this information may

be all 0 bits. Likewise, when an operator-queue location is addressed by the read pointer, the associated locations in K and L are also addressed.

The first 16 bits of EWR are used as the input register for the K and L variant information. The variant-residue (VNR) register in the PCU supplies the residue for K, but the residue for L is written directly into the queue from the PCU residue generator.

The outputs of the K and L queues are entered into the K and L registers in the EU. Like the operator register, the K and L registers may be loaded directly from the PCU if the queue is empty.

THE PIR AND PSR QUEUE

The PIR and PSR queue (PQ) contains the PIR and PSR values associated with the micro operators in the operator queue. (The PIR and PSR, together with the contents of PBR, identify the absolute address and starting syllable position of the associated program operator.) The PIR queue is loaded with the contents of the program current and current syllable registers. The PIR queue is written into at the same time as is the operator queue and uses the same write pointer (OQW) for addressing.

A separate read pointer PQR is used for the PIR queue. The PQR is counted in a manner which causes the PIR and PSR information associated with each micro-operator to be available at the output of the PIR queue when the micro operator is at the store level of execution in the EU.

Usually the output of the PIR queue is not used. However, when the EU starts interrupt processing, the output of the PIR queue may be gated into the PN and NS registers to provide the PSR and PIR values associated with the program operator being re-executed.

THE LEXIC LEVEL QUEUE

This queue (LLQ) contains either the lexicographical addressing level of a particular operator in the operator queue or the value which was in the PCU display read pointer (PRP) when the associated micro operator was issued by the PCU.

The current lexicographical level is written into the queue any time the PCU processing sequence for an operator includes timing count Phase 9 (PH9). The PRP read pointer value is entered into the queue when either a Name Call (NAMC) or Value Call (VALC) operator is processed by the PCU. For all other operators, no entry is made in the queue.

The information in the LL queue is then available at the output of the queue for use by the EU during the execution of the related micro-operators.

Writing into the LL queue is accomplished by use of the operator queue write pointer (OQW), and reading from the queue is accomplished by use of the PIR queue read pointer (PQR).

Major EU Data Circuits

As shown in figure III-1-10, the EU contains five full-word data registers, three data buses, local storage areas, and adder and shifting

mechanisms. These data circuits are described in the following paragraphs.

ADDER

The most obvious major item required by the execution unit is a fast adder. The adder used comprises a single-carry-save adder which drives a carry-propagate adder. The adder fully propagates all carries each time it is used, employing a high-speed carry look-ahead technique.

In order to accommodate all single-precision mantissas, the mantissa portion of the adder is 39 bits wide with an extension for multiply and divide. This portion of the adder has three inputs. During multiply, two of the inputs are used to insert selected multiples of the multiplicand, and the third input inserts the accumulated product. By this method, six bits of the multiplier are calculated on each adder pass. The separate, seven-bit exponent adder is included to facilitate single-precision, floating-point arithmetic.

The adder is also used to perform double-precision arithmetic. This is accomplished by buffering the double-precision operands and intermediate results in the local EU memory area.

The adder provides an efficient means for executing logical operations. The single-sum and single-carry outputs, which are developed for each bit position by the first stage of the adder, are selectively gated to derive the logical AND, OR, and EXCLUSIVE OR functions.

The adder output is checked for residue or continuity errors. If either input to the adder is invalid, a continuity error ensues. If the sum is incorrect, a residue error is detected. Separate residue is maintained on the exponent and mantissa during floating point arithmetic operations.

SHIFT MECHANISM

The barrel switch is a 48 bit wide, end-around shifting mechanism. It is capable of shifting by any number of bits from 0 to 47, to the left. The barrel output is gated with enabling logic, which allows any number of bits from 0 thru 48, out of the barrel, from the left or the right.

The barrel has three shift stages. The first stage shifts the input data by multiples of 12, the second stage shifts the input by multiples of 3, and the third stage shifts the input by multiples of 1. The multiples are selected by the contents of the shift register in which two bits are used to define the selected multiple for each shift stage. The shift amount, coded in successive bit values of 1, 2, 3, 6, 12, and 24, may be entered into the shift register in either true or complement form. When this amount is loaded in complement form, a right shift effectively occurs; when loaded in true form, a left shift is executed.

The barrel output gating enables a field of any selected length or the entire word to be extracted from the barrel. The allow register contents determine the number of bits to be extracted at the barrel output. Additional gating allows this field to be selected either right or left justified.

Because all barrel stages are implemented with logic gates, the barrel

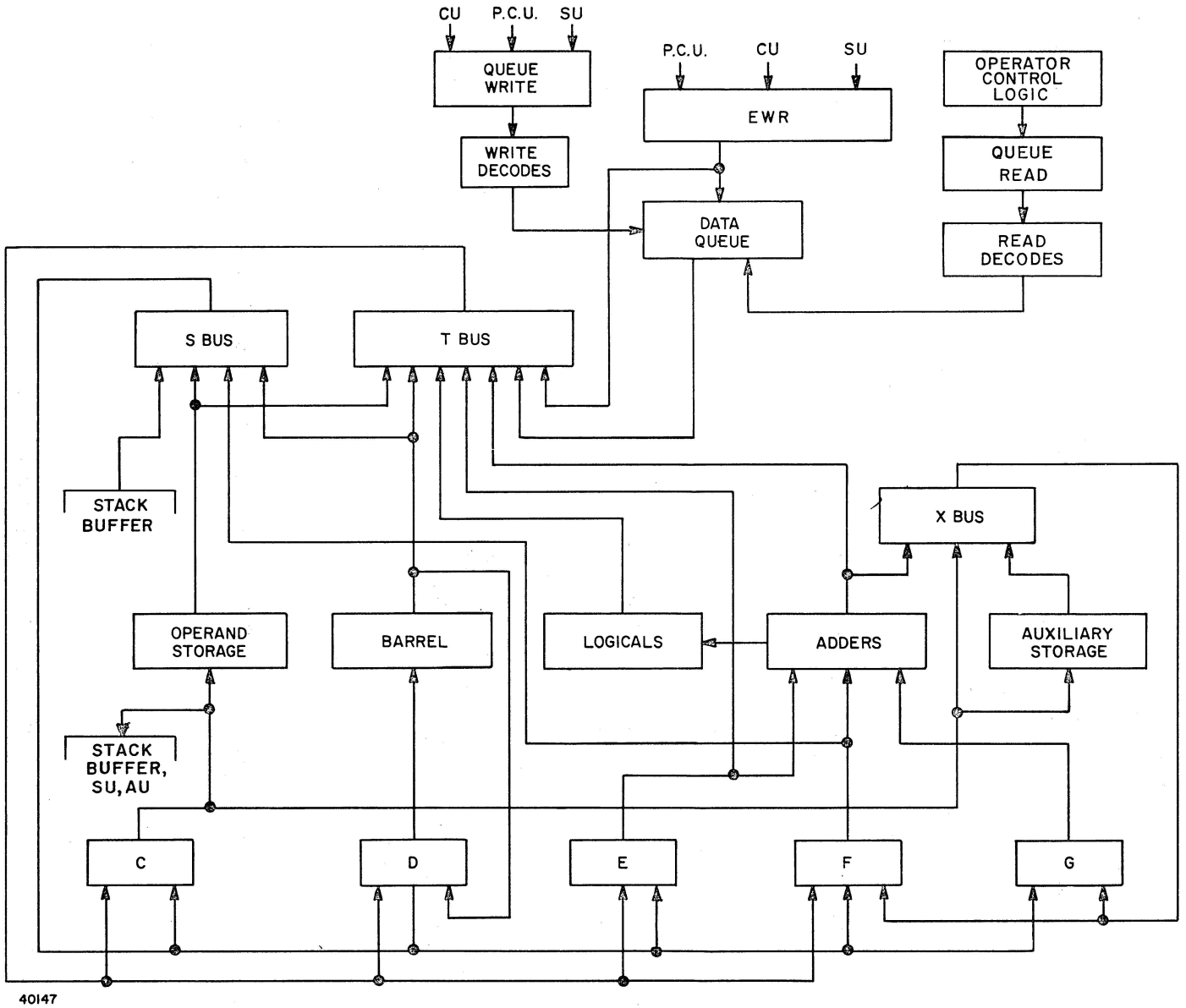


Figure III-1-10. Execution Unit, Major Data Circuits

shift mechanism may be used repeatedly, during each machine cycle. If either the allow register or the shift register was not loaded when the barrel is used, the output of the barrel is declared invalid and a continuity error is subsequently detected. Residue is maintained on the contents of both the shift and allow registers. A residue error in the contents of these registers results in the detection of an EU miscellaneous error (WPI bit in fail register is set). Residue is also maintained on the data transferred through the barrel. If the barrel output has a residue error, an EU residue error condition is detected.

LOCAL MEMORY

The execution unit also includes a local memory for storage of the top of stack operands, character strings and intermediate results. Memory chips are again used for this purpose. Parity is maintained on each word in the local memory, and the parity is checked each time a word is read from local storage. The local memory is split into two parts in order to be able to read two words simultaneously. The larger part is called the operand storage; the smaller part is called the auxiliary storage. The memory space is allocated as follows:

Operand Storage (52 bits wide)

1. A operand: Two alternate double-precision locations.
2. B operand: Two alternate double-precision locations.
3. R space: Four words.
4. W space: Four words, which is sometimes used as two alternate locations of two words each.

Auxiliary Storage (49 bits wide)

1. Intermediate exponent: one word.
2. Three times register one word.

The time required to cycle the memory chips is comparable to the time required to propagate the worst case carry through the adder, so the memory fits neatly into the overall timing scheme. To maintain the fastest possible single precision execution times, single-precision operands are buffered in the adder and barrel input registers rather than in operand storage.

The purpose for having alternate locations for A and B is to be able to form a new A or B without destroying the original input operands. This is desirable for diagnostics and error recovery. Two values for the A and B operands are maintained in storage during the execution of each program operator. The contents of the A and B operands at the start of the operator are held unchanged in the initial allocation area, while the current allocation area is successively loaded with the current values of the operands. When the final result of the program operator is obtained, the allocation is interchanged, so that the final copies of the current operands are designated as the initial configuration for the next program operator.

The adjustment of the memory allocation is controlled by use of the EU allocation flip-flops. These allo-

cation flip-flops also facilitate rapid execution of stack related operators.

When three top-of-stack operands must be maintained in the EU, as during execution of the Rotate Stack operator, the W storage area is used to maintain the third top-of-stack item.

DATA REGISTERS

The execution unit includes five full-word data registers (C, D, E, F, and G registers). The A and B words of the B 7700 are implemented in the EU local storage, as previously described. Intercommunication between the data registers is carried out by the use of three data transfer buses (T bus, S bus, and X bus).

There is one register which loads the operand storage, and which is the EU output register for communication with the rest of the processor. This register is the C register. All data error checking, as well as parity and residue generation is accomplished when a data word is in the C register.

The D register is the only barrel input register. Any data entered in the D register is immediately shifted by the barrel and allowed out of the barrel, in accordance with the contents of the shift and allow registers. Because the D register is the only barrel input no selection gating is necessary.

Since the adder is used often, it is especially important that the delay between the registers and the adder be minimized. This is accomplished by transferring the contents of the adder-input registers into the adder with no selection gating. Therefore certain registers, namely the E, F and G registers, are dedicated as adder input registers, and actual adder selection is accomplished at the input to these registers. The E register has special significance in that it is the accumulator, while the F and G registers supply new inputs. The E and F registers also have six-bit exponent fields which are inputs to the exponent adder.

Unless the contents of these data registers are specifically "held", the contents must be transferred during every machine cycle or the word will be marked invalid. Therefore to maintain validity of the A and B operands when they are not being used for processing, the E register and T bus are often assigned to keeping the A operand valid, while the F register and the S bus are assigned to keeping B valid.

DATA TRANSFER BUSES

Placing the outputs of the adder, the barrel, the registers, and the operand storage onto buses which go to the registers (which in turn are the inputs to the adder, barrel and operand storage) is the fastest way to transfer data without using excessive amounts of hardware. The data transfer buses, which consist of multiple input OR gates, are designed to match the stack type features of the simple operators and the micro operators. This approach led to the implementation of two major data transfer buses, one serving as the top element of the "bus stack" and the other serving as the second element. In simple operations on single precision data, the A operand is on the T (top) bus, and the B operand is on the S (sec-

ond) bus. During complex operations, data segments will be moved onto the T and S buses as inputs for micro operators commands.

The processing requirements of simple and micro operators suggest particular data paths for the buses. In particular, each bus must go to those registers which give the bus access to the adder, the barrel, the operand storage, and areas external to the execution unit. Inputs to the stack go to the top position so the operand input (for Value Calls, Literals, etc.) from the data queue or queue input goes to the T bus only. Arithmetic results are left in the top position, so the adder output goes only to the T bus. The extension of the stack (stack buffer) goes to the S only to handle stepping up ("popping") the stack. When executing a monadic operator with both A and B full initially, it is desirable to load B back onto the S bus at the end of the operator, so the operand storage output is routed to the S bus. However, it is also desirable that the inputs to the "bus stack" for micro operators go to the top position so the operand storage also goes to the T bus. Field results are left in the top position, so the barrel output is applied to the T bus. Certain other paths to the buses are available in order to hold operands on the buses or to facilitate particular algorithms.

Various special paths from the buses to the registers exist to speed up multiply and divide and these special paths are also useful for scale left and scale right. There is one other special path, called the X (auxiliary) bus, which is not part of the "bus stack". The X bus is used during multiply and other extended arithmetic calculations to speed up these computations. During multiply, the X bus transfers to the adder registers a quantity equal to three times the multiplicand.

Major EU Control Circuits

Figure III-1-11 shows the major EU control circuits. These circuits include the operator and command registers; the K, L, and R registers; A, B, and W control registers, and the A, B, R, and W pointer registers.

The A and B control registers are used to maintain ready access to the control information provided with the A and B operands. Such information includes the tag bits, the sign bits, and a bit which indicates if the related data is in integer form. When three top-of-stack operands are maintained in the EU, the W control register is used for the control bits of the third operand. If during execution of an operator, the control information may be altered, the contents of these control registers are declared invalid until the updated information can be loaded from the C register. The registers are initially loaded as the data becomes available to the EU.

The pointer registers (AP, BP, RP, and WP) are used for addressing a particular segment of a word contained in the EU local storage. To extract part of a word from EU local storage, the pointer contents are decoded to provide a word address and a digit location within the word. The word portion is used for reading the proper word from operand storage and the digit information is used in setting up the shift and allow registers so that the desired segment can be extracted by use of the barrel. For repeated operations, the contents of the pointer

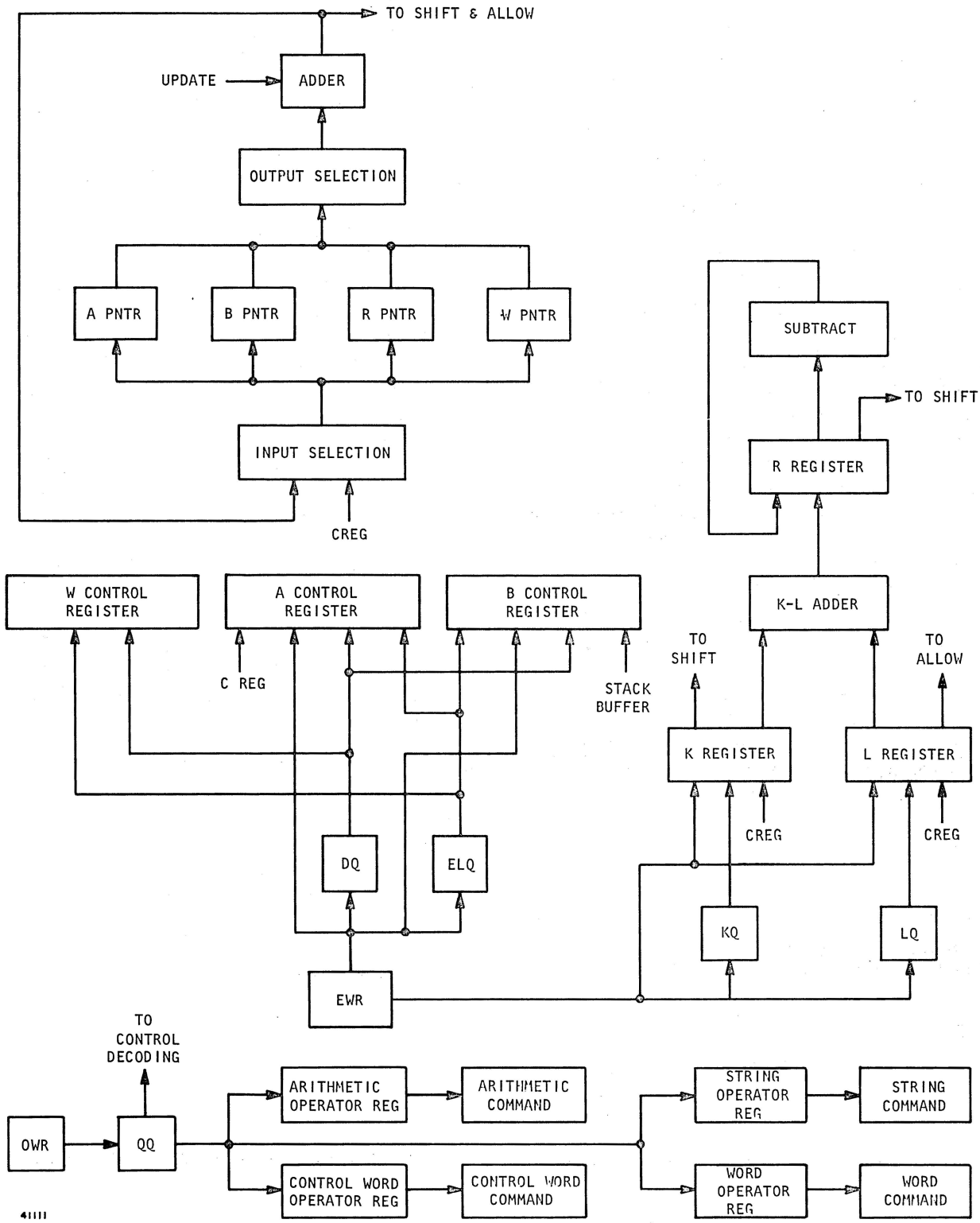


Figure III-1-11. Execution Unit, Major Control Circuits

register is updated by use of a special adder, and the updated contents are then loaded back into the pointer for use on the next cycle.

The K and L registers receive the variant information from the K and L queue, or, for dynamic field operators, from the C register. The information in K and L may be decoded or, for field operators, may be routed thru the K-L adder, to provide information for extraction of a desired field. The output of the K-L adder is K-L+1, therefore in field and bit operators when K contains the starting bit, and L contains the length, the K-L adder output gives the shift amount required to right justify the desired field in the barrel, and the L register contents provide the input for the allow register.

When the K register contains the scale value for scale operators, the amount in K is transferred through the K-L adder to the R register. The R register is a repetition counter. For scale operators, after each cycle, the count in R is decreased by use of the subtract logic and the difference is loaded back into R until the operation is complete. The R register also serves to convert the binary values of the K and L registers and of the K-L adder output to the bit values used in the shift register. (Effectively, this conversion consists of dividing the binary input by three, then placing the remainder as the two least significant bits of the quotient.) The conversion is accomplished by the decoding logic on the input of the R register.

The remaining major control registers, the operator registers and the command registers are the primary control registers of the EU. The operator registers are used to decode the micro operators and the command registers are used to generate the data transfer signals for execution of the micro operators. The use of these registers is described in the following paragraphs.

Basic EU Operation

As previously described, the EU does not operate on basic program code (program operators), but rather it executes micro operators forwarded to it by the PCU. One or more micro operators are executed in completing each program operator. The major advantage in this method is that many program operators are quite similar in execution, so that the same micro operator may be issued for a group of program operators.

The micro operators are grouped by function into four families: the arithmetic family, the string family, the control word family, and the word family. To simplify micro operator decoding, separate operator and command registers and separate timing counters are provided for each micro-operator family. The fifth and sixth bits of each eight-bit micro operator code identify the family to which the micro operator is assigned.

When a micro operator is read from the operator queue, it is entered simultaneously into each of the four operator registers. However, the fifth and sixth bits of the eight-bit micro-operator code are not loaded into the operator registers, but are decoded to enable the use of only the appropriate operator register. At this point, the timing counter for the selected family is also enabled. The micro operator is

decoded, and the first of a sequence of operator commands is generated. One coded command is entered into the appropriate command register at each count of the timing counter. Each command is held in the command register for only one machine cycle, during which the command is decoded.

The use of command registers allows groups of sub-commands to be generated as efficiently as possible. The sub-commands are data transfer and control signals. The command register generation of these data transfer signals determines the use of the data registers and data bus networks in the EU. The signal which loads a command into the command register is also used to route the information from the data buses to the data registers. On the next effective clock, the command decode causes the necessary transfers of the data units thru the selected logic (such as the adder or the barrel shifting mechanism) to the data buses. Because the command registers are loaded with a new command on each effective clock, on a given clock, one command is loading the buses while the next command is selecting the buses to the registers. The only requirement for compatibility between the two commands is that the resulting bus configuration of the first command is compatible to the input bus requirements of the next command. (The "bus configuration" refers only to the presence or absence of data on each bus.)

For example, if data in the E and F registers were to be added by one command and the next command required that the sum be routed thru the barrel shifting mechanism, then the command decode of the first would generate a data transfer signal which would route the adder output to the T bus, while the signal which loads the second command would simultaneously cause the contents of the T bus to be routed to the barrel input register (D register). The next effective clock would then load the second command into the command register, while the sum on the T bus would go to both the D register for use in the second command and, automatically, to the C register for error checking and for storage of a copy of the result of the add operation. This automatic routing of the results of each command to the C register is the "store-level" action required for each command.

When the contents of the C register are to be transferred (unloaded) to a location other than EU local storage (such as transfer to the SU, the AU, or some EU register), the transfer is controlled by a coded store-level command entered into the store (ST) register.

The execution of each micro operation entails the processing of a series of one or more commands. However, although the enabled command register is loaded with a new command on each effective clock, not all of these commands are issued by the decode of the micro operator in the operator register. Each command register is allowed to reload itself with successive commands when a natural unit of actions requires a certain sequence of commands. Such a sequence of commands is called a routine. At the end of the routine, the operator register is allowed to load the command register and the timing count is advanced. The command load for the last command of a routine generates a final-command signal, which enables the operator-

register output and the timing count.

For each micro-operator, when the last command generated by the operator-register decode logic is loaded to the command register, the operator-level signal, final routine, is produced. When the final-routine signal has been generated, the operator register is loaded with the next micro operator from the operator from the operator queue.

The last micro-operator of each program operator is specially tagged by the PCU. When this micro operator has been fully executed, the EU prepares for the next program operator. This preparation consists of checking for interrupts and of switching the allocation of the EU local storage areas so that the final stored results of this program operator are designated as the initial A and B configuration of the next program operator. During execution of this next program operator, this initial operand configuration will be saved, while the current A and B operands are repeatedly entered into the alternate A and B storage locations. This initial configuration must be maintained for possible use in error recovery and diagnostics.

EU Error Detection Methods

Three error-detection systems are used in the EU, namely, parity checks, residue checks, and continuity checks. Parity is used to detect errors in EU local storage and in data received from other units. Mod 3 residue is used to detect errors anywhere in the EU data paths and data registers, but not in either the EU local storage or most control registers. Also, the residue logic checks addresses sent to the EU from the AU or SU. In addition, residue is the primary means for detecting errors caused by an extra data transfer signal. Continuity checking, the use of a validity bit which indicates if the current contents of a register are valid, is used to detect missing, and sometimes extra, data transfer signals for the most commonly used EU data paths.

The implementation of these methods produced the following results:

a. The EU continuity logic provides a considerable check on the sequence of operators and control bits sent to the EU by the PCU.

b. The EU residue system detects errors in arithmetic operations, field and bit extractions and insertions, and barrel shift operations.

c. The EU parity logic detects data storage and transfer errors involving the EWR, the EU local storage, the associative memory, and the stack buffer.

d. Continuity and residue checks detect most errors involving the generation of subcommands and data transfer signals.

EU Interrupt Processing

To allow interrupts to be processed in an orderly manner, the EU, which is located at the end of the processing pipeline, is the unit which checks for interrupt conditions. As each command reaches the store level, the EU checks for syllable dependent interrupts. At the end of each program operator, the EU checks the contents of the fault condition (FC) register to determine if an external or special interrupt must be

processed. If an interrupt is awaiting processing, the EU notifies the PCU of this fact by setting the EUT flip-flop and entering coded information into the branch type (BRT) register to inform the PCU what type of processing is required. For interrupts, the PCU clears the EU queues, then executes an interrupt processing sequence which prepares the stack for MCP interrupt processing as described in Section 2 of this chapter. In general, when the EU detects that an interrupt must be processed, it obtains the required interrupt parameters (P1 and P2 parameters) and stores the parameters temporarily in the R area of EU local storage. Then, the PCU issues a micro operator sequence which causes the EU to mark the stack, to place an IRW pointing to D0+3 on top of the MSCW, to transfer P1 and P2 to the top of the stack, and then to execute an Enter operator.

Syllable dependent interrupts detected by the EU during execution of an operator are recorded by the EU in the operator delay register (ODR) and the fault condition register immediately, and the processing of the interrupt is initiated when the current micro operator reaches the store level. For certain syllable dependent interrupts, the P2 parameter must be supplied by the PCU, while in other cases, the EU itself developes the P2 parameter. If the PCU is to send the P2 parameter to the EU, the EU will inform the PCU of the required action by placing a code in the PU react register and notifying the PCU that assistance is required. The P1 parameter required for interrupt processing is always developed by the PCU from the contents of the fault condition register, as part of the interrupt processing sequence.

When the EU detects a hardware failure in the EU, the appropriate bit (bit 6) in the FC register is set, more definitive information is entered into the CPM fail register, and the PCU is notified to begin interrupt sequence immediately. In this case, the current program operator sequence is aborted and the EU is prevented from writing additional information into its operand storage area.

When the PCU detects an error in the program code, it suspends issuing additional micro operators until the EU reports that it has completed all pending processing. Then, if the EU has not caused a branch away from the portion of code containing the error, the PCU initiates the interrupt processing sequence. Of course, if the EU encounters a dynamic branch or other change in direction before the faulty code is reached, the error is not reported.

When the communications unit fetches data from main memory and encounters an error, the data is returned to the requestor with a bit indicating that an error occurred and an additional bit indicating if the data is valid. If the EU attempts to use this data, the error is reported. If the data is not used, the error is not reported. If the communications unit receives a control word in place of the requested data, then the control word is passed to the requestor. If the EU accesses this data, it will recognize the control word, initiate the interrupt processing, and use the control word as the P2 parameter for the interrupt.

STACK BUFFER

The stack is an area of memory assigned to a job to provide storage

for basic program and data references. The stack also provides temporary storage of data and job history. When a job is activated, a linkage between its stack and the top-of-stack operands (A and B) is established by the stack pointer register (S), which contains the memory address of the last word placed in the stack. The stack buffer serves to extend the stack memory area into processor local IC memory and to provide quick access for stack manipulation by the execution unit. (See figure III-1-12.)

Stack Buffer Function

The primary purpose of the stack buffer is to hold, locally, a portion of the stack environment. New stack items are entered into the stack buffer from the EU in such a manner that the last item placed in the stack is the first to be extracted. After the two top-of-stack positions in the EU are filled, loading a third operand onto the top of the stack causes the first to be pushed into the stack buffer. As entries are pushed into the stack buffer, and saturation is attained, a segment of buffer entries is autonomously moved into main memory so that the stack buffer maintains the top area of the stack memory area. Any stack adjustment to main memory is always accomplished in multi-word segments in order to take full advantage of the phased memory system. Thus, the stack buffer tends to capture the current addressing environment of the executing program stack.

In the B 7700 processing system the stack buffer can be directly addressed, within limits, as if it were actually an area of main memory. The direct addressing of the stack buffer is transparent to the programmer. Therefore, knowledge of this action is not necessary for the programmer.

Stack Buffer Operation

As shown in figure III-1-13, the stack buffer is a 32-word local storage area consisting of eight 4-word segments. Each storage word in the stack buffer consists of 54 bits. These bits comprise 48 bits of information, three tag bits, a parity bit, a data-valid bit, and a memory-error bit. The buffer may contain from 0 to 32 words of the active stack. All valid contents of the buffer have contiguous main-memory addresses. The range of main-memory addresses for the contents of the stack buffer is indicated by three 20-bit registers: the S register, the stack link register, and the stack address register.

The stack register (S register) holds the main-memory address of the top item in the active stack. If the stack buffer is not empty, this address is the main-memory address for the top item in the stack buffer.

The stack address register (SAR) contains the main-memory address for the bottom (or oldest) item in the stack buffer. When the buffer is empty, SAR is set to S+1.

The third register, the stack link register, is required because several entries at the bottom of the stack buffer may be copies of data already contained in main memory. The stack link register (SLR) contains the main-memory address for the bottom (or oldest) buffer entry which is not a copy of the contents of main memory. When the stack buf-

fer is empty or when it contains only words which are copies of main memory, the address in SLR is set to S+1.

Three pointer registers are used to identify the stack-buffer word locations which correspond to the main memory addresses held in S, SLR, and SAR. The top (TP) pointer contains the buffer address corresponding to the address in the S register, the bottom (BT) pointer contains the buffer address corresponding to the address in the SAR, and the link (LK) pointer contains the buffer address corresponding to the address in the SLR. A fourth pointer, the local read (LR) pointer, provides the buffer addresses for fetching or storing the addresses local in the buffer. The contents of these four pointers are stack-buffer word locations, which range from 0 thru 31. When adding new items to the stack buffer from the EU, entries are made into successive buffer locations. When location 31 is filled, the next new entry is placed in location 0. To prevent loss of the previous contents of a location when new entries are made, each time a new entry is to be made into the first word location of a four-word group, the words in that buffer group which are not copies of main-memory are stored to main memory.

FETCHING THE TOP ITEM FROM THE STACK BUFFER

When the top item in the active stack is required in the EU for use as the A or B operand (an action referred to as a "pop" of data from the stack), a main-memory fetch is not required unless the stack buffer is empty.

If the stack buffer is not empty, the top item in the buffer is read from the buffer and transferred to the EU, and the contents of both the TP pointer and the S register are decreased by 1. At the same time, if SLR is set to S + 1, the contents of the LK pointer and SLR are also decreased by 1; otherwise, the contents of SLR and LK are not changed. Each time a pop is performed the stack buffer checks to ensure that the address in S is not less than the address in F. If S is less than F, then stack underflow is detected. (F contains the address of the top MSCW in the active stack.)

If the stack buffer is empty and a "pop" is required, first the address in SAR and the contents of BT are decreased by 4. The SAR address is then used to fetch the top four words of the active stack from main memory. The contents of BT are used as a buffer address then counted up as each word is received. After the four words have been entered into the stack buffer, the contents of BT are decreased by four so that it again points to the bottom word in the buffer. (This action is called a "fill" operation.) With the required information now in the stack buffer, the "pop" described in the preceding paragraph occurs.

ADDING A NEW ITEM TO THE CONTENTS OF THE STACK BUFFER

When the A or B operand in the EU must be placed in the active stack to establish the proper operand configuration for execution of the next operator, the operand is transferred into the stack buffer. Because this

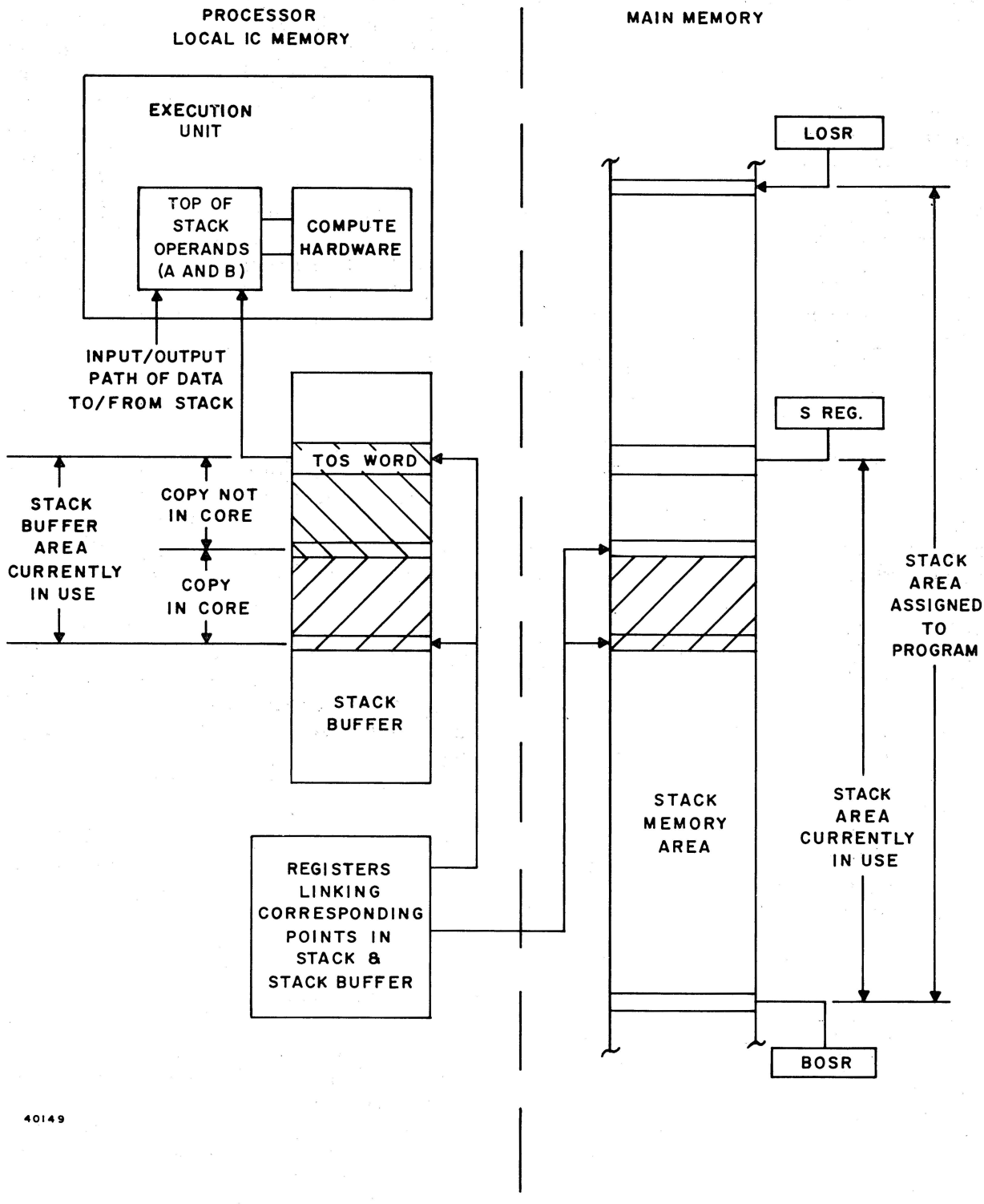


Figure III-1-12. Stack Buffer and Stack Memory Area

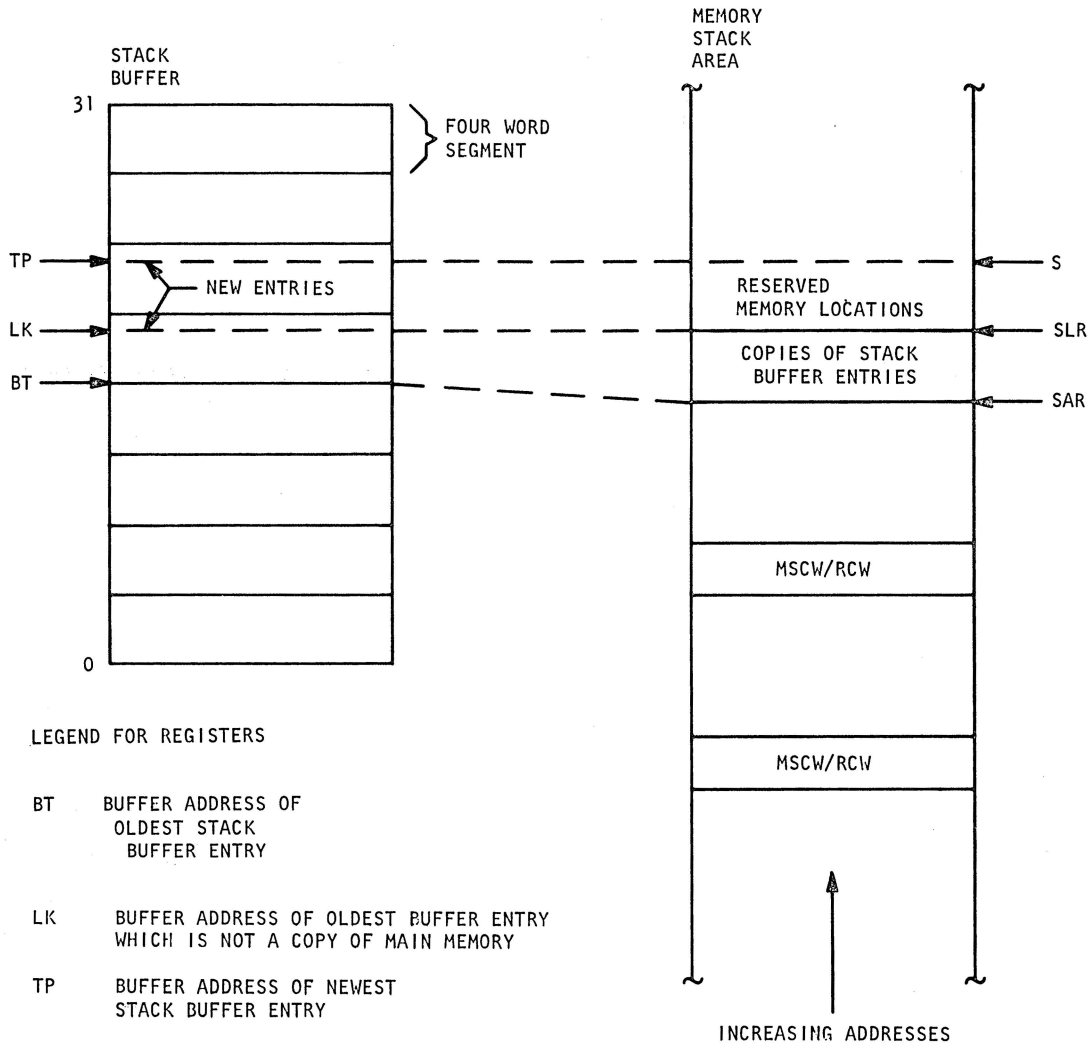


Figure III-1-13. Stack Buffer Operation

operand is added to the top of the active stack, the operation is referred to as a "push" of data into the stack. A "push" of data from the EU into the buffer will not cause a store to main memory unless the stack buffer is full.

If a "push" is required when the stack buffer is not full, the TP pointer and S register contents are increased by 1, then the operand to be added to the stack is transferred from the EU to the stack buffer and written into the stack-buffer location identified by the contents of TP.

If the stack buffer is full and a push is to be performed, first from one to four buffer words are stored to main memory, and then the data in the EU may be placed in the stack buffer. (This operation is referred to as an "empty" operation.) Note that buffer entries with main memory addresses between SAR and SLR are not stored, as such entries are copies of main-memory contents. Thus, the number of words stored to main memory is four minus the difference between SLR and SAR. (SAR can never be more than three below SLR when a push is required.) After the store of data from the stack buffer, the number of words stored is added to the SLR and LK contents, then SAR is set to SLR and BT is set to LK, but the contents of S and TP remain unchanged. After the "empty" operation, the push of new entries into the stack buffer from the EU may proceed as described in the preceding paragraph.

Each time a push is to be performed, the stack buffer compares the contents of the S register to the contents of the limit of stack register (LOSR). The LOSR contains the address of the highest usable location in the active program stack. If S equals LOSR, a stack overflow interrupt is generated.

FETCHING AND STORING OF LOCAL INFORMATION

The stack buffer maintains up to 32 words of the top items in the active stack. The storage unit can fetch from or store to any point in the buffer that is within the local addressing environment. Therefore fetch requests made by the EU for any of the items "captured" locally in the stack buffer can be fulfilled without a main memory access. When updated information is stored to an address which is local, to avoid conflicts, the store operation is performed to both the stack buffer and main memory.

The local addressing environment includes all items between S and SAR for a store and all items between S and SLR for a fetch. To read information from the buffer, when a requested memory address is within the local addressing limits, the associated buffer address is entered into the local read (LR) pointer. The LR pointer is set from the output of an adder in the control logic area of the stack buffer. Effectively, for a store operation, this local adder subtracts the contents of SAR from the requested address, then adds to the difference the contents of the BT pointer. (For a fetch operation, SLR is considered to be the bottom limit of the local addressing environment and in such cases, the contents of SLR are used by the adder in place of the contents of SAR.) The five least-significant bits of the adder output are loaded into the LR

pointer to provide the buffer address for a local fetch or store.

STACK BUFFER PURGE OPERATION

A store to main memory is initiated when the stack buffer is purged. A purge causes all buffer words with main memory addresses between SLR and S to be stored to memory. After a purge operation, the stack buffer is marked empty. A stack-buffer purge occurs each time an SPRR, RDLK, or MVST operator is executed.

ASSOCIATIVE MEMORY

The associative memory (ASM) is a general data buffer, implemented to provide fast access to frequently used variables and descriptors which are outside the area contained in the stack buffer (see figure III-1-14). The physical layout of the ASM storage area is shown in figure III-1-15. The arrangement of the data words and associated addresses allow the data and address to be written simultaneously. Also, by placing the addresses in word one of each four-word ASM addressing segment, all addresses can be read simultaneously when checking to see if a requested address is local in the ASM. The relationship of the logic circuits used with the ASM is shown in figure III-1-16.

Write Operation

The associative memory is loaded with any item referenced by an IRW unless that item is a double-precision operand or a non-present data descriptor. Such entries include data descriptors, step index words, and single-precision operands. The data descriptors retained may include dope vector entries such as those used in multi-dimensional and segmented array implementation. When such items, requested by either the program unit or the execution unit, are brought into the communications unit, they are copied along with their main memory address into the associative memory. A future reference to the item may find it still resident in the ASM and thus can eliminate a reference to main memory.

After the associative memory is full, the oldest resident entry is overwritten each time a new item is brought into the ASM. When an item has been overwritten, it is reentered into the ASM on the next reference to the item, so that frequently used items tend to be available in the current contents of the ASM.

When information is being stored in a main memory address currently available in the ASM, the data in that ASM location is updated along with the data in the main memory location. Therefore, valid entries in the ASM are current copies of the associated items in main memory. Any store operation (store destructive, store nondestructive, overwrite destructive, and overwrite nondestructive) performed by the storage control is executed to main memory as well as to local areas if applicable. Since stores always update the contents of main memory, the contents of the ASM never need to be written into main memory. After successful completion of the local memory store, the execution unit may continue to execute operators in its queue, even though the store to main memory is not complete. This is possible because conflicts such as protected writes and accidental procedure entries will have been de-

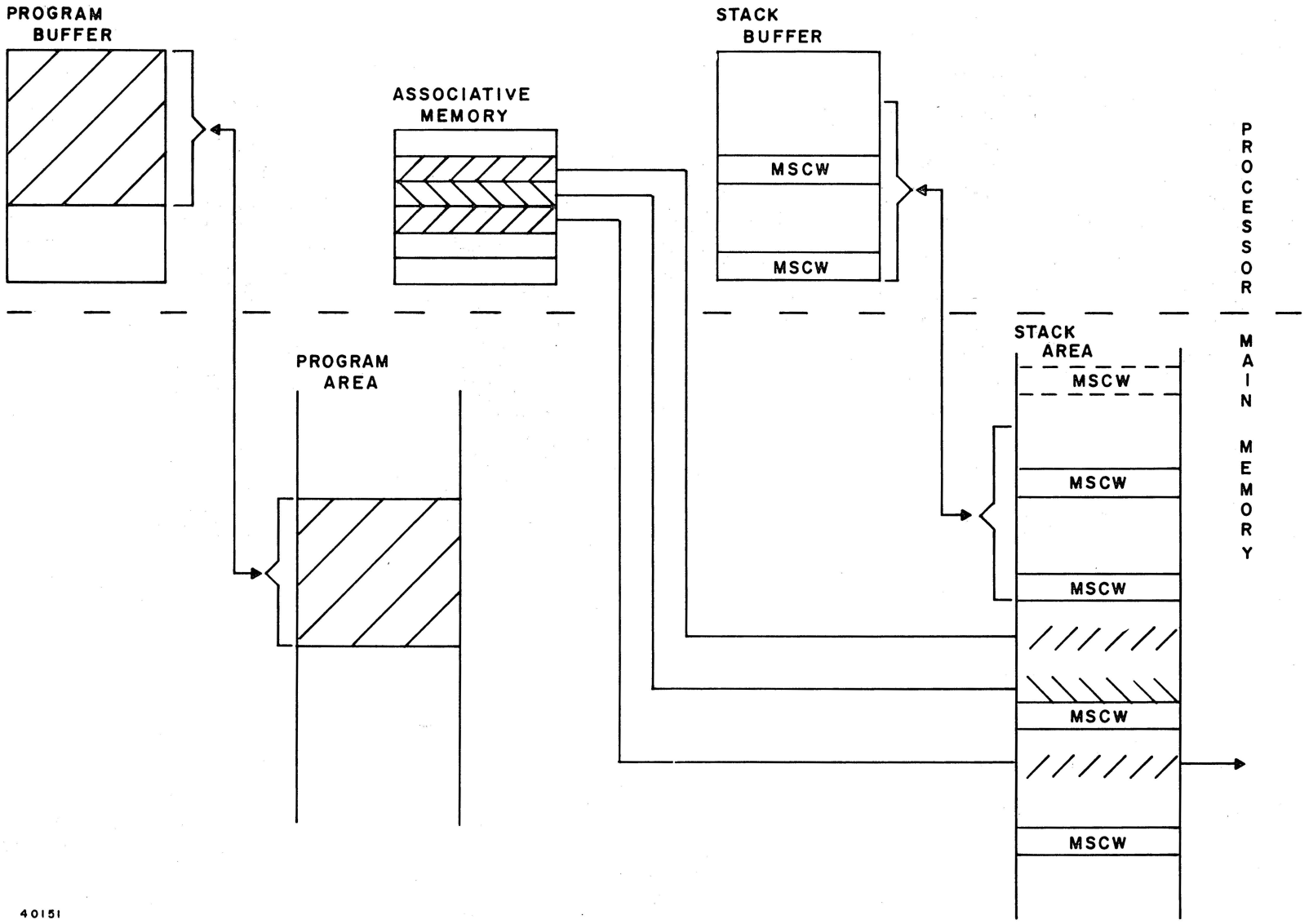


Figure III-1-14. Local Buffer System

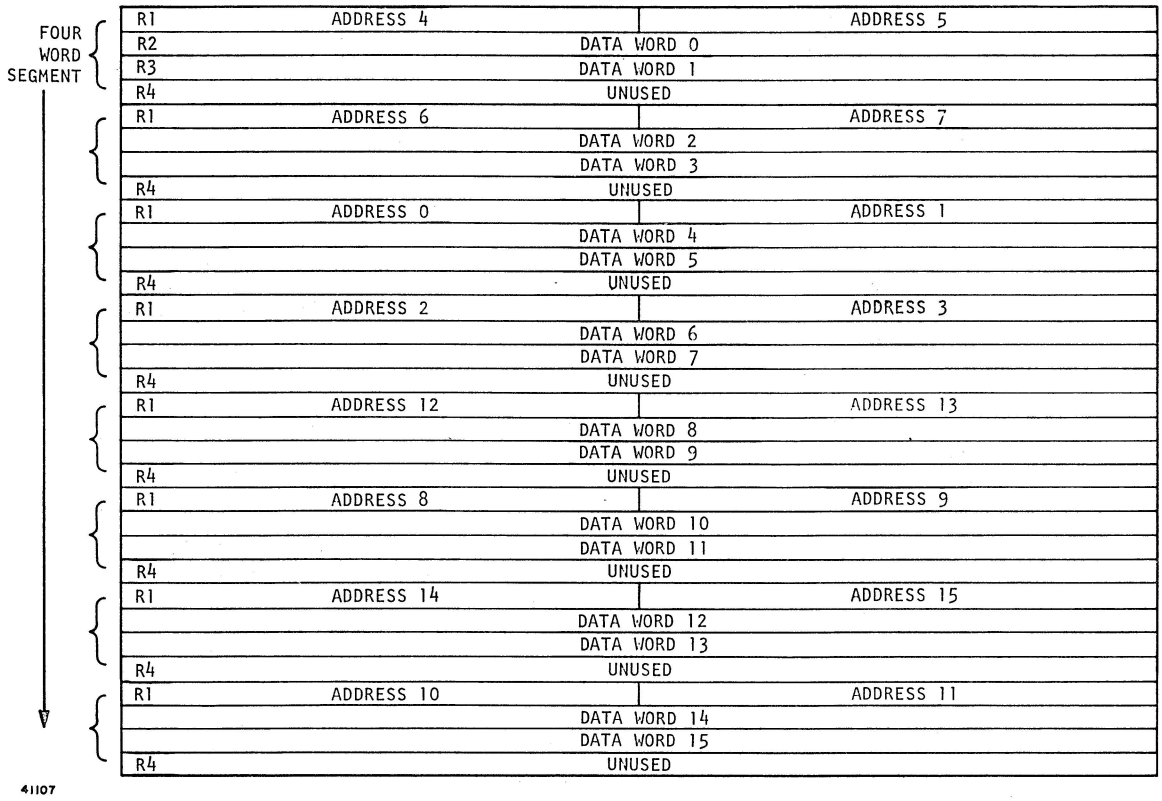


Figure III-1-15. Associative Memory Storage Area

tected on the store to local memory. The hardware can invalidate all information in the ASM when necessary, such as when entering the MCP for reallocation. A record is maintained of the validity of each word in the ASM by use of the residue bits of the address. To purge the ASM, 0's are written into all address-residue bits.

The write pointer for the ASM consists of two counters, the upper counter and the lower counter. The upper and lower counts are both advanced each time an entry (a data word and its absolute address) is written into the ASM. When data-and-address locations 0 thru 7 are being loaded, the output of the lower counter is enabled, and when data-and-address locations 8 thru 15 are being loaded, the output of the upper counter is enabled. Each time a write operation is performed, decoding logic develops simultaneous address-write and data-write signals from the contents of the enabled counter. Because the address and data are entered into different ASM segments, they can be written simultaneously.

Read Operation

When information is requested from main memory, a check is made under control of the storage unit to determine if the requested information is currently contained in either the stack buffer or the associative memory. This action of local detection

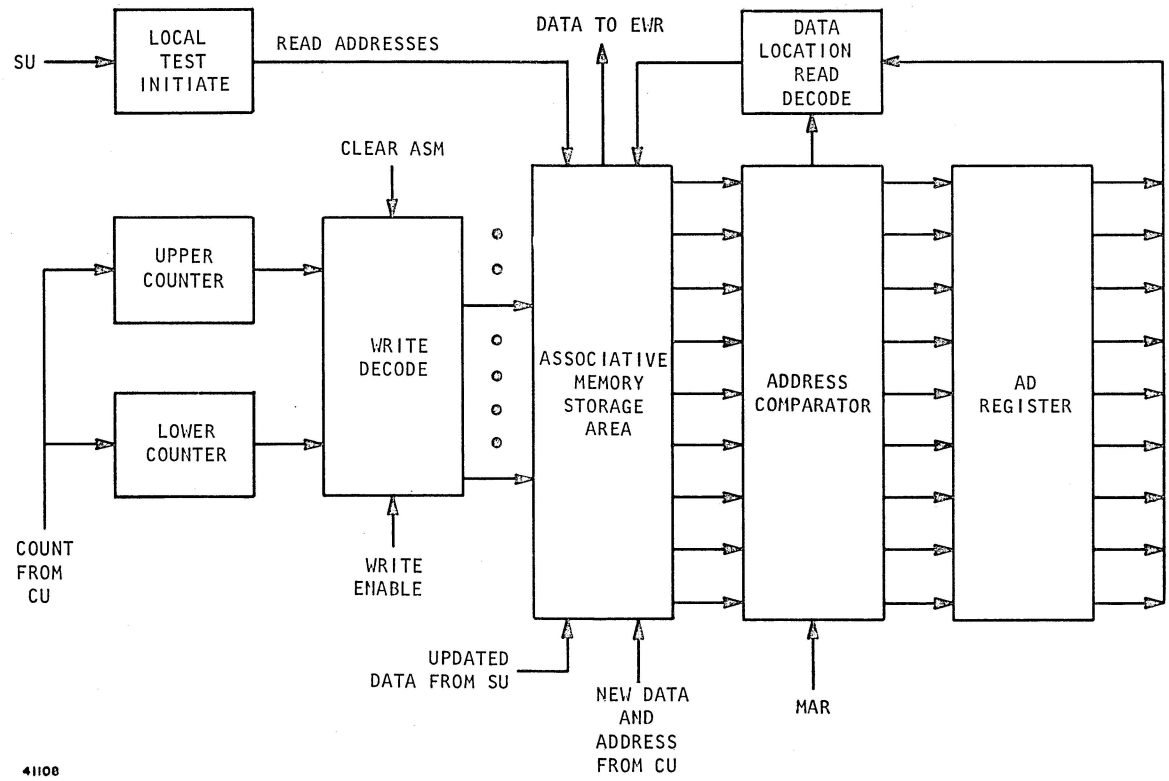


Figure III-1-16. Associative Memory

takes place as an operator and address are removed from the storage unit input queue. The operation is placed in the storage output register and the address is placed in the memory address register. The SU then begins processing the operation. The contents of the memory address register are presented, in parallel, to the stack buffer's local test and to the ASM. The control circuitry of the ASM reads all addresses in the ASM storage, then the comparator circuits execute 16 simultaneous "compare for equals". The result of each of the comparators becomes the read line for the associated data, so that a compare not only results in a signal which informs the SU that the information was found "local", but automatically initiates a read out of the proper data. This is accomplished by use of the 16-bit AD register. When the address in the MAR, including the residue of the address compares exactly with an address in the ASM, the associated bit in the AD register is set. Also, if the address compares, a local (LOC) flip-flop is set. When the local flip-flop is set, the contents of the AD register are decoded to provide a read line for the data. No word from the ASM is used unless the corresponding bit in the AD register indicates that the word is local and valid. In the event that a reference is found in both the associative memory and the stack buffer, the stack-buffer copy

is given preference, since it conceivably could be a later copy of that reference which was created by a series of functions causing push-down operations.

STORAGE UNIT

The storage unit (SU) controls the initiation of memory fetches other than program code fetches and of all store operations. The actual operations with main memory are performed by the communications unit, but the storage unit assembles the memory address and control information and for stores supplies the data to the communications unit. As part of its functions, the SU controls the checking of the stack buffer and associative memory local addresses to determine if the required operation is to be performed locally.

The storage unit is capable of performing overlapping operations. Such overlap occurs whenever a main-memory fetch is initiated. While the communications unit is fetching the requested information, the storage unit can go on to the next entry in its operations queue. If that operation references an item that is local in the stack buffer or the associative memory, then the local reference is completed in parallel with the main memory reference. The overlap is not restricted to one operation. The storage unit is free to process operations from its queue until the external reference is complete or until another main-memory reference is required. A major benefit of the overlap is realized because most of the variables used in constructing a descriptor are local. Thus, when a descriptor is being constructed during the time that the communications unit is completing a previous store operation, the time spent in main memory is effectively masked by the concurrent construction of the descriptor.

The block diagram of the storage unit, figure III-1-17, shows the general flow of information through the unit. In preprocessing the operators in the program code string, the PCU determines if a fetch or store-type operation is required for the execution of the operator. If such an operation is required, the PCU enters the available information into the operation portion of the storage input register (SIR). If the memory address for the operation can be calculated from the information in the program code string, the PCU directs the AU to calculate the address and load the memory address into the address portion of the SIR. If the address information is not available the request is queued without an address, and the address is supplied during execution by the EU. Operations and addresses in the SIR are entered into the storage unit operations queue. The current operation being performed by the storage unit is held in the storage output register (SOR). The address for the current operation is held in the memory address register (MAR). If the operations queue is empty, when a job is available in the SIR, in addition to being written into the queue, the operation is passed directly to the SOR, and the address, if available, is passed to the MAR. The storage unit data queue is used to hold the information for store-type operations. The information to be stored is always supplied by the EU.

Storage Input Register

The 29-bit storage input register is used as the write register for the storage unit operations queue. The

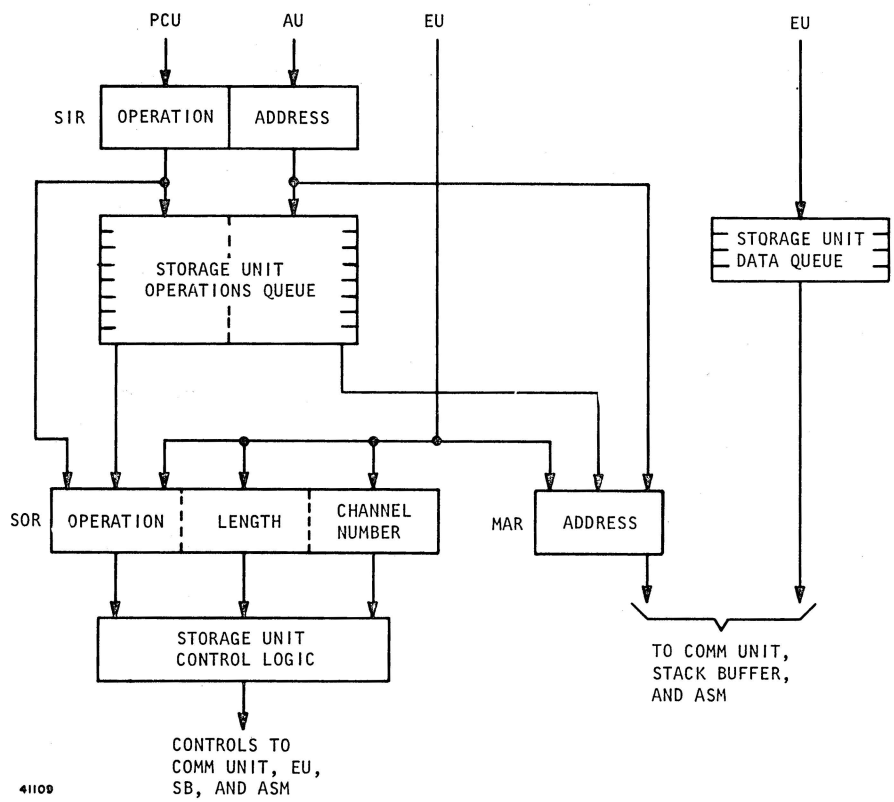


Figure III-1-17. Storage Unit

contents of the register include a 20-bit memory address field, two bits of residue on the address, two bits which identify the EU-data-queue address for fetch operations, and five bits of control information which identify the operation to be performed. The operation and EU-data-queue address are supplied by the PCU, the memory address and residue are loaded by the AU when available. If the EU is to determine the address, the address field is left empty.

Storage Unit Operations Queue

The storage unit operations queue is a local storage area containing eight locations. The operations in the SIR are written into successive locations in the queue, and subsequently read from the queue and processed by the SU in the order in which they were queued. The queue address for writing into the queue is provided by the storage operations write pointer (SQW). The pointer is counted up after each write operation. A read pointer (SQR) provides the location in the queue of the next word to be read from the queue. At least one micro operator is placed in the EU operator queue for each entry in the SU operations queue, and each fetch or store placed in the queue must be completed before the EU can complete the associated micro operator. Therefore, the storage unit opera-

tions queue can never be full when the PCU attempts to write a new operation into the queue, because PCU processing would have been suspended prior to the attempt to write by the EU operator queue being full.

Storage Output Register

The SOR contains the operation currently being performed by the storage unit. The SOR contains 21 bits of information. Five of these bits are loaded from either the queue or the EU, and the remaining bits are supplied when required by the EU. This additional control information includes a length field, a channel number field, and bits which distinguish overwrites, protected stores, and special operations. The length field identifies the number of words to be stored or fetched for EU-initiated operations. Operations from the queue have an implied length of 1. The channel number field specifies the memory control module to be accessed during the special operations required for the SINH, SLMT, and FMFR operators.

The SOR is loaded on a priority basis with EU-initiated fetches or stores having priority over operations from the operations queue or the SIR. EU-initiated operations include fetches or stores, such as those arising from tracing through a chain of linked descriptors or from the execution of string operators.

PCU-initiated fetch operations, such as Value Calls, may be completed prior to the start of EU execution of the related operator, but the data for store operations must be supplied by the EU.

Storage Unit Data Queue

The storage unit data queue contains four 52-bit word locations. A separate write pointer (SDW) and read pointer (SDR) are used for addressing in the SU data queue. Writing into the queue is performed only by the EU. Information is read from the queue under control of the SU. The information taken from the queue is always passed to the communications unit for storage in main memory, but is also sent to the associative memory and stack buffer when the store is to an address which is local.

Because the EU must determine that the store was completed successfully before additional operators can be executed, the SU data queue contains the data for only one store operation at a time. The queue is four words long so that it may hold the data for any multi-word store operation.

Memory Address Register

The MAR, which contains the memory address for the current operation, is loaded from the queue, from the EU, from the AU, or directly from the SIR. The address in the MAR is sent to the stack buffer and associative memory to allow these units to determine if the address is local, and is forwarded to the communications unit for use in main memory operations.

For multi-word operations, the starting address is forwarded to the CU, then the address in the MAR is counted up so that a check for local addresses may be performed on each

address involved in the multi-word operation.

Storage Unit Control Logic

The timing and controls for SU operations are provided by this portion of the SU. The operations include special fetches, read with lock functions, single or multi-word fetches, protected stores, or overwrites. At the start of any operation, the control logic obtains access to the ASM, the SB, the CU, and if necessary the EWR. Then the CU is started and a check is made to determine if the address is local in the SB or ASM.

For single-word fetch operations, if the information is locally available in the SB or ASM, the communications unit operation is terminated and the fetch operation is performed locally. For multi-word fetch operations if all of the information is available only in the stack buffer (above SLR), the CU operation is terminated. If some of the information is available only in the stack buffer, the portion in main memory is fetched, then the portion in the stack buffer is fetched. The ASM is not used for multi-word operations.

For store and overwrite operations queued by the PCU, a wait bit is set in the operation code. When the operation is in the SOR, this wait bit causes the control logic to hold the operation until the required data has been entered into the SU data queue. If the address is not provided from the queue, the SU also waits until the MAR is loaded by the EU or the AU. When the information is available, the control logic enables the read lines of the SU data queue and disables the write lines of the queue until the operation is complete. For single-word operations, if the address is local, the SU can begin processing the next item in the queue as soon as the local store is completed. The CU continues the operation and will report errors if detected. Note also that on local protected stores, the flashback action can be performed locally.

On multi-word overwrite and store operations, the phased-memory system allows the main memory operations to proceed rapidly. Therefore in these operations, the main memory store is performed before the local stack buffer operation. Only the starting address and the data need to be passed to the communications unit, then the MAR is counted so that each address can be checked for local action. The ASM is purged (invalidated) prior to a multi-word store or overwrite, and thus the operation could be local only in the stack buffer.

On completion of any store or overwrite operation, the control logic resets the SU data queue pointers and enables the write lines.

COMMUNICATIONS UNIT

The communications unit (CU) provides the interface between the CPM and main memory. All main memory accesses are performed by this unit. Requests for memory operations are made to the CU by the program buffer, the storage unit, and the stack buffer. Information fetched by the CU from memory is forwarded to the execution unit, the stack buffer, the associative memory, or, for program code, to the program buffer.

Access to the CU is granted to the requesting CPM units on a priority basis. First priority is given to the stack buffer, because the EU is waiting for the results of any request made by the stack buffer. The stack buffer requests are made when performing a stack-buffer fill, empty, or purge operation. The storage unit has second priority as the EU may be waiting for the results of an SU request. The program buffer requests have third priority as these requests are made in anticipation of the actual need for additional program code.

The major logic elements of the CU, as shown in figure III-1-18, include input (IN) and output (OP) registers, the communications address (CA) register, the communications length (CLN) register, the remember-suspend (RS) register, the fail (FL) register, and the control logic. The fail register, while accessible to the CU, is used by the fault control logic of the CPM and is described with that logic.

On single-word memory operations, the absolute memory address of the operation is contained in the CA register. For multi-word operations, the starting address is in the CA register and the number of words to be fetched or stored is in the CLN register. During the operation, both the address and the word count are adjusted for each word fetched or stored. Program code is fetched in eight-word blocks, which requires two four-word fetches (if the memory configuration allows four-word phasing). If, at the end of the first four-word fetch of program code, a higher priority request has been made for CU use, the current memory address and word length are transferred to the remember-suspend register for temporary storage. Then the second four-word fetch is delayed until after the higher-priority request has been serviced. When no other requests are pending, the RS register contents are loaded back into the CA and CLN registers and the fetching of code is resumed.

When access to main memory is required, the CU control logic compares the six most-significant bits of the address in the CA register with the limits established for each MCM, and selects the appropriate module. Then, the starting address and other control information for the operation are sent to the selected module in a memory control word. The control word is assembled in the input register, then transferred to the output register and is sent to the addressed memory control module. The receipt of the control word is acknowledged by the MCM.

For fetch operations, the MCM notifies the CU that access has been granted by sending a data-present (DAP) signal and the requested data to the CU. The data is received by the IN register and is subsequently forwarded to the program buffer, the stack buffer, the associative memory, or the EWR, as appropriate.

Data for store operations is received by the CU from either the SU data queue or the stack buffer. Data for store operations is buf-

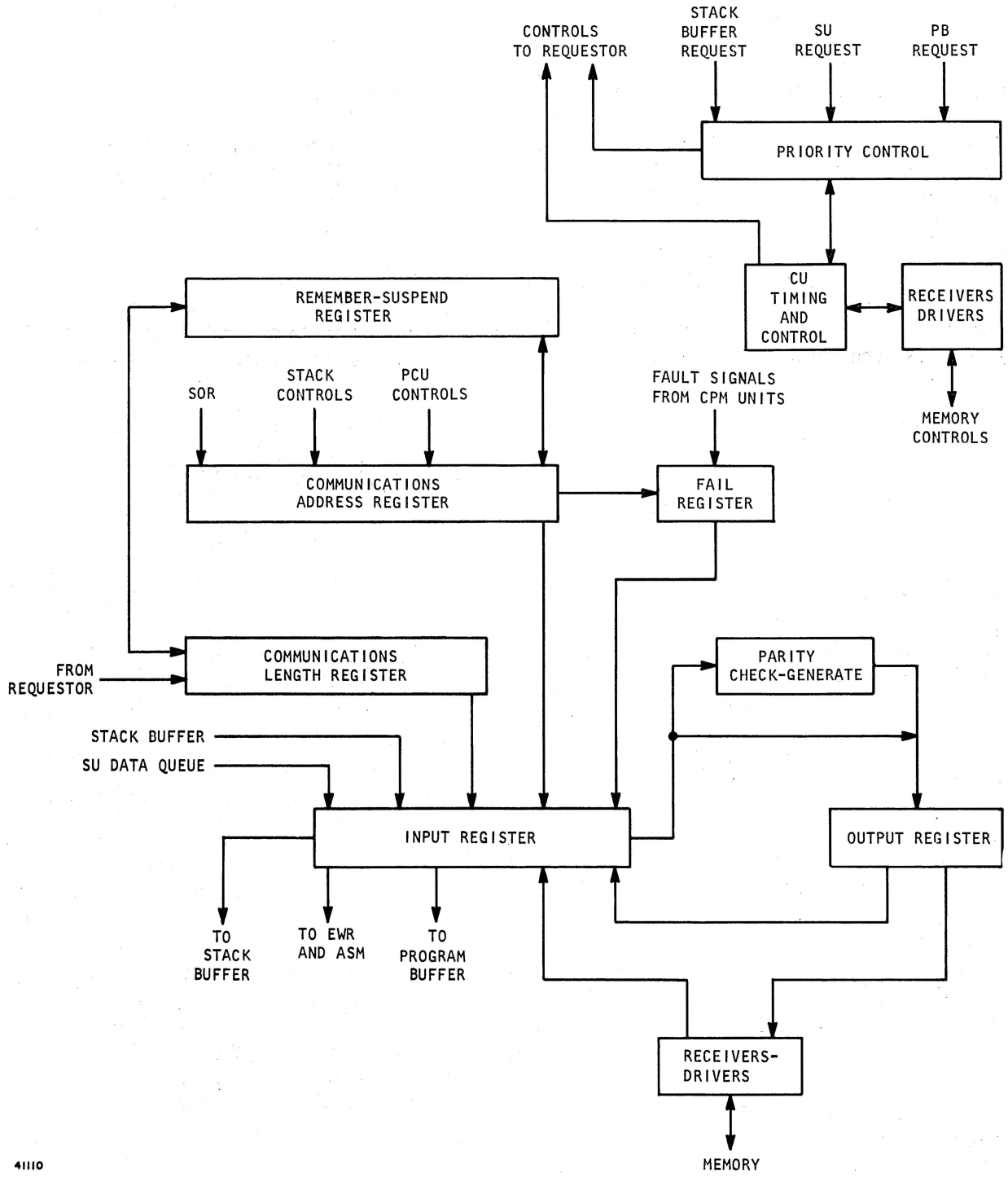


Figure III-1-18. Communications Unit

fered in the IN register until the CU gains memory access. Following the transfer of the control word and the acknowledgement of the receipt of this word, the selected MCM will inform the CU of access by sending a send-data signal to the CU. On obtaining access, the CU transfers the data into the output register and the word is then sent to the selected MCM.

FAULT CONTROL LOGIC

The fault control logic is mechanized in such a way as to aid in general maintenance and error recovery under the guidance of the MCP. Error recovery is aided by a system of multiple levels of control mode coupled with alternate stack and display zero capabilities. The fault condition register records system interrupts and conditions the processor to take the necessary action in order to handle these interrupts (see figure III-1-19). This register records both operator dependent and operator independent interrupts. The program index and program syllable counts on operator dependent interrupts are adjusted to allow re-execution of the interrupted operator. The operator independent interrupts are processed at the conclusion of the operator being executed at the time of the interrupt.

The interrupt system allows for special "complete" communication and for recording single-bit errors and two-bit errors from main memory. It also allows for the sensing and recording of special errors internal to the main memory and Central Processor.

The following is a complete list of interrupts (alarm, syllable, special and external) available to the B 7700 Processing System.

A. Alarm (first priority)

Loop

Memory Parity

Memory Fail 1

Invalid Address (No Access to Memory)

Stack Underflow

Invalid Program Word

Processor Internal

B. Syllable (second priority)

Memory Protect

Invalid Operand

Divide by Zero

Exponent Overflow

Exponent Underflow

Invalid Index

Integer Overflow

Bottom of Stack

Presence Bit

Sequence Error

Segmented Array

Programmed Operator

Privileged Instruction

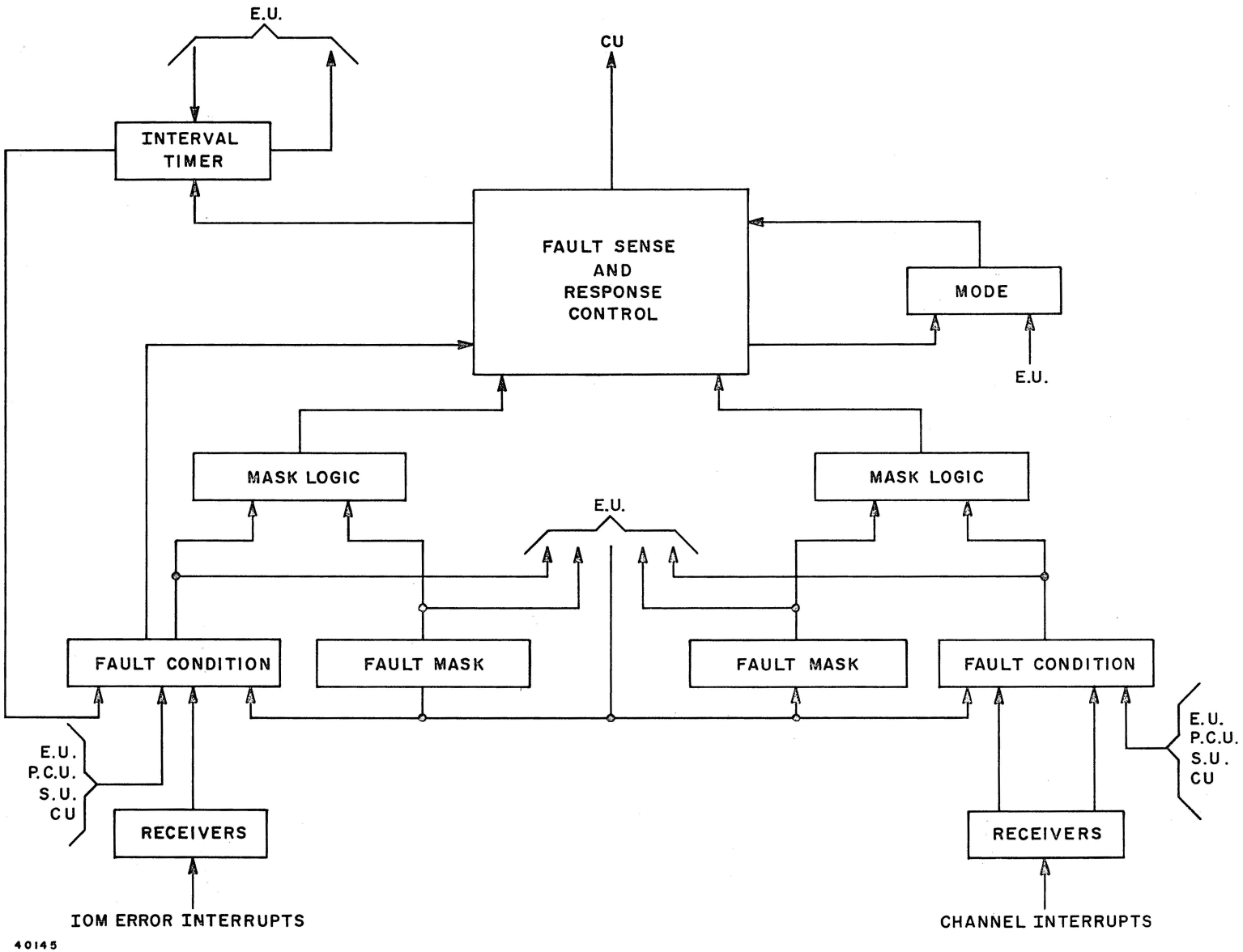


Figure III-1-19. Fault Control Logic

C. Special (third priority)

Interval Timer

Stack Overflow

D. External (fourth priority)

CHANNEL 0

CHANNEL 1

CHANNEL 2

CHANNEL 3

CHANNEL 4

CHANNEL 5

CHANNEL 6

CHANNEL 7

IOM ERROR 0

IOM ERROR 1

IOM ERROR 2

IOM ERROR 3

IOM ERROR 4

IOM ERROR 5

IOM ERROR 6

IOM ERROR 7

MEMORY FAIL 2

Description of these interrupts is provided in Section 2 of this chapter.

General Discussion of Interrupts

Special and external types of interrupts in the B 7700 Processing System can be masked so that the processor is insensitive to their occurrence. Each bit of the fault condition register has a corresponding bit in the fault mask register. An interrupt is only recognized if its corresponding mask bit is set (logical one). If at a later time the mask configuration is changed, then resident interrupts, which are now unmasked, will be sensed and processed. In this way, processing of selected interrupts can be deferred. The mask logic accumulates all the unmasked interrupts into a general signal indication to alert the fault control logic that one or more interrupts requires processing. The interrupt mode of the processor is recorded in a four-bit mode register. The bit configuration of the mode register represent Normal Mode and Control Modes 1 through 3. Automatic hardware control administers the progression to higher modes, but a Set Processor Register operator is available for programmatic establishment of modes. Another operator, Read Processor Register, allows for programmatic sensing of the mode register.

General Discussion of Fail Register

The fail register (FR) in the communications unit of the central processor is used to provide additional information concerning processor internal and memory related error conditions. The processor internal portion of the FR reports parity, residue, continuity, and decoding errors.

The memory related portion of the FR reports the following types of interrupt conditions:

a. Processor internal memory related error (PIMR) - an interface error detected during an operation between the communications unit and the other sections of the Central Processor.

b. Memory parity error (MPE) - a parity error detected during an access to main memory.

c. Invalid address error (IAE) - an address does not exist in main memory and a memory time out occurred in the Central Processor.

In addition to the above interrupt conditions, the memory portion of the FR also reports fail 2 interrupt recovery by the Memory Control Module.

When an interrupt condition is detected, a bit assigned to designate that condition is set in the 27-bit FR register. However, the indication of the error is queued with the operand, and the Central Processor module is not interrupted until after the affected operation is completed by the execution unit.

SECTION 2
INTERRUPTS

INTRODUCTION

An interrupt is a means of diverting a processor temporarily from the job which it is doing if certain predetermined conditions occur, so that some higher priority job may be done immediately. Interrupts are processed by the interrupt handling mechanism of the MCP. When the interrupt has been processed, the MCP will (if conditions permit) reactivate the interrupted process.

The interrupt handling mechanism of the MCP deals with two classes of interrupts: hardware interrupts and software interrupts. Hardware interrupts are generated automatically by the B 7700 system (when interrupt conditions occur) and are processed by the MCP interrupt procedure. Software interrupts are programmatically defined, and are used both by the MCP and by object programs for communication between processes. This discussion deals only with hardware interrupts.

HARDWARE INTERRUPT SYSTEM

The B 7700 hardware interrupt system is a primary interface between the MCP and the hardware. Interrupt conditions may be detected by the Central Processor Module (CPM), the Input/Output Module (IOM); or the Memory Control Module (MCM). When detected, interrupt conditions are processed by the Fault Control Logic of the CPM. Normally, the CPM prepares the stack for procedure entry, places the necessary parameters in the stack, and causes an entry into the MCP interrupt procedure.

CPM STATES AND MODES

The CPM operates in either of two states: control state, used only by the MCP; or normal state, used both

by user programs and by the MCP. Normal state is always used when executing user programs. Control state is used when executing certain portions of the MCP, including the MCP interrupt handling procedure. The differences between the two states are that in control state external interrupts and interval timer are inhibited (except during an IDLE or PAUS instruction) and in control state the CPM may execute privileged instructions which it may not execute in normal state.

In addition to the two states, the CPM can be in any one of five interrupt handling modes: Normal Mode (Control Mode 0), Control Mode 1 (CM1), Control Mode 2 (CM2), Control Mode 3 (CM3), and Control Mode 4. The CPM operates in normal mode until an interrupt condition is detected. The first three control modes allow for recursive attempts to enter the hardware interrupt routine (the fault control logic of the (CPM). Control Mode 4 indicates that these attempts were not successful.

There is no direct connection between the states of operation and the modes of operation of the CPM. The CPM may be in any of the four interrupt modes while either in control state or in normal state. In a system which contains more than one CPM, any or all of the CPM's may operate in control state or normal state, as well as in any of the interrupt modes. The CPM states are described below; the interrupt modes are further described in the discussion of interrupt processing.

Control State

Entry into control state (from normal state) occurs when the MCP enters or returns to a control state procedure (an MCP SAVE procedure), or when the CPM executes a Disable External Interrupts operator (DEXI). (Control state procedures have bit 19, the N bit, of the PCW set.) While the CPM is operating in control state the reporting of external interrupts to the MCP interrupt handling routine is disabled. Additionally, the CPM may execute certain privileged operators while in control state which may not be executed in normal state. When the CPM is operating in control state, the normal control state flip-flop (PST) and the inhibit interrupt flip-flop (IIHT) are both set (except during an IDLE instruction).

The interrupts which are normally inhibited while in control state include the Channel interrupts, the IOM Error interrupts, the Interval Timer interrupt, and the Memory Fail 2 interrupts. Although the processing of these interrupts is inhibited, the appropriate bit in the CPM Interrupt (Fault) register will be set if one of these interrupts is detected, and the interrupt will be processed when the CPM enables External interrupts either by returning to normal state or by executing an IDLE or PAUS operator.

The operators which are enabled only when the CPM is in control state include Set Interval Timer (SINT), Inhibit Parity (IGPR), Set Memory Inhibits (SINH), and Set Memory Limits (SMLT).

Normal State

Return to normal state (from control state) occurs whenever the MCP initiates or returns to a normal state procedure (non-SAVE procedure), or when the CPM executes an Enable Ex-

ternal Interrupt operator (ECXI). (Normal state procedures have bit 19, the N bit, of the PCW reset). When the CPM is operating in normal state, the privilege state flip-flop (PST) and the inhibit interrupt flip-flop (IIHT) are both reset. When a CPM returns to normal state after servicing an interrupt, it does not necessarily return to the program which was executing when the interrupt was detected. The selection of the job to be run is a function of the MCP.

FAULT CONTROL LOGIC

The fault control logic of the CPM contains four registers which are used to record and process hardware interrupts: the Fault Condition, Fault Mask, Fail, and Control Mode registers. The Fault Condition register is used to indicate the detection of one or more interrupt conditions (one bit for each condition). The Fault Mask register is used to inhibit (mask out) the processing of one or more interrupt conditions. (The Fault Condition register may be read in such a way as to obtain only interrupt conditions which are not masked out; thus indicating an interrupt condition which must be processed by the MCP.) The Fail register further identifies errors which are internal to the CPM and CPM-MCM interface errors. The Control Mode register is used to identify the interrupt mode (Normal, Control Mode 1, Control Mode 2, and Control Mode 3) in which the CPM is operating.

In addition to the four CPM registers, the B 7700 system makes use of a Fail Register in each Memory Control Module (MCM), and a Fail Register in each Input/Output Module (IOM). The MCM Fail Register will contain detailed information describing memory failures within the MCM. The IOM Fail Register will contain detailed information describing errors detected within the IOM.

Fault Condition Register

The Fault Condition register contains one bit for each of the possible interrupt conditions (see table III-2-1). The low order bits of the register are associated with interrupts which have the highest priority for being processed by the CPM; the high order bits are associated with interrupts which have the lowest priority. When interrupt conditions are detected, the bits associated with those conditions are set in the Fault Condition register.

Normally, the Fault Condition register is set by the interrupt condition. As each interrupt condition is processed, the bits in the register are selectively reset. Programmatic control of the Fault Condition register is accomplished by use of the Set Processor Register (SPRR) and Read Processor Register (RPRR) operators. The RPRR operator causes the contents of the register to be placed in the stack, and the register itself to be reset. The SPRR operator causes an inclusive OR setting of the register; that is, bits are set, but bits which already are set are not reset.

Fault Mask Register

The Fault Mask register allows the processing of certain interrupts to be inhibited or deferred. Alarm interrupts and Syllable interrupts may

not be masked. Each Special interrupt and each External interrupt has a corresponding bit in the Fault Condition and Fault Mask registers (see table III-2-1). An interrupt condition will only be recognized by the CPM if the Fault Mask register bit for that condition is set (logical one). If the Fault Mask bit is reset for an interrupt condition, that interrupt bit will still be recorded in the Fault Condition register but will go unnoticed by the fault control logic. If the mask configuration is later changed, then interrupts (including those resident in the Fault Condition register when the mask is changed) which are now unmasked will be recognized and processed. In this way, processing of selected interrupts can be deferred.

The Fault Mask register may only be set programmatically. The Read Processor Register operator causes a simple read of the register (without reset); the Set Processor Register operator causes a simple set of the register (each bit is set either to logical one or to logical zero).

Table III-2-1. B 7700 Interrupt Bit Assignments

Interrupt (by Priority)	Fault Condition Register	Fault Mask Register	Interrupt Identification (P1)
Alarm (First Priority)			
Loop	Bit 0	NO NE	Bit 0
Memory Parity	1		1
Memory Fail 1	2		2
Invalid Address (No Access)	3		3 Plus Bit 25
Stack Underflow	4		4
Invalid Program Word	5		5
Processor Internal	6		6
Syllable (Second Priority)			
Memory Protect	Bit 9		Bit 0
Invalid Operand	10		1
Divide By Zero	11		2
Exponent Overflow	12	NO NE	3
Exponent Underflow	13		4 Plus Bit
Invalid Index	14		5 24 or Bit
Integer Overflow	15		6 23 (See
Bottom Of Stack	16		7 Note 1)
Presence Bit	17		8
Sequence Error	18		9
Segmented Array	19		10
Programmed Operator	20		NONE
Privileged Instruction	21		Bit 11

Table III-2-1. B 7700 Interrupt Bit Assignments

Interrupt (by Priority)	Fault Condition Register	Fault Mask Register	Interrupt Identification (P1)
Special (Third Priority)			
Interval Timer	Bit 23	Bit 23	Bit 0 } Plus
Stack Overflow	24	24	1 } Bit 22
External (Fourth Priority)			
Channel 0	Bit 26	Bit 26	Bit 0
Channel 1	27	27	1
Channel 2	28	28	2
Channel 3	29	29	3
Channel 4	30	30	4
Channel 5	31	31	5
Channel 6	32	32	6
Channel 7	33	33	7 } Plus
IOM Error 0	34	34	8 } Bit 21
IOM Error 1	35	35	9
IOM Error 2	36	36	10
IOM Error 3	37	37	11
IOM Error 4	38	38	12
IOM Error 5	39	39	13
IOM Error 6	40	40	14
IOM Error 7	41	41	15
Memory Fail 2	42	42	16

NOTE 1: On syllable interrupts ID Bit 24 indicates class 1 (PIR, PSR, PDR, PBR have not been modified), ID Bit 23 indicates class 2 (PIR, PSR, PDR, and PBR are undefined).

Interrupt Identification

Each interrupt condition reported to the MCP is identified by a unique literal value, known as interrupt parameter P1 (see table III-2-1). This parameter is passed to the MCP interrupt procedure by the fault control logic to identify the condition which is to be processed. The P1 parameter is derived from the contents of the Fault Condition and Fault Mask registers through a series of gates. Interrupt conditions reported in the Fault Condition register which are not masked out by the Fault Mask register are used to make up the P1 parameter.

Normally, this parameter is read and placed into the stack by the fault control logic, although it may be read into the stack programmatically. In either case, the resultant action is as follows. The value of P1 is read into the stack and the bits which were set in P1 are reset in the Fault Condition register. In a particular P1 parameter, all interrupts of a particular priority level which are not masked out are reported, but only one priority lev-

el is reported on each read. The priority level reported will be the highest priority level for which there is at least one bit set in the Fault Condition register which is not masked out. If the value of the P1 parameter is read programmatically (using the Read Processor Register operator), and if there are no unmasked interrupts to report, a word of all zeros is read into the stack. (The fault control logic will read P1 only when there is an unmasked interrupt to report).

CPM Fail Register

The CPM Fail Register (see figure III-2-1) is used to provide more detailed information about specific interrupts. It may be considered to have three parts: a part concerning errors which are internal to the CPM, a part concerning errors which are memory-related, and a single bit indicating continuability after alarm interrupts. Each of these parts is independently set by the fault control logic: the three parts are read and cleared (read destructive) as one. If more than one interrupt affecting one of the three parts of the register occurs before the register is read and cleared, the part is completely overwritten with the information about the most recent interrupt.

Errors which are internal to a CPM are described by the setting of the CPM Fail Register. Error conditions reported include:

a. parity, residue, continuity, and decoding errors in the execution unit,

b. queue overwrite,

c. residue error in the address unit,

d. internal error in the program unit,

e. memory error on protected store.

Memory-related errors which are described in the CPM Fail Register include all single-bit correctable errors reported to a CPM, all Processor Internal Memory Related (PIMR) errors, and all Memory Fail 1 (MF1), Memory Parity (MPE), and Invalid Address (IAE) errors which occur on overwrites for the stack buffer or for the execution unit. The PIMR errors are detected by the CPM communications section, and include bad parity, address bad residue, channel number error, and pseudo no access to memory (no complete signal).

The third section of the CPM Fail register consists of a single bit which indicates, on alarm interrupts, whether PIR, PSR, PBR, and PDR are consistent and, (so far as the fault control logic can tell) can be used to continue the interrupted process. A zero indicates consistency; a one indicates that the restart point is not reliable. This indication is similar to the distinction between class 1 and class 2 syllable interrupts.

Whether or not the CPM Fail register contains useful information for a given interrupt depends on the type of interrupt. The register is not used for Syllable interrupts. It is not necessarily used for Loop, Stack Underflow, or Invalid Program Word Alarm interrupts. It is not used

for Channel or IOM Error interrupts. The register is used for only one External interrupt: Memory Fail 2. The register is used for four Alarm interrupts: Memory Parity, Memory Fail 1, Invalid Address, and Processor Internal.

The individual sections of the CPM Fail register are used as follows: the processor internal error section is used for all processor internal errors (processor internal memory related errors use the memory related section as well). The memory related section is also used for memory parity, Memory Fail 1, and invalid access errors, but only when parameter P2 is not used, in which case P2 will be all zeros. The memory related section is also used for all Memory Fail 2 interrupts. The continuability bit is only applicable to Alarm interrupts.

The CPM Fail register may be read (destructive) using the Read Processor Register operator.

Control Mode Register

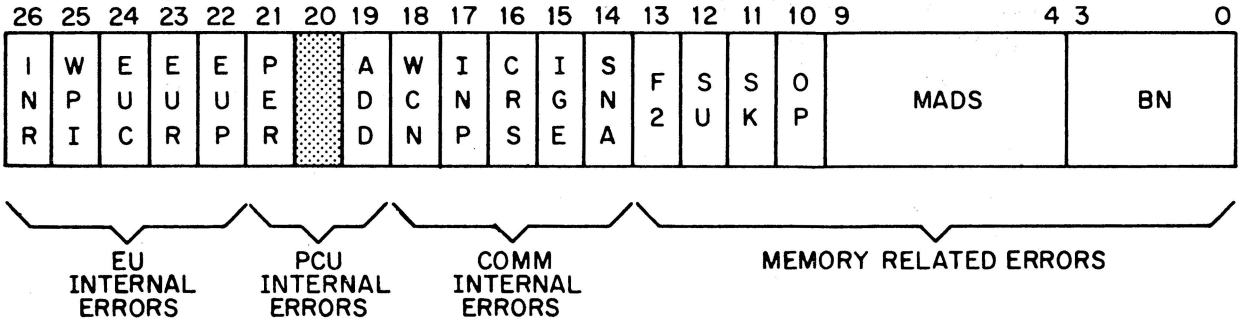
The Control Mode register indicates the interrupt mode in which the CPM is operating. The use of interrupt modes provides for recursive entries into the fault control logic. The progression to higher interrupt modes is controlled automatically by the hardware. In addition, programmatic control of the Control Mode register may be accomplished by use of the Read Processor Register and Set Processor Register operators.

The Control Mode register contains three bits which display the interrupt modes of the CPM: the CM1 bit, the CM2 bit, and the CM3 bit. In Normal Mode, none of these bits are set. In Control Mode 1, only the CM1 bit is set. In Control Mode 2, only the CM2 bit is set. In Control Mode 3, the CM1 and CM2 bits are both set. In Control Mode 4, only the CM3 bit is set. The CPM will be halted with the last interrupt displayed in the Fault Condition register and the HSI flip-flop set if an interrupt is detected in CM3.

The CPM operates in Normal Mode while not attempting to process an interrupt. When an interrupt condition is detected, the CPM advances to CM1 and attempts to call the procedure pointed to by D[0]+3 (the MCP interrupt procedure) from the stack of the user program. If an interrupt is detected while in CM1, the CPM advances to CM2, moves to an alternate stack (determined by indexing the stack vector by the CPM number), and attempts to call the MCP interrupt procedure again. If an interrupt condition is detected in CM2, the CPM advances to CM3, moves to an alternate D[0] stack (pointed to by the ADZ register), and attempts to enter a different interrupt procedure pointed to by alternate D[0]+3. If still another interrupt is detected while in CM3, it is obvious that a recursive interrupt processing situation exists, and the CPM advances to CM4 and halts. If the CPM succeeds in entering the MCP interrupt procedure, the Control Mode register is reset to Normal Mode programmatically.

MCM Fail Register

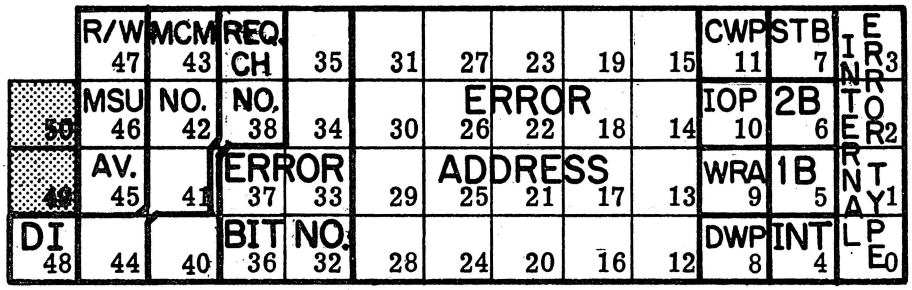
The MCM Fail Register (see figure III-2-2) is a 51-bit register which is used to provide further information necessary to identify and define failures within the Memory Stack Units controlled by the Memory



Field	Bits	Description
INR	26:1	Inhibit normal return.
WPI	25:1	Wrong PIR (PIR may be one less).
EUC	24:1	EU continuity error.
EUR	23:1	EU residue error.
EUP	22:1	EU parity error.
PER	21:1	PCU error.
	20:1	Not used
ADD	19:1	Residue error in adder.
WCN	18:1	Wrong channel number.
INP	17:1	Comm. unit received bad parity from stack buffer or stack data queue.
CRS	16:1	Comm. unit residue error.
IGE	15:1	Error occured on operation that was ignored.
SNA	14:1	Comm. unit did not receive a requestor operation complete signal from MCM.
F2	13:1	Single bit error.
SU	12:1	0 = Not SU operation. 1 = SU operation.
SK	11:1	0 = Not stack operation. 1 = Stack operation.
OP	10:1	0 = Fetch. 1 = Store.
MADS	9:6	Memory address.
BN	3:4	Box number (MCH NO.)

40991

Figure III-2-1. CPM Fail Register



Field	Bits	Description
	50:2	Not used.
DI	48:1	Delayed Interrupt. A one indicates that the MCM detected an internal error after the ROC flip-flop was set
R/W	47:1	Read/Write bit. 0 = read
MSU AV.	46:2	MSU availability. 00 = No MSU's 01 = Single MSU 10 = Two MSU's 11 = Four MSU's
MCM NO.	44:4	MCM number (0-7)
REQ. CH. NO.	40:3	Requestor channel number.
ERROR BIT NO.	37:6	Number of bit that failed. (Single-bit corrected error)
ERROR ADDRESS	31:20	Address of location being accessed. (One-bit or two-bit error)
CWP	11:1	Control word parity error.
IOP	10:1	Illegal op code.
WRA	9:1	Wrong MCM.
DWP	8:1	Data word parity error.
STB	7:1	Data strobe error.
2B	6:1	Two-bit error.
1B	5:1	One-bit error.
INT	4:1	Internal error (type defined by bits 3:4).
INTERNAL ERROR TYPE (CODED)	3:0	Internal error. Binary value of bits indicates type of error as follows:

Value	Type
8	MSU availability error.
7	Data transfer control (DTC) error.
6	Data timer failure.
5	Parity generator (MSU control) failure.
4	MSU parity error.
3	Address counter failure.
2	Checker/generator failure.
1	Read available failure.
0	MSU unavailable.

40992

Figure III-2-2. MCM Fail Register

Control Module (MCM). There are three parts of the MCM Fail Register: the first part is applicable to irrecoverable errors (reported to the CPM by a Memory Fail 1 interrupt); the second part (the error bit number and the error type bit) is applicable only to one-bit correctable errors; and the third part, which provides the address of the first word detected in error and may be used in conjunction with either of the other two parts. The address of the word in error may be provided in the P2 parameter. It should be understood however, that the MCM Fail register is locked when an error is detected, and that subsequent errors are not identified in this register. Thus, the MCM Fail register may not contain information regarding the most recent error.

IOM Fail Register

The IOM Fail register (see figure III-2-3) is a 48-bit register which contains information regarding errors which cannot be associated with a particular channel or device. (Such errors cause an IOM Error interrupt). When an IOM Error interrupt occurs, the contents of the IOM Fail register are placed in the Result Descriptor field of the "Fail IOCB". The Fail IOCB is associated with Unit Designate Number 0. The Fail IOCB is delinked from the queue of Fail IOCB's and linked into the queue of completed IOCB's (defined by the Status Queue Header) in the same manner as a normal I/O termination. If the Unit Table word for the fail IOCB or the Status Queue is locked, the result descriptor is left in memory at HA (Home Address) +5.

INTERRUPT PROCESSING

All interrupt conditions which are reported in the Fault Condition Register and which are not masked out by the Fault Mask register are accumulated into a general signal to alert the fault control logic of the CPM to the fact that one or more interrupts require processing. When an interrupt requires processing the CPM will advance the Control Mode register (in most cases from Normal Mode to CM1) and will attempt to enter the MCP interrupt procedure.

Interrupt Processing In Normal Mode

After advancing the Control Mode register from Normal Mode to CM1, the CPM will attempt to perform the following sequence of operations:

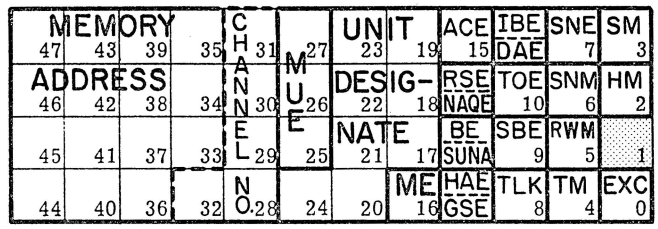
a. read and save the P1 parameter.

b. place a Mark Stack Control Word (MSCW) into the stack, normally, the stack of the user program.

c. place an Indirect Reference Word (IRW) into the stack. The IRW references a reserved location (D[0] + 3) in the MCP stack. (When in Control Mode 3, the IRW references a reserved location (D[0] + 3) in the Alternate D[0] stack.)

d. place an integer value defining the interrupt (the P1 parameter previously saved) into the stack.

e. place a second parameter into the stack (the P2 parameter)



Field	Bits	Description
MEMORY ADDRESS	47:20	This field contains the location in memory that was last accessed at the time of the error. This field is not valid if bit 15 (ACE) is set.
CHANNEL NO.	32:5	This field contains a channel number only when bit 15 (ACE) is set.
MUE	27:3	This field is valid only when bit 16 (ME) is set. The bits specify the following meanings:

(001) Memory detected error - This error condition is declared when the addressed memory module responds with a fail 2 (1-bit corrected error) indication to a requestor unit. (This error condition does not cause termination of the memory access operation.)

(010) LIA address residue error - This error condition is declared if the MIU receives a UCW whose residue bits do not agree with its level-1 address field configuration (DCP words are not residue checked).

(011) Store disparity - This error condition is declared if a data transfer from an internal unit is received by the MIU with incorrect parity. The data with incorrect parity is transferred to the memory.

(100) Memory protect error - This error condition is declared when the addressed memory module responds with a protect-error signal during a memory protect store operation.

(101) No access to memory - This error condition is declared if the MIU receives no response from the requestor memory module during a waiting period not to exceed 25 microsecond writing period. No response means:

1. Failure to receive, at the MIU, an acknowledged signal from an addressed memory module, or that
2. An addressed memory module attempted to transfer incomplete data.

(110) Fetch disparity - This error condition is declared if a fetch of data from memory is received by the MIU with incorrect parity.

(111) Memory detected error - This error condition is declared when the addressed memory module responds with a fail 1 (uncorrectable error) indication to a requestor unit.

UNIT DESIGNATE	24:8	A Unit Designate of all zeroes signifies a Fail Register Result Descriptor.
ME	16:1	Memory Error - The memory error or MIU detected error is found by decoding bits 25, 26, 27 of the Fail Register.
ACE	15:1	Active Channel Stack Error. The address (channel no.) of the word in the stack that caused the parity error is contained in bits 32:5.
RSE (SNE=0)	14:1	Residue Error (Memory Address). The address in error is contained in bits 47:20.
NAQE (SNE=1)	14:1	No Access to DFO Exchange.
BE (SNE=0)	13:1	Buffer Register Parity Error.
SUNA (SNE=0)	13:1	Storage Unit Not Available.
HAE=1 (SNE=0)	12:1	Home Address Illegal Command.
QSE (SNE=1)	12:1	DFO Stack Parity Error.
IBE (SNE=0)	11:1	Initiate Busy Channel Error. An attempt was made to start a non exchange channel that was either busy or in the process of being terminated.
DAE (SNE=1)	11:1	Disk Address Error.
TOE	10:1	Time Out Error - When SNE (bit 07) is set, TOE represents a Scan Bus Time Out error. When SNE is reset, TOE represents a Data Service Time Out error.
SBE	9:1	Scan Bus Error - Indicates a parity error on the Scan Bus. SNE (Bit 07) will also be set.
TLK	8:1	Table locked - Translator timed-out trying to fetch a locked unit table or status queue header.
SNE	7:1	When set, bits 14-9 represent scan errors.
SNM	6:1	Scan Mode.
RWM	5:1	Ring Walk Mode
TM	4:1	Terminate Mode
SM	3:1	Start Mode
HM	2:1	Home Address Mode
		Indicates the translator mode of operation when error occurred.
	1:1	Not used.
EXC	0:1	Exception Bit - this bit indicates that a "1" exists in the Fail Register.

40993

Figure III-2-3. IOM Fail Register

giving their information about the interrupt. If P2 is not used by the interrupt condition being reported, P2 will be set to zero.

f. execute an Enter operator. The fault control logic expects to find a Program Control Word (PCW) at D[0] + 3; however, an SIRW, an IRW or an IRW chain which points to a PCW are possible conditions.

The two interrupt parameters (P1 and P2) that are inserted into the stack are used to supply information describing the interrupt condition. The P1 parameter provides information concerning the type of interrupt, the interrupt priority level, and the interrupt class. The P2 parameter supplies supplementary information about the interrupt condition, such as a memory address (memory related interrupts) or a copy of the non-present descriptor (presence bit interrupts), etc.

When the interrupt procedure of the MCP is entered, the IRW in the stack (step b above) is overwritten with a Return Control Word (RCW) by the ENTER operator. As with any procedure entry, this RCW points to the point in the code string to which control is to be returned following execution of the procedure. For some operators the PIR and PSR settings are not advanced; thus the RCW must be changed before exiting the interrupt procedure.

Figure III-2-4 depicts the stack format just prior to and just after entering the interrupt procedure.

Interrupt Processing in CM1

When an interrupt is detected while in CM1, the CPM advances to CM2 and attempts to call the MCP interrupt procedure from a different stack. The new stack is found by using the processor number as an index into the Stack Vector Array. (This array is pointed to by the Stack Vector Descriptor, located at D[0] + 2.) The index into the Stack Vector Array results in a data descriptor, which points to the base of the stack for the new stack. This stack was established by the MCP at the time of system initialization.

The Bottom Of Stack Register (BOSR) is set to the base address of the new stack, which contains the Top Of Stack Control Word (TOSCW) for the new stack. A modified move-to-stack operation then causes the TOSCW for the old stack, the old BOSR setting, and the old SNR register (stack number) setting to be placed in the top of the new stack. After these parameters have been placed, the stack is marked, the IRW and the P1 and P2 parameters are placed in the stack, and the MCP interrupt procedure is entered. The stack structure just prior to entering the MCP interrupt procedure is shown in figure III-2-5.

Interrupt Processing In CM2

When an interrupt is detected while in CM2, the CPM advances to CM3 and attempts to call the interrupt procedure pointed to by alternate D[0] + 3 from yet another stack (located either in a different Memory Control Module if one exists or in the high end of memory above MCP linked memory if no other MCM exists). This stack is found as follows. Each processor contains an alternate D[0] register (D[0] A). The contents of D[0] A are moved to D[0], and the processor attempts to move to a new stack as described above.

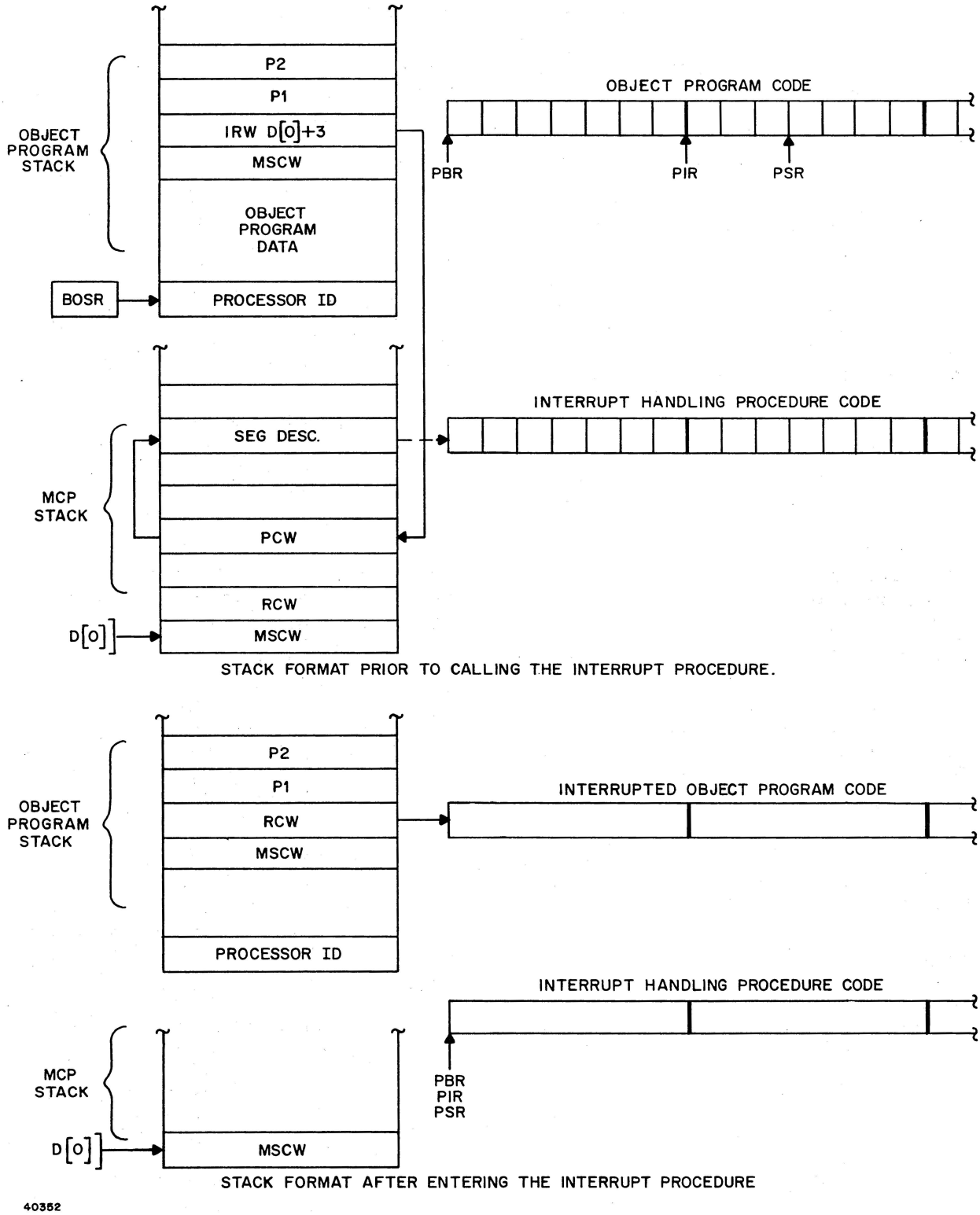


Figure III-2-4. Stack Format

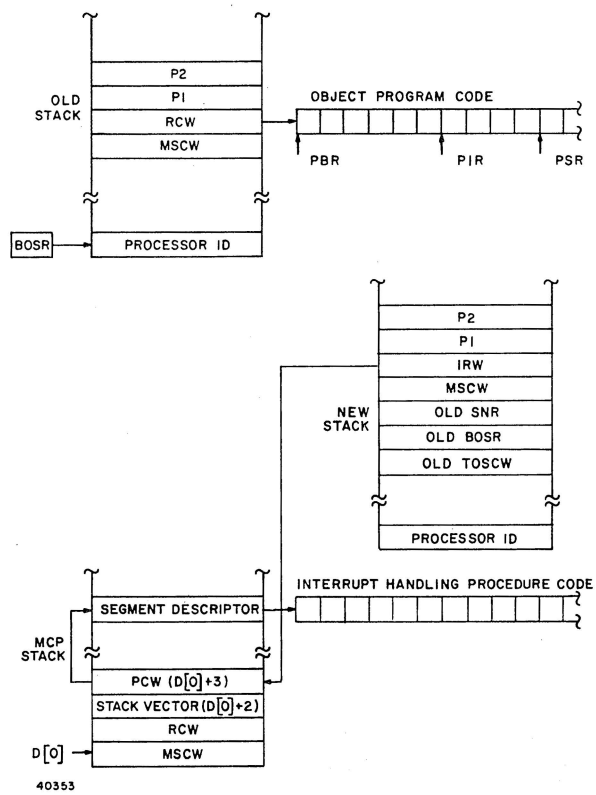


Figure III-2-5. Stack Format Prior To Calling Interrupt Procedure While In CM1 (Move Stack Operation)

Interrupt Processing In CM3

If an interrupt is detected while in CM3, It is obvious that a recursive interrupt condition exists. In such cases the CPM is halted in CM4, the most recent interrupt is identified in the Fault Condition Register, the HSI flip-flop is set, and the CM3 bit is set.

Control Mode Advancement

As earlier described, the CPM may be either in Normal Mode or in one of the four Control Modes. When no interrupts are being processed, the CPM operates in Normal Mode. CM1 is entered when an interrupt is detected in Normal Mode, CM2 is entered when certain interrupt conditions are detected while the CPM is in CM1, CM3 is entered when those interrupt conditions are detected while the CPM is in CM2, and when those interrupt conditions are detected in CM3 the CPM halts in CM4. The conditions which cause the Control Mode register to advance are:

a. detection of an alarm interrupt.

b. detection of a syllable dependent interrupt.

c. detection of an unmasked stack overflow interrupt while the CPM is in Normal Mode.

d. detection of two consecutive stack overflow interrupts while the CPM is in CM1, CM2, or CM3.

e. detection of an interval timer interrupt which is not masked out.

f. detection of an external interrupt which is not masked out.

Figure III-2-6 illustrates the priority scheme for reporting interrupts, the conditions for advancing the Control Mode register, and the interrupt conditions which may be left in the Fault Condition register for later servicing. As shown in case one, the Fault Condition register contains Alarm interrupts (first priority) and Stack Overflow interrupts (third priority), and may also contain Syllable interrupts (second priority), Interval Timer interrupts (third priority), and External interrupts (fourth priority). The Alarm interrupt causes the Control Mode register to be advanced (from Normal to CM1, CM1 to CM2, CM2 to CM3, or from CM3 to CM4), the P1 parameter reports the Alarm interrupt, and the Stack Overflow and External interrupts are still contained in the Fault Condition register (all other interrupts are cleared from the register). Following entry to the software interrupt procedure, the Stack Overflow interrupt is reported by another P1, the Control Mode register is left unchanged (not advanced), and the External interrupt is left in the Fault Condition register for later servicing. (The External interrupt is not recognized because the CPM operates in Control State while the MCP interrupt procedure is being executed.)

Case two shows all priorities of interrupts except Alarm interrupts present in the Fault Condition Register. The resultant action is similar to case one, in that the highest priority interrupt (Syllable) is serviced first. P1 reports the Syllable interrupt, the Control Mode register is advanced, and the Stack Overflow and External interrupts are still contained in the Fault Condition register (in this case the Interval Timer interrupt is also left in the Fault Condition register). Following entry into the software interrupt procedure, the Stack Overflow interrupt is reported by another P1, the Control Mode register is left unchanged, the Interval Timer interrupt is cleared from the Fault Condition register, and the External interrupt is left for later servicing. The stack structure for either case one or case two is shown in figures III-2-7 and III-2-8.

Case three of figure III-2-6 shows a Syllable interrupt (second priority), an Interval Timer interrupt (third priority), and an External interrupt (fourth priority) all present in the Fault Condition register. In this case, the highest priority interrupt present (Syllable) is reported in P1, the Control Mode register is advanced, and the Interval Timer and External Interrupts are left for later servicing.

Case four shows a Stack Overflow interrupt, an Interval Timer interrupt, and an External interrupt present in the Fault Condition register. The Stack Overflow interrupt is reported in P1, the Interval Timer interrupt is cleared from the register, and the External interrupt is left for later servicing.

Case five shows servicing of an Interval Timer interrupt, leaving an External interrupt for later servicing. Case six shows servicing of an External interrupt. Notice that these two cases can only occur when

FAULT CONDITION REGISTER (BEFORE REPORTING INTERRUPT)					CONTROL MODES REGISTER				REPORTED IN	FAULT CONDITION REGISTER (AFTER REPORTING INTERRUPT)					
EXTERNAL	INTERVAL TIMER	STACK OVERFLOW	SYLLABLE	ALARM	NORMAL	CM1	CM2	CM3	PARAMETER P1	EXTERNAL	INTERVAL TIMER	STACK OVERFLOW	SYLLABLE	ALARM	
ϕ	ϕ	1	ϕ	1	CM1 CM1	CM2 CM2	CM3 CM3	* —	ALARM STACK OVERFLOW	ϕ ϕ	0 0	1 0	0 0	0 0	CASE ONE
ϕ	ϕ	1	1	0	CM1 CM1	CM2 CM2	CM3 CM3	* —	SYLLABLE STACK OVERFLOW	ϕ ϕ	ϕ 0	1 0	0 0	0 0	CASE TWO
ϕ	ϕ	0	1	0	CM1	CM2	CM3	*	SYLLABLE	ϕ	ϕ	0	0	0	CASE THREE
ϕ	ϕ	1	0	0	CM1	CM1	CM2	CM3	STACK OVERFLOW	ϕ	0	0	0	0	CASE FOUR
ϕ	1	0	0	0	CM1	—	—	—	INTERVAL TIMER	ϕ	0	0	0	0	CASE FIVE
1	0	0	0	0	CM1	—	—	—	EXTERNAL	0	0	0	0	0	CASE SIX

* PROCESSOR HALTS
ϕ MAY BE A ONE OR A ZERO

40354

Figure III-2-6. Interrupt Reporting

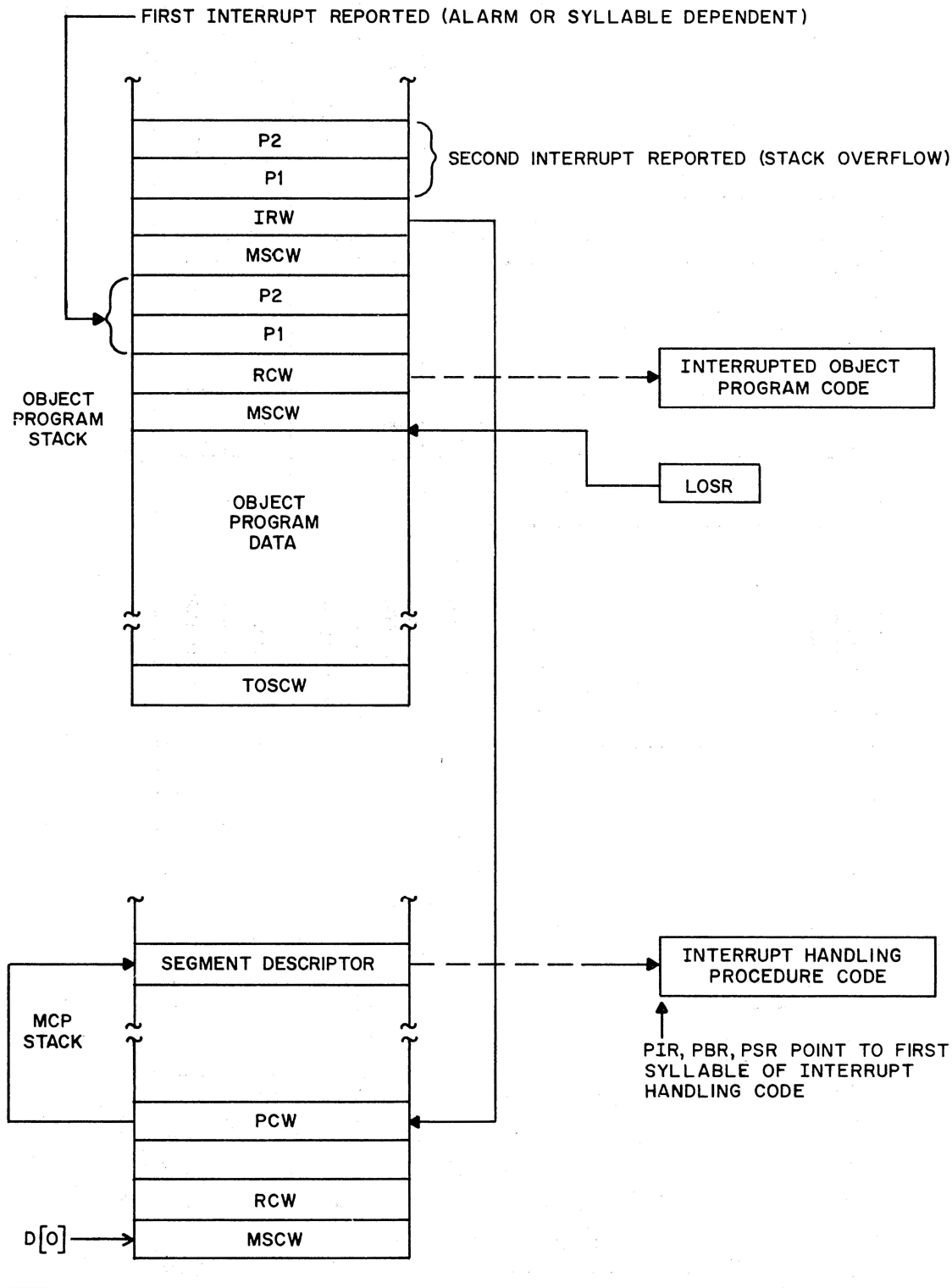


Figure III-2-7. Stack Format Before Reentering Interrupt Procedure to Report Stack Overflow

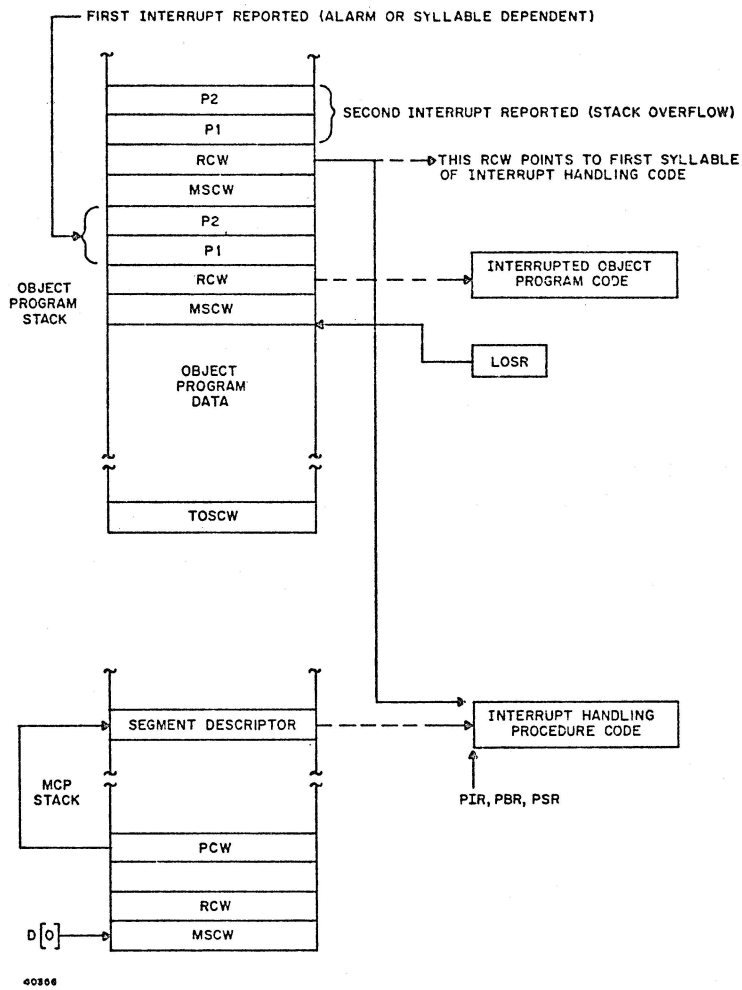


Figure III-2-8. Stack Format After Reentering Interrupt Procedure and Reporting Stack Overflow

the CPM is in Normal State. (When the CPM advances to CM1 and the MCP interrupt procedure is entered, the CPM operates in Control State and the recognition of Interval Timer and External interrupts is inhibited.)

ALARM INTERRUPTS (FIRST PRIORITY)

Detection of an Alarm interrupt causes an immediate entry (or reentry) into the fault control logic. The Control Mode register is advanced and a P1 parameter is formed which identifies all Alarm interrupts which are present in the Fault Condition register. Syllable Dependent interrupts and Interval Timer interrupts (if present) are cleared from the Fault Condition register and the interval timer is disarmed. The MCP interrupt procedure is entered.

SYLLABLE DEPENDENT INTERRUPTS (SECOND PRIORITY)

Detection of a Syllable Dependent interrupt (if no Alarm interrupts are present) causes an immediate entry (or reentry) into the fault control logic. The Control Mode register is advanced and a P1 parameter is formed which identifies all Syllable Dependent interrupts which are present. The MCP interrupt procedure is entered.

SPECIAL INTERRUPTS (THIRD PRIORITY)

STACK OVERFLOW. All Stack Overflow interrupts are processed by the fault control logic; however, a Stack Overflow interrupt will cause an advance of the Control Mode register only if the CPM is in Normal Mode when the interrupt is detected, or if the CPM is in one of the control modes and the interrupt is the second consecutive Stack Overflow interrupt to be reported. A flip-flop in the CPM is set when the first Stack Overflow interrupt is reported while the CPM is in a control mode. This flip-flop is reset each time the Control Mode register is advanced. (The Control Mode register will be advanced by a Stack Overflow interrupt while the CPM is in a control mode only if that flip-flop is set). All Stack Overflow interrupts do cause a P1 parameter reporting the interrupt to be formed. Interval Timer interrupts (if present) are cleared from the Fault Condition register and the Interval Timer is disarmed. The MCP interrupt procedure is entered.

INTERVAL TIMER. Interval Timer interrupts are cleared (and the interval timer is disarmed) when Alarm, Stack Overflow, or External interrupts are reported. All uncleared Interval Timer interrupts cause entry into the fault control logic if the mask is set and either is in normal state or if executing an IDLE in Control state. The Control Mode register is advanced to CM1 (from Normal). (Interval Timer interrupts are inhibited when the CPM is in Control State.) The MCP interrupt procedure is entered.

EXTERNAL INTERRUPTS (FOURTH PRIORITY)

Although External interrupts can occur at any time, these interrupts (as with Interval Timer interrupts) are inhibited when the CPM is in Control State. If an External interrupt occurs when the CPM is in Normal State, the Control Mode register is advanced to CM1, a P1 parameter describing the external interrupts is formed, the Interval

Timer is disarmed, and the MCP interrupt procedure is entered.

Memory Related Interrupts

The memory related interrupts may be irrecoverable memory related errors, recoverable memory related errors (Memory Fail 2), or delayed Memory Fail 1 interrupts. Explanatory information about these errors may be found either in the P2 parameter or in the CPM Fail register. If P2 is not used it will be set to zero. Details regarding the handling of these interrupts are provided in the following paragraphs, and are summarized in table III-2-2.

Table III-2-2. CPM Handling of Memory Related Errors

Source of Request for Memory Access	MF1, MPE, IAE, & PIMR	MF2 (Single Bit Correctable)
Explicit queue fetch	Queued until operator reaches execution unit. Uses P2. (1)	Queued until after operator is executed. Uses CPMFR.
Program buffer fetch	Queued until operator reaches execution unit. Uses P2. (1)	Queued until program buffer is initialized. Uses CPMFR.
Stack buffer fetch	Queued until execution unit tries to use. Uses P2. (1)	Queued until next fetch for stack buffer. Uses CPMFR.
Stack buffer overwrite	Immediate. Uses CPMFR.	N/A
Execution unit store (protected)	Immediate. Uses P2. (1)	N/A MF2 occurs on fetch only.
Execution unit overwrite	Immediate. Uses CPMFR. (PIR may be incorrect).	N/A

(1) PIMR (Processor Internal Memory Related) also uses CPMFR.

IRRECOVERABLE MEMORY RELATED ERRORS

Irrecoverable memory related errors include Memory Fail 1 (MF1) errors (which are discovered by the MCM and reported to the requestor), Memory Parity Errors (MPE) (which are discovered by the communications section), and Processor Internal Memory Related (PIMR) errors (which are detected by the communications section in its interface with the other sections of the CPM). These four types of errors are handled as a group; they are differentiated in the P1 interrupt parameter (PIMR errors causing the PI interrupt). Memory Fail 1 interrupts from all MCM's are combined into a single Alarm interrupt, represented by one bit in the Fault Control register and the Interrupt ID; the identification of which MCM was involved is given in the P2 parameter or in the CPM Fail register, depending on which part of the CPM initiated the memory access. The applicable kinds of accesses by the various parts of the CPM are Explicit Queue accesses (fetches made by the Program Control Unit in response to explicit references in the code), Program Buffer fetches, Stack Buffer fetches and overwrites. The interrupt operations vary with these parts of the CPM as follows.

Explicit Queue Fetch

When a MF1, MPE, IAE, or PIMR error is detected on a memory access for the explicit queue (for operands, etc., as required by the PCU in its look-ahead role), the indication of the error is queued in place of the result of the access. When this operation gets to the execution unit, the interrupt occurs. The P1 parameter differentiates between MF1, MPE, IAE, and PIMR. The MCM control word is placed in the stack as the P2 parameter; it includes the memory address involved in the error. The RCW points to the operator for which the erroneous access was made except for EU-initiated overwrites. If the operation never reaches the Execution Unit (because of a branch or an interrupt), the error is never reported. In the case of the MF1, the MCM Fail register is locked until it is read for some other reason. The CPM Fail register is not involved except for the PIMR error, in which case a further breakdown of the internal error is given in the CPM Fail register. This register is set at the time of the error, not at the time of the interrupt.

Program Buffer Fetch

When a MF1, MPE, IAE, or PIMR error occurs on a fetch for the program buffer, the error condition is queued in the buffer and not reported until the execution unit attempts to execute the operator involved. The interrupt operation is identical to the Explicit Queue case described above.

Stack Buffer Fetch

When a MF1, MPE, IAE, or PIMR error occurs on a fetch for the stack buffer, the error is stored in the stack buffer and the interrupt does not occur until the execution unit attempts to use the erroneous word. The interrupt operation is identical to the Explicit Queue case described above. If the execution unit does not try to use the erroneous word, but tries to push it back into memory, the stack buffer will not perform the memory access, but will leave the contents of memory unchanged.

Stack Buffer Overwrite

When a MF1, MPE, IAE, or PIMR error occurs on a write from the stack buffer, the interrupt occurs immediately. The reason for this is that writes from the stack buffer are initiated only when the execution unit attempts to push the stack down and the stack buffer is full or when the execution unit purges the stack buffer: In either case, the execution unit is waiting for the memory access and cannot complete the current operator until the memory access is complete or, as in the error cases, aborted. There is no P2 parameter: rather, the explanatory information (including the identity of the involved (MCM) is contained in the CPM Fail register. The existence of this related information in the CPMFR is indicated by the zero value of the P2 parameter.

Execution Unit Store (Protected)

When a MF1, MPE, IAE, or PIMR error occurs on a protected store by the execution unit, the P2 parameter is used as described above for the explicit queue. However, this result is returned to the execution unit immediately, and the interrupt is immediate, since the execution unit is waiting for the completion of this memory access.

There is one exception; if the operand is in local memory (associative memory), the processor does not wait for the actual write to main memory. If an error occurs on this write, a Processor Internal interrupt occurs and a bit is set in the CPM Fail register. Notice that this interrupt occurs at an unknown PIR setting, but that no other memory reference may proceed until the interrupt occurs.

Execution Unit Overwrite

When a MF1, MPE, IAE, or PIMR error occurs on an overwrite for the execution unit, the interrupt occurs immediately and the CPM Fail register is used. The operation is exactly as described for the Stack Buffer Overwrites, including the use of the CPM Fail register.

MEMORY FAIL 2

Memory Fail 2 interrupts (single bit errors, corrected) are also combined into a single interrupt in the CPM Fault Condition and Fault Mask registers. The identification of the MCM involved is given in the CPM Fail register. Since the error is corrected (or assumed to be corrected), the corrected data can be and is used. In order to avoid numerous repeats of a given MF2 error, the data is used before action is interrupted to report the error. As in the case of MF1, MPE, IAE, and PIMR errors, the action taken varies with the part of the CPM involved as follows.

Explicit Queue Fetch

When a MF2 occurs on a fetch for the explicit queue, the CPM Fail register is set (including the most significant six bits of address, the MF2 bit and the "reference for execution unit" bit). The indication of the error is queued with the operand and the CPM is not interrupted until after the affected operation has been completed by the execution unit. If that operation never reaches the execution unit, that MF2

is never reported; the CPM Fail register is overwritten by a subsequent error, but a portion of the MCM Fail register remains locked until it is read for some other reason.

Program Buffer Fetch

On a MF2 related to the program buffer, the CPM Fail register is set, but the interrupt does not occur until the next fetch of code from the program buffer.

Interrupt Descriptions

Interrupts which can occur in the CPM are described in the following paragraphs. The interrupts are described in order of their priority. Alarm interrupts are described first, Syllable Dependent interrupts second, Special interrupts third, and External interrupts last.

ALARM INTERRUPTS

Alarm interrupts are caused by conditions which were not expected by the CPM. They inform the system of some detrimental change in environment. Normally, Alarm interrupts result either from programming errors or from hardware failures. The Alarm interrupts cannot be inhibited, and always cause entry into the fault control logic. The fault control logic terminates the current operator, clears the top of stack registers, prepares the stack (MSCW, IRW, P1, P2), and causes the MCP interrupt procedure to be entered. When an Alarm interrupt is cleared from the Fault Condition register, all Syllable Dependent interrupts present in the register are cleared. The Alarm interrupts are:

Loop

Memory Parity

Memory Fail 1

Invalid Address

Stack Underflow

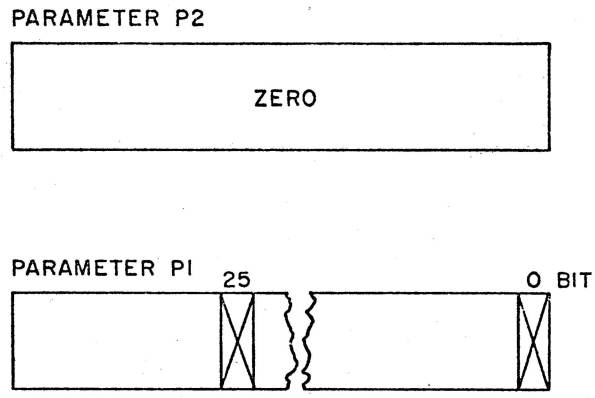
Invalid Program

Processor Internal

Alarm interrupts generally result in termination of the process involved. The single exception occurs during a halt load. The MCP uses an alarm interrupt (Invalid Address) to determine the amount of memory available. The process is not terminated; however, the MCP must advance PIR and PSR by one syllable, as the CPM does not do this.

Loop

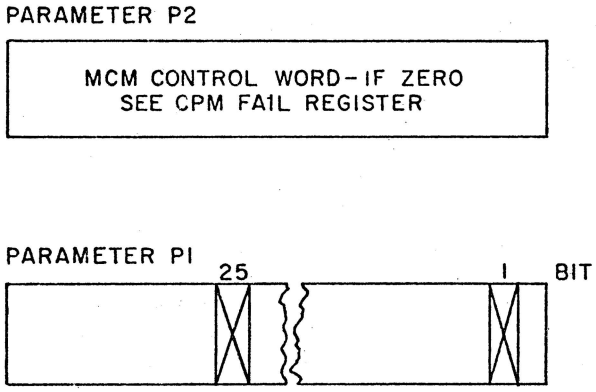
This interrupt occurs when the CPM has expended two seconds in the execution of one operator. (The PCU must issue a micro-operator at least every two seconds.) This interrupt can be caused either by a hardware failure or by bad data. Should this interrupt occur, PIR may not be accurate.



Loop Interrupt Parameters

Memory Parity

This interrupt occurs if the CPM attempts to use a memory word with an even number of 1's. Should this interrupt occur, PIR points to the word containing the operator which initiated the interrupt. Supplementary information describing the error will be contained in the CPM Fail register (see "Memory Related Interrupts").



Memory Parity Interrupt Parameters

Memory Fail 1

This interrupt occurs if any of the following errors occur:

a. Memory Control Word parity error

b. Illegal operation code

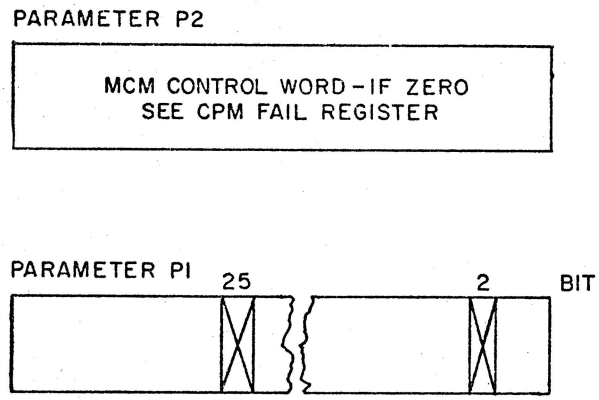
c. Address is for a different Memory Module than requested

d. Data strobe error

e. Internal control error

f. Two bit data error

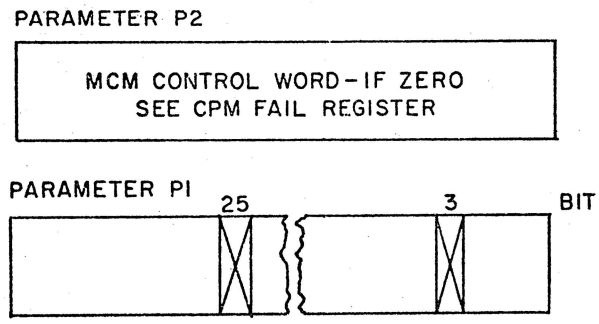
In all of the above cases, supplementary information describing the error will be contained in the MCM Fail register (see "Memory Related Interrupts"). PIR points to the word containing the operator which initiated the interrupt.



Memory Fail 1 Interrupt Parameers

Invalid Address

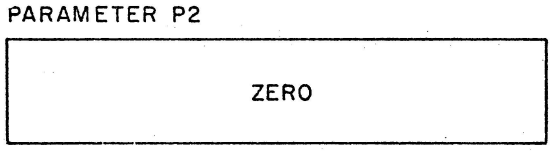
This interrupt occurs when the CPM attempts to access a memory address which is not available to the system. The Memory Module may not exist or it may be inoperative. Should this interrupt occur, PIR points to the syllable that caused the interrupt. Supplementary information is placed in the CPM Fail register (see "Memory Related Interrupts").

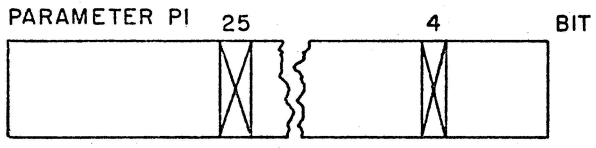


Invalid Address Interrupt Parameters

Stack Underflow

This interrupt occurs if the CPM attempts to move the top of stack (S register setting) to an address less than the address of the most recent MSCW (F register setting) during a stack adjustment. (This could occur as a result either of hardware failure or of a compiler error.) Should this interrupt occur, PIR points to the word containing the operator which initiated the interrupt.





Stack Underflow Interrupt Parameters

Invalid Program Word

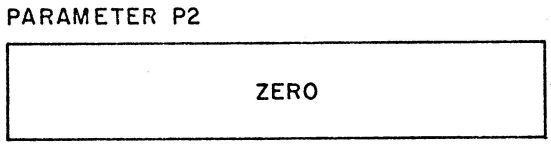
This interrupt occurs under any of the following conditions:

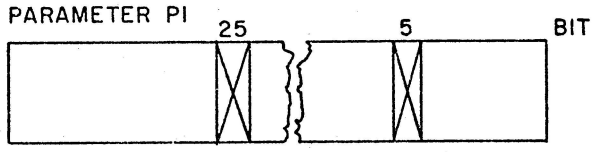
a. An attempt is made to execute a program word which does not have a tag of three (except in Table mode)

b. The Variant code (Escape to 16-bit Instruction, VARI) is detected as the second syllable of a Variant operator.

c. An attempt is made to execute an operator which is considered illegal in Edit mode or Vector mode.

Should this interrupt occur, PIR points to the word containing the operator which initiated the interrupt.

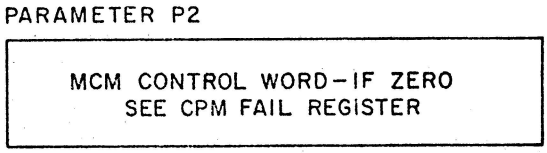


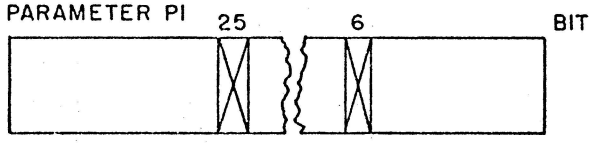


Invalid Program Word Interrupt Parameters

Processor Internal

This interrupt occurs whenever an internal logic failure is detected within the CPM. The CPM Fail register will provide additional information regarding the failure (see table III-2-3). For further information regarding memory related Processor Internal interrupts, see "Memory Related Interrupts". Should this interrupt occur, PIR normally points to the word containing the operator which initiated the interrupt.





Processor Internal Interrupt Parameters

Table III-2-3. Processor Internal Interrupts

Fail Register Indication	Possible Error Source
Execution Unit Parity	A storage
	B storage
	R storage
	W storage
	Stack Buffer
	Data input queue
	Operation register
Execution Unit Residue	Main residue
	Exponent adder
	Shift register
	Allow register
	Operator counter
	Repetition counter
Execution Unit Continuity	Main continuity
	Exponent continuity
	Decoder continuity
Execution Unit decode error	Divide table error
	Illegal operator
	Illegal command
Program Unit overwrite	Execution unit input queue
	Storage unit input queue
Address Unit residue error	Adder to write register
	Execution unit to write register
	Storage unit to write register
	Program unit to write register
COMM Unit residue error	Storage unit to Control word
	Program Buffer to Control word
	Stack Buffer to Control word
	Control word counter
Program Unit internal	Parity error in instruction decode register
	Program Index residue error

SYLLABLE DEPENDENT INTERRUPTS

Syllable Dependent interrupts generally result from programming errors. These interrupts cannot be inhibited, and always cause entry into the fault control logic. The fault control logic terminates the current operator, prepares the stack (MSCW, IRW, P1, P2), and causes the MCP interrupt procedure to be entered. The contents of the top of stack registers may or may not be saved, depending upon the type of interrupt.

Syllable Dependent interrupts are divided into two classes. Class 1 interrupts (identified by the setting of bit 24 of parameter P1) are those interrupts in which the values of PIR, PSR, PBR, and PDR have not been modified by the operator. Class 2 interrupts (identified by the setting of bit 23 of parameter P1) are those interrupts in which the value of PIR, PSR, PBR, and PDR have been changed. Thus, class 1 interrupts permit the operator to be reexecuted; class 2 interrupts prohibit the operator from being reexecuted.

Most Syllable Dependent interrupts occur as class 1 interrupts. The only Syllable Dependent interrupts which can occur as class 2 interrupts are the Invalid Index, Bottom Of Stack, and Sequence Error interrupts. The Syllable Dependent interrupts are:

- Memory Protect
- Invalid Operand
- Divide By Zero
- Exponent Overflow
- Exponent Underflow
- Invalid Index
- Integer Overflow
- Bottom Of Stack
- Presence Bit
- Sequence Error
- Segmented Array
- Programmed Operator
- Privileged Instruction

Memory Protect

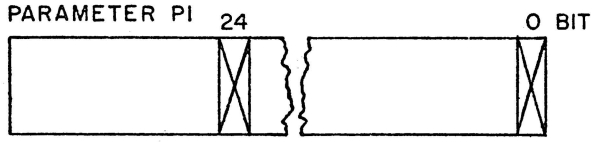
This interrupt occurs when:

a. a store, overwrite, read/lock, or string transfer operation is attempted using a data descriptor that has the read only bit (bit 43) set. The operation is terminated before the memory access. The data descriptor is used as the P2 parameter.

b. a store operation is attempted into a word in memory that has a tag field representing a PCW, RCW, MSCW, or segment descriptor (tag = 1, 3, 5, 7). The memory write is discontinued when bit 48 is detected set in the code word being referenced. The original addressing word is used as the P2 parameter.

PARAMETER P2

DATA DESCRIPTOR WITH BIT 43 SET, OR MEMORY WORD WITH BIT 48 SET, OR NUMBER OF ITEMS BELOW THE MSCW NEEDED TO GET THE DATA DESCRIPTOR

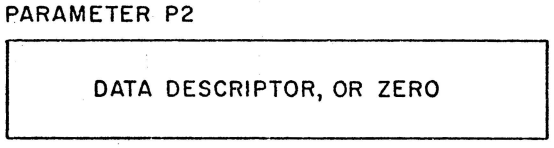


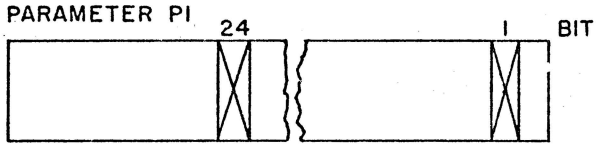
Memory Protect Interrupt Parameters

Invalid Operand

This interrupt occurs when the CPM attempts to execute a valid operator on data which is invalid for that operator or attempts to execute the

invalid operator NVLD. Each operator executes checks to insure that control words and data meet the necessary requirements of the operator. Should this interrupt occur, PIR and PSR are left pointing to the current syllable.

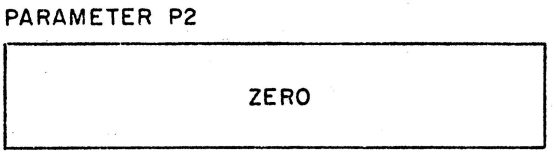


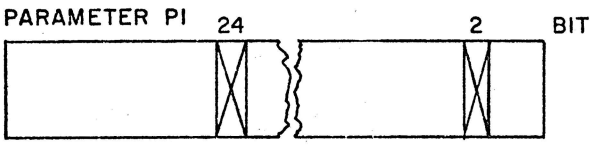


Invalid Operand Interrupt Parameters

Divide By Zero

This interrupt occurs when a division operation is attempted with the divisor (contained in the A register) equal to zero. Should this interrupt occur, PIR and PSR point to the initiating operator, and the divisor and dividend will be left on the top of the stack (below the MSCW, RCW, P1, and P2).

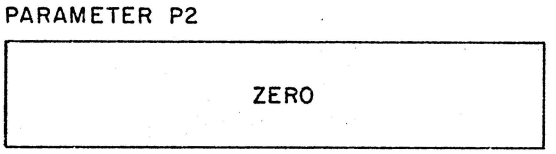


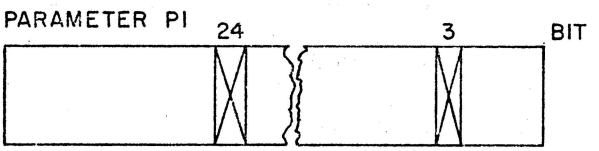


Divide By Zero Interrupt Parameters

Exponent Overflow

This interrupt occurs when the capacity of a positive sign exponent field is exceeded for either single or double precision arithmetic results. Should this interrupt occur, PIR and PSR point to the initiating operator.

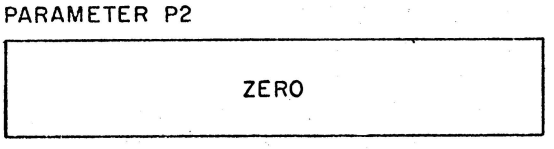


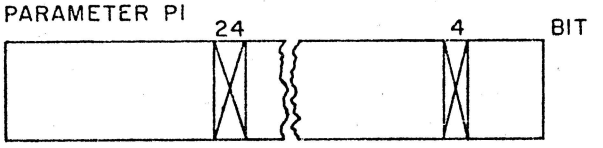


Exponent Overflow Interrupt Parameters

Exponent Underflow

This interrupt occurs when the capacity of a negative sign field is exceeded for either single or double precision arithmetic results. Should this interrupt occur, PIR and PSR point to the initiating operator.





Exponent Underflow Interrupt Parameters

Invalid Index

This interrupt occurs if an attempt is made to index a descriptor by an amount which is less than zero or which is greater than or equal to the upper bound (length) in any of the following operations:

a. Occurs Index

b. Linked List Lookup

c. Index

d. Move Stack

e. Display Update

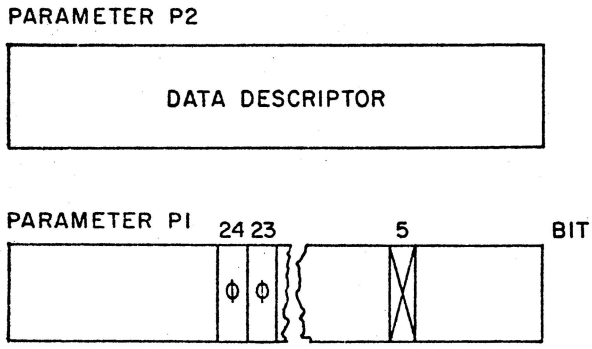
f. Dynamic Branch

g. Stuffed IRW (pseudo operator)

h. Index and Load Name

i. Index and Load Value

Should this interrupt occur, the operation is terminated prematurely. The input operands will be left on the top of the stack (below the MSCW, RCW, P1, and P2). The first two operations in the list above will cause PIR and PSR to point to the initiating operator. In all of the remaining operations, PIR and PSR will not point to the initiating operator. The interrupt occurs as a class 2 interrupt (bit 23 = 1) if an attempt was made to index the Stack Vector Array descriptor (D [0] + 2) during a display update operation using a stack number which is greater than or equal to the length field of the Stack Vector Array descriptor.



Note: Bit 23 and bit 24 may not both be set simultaneously

Invalid Index Interrupt Parameters

Integer Overflow

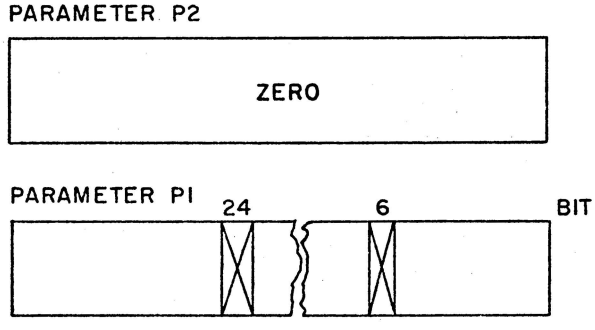
This interrupt occurs upon detection of the attempted use of an operand which exceeds the maximum integer value ($2^{39}-1$) by an operator which requires an integer. The following is a partial list of operators which may cause this interrupt to occur:

a. Integer Divide

b. Intergize Truncate

c. Intergize Rounded

d. Occurs Index



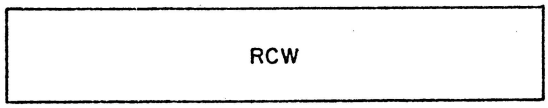
Integer Overflow Interrupt Parameters

Bottom Of Stack

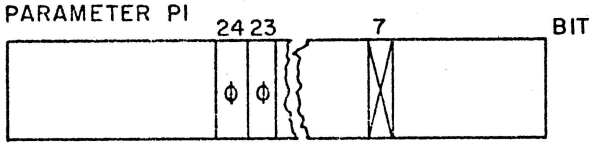
This interrupt occurs if a Return operator or an Exit operator causes the program stack to be cut back to its base. (The F register points to the MSCW located at the BOSR setting plus 1.) The P2 parameter will be a copy of the last accessed RCW.

This interrupt can occur as a class 2 interrupt (bit 23 = 1) if the F register is already pointing to the dummy MSCW (BOSR + 1) before starting to execute the Exit or return operator.

PARAMETER P2



PARAMETER PI



Note: Bit 23 and Bit 24 may not be set simultaneously.

Bottom Of Stack Interrupt Parameters

Presence Bit

This interrupt occurs when an attempt is made to access a word or group of words which are not present in main memory. All operators that access memory with descriptors may be interrupted with this interrupt. The interrupt occurs if an attempt is made to reference memory through a descriptor which has the presence bit (bit 47) reset, indicating that the descriptor points to words which are not present in main memory. There are two classes of presence bit interrupt conditions; data dependent and procedure dependent.

DATA DEPENDENT. Data dependent presence bit interrupt conditions occur when the CPM is seeking data from within its current addressing environment. In all cases except Value Call, recovery is achieved by re-executing the operator upon return from the MCP interrupt procedure. The MCP interrupt procedure makes the absent words present before return is made to the interrupted program. To permit this re-execution, the PIR and PSR settings for the current operator are saved in the RCW. Value Call always sets this RT bit for data dependent interrupts; however, Value Call never sets this RT bit for procedure dependent interrupts. Value Call or pseudo value call will always turn on the VS bit (bit 39), causing the V bit in the MSCW to be turned on. Figure III-2-9 illustrates the PIR, PSR, Exit/Return, RT, VS, and RE bit relationships in the various presence bit interrupt conditions.

ACCIDENTAL ENTRY. Procedures which have been entered accidentally during the VALC operator also require special consideration for the manipulation of PIR and PSR settings for the RCW. The VALC operator is completed after the return operator mechanism when returning from an accidentally entered procedure. A pseudo value call operator provides the facility to continue searching an IRW or data descriptor chain until an operand is located. The pseudo value call operator is activated at the end of a normal return operator if the V bit of the MSCW had been set. The V-bit is set when

PRESENCE BIT INTERRUPT CONDITIONS		P_2	P_1 PRESENCE BIT ID			RETURNING OPERATOR	PIR, PSR NEW RCW	SOFTWARE FUNCTION
			RT (46)	VS (39)	RE (45)			
Data Dependent	Stack Vector DD or Stack D.D. During Reference Through Stuffed IRW	DESC (4)	0	0	0	EXIT	S_n (8)	Locate Not Present D.D. By the IRW. If necessary, make the D.D. present and return an IRW where noted
		IRW (1)	0	0	0	EXIT	S_n (8)	
		IRW (2)	1	1	0	RETURN ·	S_n+ 2 (8)	
		IRW (3)	0	0	0	EXIT	S_n (8)	
		DESC (5)	0	0	1	EXIT	S_n (8)	
	Data Descriptor	DESC (1) (copy)	0	0	0	EXIT	S_n (8)	Search Stack for copies of Not Present D.D. Make MOM and copies present, return present D.D. where noted
		DESC (2) (copy)	1	1	0	RETURN ·	S_n+ 2 (8)	
		DESC (7) (copy)	1	0	0	RETURN ·	S_n (8)	
Procedure Dependent	Stack Vector DD or Stack D.D. During Display Update	DESC (6) (copy)	0	0	0	EXIT	From RCW or PCW	Search Stack for copies of Not Present D.D. Make MOM and copies present, Return D.D. where noted
		DESC (5) (copy)	0	0	1	EXIT		
		DESC (2) (copy)	0	1	0	EXIT		
	Segment Desc	DESC (2) (copy)	0	1	0	EXIT	From RCW or PCW	Locate S.D. (MOM) via copy in P_2 AD Field Of Copy Points to MOM
		DESC (6) (copy	0	0	0	EXIT		
		DESC (5) (copy)	0	1	1	RETURN ·		

1. Enter or IRWL
2. VALC
3. All Operators Except VALC, ENTR, MVST, RETN, IRWL
4. MVST
5. RETN
6. All Operators Except RETN and VALC
7. All Operators Except ENTR, VALC, or IRWL
8. S_n indicates that PIR and PSR point to current operator syllable.

40994

Figure III-2-9. Presence Bit Interrupt Chart

either a VALC or pseudo value call operator enters a procedure accidentally. If a not present segment descriptor causes an interrupt during a return from an accidental entry of value call, a pseudo RT bit (Bit 45) is turned on in P1 so the presence bit procedure will finish with a return instead of an exit if the VS (Bit 39) is also on. The RT bit and pseudo RT bit are used by the software to execute the proper code. The V bit is used by the hardware to change the return into a pseudo value call so the IRW or data descriptor chain may be chased.

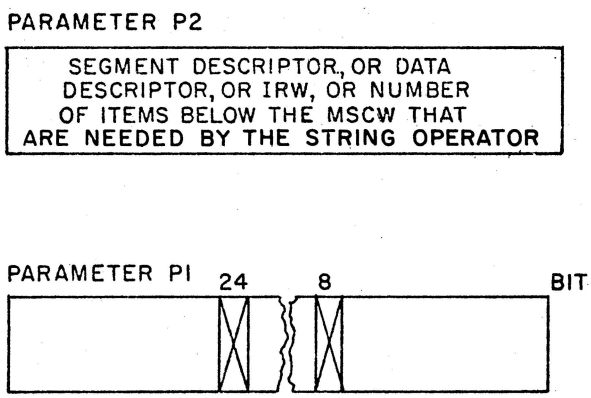
PIR and PSR values, pointing to the next operator syllable, are inserted into the RCW for VALC while the PIR and PSR values from the old RCW are inserted into the RCW for a value call pseudo operator.

All other operators which may incur accidental entries are restarted; therefore, the PIR and PSR settings which point to the current operator syllable are saved in the RCW. The V-bit is set to zero.

PROCEDURE DEPENDENT. Procedure dependent interrupts occur when the CPM is attempting to enter a new addressing environment, or attempting to return to an old addressing environment. These interrupts occur during display update, and also when trying to process a non-present segment descriptor. Recovery is achieved by the Exit operator or the Return operator after the MCP interrupt procedure has made the referenced environment present. Because the CPM has not yet fetched the first operator of the new procedure when this interrupt occurs, the PIR and PSR settings from the PCW (for entry) or the RCW (for return) are stored in the RCW which is made when the MCP interrupt procedure is entered. Thus, when the reference environment is made present, the entry or return is to the referenced environment.

PROGRAM RESTART. Following a Presence Bit interrupt, a program may be restarted either by executing a Return operator or an Exit operator. The Return operator must return either an IRW or a Data Descriptor. The RT bit of the P1 parameter (bit 46) indicates to the MCP interrupt procedure whether to perform an Exit operator (bit 46 is reset) or a Return operator (bit 46 is set) when returning to the interrupted procedure.

PARAMETER P2. During the execution of certain string operators, if a Presence Bit interrupt occurs the P2 parameter may contain a number which indicates the number of items below the MSCW which are needed by the string operator.



Presence Bit Interrupt Parameters

Sequence Error

This interrupt occurs if an indirect reference encounters an invalid condition or reference sequence. Generally, this interrupt is caused ei-

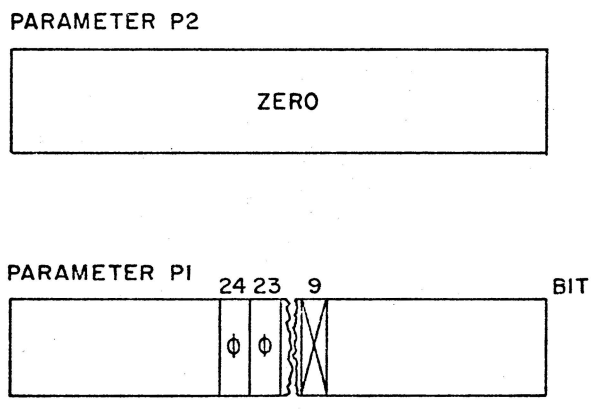
ther by a hardware error or a systems software error, and the MCP will terminate the program which generated the interrupt. The interrupt can occur as a class 2 interrupt (bit 23 = 1) only under the following conditions:

a. When a word other than a Segment Descriptor is fetched relative to the PDR during the final algorithm for the Enter, Exit, or Return operators.

b. When the F register points to a word which is not an MSCW at the beginning of execution of the Exit or Return operators.

c. When tracing back through the DF links of an MSCW chain (DF locates the preceding MSCW in the stack) during an Exit, Return, or Move Stack operation and a word which is not an MSCW is fetched.

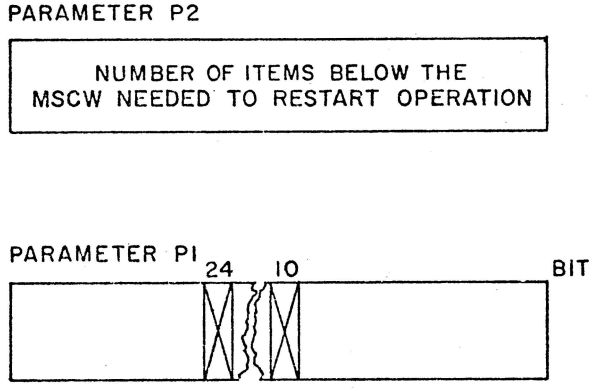
d. When a word which is not a Segment Descriptor is fetched relative to the PDR during a Dynamic Branch operator execution.



Sequence Error Interrupt Parameters

Segmented Array

This interrupt occurs when a string operator attempts to index beyond the end of the current segment of a segmented array. Arrays in main memory may be segmented into groups of 256 words each, bounded on both ends by memory links. The memory link words are created by the MCP with the memory protect bit (bit 48) set. During string operations, each word read from memory is checked to see if bit 48 is set. If such a word is referenced, the Segmented Array interrupt will occur. The P2 parameter will indicate how many words (in the stack below the MSCW, RCW, P1, and P2) are needed to restart the operation after the new segment of the array has been made available in main memory.

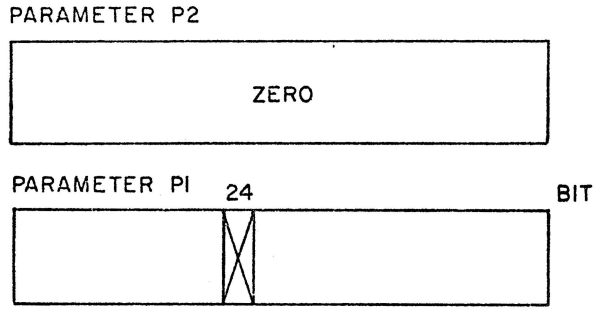


Segmented Array Interrupt Parameters

Programmed Operator

This interrupt occurs if the CPM attempts to execute an operator code which is not currently assigned. The Programmed Operator interrupt acts as a communicate operator to the MCP, and allows the MCP to simulate the action of the operator programmatically, if desired. All

unassigned operator codes cause this interrupt. (None of the unassigned operator codes cause Loop, Invalid Program, or Invalid Operand interrupts. Scan In Time Of Day Clock is an assigned operator: any other variation of Scan In causes the Invalid Operand interrupt.)



Programmed Operator Interrupt Parameters

Privileged Instruction

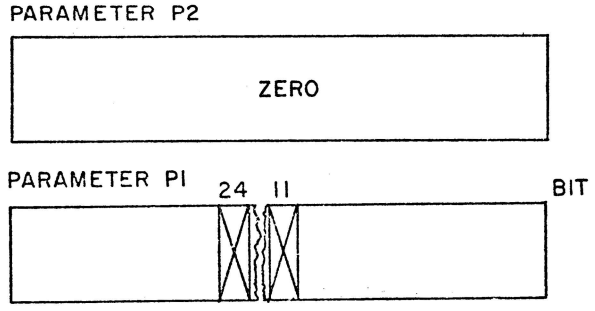
This interrupt occurs if an attempt is made to execute a Control State operator while the CPM is in Normal State. The Control State operators are:

a. Set Interval Timer (SINT)

b. Inhibit Parity (IGPR)

c. Set Memory Inhibits (SINH)

d. Set Memory Limits (SMLT)



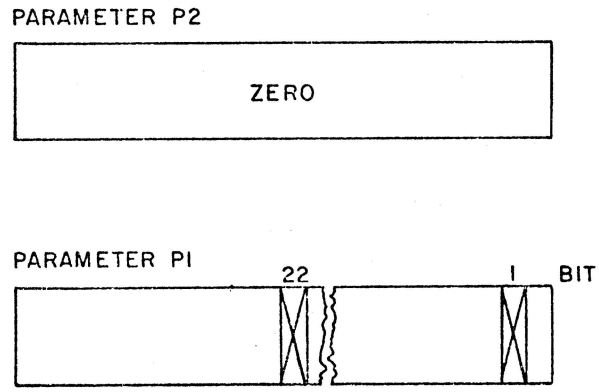
Privileged Instruction Interrupt Parameters

SPECIAL INTERRUPTS

Special interrupts take third priority for processing. There are just two Special interrupts: Stack Overflow and Interval Timer.

Stack Overflow

This interrupt occurs when the Stack Controller senses the use of the highest address allotted for the stack of the program (the S register and the Limit of Stack register (LOSR) point to the same address). The MCP interrupt procedure may either allocate a larger stack area, or it may terminate the program. If the current operator has not been completely executed, PIR and PSR are changed to point to the operator.

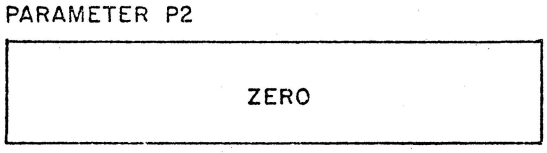


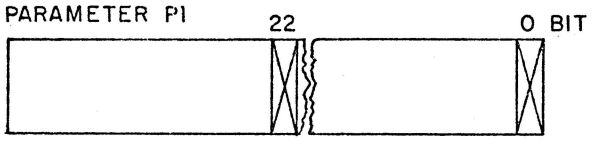
Stack Overflow Interrupt Parameters

Interval Timer

This interrupt occurs if the value in the hardware interval timer is zero and the interval timer is armed. The timer is armed and an initial value is stored by the Set Interval Timer operator (SINT). The count in the timer is decreased every 512 microseconds until the count reaches zero or until the timer is reset. If the timer is still armed when the count reaches zero, the interrupt occurs. The maximum inter-

val to which the timer can be set is one second. This interrupt is used by the MCP to insure that no process can control a CPM for more than one second without giving the MCP a chance to regain control of the CPM.





Interval Timer Interrupt Parameters

External Interrupts

External interrupts are used to inform the MCP of changes in external environment, and also to permit communications between requestor modules (CPM or IOM). Normally, these interrupts result in the momentary interruption of a program while the interrupt is handled or recorded by the MCP. Following the handling of the interrupt, the program is continued. The External interrupts are:

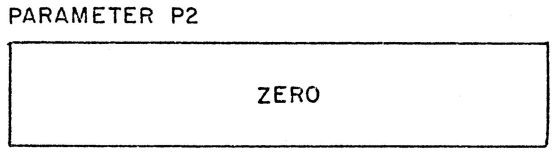
Channel (0 thru 7)

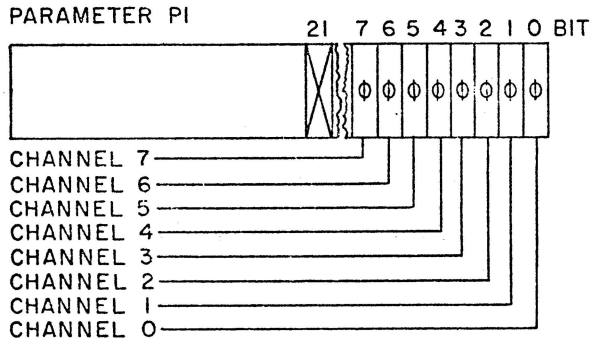
IOM Error (0 thru 7)

Memory Fail 2

Channel Interrupt

This interrupt may be generated by any of the eight possible requestor modules (CPM or IOM). The interrupt identification (parameter P1) indicates the source of the interrupt. This interrupt may be generated to indicate an expected event (such as IO Complete) or it may be generated by the Interrupt Channel N operator (which allows any CPM to interrupt any requestor module).

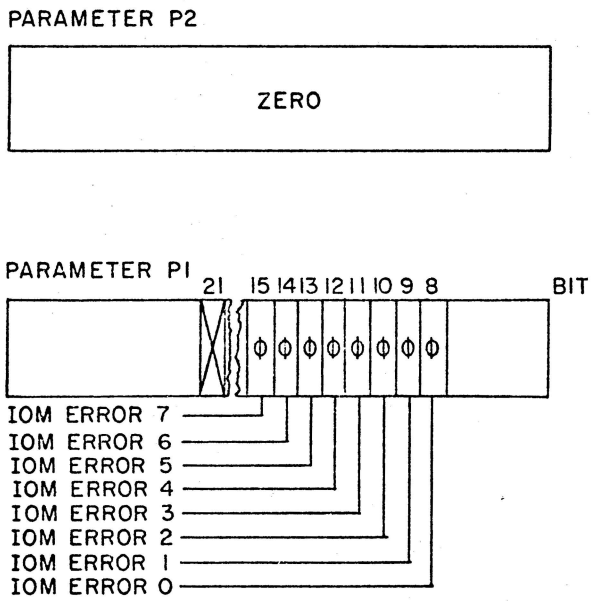




Channel Interrupt (0-7) Parameters

IOM Error Interrupt

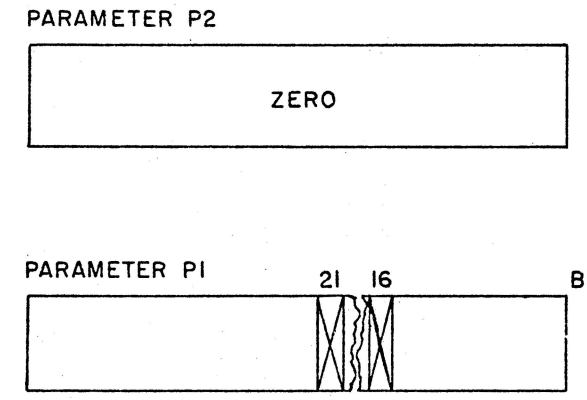
This interrupt may be generated by any of the IO modules in the system. The interrupt identification (parameter P1) indicates the channel (0 thru 7) to which the IOM is connected. This interrupt is used to report errors detected by an IOM which are not device related. If possible, the IOM will link a dummy IOCB into the status queue (RESULTQ). The dummy IOCB will contain a Result Descriptor which will further describe the error. Otherwise, the Fail Result Descriptor will be placed at Home Address (HA) + 5. The error will be described in the IOM Fail register.



IOM ERROR (0-7) Interrupt Parameters

Memory Fail 2

This interrupt occurs if a Memory Control Module detects and corrects a single-bit error. It is transmitted from the MCM (with the corrected data) to the requestor module (CPM or IOM). The MCM Fail register contains the absolute address and the bit number of the word in error. The identification of the MCM involved is contained in the CPM Fail register.



Memory Fail 2 Interrupt Parameters

SECTION 3
OPERATORS

INTRODUCTION

Operators are machine language code generated by the compiler and stored by the master control program in memory in the area allocated to program segments. (Program segments contain no data and are not modified by the processor as the program is executed.) Program segments are sequences of instructions which are moved by the program control unit as 52-bit words from memory into the program buffer. Parity is checked on all 52-bits of each program word as it is brought to the program buffer.

The program buffer, a 32-word, 60-bit IC memory within the processor, locally maintains enough code to keep the processor busy at all times. The buffer may contain 8, 16, 24, or 32 program words. A request is generated to replenish the buffer by fetching 8 words at a time whenever the read pointer is within two words of the writer pointer.

The buffer is interleaved so that it alternately stores all odd-address words from memory in one division of the buffer and stores all even-address words in the other division.

Each division consists of four segments each of which contains four words as shown in Figure III-3-1.

Each program word consists of 48 bits, 3 tag bits, and an overall parity bit. Since information will be extracted from the program buffer in syllable form there is no way to check overall parity. Therefore, as the program word is parsed into six 8-bit syllables while being loaded into the program buffer, parity is also generated on each syllable of the word and stored in the buffer with each word. The parity of each syllable can thus be checked and the integrity of the program word main-

EVEN DIVISION	
Segment 0	Word Address 00
	Word Address 02
	Word Address 04
	Word Address 06
Segment 1	Word Address 08
	Word Address 10
	Word Address 12
	Word Address 14
Segment 2	Word Address 16
	Word Address 18
	Word Address 20
	Word Address 22
Segment 3	Word Address 24
	Word Address 26
	Word Address 28
	Word Address 30

ODD DIVISION	
Segment 0	Word Address 01
	Word Address 03
	Word Address 05
	Word Address 07
Segment 1	Word Address 09
	Word Address 11
	Word Address 13
	Word Address 15
Segment 2	Word Address 17
	Word Address 19
	Word Address 21
	Word Address 23
Segment 3	Word Address 25
	Word Address 27
	Word Address 29
	Word Address 31

41050

Figure III-3-1. Format of Program Buffer Word Storage

tained. Figure III-3-2 illustrates the format of the program buffer word.

Since the B 7700 allows operators to overlap word boundaries, the program buffer is read out in serial manner. Two words are read out of the buffer at the same time, one even address (indicated by the PEB pointer) and one odd (indicated by the POB pointer). These pointers together with the odd/even flip-flops (OEA, OEB, and OEC) then select which of the two words will be left-justified in the barrel switch of the program unit. (These pointers may also be adjusted to facilitate a local branch or entry into an edit table.) The syllables will then be processed left to right in sequence.

An instruction may be either a Value Call, a Name Call, or an operator. The two high-order bits (e.g., bits 7 and 6 in Figure III-3-2) of each instruction determine the type of instruction to be executed.

Instruction Decode Table

Instruction Type	Identification (Bits 7 & 6)	No. of Syllables	Function
Value Call	00	2	Brings an operand into the stack
Name Call	01	2	Brings an IRW into the stack
Operator	1x	1 to 12	Performs the specified operation

Value Call is a two-syllable instruction that brings an operand from memory into the top-of-stack. A concatenation of the two Value Call syllables gives a 14-bit address couple. If the referenced memory location is an indirect reference word or a data descriptor, additional memory accesses are made until the operand is located. The operand is then placed in the top-of-stack register. The operand may be either single-precision or double-precision, causing either one or two words to be loaded into the stack.

Name Call builds an indirect reference word in the stack. Stack adjustment takes place so that the top-of-stack is empty. The six low-order bits of the first syllable for this operator are concatenated with the eight bits of the following syllable to form a 14-bit address couple. The address couple is placed, right-justified into the top-of-stack; the remainder of the top-of-stack register is set to zero. The tag field is set to 001 and the register is marked full.

Operators vary from 1 to 12 syllables in length. The first syllable of each operator indicates the number of additional syllables forming the operator. (The descriptions of the operators in this document indicate the number of syllables associated with the operator.)

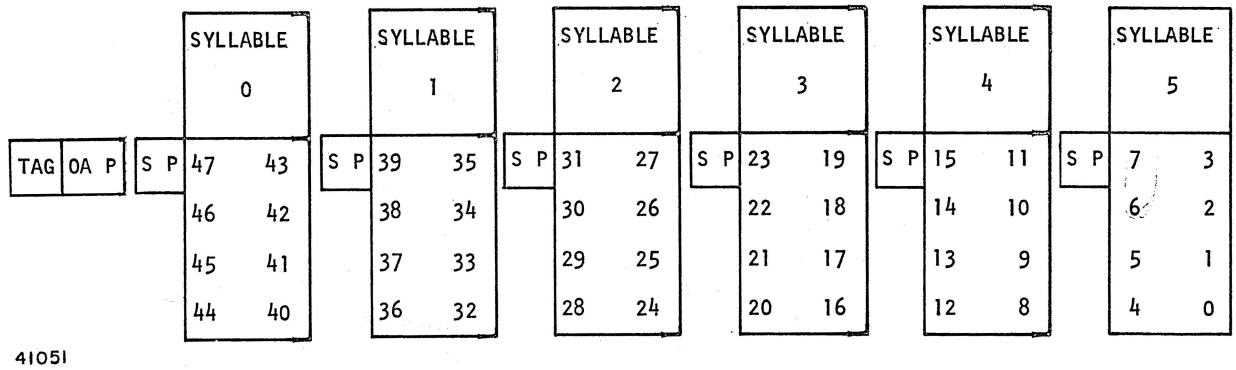


Figure III-3-2. Program Buffer Word Format

Operators work on data as either full words (48 data bits plus 3 tag bits) or as strings of data characters. Word operators work with operands (single-or double-precision) in the top of the stack.

String operators are used for transferring, comparing, scanning, and translating strings of digits, characters, or bytes. In addition, a set of micro-operators (EDIT Mode operators) provides a means of formatting data for input/output. String operators and edit mode operators use source and destination pointers located in the stack to set hardware registers.

In some of the string operators the source pointer may not be used. In this case, an operand may be in the stack; its characters are circulated as it is being used. String operators have an optional update function, producing updated source and destination pointers and counts.

If both the source and destination descriptors have size fields equal to zero, the size registers indicate 8-bit character size. When both a source and destination are required and the size field of one is equal to zero and the other is not, then the size field of the non-zero descriptor is used.

If neither size field is equal to zero and the size fields are not equal and the operator is not Translate or Transfer Words, the invalid-operand interrupt is set and the operator is terminated. The size field is considered equal to zero when the source is an operand.

In the B 7700 Systems operands may be used to represent either numeric or logical information. An operand may be a single-precision (SP) operand or double-precision (DP) operand. Memory word tag bits (bits 50, 49, and 48), when 000, designate an SP operand and, when 010, designate a DP operand.

Logical operands may be either true (ON) or false (OFF). Logical values are the result of Boolean operations or relational operations. Relation operators generate a logical value as the result of an algebraic comparison of two arithmetic expressions. Bit 0 contains the logical value. Relational operators set bit 0, and conditional operators use bit 0 for the decision. Logical (Boolean) operators consider each

bit from 47 to 0 as an individual logical value and operate on the whole operand.

GROUPING OF OPERATORS

Operators may be identified by name, mnemonic, or hexadecimal code. In this document to facilitate reference to the description of the operators, the operators are listed in the appendix in three ways: alphabetically by mnemonic, alphabetically by name, and sequentially by hexadecimal code. In each case the page number of the operator description is given.

When describing operators, considerable redundancy is eliminated by grouping operators with similar functions and only describing their differences. Also, for convenience of the user, operators used for related manipulations (e.g., arithmetic operators i.e. ADD, SUBT, MULT, DIVD, etc.) are described sequentially.

As shown in Figure III-3-3 all central processor program operators are grouped into one of four modes: primary (P), variant (V), edit (E), or vector (Z). Several operators are classed as universal (U) because they can operate in any mode. (The letters in the above parentheses are used in this document as a mode-identifier prefix before the hexadecimal code associated with each operator; e.g., (P)80 indicates a primary mode operator and 80 is the hexadecimal code for the ADD operator.) In this document, the operator descriptions are grouped by mode and preceding each group of descriptions for each mode there is a listing giving the order of specific operator descriptions.

The most frequently used operators are called primary mode operators. Each of the other modes is entered by first executing certain operators in primary mode. The "operator" portion of the primary mode operators begins with the first syllable and may extend for several syllables.

Primary mode operators are described in this document in the following groups: arithmetic, bit, branch, compare, enter edit mode, enter vector mode, index and load, input convert, literal call, logical, pack, relational, scale, stack, store, string, string transfer, subroutine, transfer, type-transfer, miscellaneous, and universal. (In several cases a variant mode operator is conveniently described with a group of primary mode operators.)

Variant mode operators are less frequently used than primary mode operators and extend the number of hexadecimal codes available to identify the operators. Variant mode operators require two syllables. The first syllable of a variant mode operator has the hexadecimal code 95 which is the primary mode operator called Escape to 16-Bit Instruction (the mnemonic for this operator is VARI). The second syllable then gives the actual variant mode operation to be performed. The variant mode operators are described in this document in the following groups: scan, scan while, tag field, unpack, miscellaneous, operators exclusive to the B 7700, and universal operators.

Edit mode operators perform edit functions (such as insert, move, and skip) on strings of data being prepared for output. The Edit mode is entered from the primary mode via one of the enter edit operators

PRIMARY MODE (P)xx

1st \ 2nd	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	0
1	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	1
2	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	2
3	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	VALC	3
4	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	4
5	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	5
6	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	6
7	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	NAMC	7
8	ADD	SUBT	MULT	DIVI	IDIV	RDIV	NTIA	NTGR	LESS	GREQ	GRTR	LSEQ	EQUL	NEQL	CHSN	MULX	8
9	LAND	LOR	LNOT	LEQV	SAME	VARI	BSET	DBST	FLTR	DFTR	ISOL	DISO	INSR	DINS	BRST	DBRS	9
A	BRFL	BRTR	BRUN	EXIT	STBR	NXLN	INDX	RETN	DBFL	DBTR	DBUN	ENTR	EVAL	NXLV	MKST	STFF	A
B	ZERO	ONE	LT8	LT16	PUSH	DLET	EXCH	DUPL	STOD	STON	OVRD	OVRN		LOAD	LT48	MPCW	B
C	SCLF	DSLF	SCRT	DSRT	SCRS	DSRS	SCRF	DSRF	SCRR	DSRR	ICVD	ICVU	SNGT	SNGL	XTND	IMKS	C
D	TEED	PACD	EXSD	TWSD	TWOD	SISO	SXSN	ROFF	TEEU	PACU	EXSU	TWSU	TWOU	EXPU	RTFF	HALT	D
E	TLSD	TGED	TGTD	TLED	TEQD	TNED	TUND	VMOS	TLSU	TGEU	TGTU	TLEU	TEQU	TNEU	TUNU	VMOM	E
F	CLSD	CGED	CGTD	CLED	CEQD	CNED			CLSU	CGEU	CGTU	CLEU	CEQU	CNEU	NOOP	NVLD	F

VARIANT MODE (V)xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
4			JOIN	SPLT	IDLE	SINT	EEXI	DEXI	IGPR		SCNI	SCNO			VHOI		4
8					PAUS	OCRX		NTGD				LOG2				INCN	8
A									SINH		SLMT		FMFR			MVST	A
B					STAG	RTAG	RSUP	RSDN	RPRR	SPRR	RDLK	CBON	LODT	LLLU	SRCH	STOP	B
D	USND	UABD	TWFD	TWTD	SWFD	SWTD		TRNS	USNU	UABU	TWFU	TWTU	SWFU	SWTU		HALT	D
E																	E
F	SLSD	SGED	SGTD	SLED	SEQD	SNED			SLSU	SGEU	SGTU	SLEU	SEQU	SNEU	NOOP	NVLD	F

EDIT MODE (E)xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
D	MINS	MFLT	SFSC	SRSC	RSTF	ENDF	MVNU	MCHR	INOP	INSG	SFDC	SRDC	INSU	INSC	ENDE	HALT	D
F															NOOP	NVLD	F

VECTOR MODE (Z)xx

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	0
1	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	1
2	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	2
3	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	FTCH	3
4	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	4
5	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	5
6	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	6
7	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	STOR	7
E	LDA	LDAI	LDB	LDBI	LDC	LDCI	VXIT	VMOS	STA	STAI	STB	STBI	STC	STCI	VEBR	VMOM	E
F	DLA	DLAI	DLB	DLBI	DLC	DLCI			DSA	DSAI	DSB	DSBI	DSC	DSCI	NOOP	NVLD	F

41052

Figure III-3-3. B 7700 CPM Program Operator Hexadecimal Code Assignments

(EXSD, EXSU, EXPU, TEED, or TEEU). Subsequent edit operators follow as either single micro operators in the program string or as edit operators in a separate table which is executed as a program string. In edit mode the program buffer memory is reduced to 16 words (total available area) for processing the edit operators; the other 16 words contain the primary program syllables.

The basic B 7700 architecture avoids index registers in order to facilitate block or procedure entries. Although this improves the machine's overall performance, it does impede processes of an iterative nature such as the ordinary handling of arrays. The B 7700 overcomes this difficulty with vector mode operation (a variation of string operator edit mode) which permits successive accesses to the elements of an array by using the source, destination, and table pointer areas of the IC memory as index registers and by improved loop control. Vector mode hardware provides additional register capabilities and permits operators to be generated by the compilers to effect improved handling of vectors.

Vector mode is entered from primary mode by using either of two operators: a Single Word Vector Mode (VMOS) entry or Multiple-Word Vector Mode (VMOM) entry. The two enter vector operators assign addresses to the index registers and perform either a VMOS or a VMOM operation. Single-Word Vector Mode forces an automatic one-word branch backward while the processor is in vector mode. The Multiple-Word Vector Mode uses the automatic local branch point detection. The operators Vector Branch and Vector Exit are used only in the Multiple-Word Vector Mode and provide control of program iterations and exiting. Twenty-four vector stack operators link the top of stack with the word addressed by a specific IC register, thus enabling direct, indexable transfers between memory and the top of stack. Forty operators are permitted for vector and matrix manipulations.

In vector mode certain limitations must be considered; for example, the processor is in control state and cannot be interrupted to service other needs. The arrays manipulated by vector mode cannot be segmented and must be present in their entirety while in vector mode. Therefore, the use of vector mode in a general multiprocessing environment must necessarily be restricted; nevertheless, it provides a powerful tool for a particular class of problems.

Detection of an invalid operator condition terminates the operator, and an invalid operator interrupt is set in the processor interrupt register. The processor will proceed to process the interrupt whether it is in normal state or control state.

Invalid instructions are detected by the following methods:

1. Testing for unassigned operator codes. In the B 7700 all unassigned operators cause a programmed operator interrupt.

2. Testing for any value other than 011 in bit positions 50, 49, and 48 of any program word (an attempt to execute something which is not code). This results in an invalid

program word interrupt except when in table mode which allows a tag 0 or a tag 3.

3. Testing for an invalid operator function; for example, an attempt to dial to a non-existent bit. This results in an invalid operand interrupt.

Bit 48 of each word in main memory is a memory protect bit. This bit is ON in all program words, indirect reference words, data descriptors, program descriptors, main memory storage links, and processor-generated control words.

Except for stack pushdowns and the overwrite operators an attempt by a processor to write into a location when the contents of that location has the memory protect bit set will cause a memory-protect interrupt to be set in the processor interrupt register. The overwrite operators will overwrite whatever is in the addressed area. When the string or edit operators attempt to access the source or destination areas they will get a segment array interrupt but when they attempt to access a table they will get a memory-protect interrupt.

Order of Primary Mode Operator Descriptions

Name	Mnemonic	Mode ID & Hex Code	Page
Arithmetic Operators			
Add	ADD	(P)80	3-105
Subtract	SUBT	(P)81	3-106
Multiply	MULT	(P)82	3-106
Extended Multiply	MULX	(P)8F	3-106
Divide	DIVD	(P)83	3-106
Integer Divide	INDIV	(P)84	3-106
Remainder Divide	RDIV	(P)85	3-106
Integerize, Truncated	NTIA	(P)86	3-106
Integerize, Rounded	NTGR	(P)87	3-106
Integerize Rounded, Double Precision	NTGD	(V)87	3-107
Bit Operators			
Bit Set	BSET	(P)96	3-107
Dynamic Bit Set	DBST	(P)97	3-107
Bit Reset	BRST	(P)9E	3-107
Dynamic Bit Reset	DBRS	(P)9F	3-107
Change Sign Bit	CHSN	(P)8E	3-107
Count Binary One's	CBON	(V)BB	3-107
Leading One Test	LOG2	(V)8B	3-107
Branch Operators			
Branch Unconditional	BRUN	(P)A2	3-108
Branch on True	BRTR	(P)A1	3-108

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Branch on False	BRFL	(P)A0	3-108
Dynamic Branch Unconditional	DBUN	(P)AA	3-108
Dynamic Branch True	DBTR	(P)A9	3-108
Dynamic Branch False	DBFL	(P)A8	3-109
Step and Branch	STBR	(P)A4	3-109
Compare Operators			
Compare Characters Greater, Destructive	CGTD	(P)F2	3-110
Compare Characters Greater, Update	CGTU	(P)FA	3-110
Compare Characters Greater or Equal, Destructive	CGED	(P)F1	3-110
Compare Characters Greater or Equal, Update	CGEU	(P)F9	3-110
Compare Characters Equal, Destructive	CEQD	(P)F4	3-110
Compare Characters Equal, Update	CEQU	(P)FC	3-110
Compare Characters Less or Equal, Destructive	CLED	(P)F3	3-111
Compare Characters Less or Equal, Update	CLEU	(P)FB	3-111
Compare Characters Less, Destructive	CLSD	(P)F0	3-111
Compare Characters Less, Update	CLSU	(P)F8	3-111
Compare Characters Not Equal, Destructive	CNED	(P)F5	3-111
Compare Characters Not Equal, Update	CNEU	(P)FD	3-111
Enter Edit Mode Operators			
Table Enter Edit, Destructive	TEED	(P)D0	3-111
Table Enter Edit, Update	TEEU	(P)D8	3-112

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Execute Single Micro, Destructive	EXSD	(P)D2	3-112
Execute Single Micro, Update	EXSU	(P)DA	3-112
Execute Single Micro, Single Pointer Update	EXPU	(P)DD	3-112
Enter Vector Mode Operators			
Single-Word Vector Mode	VMOS	(P)E7	3-117
Multiple-Word Vector Mode	VMOM	(P)EF	3-117
Index and Load Operators			
Index	INDX	(P)A6	3-117
Index and Load Name	NXLN	(P)A5	3-118
Index and Load Value	NXLV	(P)AD	3-118
Load	LOAD	(P)BD	3-118
Load Transparent	LODT	(V)BC	3-118
Input Convert Operators			
Input Convert, Destructive	ICVD	(P)CA	3-119
Input Convert, Update	ICVU	(P)CB	3-119
Literal Call Operators			
Lit Call Zero	ZERO	(P)B0	3-119
Lit Call One	ONE	(P)B1	3-120
Lit Call Eight Bits	LT8	(P)B2	3-120
Lit Call 16 Bits	LT16	(P)B3	3-120
Lit Call 48 Bits	LT48	(P)BE	3-120
Make Program Control Word	MPCW	(P)BF	3-120

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Logical Operators			
Logical AND	LAND	(P)90	3-120
Logical OR	LOR	(P)91	3-121
Logical Negate	LNOT	(P)92	3-121
Logical Equivalence	LEQV	(P)93	3-121
Pack Operators			
Pack, Destructive	PACD	(P)D1	3-121
Pack, Update	PACU	(P)D9	3-122
Relational Operators			
Greater Than	GRTR	(P)8A	3-122
Greater Than or Equal	GREQ	(P)89	3-122
Equal	EQUL	(P)8C	3-122
Less Than or Equal	LSEQ	(P)8B	3-122
Less Than	LESS	(P)88	3-123
Not Equal	NEQL	(P)8D	3-123
Logical Equal	SAME	(P)94	3-123
Scale Operators			
Scale Left	SCLF	(P)C0	3-123
Dynamic Scale Left	DSLF	(P)C1	3-123
Scale Right Save	SCRS	(P)C4	3-124
Dynamic Scale Right Save	DSRS	(P)C5	3-124
Scale Right Truncate	SCRT	(P)C2	3-124

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Dynamic Scale Right Truncate	DSRT	(P)C3	3-124
Scale Right Rounded	SCRR	(P)C8	3-124
Dynamic Scale Right Rounded	DSRR	(P)C9	3-124
Scale Right Final	SCRF	(P)C6	3-124
Dynamic Scale Right Final	DSRF	(P)C7	3-124
Stack Operators			
Exchange	EXCH	(P)B6	3-125
Rotate Stack Down	RSDN	(V)B7	3-125
Rotate Stack Up	RSUP	(V)B6	3-125
Duplicate Top-of-Stack	DUPL	(P)B7	3-125
Delete Top-of-Stack	DLET	(P)B5	3-125
Push Down Stack Registers	PUSH	(P)B4	3-125
Store Operators			
Store Destructive	STOD	(P)B8	3-125
Store Non-Destructive	STON	(P)B9	3-126
Overwrite Destructive	OVRD	(P)BA	3-126
Overwrite Non-Destructive	OVRN	(P)BB	3-126
Read With Lock	RDLK	(V)BA	3-126
String Operators			
String Isolate	SISO	(P)D5	3-128
String Transfer Operators			
Transfer Words, Destructive	TWSD	(P)D3	3-128

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Transfer Words, Update	TWSU	(P)DB	3-129
Transfer Words, Overwrite Destructive	TWOD	(P)D4	3-129
Transfer Words, Overwrite Update	TWOU	(P)DC	3-129
Transfer While Greater, Destructive	TGTD	(P)E2	3-129
Transfer While Greater, Update	TGTU	(P)EA	3-130
Transfer While Greater or Equal, Destructive	TGED	(P)E1	3-130
Transfer While Greater or Equal, Update	TGEU	(P)E9	3-130
Transfer While Equal, Destructive	TEQD	(P)E4	3-130
Transfer While Equal, Update	TEQU	(P)EC	3-130
Transfer While Less or Equal, Destructive	TLED	(P)E3	3-130
Transfer While Less or Equal, Update	TLEU	(P)EB	3-130
Transfer While Less, Destructive	TLSD	(P)E0	3-131
Transfer While Less, Update	TLSU	(P)E8	3-131
Transfer While Not Equal, Destructive	TNED	(P)E5	3-131
Transfer While Not Equal, Update	TNEU	(P)ED	3-131
Transfer While True, Destructive	TWTD	(V)D3	3-131
Transfer While True, Update	TWTU	(V)DB	3-132
Transfer While False, Destructive	TWFD	(V)D2	3-132
Transfer While False, Update	TWFU	(V)DA	3-132
Transfer Unconditional, Destructive	TUND	(P)E6	3-132
Transfer Unconditional, Update	TUNU	(P)EE	3-132

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Subroutine Operators			
Mark Stack	MKST	(P)AE	3-132
Insert Mark Stack	IMKS	(P)CF	3-133
Name Call	NAMC	(P)40 thru (P)7F	3-133
Value Call	VALC	(P)00 thru (P)3F	3-133
Evaluate Descriptor	EVAL	(P)AC	3-136
Enter	ENTR	(P)AB	3-136
Exit	EXIT	(P)A3	3-136
Return	RETN	(P)A7	3-139
Transfer Operators			
Field Transfer	FLTR	(P)98	3-139
Dynamic Field Transfer	DFTR	(P)99	3-139
Field Isolate	ISOL	(P)9A	3-139
Dynamic Field Isolate	DISO	(P)9B	3-142
Field Insert	INSR	(P)9C	3-142
Dynamic Field Insert	DINS	(P)9D	3-142
Type-Transfer Operators			
Set to Single-Precision, Truncated	SNGT	(P)CC	3-142
Set to Single-Precision, Rounded	SNGL	(P)CD	3-142
Set to Double-Precision	XTND	(P)CE	3-142
Set Double to Two Singles	SPLT	(V)43	3-142
Set Two Singles to a Double	JOIN	(V)42	3-142

Order of Primary Mode Operator Descriptions (Cont'd)

Name	Mnemonic	Mode ID & Hex Code	Page
Miscellaneous Primary Mode Operators			
Escape to 16-Bit Instruction	VARI	(P)95	3-143
Read and Clear Overflow Flip-Flop	ROFF	(P)D7	3-143
Read True False Flip-Flop	RTFF	(P)DE	3-143
Set External Sign	SXSN	(P)D6	3-143
Stuff Environment	STFF	(P)AF	3-143
Universal Operators			
Conditional Halt	HALT	(U)DF	3-143
Invalid Operator	NVLD	(U)FF	3-143
No Operation	NOOP	(U)FE	3-143

PRIMARY MODE OPERATORS

Primary mode operators may consist of as many as seven syllables but the first syllable defines the operation.

Arithmetic Operators

Dyadic arithmetic operators require two operands in the top-of-stack storage. These operands are combined by the arithmetic process specified and are replaced with the resulting operand. Both operands may be either single-precision, double-precision, or intermixed types. The specified arithmetic process adapts automatically to the environment: a single-precision process is invoked if both operands are of the single-precision type and a double-precision process is invoked if either operand is of the double-precision type. Each double-precision operand occupies two words. The second word of the operand is an extension of the first word of the operand, i.e., the mantissa of the first word of the operand may be an integer but the mantissa of the second word is always a fraction.

Add, subtract, multiply, and integer divide operations with two integer operands yield an integer result if no overflow occurs. If one or both operands are noninteger or if the result overflows, the result is noninteger.

ADD (ADD) (P)80

The Add operator causes the two top-of-stack operands to be added algebraically and the sum to be left in the top-of-stack.

SUBTRACT (SUBT) (P)81

The Subtract operator causes the top-of-stack operand to be algebraically subtracted from the second operand in the stack and the result to be left in the top-of-stack.

MULTIPLY (MULT) (P)82

The Multiply operator causes the two top-of-stack operands to be algebraically multiplied and the product to be left in the top-of-stack.

EXTENDED MULTIPLY (MULX) (P)8F

The Extended Multiply operator causes the two top-of-stack operands to be algebraically multiplied and a double-precision product to be left in the top-of-stack.

DIVIDE (DIVD) (P)83

The Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand and the quotient to be left in the top-of-stack. If the mantissa of the second operand in the stack is zero, the exponent and quotient are set to zero. If the top-of-stack mantissa is zero, the divide-by-zero interrupt is set. In either case the operation is terminated.

INTEGER DIVIDE (IDIV) (P)84

The Integer Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand and the integer part of the quotient to be left in the top-of-stack in integer form. If the mantissa of the second operand in the stack is zero, the exponent and quotient are set to zero. If the top-of-stack mantissa is zero, the divide-by-zero interrupt is set. In either case the operation is terminated.

REMAINDER DIVIDE (RDIV) (P)85

The Remainder Divide operator causes the second operand in the stack to be algebraically divided by the top-of-stack operand to develop an integer quotient. The remainder of this division is left in the top-of-stack. If this remainder is an integral value, it is in the form of an integer. If the mantissa of the second operand in the stack is zero, the exponent and quotient are set to zero. If the top-of-stack mantissa is zero, the divide-by-zero interrupt is set. In either case the operation is terminated.

INTEGERIZE, TRUNCATED (NTIA) (P)86

The Integerize (Truncated) operator converts the top-of-stack operand to an integer without rounding. If the operand cannot be integerized, i.e., the exponent is greater than the number of leading zeros in the operand. The integer-overflow interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED (NTGR) (P)87

The Integerize (Rounded) operator converts the top-of-stack operand to an integer with rounding. Rounding takes place if the absolute value of the fraction is greater than 4. If the operand cannot be integerized, i.e., the exponent is greater than the number of leading zeros in the operand or a non-integer results from the rounding operation, the integer-overflow interrupt is set and the operation is terminated.

INTEGERIZE ROUNDED, DOUBLE PRECISION (NTGD) (V)87

The Integerize (Rounded, Double Precision) operator converts the top-of-stack operand to a double-precision integer (exponent +13) with rounding.

Bit Operators

Bit operators set or reset bits in the top-of-stack or in the second item in the stack.

BIT SET (BSET) (P)96

The Bit Set operator sets a bit in the top-of-stack. The bit set corresponds to the value of the bit specified by the second syllable of the operator. If the program syllable defining the bit to be set has a value greater than 47, the invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT SET (DBST) (P)97

The Dynamic Bit set operator sets a bit in the second item in the stack. The bit set corresponds to the value of the bit specified by the top-of-stack operand. If the word in the top-of-stack is not an operand an invalid-operand interrupt is set and the operation is terminated. The word is integerized before it is used as a bit number. If after being integerized the operand is less than zero or greater than 47, an invalid-operand interrupt is set and the operation is terminated.

BIT RESET (BRST) (P)9E

The Bit Reset operator resets a bit in the top-of-stack. The bit reset corresponds to the bit specified by the second syllable of the program operator. If the program syllable defining the bit to be reset has a value greater than 47, an invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT RESET (DBRS) (P)9F

The Dynamic Bit Reset operator resets a bit in the second item in the stack. The reset bit corresponds to the value of the bit specified by the top-of-stack operand. If the word in the top-of-stack is not an operand an invalid-operand interrupt is set and the operation is terminated. The word is integerized before it is used as a bit number. If, after being integerized, the operand is less than zero or greater than 47, an invalid-operand interrupt is set and the operation is terminated.

CHANGE SIGN BIT (CHSN) (P)8E

The Change Sign Bit operator complements (changes from 1 to 0 or from 0 to 1) the sign bit (bit 46) of the top-of-stack operand.

COUNT BINARY ONE'S (CBON) (V)BB

The Count Binary One's operator counts the number of binary ones in the information part of the word in the top-of-stack and places this count in the top-of-stack.

LEADING ONE TEST (LOG2) (V)8B

The Leading One Test operator locates the most significant information bit of the word in the top-of-stack. The number of that bit plus one is placed in the top-of-stack. If a one bit is not located, a zero is placed in the top-of-stack.

Branch Operators

Branch instructions function to break the normal sequence of serial instruction fetches. Branching may be either relative to the base address of the current program segment or to a location in some other program segment. Branch operators may be conditional or unconditional. Branch addresses are always checked for possible residency in the local program buffer.

BRANCH UNCONDITIONAL (BRUN) (P)A2

The Branch Unconditional operator replaces the contents of the program index register (PIR) and the program syllable register (PSR) with the next two syllables from the program string. The two syllables following the actual operator syllable provide the new PIR and PSR settings: the three high-order bits are placed in the PSR and the next 13 low-order bits are placed in the PIR.

BRANCH ON TRUE (BRTR) (P)A1

If the low-order bit of the top-of-stack word is a one, the Branch on True operator replaces the contents of the program index register and the program syllable register with the next two syllable positions and the program continues in sequence. Otherwise, the PIR and PSR are advanced three syllable positions and the program string continues in sequence.

BRANCH ON FALSE (BRFL) (P)A0

If the low-order bit of the top-of-stack word is a zero, the Branch on False operator replaces the contents of the program index register and the program syllable register with the next two syllables from the program string. Otherwise, PIR and PSR are advanced three syllable positions and the program string continues in sequence.

DYNAMIC BRANCH UNCONDITIONAL (DBUN) (P)AA

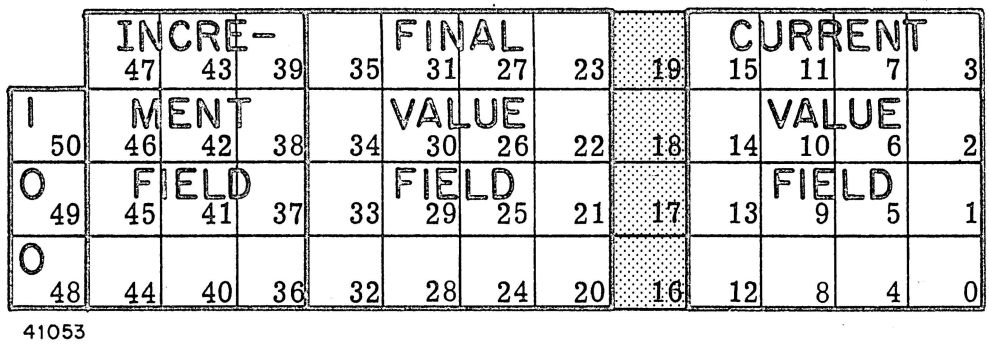
If the top-of-stack word is either a program control word or an indirect reference to a PCW, the Dynamic Branch Unconditional operator branches to the specified syllable of the program segment.

If the top-of-stack word is an operand, the program index register and program syllable register are set according to the contents of this operand as follows: The operand is made into an integer. If it is negative or if it is greater than 16384 the invalid-index interrupt is set and the operation is terminated. If bit zero of the operand is zero, PSR is set to zero; otherwise, if bit zero of the operand is one, PSR is set to three. The next higher-order 13 bits are placed in the PIR.

DYNAMIC BRANCH TRUE (DBTR) (P)A9

If the low-order bit of the second word in the stack is a one and the top-of-stack word is a program control word (PCW) or an indirect reference to a PCW, the Dynamic Branch True operator will cause a branch to the specified syllable in the program segment. Otherwise, a one is added to the PIR and PSR and the program continues in sequence.

If the low-order bit of the second word in the stack is a one and the top-of-stack word is an operand, PIR/PSR are replaced from this operand as in the DBUN operator. Otherwise, PIR and PSR are advanced and the program string continues in sequence.



41053

Figure II-3-4. Step Index Word Format

DYNAMIC BRANCH FALSE (DBFL) (P)A8

If the low-order bit of the second word in the stack is a zero, and the top-of-stack word is a program control word or an indirect reference to a PCW, the Dynamic Branch False operator causes a branch to the specified syllable of the program segment. Otherwise, the PIR/PSR are continued in sequence.

If the low-order bit of the second word in the stack is a zero and the top-of-stack word is an operand, PIR/PSR are replaced from this operand as in the DBUN operator. Otherwise, PIR and PSR are advanced and the program string is continued in sequence.

STEP AND BRANCH (STBR) (P)A4

The Step and Branch operator is initiated with a reference to either a step index word (SIW) or an operand in the top-of-stack. The target item may be reached through a chain of indirect reference words, indexed data descriptors, and/or accidental entries. The format of the SIW is shown below:

If the target item is an SIW, the increment field of the SIW is added to the current-value field of the SIW, and the SIW is replaced in memory. If the current-value field after adding the increment is less than or equal to the final-value field, then PIR and PSR are advanced three syllable positions, the program string is continued in sequence, and an operand representing a Boolean value "true" (bit 0 ON) is left in the top-of-stack. If the current-value field is greater than the final-value field, the program takes the branch by replacing PIR and PSR with the next two syllables from the program string. If the branch is taken, no Boolean is left in the top-of-stack.

If the target item is an operand, the operand will be left in the second stack position, the top-of-stack will be set to zero, representing a Boolean value "false," and PIR and PSR will be advanced to the next operator. In the B 7700, the operand will be left in the stack only if the STBR operator is followed by a Branch on False (BRFL) operator. If STBR is not followed by BRFL, the second stack position will be left empty.

If the target item is other than an operand or SIW, the invalid-operand interrupt is set.

Compare Operators

The compare operators perform the specified compare of two strings of

data. The true/false flip-flop is conditioned by the results of the compare.

COMPARE CHARACTERS GREATER, DESTRUCTIVE (CGTD) (P)F2

The Compare Characters Greater Destructive operator makes a character-by-character comparison of two strings of data until it finds an unequal pair. (All comparisons are by the binary character position in the collating sequence.) If the characters in the B string (destination) are greater than the characters in the A string (source), then the true/false flip-flop is set to one; otherwise, the true/false flip-flop is set to zero. If the repeat count is less than or equal to zero, the true/false flip-flop is reset.

The top-of-stack is an operand which specifies the number of characters to be compared. The second item in the stack is an operand or descriptor pointing at the source character string against which comparisons are to be made. The third item in the stack is a descriptor pointing to the character string to be compared. If either of the data strings has the memory protect bit ON (bit 48=1), the segmented array interrupt is set, and the operation is terminated.

COMPARE CHARACTERS GREATER, UPDATE (CGTU) (P)FA

The Compare Characters Greater, Update operator performs a Compare Characters Greater, Destruction operation except that the accesses to memory continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE (CGED) (P)F1

The Compare Characters Greater or Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the destination is greater than or equal to the source.

COMPARE CHARACTERS GREATER OR EQUAL, UPDATE (CGEU) (P)F9

The Compare Characters Greater or Equal, Update operator performs a Compare Characters Greater or Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS EQUAL, DESTRUCTIVE (CEQD) (P)F4

The Compare Characters Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the source is equalto the destination.

COMPARE CHARACTERS EQUAL, UPDATE (CEQU) (P)FC

The Compare Characters Equal, Update operator performs a Compare Characters Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE (CLED) (P)F3

The Compare Characters Less Or Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the destination is less than or equal to the source.

COMPARE CHARACTERS LESS OR EQUAL, UPDATE (CLEU) (P)FB

The Compare Characters Less or Equal, Update operator performs a Compare Less or Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS LESS, DESTRUCTIVE (CLSD) (P)F0

The Compare Characters Less, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the destination is less than the source.

COMPARE CHARACTERS LESS, UPDATE (CLSU) (P)F8

The Compare Characters Less, Update operator performs a Compare Characters Less, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE (CNED) (P)F5

The Compare Characters Not Equal, Destructive operator performs a Compare Characters Greater, Destructive operation except that the true/false flip-flop is set to true if the source is not equal to the destination.

COMPARE CHARACTERS NOT EQUAL, UPDATE (CNEU) (P)FD

The Compare Characters Not Equal, Update operator performs a Compare Characters Not Equal, Destructive operation except that memory accesses continue until the repeat count is exhausted. At the completion of the operation, the source and destination pointers are updated.

Enter Edit Mode Operators

Enter edit mode operators provide the means for transition from primary mode operation to edit mode operation. The edit mode operators in a program string are entered via the Execute Single Micro or Single Pointer. The edit mode operators may also be in a table and in which case they are entered by the Table Enter Edit operator. (See also the descriptions under "Edit Mode Operators.")

TABLE ENTER EDIT, DESTRUCTIVE (TEED) (P)D0

The Table Enter Edit, Destructive operator is used to control edit micro instructions which are contained in memory as a table rather than as part of the normal program string. This operator causes characters to be transferred from the source string to the destination string.

The transfer is under control of the string of edit micro-operators which are located by the table pointer.

The top-of-stack word (a descriptor) is the table pointer, the second word (a single-precision operand or descriptor) in the stack is the source pointer, and the third word in the stack (a descriptor) is the destination pointer. If the first word in the stack is not a descriptor, the invalid-operand interrupt is set and the operation is terminated. If the second item in the stack is a single-precision operand, it is a source string. If the third item in the stack is not a descriptor, the invalid-operand interrupt is set and the operation is terminated. In table mode, the micro-operator words can be tagged as single-word operands (tag-0).

TABLE ENTER EDIT, UPDATE (TEEU) (P)D8

The Table Enter Edit, Update operator performs a Table Enter Edit Destructive operation. At the completion of the operation, the source pointer and destination pointer are updated.

EXECUTE SINGLE MICRO, DESTRUCTIVE (EXSD) (P)D2

The Execute Single Micro, Destructive operator transfers characters from the source string to the destination string under the control of the single micro-operator which follows this operator syllable. The first item in the stack is a single-precision operand that defines the field length and is used as a micro-operator repeat field. The second item in the stack is the source pointer, the third item in the stack is the destination pointer.

EXECUTE SINGLE MICRO, UPDATE (EXSU) (P)DA

The Execute Single Micro, Update operator performs an Execute Single Micro, Destructive operation. At the completion of the operation, the source pointer and destination pointer are updated.

EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE (EXPU) (P)DD

The Execute Single Micro, Single Pointer Update operator performs an Execute Single Micro, Destructive operation. At the completion of the operation, the pointer is updated.

The top-of-stack operand is used as a micro-operator repeat field. The second item in the stack is used to set both the source and destination pointers. Only the destination pointer is updated.

Enter Vector Mode Operators

The enter vector mode operators provide the means of transition from primary mode to vector mode. Either one of two operators are available to enter vector mode: for operations using instructions in only one program word, operator VMOS is used; for operations involving instructions in more than one program word, operator VMOM is used. (See also the descriptions under "Vector Mode Operators.")

Vector mode hardware provides increased efficiency in the ordinary handling of arrays that frequently dominates a FORTRAN or an ALGOL program. For example, when processing the following FORTRAN DO loop:

```
DO 10 I = 1, 100
```

```
      A(I)  = B(I)  + C(I)

   10 CONTINUE
```

Each trip through the loop (for each value of I) requires a descriptor for each of the three arrays and the value of I with which to index each descriptor. This means six memory accesses, in addition to code fetching and execution.

The Vector mode hardware does the following:

1. Utilizes registers to hold the actual addresses of array elements that are referenced.

2. Uses additional registers to contain the increment values used for altering the addresses (indexing) to reference successive array elements.

3. Uses one register to contain a "count" or "length" that controls the number of iterations.

4. Provides for cycling through one (single-word mode) or more (multiple-word mode) words of code for each iteration.

5. Introduces new operators (for use while in vector mode) to load into and store from the top-of-stack, to control the number of iterations, and to provide exiting from vector mode.

6. Provides two primary mode operators used to enter vector mode.

Seven IC memory locations are used as the registers mentioned above to hold the three absolute addresses, the three corresponding increment values, and the length.

The three addresses are referred to as A, B, and C, respectively.

These registers are loaded automatically from the stack upon execution of either of two enter vector mode operators.

To example above (the FORTRAN DO loop) can be expressed in ESPOL as follows:

```
DO    VECTORMODE    ([1,1,1]
A[*],B[*],C[*], for 100)

BEGIN

A=B+C;

INCREMENT A,B,C;

END;
```

The example has specified three increment values, three beginning addresses, a length (or number of iterations), and the operations to be performed on the array elements in each iteration. From this information the compiler generates the following:

1. Primary mode code to place the seven parameters in the stack.

2. The enter vector mode operator.

3. Vector mode code to perform the operations on the array elements.

Before entering vector mode, the values to be stored in the IC memory registers (three absolute addresses,

their corresponding increments, and an optional length) must be placed in the stack. They are arranged in the stack in the following order (from the top-of-stack down):

Pointer C	(descriptor)
Length (optional)	(operand)
Pointer A	(descriptor)
Pointer B	(descriptor)
Pointer C increment	(operand)
Pointer A increment	(operand)
Pointer B increment	(operand)

Length specifies the number of iterations through the code to be executed while in vector mode, usually the number of elements in the arrays being manipulated. The presence of a length value in the stack is indicated by bit 44=1 in Pointer C. If bit 44=0, a default length of 2^{20}-1 is stored in the IC length register. Bit 44 (segmented bit) must be OFF in Pointer A and Pointer B. (The software ascertains that bit 44 is OFF in Pointer C before using it to indicate the presence of a length value.)

The seven parameters are inserted in IC memory as follows:

```
BRS3 <-- Pointer C   [19:20]   (+ Pointer C   [35:16]   if I = 1)

BRS7 <-- LENGTH   [19:20] (or 2^20-1)

BRS1 <-- Pointer A   [19:20]   (+ Pointer A   [39:20]   if I = 1)
BRS2 <-- Pointer B   [19:20]   (+ Pointer B   [39:20]   if I = 1)

IRS3 <-- Pointer C increment  [19:20]
IRS1 <-- Pointer A increment  [19:20]
IRS2 <-- Pointer B increment  [19:20]
```

I is the indexed bit, bit 45, in the descriptor.

The enter vector mode operator may be terminated by one of the following interrupts:

a. INVALID OP: Pointer A, B or C not tagged as a data descriptor or Pointer A or B has bit 44=1.

b. MEMORY PROTECT: Pointer A is read only (bit 43=1).

c. PRESENCE BIT: Pointer A, B or C has bit 47=0.

d. PROGRAMMED OPERATOR: Processor is not equipped with vector mode hardware.

If the entry to vector mode is the single-word vector mode (VMOS) operator, the single word of code following that entry is executed a number of times equal to the length parameter. Each time the word is executed length is decremented by one until it becomes zero. Then vector mode is exited and normal operation continues with the next word of code in sequence.

When the entry to vector mode is the multiple-word (VMOM) operator, whatever code that follows it is executed under vector mode rules. The vector mode operators explained be-

low are used only in conjunction with the VMOM operator.

a. Vector Branch (VEBR) is a three-syllable operator. The two syllables following the operator name contain the branch address. The Vector Branch operator examines length. If it is greater than zero, length is decremented by one, the next two program syllables containing the branch address are skipped, and the program is resumed at the following syllable. If the examined length is zero, vector mode is exited, and normal operation commences with the program word located by the branch address.

b. Vector Exit (VXIT) operator causes the program to return to normal operation.

There are 24 Vector Stack operators (with a common syllable format) which are used to move operands between the top-of-stack and absolute memory addresses pointed to by descriptors. Variations of this syllable provide the capabilities of storing or loading the top-of-stack with a single - or double-precision operand and choosing whether or not to increment the pointer. If the memory address is protected, the following recovery procedure is followed:

a. If a store operation, vector mode is terminated with a Memory Protect Interrupt.

b. If a load operation, then:

(1) If the length parameter was passed to the vector mode, vector mode is terminated with a Memory Protect Interrupt.

(2) If no length parameter was passed to the vector mode, vector mode is terminated but no interrupt is set. The stack is then cut back as defined by the operator that sensed the Memory Protect condition (refer to RA and RB below).

The word for which a protect is sensed is not marked present in the stack.

The format of the vector operator syllable is defined as follows:

a. If a length is not passed when vector mode is entered, the format is as follows:

0	LS	RA	RB	D	A1	A0	I

where 0, the high-order bit, must be OFF (0).

b. When a length is passed, the format is as follows:

1	1	1	LS	D	A1	A0	I

where the three high-order bits must be ON (1).

In either format:

Bit	Description
D	Double-precision bit. If D=0, load or store a single-precision operand. If D=1, load or store a double-precision operand.
RA	If a memory protect interrupt is sensed and no length is passed to the vector mode and RA=0, the top-of-stack word is deleted. If RA=1, the top-of-stack word is not deleted.
RB	Same as the RA bit except that it governs the action taken on the second word of the stack.
LS	Bit is OFF (0) for a top-of-stack load operator and ON (1) for a top-of-stack store operator.
A1, A0	Selects the IC memory address register.

A1	A0	
0	0	Load from Pointer A (BRS1)
0	1	Load from Pointer B (BRS2)
1	0	Load from Pointer C (BRS3)

I	When I equals 1, the pointer used for the memory address is increased by its corresponding pointer increment following the load or store operator. When I equals 0, the pointer increment is inhibited.

The Vector Stack operators are described under "Vector Mode Operators."

Two other operators (FTCH and STOR) are used to load/store the top-of-stack from/to relative memory addresses designated by an address couple. They are enabled only when a length is passed by the vector mode entry. (The operators FTCH and STOR are described under "Vector Mode Operators.")

External interrupts are disabled during Vector Mode. Exponent underflow interrupts are inhibited for arthmetic operators, and in lieu of the interrupt, an answer of zero is returned and the TFFF is turned ON.

No facilities are provided for recovery from interrupts that occur while in vector mode. P1 is returned with bit 19=1.

Vector mode operation allows only restricted use of the B 7700 operator set. All string operators and

subroutine operators, as well as those that pseudo-call them, are forbidden with the exception of NOOP, NVLD, and branch instructions. LIT 48 and branch instructions are not used in the single-word vector mode (VMOS) because of their size.

Vector mode operations can be used only on processors equipped with the vector mode hardware. On other systems, an attempt to enter vector mode results in a "programmed operator" interrupt.

SINGLE-WORD VECTOR MODE (VMOS) (P)E7

The Single-Word Vector Mode operator is the primary mode operator used to access vector mode to perform on a vector those operations defined by one program word. VMOS extracts the seven parameters (described above) from the stack, inserts them in their IC address, and, after skipping up to five syllables, repeatedly executes the next complete word of program code from the program register. VMOS is inhibited from accessing additional program code, thus causing vector mode exit.

If the descriptor in the top-of-stack has bit 44 ON, then the second item in the stack is the length operand and it gives the iteration repeat count; otherwise, the default repeat count of 1,048,575 is used.

Each of the three descriptors (pointers) represents a full or partial array of operands which will be operated on repeatedly by the same word of code.

MULTIPLE-WORD VECTOR MODE (VMOM) (P)EF

The Multiple-Word Vector Mode operator provides access to the vector mode operators for multiple program words. The VMOM operator performs a VMOS operation supplemented by operator Vecor Branch which leads to additional program word loops. The special vector exit operator VXIT returns the program to normal operation.

Index and Load Operators

The index and load operators provide the means to index the top-of-stack word and the means to load an operand or descriptor into the top-of-stack.

INDEX (INDX) (P)A6

The two top-of-stack items are a descriptor (or indirect reference to a descriptor) and an operand. The operand is used to index the descriptor. The Index operator places the integerized value of the second item in the stack into the 20-bit length/index field of the descriptor in the top-of-stack. The descriptor is marked indexed (i.e., bit 45 is set to "one").

If the word in the top-of-stack is an operand, the top-of-stack operand is exchanged with the second-item operand. If the word in the top-of-stack is neither a descriptor nor an indirect reference word pointing to a descriptor, the invalid-operand interrupt is set and the operation is terminated.

If the indexing value is negative or greater than or equal to the length field of the descriptor, the invalid-index interrupt is set and the operation is terminated.

If the descriptor represents an array which is segmented, the index is partitioned into two portions by dividing it by the proper divisor determined by the type of data referenced by the descriptor, (D.P. word-128, S.P. word-256, 4-bit digit-3072, 6-bit character-2048, or 8-bit byte-1536). The quotient is used as an index to the given descriptor to fetch the array-row descriptor. The remainder is used to index the row descriptor.

If the double-precision bit (bit 40) in the descriptor is "one", the index value in the second item is doubled. The balance of the operation is as described in the first paragraph of this operator.

INDEX AND LOAD NAME (NXLN) (P)A5

The Index and Load Name operator performs an Index operation. After the word in the top-of-stack is indexed, the data descriptor pointed to by this word is brought to the top-of-stack, the copy bit (bit 46) of the data descriptor is set to "one", and the top-of-stack is marked full.

If the presence bit (bit 47) is OFF, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the top-of-stack is not a data descriptor, the invalid-operand interrupt is set and the operation is terminated. If the data descriptor accessed by the indexed word in the top-of-stack has the index bit (bit 45) set to "one", the invalid-operand interrupt is set and the operation is terminated.

INDEX AND LOAD VALUE (NXLV) (P)AD

The Index and Load Value operator performs an Index operation. After the word in the top-of-stack is indexed, the operand pointed to by this descriptor is brought to the top-of-stack. The top-of-stack is marked full.

If the word accessed is other than an operand the invalid-operand interrupt is set and the operator is terminated.

LOAD (LOAD) (P)BD

The Load operator places the word addressed by the indirect reference word or by the indexed data descriptor in the top-of-stack.

If at the start of this operator the top-of-stack contains other than a data descriptor or an indirect reference word pointing at a data descriptor, the invalid-operand interrupt is set and the operation is terminated.

If the word pointed at by the data descriptor is another data descriptor, that descriptor is marked as a copy (copy bit [bit 46] is set to

"one") and if the presence bit (bit 47) is OFF, the address of the original data descriptor is placed in the field defined by bits 19:20 of the copy in the stack.

LOAD TRANSPARENT (LODT) (V)BC

If the top-of-stack word is a data descriptor or an indirect reference word, the Load Transparent operator performs a Load operation; otherwise, the word addressed by the 20 least-significant bits of the top-of-stack word is loaded to the top-of-stack. Copy-bit action does not occur.

Input Convert Operators

The input convert operators convert the various character sets (digit, BLC, EBCDIC, or ASCII) to operands for arithmetic operations.

INPUT CONVERT, DESTRUCTIVE (ICVD) (P)CA

The Input Convert, Destructive operator converts 4-bit digit, or 6-bit BCL, or 8-bit EBCDIC (or ASCII) to an operand for internal arithmetic operations.

The first item in the stack is an operand that is integerized to form the repeat field. The second item in the stack is a descriptor used as a source pointer.

The specified number of characters are transferred from the source string to the top-of-stack. Only the numeric portion of the character is transferred. The transferred string is converted to a double-precision operand if the length is greater than 12. If a double-precision operand is produced, the true/false flip-flop is set to false; otherwise, it is set to true. The sign bit of the operand is set negative if the zone of the last character transferred is $10_{(2)}$ (for six-bit characters) or $1101_{(2)}$ (for eight-bit characters). At the completion of the operator the second item in the stack is marked full. The tag field is set to indicate a single - or double-precision operand.

INPUT, CONVERT, UPDATE (ICVU) (P)CB

The Input Convert, Update operator performs an Input Convert, Destructive operation. At the completion of the operation, the source pointer is updated.

Literal Call Operators

The literal call operators place defined-value operands in the top-of-stack.

LIT CALL ZERO (ZERO) (P)BO

The Lit Call Zero operator places in the top-of-stack a single-precision operand with a value of zero.

LIT CALL ONE (ONE) (P)B1

The Lit Call One operator places in the top-of-stack a single-precision operand with a value of one.

LIT CALL 8 BITS (LT8) (P)B2

The Lit Call 8 Bits operator places in the top-of-stack a single-precision operand equal in value to the second syllable of this operator.

LIT CALL 16 BITS (LT16) (P)B3

The Lit Call 16 Bits operator places in the top-of-stack a single-precision operand equal in value to the second and third syllables of this operator.

LIT CALL 48 BITS (LT48) (P)BE

The Lit Call 48 Bits operator places in the top-of-stack a single-precision operand equal in value to the next program word.

NOTE

Since the literal is synchronized by word, this operator can be 7-12 syllables long. Any unused syllables are filled in with the invalid operator code.

MAKE PROGRAM CONTROL WORD (MPCW) (P)BF

The Make Program Control Word operator performs a Lit Call 48 Bits operation except that the tag field is set to 111 to indicate a program control word and the stack number field of the PCW is inserted from the stack number register.

Logical Operators

Logical operators operate on the two top-of-stack operands bit for bit from bit 47 thru bit 0 to obtain logical values (48 logical values for single-precision operands and 96 for double-precision operands) which are left as the top-of-stack operand. If only one of the operands associated with LAND, LOR, LNOT, or LEQV is a double-precision operand, then the other operand will be extended with zeros. Logical operators may be used to operate on logical, string, or numeric operands.

LOGICAL AND (LAND) (P)90

The Logical And operator logically AND's each bit (except tag bits) of the two top-of-stack operands leaving the result in the top-of-stack. Each bit of the top-of-stack operand is set to one where a one appears in the corresponding bit positions of the two top-of-stack operands; the other information bits in the top-of-stack operand are set to zero. The tag of the second operand is undisturbed except for a double-precision operand in the top-of-stack, in which case the second operand is made double precision and the tag field is changed accordingly. AND is defined as follows:

Operand A	Operand B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

NOTE

The tag field is set equal to the second item in the stack.

LOGICAL OR (LOR) (P)91

The Logical Or operator logically OR's each bit (except tag bits) of the two top-of-stack operands leaving the result in the top-of-stack. OR is defined as follows:

Operand A	Operand B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOTE

The tag field is set equal to the second item in the stack.

LOGICAL NEGATE (LNOT) (P)92

The Logical Negate operator complements each bit position (except tag bits) of the top-of-stack operand.

LOGICAL EQUIVALENCE (LEQV) (P)93

The Logical Equivalence operator compares the corresponding bits of the two items in the top-of-stack (except the tag bits). The two items are replaced by a single item with a tag field equal to the tag field of the second item in the stack and by a one in each bit position where the corresponding bits of the two top-of-stack items were equal.

Pack Operators

PACK, DESTRUCTIVE (PACD) (P)D1

The Pack, Destructive operator packs data (as addressed by the source pointer) right-justified into the top-of-stack in 4-bit (digit) format.

The top-of-stack operand defines the length/repeat field (in digits) to be packed. The source pointer is the second item in the stack. The specified number of digits are transferred from the source to the top-of-stack (dropping the zones when required). If the digit length transferred is less than 13 the tag field in the top-of-stack is set to a single-precision operand; otherwise, the tag field is set to a double-precision operand.

If the length is not less than 25 an invalid-operand interrupt is set and the operation is terminated. If the source data has the memory protect bit (bit 48) set to "one," the segmented-array interrupt is set and the operation is terminated.

If the sign of the source data is negative, the true/false flip-flop is set to "one;" otherwise, the flip-flop is reset. Sign conventions are as follows:

Data Bit Format	Sign Location	Neg. Sign Zone Bit Config.
4-bit	most significant digit	1101
6-bit	least significant character	10
8-bit	least significant byte	1101 (EBCDIC)
8-bit	least significant byte	1111 (ASCII)

PACK, UPDATE (PACU) (P)D9

The Pack, Update operator performs a Pack, Destructive operation. At the completion of the operation, the source pointer is updated.

Relational Operators

The relational operators perform algebraic comparisons on the two top-of-stack operands. The operands are removed from the stack and the result of the comparison is a logical operand which is placed in the top-of-stack. The result is a single-precision operand with the least significant bit set to one if the relation is true or a single-precision operand with all information bits set to zero if the relation is false.

GREATER THAN (GRTR) (P)8A

If the second operand in the stack is greater-than the top-of-stack operand, the Greater Than operator replaces the two operands with a single-precision operand which has the least-significant bit set to one. If the second operand in the stack is not greater than the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

GREATER THAN OR EQUAL (GREQ) (P)89

If the second operand in the stack is greater than or equal to the top-of-stack operand, the Greater Than or Equal operator replaces the two operands with a single-precision operand which has the least-significant bit set to one. If the second operand in the stack is not greater than or equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

EQUAL (EQUL) (P)8C

If the second operand in the stack is algebraically equal to the top-of-stack operand, the Equal operator replaces the two operands with a single-precision operand which has the least-significant bit set to "one". If the second operand in the stack is not equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

LESS THAN OR EQUAL (LSEQ) (P)8B

If the second operand in the stack is less than or equal to the top-of-stack operand, the Less Than or Equal operator replaces the two operands with a single-precision operand which has the least significant bit set the "one". If the second operand in the stack is not less than or equal to the top-of-stack operand, the two operands are re-

placed with a single-precision operand which has all information bits set to zero.

LESS THAN (LESS) (P)88

If the second operand in the stack is less than the top-of-stack operand, the Less Than operator replaces the two operands with a single-precision operand which has the least-significant bit set to "one". If the second operand in the stack is not less than the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

NOT EQUAL (NEQL) (P)8D

If the second operand in the stack is not equal to the top-of-stack operand, the Not Equal operator replaces the two operands with a single-precision operand with the least significant bit set to "one". If the second operand in the stack is equal to the top-of-stack operand, the two operands are replaced with a single-precision operand which has all information bits set to zero.

LOGICAL EQUAL (SAME) (P)94

The Logical Equal operator compares all bits (including tag bits) of the two items (operands, control words, descriptors, etc.) in the top-of-stack. If all bits are equal, a single-precision operand (with the least significant bit set to one and all other information bits set to zero) is stored in the top-of-stack; otherwise, a single-precision operand with all information bits set to zero is stored in the top-of-stack.

Scale Operators

Some higher level languages such as COBOL and PL-I require integer arithmetic. The Scale-Left operators provide a means of aligning the decimal points prior to performing arithmetic operations. The Scale-Right operators provide a means of converting binary arithmetic to decimal arithmetic.

SCALE LEFT (SCLF) (P)C0

The Scale Left operator shifts the operand in the top-of-stack for decimal point alignment. The operand in the top-of-stack is first converted to an integer and then multiplied by 10 raised to the power specified by the scale factor. The scale factor is obtained from the second syllable (i.e., the program syllable following the operator syllable).

If scaling of a single-precision operand would result in overflow, the single-precision operand is converted to a double-precision integer. For the Scale operators, a double-precision integer is defined as a double-precision operand with an exponent equal to 13 (octal). If scaling of the operand results in an exponent greater than 13 (double-precision operand), the overflow flip-flop is set to "one".

DYNAMIC SCALE LEFT (DSLF) (P)C1

The Dynamic Scale Left operator performs a Scale Left operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack. The operand in the top-of-stack is converted to an integer before scaling takes place.

SCALE RIGHT SAVE (SCRS) (P)C4

The Scale Right Save operator shifts the top-of-stack operand to the right for conversion from a binary to a decimal numbering system. The operand in the top-of-stack is converted to an integer and divided by 10 raised to the power specified by the scale factor. The scale factor is obtained from the second syllable. If the scale factor is greater than 12, the invalid-operand interrupt is set and the operation is terminated.

The quotient resulting from the division is left in the top-of-stack. The second operand in the stack is the remainder which is converted to decimal (4-bit digits) and left justified.

DYNAMIC SCALE RIGHT SAVE (DSRS) (P)C5

The Dynamic Scale Right Save operator performs a Scale Right Save operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second item in the stack. The top-of-stack operand is converted to an integer before scaling takes place.

SCALE RIGHT TRUNCATE (SCRT) (P)C2

The Scale Right Turncate operator performs a Scale Right Save operation except that the remainder resulting from the division is deleted from the stack.

DYNAMIC SCALE RIGHT TRUNCATE (DSRT) (P)C3

The Dynamic Scale Right Truncate operator performs a Scale Right Truncate operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack.

SCALE RIGHT ROUNDED (SCRR) (P)C8

The Scale Right Rounded operator performs a Scale Right Save operation except that the remainder resulting from the division is deleted from the stack. If the most significant digit of the remainder is greater than or equal to five the quotient from the division is rounded by adding "one" to it.

DYNAMIC SCALE RIGHT ROUNDED (DSRR) (P)C9

The Dynamic Scale Right Rounded operator performs a Scale Right Rounded operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second operand in the stack.

SCALE RIGHT FINAL (SCRF) (P)C6

The Scale Right Final operator performs a Scale Right Save operation except that the quotient is deleted from the stack and the sign of the quotient is copied into the external sign flip-flop. If the quotient was not equal to zero at the conclusion of the operation, the overflow flip-flop is set.

DYNAMIC SCALE RIGHT FINAL (DSRF) (P)C7

The Dynamic Scale Right Final operator performs a Scale Right Final operation except that the scale factor is obtained from the top-of-stack operand and the operand to be scaled is the second item in the stack.

Stack Operators

The stack operators are used to adjust the relative positions of the top items in the stack and to copy or delete the top of stack item.

EXCHANGE (EXCH) (P)B6

The Exchange operator causes the two top-of-stack items to be exchanged.

ROTATE STACK DOWN (RSDN) (V)B7

The Rotate Stack Down operator rotates the three top-of-stack words as follows:

Before Rotation	After Rotation
Word 1	Word 2
Word 2	Word 3
Word 3	Word 1

ROTATE STACK UP (RSUP) (V)B6

The Rotate Stack Up operator rotates the three top-of-stack words as follows:

Before Rotation	After Rotation
Word 1	Word 3
Word 2	Word 1
Word 3	Word 2

DUPLICATE TOP-OF-STACK (DUPL) (P)B7

The Duplicate Top-of-Stack operator duplicates the item in the top-of-stack.

DELETE TOP-OF-STACK (DLET) (P) B5

The Delete Top-of-Stack operator deletes the top-of-stack item.

PUSH DOWN STACK REGISTERS (PUSH) (P)B4

The Push Down Stack Registers operator pushes down the top-of-stack items and stack buffer contents into memory.

STORE DESTRUCTIVE (STOD) (P)B8

The Store Destructive operator stores the second item in the stack into memory. The address into which the item is to be stored is indicated by an indirect reference word or indexed data descriptor in the top-of-stack. If the top-of-stack item is an operand, the two top-of-stack items are exchanged so that the address item is in the top-of-stack and the item to be stored is in the second position. After the item is stored, both the item and its address are deleted from the stack.

If the word addressed by the indirect reference word is another indirect reference word or indexed data descriptor, or the word addressed by the data descriptor is another indexed data descriptor, the store operation will not occur to that location, but will be retried using the address indicated by that word. This chaining of address items will continue until a "target" location is reached; however, once a data descriptor has been encountered, an indirect reference word or PCW is not allowed, and once a stuffed indirect reference word has been encountered, a normal IRW is not allowed. Either of these conditions will cause an invalid-operand interrupt.

If the word addressed by the indirect reference word is a program control word, accidental procedure

entry occurs. The spontaneously generated RCW causes STOD to be re-executed upon return from the procedure.

If a data descriptor used as an address item has the read-only bit (bit 43) ON, or if the addressed word has the memory protect bit (bit 48) ON and is not a data descriptor, IRW, or PCW, the memory-protect interrupt is set and the operation is terminated.

If the presence bit in the data descriptor is zero, the presence-bit interrupt is set. After the data has been made present, the operation is restarted.

STORE NON-DESCTRUCTIVE (STON) (P)B9

The Store Non-Destructive operator performs a Store Destructive operation, except that only the address item is deleted from the stack. The item which was stored is left in the top-of-stack.

OVERWRITE DESTRUCTIVE (OVRD) (P)BA

The Overwrite Destructive operator performs a Store Destructive operation, except that the addressed location will be overwritten regardless of its contents. Chaining of address items, memory protection checks, or accidental procedure entry do not occur.

OVERWRITE NON-DESTRUCTIVE (OVRN) (P)BB

The Overwrite Non-Destructive operator performs a Store Non-Destructive operation, except that the addressed location will be overwritten regardless of its contents. Chaining of address items, memory protection checks, or accidental procedure entry do not occur.

READ WITH LOCK (RDLK) (V)BA

The Read With Lock operator is a variant of the Overwrite Non-Destructive operator. The word in the top-of-stack and the specified word in memory are interchanged after all local data is purged.

String Operators

The string operators are used for transferring, comparing, scanning, and translating strings of data. In addition, a set of micro-operators provide a means of formatting data for input/output.

The string operators use source and destination pointers which are located in the stack. These pointers are used to set hardware registers as follows:

Register Name	Register Usage
SBR	-Source Base Register
SIR	-Source Word Index Register
SIB	-Source Bit Index
SSZ	-Source Size
DBR	-Destination Base Register
DIR	-Destination Word Index Register
DIB	-Destination Bit Index
DSZ	-Destination Size

Pointer Type	Register Settings	
Non-Indexed Data Descriptor	SBR/DBR	19:20
	SIR/DIR	0
	SIB/DIB	most significant bit
	SSZ/DSZ	0
Indexed Data Descriptor	SBR/DBR	19:20
	SIR/DIR	39:20
	SIB/DIB	0
	SSZ/DSZ	0
Non-Indexed String Descriptor	SBR/DBR	19:20
	SIR/DIR	0
	SIB/DIB	0
	SSZ/DSZ	42:3
Operand	SBR/	Operand Address
	SIR/	0
	SIB/	0
	SSZ/	0
Indexed String Descriptor	SBR/DBR	19:20
	SIR/DIR	34:15
	SIB/DIB	39:4
	SSZ/DSZ	42:3

NOTE

1. A:B A is the starting bit and B the number of succeeding bits.

Some operators have an update variant which denotes that the source and destination pointer should be updated. At the completion of an operation, instead of being deleted from the stack, the pointers are updated as follows:

1. If the source is an operand, it is left in the stack (refer to Note 2).

2. If the pointer type is an indexed string descriptor, then the word index and byte index fields are updated.

3. If the pointer type is a data descriptor, an indexed data descriptor, or a non-indexed string descriptor, it is converted to an indexed string descriptor and then updated.

Update of descriptors consists of setting the character index field (39:4) and the word index field (35:16) of the descriptors in the stack from the SIB/DIB registers and from the SIR/DIR registers, respectively.

NOTE

1. If the count for a string operator is ≤ 0, then the operation terminates after checking the validity of any pointer.

2. The source can be a single-precision or double-precision operand. As the source pointer is advanced the operand is rotated left.

3. If both the source and destination character size are zero, an 8-bit character size is used.

4. When both a source and destination are required, if one character size is zero and the other is not, the character size which is not zero is used. If neither character size is zero and the destination size is zero and the character sizes are not equal, the invalid-operator interrupt is caused.

Exceptions to the above are Translate and Transfer Words, where the source and destination characters can have different sizes.

STRING ISOLATE (SISO) (P)D5

The String Isolate operator transfers from the source string to the top-of-stack the number of bytes specified by the repeat field. This string is right-justified and filled with leading zeros.

At the start of the operation, the top-of-stack operand specifies the length of the byte string and the second item in the stack is an operand or a descriptor used as the source pointer. If the number of bytes exceeds one word (6 bytes or 48 bits), the tag of the result is set to double precision. If the number of bits is greater than 96, an invalid operand interrupt is set and the operation is terminated. If the source data has the memory-protect bit (bit 48) set to "one", the segmented-array interrupt is set and the operation is terminated.

String Transfer Operators

String transfer operators give the system the ability to transfer characters or words from one location in memory to another location in memory. The source and destination pointers are set from string descriptors in the stack.

TRANSFER WORDS, DESTRUCTIVE (TWSD) (P)D3

The Transfer Words, Destructive operator transfers the number of words specified by the top-of-stack oper-

and from the source string to the destination string. The first operand is integerized and is used as the count or repeat field. The second item in the stack (a string descriptor or operand) is the source pointer; i.e., it points at the source string. The third item in the stack (a string descriptor) is the destination pointer which is used to provide the address of the destination string. The number of words specified by the repeat field are transferred from the source to the destination. If the memory protect bit is ON during execution of the Transfer Words operator, then the segmented-array interrupt is set and the operation is terminated.

TRANSFER WORDS, UPDATE (TWSU) (P)DB

The Transfer Words, Update operator performs a Transfer Words, Destructive operation. At the completion of the operation the source and destination pointers are updated to point to the memory location where the transfer ended. If either pointer was a data descriptor, then an indexed data descriptor is updated.

TRANSFER WORDS, OVERWRITE DESTRUCTIVE (TWOD) (P)D4

The Transfer Words, Overwrite Destructive operator performs a Transfer Words, Destructive operation bypassing the memory-protection checks.

TRANSFER WORDS, OVERWRITE UPDATE (TWOU) (P)DC

The Transfer Words, Overwrite Update operator performs a Transfer Words, Update operation bypassing the memory-protection checks.

TRANSFER WHILE GREATER, DESTRUCTIVE (TGTD) (P)E2

The Transfer While Greater, Destructive operator transfers the number of characters specified by the second operand (bits 19:20) in the stack or while the source character is greater than a delimiter. The top-of-stack operand is the delimiter. The third item in the stack is the source pointer, and the fourth item is the destination pointer.

If the second item in the stack is a descriptor, it is used as a source pointer. This means that no repeat field was given and the default field length is 1,048,575.

If either the source or destination word has the memory protect bit ON (bit 48 = 1), the segmented-array interrupt is set and the operation is terminated.

All comparisons are binary (EBCDIC collating sequence). When the source pointer is an operand, it must be a single-precision operand or an invalid-operand interrupt is set and the operation is terminated.

The source character is compared with the delimiter. If the comparison is true, the true/false flip-flop is set to "one"; if the comparison fails, the true/false flip-flop is set to zero.

TRANSFER WHILE GREATER, UPDATE (TGTU) (P)EA

The Transfer While Greater, Update operator performs a Transfer While Greater, Destructive operation. At the completion of the operation, the source and destination pointers are updated to point at the next character in the source and destination strings, respectively. At the completion of the operation, a count of the number of characters not transferred is placed on the top-of-stack. If all the characters specified by the length field are transferred, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false.

If the operation is terminated because the relationship is not met, the source pointer points to the character which stopped the transfer.

TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE (TGED) (P)E1

The Transfer While Greater or Equal operator performs a Transfer While Greater, Destructive operation while the source character is greater than or equal to the delimiter.

TRANSFER WHILE GREATER OR EQUAL, UPDATE (TGEU) (P)E9

The Transfer While Greater or Equal, Update operator performs a Transfer While Greater Than or Equal operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE EQUAL, DESTRUCTIVE (TEQD) (P)E4

The Transfer While Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is equal to the delimiter.

TRANSFER WHILE EQUAL, UPDATE (TEQU) (P)EC

The Transfer While Equal, Update operator performs a Transfer While Equal, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE (TLED) (P)E3

The Transfer While Less or Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is less than or equal to the delimiter.

TRANSFER WHILE LESS OR EQUAL, UPDATE (TLEU) (P)EB

The Transfer While Less or Equal, Update operator performs a Transfer While Less or Equal, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE LESS, DESTRUCTIVE (TLSD) (P)E0

The Transfer While Less, Destructive operator performs a Transfer While Less or Equal, Destructive operation while the source character is less than the delimiter.

TRANSFER WHILE LESS, UPDATE (TLSU) (P)E8

The Transfer While Less, Update operator performs a Transfer While Less, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE NOT EQUAL, DESTRUCTIVE (TNED) (P)E5

The Transfer While Not Equal, Destructive operator performs a Transfer While Greater or Equal, Destructive operation while the source character is not equal to the delimiter.

TRANSFER WHILE NOT EQUAL, UPDATE (TNEU) (P)ED

The Transfer While Not Equal, Update operator performs a Transfer While Not Equal Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER WHILE TRUE, DESTRUCTIVE (TWTD) (V)D3

The Transfer While True, Destructive operator transfers characters from the source string to the destination string for the number of characters specified by the length operand while the stated relationship is met. If the relationship is not met the transfer is terminated at that point. The relationship is determined by using the source character to index a bit in the table. If the bit indexed is a "one" the relationship is true. An all zero's character indexes to the most significant bit of the table.

The operator uses the top four words in the stack as follows: The top-of-stack word is a table pointer to specific addresses in the table; the second word in the stack provides the length of the string to be transferred or, if it is a descriptor, it is used as a source pointer since no repeat field was given and the default field length is set at 1,048,575; the third word in the stack is an operand or a descriptor which gives the address of the source string or is a single-precision operand which is the source string; the fourth word in the stack is a descriptor pointing at the destination string.

The table is indexed as follows to obtain the decision bit: The source character is expanded to eight bits, if necessary, by appending two or four leading-zero bits. The three high-order bits of the source character select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select (by their value) a bit from this word.

At the completion of the operation, a count of the number of characters not transferred is placed on the top of stack.

If all the characters specified by the length field are transferred, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false.

The table format is as follows:

Source Size	Table Length	Bits/Word
4	1 word	(31:16)
6	2 words	(31:32)
8	8 words	(31:32)

TRANSFER WHILE TRUE, UPDATE (TWTU) (V)DB

The Transfer While True, Update operator performs a Transfer While True, Destructive operation. At the completion of the operation, the source destination pointers and the count are updated. If all the characters specified by the length field are transferred, the true/false flip-flop is set to "one" (true); otherwise it is set to zero.

TRANSFER WHILE FALSE, DESTRUCTIVE (TWFD) (V)D2

The Transfer While False, Destructive operator performs the Transfer While True operation except that the relationship is true if the bit found by indexing into the table is a zero.

TRANSFER WHILE FALSE, UPDATE (TWFU) (V)DA

The Transfer While False, Update operator performs a Transfer While False, Destructive operation. At the completion of the operation, the source and destination pointers and the count are updated.

TRANSFER UNCONDITIONAL, DESTRUCTIVE (TUND) (P)E6

The Transfer Unconditional, Destructive operator transfers from the source to the destination the number of characters specified by the top-of-stack operand. If the top-of-stack item is a descriptor, it is used as a source pointer. Since no repeat field was given, the field length is set by default at 1,048,575. The second item in the stack is the destination pointer. If all characters specified by the length field are transferred, the true/false flip-flop is set to "one" (true) by this operand; otherwise, the flip-flop is set to zero (false).

TRANSFER UNCONDITIONAL, UPDATE (TUNU) (P)EE

The Transfer Unconditional, Update operator performs a Transfer Unconditional, Destructive operation. At the completion of the operation, the source and destination pointers are updated.

Subroutine Operators

Subroutine operators are those operators which can move the program operation across machine architecture such as from stack to stack, or from subroutine to subroutine, etc.

Any subroutine operator which can "chain" indirect reference words (IRW's) or stuffed indirect reference words (SIRW's) can obtain accidental procedure entry if a program control word (PCW) is pointed to by the IRW or SIRW last in the chain.

MARK STACK (MKST) (P)AE

The Mark Stack operator inserts a mark into the stack which is to be subsequently used by an Enter operator. The mark is placed in the top-

of-stack in the form of a mark stack control word. The F register is set to the location of the MSCW.

The Mark Stack operator is normally used when an entry to a procedure is anticipated. The normal sequence of events to enter a procedure is (1) mark the stack; (2) insert an indirect reference to a program control word; (3) insert parameters, if any are to be passed to the procedure; and then (4) execute an Enter operator, which will in turn, cause an entry into the program segment located by the program control word.

INSERT MARK STACK (IMKS) (P)CF

The Insert Mark Stack operator inserts a mark stack control word in the current stack below the two top-of-stack items.

NAME CALL (NAMC) (P)40 THRU (P)7F

Name Call builds an indirect reference word in the top-of-stack. The six low-order bits of the first syllable and the eight bits of the second syllable form a 14-bit address couple. This address couple is placed in the top-of-stack with the tag field set to 001.

In the B 7700, if the Name Call is followed by a ENTR, INDX, NXLN, NXLV, STOD, STON, OVRD, or OVRN operator, the IRW is not placed in the stack. Instead, the referenced memory address is calculated and, if appropriate, the memory access is initiated by the program control unit. The following operator is sent to the execution unit along with an indication that the operator has been started. Since the address computation and, in some cases, the memory fetch, is complete by the time the operator reaches the execution unit, a considerable time savings is realized.

VALUE CALL (VALC) (P)00 THRU (P)3F

Value Call is a two-syllable instruction that brings an operand from memory into the top-of-stack. A concatenation of the two Value Call syllable gives a 14-bit address couple. If the referenced memory location contains an indirect reference word or a data descriptor, additional memory accesses are made until the "target" operand is located. The operand is then placed in the top-of-stack register. The operand may be either signle-precision or double-precision, causing either one or two words to be loaded into the top-of-stack. (Figures III-3-5 and III-3-6 are simplified flow charts of the Value Call operator.)

If the word accessed is an indexed data descriptor, the word addressed by the data descriptor is brought to the top-of-stack. If the word accessed is a non-indexed word data descriptor, the descriptor is indexed using the second word in the stack as the index value, and the word addressed by the non-indexed data descriptor is brought to the top-of-stack. If the double-precision bit (bit 40) in the data descriptor is set, the second half of the double-precision operand is placed in the second half of the top-of-stack location.

If the presence bit in the data descriptor is zero, the presence-bit interrupt is set. After the data has been made present, the operation is restarted.

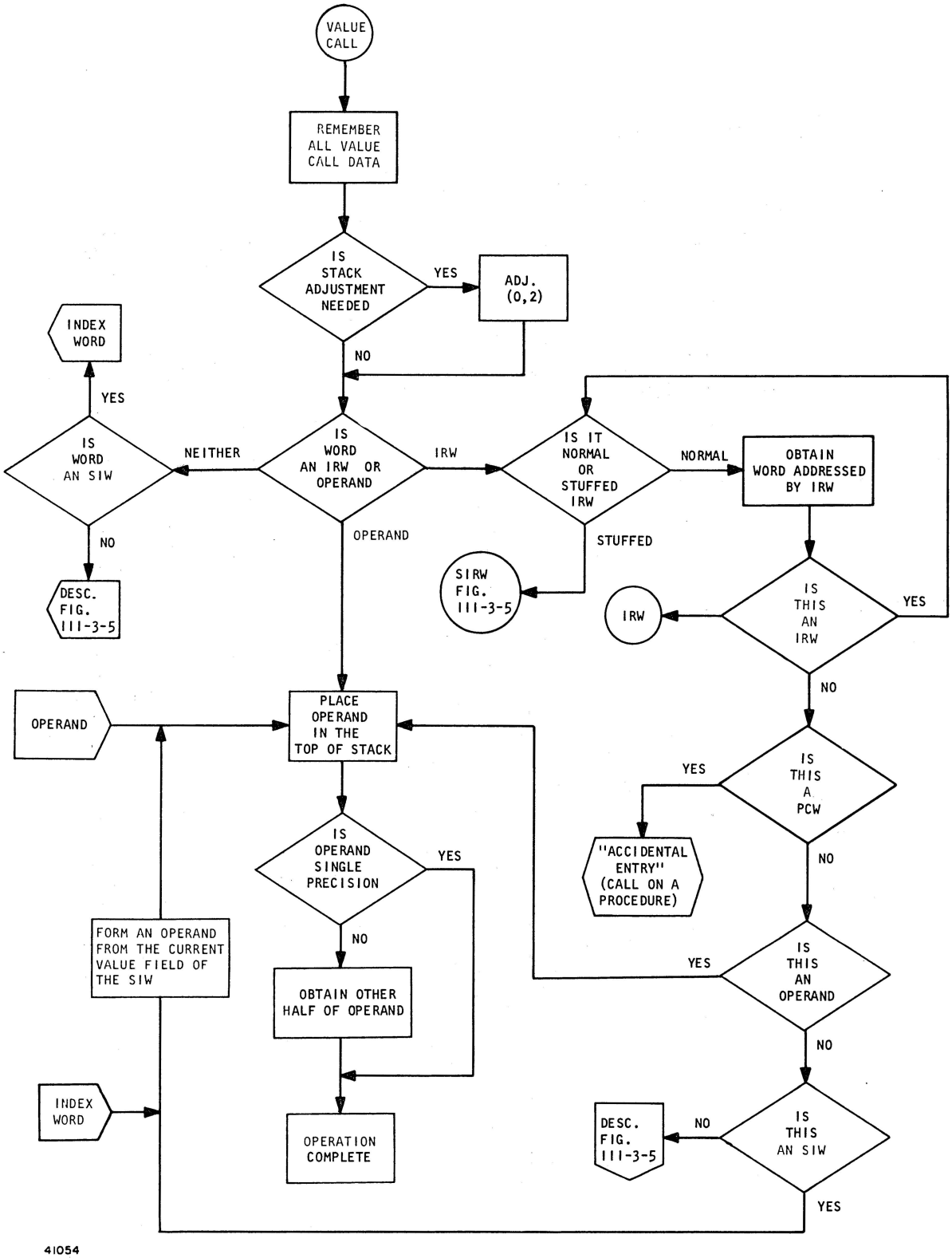


Figure III-3-5. Flow Chart of Value Call Operator

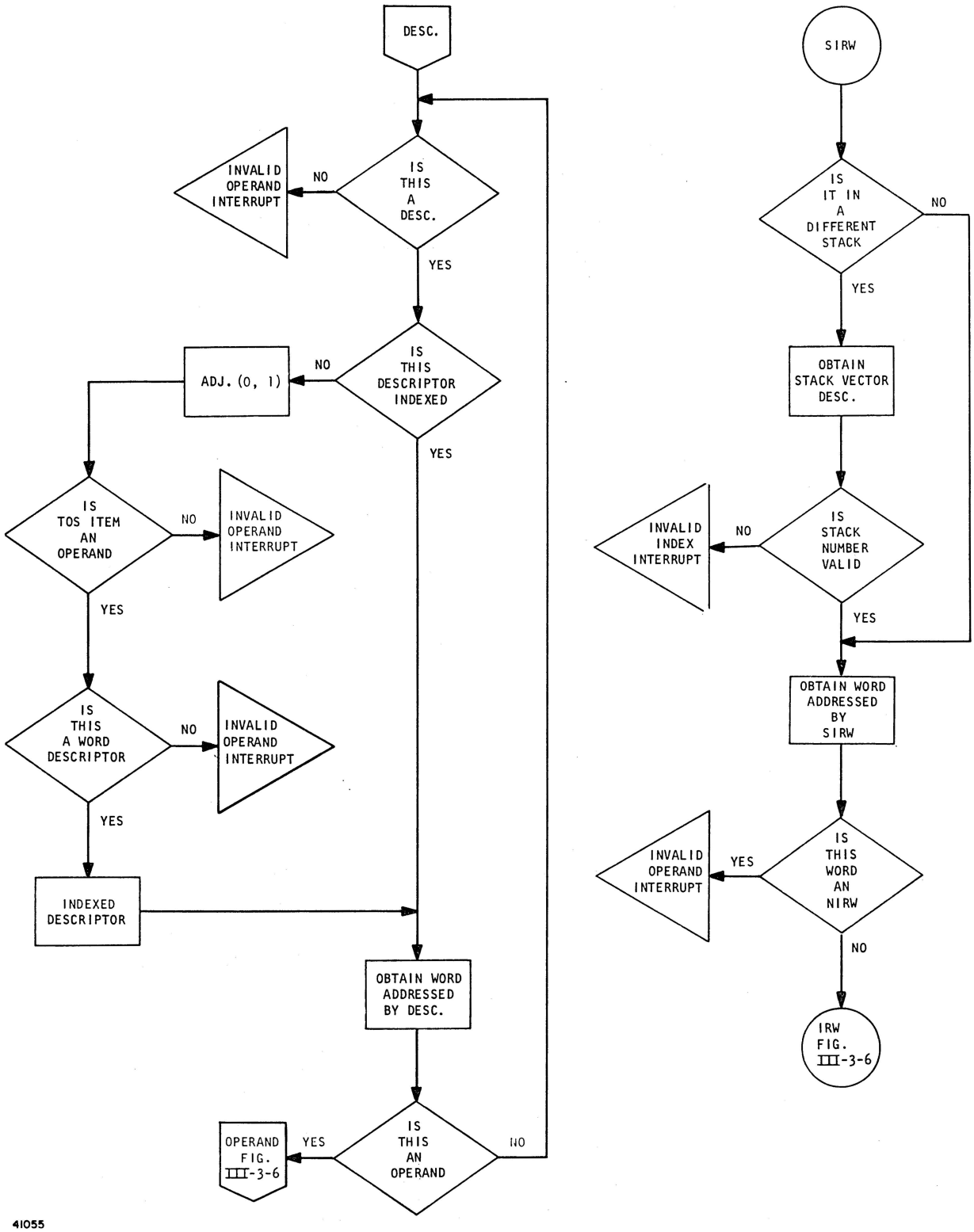


Figure III-3-6. Flow Chart of Value Call Operator

If a data descriptor does not address an operand, step index word, or a word descriptor or indexed string descriptor, an invalid-operand interrupt is set and the operation is terminated.

If the word accessed by the Value Call is an indirect reference word (IRW), the word addressed by the IRW is brought to the top-of-stack.

If the word accessed is a program control word (PCW), an accidental entry into the subroutine addressed by the PCW is initiated. A mark stack control word and return control word are placed in the stack and an entry is made into the subprogram. Upon completion of the subprogram, a return operation will re-enter the Value Call operator flow.

If the target operand is a step index word (tag = 4) instead of an operand, the current-value field (bits 15:16) of the SIW will be placed in the top-of-stack with the tag set to zero.

The "chaining" of memory accesses continues until a target operand is reached; however, once a data descriptor has been encountered, an indirect reference word or PCW is not allowed, and once a stuffed indirect reference word has been encountered, a normal IRW is not allowed. Either of these conditions will cause an invalid-operand interrupt.

EVALUATE DESCRIPTOR (EVAL) (P)AC

The Evaluate Descriptor operator loads into the top of stack a data descriptor or an indirect reference word which points to the referenced operand. This operand may be referenced through a chain of indirect reference words or descriptors. (Figure III-3-7 is a simplified flow chart of the Evaluate Descriptor operator.)

A descriptor is left in the stack if the operand was referenced by a descriptor. If only indirect reference words are used, multiple memory accesses are made until the operand is located. A stuffed indirect reference word pointing to the operand is left in the stack.

An invalid-operator interrupt is set and the operation is terminated if the top-of-stack word is not a descriptor or an indirect reference word at the start of the Evaluate operator.

ENTER (ENTR) (P)AB

The Enter operator causes an entry into a procedure from a calling procedure. (The sequence of events to enter a procedure is: (1) mark the stack; (2) insert an indirect reference to a program control word; (3) insert parameter(s), if any are to be passed to the procedure; and, (4) execute an Enter operator.) The Enter operator causes entry into the program segment located by the program control word. A return control word is stored at stack location F+1. (Figure III-3-8 is a simplified flow chart of the Enter operator.)

EXIT (EXIT (P)A3

The EXIT operator causes a called procedure to return to a calling procedure and is used when the called procedure is not required to return a result. The Exit operator returns all control registers to the position they were in prior to the

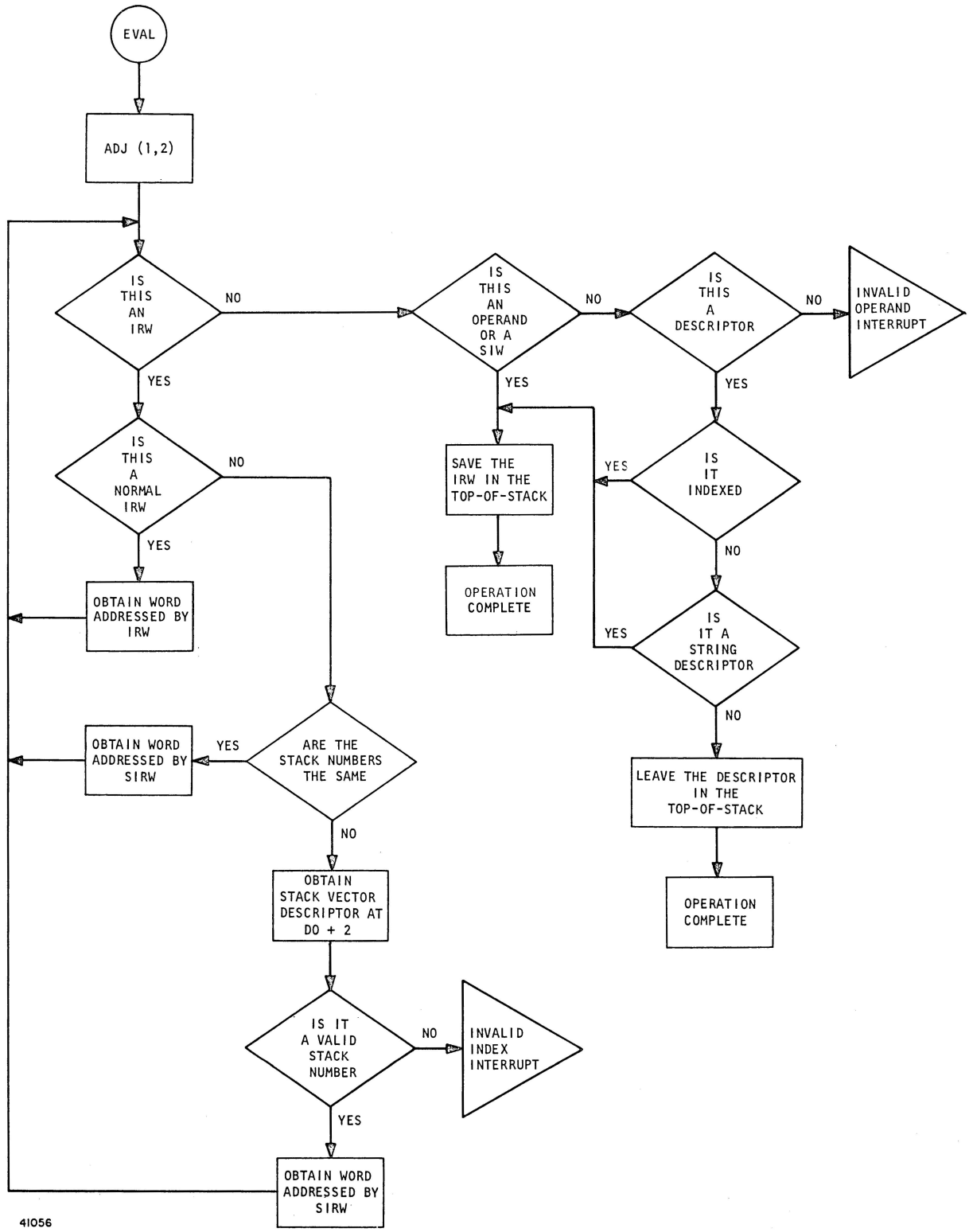


Figure III-3-7. Flow Chart of Evaluate Descriptor Operator

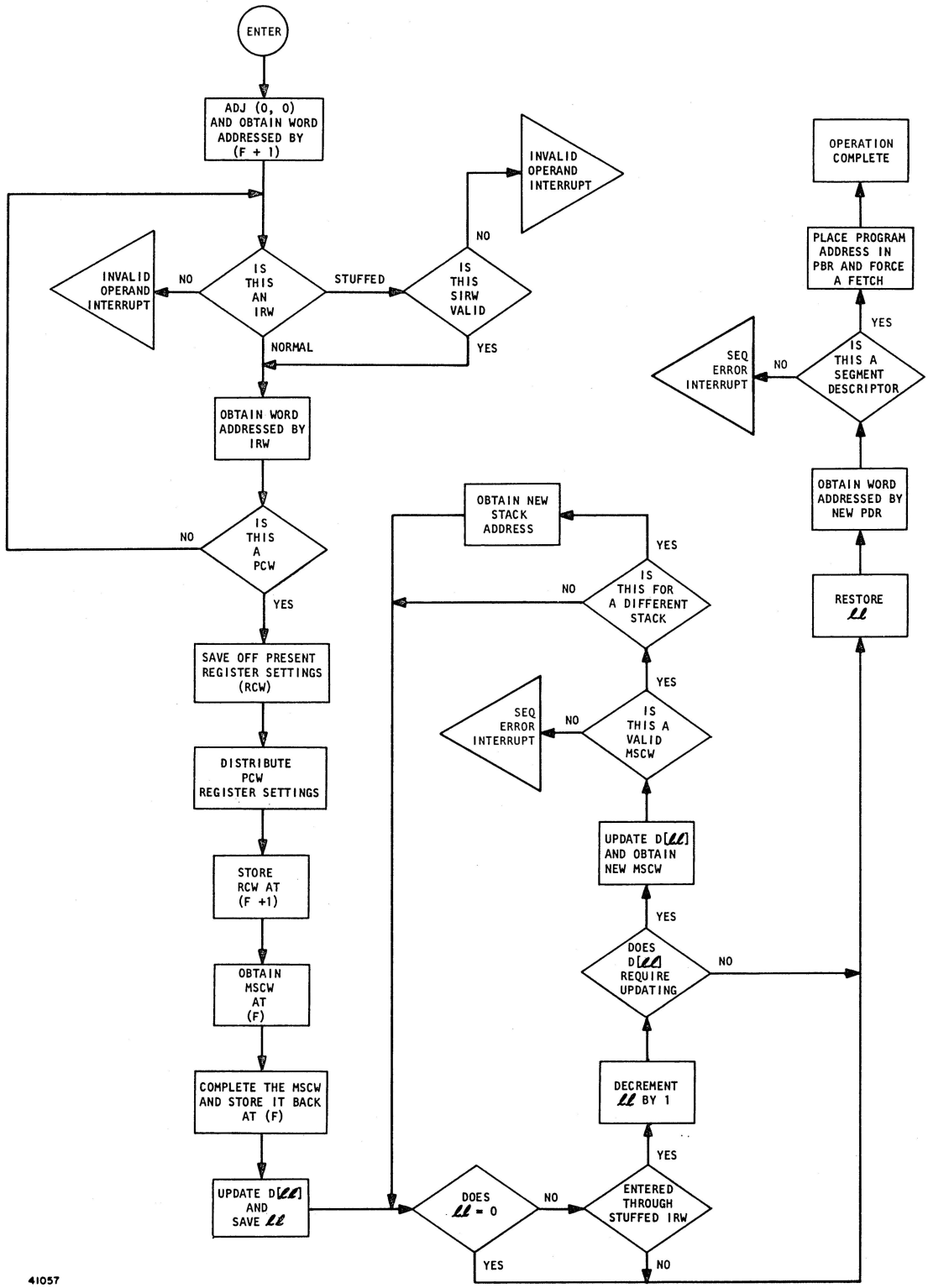


Figure III-3-8. Flow Chart of Enter Operator

calling procedure, saves the bottom of stack register (BOSR), and cuts back the stack. (Figure III-3-9 is a simplified flow chart of the Exit operator.)

RETURN (RETN) (P)A7

The Return operator causes a called procedure to retrun to a calling procedure (as in EXIT) but is used when the called procedure is required to return a result. An operand or name in the top-of-stack is returned to the calling procedure. If a name is returned and the V bit (bit 19) in the MSCW is ON, the name is evaluated to yield an operand as in VALC (since the V-bit indicates that the RETN is to VALC which caused accidental entry). (Figure III-3-10 is simplified flow chart of the Return operator.)

Transfer Operators

The transfer operators transfer any field of bits from one word in the stack to any field of another word in the stack.

NOTE

For all transfer operators the values specified in the stack must be non-negative.

FIELD TRANSFER (FLTR) (P)98

The Field Transfer operator uses the three syllables following it to establish the pointers used in the field transfer. Stack adjustment takes place so that the two top-of-stack locations are full. The contents of the field in the top-of-stack, starting at the bit position addressed by the third syllable of FLTR, is transferred into a field of corresponding length in the second location in the stack. The field in the second location in the stack starts at the bit position indicated by the second syllable of FLTR and proceeds toward the low-order-bit positions. When the number of bits specified by the fourth syllable of FLTR has been transferred the top-of-stack word and the operation is complete.

If the second or third syllables of the operator are found to be greater than 47 or the fourth syllable is greater than 48, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD TRANSFER (DFTR) (P)99

The Dynamic Field Transfer operator causes a Field Transfer operation using the top-of-stack operand to specify the field length, using the second operand in the stack to specify the starting-bit position of the field from which the transfer will be made, and using the third operand in the stack to indicate the starting bit of the field to which the transfer will be made.

As each of these operands is used to establish a pointer for the transfer, it is first integerized and checked for being greater than 47 or 48, as above, then is deleted from the stack. The fourth and fifth stack operands become the two top-of-stack operands, and the transfer takes place as in the FLTR operator.

FIELD ISOLATE (ISOL) (P)9A

The Field Isolate operator isolates a field in the top-of-stack word. The second syllable of the operator specifies the starting bit. The third syllable specifies the length

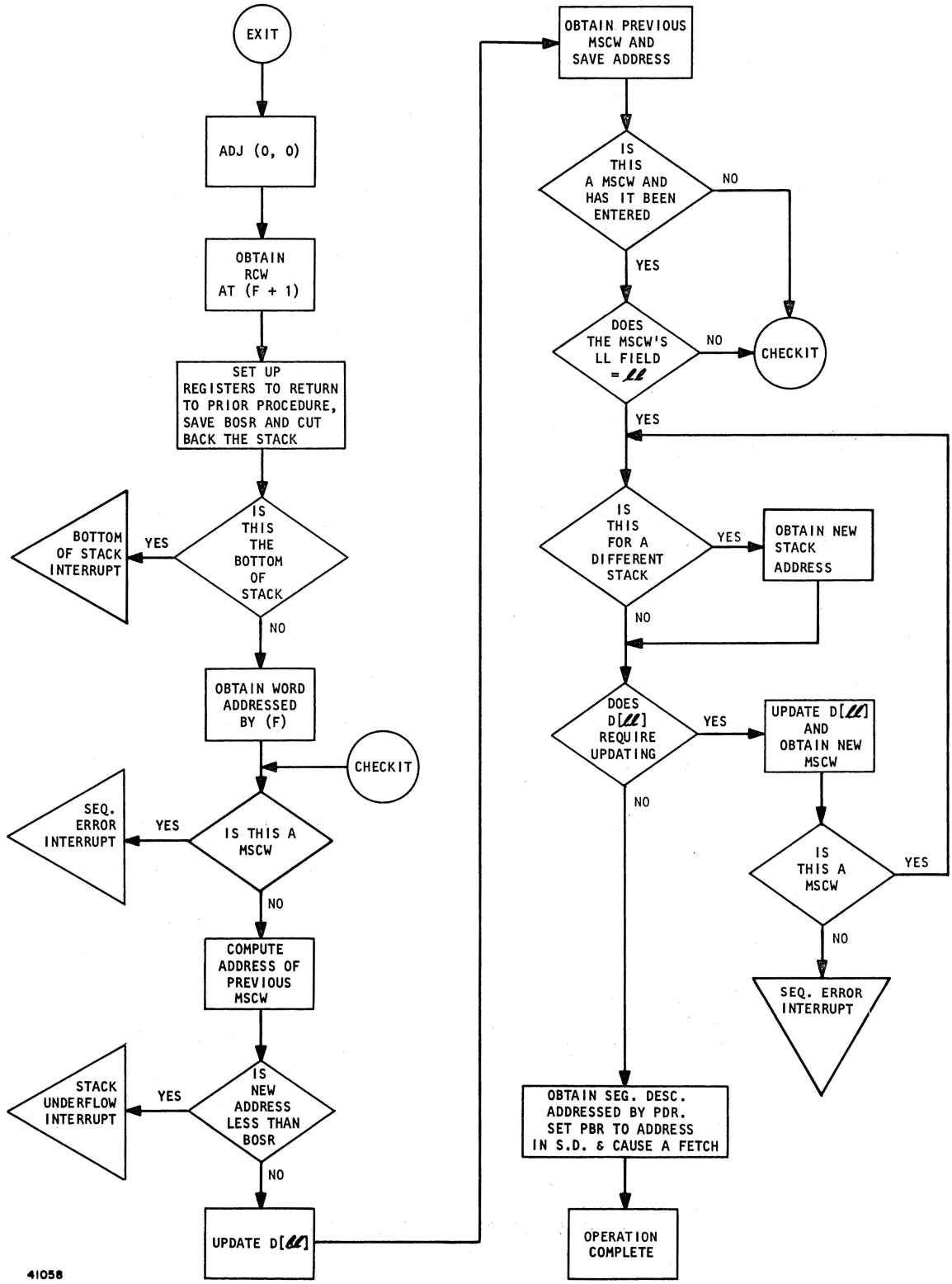


Figure III-3-9. Flow Chart of Exit Operator

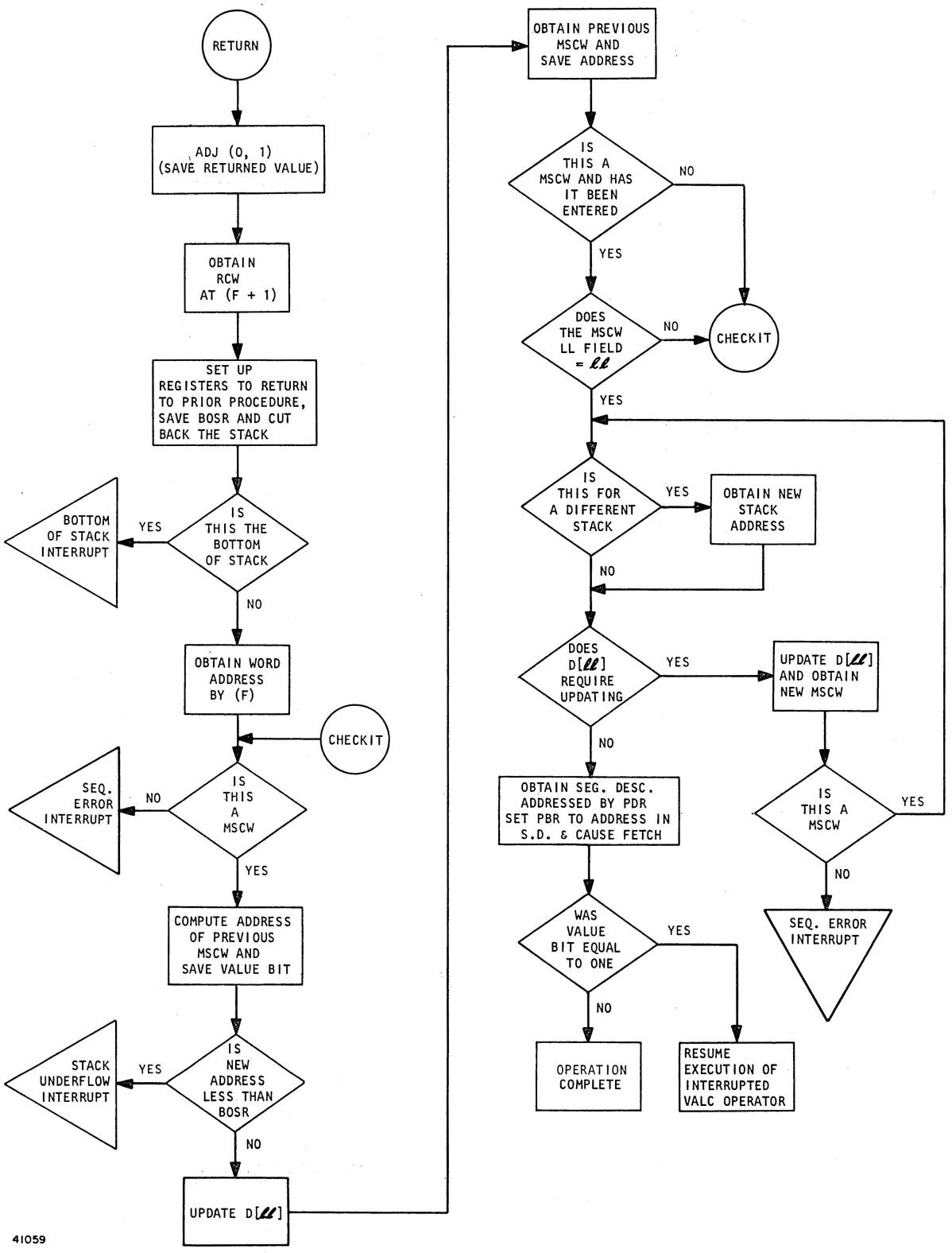


Figure III-3-10. Flow Chart of Return Operator

of the field in bits. The isolated field is right-justified with all other information bits set to zero. The tag bits are not changed.

DYNAMIC FIELD ISOLATE (DISO) (P)9B

The Dynamic Field Isolate operator performs a Field Isolate operation using the top-of-stack operand to specify the length of the field to be isolated and using the second operand in the stack to specify the starting bit. These operands are then deleted from the stack and the Field Isolate operation is performed on the next operand.

FIELD INSERT (INSR) (P)9C

The Field Insert operator inserts a field from the top-of-stack into the second word. Stack adjustment assures that the top two positions are occupied. The right-justified field in the top-of-stack is inserted into the second word starting at the position specified by the second syllable of the Field Insert operator. The third syllable specifies the length of the field to be inserted. The top-of-stack word is deleted after the field is inserted in the second word.

DYNAMIC FIELD INSERT (DINS) (P)9D

The Dynamic Field Insert operator performs a Field Insert operation, transferring a field from the top operand in the stack into the fourth operand in the stack. The second operand in the stack specifies the length of the field to be inserted, and the third operand in the stack specifies the starting bit of the field.

Type-Transfer Operators

Type-Transfer operators are used to manipulate operand relative to single-precision or double-precision operands.

SET TO SINGLE-PRECISION, TRUNCATED (SNGT) (P)CC

The Set to Single-Precision, Truncated operator sets the top-of-stack operand to a single-precision operand without rounding.

SET TO SINGLE-PRECISION ROUNDED (SNGL) (P)CD

The Set to Single-Precision, Rounded operator sets the top-of-stack operand to a single-precision operand with rounding.

SET TO DOUBLE-PRECISION (XTND) (P)CE

The Set Double-Precision operator sets the top-of-stack operand to a double-precision operand.

SET DOUBLE TO TWO SINGLES (SPLT) (V)43

The Set Double to Two Singles operator splits a double-precision operand into two single-precision operands.

SET TWO SINGLES TO A DOUBLE (JOIN) (V)42

The Set Two Singles to a Double operator joins two single-precision operands to form one double-precision operand.

Miscellaneous Primary Mode Operators

Miscellaneous primary mode operators are those operators which cannot be readily described or grouped with other operators.

ESCAPE TO 16-BIT INSTRUCTION (VARI) (P)95

The Escape to 16-Bit Instruction operator provides transition from the primary mode operators to the variant mode operators, i.e., the first syllable (VARI) indicates that the actual operator is in the second syllable. (Interrupts are not allowed between the VARI syllable and the following syllable.)

READ AND CLEAR OVERFLOW FLIP-FLOP (ROFF) (P)D7

The Read and Clear Overflow Flip-Flop operator places a single-precision operand in the top-of-stack with the least-significant bit set equal to the overflow flip-flop. The overflow flip-flop is reset.

READ TRUE FALSE FLIP-FLOP (RTFF) (P)DE

The Read True False Flip-Flop operator places a single-precision operand in the top-of-stack with the least significant bit set equal to the true/false flip-flop.

SET EXTERNAL SIGN (SXSN) (P)D6

The Set External Sign operator places the operand sign bit of the top-of-stack word into the external sign flip-flop.

STUFF ENVIRONMENT (STFF) (P)AF

The Stuff Environment operator places the current stack number and displacement into the stack number field and displacement field of the top-of-stack IRW. Bit 46 is set to indicate that it is now a stuffed indirect reference word. (Figure III-3-11 is a simplified flow chart of the Stuff Environment operator.)

Universal Operators

The operators NOOP, HALT, and NVLD are universal except that they cannot follow operators EXSU, EXSD, EXPU, and EXPD; in these cases a Loop Timeout will occur.

CONDITIONAL HALT (HALT) (U)DF

The Conditional Halt operator halts the processor if the confitional halt switch is in the ON position; if the conditional halt switch is OFF, the operator is treated as a NOOP.

INVALID OPERATOR (NVLD) (U)FF

The Invalid Operator sets the invalid-operator interrupt.

NO OPERATION (NOOP) (U)FE

No operation occurs when the No Operation operator is encountered except that the PSR and PIR are advanced to point at the next operator.

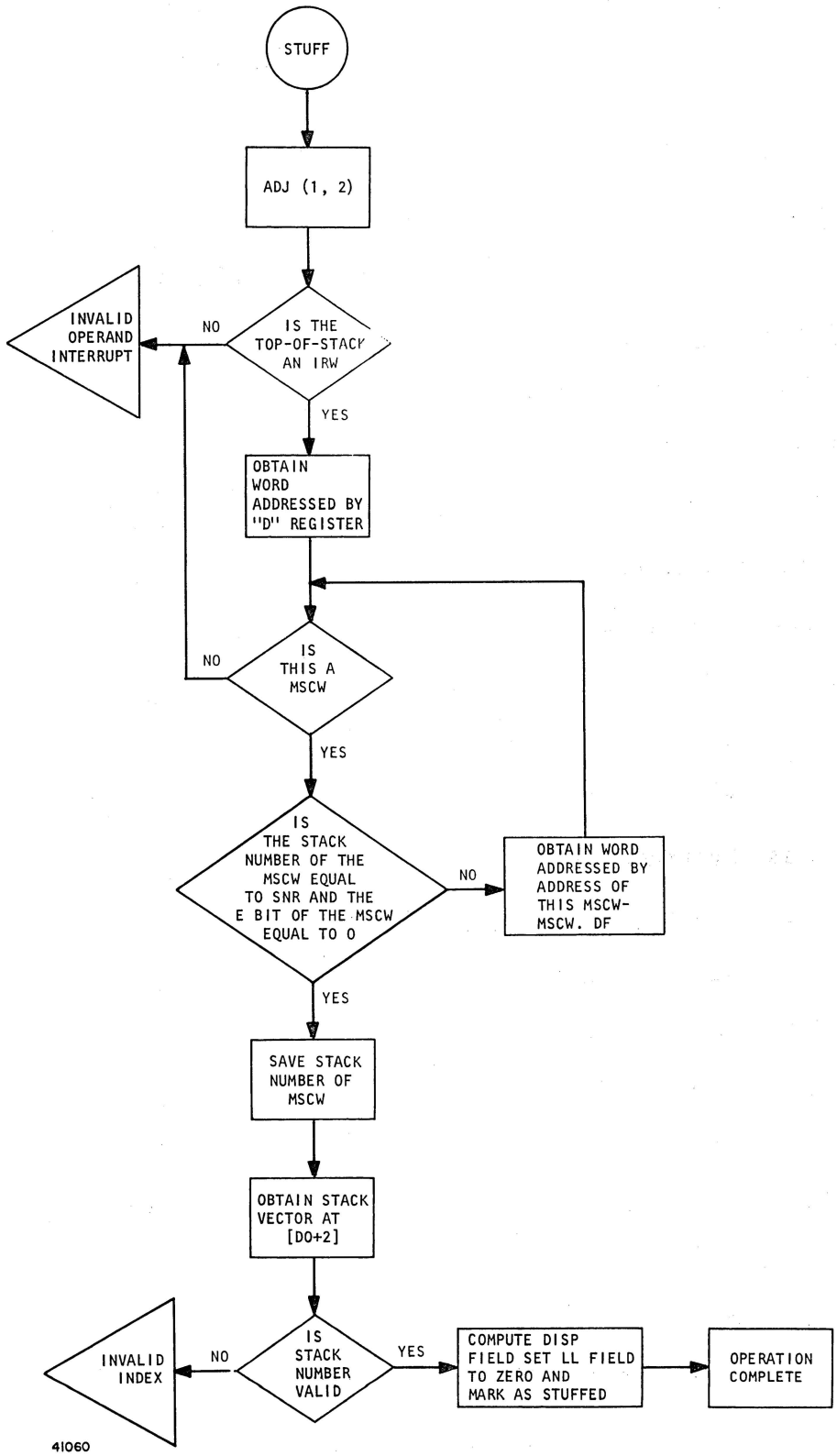


Figure III-3-11. Flow Chart of Stuff Environment Operator

ORDER OF VARIANT MODE OPERATOR DESCRIPTIONS

Name	Mnemonic	Mode ID and Hex Code	Page
Scan Operators			
Scan In	SCNI	(V)4A	3-148
Scan While Operators			
Scan While Greater, Destructive	SGTD	(V)F2	3-148
Scan While Greater, Update	SGTU	(V)FA	3-148
Scan While Greater or Equal, Destructive	SGED	(V)F1	3-149
Scan While Greater, or Equal, Update	SGEU	(V)F9	3-149
Scan While Equal, Destructive	SEQD	(V)F4	3-149
Scan While Equal, Update	SEQU	(V)FC	3-149
Scan While Less or Equal Destructive	SLED	(V)F3	3-149
Scan While Less or Equal, Update	SLEU	(V)FB	3-149
Scan While Less, Destructive	SLSD	(V)F0	3-149
Scan While Less, Update	SLSU	(V)F8	3-149
Scan While No Equal, Destructive	SNED	(V)F5	3-149
Scan While Not Equal, Update	SNEU	(V)FD	3-149
Scan While True, Destructive	SWTD	(V)D5	3-150
Scan While True, Update	SWTU	(V)DD	3-150
Scan While False, Destructive	SWFD	(V)D4	3-150
Scan While False, Update	SWFU	(V)DC	3-150
Tag Field Operators			
Set Tag Field	STAG	(V)B4	3-150
Read Tag Field	RTAG	(V)B5	3-150

ORDER OF VARIANT MODE OPERATOR DESCRIPTIONS (Cont'd)

Name	Mnemonic	Mode ID and Hex Code	Page
Set Interval Timer	SINT	(V)45	3-151
Read Processor Identification	WHOI	(V)4E	3-151
Enable External Interrupts	EEXI	(V)46	3-151
Disable External Interrupts	DEXI	(V)47	3-151
Idle Until Interrupt	IDLE	(V)44	3-151
Read Processor Register	RPRR	(V)B8	3-151
Set Processor Register	SPRR	(V)B9	3-152
Unpack Operators			
Unpack Absolute, Destructive	UABD	(V)D1	3-154
Unpack Absolute, Update	UABU	(V)D9	3-154
Unpack Signed, Destructive	USND	(V)D0	3-154
Unpack Signed, Update	USNU	(V)D8	3-155
Miscellaneous Variant Mode Operators			
Linked List Lookup	LLLU	(V)BD	3-155
Masked Search for Equal	SRCH	(V)BE	3-155
Move to Stack	MVST	(V)AF	3-156
Occurs Index	OCRX	(V)85	3-157
Translate	TRNS	(V)D7	3-157
Variant Mode Operators Exclusive to the B 7700			
Set Memory Inhibits	SINH	(V)A8	3-158
Set Memory Limits	SLMT	(V)AA	3-158
Fetch Memory Fail Register	FMFR	(V)AC	3-158

ORDER OF VARIANT MODE OPERATOR DESCRIPTIONS (Cont'd)

Name	Mnemonic	Mode ID and Hex Code	Page
Ignore Parity	IGPR	(V)48	3-158
Pause Until Interrupt	PAUS	(V)84	3-158
Interrupt Channel N	INCN	(V)8F	3-159
Stop	STOP	(V)BF	3-159
Universal Operators			3-159
Conditional Halt	HALT	(U)DF	3-159
Invalid Operator	NVLD	(U)FF	3-159
No Operation	NOOP	(U)FE	3-159
Variant Mode Operators Described Elsewhere			
Integerize Rounded, Double Precision	NTGD	(V)87	3-107
Count Binary One's	CBON	(V)BB	3-107
Leading One Test	LOG2	(V)8B	3-107
Load Transparent	LDT	(V)BC	3-118
Rotate Stack Down	RSDN	(V)B7	3-125
Rotate Stack Up	RSUP	(V)B6	3-125
Read With Lock	RDLK	(V)BA	3-126
Transfer While True, Destructive	TWTD	(V)D3	3-131
Transfer While True, Update	TWTU	(V)DB	3-132
Transfer While False, Destructive	TWFD	(V)D2	3-132
Transfer While False, Update	TWFU	(V)DA	3-132
Set Double to Two Singles	SPLT	(V)43	3-142
Set Two Singles to a Double	JOIN	(V)42	3-142

VARIANT MODE OPERATORS

Variant mode operators is the name used to describe those primary mode operators which are less frequently used. There is no functional significance to the category "variant mode." Variant mode operation extends the number of operation codes. Variant mode operators require two syllables: the first syllable is the Escape to 16 Bit Instruction (VAR1) operator: The syllable following VARI is the actual operation and the syllable pointer is positioned beyond the two syllables.

Variant mode codes VE0 thru VEF are detected and cause a programmed operator interrupt. All other unassigned variant mode codes cause no action and result in a loop timer interrupt.

Variant mode operations are both word-and string-oriented operators.

Scan Operators

The Scan operators are used for communicating between the processors and units on the scan bus. The B 7700 central processor does not use the scan bus - its communication is accomplished via main memory.

SCAN IN (SCNI) (V)4A

The Scan-In operator uses the 20 low-order bits of the top-of-stack word as a scan address and reads the scan response (in binary, bits 0:36) to the top-of-stack. The scan response replaces the scan address. Only the time-of-day variant is legal for B 7700. Any other variant gives an invalid operand interrupt.

Scan While Operators

SCAN WHILE GREATER, DESTRUCTIVE (SGTD) (V)F2

The Scan While Greater, Destructive operator scans the number of characters specified by the second operand in the stack or while the source character is greater than a delimiter. The top-of-stack operand is the delimiter. The third item in the stack is the source pointer. If the second item in the stack is a descriptor, it is used as a source pointer and the length of the character string is set to 1,048,575. All comparisons are binary. When the source is an operand, it must be a single-precision operand.

At the completion of this operator if all the characters have been scanned the true/false flip-flop is set to one. If the scan was stopped by the delimiter test before the end of the string the true/false flip-flop is set to zero.

SCAN WHILE GREATER, UPDATE (SGTU) (V)FA

The Scan While Greater, Update operator performs a Scan While Greater, Destructive operation. At the completion of the operation, the source pointer and count are updated. At the completion of the operation, a count of the number of characters not scanned is placed in the top-of-stack. If all the characters specified by the length field are scanned, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false. The source pointer locates the character which stopped the scan.

SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE (SGED) (V)FL

The Scan While Greater or Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is greater than or equal to the delimiter.

SCAN WHILE GREATER OR EQUAL, UPDATE (SGEU) (V)F9

The Scan While Greater or Equal, Update operator performs a Scan While Greater Than or Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE EQUAL, DESTRUCTIVE (SEQD) (V)F4

The Scan While Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is equal to the delimiter.

SCAN WHILE EQUAL, UPDATE (SEQU) (V)FC

The Scan While Equal, Update operator performs a Scan While Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE LESS OR EQUAL, DESTRUCTIVE (SLED) (V)F3

The Scan While Less or Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is less than or equal to the delimiter.

SCAN WHILE LESS OR EQUAL, UPDATE (SLEU) (V)FB

The Scan While Less or Equal, Update operator performs a Scan While Less or Equal, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE LESS, DESTRUCTIVE (SLSD) (V)F0

The Scan While Less, Destructive operator performs a Scan While Greater, Destructive operation while the source character is less than the delimiter.

SCAN WHILE LESS, UPDATE (SLSU) (V)F8

The Scan While Less, Update operator performs a Scan While Less, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE NOT EQUAL, DESTRUCTIVE (SNED) (V)F5

The Scan While Not Equal, Destructive operator performs a Scan While Greater, Destructive operation while the source character is not equal to the delimiter.

SCAN WHILE NOT EQUAL, UPDATE (SNEU) (V)FD

The Scan While Not Equal, Update operator performs a Scan While Not Equal, Destructive operation. At the completion, the source pointer and count are updated.

SCAN WHILE TRUE, DESTRUCTIVE (SWTD) (V)D5

The Scan While True, Destructive operator uses each source character as an index into a table to locate a bit in the table. In order to index the table the source character is expanded to eight bits (if necessary) by appending two or four leading-zero bits. The three high-order bits of these eight select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select a bit from this word by their value. If the bit located is a one, the relationship is true and the scan continues. An all zero's character indexes to the most significant bit of the table.

The top-of-stack word is a table pointer. The second item in the stack specifies the number of characters to be scanned or, if it is a descriptor, it is used as a source pointer and the length of the character string is set at 1,048,575. The third item in the stack is the source pointer. If all the characters specified by the length field are scanned, the true/false flip-flop is set to true; otherwise, the true/false flip-flop is set to false. At the completion of the operation, a count of the number of characters not scanned is placed on the top of stack. The table format is as follows:

Source Size	Table Length	Bits/Word
4	1 word	(31:16)
6	2 words	(31:32)
8	8 words	(31:32)

SCAN WHILE TRUE, UPDATE (SWTU) (V)DD

The Scan While True, Update operator performs a Scan While True, Destructive operation. At the completion of the operation, the source pointer and count are updated.

SCAN WHILE FALSE, DESTRUCTIVE (SWFD) (V)D4

The Scan While False, Destructive operator performs a Scan While True, Destructive operation except that the relationship is true if the bit found by indexing into the table is a zero.

SCAN WHILE FALSE, UPDATE (SWFU) (V)DC

The Scan While False, Update operator performs a Scan While False, Destructive operation. At the completion of the operation, the source pointer and count are updated.

Tag Field Operators

SET TAG FIELD (STAG) (V)B4

The Set Tag Field operator sets the tag field (bits 50:3) of the second word in the stack to the contents of bits 2:3 of the top-of-stack word.

READ TAG FIELD (RTAG) (V)B5

The Read Tag Field operator replaces the top-of-stack word with a single-precision operand with bits 2:3 equal to the tag field of the original top-of-stack word.

SET INTERVAL TIMER (SINT) (V)45
(CONTROL STATE OPERATOR)

The Set Interval Timer operator integerizes the top-of-stack operand. If the operand cannot be integerized, an integer-overflow interrupt is set and the operation is terminated. The value of the 11 low-order bits of the top-of-stack operand is used to set the interval timer associated with the processor which is executing this operator. Once set, the interval timer will start to decrement once each 512 microseconds. The associated processor is interrupted when the value has been counted to zero if the timer is still armed.

The interval timer is disarmed whenever the associated processor is interrupted by an external interrupt.

READ PROCESSOR IDENTIFICATION (WHOI)
(V)4E

The Read Processor Identification operator places a single-precision operand with a value equal to the processor's number on the top-of-stack.

ENABLE EXTERNAL INTERRUPTS (EEXI)
(V)46

The Enable External Interrupts operator allows this processor to respond to external interrupts.

DISABLE EXTERNAL INTERRUPTS (DEXI)
(V)47

The Disable External Interrupts operator prohibits this processor from responding to external interrupts.

IDLE UNTIL INTERRUPT (IDLE) (V)44

The Idle Until Interrupt operator suspends program execution by this processor. External interrupts are allowed, and the processor will enter its interrupt-handling routine upon receipt of an interrupt.

READ PROCESSOR REGISTER (RPRR)
(V)B8

The Read Processor Register operator reads into the top-of-stack the contents of one of the eight base registers, or one of the eight index registers, or one of the 32 D registers.

The six low-order bits of the top-of-stack word select the processor register to be read. The decoding of these six bits is as follows:

a. If bits 5 & 4 = 10 = Index Register

and bits 2:3 = 0, = PIR

= 1, = SIR

= 2, = DIR

= 3, = TIR, BUF3

= 4, = LOSR

= 5, = BOSR

= 6, = F

= 7, = BUF

b. If bits 5 & 4 = 11 = Base Register

and bits 2:3 = 0, = PBR

= 1, = IBR

= 2, = DBR

= 3, = TBR, BUF2

= 4, = S

= 5, = SNR

= 6, = PDR

= 7, = TEMP

If bit 5 is zero, then bits 4:5 select the D register equal to the binary value of the bits (e.g., if bits 4:5 = 00101 then D register 5 is selected).

SET PROCESSOR REGISTER (SPRR) (V)B9

The Set Processor Register operator sets the processor register addressed by the second word in the stack to the value contained in the top-of-stack word. On every SPRR the contents of the stack buffer are purged and stored in main memory.

NOTE

Register Address assignments are as follows:

Address (decimal)	Register Name	Register Usage
0-31		Display Registers
32	PIR	Program Index
33	SIR*	Source Index
34	DIR*	Destination Index
35	TIR*	Table Index
36	LOSR	Limit of Stack
37	BOSR	Base of Stack
38	F	Most Recent MSCW Address

Address (decimal)	Register Name	Register Usage
39	BUF*	Scratch
40	ID*	Interrupt Identifier
41		Spare
42	IMR	Interrupt Mask Register
43		Spare
44	IFR	Interrupt Fault Register
45		Spare
46	INT*	Interval Timer
47	TOD*	Time of Day
48	PBR	Program Base Register
49	SBR	Source Base Register
50	DBR	Destination Base Register
51	TBR	Table Base Register
52	S	Top of Stack Address
53	SNR	Current Stack Vector Index
54	PDR	Current Segment Descriptor Index
55	TEMP	Scratch
56	ADZ*	Alternate D0 Register
57	APIR*	Alternate PIR
58	ALL1*	All ones
59		Spare

Address (decimal)	Register Name	Register Usage
60	PFR*	Processor Fail Register
61	PMR*	Processor Mode Register
62	HMT*	Hardware Measurement Timer
63	CSMA*	Cold Start Memory Address

*B 7700 Exclusive

Unpack Operators

UNPACK ABSOLUTE, DESTRUCTUVE (UABD)
(V)D1

The Unpack Absolute, Destructive operator unpacks a string of left-justified digits from the second operand in the stack. The top-of-stack operand defines the string length (in 4-bit digits) of the second operand in the stack. The specified number of digits are transferred from the second operand to the destination. The third item in the stack is a string descriptor destination pointer. Zone fill in the destination is as follows:

1. If the destination bit format is 8-bit ASCII, the digits are transferred to the destination string with the leading-zone bits set to 0011.

2. If the destination bit format is 6-bit BCL, the digits are transferred to the destination with the two leading-zone bits set to zero.

3. If the destination bit format is 8-bit EBCDIC, the digits are transferred to the destination string with the four leading-zone bits set to one.

4. If the destination character size is 0, it is set to 6 and the digits are transferred to the destination string with the two leading-zone bits set to zero (BCL).

UNPACK ABSOLUTE, UPDATE (UABU)
(V)D9

The Unpack Absolute, Update operator performs an Unpack Absolute, Destructive operation. At the completion of the operation the destination pointer is updated.

UNPACK SIGNED, DESTRUCTIVE (USND)
(V)D0

The Unpack Signed, Destructive operator performs an Unpack Absolute, Destruction operation except that the external sign is considered.

If the external sign flip-flop is ON (indicating negative data) then a zone of 10 is inserted in the last 6-bit character or a zone of 1101 is inserted in the last 8-bit byte. For 8-bit ASCII formatted data the negative sign is indicated in the least-significant byte by a zone of 1111. If the data format of the destination is 4 bits, the first digit position of the destination string is set to 1101 if the external sign flip-flop is ON; if the external sign flip-flop is OFF the first digit of the destination string is set to 1100.

UNPACK SIGNED, UPDATE (USNU) (V)D8

The Unpack Signed, Update operator performs an Unpack Signed, Destructive operation. At the completion of the operation the destination pointer is updated.

LINKED LIST LOOKUP (LLLU) (V)BD

The Linked List Lookup operator searches a linked list of words.

This operator expects the third stack entry (bits 27:28) to contain an argument, the second stack entry to contain a non-indexed data descriptor, and the top-of-stack to contain an operand index value pointing into a linked-list of words. The argument is not required to be an integer, but only the right-most 28 bits are significant after the argument has been integerized as required. The base address, size field, and argument are saved throughout the operator.

The word addressed by the base plus the index value is read into local storage. Bits 47:28 are compared to the argument value. If the argument of the linked-list word is less than the argument value, this process is repeated using the link as the new index. If the linked-list argument is greater than or equal to the argument value, the operation is complete. At completion the top-of-stack register contains an index which points to the link that points to the satisfying argument.

If the value of the link portion of the linked-list word is equal to zero, the top-of-stack register is set to minus one (-1) and marked full as the operation is completed.

If the index value in the linked-list word is greater than the length value from the descriptor, an invalid-index interrupt is set and the operation is terminated.

When the first word in the stack at the start of this operator is not an operand an invalid-operand interrupt is set and the operation is terminated.

If the data descriptor has been indexed, the invalid-operand interrupt is set and the operation is terminated.

MASKED SEARCH FOR EQUAL (SRCH) (V)BE

The Masked Search for Equal operator searches a data word list for a word identical to the third word in the

stack. At the beginning of this operator, the top word in the stack contains a data descriptor, the second word in the stack contains a 51-bit mask, and the third word in the stack contains a 51-bit argument value. If the descriptor is not present, the presence-bit interrupt is set and the operator is exited. Otherwise, if the descriptor is indexable (i.e., bit 45 equals zero), the indexed bit (bit 45) is turned ON and the length/index field value is decreased by 1.

The descriptor points to a word which is then fetched into the processor. This word is ANDed with the mask and a test is made to determine whether the result is identical to the argument.

When an equal compare is made, the second stack register is marked empty; the top-of-stack contains an index which gives the address of the last word inspected.

When a not-equal compare is made, the index value is decreased by one and the operation is repeated (except when the index value is zero). When the index value is zero, the top-of-stack register is set to -1 and marked full, the second stack register is marked empty, and the operator is exited.

MOVE TO STACK (MVST) (V)AF

The Move to Stack operator causes the processor's environment (or addressing space) to terminate and to be moved from the current stack to the program stack specified by the operand in the top of stack.

The operator builds a top-of-stack control word and places it at the base of the current stack as addressed by the base-of-stack register.

The operand in the B register is integerized and checked for invalid index against the stack vector. The value in the B register is added to the address field of the stack vector descriptor (at D[0]+2), to address the descriptor for the new stack.

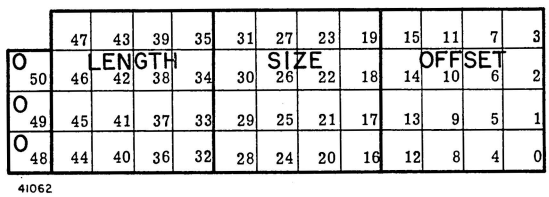
The data descriptor for the requested stack is accessed. If its presence bit is ON, the address field is placed into the base-of-stack register. The top-of-stack control word is brought up and the stack is marked "active" by storing the processor ID at the base of the stack. The TSCW is distributed and the D registers are updated.

If during the integerization the operand in the B register is too large, the integer-overflow interrupt is set and the operation is terminated. The stack buffer is purged on every execution of MVST.

If the index value is less than zero or greater than the length field of the data descriptor for the stack vector array, an invalid index interrupt is set and the operation is terminated.

OCCURS INDEX (OCRX) (V)85

The Occurs Index operator is used to index a field in an array. This operator requires an Occurs Index Word (OIW) in the top-of-stack and an index value (operand) in the second stack position. The format of the OIW is shown below.



41062

The operator creates a new index value from the OIW and the operand in the following manner:

The operand is integerized. If the resulting index is greater than the maximum integer value (549,755, 813,887), the integer overflow interrupt is set and the operation is terminated. If either the OIW or the index has a value of zero, or if the index is less than zero or greater than the SIZE field of the OIW, the invalid index interrupt is set and the operation is terminated.

The LENGTH field of the OIW is multiplied by the index value ±15:161 minus 1, and that value is added to the OFFSET field of the OIW, resulting in the new index value. The two original top-of-stack items are deleted and the new index value is left in the top-of-stack.

In the OIW the "length" field gives the number of characters in a field; the "size" field gives the number of fields in the array; the "offset" field indicates the beginning of the first character position in the first field of the first word.

TRANSLATE (TRNS) (V)D7

The Translate operator transfers from the source to the destination the number of characters specified by the second item in the stack while performing the following translation.

The translation uses a table containing the translated characters. The word in the top-of-stack is a descriptor that addresses the translation table. The second operand in the stack specifies the length of the string. The third word in the stack is a descriptor addressing the source string (or an operand which is the source string). The fourth word in the stack is a descriptor addressing the destination string. Source and destination are updated at the end of the operation.

Translation occurs as follows: The specified string character is used as an index into the table to locate a character. An all zeroes character locates the most significant character in the table. The located character is transferred to the destination string.

The least significant 32 bits of each table word provide four 8-bit characters. Table sizes are as follows:

a. 4-bit digits provide a 4-word table length.

b. 6-bit characters provide a 16-word table length.

c. 8-bit bytes provide a 64-word table length.

Operators Exclusive to the B 7700

SET MEMORY INHIBITS (SINH) (V)A8 (CONTROL STATE OP)

The Set Memory Inhibits operator transfers the inhibit settings in the second stack register to the memory module specified in the top stack register. The two top-of-stack registers are marked empty. (All tags are legal.) The memory module number is given in the top-of-stack (bits 3:4). The inhibit field setting is given in the second item in the stack (bits 7:8).

SET MEMORY LIMITS (SLMT) (V)AA (CONTROL STATE OP)

The Set Memory Limits operator transfers the limits and availability settings in the second stack register to the memory module specified in the top-of-stack register. The two top-of-stack registers are then marked empty. (All tags are legal.) The limits specify the range of addresses (in 16K increments) behind the module and the availability setting specifies which stack(s) (of a possible four) are to be used. (All tags are legal.) The top-of-stack gives the memory module number (bits 3:4). The second item in the stack gives module availability (bits 3:4) and memory addressing limits: upper limit (bits 15:6) and lower limit (bits 9:6).

FETCH MEMORY FAIL REGISTER (FMFR) (V)AC (CONTROL STATE OP)

The Fetch Memory Fail Register operator fetches the contents of the fail register from the memory module specified in the top-of-stack (bits 3:4). The contents of the fail register are placed in the top-of-stack.

IGNORE PARITY (IGPR) (V)48 (CONTROL STATE OP)

The Ignore Parity operator is used for confidence checking and requires the processor to be in the control state. In control mode 0, words entering the CPM are checked for correct parity but the IGPR operator sets the IGP flip-flop which inhibits transmission of parity error messages for those words with incorrect parity. Likewise, IGPR inhibits correct parity generation before storage for those words detected in the CPM with incorrect parity.

Parity error interrupts and new parity generation will be inhibited with the CPM in control mode 0 by IGPR until any one of the following occurs:

1. Some other interrupt causes the CPM to move to control mode 1.
2. Another IGPR is decoded while the CPM is in a control mode greater than zero.
3. Or the CPM returns to normal state.

Any one of the above conditions cause the IGP flip-flop to be reset and the CPM to resume parity error interrupts and generation of new parity.

PAUSE UNTIL INTERRUPT (PAUS) (V)84

The Pause Until Interrupt operator suspends program execution until an external interrupt or an interval timer interrupt occurs. If the processor is operating in control state, the operation continues in sequence; to clear the interrupt the INT. I.D. must be read. If the

processor is operating in normal state, the interrupt is handled as in IDLE.

INTERRUPT CHANNEL N (INCN) (V)8F

The Interrupt Channel N operator sends signals to the channel or channels specified by the top-of-stack. The top-of-stack is then marked empty. Bit 0 interrupts channel 0; bit 1 interrupts channel 1; etc.

STOP (STOP) (V)BF

The STOP operator causes an unconditional halt of the central processor. The STOP operator is primarily used for diagnostic purposes. The processor may be restarted by pressing and releasing the START button on the processor control panel.

ORDER OF EDIT MODE OPERATOR DESCRIPTIONS

Name	Mnemonic	Mode Prefix & Code	Page
Insert Operators			
Insert Unconditional	INSU	(E)DC	3-161
Insert Conditional	INSC	(E)DD	3-161
Insert Display Sign	INSG	(E)D9	3-161
Insert Overpunch	INOP	(E)D8	3-161
Move Operators			
Move With Insert	MINS	(E)D0	3-162
Move With Float	MFLT	(E)D1	3-162
Move Characters	MCHR	(E)D7	3-162
Move Numeric Unconditional	MVNU	(E)D6	3-163
Skip Operators			
Skip Forward Source Characters	SFSC	(E)D2	3-163
Skip Reverse Source Characters	SRSC	(E)D3	3-163
Skip Forward Destination Characters	SFDC	(E)DA	3-163
Skip Reverse Destination Characters	SRDC	(E)DB	3-163
Miscellaneous Edit Mode Operators			
Reset Float	RSTF	(E)D4	3-163
End Float	ENDF	(E)D5	3-163
End Edit	ENDE	(E)DE	3-164
Universal Operators			
Conditional Halt	Halt	(U)DF	3-164
Invalid Operator	NVLD	(U)FF	3-164
No Operation	NOOP	(U)FE	3-164

EDIT MODE OPERATORS

Edit Mode operators perform editing functions on strings of data. Edit functions are normally involved in preparing information for output. These operators include Insert, Move, and Skip, in the form of micro-operators in either the program string or in a separate table. In the program string, they are single micro-operators and are entered by use of the Execute Single Micro or Single Pointer operators. (See the "Enter Edit Mode Operator" descriptions.) If the micro-operators are in a table, the table becomes the program string that is to be executed. This table is entered by means of the Table Enter Edit operators, and is exited through the End Edit micro-operator.

If the source or destination data has the memory protect bit (bit 48) equal to one, the segmented-array interrupt is set and the current micro-operator is terminated.

Insert Operators

INSERT UNCONDITIONAL (INSU) (E)DC

The Insert Unconditional micro-operator places an insert character into the destination string for the number of times specified by the repeat value. When this operator is entered by a Table Enter Edit operator, the repeat is in the syllable following the micro-operator syllable, and the insert character is in the next syllable (the third syllable).

When this operator is entered via an Execute Single Micro Instruction operator, the repeat field is in the top-of-stack operand and the insert character is the second syllable. The operator length is then two syllables.

INSERT CONDITIONAL (INSC) (E)DD

The Insert Conditional operator inserts the character defined by the third syllable into the destination string if the float toggle is OFF. If the float toggle is ON, the character defined by the fourth syllable is inserted into the destination string. The insertion is repeated the number of times specified by the second syllable when this operator is entered by the Table Enter Edit operation.

When this operator is entered via an Execute Single Micro Instruction operator, the repeat field is the top-of-stack operand. The operator length is then three syllables.

INSERT DISPLAY SIGN (INSG) (E)D9

The Insert Display Sign operator inserts the character defined by the second syllable into the destination string if the external sign flip-flop is set; otherwise, the character defined by the third syllable is inserted.

INSERT OVERPUNCH (INOP) (E)D8

The Insert Overpunch micro-operator places a sign overpunch in the destination string character. If the external sign flip-flop is reset, the operator skips one destination string character. If the external

sign flip-flop is set, the zone bits of the destination character are set to 10 for 6-bit data and to 1101 for 8-bit EBCDIC data; the destination pointer is then advanced one character. The zone bits for 8-bit ASCII data are set to 1111.

Move Operators

MOVE WITH INSERT (MINS) (E)D0

The Move With Insert micro-operator performs a Move Numeric Unconditional or an Insert operation under control of the float flip-flop.

If the float flip-flop is set, a Move Numeric Unconditional operation is performed. If the float flip-flop is reset and the source character numeric is zero, the character defined by the third syllable is transferred to the destination string. If the float flip-flop is reset and the source character numeric is not zero, then the float flip-flop is set and a Move Numeric Unconditional is performed.

The number of characters transferred from the source string to the destination string is defined by the repeat value. In Table Edit mode the second syllable is the repeat value and the third syllable is the character to be inserted under control of the float flip-flop. In Execute Single Micro mode the repeat field value is in the word in the top-of-stack and the insert character is in the syllable following the micro-operator syllable.

MOVE WITH FLOAT (MFLT) (E)D1

If the float flip-flop is set, the Move with Float operator causes a Move Numeric Unconditional operation to be performed.

If the float flip-flop is reset and the source character numeric is zero, then the character defined by the third syllable is transferred to the destination string.

If the float flip-flop is reset and the source character numeric is not zero, then the float flip-flop is set. If the external sign flip-flop is set, the character defined by the fourth syllable (the second insert character) is transferred to the destination string; otherwise, the character defined by the fifth syllable (the third character) is transferred. Then a Move Numeric Unconditional operator is performed.

In Table Edit mode, the above operation is repeated for the number of characters specified by the second syllable; the third, fourth, and fifth syllables are the insert characters.

When this operand is entered via an Execute Single Micro instruction, the repeat field is the top-of-stack operand. The operand length is then four syllables, three of which contain insert characters.

MOVE CHARACTERS (MCHR) (E)D7

The Move Characters operator transfers the number of characters specified by the second syllable from the source string to the destination string, if the operator is entered this by the Table Enter Edit, Destructive operator.

When this operator is entered via an Execute Single Micro Destructive instruction, the number of characters transferred is specified by the top-of-stack operand. The operator length is then one syllable.

MOVE NUMERIC UNCONDITIONAL (MVNU) (E)D6

The Move Numeric Unconditional operator transfers from the source string to the destination string the number of characters specified by the second syllable. The zones are not transferred but are set to 00 for 6-bit data, to 1111 for 8-bit EBCDIC data, and to 0011 for 8-bit ASCII data.

When this operator is entered via an Execute Single Micro instruction, the number of characters transferred is specified by the top-of-stack operand. The operator length is then one syllable.

Skip Operators

SKIP FORWARD SOURCE CHARACTERS (SFSC) (E)D2

The Skip Forward Source Characters operator causes a skip forward for the number of source characters specified by the syllable following the micro-operator's syllable, if the entry to this operator is by the execution of the Table Enter Edit operator. When this operator is entered via an Execute Single Micro, Destructive instruction, the number of characters skipped is specified by the top-of-stack operand. The operator length is then one syllable.

SKIP REVERSE SOURCE CHARACTERS (SRSC) (E)D3

The Skip Reverse Source Characters operator decrements the source pointer register for a skip in reverse for the number of source characters specified by the second syllable.

SKIP FORWARD DESTINATION CHARACTERS (SFDC) (E)DA

The Skip Forward Destination Characters operator causes a skip forward for the number of destination characters specified by the second syllable.

SKIP REVERSE DESTINATION CHARACTERS (SRDC) (E)DB

The Skip Reverse Destination Character operator causes a skip in reverse for the number of destination characters specified by the second syllable.

RESET FLOAT (RSTF) (E)D4

The Reset Float micro-operator sets the float flip-flop to zero.

END FLOAT (ENDF) (E)D5

The End Float operator transfers to the destination string the character defined by the second syllable if the float flip-flop is reset and the external sign flip-flop is set.

If the float flip-flop is reset and the external sign flip-flop is reset, the character defined by the third syllable is transferred to the destination string.

If the float flip-flop is set, the End Float operator is treated as a NO-OP.

END EDIT (ENDE) (E)DE

The End Edit operator terminates the execution of this string of edit micro-operators in Table Enter Edit mode. The micro program string must end with the End Edit operator.

Universal Operators

NO OPERATION (NOOP) (U)FE

No operation takes place when the NOOP operator is encountered. The program index register (PIR) and the program syllable register (PSR) are advanced to the next syllable. This operator is valid in Variant Mode and Edit Mode but is not valid in Single Micro mode.

CONDITIONAL HALT (HALT) (U)DF

This operator halts the processor if the conditional halt switch is in the ON position. If the conditional halt switch is OFF, the operator is treated as a NOOP. This operator is valid in Variant Mode and Edit Mode but is not valid in Single Micro mode.

INVALID OPERATOR (NVLD) (U)FF

This operator sets the invalid operand interrupt.

ORDER OF VECTOR MODE OPERATOR DESCRIPTIONS

Name	Mnemonic	Mode Prefix & Hex Code	Page
Enter Vector Operators (See Primary Mode)			
Single-Word Vector Mode	VMOS	(Z)E7	3-117
Multiple-Word Vector Mode	VMOM	(Z)EF	3-117
Operators Used on Vectors			
Vector Branch	VEBR	(Z)EE	3-167
Vector Exit	VXIT	(Z)E6	3-167
Vector Stack Operators			
Load A	LDA	(Z)E0	3-167
Load A Increment	LDAI	(Z)EI	3-168
Load B	LDB	(Z)E2	3-167
Load B Increment	LDBI	(Z)E3	3-168
Load C	LDC	(Z)E4	3-167
Load C Increment	LDCI	(Z)E5	3-168
Store A	STA	(Z)E8	3-168
Store A Increment	STAI	(Z)E9	3-168
Store B	STB	(Z)EA	3-168
Store B Increment	STBI	(Z)EB	3-168
Store C	STC	(Z)EC	3-168
Store C Increment	STCI	(Z)ED	3-168
Double Load A	DLA	(Z)F0	3-168
Double Load A Increments	DLAI	(Z)F1	3-169
Double Load B	DLB	(Z)F2	3-168

Name	Mnemonic	Mode Prefix & Hex Code	Page
Double Load B Increment	DLBI	(Z)F3	3-169
Double Load C	DLC	(Z)F4	3-169
Double Load C Increment	DLCI	(Z)F5	3-169
Double Store A	DSA	(Z)F8	3-169
Double Store A Increment	DSAI	(Z)F9	3-169
Double Store B	DSB	(Z)FA	3-169
Double Store B Increment	DSBI	(Z)FB	3-169
Double Store C	DSC	(Z)FC	3-169
Double Store C Increment	DSCI	(Z)FD	3-169
Vector Fetch and Store Operators			
Vector Fetch	FTCH	(Z)00→3F	3-170
Vector Store	STOR	(Z)40→7F	3-170
Universal Operators			
No Operation	NOOP	(U)FE	3-170
Invalid Operator	NVLD	(U)FF	3-170

VECTOR MODE OPERATORS

Certain scientific and mathematical analysis involves manipulation of vectors and matrices. Programming wise a vector is a string of numbers which may be a row, a column, or a diagonal in an array of numbers. Operations may be performed on each item in the vector, on multiple vectors, between vectors, on a matrix of numbers within the array, between matrices, etc. Forty operators are available for vector (arithmetic, logical, relational, etc.) manipulations. (The two operators (VMOS and VMOM) used to enter vector mode operation are described in the primary mode section of this document.)

In vector mode the source, destination, and table pointer areas are used as index registers. Absolute addresses and their corresponding increments of items in the vector are extracted from the stack by VMOS or VMOM and placed in the index registers. The stack initially contains three descriptors and three or four operands: If bit 44 of the descriptor in the top of the stack is ON, then the second item is an operand and is stored as the repeat count; otherwise, a default repeat of 1,048,575 is used. Each of the three descriptors represents a full or partial array of operands to be operated upon by the repetition of the word of code.

Vector mode operation is entered via the VMOS operator for a single-word vector mode operation or via the VMOM operator for multiple-word vector mode operations. The enter vector operators extract three absolute

addresses, their corresponding increment, and an optional length from the stack and place them as index registers in the pointer areas of the IC memory. Vector stack operators load, store, or increment the top-of-stack with a single-precision or a double-precision operand for single program word loops or for multiple program word loops. The number of program iterations is indicated by "length." Following the enter vector mode instruction, up to five syllables are ignored and the next full program word is fetched into the P register. If a repeat (length) is specified, this word will be executed that number of times. (If the repeat is not given, the execution is controlled by the operator itself.) In VMOM the number of program operators accessed is controlled by the operators Vector Branch and Vector Exit. Only those operators listed below (by mnemonic) may be used to manipulate vectors or matrices in vector mode.

Operator Type	Mnemonics of Operators Permitted in Vector Mode
Arithmetic	ADD SUBT MULT MULX DIVD INDIV RDIV NTIA NTGR
Bit	BRST CHSN DBRS DBST
Literal Call	ONE ZERO
Logical	LAND LEQV LNOT LOR
Relational	EQUL GREQ GRTR LESS LSEQ NEQL SAME
Scale	DSLF DSRF DSRR DSRS DSRT
Stack	DLET DUPL EXCH PUSH
Type Transfer	SNGL SNGT XTND
Universal	HALT NOOP

Vector Branch (VEBR) (Z)EE

Vector Branch is a three-syllable operator used only in VMOM. The two syllables following the operator name contain the branch address. Vector Branch examines length. If it is greater than zero, length is decremented by one, the next two program syllables containing the branch address are skipped, and the program is resumed at the following syllable. If the examined length is zero, vector mode is exited, and normal operation commences with the program word located by the branch address.

Vector Exit (VXIT) (Z)E6

The Vector Exit operator is used exclusively with VMOM. It causes the program to return to normal operation.

Vector Stack Operators

Vector stack operators store or load the top of stack from absolute memory addresses with a single-precision or double-precision operand, and, if specified, increment the loading or storing address.

LOAD A (LDA) (Z)E0

The stack is adjusted (0,2) and the single-precision word selected by Pointer A (BRS1) is loaded into the top of stack.

LOAD B (LDB) (Z)E2

The stack is adjusted (0,2) and the single-precision word selected by Pointer B (BRS2) is loaded into the top of stack.

LOAD C (LDC) (Z)E4

The stack is adjusted (0,2) and the single-precision word selected by Pointer C (BRS3) is loaded into the top of stack.

LOAD A - INCREMENT (LDAI) (Z)E1

The stack is adjusted (0,2) and the single-precision word selected by Pointer A (BRS1) is loaded into the top of stack. Pointer A is increased by its increment (IRS1) following the transfer.

LOAD B - INCREMENT (LDBI) (Z)E3

The stack is adjusted (0,2) and the single-precision word selected by Pointer B (BRS2) is loaded into the top of stack. Pointer B is increased by its increment (IRS2) following the transfer.

LOAD C - INCREMENT (LDCI) (Z)E5

The stack is adjusted (0,2) and the single-precision word selected by Pointer C (BRS3) is loaded into the top of stack. Pointer C is increased by its increment (IRS3) following the transfer.

STORE A (STA) (Z)E8

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer A (BRS1).

STORE B (STB) (Z)EA

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer B (BRS2).

STORE C (STC) (Z)EC

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer C (BRS3).

STORE A - INCREMENT (STAI) (Z)E9

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer A (BRS1). Pointer A is increased by its increment (IRS1) following the transfer.

STORE B - INCREMENT (STBI) (Z)EB

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer B (BRS2). Pointer B is increased by its increment (IRS2) following the transfer.

STORE C - INCREMENT (STCI) (Z)ED

The stack is adjusted (1,2) and the single-precision word in the top of stack is stored in the location given by Pointer C (BRS3). Pointer C is increased by its increment (IRS3) following the transfer.

DOUBLE LOAD A (DLA) (Z)F0

The stack is adjusted (0,2) and the double-precision word selected by Pointer A (BRS1) is loaded into the top of stack.

DOUBLE LOAD B (DLB) (Z)F2

The stack is adjusted (0,2) and the double-precision word selected by Pointer B (BRS2) is loaded into the top of stack.

DOUBLE LOAD C (DLC) (Z)F4

The stack is adjusted (0,2) and the double-precision word selected by Pointer C (BRS3) is loaded into the top of stack.

DOUBLE LOAD A - INCREMENT (DLAI)
(Z)F1

The stack is adjusted (0,2) and the double-precision word selected by Pointer A (BRS1) is loaded into the top of stack. Pointer A is increased by its increment (IRS1) following the transfer.

DOUBLE LOAD B - INCREMENT (DLBI)
(Z)F3

The stack is adjusted (0,2) and the double-precision word selected by Pointer B (BRS2) is loaded into the top of stack. Pointer B is increased by its increment (IRS2) following the transfer.

DOUBLE LOAD C - INCREMENT (DLCI)
(Z)F5

The stack is adjusted (0,2) and the double-precision word selected by Pointer C (BRS3) is loaded into the top of stack. Pointer C is increased by its increment (IRS3) following the transfer.

DOUBLE STORE A (DSA) (Z)F8

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer A (BRS1).

DOUBLE STORE B (DSB) (Z)FA

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer B (BRS2).

DOUBLE STORE C (DSC) (Z)FC

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer C (BRS3).

DOUBLE STORE A - INCREMENT (DSAI)
(Z)F9

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer A (BRS1). Pointer A is increased by its increment (IRS1) following the transfer.

DOUBLE STORE B - INCREMENT (DSBI)
(Z)FB

The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer B (BRS2). Pointer B is increased by its increment (IRS2) following the transfer.

DOUBLE STORE C - INCREMENT (DSCI)
(Z)FD

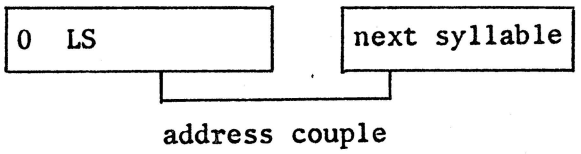
The stack is adjusted (1,2) and the double-precision word in the top of stack is stored in the location given by Pointer C (BRS3). Pointer C is increased by its increment (IRS3) following the transfer.

Vector Fetch and Store Operators

The Vector Fetch and Vector Store operators are used in conjunction with addresses relative to a Mark Stack Control Word (only when a "length" is passed) to load or store operands to or from the top-of-stack. The operands are moved to or from the memory location indicated by the normal address-couple decoding convention (as in Value Call) unless the memory location is protected. (An attempt to access a protected memory location causes a vector mode exit and a memory protect interrupt.) A single-precision operand is placed in the top-of-stack position designated A0. A double-precision operand uses both

the A_0 and the A_1 position of the top-of-stack; the least significant half of the mantissa is placed in the A_1 position.

The format of the Vector Fetch and Vector operators is:



where, if

LS=0 then load (FTCH operator) or when

LS=1 then store (STOR operator).

Certain precautions must be exercised in using the Fetch and Store operators. It must be assured that:

a. Words being loaded or stored are operands.

b. The precision of the operand to be stored agrees with that of the receiving location.

c. The address couple does not reference a lexicographical level which is higher than the level in which the address couple is now.

VECTOR FETCH (FTCH) (Z)00 THRU (Z)3F

The Vector Fetch operator loads into the top-of-stack the operand referenced by this address couple of this operator. The address couple is formed by the concatenation of the six low-order bits of the first syllable of this operator with the eight bits of the second syllable of this operator. The address couple (in the stack) references a memory storage location relative to an MSCW. The operand may be single precision or double precision, thus, one or two words will be loaded, respectively.

After the operand is brought to the top-of-stack, the top-of-stack is marked full.

VECTOR STORE (STOR) (Z)40 THRU (Z)7F

The Vector Store operator stores single-precision or double-precision operands from the top-of-stack into a memory area relative to an MSCW. The least significant six bits of the first syllable of the STOR operator are concatenated with the eight bits of the second syllable of the STOR operator to form an address couple which links the storage area with the MSCW.

After the operand is stored, the operand and its address are deleted from the stack.

CHAPTER IV
INPUT/OUTPUT SUBSYSTEM

SECTION 1
GENERAL DESCRIPTION AND OPERATION OF THE INPUT/OUTPUT MODULE

PRELIMINARY

The B 7700 Input/Output Module, which is referred to as the IOM, is designed to serve as a buffer and control unit for all B 7700-system input and output data transfers. The requests serviced by the IOM are obtained from a queue of requests constructed by the Central Processing Module (CPM) and stored in the Memory Storage Unit (MSU).

The IOM is informed, via an interrupt from the CPM, of the presence of a service request in the MSU. Once thus informed, the IOM controls the desired input/output operation in its entirety; therefore, the CPM time required to initiate and I/O operation is only that needed to construct a request, queue it in the MSU, and interrupt the IOM.

BASIC IOM CONFIGURATION

As is illustrated in figure IV-1-1, the IOM consists of six major subsections. Each subsection is totally independent of the other subsections, and operates asynchronously with them.

Control Word Flow

All control word flow between main memory and up to 255 system peripherals is via (1) an IOM subsection called the Memory Interface Unit (MIU), (2) an IOM control subsection called the Translator (XLATOR), and (3) one of four IOM subsections, each of which is

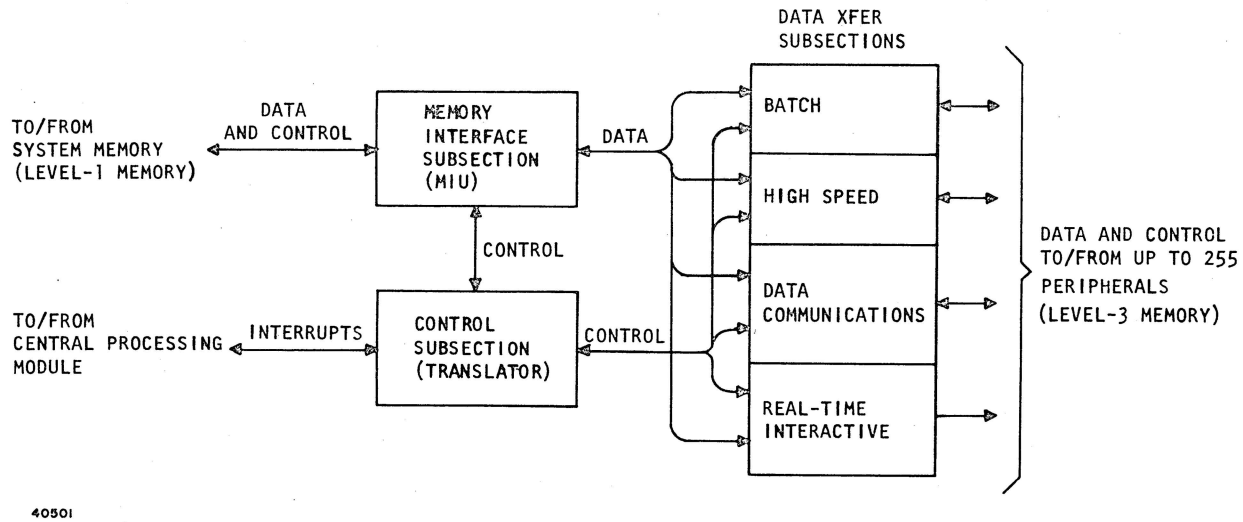


Figure IV-1-1. IOM Basic Block Diagram

uniquely buffered to match the class of data transfer assigned to it. The XLATOR subsection routes control of a given job request to one of these subsections dependent upon data class (batch, high speed, data communications, or real-time interactive).

Data Flow

All data flow between main memory and the peripherals is via the appropriate data-transfer subsection and/or the MIU; the XLATOR is not involved and is free for control of additional job requests. When a data transfer is complete, however, the XLATOR is given control over job termination, and control flow to main memory is via the appropriate data-transfer subsection, the XLATOR, and the MIU.

IOM/Peripheral Interface Configuration

Figure IV-1-2 illustrates typical peripheral devices which may be assigned to each data-transfer class; also illustrated are the data-transfer subsection names which are henceforth referred to. The following is a brief description of the interface capability of each subsection, and its physical relationship to typical peripheral equipment. The descriptions are presented in reference to figure IV-1-3, which illustrates the interface capability provided when two maximum-configuration Input/Output Modules and appropriate exchanges are used. It should be noted that a maximum of 28 peripheral controllers (excluding DFO's and DCP's) may be connected to a single IOM.

PERIPHERAL CONTROL INTERFACE (PCI)

The PCI of a single IOM consists of either one or two interface sections, dependent upon user requirements. Each section has 10-channel interface capability, for a total maximum capacity of 20 channels per IOM.

Each PCI 10-channel section can service a single peripheral control cabinet (PCI/PCC), which may contain up to five large-controller channels and up to five small-controller channels. In each PCC cabinet, the large channels are numbered 0 thru 4 and the small channels are numbered 5 thru 9. Table IV-1-1 lists the various controllers which may be housed in PCI/PCC cabinets. For further details pertaining to any of the listed equipments, refer to the appropriate equipment FETM.

Any combination of five small controls may be housed in the PCI/PCC cabinet. The large controls (SLC and MTC) may be connected to the peripheral units directly, or, in the case of the MTC only, via exchanges. Any unused channels in the PCC cabinet are left empty.

The PCI multiplexes all 20 channels by generating overlapping 1-microsecond data-service cycles and by use of "windows" in a self-contained local memory. In the typical configuration illustrated in figure IV-1-3, the use of two IOMs and appropriate exchanges (4X16) allows access by either IOM of 64 magnetic tape units. IOM number 1 is illustrated as having access to an additional non-exchange magnetic tape unit, and both IOM's are illustrated as having access to SPO units via Single Line Controls (SLC).

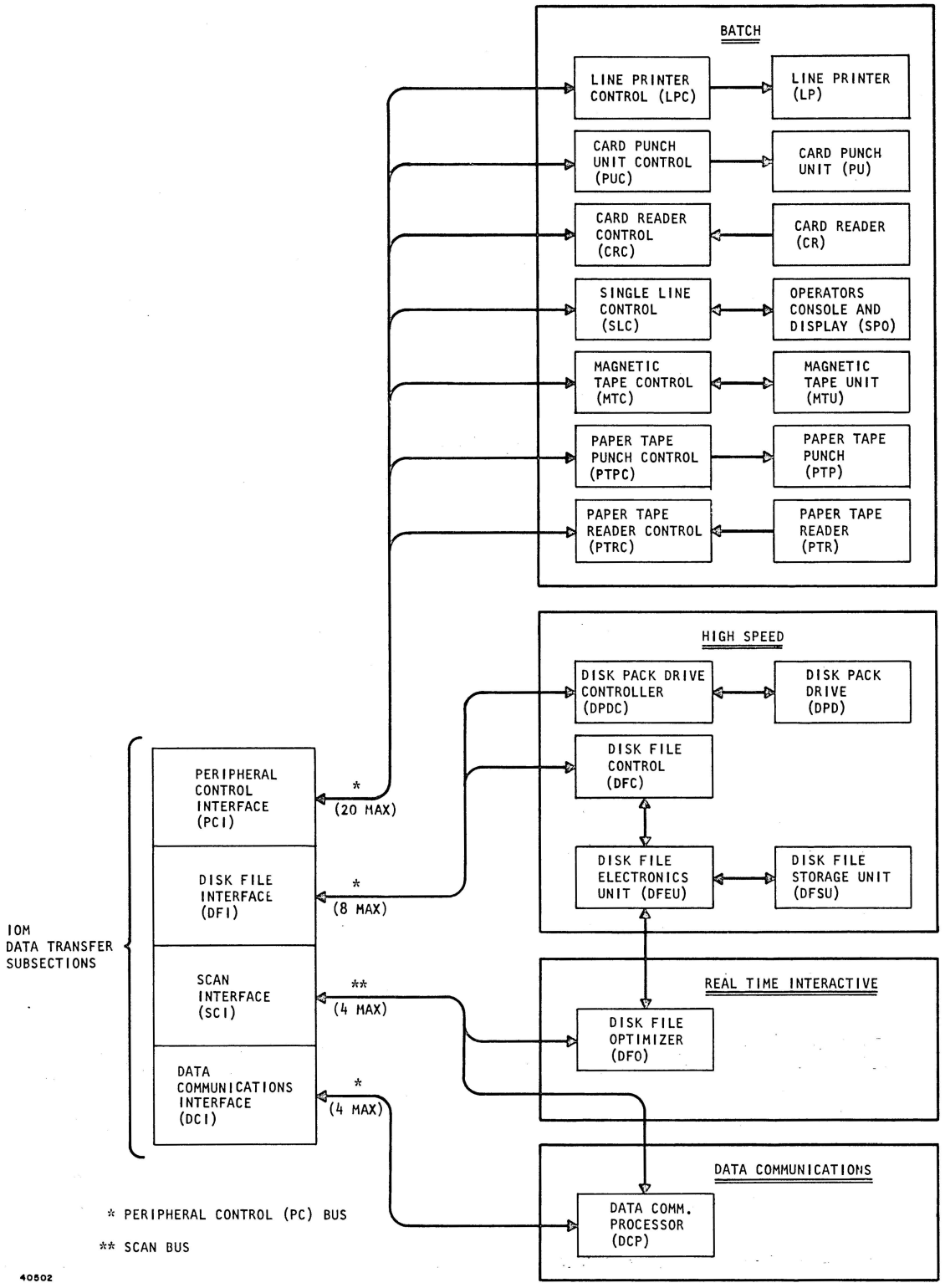


Figure IV-1-2. Typical Data-Transfer Classifications and Related IOM Subsections

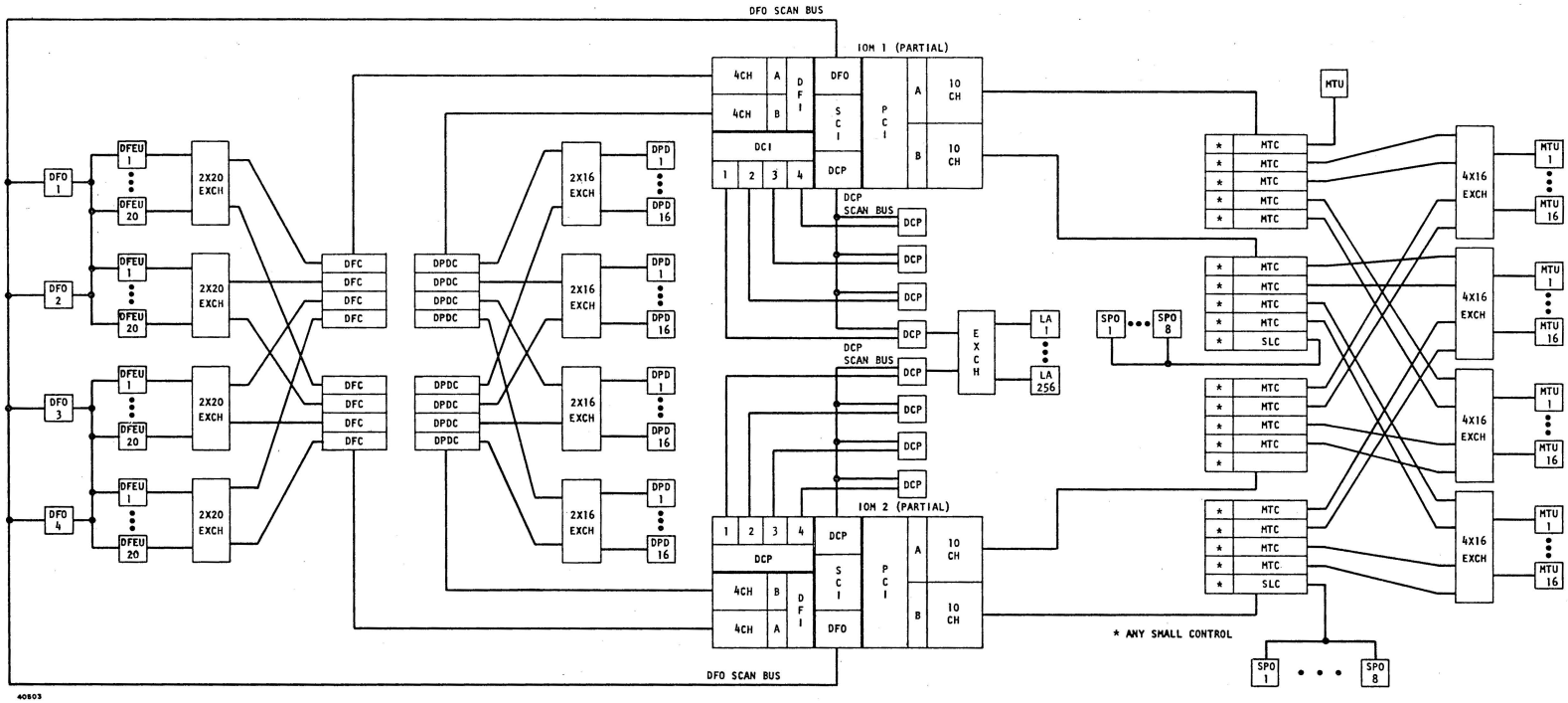


Figure IV-1-3. Typical IOM/Peripheral Configuration

Table IV-1-1. PCI/PCC Channels

Size	Peripheral Controls Model	Type	Peripheral Units
L	B 7381-11	18/36 KB, NRZ, 9 CH. MTC	18/36 KB, 9 CH. NRZ CLUSTERS (800 BPI, 2, 3, OR 4 STATION)
L	B 7381-12	36/72 KB, PE, 9 CH. MTC	36/72 KB, 9 CH. PE CLUSTERS (1600 BPI, 2, 3, OR 4 STATION)
L	B 7381-14	18/36 KB, NRZ, 9 CH., DUAL MTC WITH 2X8 EXCH.	18/36 KB, 9 CH., NRZ CLUSTERS (800 BPI, 2, 3, OR 4 STATION)
S	B 7240 WITH B 9943	PRINTER CONTROL PRINTER MEMORY	300/400 LPM, 120/132 PRINT POSITION PRINTERS 860 LPM, 120 PRINT POS; 725 LPM, OCR "A" AND "B" NUMERIC & STD; 1100 LPM, 120 PRINT POSITION; 900 LPM, OCR "A" AND "B" NUMERIC AND STD PRINTERS
S	B 7120 AND B 9926	PAPER TAPE READER CONTROL AND INPUT CODE TRANSLATOR	PAPER TAPE READER, 500-1000 CPS
S	B 7220 AND B 9928	PAPER TAPE PUNCH AND OUTPUT CODE TRANSLATOR	PAPER TAPE PUNCH, 100 CPS
L	B 7381-15	36/72 KB, PE, 9 CH., DUAL MTC WITH 2X8 EXCH.	36/72 KB, 9 CH. PE CLUSTERS (1600 BPI, 2, 3, OR 4 STATION)
L	B 7381-16	18/36 KB, 36/72 KB, NRZ/PE, 9 HC., DUAL MTC WITH 2X8 EXCH.	18/36 KB, 9 CH. NRZ/PE CLUSTERS (800/1600 BPI, 2, 3, OR 4 STATION; 36/72 KB, 9 CH. NRZ/PE CLUSTERS (800/1600 BPI, 2, 3, OR 4 STATION)
L	B 7391-3	72 KC, 200/556/800 BPI, 7 CH MTC	18/50/72 KC, 7 CH. (200/556/800 BPI) MTU

Table IV-1-1. PCI/PCC Channels (Cont'd)

Size	Peripheral Controls Model	Type	Peripheral Units
L	B 7391-4	96 KC, 200/556/800 BPI, 7 CH MTC	24/66/96 KC, 7 CH. (200/556/800 BPI) MTU
L	B 7393-1	72 KB, 800 BPI, 9 CH. MTC	72 KB, 9 CH., 800 BPI MTU
L	B 7393-2	144/240 KB, 1600 BPI, 9 CH. MTC	144 KB, 9 CH., 1600 BPI MTU
L	B 7393-3	96 KB, 800 BPI, 9 CH. MTC	96 KB, 9 CH., 800 BPI MTU
L	B 7393-5	320/400 KB, 1600 BPI, 9 CH. MTC	240 KB, 9 CH., 1600 BPI, MTU
L		SINGLE LINE CONTROL-1 (SLC-1)	BURROUGHS TERMINAL COMPUTER TC500; BURROUGHS INPUT AND DISPLAY TERMINAL B 9352
L		SINGLE LINE CONTROL-2 (SLC-2)	BURROUGHS TERMINAL COMPUTER TC500; INPUT AND DISPLAY TERMINAL B 9352; INPUT AND DISPLAY SYSTEM B 9351
S	B 7110	CARD READER CONTROL	800 AND 1400 CPM CARD READERS
S	B 7212 AND B 7610	CARD PUNCH CONTROL AND BCL-BCL CODE TRANSLATOR	300 CPM CARD PUNCH

DISK FILE INTERFACE (DFI)

The DFI of a single IOM also consists of either one or two interface sections, dependent upon user requirements. Each section has an interface capability of four channels, for a total disk-file-channel capability of eight channels per IOM.

Each DFI four-channel section can service a single DFI/PCC cabinet. This cabinet contains only large channels (four maximum), which are dedicated to either disk files or disk packs. The channels may be connected to the peripherals either directly or via exchanges. In the typical configuration illustrated in figure IV-1-3, the use of two maximum DFI-configuration IOMs (eight channels per IOM, four each disk file and disk pack) and appropriate exchanges (2X20 for disk file, 2X16 for disk pack) allows access by ei-

ther IOM of 80 disk file electronics units (400 disk file storage units) and 64 disk packs. Table IV-1-2 lists the controllers which may be housed in the DFI/PCC cabinet.

SCAN INTERFACE (SCI)

The SCI consists of two sections: a DFO scan interface and a DCP scan interface. The DFO scan interface provides scan-in and scan-out control for up to four DFO's via a scan bus. If a second IOM is used, the DFO scan bus is shared by the two IOM's (see figure IV-1-3).

The DCP scan interface provides scan-out control only, and may communicate with up to four DCP's via a scan bus. The SCI is not used for DCP scan-in functions, which are initiated by the DCP. For these functions, the DCP communicates with main memory directly via the MIU. The DCP scan bus is not shared by a second IOM.

DATA COMMUNICATIONS PROCESSOR INTERFACE (DCI)

The DCI provides the data and control interface for IOM-initiated scan-out operations, and the data interface only for DCP-initiated scan-in operations. Interface is provided in each IOM for up to four DCP's. As illustrated in figure IV-1-3, the use of two IOM's in a system allows interface with eight DCP's.

IOM/Main and IOM/CPM Interface Configurations

Figure IV-1-4 illustrates a typical interface configuration between the IOM's, MCM's, and CPM's of a typical system. The following is a brief

Table IV-1-2. DFI/PCC Channels

Model	Peripheral Controls Type	Peripheral Units
B 7877	Disk File Control IV	IC-3 and IC-4 Disk Files
B 7380-1	Single Dual-Drive Control, Disk Pack	Single Data Access, Dual Drive Disk Packs, 121 Megabyte
B 7380-2	Dual Dual-Drive Control, Disk Pack	Simultaneous Data Access, Dual Drive Disk Packs, 121 Megabyte
B 7383-1	Single Dual-Drive Control, Disk Pack	Single Data Access, Dual Drive Disk Packs, 242 Megabyte
B 7383-2	Dual Dual-Drive Control, Disk Pack	Simultaneous Data Access, Dual Drive Disk Packs, 242 Megabyte

description of the interface capability of the IOM/MIU subsection with main memory and the IOM/XLATOR subsection with system CPM's.

IOM/MCM INTERFACE

As illustrated in figure IV-1-4, the MIU contains eight interface areas. Each interface area is dedicated to a distinct Memory Control Module (MCM), and is connected to it via a unique memory bus. The bussed IOM/MCM interface is referred to as a memory/user pair.

A similar capability exists within the Central Processing Module (CPM), which also contains eight MCM interface areas. Each CPM interface area is dedicated to a distinct MCM and is connected to it via a unique memory bus. The bussed CPM/MCM interface is also referred to as a memory/user pair.

The interface capability of an MCM is eight memory busses, each of which is connected to one and only one IOM or CPM. Therefore, the maximum combined number of CPM's and IOM's which may be bussed to an MCM is limited to eight.

The maximum number of MCMs which may be contained in a B 7700 system is also limited to eight. This limitation is imposed by the eight MCM-dedicated interface areas of each IOM and CPM in the system.

The typical memory-bus configuration illustrated in figure IV-1-4 indicates the use of two IOM's, two CPM's, and two MCM's. This configuration provides a total of eight memory/user pairs (MCM 0 to users 0 thru 3 and MCM1 to users 0 thru 3). The maximum number of Memory Storage Units (MSU) with which an MCM can communicate (four) is also illustrated. Each of these MCM's can access 262,144 words of memory (4 MSU's of 65,536 words each). Each IOM or CPM, when connected as illustrated on figure IV-1-4, can therefore access 524,288 words of memory.

IOM/CPM INTERFACE

The interface between the IOM's and CPM's of a B 7700 system consists only of an interrupt bus. The XLATOR section of an IOM is informed by the CPM of job requests via the bus, and the XLATOR informs the CPM of non-channel-related IOM errors via the bus. In addition, the XLATOR uses the bus to inform the CPM of (1) I/O job completion when so requested by software, and (2) status change by a single-line control device.

The interrupt bus is common to all IOM's and CPM's in a system, as is illustrated in figure IV-1-4.

IOM OPERATIONAL CHARACTERISTICS

The IOM is designed to operate asynchronously with the CPM in the initiation, service, and termination of input/output transfers by use of a "job map" which is stored in level-1 memory. The "job map" consists basically of five software-constructed elements which define the job request, the peripheral device, and the IOM channel.

In general the map elements inform the CPM of its IOM/Peripheral resources and their status. When necessary, the CPM then alters the queued job requests of the "job map" to the extent of its interest and interrupts the IOM to request service. The IOM then accesses the "job map" to determine the input/output

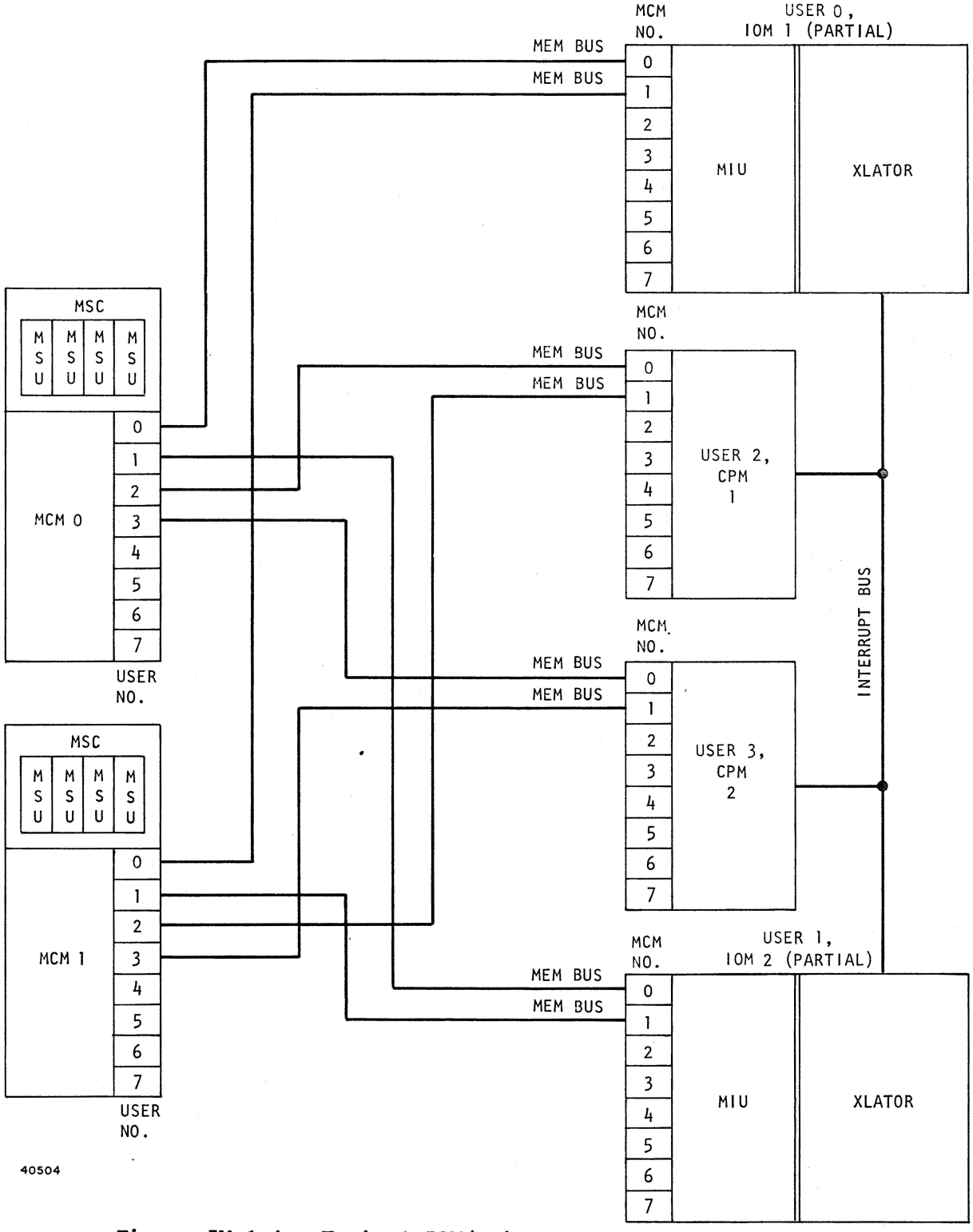


Figure IV-1-4. Typical IOM/Main Memory and IOM/CPM Interface Configurations

job and initiate it. Since the "job map" is a shared resource of the IOM and CPM, the IOM transfer times are masked by the continual processing and queueing of new requests by the CPM; thus maximum system throughput is attained with a minimum of CPM time.

The IOM also manages path selection to the requested device (instead of the programmatic preselection of the path which is generally used). This path management eliminates the occurrence of situations whereby (1) the requested device is free, (2) the preselected path is not, and (3) an alternate path exists but cannot be used due to the programmatic preselection. These situations generally require involvement of the CPM until the preselected path is free and the job is initiated, which effectively reduces the parallelism of the CPM and IOM. Since the IOM manages the path selection in the B 7700 system, CPM involvement regarding job initiation ends when an interrupt is sent to the IOM. The IOM then initiates the job request when the requested device and any path to that device is available.

The design of the IOM incorporates extensive error-detection logic which monitors the flow of control words and data between the IOM and other main-frame modules, within the IOM module itself, and between the IOM module and peripheral devices. Particular emphasis is placed upon preserving the integrity of all memory operations. In general, the error-detection hardware consists of: parity check and generate circuitry; residue check circuitry; circuitry to detect illegal commands, conditions, and control states; and timeout circuitry for memory transfers, scan bus operations, and internal IOM transfers.

IOM Job Map

The job map which the IOM's of the system access from main memory consists of the following five software-constructed elements:

a. Home Address Words (HA)

b. Unit Table Word (UT)

c. I/O Queue (IOQ)

d. I/O Control Block (IOCB)

e. Status Queue (SQ)

The following four level-1 addresses, which are loaded into the IOM XLATOR at initialize time, enable the IOM to service the job map:

a. Home Address

b. UT Base Address

c. IOQ Header (IOQH) Address

d. SQ Header (SQH) Address

By use of these stored addresses and the contents of previously-fetched map elements, job requests originally constructed by the CPM are reconstructed in the IOM and are serviced.

The following basic description of each map element and the sequence in which the job map is serviced is presented in reference to figure IV-1-5. For detailed formats of all words discussed, refer to the appendix of IOM word formats.

HOME ADDRESS WORD

The 51-bit home address word (HA word) is the first map element fetched by the IOM when interrupted by the CPU. It is fetched by use of the pre-loaded home address stored in the IOM XLATOR, and contains information which describes the basic command and, as applicable, information which describes the device or channel to be used.

The command to be performed is basically defined by a code within the HA word, called the home code; however, in some instances further definition of the command is provided by additional bits of the HA word. Based on the command decoded, the logic of the IOM is conditioned to perform one of 20 possible input/output operations. The commands are described under the heading IOM COMMANDS later in this section.

Only one of the 20 commands, the Start I/O command, requires immediate further access of other map elements; however, some scan-out commands require access of a second 51-bit HA word. The Start I/O command is the basic command used to initiate service of new job requests, whereas the remaining commands are provided for either cold start/halt load, scan out control, configuration determination, or diagnostic purposes.

A home address (HA) word which contains a home code that defines the Start I/O command also contains a unit designate number. This unit designate (UD) number defines the device to be used for the operation, and serves as a part of the information needed to access the remaining map elements.

UNIT TABLE WORD

The unit table (UT) word is the second map element fetched by the IOM upon detection of a Start I/O in the HA word. The fetch is performed by use of the UT address preloaded in the IOM XLATOR and the UD number derived from the HA word. The preloaded address serves as a locator for the unit table, and the UD number serves as an index to a particular word of the unit table.

The unit table consists of 256 words, which are numbered 0 thru 255. Word 0 is reserved for use as a fail UT word, and is accessed when an error occurs which cannot be associated with a specific job request. In this instance, a special UD number (000), called a fail UD number, serves as an index to UT word 0. Each of the remaining 255 UT words is assigned to a unique device, and contains information which defines the device and its assignment within the system.

The device-type and assignment information specifically indicates (1) whether the device is a disk-pack or a magnetic tape unit (2) if the device is a disk file, whether it is under DFO control, (3) whether the device is connected to an exchange, (4) if the device is not connected to an exchange, the IOM channel to which it is connected, and (5) if the device is connected to an exchange, the lowest-numbered IOM channel to which the exchange is connected.

When the device is connected to an exchange, the UT word assigned to it contains the following additional information for use in IOM device/path management. The devices connected to the exchange are described

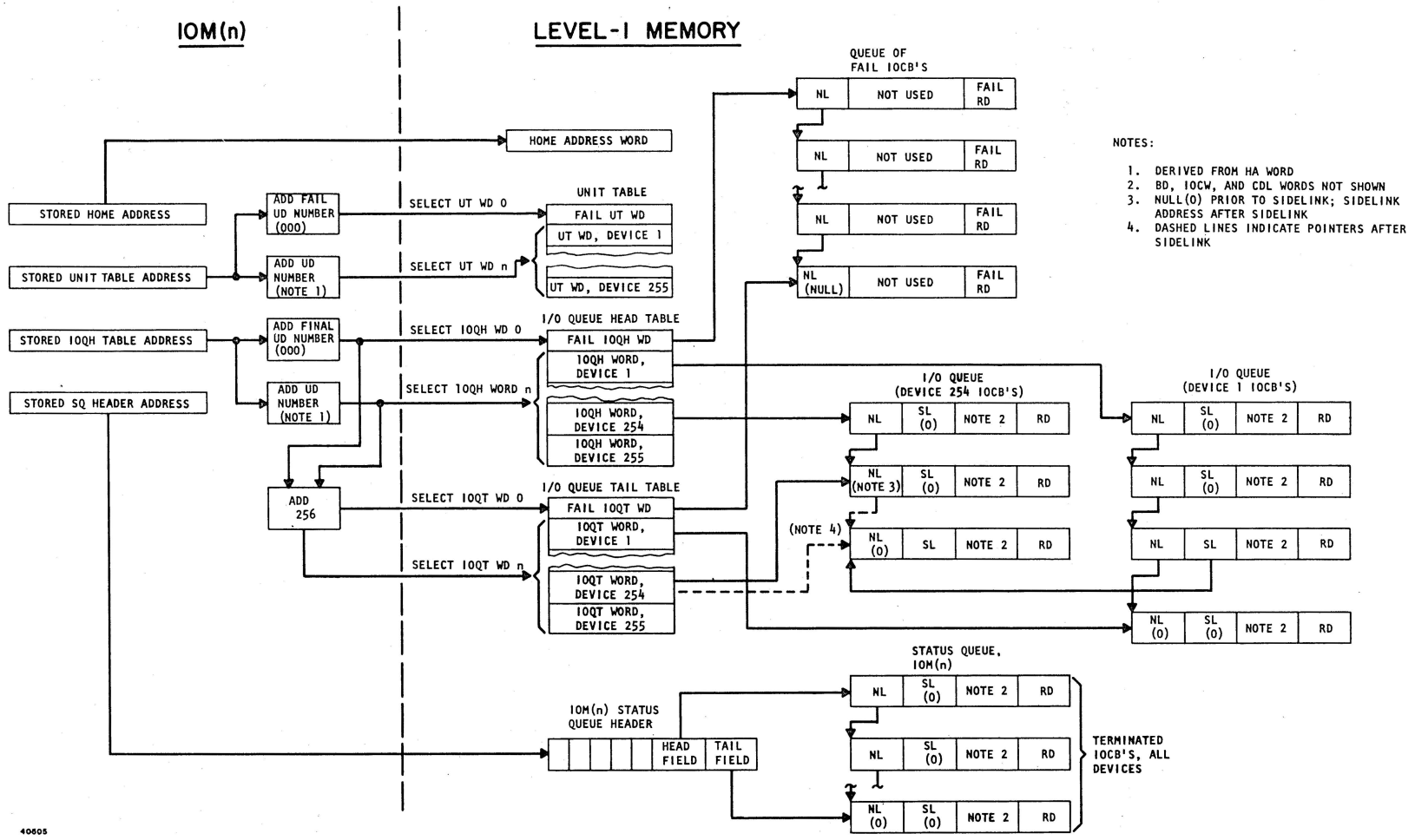


Figure IV-1-5. IOM Job Map

by use of the UD numbers of the first unit on the exchange (FUD), the next unit on the exchange (NUD), and the last unit on the exchange (LST). The number of the last IOM channel on the exchange (LCEX) is indicated, and since the specified IOM channel for an exchange device is the lowest-numbered IOM channel on the exchange, the exchange/channel configuration is effectively described. A bit (called the "job bit", or JB) is used to indicate whether device/path management is required due to the busy status of the specified IOM channel.

IOQ HEAD (IOQH) AND IOQ TAIL (IOQT) TABLES AND WORDS

The I/O Queue (IOQ), which is constructed by the CPM in main memory, contains linked job requests (I/O Control Blocks) for each device of the system. The extent of the linked job requests for each device is defined by words which indicate the main memory addresses of the first and last of the requests. These words are called the I/O Queue Head (IOQH) word and the I/O Queue Tail (IOQT) word, respectively.

The IOQH words for all devices (a maximum of 255 words) are stored in a table called the I/O Queue Head table; similarly, the IOQT words for the devices (also a maximum of 255 words) are stored in a table called the IOQT table.

The IOQH and IOQT tables also contain one additional word each (word 0) which is reserved for use by the IOM to report errors that cannot be associated with a specific job request. These words are pointers to a list of the fail I/O Control Blocks (fail IOCB's) reserved for failure-reporting by the IOM's of the system (later described).

The IOQT table is the element accessed by the CPM to queue additional requests for a device. The IOM also accesses this element when a sidelink operation to another device is specified. This access is required so that the sidelink operation indicated in the job-request queue of one device may be linked to the queue of job requests for the device designated for the sidelink operation. The IOQT word for the sidelink device is altered to reflect the main memory address of the sidelink job, which becomes the last job queued.

The IOQH table is the element accessed by the IOM in order to service job requests. The IOQH word for a device indicates the main memory address of the first job request for that device. Memory addresses of additional jobs for the device are indicated by the next link (NL) field in each job request, thus effectively linking all job requests for a given device.

As is indicated in figure IV-1-4, the last job request for a device is recognized by the IOM when the next link field of a request is found to contain zeroes (null).

The IOQH word is fetched by use of (1) the IOQH table base address (stored in the XLATOR), and (2) the UD number (derived from the previously-fetched HA word). The UD number indicates which device is to be initiated, and therefore indicates which IOQH word of the IOQH table should be fetched. The UD number is thus in essence an index to the IOQH table.

When a non-request-related error is detected by the IOM and access to the fail IOQH word (word 0) is required, the word is fetched by use of the fail UD number (000) and the IOQH base address. The memory address of the first available fail IOCB, which is contained in the fail IOQH word, is used to fetch the fail IOCB. The NL field contained in the fetched fail IOCB is then used to update the memory address of the fail IOQH word, so that if a second failure is detected, the next fail IOCB of the queue of fail IOCB's can be accessed. The fail IOQT word, which defines the last IOCB in the queue of ten fail IOCB's, is used only by software; it is not accessed by the IOM.

When a sidelink operation requires a fetch of the IOQT word for a device, 256 is added to the IOQH word address (the IOQT-word address for a device designated for a sidelink operation equals the IOQH table base address plus the UD number of the device plus 256).

I/O CONTROL BLOCKS

The job requests for each device are stored in map elements called I/O Control Blocks Each I/O Control Block (IOCB) contains words which are fetched sequentially starting with the memory address obtained from either (1) the IOQH word if the job request is the first for the device, (2) the next link (NL) field of the job request (IOCB) in process if the job is other than the first for that device, or (3) the side link field of the job request (IOCB) in process, if there are no other linked jobs queued for the device and a sidelink (SL) to another device is indicated. The six IOCB words fetched by the IOM are as follows:

a. Next Link (NL) Word

b. Side Link (SL) Word

c. Buffer Descriptor (BD) Word

d. I/O Control Word (IOCW)

e. Channel Designate Level (CDL) Word

f. Result Descriptor (RD) Word

As previously indicated, the NL word is used to indicate the address of the next IOCB for an initiated device, and is the means whereby job requests for a device are linked within the IOQ. When this word contains all zeroes (null), it indicates the request being serviced is the last for the device.

The SL word is used to indicate that a sidelink operation (the service of a job request by a device other than that presently being serviced, without intervention by the CPM) is required. The SL word contains the memory address of the sidelink job (IOCB), which is started immediately if no other jobs are queued for the designated sidelink device. If other jobs are queued for the device, the sidelink job is linked to the queue of job requests by insertion of the sidelink memory address in both the IOQT word for the sidelink device and the NL field of the last IOCB previously queued for that device.

The BD word contains the address of the first data location in memory, and the length of the memory area in words.

The IOCW contains the control information necessary to perform the input/output operation, such as read or write, whether code translation is necessary, backward/forward (tape), frame length (6 or 8 bit), etc. The contents of the IOCW and the BD word are used to format the first job word sent to the selected IOM channel.

The CDL word is used to format the second job word sent to the selected IOM channel. This word generally contains informaion such as the OP code, the device number, the device variant, and for disk, the segment address.

The RD word is used for storage of the termination status of each request. The RD word is built by the IOM, which then links the terminated request (terminated IOCB) into the status queue.

FAIL I/O CONTROL BLOCKS

A queue of ten special IOCB's, which are not related to job requests, is also built in memory. These IOCB's, which are called fail IOCB's, provide the IOM's of the system the capability of reporting errors which cannot be associated with a specific request. The fail IOCB's contain the same six words as job-request IOCB's; however, only the next link word and the result descriptor word have significance.

The result descriptor word is used for storage of a fail result descriptor. The IOM builds the fail result descriptor, inserts it in the fail IOCB RD word, and stores the fail IOCB in the status queue.

STATUS QUEUE

The Status Queue (SQ) is a queue of (1) all job-request related IOCB's which have been serviced and terminated, and (2) any fail IOCB's which have been generated by the IOM. When job-request IOCB's are terminated (or fail IOCB's are generated) and the necessary result descriptor information has been stored in the RD word of the IOCB, the IOCB is unlinked from the job IOQ (or fail IOQ) and is linked to the status queue. The linked IOCB's in the status queue represent a mix of terminated IOCB's for all devices and any fail IOCB's.

The SQ for the system consists of queues of linked IOCB's, one queue for each IOM on the system. The number of queues is dependent on the number of IOM's in the system.

The mechanism by means of which the status queue is accessed is the SQH address stored in the IOM XLATOR at initialize time. This address is unique for each IOM used, and serves as a pointer to a word in memory which defines the queue of linked IOCB's associated with a particular IOM. This word is called the status queue header (SQH) word.

When a request is terminated, the SQ address of an IOM is used to fetch the SQH word, which contains the following basic information:

a. Null (empty) state of SQ

b. Head field

c. Tail field

d. Status-Change-Vector bit

e. CPM-Interrupt bit

f. CPM Number

The null state of the SQ is checked to determine whether it contains any terminated IOCB's. If the SQ is null (empty), no linkage of the current terminated IOCB to previously-terminated IOCB's in the SQ is required. Conversely, if the SQ is not null (contains one or more IOCB's or fail IOCB's), the current terminated IOCB must be linked to the queue of terminated IOCB's in the SQ.

The head field of the SQH word contains the base address of the first terminated IOCB of the SQ. The tail field of the SQH contains the base address of the last terminated IOCB of the SQ, except when the SQ is null or contains only one terminated IOCB. If the SQ is null the tail field is not used; if the SQ contains only one IOCB, the tail field contains the same address as the head field.

A terminated IOCB is linked to previously-terminated IOCB's stored in the SQ by inserting its base address in the next link (NL) word of the terminated IOCB indicated by the SQH tail field.

The address in the tail field of the SQH is also replaced with the base address of the currently-terminated IOCB, so that link capability is present when another request is terminated.

The CPM interrupt bit determines whether an interrupt is to be sent to the CPM (indicated by the CPM number) when the terminated IOCB is linked to the SQ. For example, an error interrupt is always sent to a CPM when a fail IOCB is linked to the SQ.

The status-change-vector bit, which is used for software notification, is set only when either a SPO or a DCP has requested an input operation.

IOM HOME (HA) COMMANDS

There are 20 home commands which the IOM is directed to perform. When the IOM receives an interrupt from the CPM, it is an indication that a home command has been constructed by the CPM and placed in memory. The home address stored in the IOM is then used to fetch the HA word. A code within HA word 1, which is called the Home Code, is then decoded to determine which command or command group is to be performed.

Table IV-1-3 lists the valid home codes, and the commands and/or command groups defined by them. As indicated, scan commands are defined by the home code as only scan-in or scan-out groups; determination of type of scan-in or scan-out, and whether DFO or DCP, is defined by other portions of HA word 1. Similarly, channel busy/channel reserved commands are resolved by other portions of HA word 1. HA word 2 is not used for all commands; however, when used, it contains information to further define the command.

Table IV-1-3. IOM HA Operations and Corresponding Home Codes

Home Code	IOM Operation
0000	ILLEGAL
0001	START I/O
0010	SET CHANNEL BUSY/SET CHANNEL RESERVED

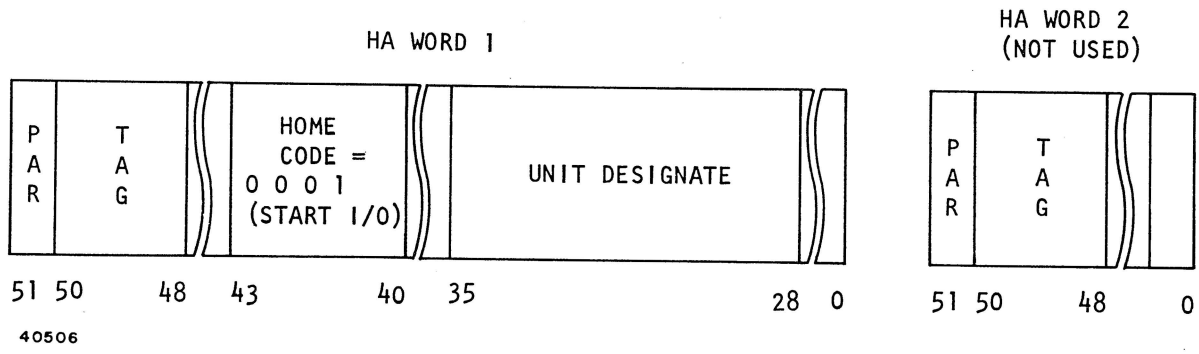


Figure IV-1-6. HA Word Content, Start I/O Command

Table IV-1-3. IOM HA Operations and Corresponding Home Codes (Cont'd)

Home Code	IOM Operation
0011	RESET CHANNEL BUSY/SET CHANNEL RESERVED
0100	LOAD HOME ADDRESS
0101	LOAD UNIT TABLE ADDRESS
0110	LOAD IOQ HEAD TABLE ADDRESS
0111	LOAD SQ HEADER ADDRESS
1000	DFO/DCP SCAN-OUT COMMANDS: DFO: CLEAR THE STACK STORE CONTROL WORD REQUEST DCP: INITIALIZE HALT SET ATTENTION
1001	DFO SCAN-IN OPERATIONS: QUEUED CONTROL WORD TOP OF STACK REPORT
1010	SYNCHRONOUS I/O
1011	INTERROGATE PERIPHERAL STATUS
1100	INHIBIT IOM
1101	ACTIVATE IOM
1110	LOAD DFO FLAGS
1111	ILLEGAL

The following brief command descriptions are presented in reference to figures IV-1-6 thru IV-1-15, which depict the basic contents of the HA words for each command. Detailed formats of the HA word for each command are presented in the appendix of IOM word formats.

Start I/O (Home Code 0001)

The Start I/O command is the basic command used to initiate input/output servicing of a new job request for a device. The device is defined by a unit designate number contained in bits 28 thru 35 of HA word 1, HA word 2 is not used. This command need only be given once in order to service all queued requests for the designated device.

Set Channel Busy/Set Channel Reserved (Home Code 0010)

Home code 0010 may represent one of two commands, dependent upon the state of bit 39 of HA word 1. If bit 39 is a "0", the Set Channel Busy command has been received; if bit 39 is a "1", the Set Channel Reserved command has been received. Both commands are for exchange

channels; the channel number is defined by bits 23 thru 27 of HA word 1. HA word 2 is not used.

The Set Channel Busy command is used primarily for diagnostic purposes. The Set Channel Reserved command is used to reserve a channel for purposes such as partitioning. Once either command has been received, the specified channel remains busy (or reserved) until a counter command is received.

Reset Channel Busy/Reset Channel Reserved (Home Code 0011)

Home code 0011 may also represent one of two commands, dependent upon the state of bit 39 of the HA word ("0" defines the Reset Channel Busy command; "1" defines the Reset Channel Reserved command). These commands are the counter commands to the Set Channel Busy/Set Channel Reserved command.

Load Address Commands

Load home address (home code 0100); load unit table address (home code 0101); load IOQ head table address (home code 0110); load SQ header address (0111).

The commands are normally used to load fixed addresses into the IOM XLATOR at initialize time; however, they may also be used to establish new base addresses at any time after initialization. The address to be loaded by each command is contained in bits 0 thru 19 of HA word 1; HA word 2 is not used.

DFO/DCP Scan-Out Commands (Home Code 1000)

Home Code 1000 specifies a scan-out command for either a DFO or a DCP; the specific device for which the scan-out command is intended, as well as the specific type of scan-out command, is defined by other bits of HA word 1.

Whether the scan-out command is for a DFO or a DCP is dependent upon bits 19 thru 16 of HA word 1. A bit configuration of 1001 indicates the command is for a DFO, whereas a bit configuration of 1100 indicates the command is for a DCP.

DFO SCAN-OUT COMMANDS

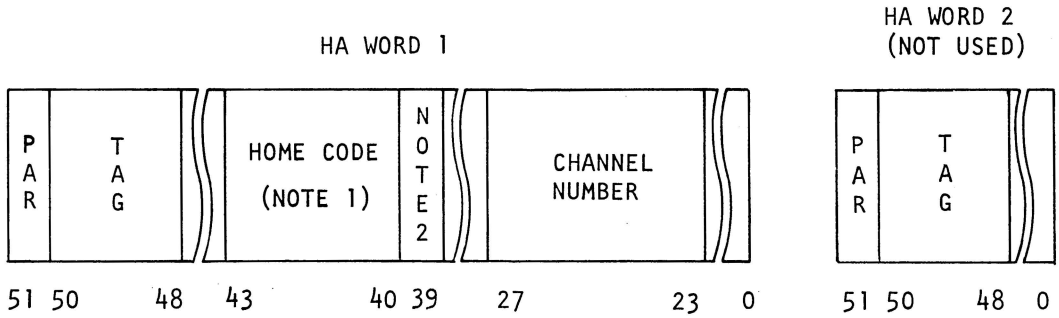
There are two specific scan-out commands for DFO's. The command type is determined by the configuration of bits 4 and 5 of HA word 1 as follows:

a. Bit 4 = 0, bit 5 = 1: Clear the Stack

b. Bit 4 = 1, bit 5 = 0: Store Control Word Request

The Clear the Stack Command is used to erase the DFO queuer stack and to reset any DFO error flip-flop previously set. The unit number of a Disk File Electronics Unit (DFEU) is contained in bits 8 thru 15 of HA word 1. This unit number is used with bit 7 (the exchange select bit) to define either the DFO with which the DFEU is directly connected or the DFO with which the DFEU is indirectly connected. HA word 2 is not used for the Clear the Stack command.

The Store Control Word Request command is used to request storage of a control word in the DFO queuer stack. The DFO is defined by bit 7 and bits 8 thru 15 of HA word 1, as described for the Clear the Stack command. A fetch of HA word 2 is



NOTES:

1. HOME CODE 0010 DEFINES SET CH BSY/SET CH RESERVED COMMANDS; HOME CODE 0011 DEFINES RESET CH BSY/RESET CH RESERVED COMMANDS.
2. "0" DEFINES SET/RESET CH BSY; "1" DEFINES SET/RESET CH RESERVED

40507

Figure IV-1-7. HA Word Content, Set/Reset Channel Busy/Channel Reserved Commands

required to implement this command. Bits 28 thru 47 of HA word 2 contain the base address of the job request in memory, and bits 0 thru 25 contain the disk address to be used.

DCP Scan-Out Commands

There are three specific scan-out commands for the DCP; the command type is determined by bits 5, 6, and 7 of HA word 1 as follows:

a. Bits 5, 6, and 7 = 0: Initialize

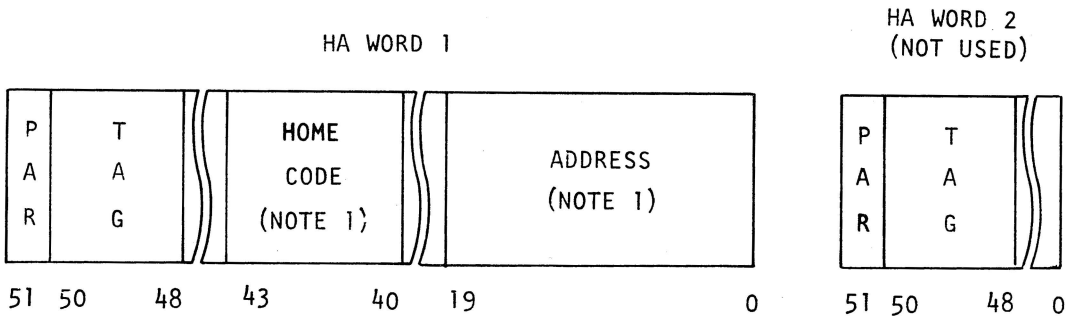
b. Bits 5 = 0, bit 6 = 1, bit 7 = 0: Halt

c. Bits 5 and 6 = 0, bit 7 = 1: Set Attention

The DCP for which the command is intended is indicated by a DCP number contained in bits 1 thru 3 of HA word 1.

The Initialize command requires access by the IOM of HA word 2, which contains an instruction base address (bits 0 thru 19). HA word 2 is not used for the Halt and Set Attention commands.

The Initialize and Halt commands cause psuedo fault interrupts to occur within the DCP. In the case of the Initialize command, the interrupt causes the 20-bit instruction base address to be loaded into the DCP scratch-pad memory. The interrupt generated by the Halt command stops DCP operations. In either



NOTES: 1. HOME CODE 0100 = LOAD HOME ADDRESS;

HOME CODE 0101 = LOAD UNIT TABLE ADDRESS;

HOME CODE 0110 = LOAD IOQ HEAD TABLE ADDRESS;

HOME CODE 0111 = LOAD SQ HEADER ADDRESS

40508

Figure IV-1-8. HA Word Content, Load Address Commands (HA, UT, IOQH, SQH)

case, stop actions which would normally occur within the DCP due to fault interrupt occurrence are inhibited.

The Set Attention command is used to notify the DCP that attention to the B 7700 system is required.

DFO Scan-In Commands (Home Code 1001)

Home code 1001 specifies that the command to be performed is one of three DFO scan-in commands. The specific command is defined by the configuration of bits 4 and 5 of HA word 1 as follows:

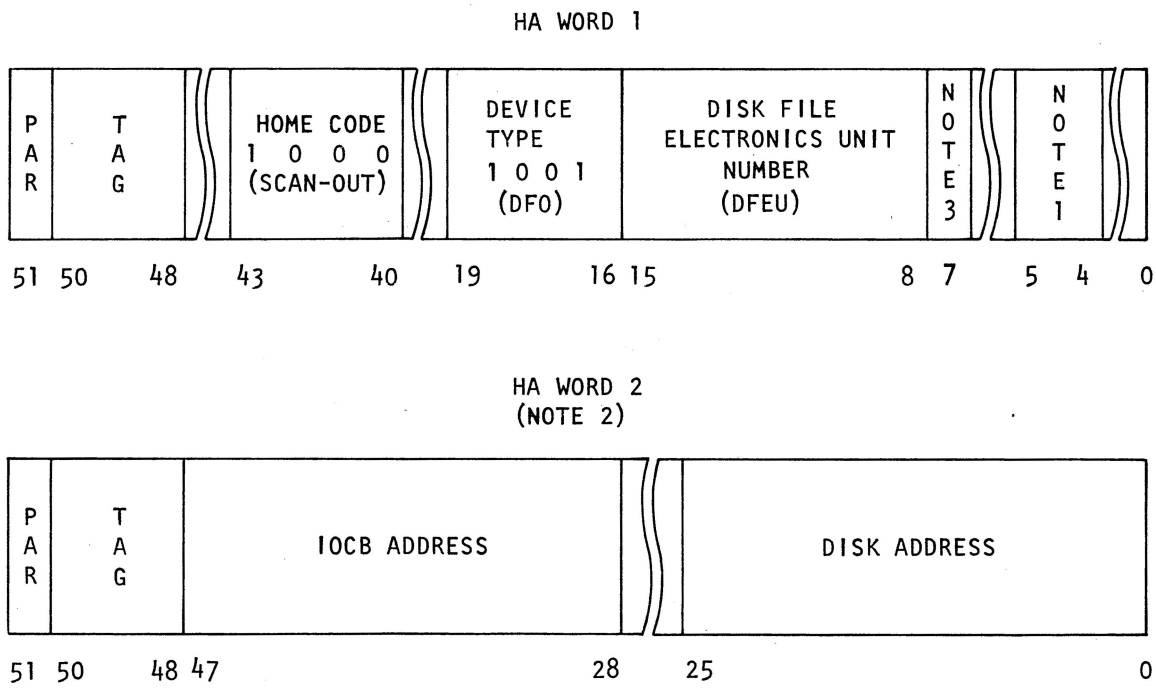
a. Bit 4 = 1, bit 5 = 0: Queued Control Word

b. Bit 4 = 0, bit 5 = 1: Top of Stack

c. Bit 4 = 1, bit 5 = 1: Report

The desired DFO is addressed by use of a DFEU number and exchange-select bit as in DFO scan-out commands. This information however, is not used for the Report commands.

HA word 2 is not accessed to further define command parameters. However, it is later used for storage of the scan-in word received via the scan information lines, and is the mechanism by means of which the CPM is notified of the location and status of the scan data.



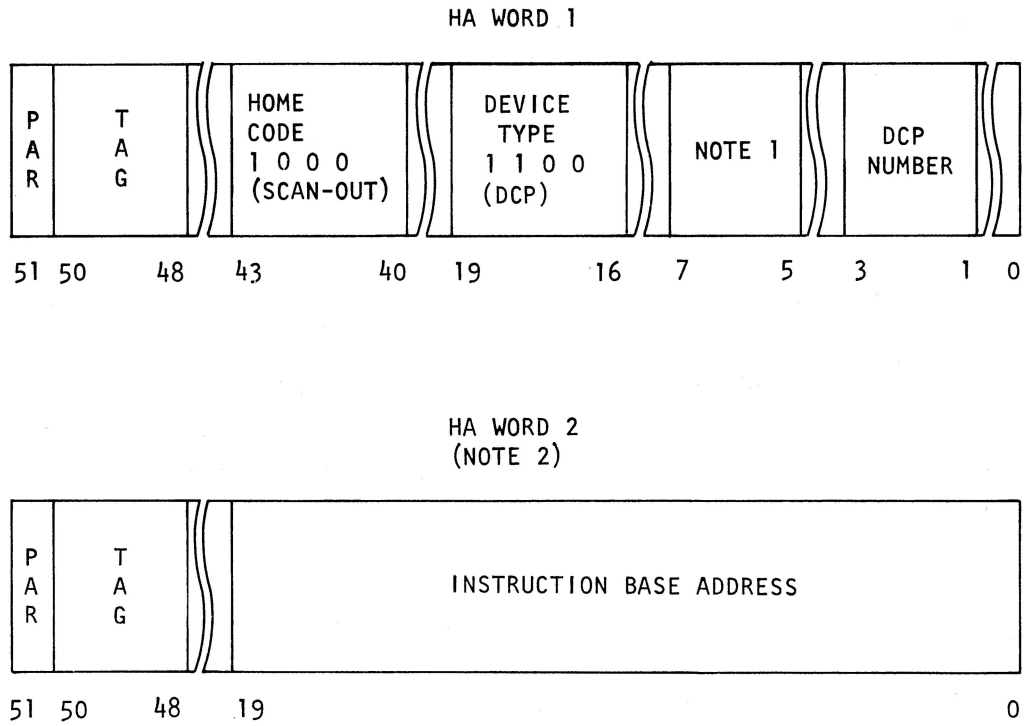
NOTES: 1. 10 = CLEAR THE STACK COMMAND;
01 = STORE CONTROL WORD REQUEST COMMAND.

2. APPLICABLE TO STORE CONTROL WORD REQUEST COMMAND ONLY; HA WORD 2 NOT USED FOR CLEAR THE STACK COMMAND.

3. EXCHANGE SELECT BIT: IF = 0, SELECT DFO DIRECTLY CONNECTED TO DFEU; IF 1, SELECT DFO INDIRECTLY CONNECTED TO DFEU.

40509

Figure IV-1-9. HA Word Content, DFO Scan-Out Commands (Clear The Stack, Store Control Word Request)

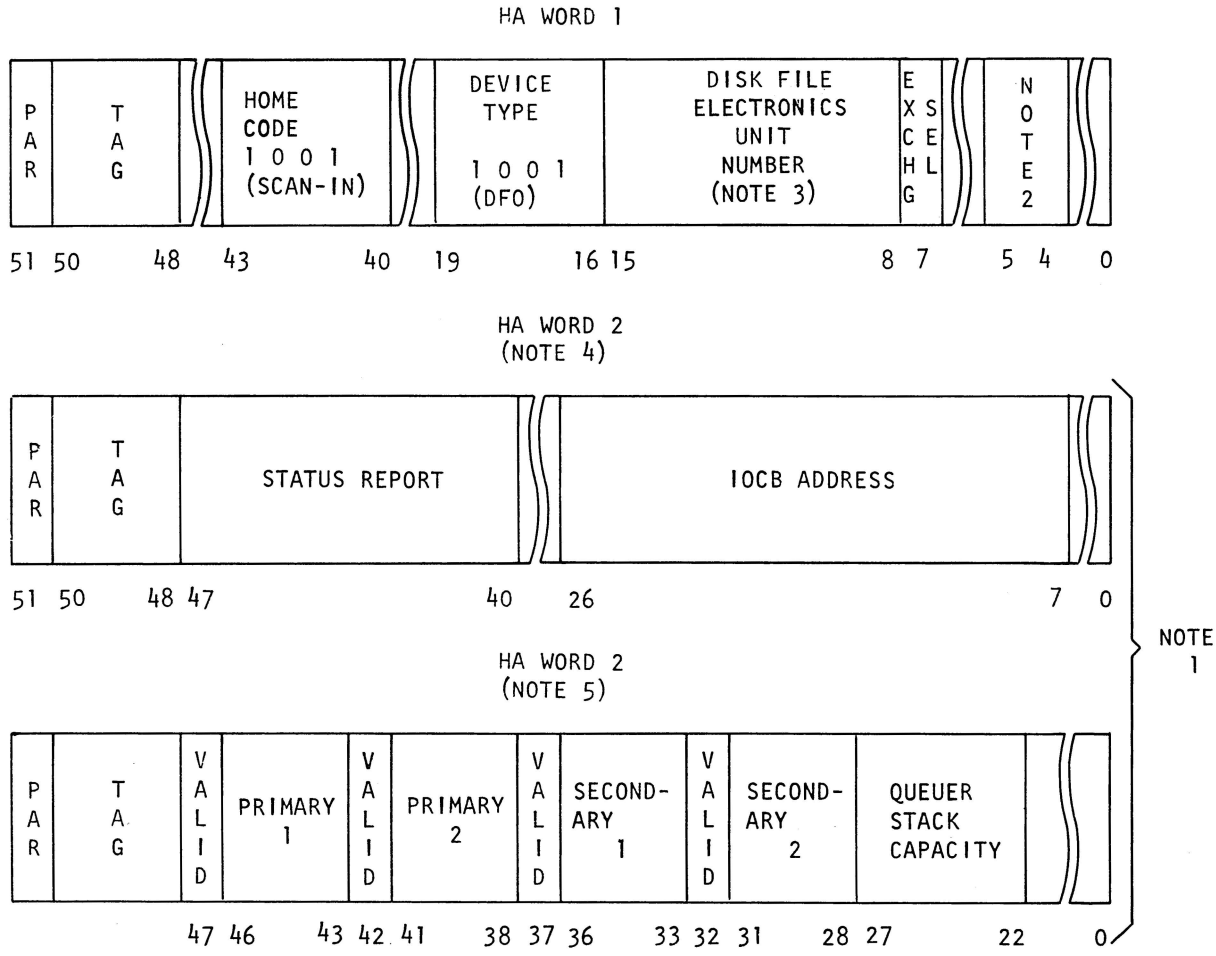


NOTES: 1. 000 = INITIALIZE COMMAND;
010 = HALT COMMAND;
100 = SET ATTENTION COMMAND.

2. APPLICABLE TO INITIALIZE COMMAND ONLY;
HA WORD 2 NOT USED FOR HALT
AND SET ATTENTION COMMANDS.

40510

Figure IV-1-10. HA Word Content, DCP Scan-Out Commands (Initialize, Halt, Set Attention)



NOTES:
1. HA WORD 2 USED ONLY FOR STORAGE OF RETURNED INFORMATION.
2. 01 = QUEUED CW COMMAND;
   10 = TOP OF STACK COMMAND;
   11 = REPORT COMMAND.
3. NOT APPLICABLE TO REPORT COMMAND.
4. APPLICABLE TO QUEUED CW AND TOP OF STACK COMMANDS ONLY.
5. APPLICABLE TO REPORT COMMAND ONLY.

40511

Figure IV-1-11. HA Word Content, DFO Scan-In Commands (Queued Control Word, Top of Stack, Report)

The Queued Control Word command is used to request an optimized control word from the queuer stack of the DFO. The Top of Stack command is used to request DFO transmission of the control word located at the top of the DFO queuer stack.

The report command is used to request a report from the DFO, which defines the DFEU's connected to all DFO primary and secondary parts. The report is stored in HA word 2. Four fields in the report stored in HA word 2 are used to define the DFEU's which are connected to the four DFO ports (two primary and two secondary). Each field consists of (1) a bit (valid) which indicates whether an EU/DFO bus is connected to the port, and therefore whether the field is valid, and (2) four bits which represent the most significant bits of the unit number of the lowest-numbered DFEU connected to the port. A maximum of 40 DFEU's may be connected to a DFO (20 direct and 20 indirect). Since the DFO provides four ports, a maximum of 10 DFEU's are connected to a port, and the four bits are sufficient to define them.

The report returned and stored in HA word 2 also contains information which defines the capacity of the DFO queuer stack. The information is reported in bits 22 thru 27 of HA word 2.

Synchronous I/O Command (Home Code 1010)

The Synchronous I/O command provides a means of servicing a single job request during initialization. Only HA word 1 of the job map, which contains the IOCB base address (bits 0 thru 19), is accessed; no queue mechanisms are used. When the single job request is terminated, the result descriptor information is stored in HA word 5, and a channel interrupt is sent to the CPM.

Interrogate Peripheral Status Command (Home Code 1011)

The Interrogate Peripheral Status command is used to determine the ready status of all devices assigned to a particular status vector. The status vector to be interrogated is indicated by bits 9 thru 12 of HA word 1.

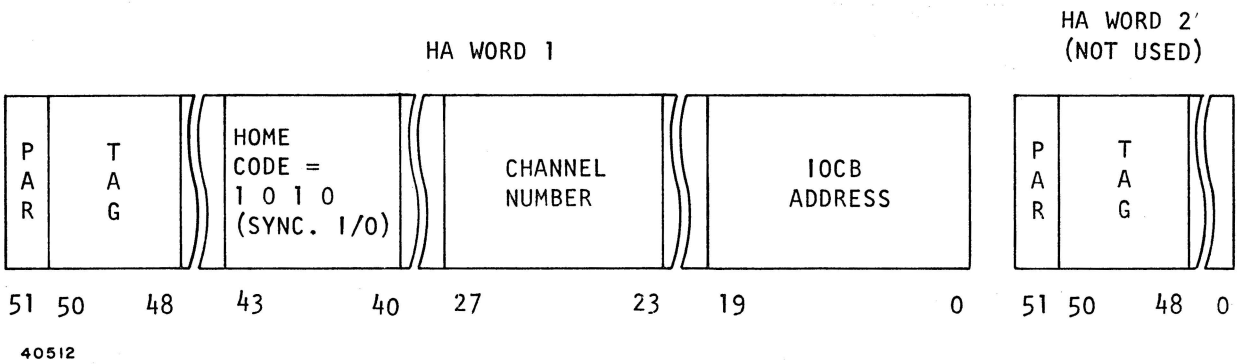


Figure IV-1-12. HA Word Content, Synchronous I/O Command

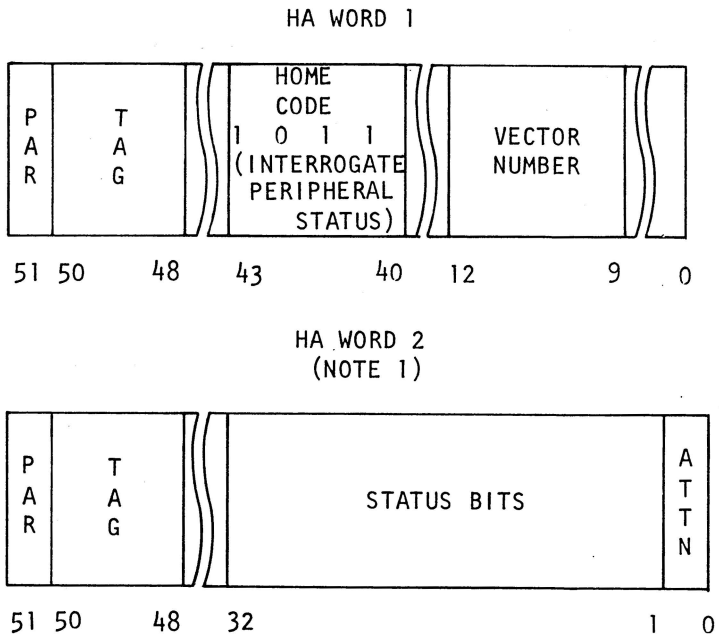


Figure IV-1-13. HA Word Content, Interrogate Peripheral Status Command

HA word 2 is not accessed for command determination, but is later used for storage of the returned status information. The status information, which is returned in bits 1 thru 32 of HA word 2, provides indication of the ready status of up to 32 devices on a vector. Bit 0 of HA word 2 (ATTN) notifies the CPM that the status word has been returned.

Inhibit IOM Command (Home Code 1100)

The Inhibit IOM command is used to inhibit all automatic IOM functions, such as data-path management, DFO scan-in and scan-out functions, and the servicing of linked and side-linked job requests. If linked job requests for a device are being serviced when the command is received, all links are completed prior to IOM response to the command.

The content of HA word 1 consists only of the home code; HA word 2 is not accessed.

Activate IOM Command (Home Code 1101)

The Activate IOM command is used to restore automatic functions of the IOM after the Inhibit IOM command has been given. The command consists only of the home code in HA word 1; HA word 2 is not used.

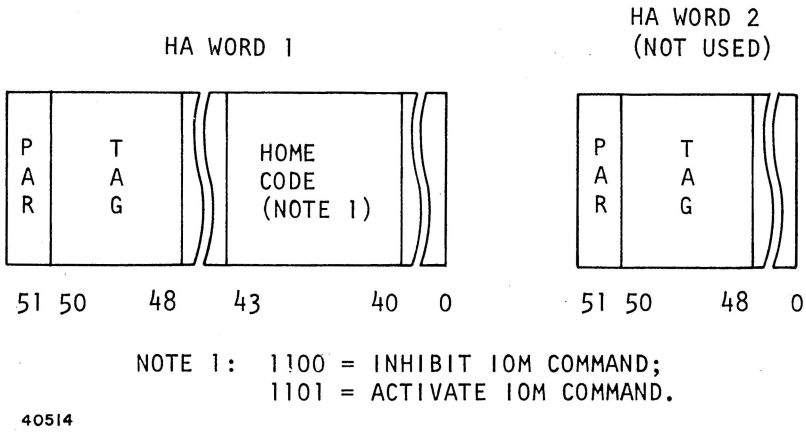


Figure IV-1-14. HA Word Content, Inhibit IOM, Activate IOM Commands

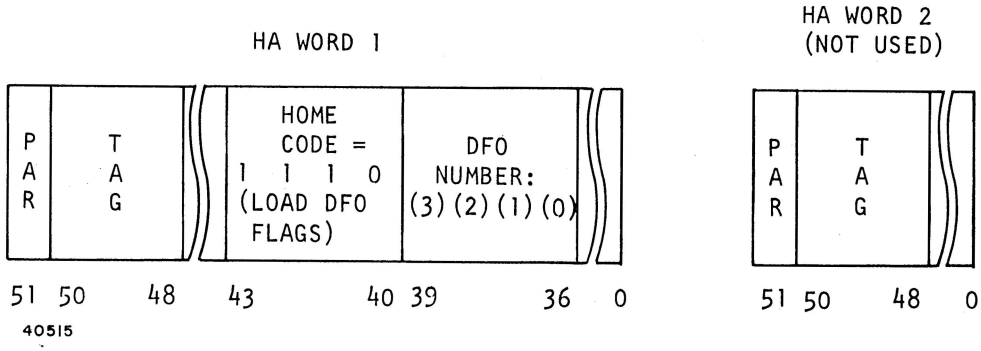


Figure IV-1-15. HA Word Content, Load DFO Flags Command

Load DFO Flags Command (Home Code 1110)

The Load DFO Flags command is used to mask out one or more DFO's connected to an IOM. The HA word in which the Load DFO Flags command is received contains four flag bits, one for each DFO which may be connected to an IOM. These bits (36 thru 39) are referred to as the DFO ON/OFF flags for DFO numbers 0 thru 3, respectively. When an OFF flag (0) is detected, scan-in and scan-out operations with the associated DFO are inhibited.

AUTOMATIC SERVICE OF DISK JOBS FOR UNITS UNDER DFO CONTROL

In B 7700 disk file subsystems where the disk jobs are not optimized, the service of multiple job requests for disk units on a common exchange involves an inherent delay between service of each job request. This

delay is partially due to the manner in which the jobs must be linked under the queue of IOCB's for each Disk File Electronics Unit (EU); that is, without regard to the relationship of the disk starting address specified by each job request and the current disk position (relative to the head), since the current disk position is unknown.

In B 7700 disk file subsystems where Disk File Optimizers are used, the inherent delay between the service of multiple job requests is reduced. The job requests are linked under the queues of IOCB's for the DFO's, rather than under the queues of IOCB's for the EU's as in a non-optimized system. Upon receipt of a Start I/O HA command, the UT word is fetched. If the DFO bit is set, the job requests are automatically scanned out to the DFO job stack when possible. The DFO's constantly monitor the disk addresses specified by the job requests in the stack and compare them with the current disk position relative to the head. This information is used to maintain a job-stack pointer which indicates the current optimum job request relative to disk/head position. This current optimum job request is referred to as a queued control word.

The DFO's communicate with the IOM via a common scan bus and individual status lines. The status lines transfer information regarding the capability of the individual DFO's to receive job requests from the IOM over the scan bus. In addition, the status lines transfer levels which indicate the availability of queued control words which require service. The SCI section of the IOM scans these status lines to determine whether queued control words are available from any DFO. If the status lines of any DFO indicate the availability of queued control words, the SCI section of the IOM determines whether a disk channel is available on the exchange to which the DFO is connected. If so, a scan address word is formatted by the Xlator and SCI sections of the IOM, and is then sent over the scan bus to all DFO's. The contents of the scan address word sent over the scan bus identify the exchange and the DFO on that exchange which has indicated availability of queued control words. The scan address word on the scan bus is recognized only by the identified DFO, and therefore is, in essence, an acknowledge to that DFO.

In response to the scan address word received, the identified DFO transfers a scan information word over the scan bus to the IOM. This word contains a complete memory link address, which is used by the IOM to further access the IOM job map for the identified job. The map access performed provides information which identifies the EU which is to control the disk job, whether that EU is available, and whether the previously-available disk channel is still available. If all conditions are met, the job is initiated and data are transferred between the DFI section of the IOM and the specified EU. Upon completion of the data transfer, the disk job is terminated in the normal manner. If the identified EU is not available or if the disk channel is not available, the disk job is relinked under the queue of IOCB's for the DFO. It is then later transferred again to that DFO for reoptimizing and another attempt at job initialization.

AUTOMATIC DISK-PACK OPERATION

The IOM has provisions to automatically re-initiate a type 225 Disk Pack Unit after the unit has completed a seek operation. This type of unit, when issued a conditional I/O command requiring head positioning (seek), must be issued the same command after the seek has been completed in order to accomplish data transfer. The IOM performs this function by examining all device result descriptors received from Disk Pack units.

CODE TRANSLATIONS

The IOM has the capability of three types of code translations: BCL, EBCDIC, and ASCII. These codes are device dependent and are performed on an IO operation as determined by the standard control bits in the IOCW. The IOCW bits used to specify code translations are:

Bit 47 - ASCII-always on for any translation having ASCII input or output.

Bit 44 - READ - (READ = 1, WRITE = 0)

Bit 42 - TRANSLATE

Bit 41 - FRAME LENGTH (8 bit characters = 1, 6 bit characters = 0)

The binary combinations of these code bits are listed in detail in table IV-1-4 and the list of specific device-related code translations is shown in detail in table IV-1-5.

Table IV-1-4. General Translation Specification Codes

R 44	T 42	FL 41	A 47	TRANSLATION
0	0	0	0	No translation
0	0	0	1	(Illegal Code)
0	0	1	0	No Translation
0	0	1	1	EBCDIC > ASCII
0	1	0	0	BCL Internal > BCL External
0	1	0	1	ASCII > BCL External (See Note 1)
0	1	1	0	EBCDIC > BCL External
0	1	1	1	ASCII > EBCDIC
1	0	0	0	No Translation

Table IV-1-4. General Translation Specification Codes (Cont'd)

R 44	T 42	FL 41	A 47	TRANSLATION
1	0	0	1	(Illegal Code)
1	0	1	0	No Translation
1	0	1	1	EBCDIC > ASCII
1	1	0	0	BCL External > BCL Internal
1	1	0	1	BCL External > ASCII (See Note 1)
1	1	1	0	BCL External > EBCDIC
1	1	1	1	ASCII > EBCDIC

Note 1: In these combinations the frame length bit should be "1". However, due to encoding considerations, it is necessary to use this code and alter the FL decode to cover these cases.

Table IV-1-5. Device Related Code Translations

DEVICE	CODE SPECIFIER R 44	T 42	FL 41	A 47	DESCRIPTION
CARD READER:	1	0	0	0	No translation, data in binary
	1	0	1	0	No translation, data in EBCDIC
	1	0	1	1	EBCDIC > ASCII
	1	1	0	0	BCL External > BCL Internal
	1	1	0	1	BCL External > ASCII
	1	1	1	0	BCL External > EBCDIC
CARD PUNCH:	0	0	0	0	No translation, data in binary
	0	0	1	0	No translation, data in EBCDIC
	0	1	0	0	BCL Internal > BCL External

Table IV-1-5. Device Related Code Translations (Cont'd)

DEVICE	CODE SPECIFIER R 44	T 42	FL 41	A 47	DESCRIPTION
	0	1	0	1	ASCII > BCL External
	0	1	1	0	EBCDIC > BCL External
	0	1	1	1	ASCII > EBCDIC
LINE PRINTER	0	0	0	0	No translation, data in BCL External
	0	1	0	0	BCL Internal > BCL External
	0	1	0	1	ASCII > BCL External
	0	1	1	0	EBCDIC > BCL External
P.T. READER:	1	0	0	0	No translation, data in binary
	1	0	1	0	No translation, data in EBCDIC/ASCII
	1	0	1	1	EBCDIC > ASCII
	1	1	0	0	BCL External > BCL Internal
	1	1	0	1	BCL External > ASCII
	1	1	1	0	BCL External > EBCDIC
	1	1	1	1	ASCII > EBCDIC
P.T. PUNCH:	0	0	0	0	No translation, data in binary
	0	1	0	0	BCL Internal > BCL External
	0	1	0	1	ASCII > BCL External
	0	1	1	0	EBCDIC > BCL External
	0	0	1	1	EBCDIC > ASCII
	0	1	1	1	ASCII > EBCDIC
7 TR. TAPE:	0	0	0	0	No translation, data in binary

Table IV-1-5. Device Related Code Translations (Cont'd)

DEVICE	CODE SPECIFIER R 44	T 42	FL 41	A 47	DESCRIPTION
	0	1	0	0	BCL Internal > BCL External
	0	1	0	1	ASCII > BCL External
	0	1	1	0	EBCDIC > BCL External
	1	0	0	0	No translation, data in binary
	1	1	0	0	BCL External > BCL Internal
9 TR. TAPE:	0	0	0	0	No translation, data in binary (see note 1).
	0	0	1	0	No translation, data in EBCDIC/ASCII
	0	0	1	1	EBCDIC > ASCII
	0	1	0	0	BCL Internal > BCL External (see note 1).
	0	1	0	1	ASCII BCL External (see note 1).
	0	1	1	0	EBCDIC > BCL External (See note 1).
	0	1	1	1	ASCII > EBCDIC
	1	0	1	0	No translation, data in EBCDIC/ ASCII
	1	1	0	0	BCL External > BCL Internal (See Note 1).
	1	1	1	1	EBCDIC > ASCII
	1	1	0	1	BCL External > ASCII (See note 1).
	1	0	1	1	ASCII > EBCDIC
	1	1	1	0	BCL External > EBCDIC (See note 1).

Table IV-1-5. Device Related Code Translations (Cont'd)

DEVICE	CODE SPECIFIER R 44	T 42	FL 41	A 47	DESCRIPTION
DISK:	0	0	1	0	No translation data in EBCDIC/ ASCII
	1	0	1	0	No translation, data in EBCDIC/ ASCII
SLC:	0	0	1	0	No translation, data in EBCDIC/ ASCII
	1	0	1	0	No translation, data in EBCDIC/ ASCII

Note 1: The hardware translation is possible, but the translation is inhibited by software.

EBCDIC-BCL Exceptions

Bi-directional translation of corresponding EBCDIC graphics to/from corresponding BCL graphics are provided with the following exceptions:

a. EBCDIC to BCL (output translator)

EBCDIC	BCL
PZ	+
MZ	x (times)
Corresponding graphics	Corresponding graphic
Non-corresponding graphics	? (See Note)

NOTE: The following graphics are printed dependent upon whether the printer is equipped for EBCDIC or BCL:

GRAPHIC PRINTED PER EBCDIC CODE				
0111 1101 0101 0110 1101 0100 1111				
EBCDIC Printer	(Apostrophe)	¬ (Logical not)	— (Underscore)	| (Vertical bar)
BCL Printer	≥	≥	≠	→

b. BCL to EBCDIC translator (input translator)

BCL	EBCDIC
X (times)	MZ
Corresponding graphics	Corresponding graphics

IOM-GENERATED INTERRUPTS

The IOM generates the following two interrupts and sends them over individual lines to each central processor:

1. Channel Interrupt
2. IOM Error Interrupt

There are three conditions under which the IOM generates an interrupt to a CPM:

a. IO Complete - During error free completion, an IO complete is generated when bit 40 of the IOCB NL word is set or when bit 40 of the Status Queue Header is set. These bits are set by software and are reset by the IOM after the interrupt is generated.

   Unless requested by software, an interrupt is not set for "exception conditions" (peripheral parity error, end of tape, etc.). The only action taken for an "exception condition" is that the next request job, if there is more than one request job queued, is not started and bit 0 of the UT word is set.

b. Status Change Single-Line Control - Reported when hardware line "Inquiry request" changes from "off" to "on" state. This generates an IO complete.

   DCP interrupts, through which the DCP alerts a Central Processor that attention is required, are directed through the IOMs. They are merged with the Channel Interrupt from the IOM, and the definition of which DCP interrupted is contained in the high order portion of the status change vector. The IOM sets bit 45 in the Status Queue Header to request software to read the "Status Change Vector" (Vector 8) via a scan-in command. Bit 45 is reset by software.

c. IOM Errors - These errors are not related to any channel or request (for example, a memory parity error on the Home Address word). The generated fail register word is placed as a result descriptor using a dummy IOCB from unit 0.

SECTION 2
FUNCTIONAL OPERATION OF INPUT/OUTPUT MODULE SUBSECTIONS

GENERAL

This section contains a brief description of the operation of each of the IOM subsections described in section 1 of this chapter. For the formats of the words discussed, refer to the appendix of IOM word formats in this manual.

FUNCTIONAL OPERATION OF TRANSLATOR

The translator (figure IV-2-1) is a special-purpose processor capable of performing specific hardwired microsequences. It is the mechanism of the IOM that services I/O requests, generates the request descriptors required to initiate pe-

ripheral devices, and reports job termination and failure status conditions to the Central Processor. The operation of the translator is keyed to respond to certain declared flag conditions.

Job Service Initiation

In response to an interrupt from the Central Processor, the IOM unlock-fetches the word in memory referenced by the 20-bit Home Address stored in the HA location of the lower stack. The HA-word control fields define the control codes and function details for the request as described in Section 1 of this chapter. When the Start I/O command is decoded, the Unit Designate (UD) field of the Home Address word is loaded into the UD register (see figure IV-2-1).

The UD field is added to the 20-bit Unit Table (UT) base address (stored in the UT location of the lower stack) to address and lock fetch from memory (write with flashback) the Unit Table word for the device to be started. (If the UT is still locked after 63 memory accesses, the IOM enters fail mode.) For devices other than a DFO, the contents of the channel number identification field of the UT word are used to access the Active Channel Stack (ACS). If the device is not connected to an exchange, and if the ACS and UT word busy bits are reset, the I/O Queue Head word is fetched from memory to obtain the base address of the IO Control Block (IOCB).

Successive fetches are made of (1) the IOCB base address plus 2, to obtain the buffer descriptor, (2) the IOCB base address plus 3 to obtain the IO control word (IOCW), and (3) the IOCB base address plus 4 to obtain the Channel Designate Level (CDL) field. The buffer descriptor is comprised of two fields: base address information and buffer-length information. The 20-bit base address field is used to locate the buffer in memory. The IOCW Standard Control Field contains information useful to the data service sections such as read/write, translate, and format bits. Information contained in the buffer descriptor, SCF, (from the IOCW), CDL and channel number fields of the IOCB is sent to the data service section for starting up the selected device. The Unit Designate number is stored in the ACS and the ACS busy bit is set. The Unit Table word busy bit is set, and the UT word stored in memory is unlocked. The HA word is unlocked and the remainder of the word is set to all zeros. Control is transferred to the initial state.

If the device to be started is connected to an exchange and the busy bit of the base channel locations in the ACS is set, the translator logic selects the next channel of the exchange and checks its busy bit. If a channel is available, information is fetched from memory and is sent to the data service section, to start the selected device. If all channels of the exchange are busy, the job bit (JB) in the Unit Table word is set, and the UT word is stored unlocked in memory. Control is transferred to the initial state. These conditions are summarized in table IV-2-1.

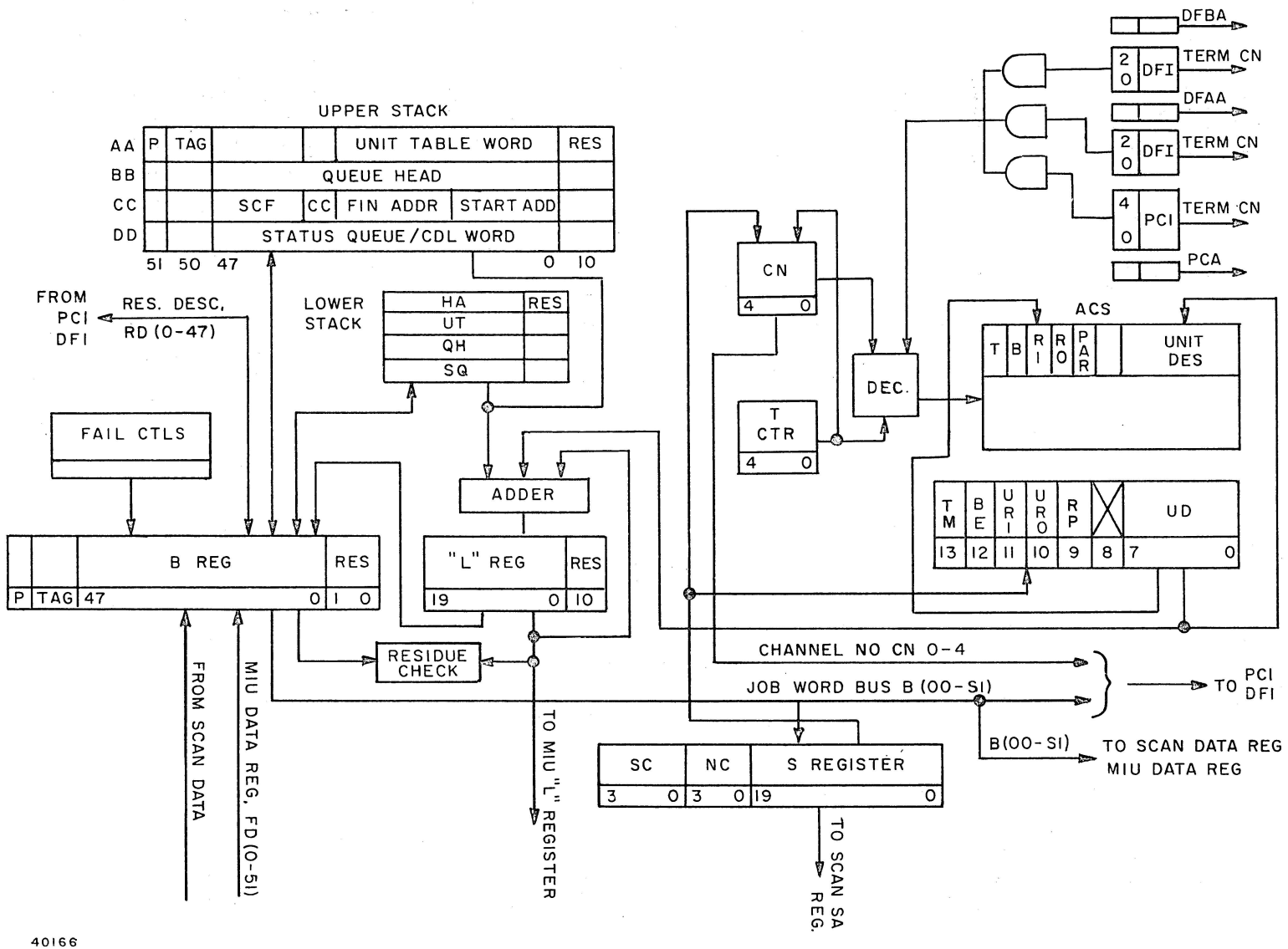


Figure IV-2-1. Translator Component Interface

Table IV-2-1. Unit Table and Active Channel Coded Decisions

Unit Table Word			A.C. Stack		
B38 EX	B37 JB	B36 BZ	CBF	CTF	
X	0	0	0	0	Start job; unlock UT; Set BZ (UT)
0	1	X	X	X	Error; Set Initiate Bus Channel Error (IBE) in Fail Word
0	X	1	X	X	
0	X	X	1	X	
0	X	X	X	1	
*1	1	0	X	1	Unlock UT; Go to initial state
*1	1	0	1	X	Unlock UT; Go to initial state
1	0	0	X	1	Unlock UT; Set JB; Go to initial state
1	0	0	1	X	Unlock UT; Set JB; Go to initial state
1	1	0	0	0	Start job; Unlock UT; Set BZ, Res JB
1	X	1	X	X	Unlock UT; Go to initial state

X = State irrelevant

EX = Exchange bit

JB = Job to be done

BZ = Job Busy

CBF = Chan Busy FF

CTF = Chan Term FF

* = Applicable only when second IOM has set JB

Job-Service Termination

When a device either completes a service or is terminated as a result of an error condition, the data service unit causes the terminate bit to be set for that channel. The terminate bits are located in the Active Channel Stack (ACS) of the translator; one bit for each of the possible 28 channels available. In response to a terminate bit being set, the translator reads the corresponding Unit Designate (UD) information from the ACS. This information is used along with the unit table base address to index and lock-fetch from level-1 memory the UT word for the terminating device. The I/O Queue Head is then fetched to obtain the base address of the I/O Control Block (IOCB). The Result Descriptor (RD) information received from the data service unit is then stored in the sixth word field of the IOCB, and the IOCB is linked to the Status Queue (SQ).

If this is the last request for this unit, the I/O Queue Head (IOQH) and I/O Queue Tail (IOQT) are nulled, and the UT word is stored unlocked, to complete the termination. If there are more requests, the address of the next IOCB is inserted in the 20-bit address field of the I/O Queue Head. Control is passed to the start section to initiate the request. If the terminating IOCB is the last request for this unit, and the unit is connected to an exchange, then control is passed to the ring-walk section and a search is made to find a request that is waiting to be initiated.

Exchange Ring Walk

The following action is taken when an exchange unit terminates and there are no more requests queued for that particular unit.

The First Unit Designate (FUD) field in the UT word of the terminating device points to the start, or first unit (device) of the exchange. The UT word for the first unit is fetched from level-1 memory and its status is checked for busy and an available request for this unit. If the unit is busy or there are no requests, then the information in the Next Unit Designate (NUD) field in the UT word is used to point to the next unit of the exchange. This process of looking for a request continues until either a request is found or the end bit is reached. When a unit is found with a request waiting, the UT word for that unit is lock-fetched from memory and the job is started.

Automatic Service of Disk Jobs Via DFO Units

IO requests for disk units under control of a Disk File Optimizer (DFO) are entered in the IO Queue for the DFO rather than the IO Queue for the appropriate disk unit. The unit table word for the DFO has bit 39 (DO) set. The start IO operation for a DFO causes the unit table word to be locked fetched from memory. If the DFO bit is set, the DFOQ flag is set in the IOM, the unit table word is written to memory and unlocked, and the translator goes to an idle state.

At any time while in idle state, when the DFOQ flag is set, the DFO stack is not full, the DFO is on, the scan bus is not busy (SBY), and

no jobs of a higher priority exist, an automatic DFO scan-out is initiated. The Queue Scan (QSN) flag is set and start mode is entered. The DFO unit table word is lock fetched from memory, and the DFO IO Queue Header (IOQH) is read. If the IOQH is null, DFOQ and QSN are reset, the unit table word is written to memory and unlocked, and the translator returns to an idle state. If the OH is not null, scan mode is entered. The CDL word is fetched from IOQH+4.

Several fields are combined to form a scan-out word. The disk address and IOCB address fields form the scan data word, and the EUD field plus device type and function code fields form the scan address word. This information is placed on the scan bus. The UT word is written to memory and unlocked, and the Scan In Process (SIP) flag is set which inhibits the translator from initiating other jobs when in the idle state. The idle state is then entered. When SIP=1 and SBY=0, SIP is reset and scan mode is re-entered.

The UT word is lock fetched from memory, the IOQH is read and the Next Link (NL) field is obtained from the IOCB; then NL is stored in the IOQH. If NL=0, then the IO Queue Tail (IOQT) is nulled and DFOQ is reset. The UT word is written to memory and unlocked, QSN is reset, and the translator returns to the idle state.

A scan-in word is formed and placed on the scan bus when the translator is in the initial state, the scan bus is not busy, and a DFI channel is not busy (served by a DFO). When the Scan-In operation is completed, the translator is notified. The 20 bit address field of the scan-in word is the base address of the IOCB for the next disk-file-EU to be started. The base address plus 4 is used to fetch the CDL information word. The EU Designate field of the CDL word and the UT Base are added, and the result is then used to fetch the UT word of the disk unit from memory. If the UT word busy-bit is zero, the Buffer Descriptor, CDL and IOCW are fetched as before and are sent to the DFI section to start the device. The base address is stored in the IO Queue Head and is used for generating information during the terminate operation of the device. The busy-bit is set and the UT word is stored unlocked in memory and control is returned to the initial state.

If the busy-bit is set in the disk UT word, the IOCB is linked into the tail of the DFO's IO Queue.

Disk-Pack Control

The following actions are taken upon the receipt and examination of result descriptors from Disk Pack units.

When a result descriptor indicating "Seek Initiated" is received, the IOM will not de-link the IOCB or store the result descriptor as in normal terminate operations. Instead, the unit number of the Disk Pack which has begun a seek operation will be stored in a local stack. The contents of this stack are then used to monitor the ready lines of all Disk Pack Units which are currently seeking.

A "Seek Complete" will be detected when the ready line of a seeking Disk Pack returns to the true state. At this time, the Translator will perform a Start I/O for that Disk Pack unit. Since the original job

was not de-linked from the job queue when the Disk Pack initiated its seek operation, the same job will be issued a second time, and the data transfer will occur. After this point, all IOM operations proceed the same as for normal peripheral units. If the Disk Pack is issued a conditional I/O command which does not require head positioning, data transfer occurs directly, and the Automatic Disk Pack functions of the IOM are not used.

Fail Mode of Operation

The fail mode is used to provide the IOM the capability of reporting errors that cannot be associated with a specific request. When an error occurs, such as Scan Bus Error, Memory Error, Home Address Error, Illegal Command, etc., control is passed to the Fail Mode (MF) with appropriate error flags.

A fail result descriptor is built in the fail register. This RD indicates (1) the operational mode when the error occurred, (2) a possible channel number (or memory address, depending on the type of failure), and (3) error flags describing the type of error.

A Fail Unit Designate number (Fail UD = zero) is used with the UT word to access a Fail UT word. Then the QH and Fail UD is used to access the I/O Queue Head. The fail result descriptor is placed in the Result Descriptor word of the I/O Control Block (IOCB). The Fail IOCB is then delinked from the queue of Fail CB's and linked to the Status Queue as on a normal termination. An IOM Error Interrupt is then sent by the IOM to the Central Processor designated in the Status Queue Header.

FUNCTIONAL OPERATION OF MIU

The MIU (figure IV-2-2) performs all data and map-word transfers between the IOM and a maximum of 8 system Memory Control Modules, and detects and reports memory error conditions to the requesting functional unit of the IOM (and to the translator when applicable). The MIU manages level-1 memory access requests by the functional units of the IOM on a preassigned priority basis. The access priority scheme for the functional units of the IOM is as follows:

a. First Priority: Data Service requests

b. Second priority: Data Communications Processor interface requests

c. Third Priority: translator requests

When a functional unit of the IOM requires the services of the MIU for the purpose of performing a data transfer, it is required to raise its Access Request Line to the MIU and place a 26-bit Unit Control Word (UCW) on its UCW lines to the MIU. When the requesting unit has priority, the MIU loads the UCW into its control word register and performs one of the following operations:

a. single data word fetch

b. N-length data word fetch

c. single word overwrite with flashback

d. single word protected write

e. single word protected write with flashback

f. N-length overwrite

g. single word overwrite

h. N-word protected write

Upon determining the type of operation requested, the MIU constructs a Memory Control Word (MCW) and transfers it to memory. Upon transferring the MCW to memory, the MIU is required to perform one of the operations listed below:

a. If a Single Word Store operation was specified: The MIU raises its request lines to the specified Memory Control Module (MCM) and (in order to alternately transmit the MCW and the data word to be stored) to the addressed MCM. The MIU continues to transmit the MCW, followed by the data word to be stored, until an acknowledge signal is received from the MCM.

b. If a Multiple Word Store operation is specified: The MIU raises its request lines to the applicable MCM, and then sends the MCW to the MCM. When the MCM acknowledges receipt of the MCW, the MIU commences the data transfer under the control of the Data Request signal.

c. If a fetch operation is specified: The MIU raises its request lines and sends the MCW to the applicable MCM. When the MCM acknowledges receipt of the MCW, the MIU enables its memory-bus receiver circuits. Information from the MCM will now be accepted by the MIU. However, the MCM is required to transmit a Data Present Strobe pulse to the MIU to cause the information present on the memory bus to be transferred to and detected by the requesting IOM. The Data Present Strobe pulse is required for each word transferred from Memory to a requesting IOM.

While performing a data transfer, the MIU is required to detect and/or report memory error conditions. Memory errors are divided into two categories by the IOM: MIU-detected errors, and memory-detected errors. Memory errors cause termination of the memory request being processed, and the MIU transfers a 3-bit error code to the requesting section. The Translator reports these errors through the Fail Register. Data-service-section units return these errors in the Result Descriptor. A decode of these three bits specifies whether the error is an MIU or memory-detected error.

Errors detected and/or reported by the MIU and their associated 3-bit error reporting codes are listed in descending exclusive order as follows:

a. Store disparity (011) - This error condition is declared if a data transfer from an internal unit is received by the MIU with incorrect parity. The data with incorrect parity is transferred to the memory.

b. L1A Address Residue Error (010) - This error condition is declared if the MIU receives a UCW whose residue bits do not agree with its memory address field configuration (DCP words are not residue checked.)

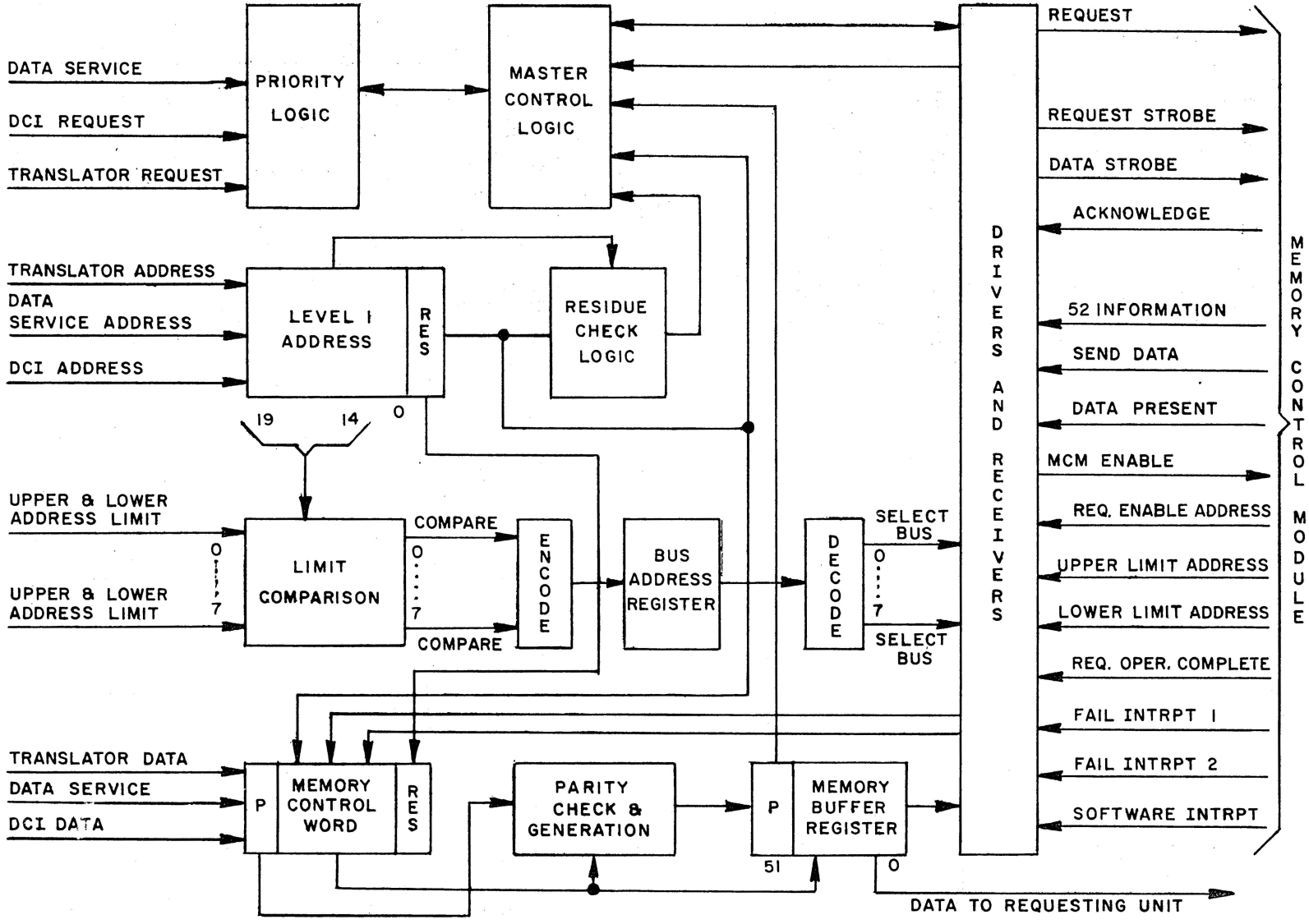


Figure IV-2-2. Memory Interface Unit

c. Memory Detected Error (111) - This error condition is declared when the addressed memory module responds with a fail 1 (uncorrectable error) indication to a requestor unit.

d. No Access to Memory (101) - This error condition is declared if the MIU receives no response from the requestor memory module during a waiting period not to exceed 25 microsecond writing period. No response is defined as:

1. Failure to receive, at the MIU, an acknowledged signal from an addressed memory module, or

2. when incomplete data is transferred by an addressed memory module.

e. Fetch Disparity (110) - This error condition is declared if a fetch of data from memory is received by the MIU with incorrect parity.

f. Memory Protect Error (100) - This error condition is declared when the addressed memory module responds with a protect-error signal during a memory protect store operation.

g. Memory Detected Error (001) - This error condition is declared when the addressed memory module responds with a fail 2 (1-bit corrected error) indication to a requestor unit. (This error condition does not cause termination of the memory access operation.)

Functional Components of the MIU

The MIU consists of nine functional components interfaced as illustrated in figure IV-2-2 and operated as described below:

a. Priority Logic - This section is responsible for granting the services of the MIU to the highest priority requesting unit.

b. Master Control Logic - This section contains the control logic necessary to execute all MIU operations, including the controls required to complete receiver and driver paths.

c. Residue Check Logic - This section is responsible for checking and verifying the residue bits of the memory addresses transferred from the translator and data service unity.

d. Parity Check and Generate Logic - This section is required to generate odd parity for all words being transferred to memory, and to check for odd parity of all words being fetched from memory.

e. Data Buffer Register - This is a 52-bit register and is used to buffer all data transfers between the requesting area of the IOM and the MIU.

f. Memory Buffer Register - This is a 52-bit register and is used to buffer all input data to memory.

g. L1A and Residue Logic - This section is responsible for constructing and routing the MCW to the data buffer register, as bits of the address.

h. Receivers and Drivers - There are 8 discrete groups of receiver and driver circuits in the MIU, one group per Memory Control Module interface. The state of these groups is determined by master control; only one group is active at any one time.

i. Limit Comparison Logic - This section is responsible for comparing the upper and lower address limits for the memory address: It is comprised of the limit comparison logic, encode logic, bus address register, and decode logic. It is the function of these logic areas to inform the requestor as to which MCM will be servicing the memory request and is accomplished by comparing the 6 most significant bits of the memory address to a known upper and lower limit for the answering MCM.

FUNCTIONAL OPERATION OF PCI

The PCI (figure IV-2-3) enables the IOM to interface with from one to twenty peripheral controllers (PCs), and coordinates data transfers between the PC's and the MIU as directed by the translator section of the IOM. The PCI interfaces with memory by one-word transfers via the MIU. Each PC requires a 1-microsecond service cycle to transfer data. By means of overlapping service cycles, and by use of local memory windows (a one-clock period during which a particular operation may be performed if no higher priority operation is required), it is possible to multiplex all twenty channels.

There are four operational phases of the PCI: (1) channel initiation (2) channel service, (3) memory operations and (4) channel termination. Each of these phases is described as follows. A general block diagram of the PCI is as shown in figure IV-2-3.

Channel Initiation Operation

This phase of PCI operation is controlled by Channel Designate Level (CDL) Control (referred tq as CC), and includes all functions required of the PCI to start a device. The channel initiation phase commences when the translator control logic places the first job descriptor word (DSU word) on the translator bus and raises the request line to the PCI. When a local memory window becomes available, the CC strobes the DSU word into the local memory channel allocated to the device to be started, and lowers the PCI-available line to the translator. The translator control logic then lowers the request line and places the second job descriptor word (CDL word) onto the translator bus. When the next available local memory window occurs, the CC strobes the CDL word into the appropriate location in local memory, strobes the channel number of the new request into the initiate queue stack (INQ), and raises the PCI-available line. The translator is thus informed that the entire request descriptor has been received and stored in local memory. If no channel is currently being initiated, the CC selects the highest priority channel in the INQ, transfers this channel number to the

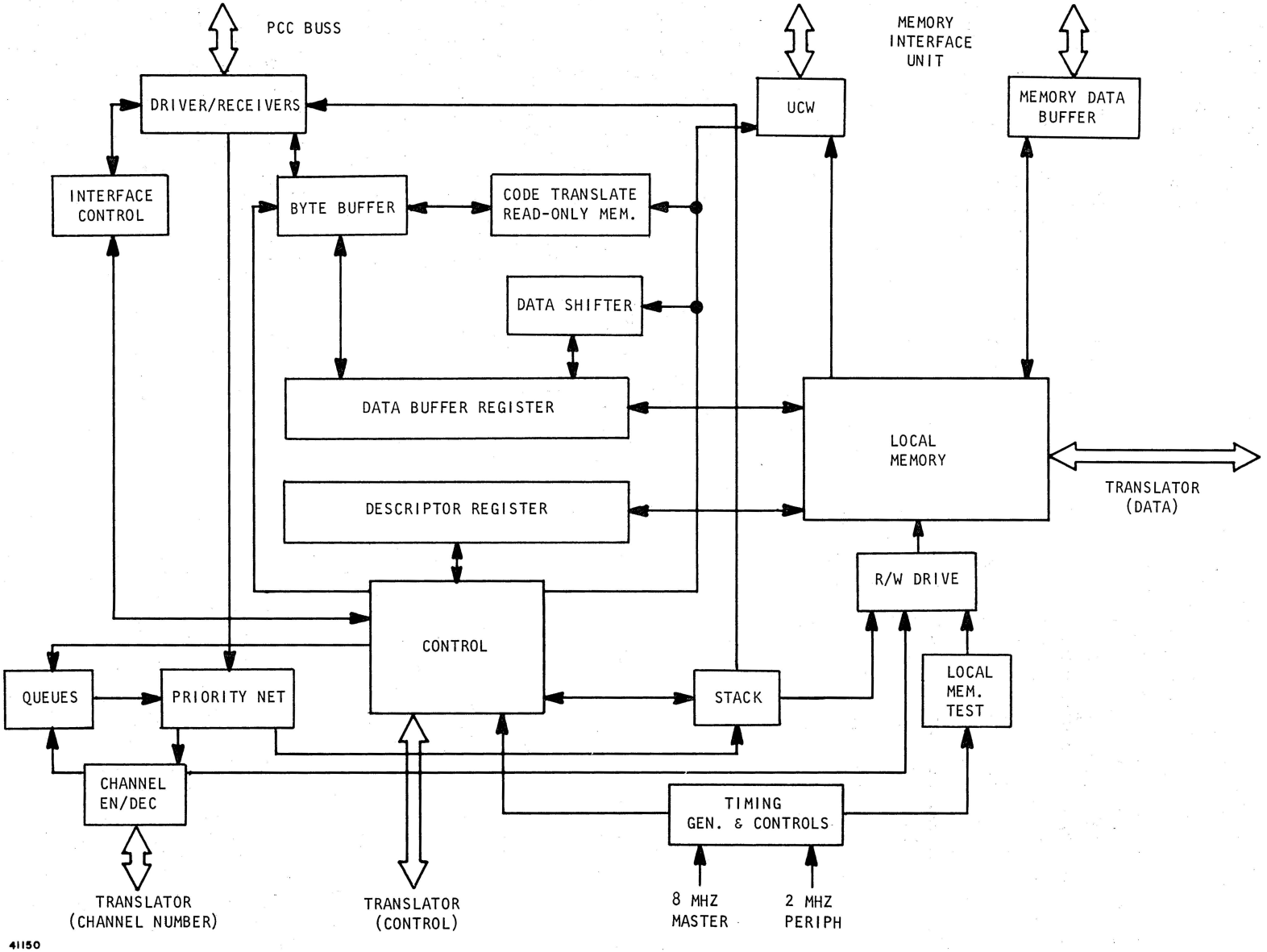


Figure IV-2-3. Peripheral Control Interface

CDL stack (CDS) which contains the request descriptor currently being initiated, and resets the INS bit for the selected channel. If no channel requires channel service, the CC checks the busy line of the channel. If a not busy condition is detected, the CC commences transfer of CDL characters to the appropriate PC at the rate of one character per available service cycle. If the selected channel is busy when initiation is commenced or if the selected channel becomes not busy during transmittal of CDL characters, the request is terminated and appropriate result descriptor information is generated and transferred to the translator (see channel termination). After sending the correct number of CDL characters (4 for standard devices and 8 for disk file devices), the CC raises the start channel bus line to the PC and resets the CDS, and the initiation operations are completed.

Channel Service Operation

After completion of the initiation phase, a channel is serviced upon demand at a rate dependent upon the type of peripheral device involved. The PC requests service by raising the access-request line (ARL) to the PCI. The PCI selects the highest priority channel requesting service and generates the appropriate access granted level (AGL). The presence of this signal grants the next service cycle to the accessed peripheral. The service cycle consists of two T-time periods (T1 and T2) of 500 nanoseconds each; T1 is used for input from the PC, and T2 is used for output to the PC. The AGL signal for the next service cycle is generated during the previous service cycle's T2 time period.

Each data transfer is controlled by a channel descriptor which has been generated from information contained in the DSU word of the job descriptor.

If an error is detected at any time during channel service, the PCI generates the appropriate result descriptor information for the translator (see channel termination) and terminates all operations on that channel.

Memory Operation

When the PCI determines that one 52-bit data word is required from or is ready to be sent to memory, the channel number selected for the transfer is placed in the Memory Queue (MQ). The MQ is a stack which contains the channel numbers of all channels requiring memory access. If no memory operation is currently being executed, the PCI selects the highest priority job in the MQ, and transfers this number to the Memory Operation Stack (MOS). The PCI then resets the MQ bit for the selected channel, transfers the Unit Control Word to the Unit Control Word Register (UCWR), raises the PCI-memory-request line to the MIU, and, if necessary, transfers data into the Memory Transfer Area (MTA).

Once access to the MIU is granted, the PCI strobe fetches data from the MTA, strobes the data to the appropriate data buffer in local memory, awaits the release signal which indicates that this memory request is completed, and then resets the MOS bit. The memory operation is thus completed.

If the MIU detects an error at any time during this sequence, the error information is transferred to the

PCI. The PCI then causes the request to be terminated, and an appropriate result descriptor is generated.

Channel Termination Operation

After completing the required data transfer, the PCI sends an I/O complete level to the PC. The PC then returns the result descriptor available level and returns a result descriptor. This result descriptor information, along with the present Channel Descriptor information, is used by the PCI to create the result descriptor word to bring about a normal termination.

Abnormal-termination result descriptors can occur (1) during either channel initiation, channel service, or channel memory operations (when errors are detected by the PCI), (2) during channel memory operation (when errors are detected by either the MIU or the MCM, and (3) during channel initiation or channel service (when errors are detected by the PC). No matter what the source, all result descriptors are treated identically.

Once the result descriptor has been generated, it is stored in the local-memory location allocated to the channel to be terminated. The channel number of this request is strobed to the translator. Should the translator be unable to accept the channel number, the PCI stores this channel information in the Termination Queue (TMQ), which contains all the requests to be terminated. Whenever possible, the PCI selects the highest priority request from this stack, transmits the channel number to the translator, and resets the TMQ bit for the selected channel.

The translator replies to the PCI termination with a read-result-descriptor request, which causes the result word to be placed on the translator bus. This completes the termination operation.

Functional Components of the PCI

The PCI consists of nine functional components as illustrated in figure IV-2-3. These functional components are described as follows.

a. PCI Local Memory (PCLM) - Consists of twenty 126-bit word locations, one for each PC channel within the PCI. Is used to store the channel descriptors and data for the twenty channels.

b. Descriptor Register (DR) - A 105-bit register which is loaded with the active channel descriptor.

c. Shift Logic (SFT) - Consists of right and left shift logic to properly locate data within the data buffer of the channel descriptor.

d. Byte Buffer (BB) - A 16-bit register which is loaded and unloaded (two 8-bit bytes at a time) during data transfers between the IOM and a PC.

e. Code Translator (CDT) - Consists of the logic circuits necessary to perform EBCDIC TO BCL - EXTERNAL or vice-versa, BCL-External to BCL-Internal or vice-versa, ASCII to EBCDIC or vice-versa, and ASCII to BCL-External code translations or vice-versa.

f. Stack Control - Contains stacks whose functions are described in functional operations.

g. Control Section - Contains the CDL control, channel service control, memory transfer control, and termination control which are described in preceding paragraphs.

h. Driver/Receiver Section - Consists of special drivers and receivers used for transmitting and receiving data and controls for the PC.

i. Master Timing and Interface Control (STC) - Consists of logic to control the timing and interface sections and to provide control logic for the control panel.

FUNCTIONAL OPERATION OF DFI

The DFI (figure IV-2-4) enables an IOM to be interfaced with up to eight Disk File Controls (DFC). It consists of two independent, modular sections, each of which is capable of handling 4 data channels; each channel is interfaced to one DFC.

Each section controls data transfers with the DFC's via a 16-bit data bus at a transfer rate of 2 eight-bit characters per transfer time. The transfer rate to memory is 2 words (2 x 48 bits) per transfer time.

Each data channel comprises a four-word data-buffer area, called local data mode memory (LMD), and a 66-bit Channel-Descriptor Local Memory (LMC). Upon command from the translator, the DFI initiates requests with its associated DFC's. During data transfer operations, the DFI communicates with the Memory Interface Unit (MIU) to obtain memory accesses. Upon request completion, the DFI notifies the translator of the termination status and awaits re-initiation.

The translator, upon receipt from memory of a disk file job, requests transmission of a job word to the DFI section assigned to handle that job. The selected DFI section loads the new request word into the proper channel-descriptor location in local memory and then releases the translator.

The DFI section, according to priority, reads the channel descriptor of an active request. An active request may be in any one of 3 separate modes. The exact mode is determined by the FAZ field in the channel descriptor. The three modes are (1) channel initiation, (2) channel service, and (3) channel termination.

Channel Initiation Operation

In the channel initiation mode, 48 bits of information are sent to the addressed DFC. This transfer is accomplished as 6 transfers of 8-bits each. These 48 bits are contained in the second request word (job word 2) which was, upon receipt from the translator, stored in the LMD. The W/BP field of the channel descriptor points to the location in the LMD of the information to be transferred to the DFC. A 16-bit byte, of which only the most significant 8 bits are valid, is loaded into the transfer register (TR) and is then sent to the DFC. After 6 such transfers, a Start Channel Bus (STCB) signal is sent to the DFC to indicate the end of the initiation phase, and the Phase (FAZ) field of the channel

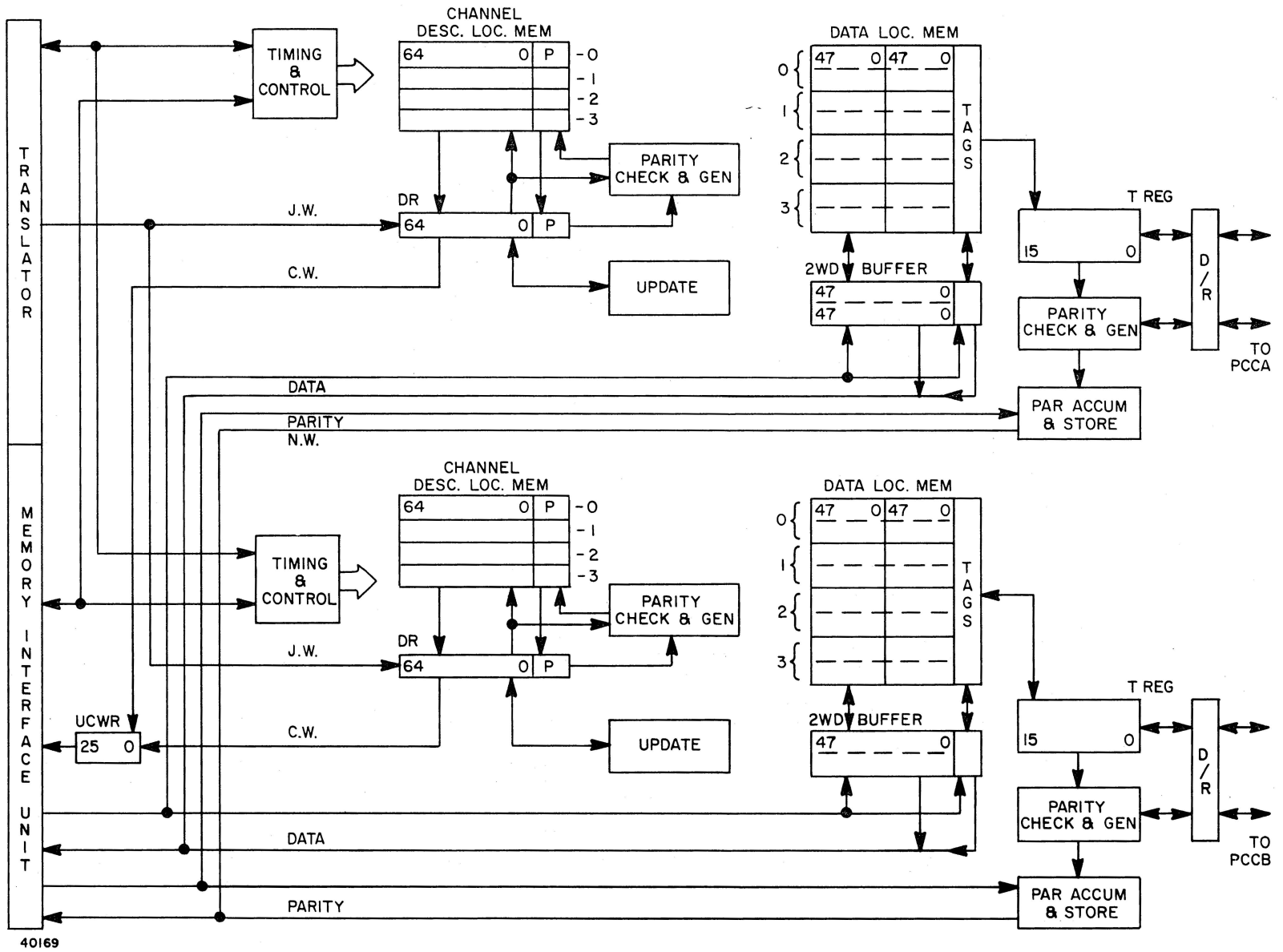


Figure IV-2-4. Disk File Interface

descriptor word is incremented by one. If the job word indicates an output operation, two fetches of two words each are requested from memory by way of the MIU during the initiation phase. The first of these two-word fetches is stored in word positions 00 and 01 of the LMD, and the second two-word fetch is stored in word positions 10 and 11 of the LMD.

Channel Service Operation

The channel service operation, as indicated by the Phase (FAZ) field of the channel descriptor, consists of transferring 16-bit bytes of information to or from the DFC. If the DFC corresponding to the active channel has raised its access-request-level (ARL) line, the DFI responds by raising its access-granted-level (AGL) line and follows this by granting a one-microsecond service cycle.

If the transfer is to be an input, data is accepted during the first half of the one-microsecond service cycle and written into the Data Local Memory (LMD). If the transfer is to be an output, data is read from the LMD, placed in the transfer register (TR), and put on the data bus during the second half of the one-microsecond service cycle. The location in the LMD of the 16-bit byte to be transferred is determined by the Word/Byte Position (W/BP) field of the channel descriptor. With each 16-bit byte transfer, the W/BP field is incremented by one. After two words of information have been transferred, memory request is again made by way of the MIU. The current memory address (CL1A) is sent along with certain control bits to the MIU, and if the request is for a read, two words are loaded from the LMD into the two-word buffer. If the request is for a write, two data words from memory will be loaded into the two-word buffer and then into the LMD. At this time the current memory address (CL1A) is compared to the final memory address (FL1A). If they are equal, the channel service phase is completed and the FAZ field is incremented by one; if they are unequal L1A is incremented by 2. The channel descriptor is restored into LMC and the next channel is serviced.

Channel Termination Operation

The channel termination operation is initiated when either of the following conditions occurs:

a. Normal completion of transfer (CL1A = FL1A)

b. Detection of an error by the DFI or the Disk File Control (DFC).

If the DFI detects the condition CL1A = FL1A or detects an error, it sends an I/O-complete signal to the DFC. When the DFC detects either the I/O-complete signal or an error, it sends a result descriptor available (RDAV) signal along with the next ARL signal. The DFI accepts the result descriptor and stores it in the FL1A field of the channel descriptor. The DFI sends the terminating channel number to the translator, and then awaits receipt of a Read-Result-Word request. When this request is received at the DFI, the result descriptor is sent to the translator and the channel termination phase is completed.

Functional Components of the DFI

Figure IV-2-4 illustrates the two DFI sections and their respective

interfaces with the Translator, MIU, and Peripheral Control Cabinets (PCC). The two DFI sections are identical and contain the following components:

a. Channel Descriptor Local Memory (LMC) - Provides storage for four 66-bit channel descriptors; one for each DFC channel.

b. Descriptor Register (DR) - Used to store the descriptor of the active channel. The DR contents are used in conjunction with update logic to update the current memory address (CL1A) of the active job and to update various control bits.

c. Update Logic - Used to update the CL1A by two words each time a memory access is requested. It is also used to update the Word/Byte Position (W/BP) field, residue and phase (FAZ) fields, and various other control bits.

d. Parity Check and Generate Logic - Generates and stores odd parity for each descriptor to be stored in the LMC, and checks for odd parity on all descriptors read from the LMC.

e. Data Local Memory (LMD) - Provides storage for sixteen 48-bit data word locations (two double-word locations for each of the 4 DFC channels). The LMD acts as a buffer for data read from or written onto disk files. Also, during initiation of a request, the LMD contains the 6 CDL characters (48 bits total) which are sent to the DFC.

f. Two-word Buffer - Acts as a buffer for data being transferred between the LMD and the MIU. It contains storage for two 48-bit words.

g. Transfer Register (TR) - A 16-bit register used to buffer all data transfers to or from the DFC.

h. Parity Check, Generate, Accumulate, and Store - Checks and generates parity on data transferred from or to the DFC. When data is sent to the DFC, a parity bit is received from the MIU with each data word, and is stored in the accumulator. The parity bit setting of the accumulator is updated with each 16-bit transfer to the DFC, and is checked against the parity of the last such transfer. When data is received from the DFC, a parity bit is received with each data transfer and is stored in the accumulator. The parity bit setting is updated for each new 16-bit transfer, and a final setting is sent to the MIU for each new 48-bit data word. The MIU then checks parity on the full 48-bit word (three 16-bit transfers per 48-bit word).

FUNCTIONAL OPERATION OF SCI

The SCI (figure IV-2-5) contains the storage and controls required to provide a scan bus for communicating with four DCP's and four DFO's. The scan bus to the four DFO's is shared between two IOMs.

The translator initiates scan operations by transmitting a scan control word to the SCI. If a scan-out is

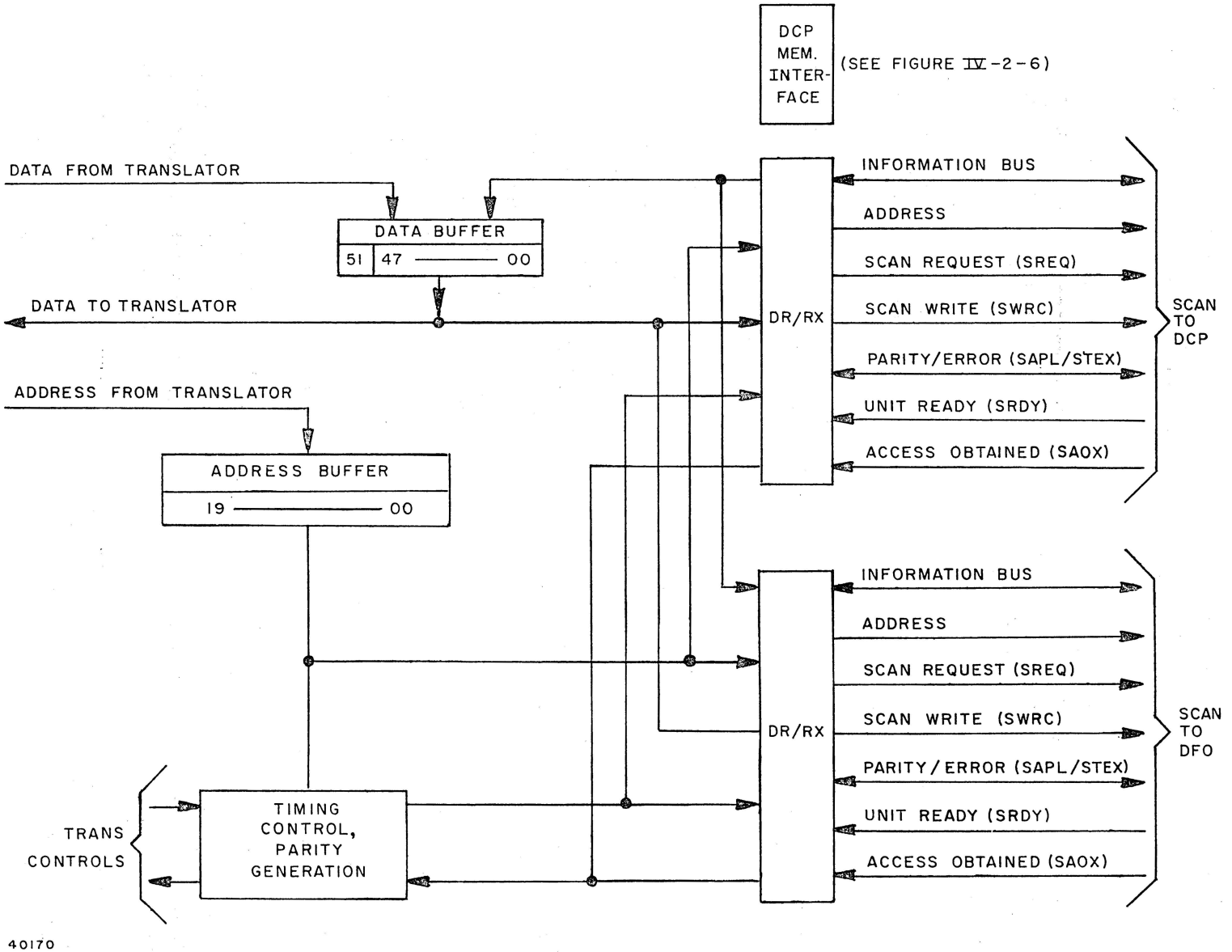


Figure IV-2-5. Scan Bus Interface

required, the translator also transmits the scan-out information to the SCI. Upon completion of the scan operation by the scan-out information to the SCI. Upon completion of the scan operation by the scan-out information to the SCI. Upon completion of the scan operation by the SCI, the translator is notified. In the case of a scan-in operation, the scan-in information is loaded into the translator B register. If an error has been detected by the SCI, error information is loaded into the translator F register.

There are two error conditions which can be reported to the translator by the SCI:

a. Not Ready - If the DFO or DCP addressed by the scan bus does not respond with a ready signal within 3 usec, a not ready error is reported to the translator.

b. Module Error - If the DFO or DCP addressed by the scan bus detects an error on a scan-out or scan-in operation, an error signal is transmitted to the SCI. The SCI then reports a scan error to the translator.

Scan Interface

A DFO is selected by means of the 8-bit EUD code presented over the scan address lines. An EU may be connected to 2 DFOs (directly to one DFO and indirectly via it to the second DFO). Therefore, when the system places an EUD code on the scan bus lines in order to communicate with a DFO, both DFOs could respond were it not for a means of inhibiting the response of one DFO. This means is bit 7 (ES) of the scan address lines. If bit 7 is low, the DFO connected directly to the referenced EU is selected. If bit 7 is high the DFO connected indirectly to the referenced EU is selected. During execution of the report request operation, the DFO responds regardless of the state of bit 7, if it is not part of a DFO pair. The scan information lines constitute the scan-in and scan-out words during the corresponding operations.

DCP Scan Interface

During scan-out operations (only scan-out orders are accepted by a DCP), the scan information lines constitute the scan-out word. The SCI provides a maximum of four DCP memory interfaces (see figure IV-2-6) in a DCI unit.

The DCI unit contains all storage capability and controls required to interface with the DCP memory buses. The memory transfer operations performed are:

a. Fetch (one word)

b. Store with flashback (one word)

c. Protected store with flashback (one word).

If an interface is not used by a DCP, it may be used to accommodate a suitable device.

All errors detected by the DCI or MIU for a DCI memory request are transmitted to the DCP that initiated the memory request.

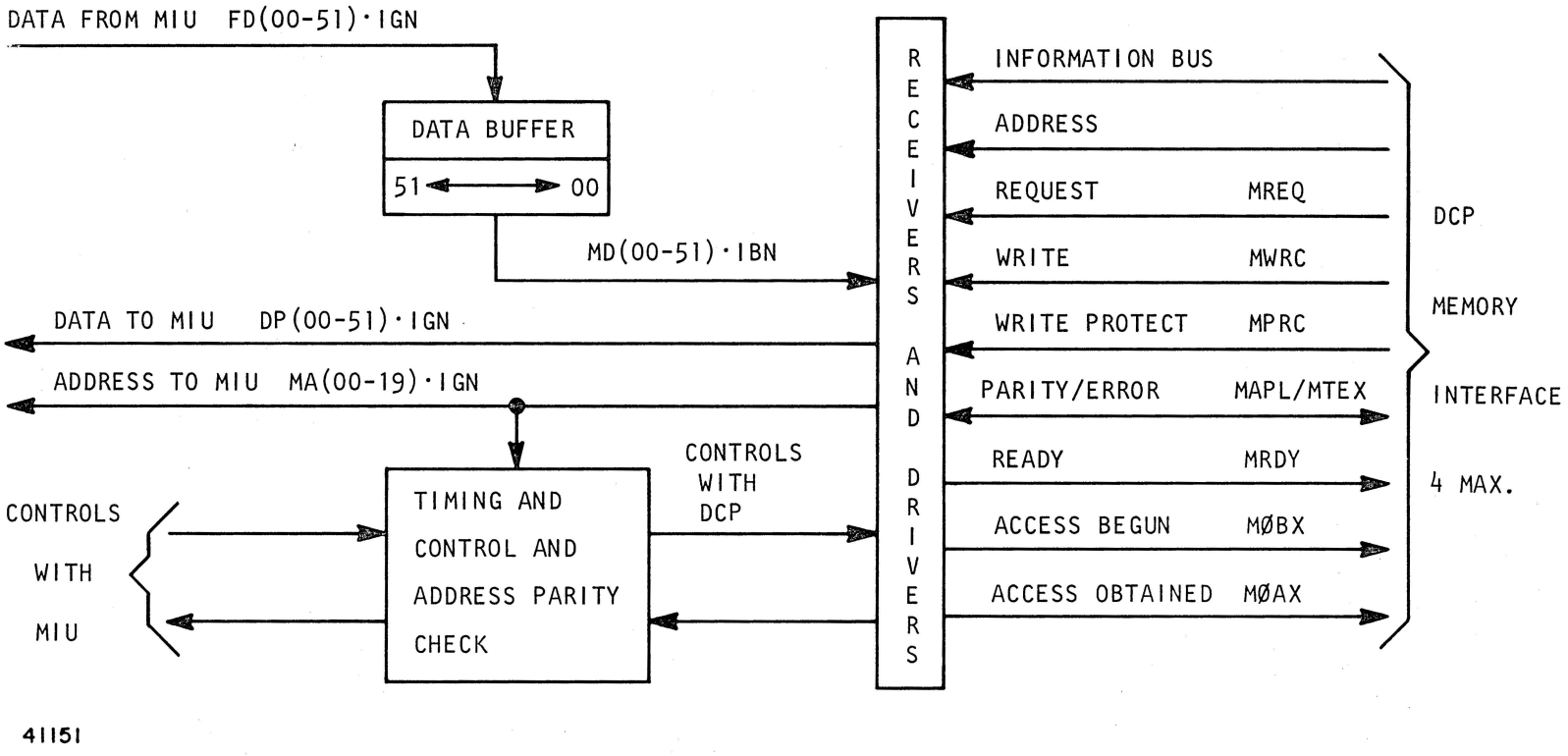
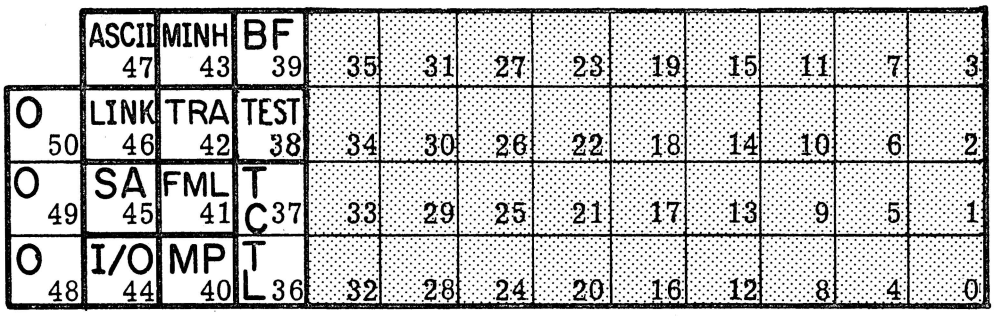


Figure IV-2-6. DCI Unit

SECTION 3
PERIPHERALS AND CONTROL WORD FORMATS

IOCW

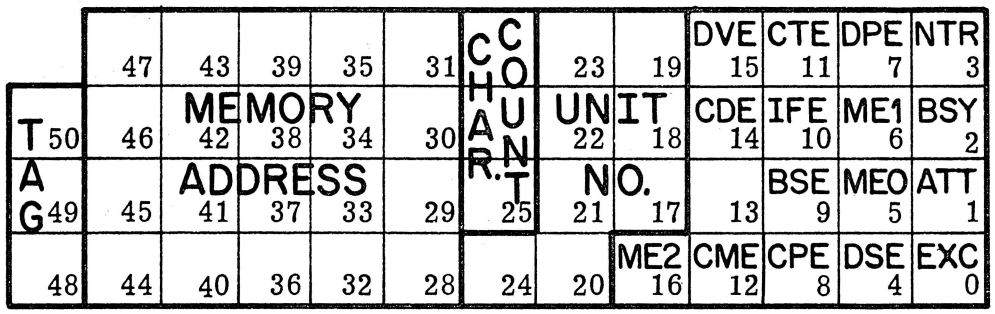


41009

FIELD		DESCRIPTION
ASCII	47:1	ASCII. When set, indicates that ASCII translation is required.
LINK	46:1	SIDELINK. When set, indicates that a link to another IOCB is required. The actual link is contained in the Side Link Word which is the second word of a IOCB and which has the following format: 47:8 Unit descriptor 39:20 Side link address
SA	45:1	SOFTWARE ATTENTION. This bit, when set, will cause bit 1:1 (Software Attention) to be set in the Result Descriptor.
I/O	44:1	INPUT/OUTPUT. When set, indicates that the transfer is to be an input operation. When reset, indicates that the transfer is to be an output operation.
MINH	43:1	MEMORY INHIBIT. When set, indicates that data will not be transferred to/from memory.
TRA	42:1	TRANSLATE. When set, indicates that internal IOM translation is needed.
FML	41:1	FRAME LENGTH. When set, indicates that the frame length is to be 8 bits. When reset, indicates that the frame length is to be 6 bits.

FIELD		DESCRIPTION
MP	40:1	MEMORY PROTECT. When set, indicates that an attempt to store into a protected word in main memory (bit 48 = 1) will result in the store not being done and fail signal being returned.
BF	39:1	BACKWARD/FORWARD. When set, indicates a backward operation on a tape unit. When reset, indicates a forward operation on a tape unit.
TEST	38:1	TEST. When set, indicates that a test operation is being performed.
TCTL	37:2	TAG CONTROL. The bits are defined as follows: 0 = Store single precision tags. 1 = Store program tags. 2 = Tag field transfer. 3 = Store double precision tags.

RESULT DESCRIPTOR (NORMAL)



41010

FIELD		DESCRIPTION
TAG	50:3	0 = Normal 4 = The IOM was unable to do a sidelinked job.
MEMORY ADDRESS	47:20	The main memory address of the last word executed by the IOM.
CHAR COUNT	27:3	The number of characters in the last word in memory which have been validly executed by the IOM.
UNIT NO	24:8	The unit designate number of the device on which the job was executed.
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
Unit or DSU error bits	15:9	These bits describe either unit related errors or errors detected by the DSU, depending on the value of bit 4 (DSE). If DSE is false, bits 15:9 describe unit related errors as detected by the peripheral control. Unit related errors are discussed with the individual peripherals. If DSE is true, bits 15:9 contain a description of errors detected by the Data Service Unit. These descriptions are applicable for all units and are as follows: DVE 15:1 DEVICE DETECTED ERROR. An error detected by the device and reported in the device result descriptor, but which, due to the precedence of a DSU detected error, cannot be reported normally within bits 15:9.

FIELD		DESCRIPTION
	CDE 14:1	CDL CHARACTER ERROR. An illegal CDL character detected by the PC but not reported in the device result descriptor.
	CME 12:1	COMBINATION ERROR. An error detected by the DSU, of an illegal combination of control bits within the channel descriptor.
	CTE 11:1	COUNTER ERROR. An error detected by the DSU on internal count operations.
	IFE 10:1	INTERFACE ERROR. An error detected by the DSU on the various interface control lines between the DSU and the PC.
	BSE 9:1	BUSS PARITY ERROR. A parity error detected by the DSU on the data buss from the PC.
	CPE 8:1	CONTROL PARITY ERROR. A parity error detected by the DSU on the various control bits of the channel descriptor.
	DPE 7:1	DATA PARITY ERROR. A parity error detected by the DSU on the 48 bit data buffer plus the 3 bit tag field.
ME1	6:1	MEMORY RELATED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY RELATED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error. When reset, this bit indicates that bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY. This bit indicates that the specified device was found to be not ready by the peripheral controller. This bit is generated and forwarded by the peripheral controller (PC).
BSY	2:1	CHANNEL BUSY ON INITIATE. This bit indicates that the channel-controller was busy when the Initiate I/O command was received by the DSU.

RESULT DESCRIPTOR (NORMAL)

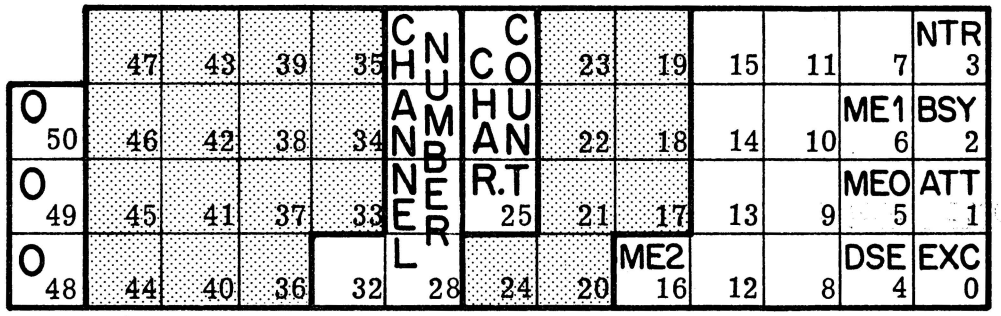
FIELD		DESCRIPTION
ATT	1:1	SOFTWARE ATTENTION. This bit indicates software attention to this descriptor is necessary. The bit is included in the initial job word to the Data Service Unit and is merely stored and returned, not generated, by the DSU.
EXC	0:1	EXCEPTION. This bit indicates that a DSU error has been detected or that the PC has returned a Result Descriptor with the PC exception bit set.

MEMORY REPORTED ERRORS. The following error descriptions are a result of decoding ME2, ME1, and ME0 and are listed in descending exclusive order:

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY. This error condition is declared if a data transfer from an internal unit is received by the MIU with incorrect parity. The data with incorrect parity is transformed to memory.
0	1	0	L1A ADDRESS RESIDUE ERROR. This error condition is declared if the MIU receives a UCW whose residue bits do not agree with its main memory address field configuration (DCP words are not residue checked).
1	1	1	MEMORY DETECTED ERROR. This error condition is declared when the addressed memory module responds with a Fail 1 (uncorrectable error) indication to a requestor.
1	0	1	NO ACCESS TO MEMORY. This error condition is declared if the MIU receives no response from the requested memory module during a waiting period not to exceed 25 microsecond writing period. No response is defined as: Failure to receive, at the MIU, an acknowledge signal from an addressed memory module, or When incomplete data is transferred by an addressed memory module.
1	1	0	FETCH DISPARITY. This error condition is declared if a fetch of data from memory is received by the MIU with incorrect parity.

ME2	ME1	ME0	DESCRIPTION
1	0	0	MEMORY PROTECT ERROR. This error condition is declared when the addressed memory module responds with a protect error signal during a memory protect store operation.
0	0	1	MEMORY DETECTED ERROR. This error condition is declared when the addressed memory module responds with a Fail 2 (1 bit corrected error) indication to a requestor. (This error condition does not cause termination of the memory access operation).

HARD LOAD RESULT DESCRIPTOR



41011

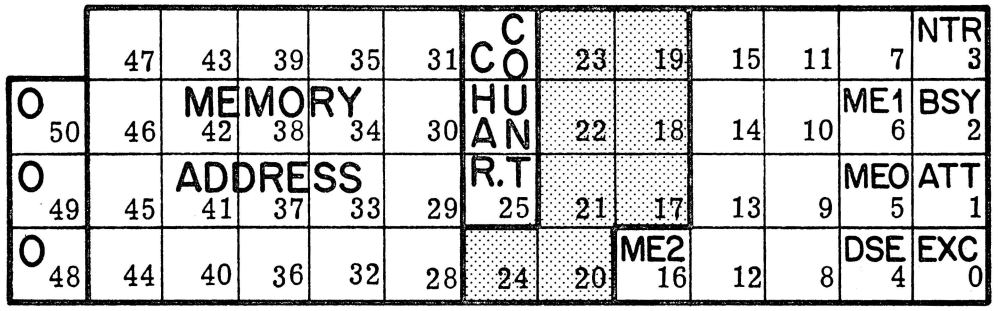
FIELD		DESCRIPTION
CHANNEL NUMBER	32:5	The number of the channel used in the operation.
CHAR COUNT	27:3	COUNT EXTENSION. This indicates the number of characters in the last word in memory which have been validly executed by the DSU.
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and MEO are encoded to report various errors detected by the MIU.
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and MEO are encoded to report various errors detected by the MIU.
MEO	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and MEO are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.

FIELD		DESCRIPTION
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

SYNCHRONOUS I/O RESULT DESCRIPTOR



41012

FIELD		DESCRIPTION
MEMORY ADDRESS	47:20	The main memory address of the last word executed by the Data Service Unit of the IOM.
CHAR COUNT	27:3	COUNT EXTENSION. Indicates the number of characters in the last word in memory which have been validly executed by the DSU. If normal termination occurred, this field will equal the Count Extension field of the initial IOCW.
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table:

15:1	DVE	DEVICE DETECTED ERROR.
14:1	CDE	CDL CHARACTER ERROR.
12:1	CME	COMBINATION ERROR.
11:1	CTE	COUNTER ERROR.
10:1	IFE	INTERFACE ERROR.
9:1	BSE	BUSS PARITY ERROR.
8:1	CPE	CONTROL PARITY ERROR.
7:1	DPE	DATA PARITY ERROR.

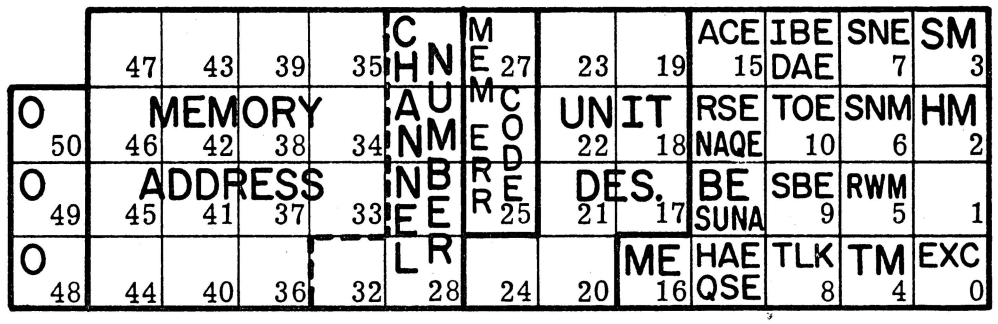
If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.

FIELD		DESCRIPTION
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

FAIL RESULT DESCRIPTOR



41013

FIELD		DESCRIPTION
MEMORY ADDRESS	47:20	This field contains the location in memory that was last accessed at the time of the error. This field is not valid if bit 15 (ACE) is set.
CHANNEL NUMBER	32:5	This field contains a channel number only when bit 15 (ACE) is set.
MEM ERR CODE	27:3	This field contains a memory error code and is valid only when bit 16 (ME) is set. The error bits are interpreted as follows: 1 = Memory detected error. 2 = LIA address residue error. 3 = Store disparity. 4 = Memory protect error. 5 = No access to memory. 6 = Fetch disparity. 7 = Memory detected error.
UNIT DES.	24:8	0 = A Unit Designate of all zeros signifies a Fail Register Result Descriptor.
ME	16:1	MEMORY ERROR. The memory error or MIU detected error is found by decoding bits 27:3 of the Fail Register.
ACE	15:1	ACTIVE CHANNEL STACK ERROR. The address (channel number) of the word in the stack that caused the parity error is contained in bits 47:20.

FIELD		DESCRIPTION
	14:6	If SNE (7:1) = 0, then bits 14:6 are defined as follows:
		14:1 RSE RESIDUE ERROR (MEMORY ADDRESS). The address in error is contained in bits 47:20.
		13:1 BE BUFFER REGISTER PARITY ERROR.
		12:1 HAE HOME ADDRESS ILLEGAL COMMAND.
		11:1 IBE INITIATE BUSY CHANNEL ERROR. An attempt was made to start a non-exchange channel that was either busy or in the process of being terminated.
		10:1 TOE TIME OUT ERROR. A data service time out error.
	14:6	If SNE (7:1) = 1, then bits 14:6 represent scan errors and are defined as follows:
		14:1 NAQE NO ACCESS TO DFO EXCHANGE.
		13:1 SUNA STORAGE UNIT NOT AVAILABLE.
		12:1 QSE DFO STACK PARITY ERROR.
		11:1 DAE DISK ADDRESS ERROR.
		10:1 TOE TIME OUT ERROR. A scan buss time out error.
		9:1 SBE SCAN BUSS ERROR. Indicates a parity error on the scan buss.
TLK	8:1	TABLE LOCKED. The translator timed out trying to fetch a locked Unit Table or Status Queue header.
SNE	7:1	SCAN ERROR. When set, indicates that bits 14:6 represent scan errors.
	6:5	Indicate the translator mode of operation when the error occurred as follows:
		6:1 SNM SCAN MODE.
		5:1 RWM RING WALK MODE.

FAIL RESULT DESCRIPTOR

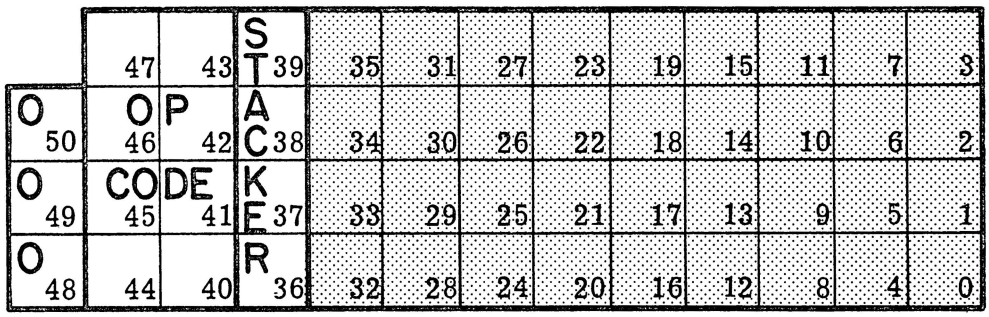
FIELD		DESCRIPTION
	4:1 TM	TERMINATE MODE.
	3:1 SM	START MODE.
	2:1 HM	HOME ADDRESS MODE.
EXC	0:1	EXCEPTION BIT. Indicates that a "1" exists in the Fail Register.

CARD PUNCH

Card Punch Control

The B 7212 Card Punch Control is used with the B 9213 "300 CPM Punch" which can punch either Binary, Alpha, or EBCDIC code at a rate of 300 cards per minute. Pre-punched cards may be used, but previously punched columns cannot be repunched. The Card Punch has a 1000-card capacity input hopper, and three output stackers (primary, auxiliary, and error) which have a capacity of 1200 card each. Stacker selection is accomplished programmatically.

CDL Word Format



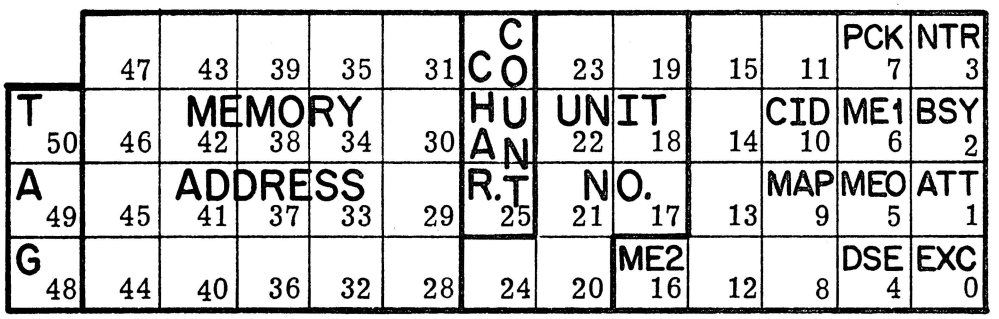
41014

FIELD		DESCRIPTION
OP CODE	47:8	23 = Punch BCL 24 = Punch BINARY 25 = Punch EBCDIC 99 = Test
STACKER	39:4	Stacker (not used on Test) 0 = Normal 1 = Auxiliary

IOCW Information

OPERATION	IOCW 47	42	41	CDL OP CODE
BCL External (no translate)	0	0	0	23
BCL (from INT. BCL)	0	1	0	23
BCL (from ASCII)	1	1	0	23
BCL (from EBCDIC)	0	1	1	23
BINARY (6 bit from 6 bit)	0	0	0	24
EBCDIC (no translate)	0	0	1	25
EBCDIC (from ASCII)	1	1	1	25

Result Descriptor Format



41016

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
CID	10:1	CONTROL ID. 1 for control 2 (TEST only).
MAP	9:1	MEMORY ACCESS OR PARITY (0281).
PCK	7:1	PUNCH CHECK.
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
MEO	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.

FIELD		DESCRIPTION
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operations

CARD PUNCH BCL (OP 23)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or punching 80 columns. The descriptor word count cannot exceed 10 words for punch BCL. BCL Internal Code, ASCII, or EBCDIC is converted to BCL code by translators in the IOM. The control can include one and only one of the following translators which are used to convert BCL code to BCL card code, ICT card code or BULL card code.

a. BCL-BCL Card Code Translator

b. BCL-ICT Card Code Translator

c. BCL-BULL Card Code Translator

CARD PUNCH BINARY (OP 24)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or by punching 80 columns. The descriptor word count cannot exceed 20 words for punch binary. A total of 160 six-bit characters of memory are required to punch 80 columns. The contents of each card column are divided into two 6-bit characters. The upper six bits are punched from the first 6-bit character received and the lower six bits from the next 6-bit character. Tag field transfers are compatible with this operator and must not be specified.

CARD PUNCH

CARD PUNCH EBCDIC (OP 25)

Punch one card on the card punch. The operation is terminated by punching the specified number of words or by punching 80 columns. The descriptor word count cannot exceed 13 words for punch EBCDIC. ASCII is translated to EBCDIC by translators in the IOM. The card punch control converts EBCDIC 8-bit code to EBCDIC card code.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

PUNCH CHECK ERROR

When a Punch Check is detected by Card Punch Control 1, the punching of that card is completed. The next card is punched and both cards are sent to the error stacker. The punch check bit is set in the result descriptor returned for the second card.

When Punch Check is detected by Card Punch Control 2, the punching of that card is completed, and it is sent to the error stacker. The Punch Check bit in the result descriptor is set.

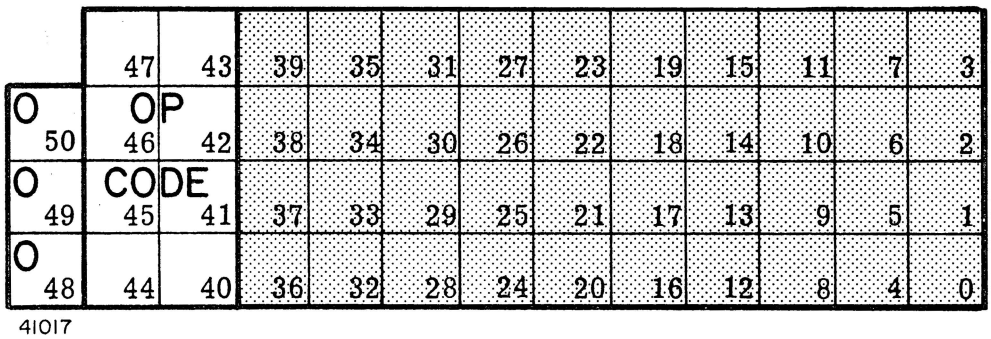
The Punch Check bit may be present in the Result Descriptor when addressing a non-present punch or one that is powered-down.

CARD READER CONTROL

The B 7110 Card Reader Control can be used with either B 9111 (800 cpm) or B 9112 (1400 cpm) card readers. The input hopper and the output stacker have a capacity of 2400 cards each. The card readers accept alpha, binary or EBCDIC card codes. The card reader converts alpha card code to BCL, which is then converted into internal BCL or EBCDIC by translators in the I/O Processor. EBCDIC card code is converted to internal EBCDIC by the B 7110 card reader control. When binary punched cards are read no translation is made.

The card readers can read 51-, 60-, or 80-column punched cards. Optional features include the ability to read 40-column Treasury checks and round holes in Postal Money Orders. Cards of varying thickness are acceptable; however, card thickness and length must be consistent during any one run.

CDL Word Format



41017

FIELD		DESCRIPTION
OP CODE	47:8	20 = Read BCL 21 = Read BINARY 22 = Read EBCDIC 99 = Test

CARD READER

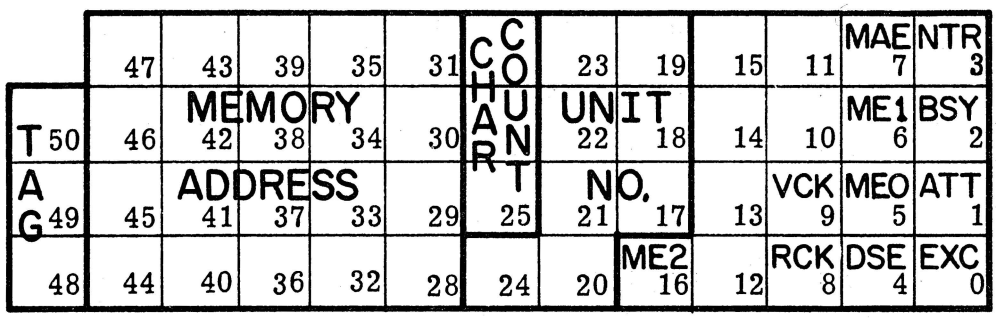
IOCW Information

OPERATION	IOCW 47	42	41	CDL OP CODE
BCL EXTERNAL (no translate)	0	0	0	20
BCL to INT. BCL	0	1	0	20
BCL EXTERNAL to ASCII	1	1	0	20
BCL to EBCDIC	0	1	1	20
BINARY (6 bit to 6 bit)	0	0	0	21
EBCDIC (no translate)	0	0	1	22
EBCDIC to ASCII	1	0	1	22

NOTE

On a BINARY operation, bits 37:2 of the IOCW can only have a value of 0 or 1, values of from 0 through 3 are possible for all other operations.

Result Descriptor Format



41019

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
VCK	9:1	VALIDITY CHECK (0281).
RCK	8:1	READ CHECK (0101).
		READ CHECK AND VALIDITY CHECK (0381).
MAE	7:1	MEMORY ACCESS ERROR (0081).

FIELD		DESCRIPTION
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

CARD READER

Operations

READ ALPHA (OP 20)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 characters from the reader. The card reader converts BCL card code to BCL code. BCL Code is converted to BCL Internal Control Code, ASCII, or EBCDIC by translators in the IOM.

READ BINARY (OP 21)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 columns of information from the reader. The contents of each card column are divided into two 6-bit fields. The upper six bits are stored in memory followed by the lower 6-bits. There is no code translation or invalid code detection. Tag field transfers are not compatible with this operator and must not be specified.

READ EBCDIC (OP 22)

Read one card from the card reader. The operation is terminated by reading the specified number of words, or by receiving 80 characters from the reader. The card reader control converts EBCDIC card code to EBCDIC. EBCDIC is stored as received, or is translated to ASCII by the IOM.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

DISK FILE SUBSYSTEM

A disk file subsystem can include from one to eight controls and from one to twenty electronics units. If more than eight controls are used, they must be divided between IOM's so that not more than 8 controls of 1 subsystem are allocated to one IOM. Model 1A-2, 1C-3, and 1C-4 disks may be mixed in the same subsystem but model 2B disks must have a separate subsystem.

A disk file exchange is required when more than two electronics units are used. A 1 x 2 adapter may be used when two E.U.'s are required. A segment size of 180 bytes (equivalent to 240 6-bit characters) is used in the B 7700 systems. The Disk File Control 4 is used to control disk file electronic units for model 1C and 1A-2 disks.

The Disk File Control 5 is used to control disk file electronic units for model 2B disk.

Disk File Exchanges

From 1 to 4 disk file controls can be used to form a multiple control subsystem connected to a single IOM.

The 4 x 10 disk file exchange is used to interface disk file control 4 with Model 1 Disk Electronic Units when multiple controls and/or E.U.'s are required in a subsystem. The maximum combination for this exchange is 4 controls and 10 E.U.'s. The number of E.U.'s may be extended up to 20 by adding the 4 x 20 disk file exchange extension and the appropriate adapters.

The 4 x 20 Disk File Exchange 5 is used to interface Disk File Control 5 with the Model 2B Disk Electronic Units when multiple controls and/or E.U.'s are required in a subsystem. The maximum combination for this exchange is 4 controls and 20 E.U.'s. The number of E.U.'s may be extended up to 8 by adding the 8 x 20 disk file exchange extension and the appropriate adapters.

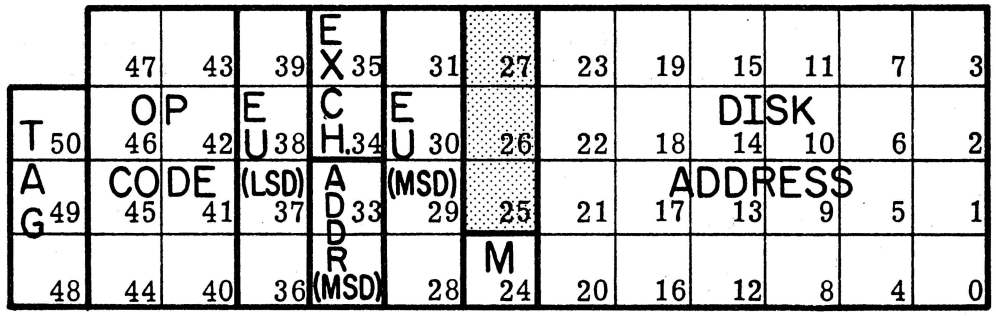
A maximum of 8 controls for a given subsystem are allowed on an IOM. If the 8 x 20 extension is used, more than 1 IOM DF adapter is required and the maximum of 4 controls per DFI adapter must be maintained.

Exchange Modularity

The exchanges are modular by the use of adapters. The modularity of the adapters is such that E.U.'s and controls are added in increments of 2 until the maximum complement is reached.

DISK

CDL Word Format

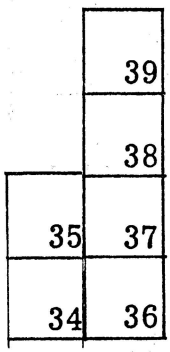


41020

FIELD		DESCRIPTION
TAG	50:3	0 = Selects DFO directly serving EU. 4 = Selects DFO indirectly serving EU.
OP CODE	47:8	50 = WRITE 51 = READ 52 = CHECK 99 = TEST
EU (LSD)	39:4	EU Unit Designate and least significant decimal digit of position on exchange.
EXCH	35:2	Most significant digit of EU position on exchange. For the n X 20 exchange: 00 or 01 For the n X 40 exchange: 00, 01, 10, or 11 The contents of this field are usually equal to bits 29:2.
ADDR (MSD)	33:2	The most significant digit of the disk segment address.
EU (MSD)	31:4	Most significant hexidecimal digit of the EU Unit Designate which is used only in DFO operations.
M	24:1	1 = Maintenance segment operation
DISK ADDRESS	23:24	6 decimal digits of the disk segment address.

NOTE

The following is a clarification of the EU Position on Exchange, ADDR (MSD), and EU (LSD). This information must be specified for all operations.



Most significant digit of EU position on exchange.

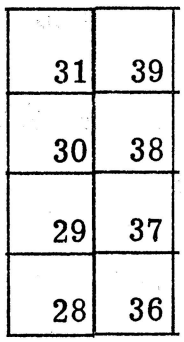
For N x 20 Exchange, 00 or 01.

For N x 40 Exchange, 00, 01, 10 or 11.

Least significant decimal digit - 0...9

NOTE

The following is a clarification of the EU Unit Designate fields, EU (MSD) and EU (LSD), which are used as an index into the Unit Table for DFO operations:



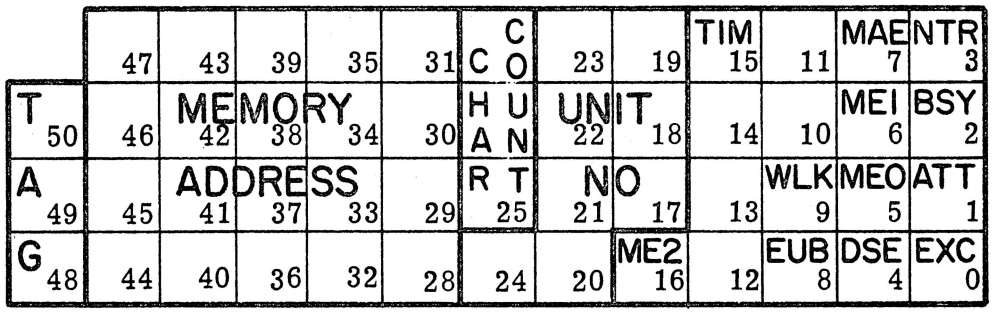
For N x 20 Exchanges, disk unit designates must be 0-9, 16-25, 64-73, 80-89.

For N x 40 Exchanges, disk unit designates must be 32-41, 48-57, 96-105, 112-121.

IOCW Information

OPERATION	IOCW 47	44	43	42	41	CDL OP CODE
WRITE	0	0	0	0	1	50
READ	0	1	0	0	1	51
CHECK	0	1	1	0	1	52

Result Descriptor Format



41022

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
TIM	15:1	TIMEOUT (8001).
WLK	9:1	WRITE LOCKOUT (0201). PARITY ERROR (0281).
EUB	8:1	EU BUSY.
MAE	7:1	MEMORY ACCESS ERROR.
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR.

FIELD		DESCRIPTION
	9:1 BSE	BUSS PARITY ERROR.
	8:1 CPE	CONTROL PARITY ERROR.
	7:1 DPE	DATA PARITY ERROR.
		If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operations

READ (OP 51)

Read a record from the disk file. The operation is terminated by reading the specified number of words.

CHECK (OP 52)

Read a record from the disk file, but do not store any information in memory. If a read error is encountered, the control sets the read error bit in the result descriptor.

WRITE (OP 50)

Write a record on the disk file. The operation is terminated by writing the specified number of words. Incomplete segments are filled out with NULL characters (0000 0000) by the disk file control.

Error Termination

If, during a read or check operation, a segment parity error is detected, the operation is terminated.

If, during read, check or write operations, a failure to access the IOM is encountered, accesses are terminated and result descriptor returned when the segment is completed. Writes are completed with NULL characters (0000 0000), and a result descriptor stored.

The control will time out if the unit addressed sends no information for more than one disk revolution. The time out bit is set in the result descriptor.

Changing zones or changing disks within a storage unit does not result in a time loss of more than 600 microseconds. Changing storage units can result in a time loss equivalent to one disk revolution.

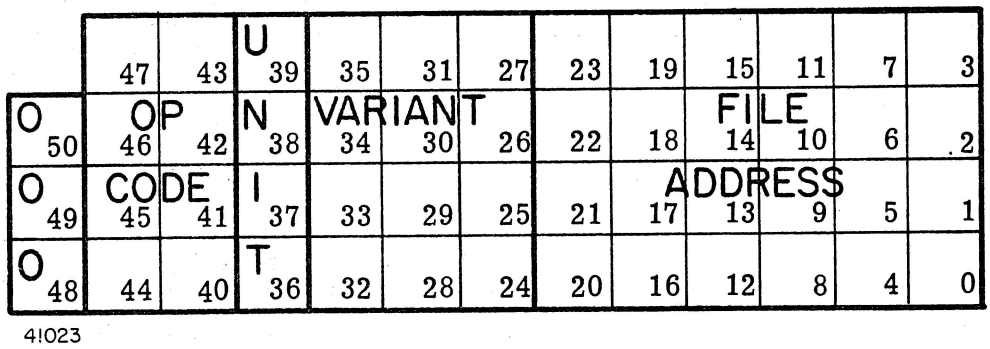
DISK-PACK DRIVE MEMORY SYSTEM

The Magnetic Actuator Disk-Pack Drive Memory Systems are extremely high-speed, modular, random information storage systems. A basic disk-pack drive memory subsystem includes the disk-pack drive controller, dual disk-pack drive, and the interconnecting cables.

The controller acts upon I/O instructions from the IOM, powers the disk-pack drive, and transfers information between disk-pack drives and the IOM. The controller performs the operation specified by the OP code (and variants) of the CDL Word, and, at the completion of the operation, generates a result descriptor which contains operation and/or error status information.

The disk-pack drive controller with single access capabilities may be used with eight disk-pack spindles (four dual drives) in a one-by-eight configuration, or two groups of eight disk-pack spindles (eight dual drives) in a one-by-16 configuration. Selection of each group is determined by a variant in the I/O descriptor. The disk-pack drive controller with dual access capability may be used in a two-by-eight configuration in which the disk-pack drive controller contains two internal control units. This allows the I/O module to execute two simultaneous operations (two reads, two writes, or a read and a write). This configuration can be expanded to a two-by-16 configuration.

CDL Word - General Format



41023

FIELD		DESCRIPTION
OP CODE	47:8	50 = WRITE 51 = READ 56 = INITIALIZE 57 = VERIFY 58 = RELOCATE 59 = ON-LINE TEST 99 = Test

FIELD		DESCRIPTION
		53 = REPORT 54 = UNLOCK Reserved File Protect Memory 55 = CLEAR descriptors.
UNIT	39:4	UNIT DESIGNATE: 0 - F designates disk pack drives 1 - 16.
VARIANT	35:12	Differs for each operation
FILE ADDRESS	23:24	DISK PACK ADDRESS.

NOTE

Due to the complexity of Disk Pack CDL words, illustrations will be provided for each of the possible operations.

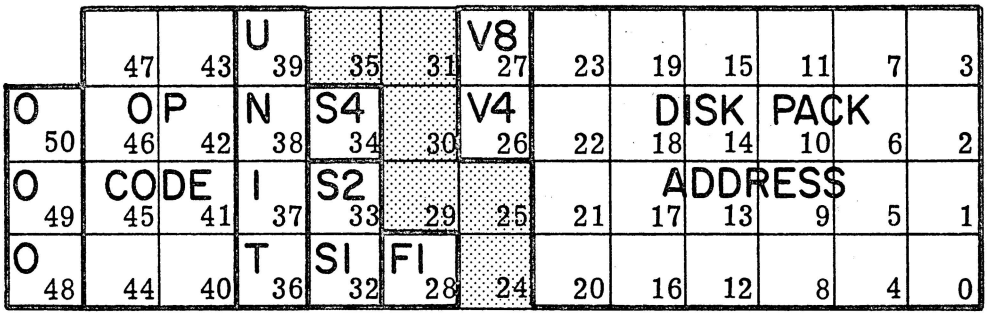
CDL WORD NOTE 1

Unconditional and conditional commands are specified by the S1 bit (32) for a disk pack write, read, initialize, verify, and relocate. The significance of the bit is as follows:

S1 32 : 0 = UNCONDITIONAL COMMAND. If a seek (head movement) is required, the disk pack controller will initiate a seek operation and will execute the original command following completion of the seek.

1 = CONDITIONAL COMMAND. If a seek is not required (no head movement required to reach the specified address), the command is executed. If a seek is required, the disk pack controller will initiate a seek operation and disregard the remainder of the original command, returning a "seek initiated" result descriptor. If the drive is seeking when selected, execution of the original command is not initiated and a "drive seeking" condition is reported in the result descriptor.

CDL Word Format - Write

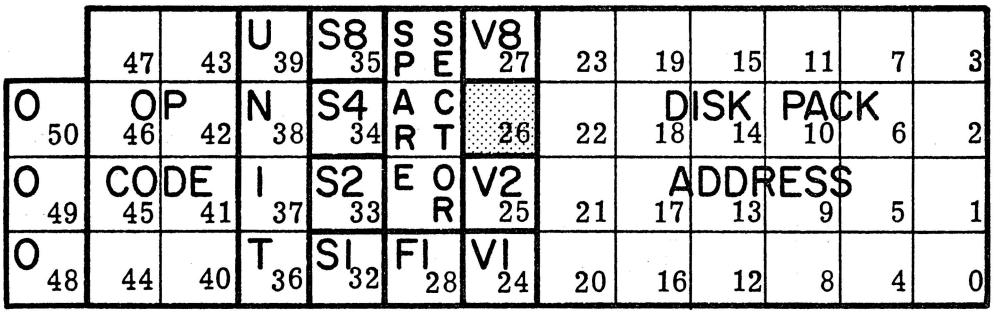


41024

FIELD		DESCRIPTION
OP CODE	47:8	50 = WRITE
UNIT	39:4	Unit Designate (U = 0-F designates disk pack drives 1-16).
S4	34:1	1 = Controller automatically executes parity check on all sectors written upon completion of the write operation.
S2	33:1	1 = The automatic restore function is disabled following a seek error condition.
S1	32:1	0 = Unconditional write. See note 1. 1 = Conditional write. See note 1.
F1	28:1	0 = Select multi-sector format. 1 = Select full-track format.
V8	27:1	1 = Enable EBCDIC-ASCII translator (valid for B 2500/ B 3500, B 4500).
V4	26:1	Reserved for FPM.
DISK PACK ADDRESS	23:24	Disk pack address.

DISK PACK

CDL Word Format - Read

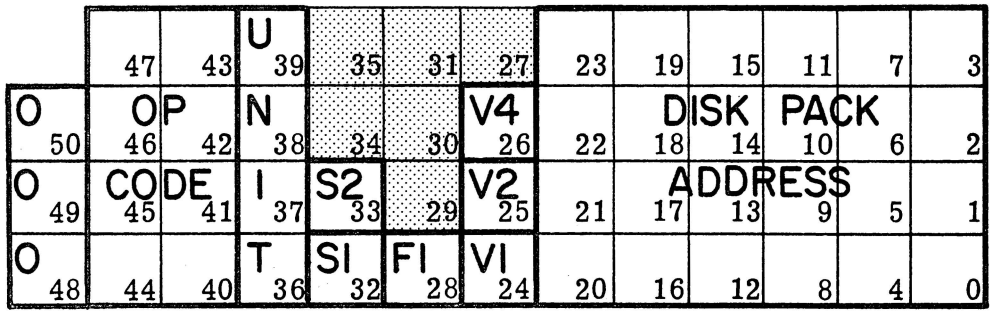


41025

FIELD		DESCRIPTION
OP CODE	47:8	51 = Disk pack read.
UNIT	39:4	Unit designate (U= 0-F designates disk pack drives 1 - 16).
S8	35:1	1 = Error correction is disabled.
S4	34:1	0 = Initiate a normal read. 1 = A binary address (3 8-bit bytes plus parity is read into memory. This address is the actual address field recorded on the disk pack for the sector specified by the disk pack address in the CDL word as altered by the spare sector variant. In the case of a previously designated unusable sector, the address returned will contain all ones.
S2	33:1	1 = The automatic restore function is disabled following a seek error condition.
S1	32:1	0 = Unconditional read. See note 1. 1 = Conditional read. See note 1.
SPARE SECTOR	31:3	If S4=1 then a value of 1 - 5 indicates a spare sector as follows: Model 215 DPD: Sectors 28 - 32 respectively on designated tracks. Model 225 DPD: Sectors 55 - 59 respectively on designated tracks.

FIELD		DESCRIPTION
F1	28:1	0 = Select multi-sector format. 1 = Select full-track format.
V8	27:1	1 = Enable EBCDIC-ASCII translator (valid for B 2500/B 3500, and B 4500 only).
V2, V1	26:2	Reserved for FPM.
DISK PACK ADDRESS	23:24	Disk pack address.

CDL Word Format - Initialize

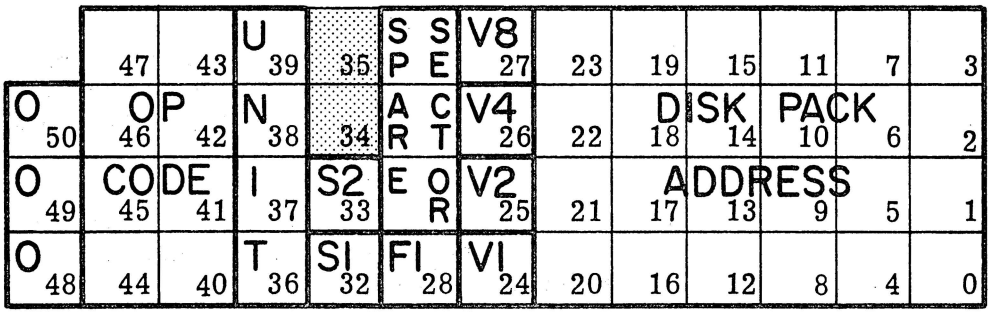


41026

FIELD		DESCRIPTION
OP CODE	47:8	56 = INITIALIZE
UNIT	39:8	Unit designate (U = 0-F designates disk pack drives 1-16).
S2	33:1	1 = The automatic restore function is disabled following a seek error condition.
S1	32:1	0 = Unconditional initialize. See note 1. 1 = Conditional initialize. See note 1.
F1	28:1	0 = Select multi-sector format. 1 = Select full-track format.
V4	26:1	0 = Write all zero test pattern into each sector. 1 = Write provided 16 bit data pattern repeatedly into each sector.

FIELD		DESCRIPTION
V2	25:1	0 = If V1 is also zero, initialize the entire pack. 1 = Initialize the designated track.
V1	24:1	0 = If V2 is also zero, initialize the entire pack. 1 = Initialize the designated cyclinder.
DISK PACK ADDRESS	23:24	Disk pack address.

CDL Word Format - Verify

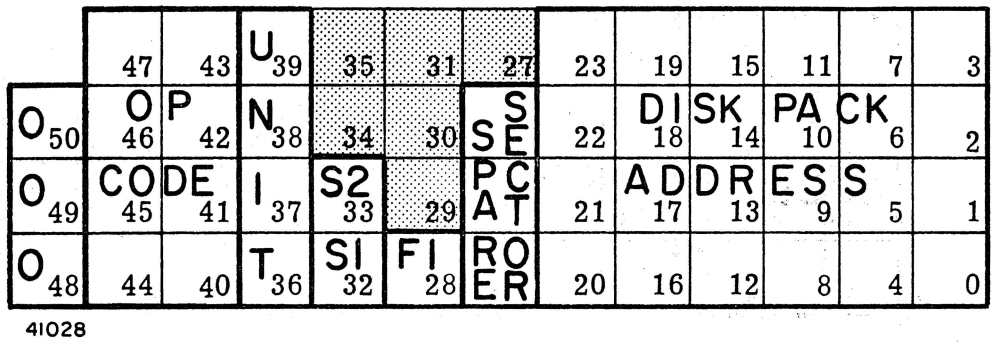


41027

FIELD		DESCRIPTION
OP CODE	47:8	57 = VERIFY
UNIT	39:8	Unit designate (U = 0-F designates disk pack drives 1-16).
S2	33:1	1 = The automatic restore function is disabled following a seek error condition.
S1	32:1	0 = Unconditional verify. See note 1. 1 = Conditional verify. See note 1.
SPARE SECTOR	31:3	Value of 1-5 indicates a spare sector as follows: Model 215 DPD: Sectors 28-32 respectively on designated tracks. Model 225 DPD: Sectors 55-59 respectively on designated tracks.

FIELD		DESCRIPTION
F1	28:1	0 = Select multi-sector format. 1 = Select full-track format.
V8	27:1	0 = If V4 = 0, the data fields of all sectors are checked for correct parity (error detection and count check fields). 1 = If V4 = 0 then verify the all zero test data pattern written during initialize.
V4	26:1	0 = If V8 = 0, the data fields of all sectors are checked for correct parity (error detection and count check fields). 1 = Data bits are verified by comparison with the last 16-bit data pattern used on an INITIALIZE command with V4 = 1 (the data pattern is stored in the controller).
V2	25:1	0 = If V1 = 0, verify the entire disk pack and terminate on the first error encountered (only one address may be reported). 1 = Verify the designated track and terminate on the first error encountered (only one address may be reported).
V1	24:1	0 = If V2 = 0, verify the entire disk pack and terminate on the first error encountered (only one address may be reported). 1 = Verify the designated cylinder and terminate on the first error encountered (only one address may be reported).
DISK PACK ADDRESS	23:24	Disk pack address.

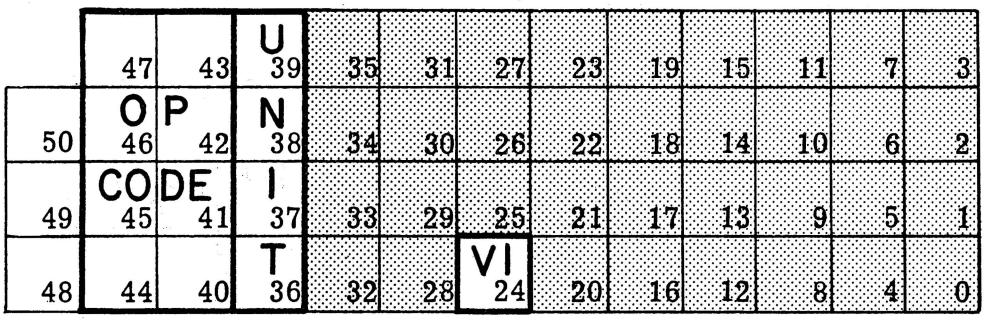
CDL Word Format - Relocate



41028

FIELD		DESCRIPTION
OP CODE	47:8	58 = RELOCATE
UNIT		Unit designate (U = 0-F designates disk pack drives 1-16).
S2	33:1	1 = The automatic restore function is disabled following a seek error condition.
S1	32:1	0 = Unconditional relocate. See note 1. 1 = Conditional relocate. See note 1.
F1	28:1	0 = Select multi-sector format. 1 = Select full-track format.
SPARE SECTOR	26:3	A value of 1-5 indicates a spare sector as follows: Model 215 DPD: Sectors 28-33 respectively on designated tracks. Model 225 DPD: Sectors 55-59 respectively on designated tracks.
DISK PACK ADDRESS	23:24	Disk pack address.

CDL Word Format - Test and On-line Test



41029

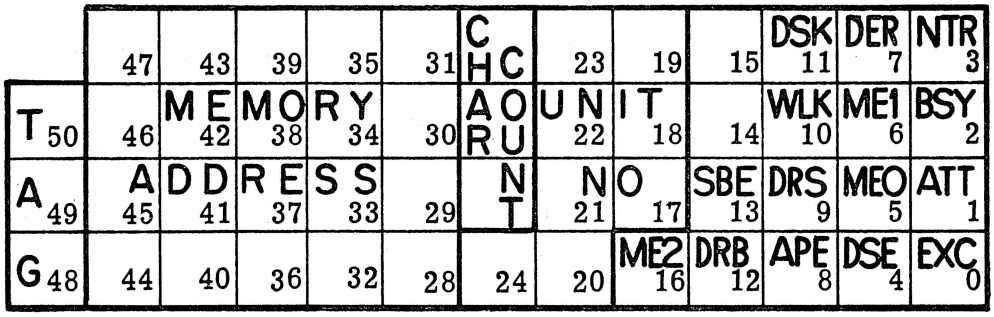
FIELD		DESCRIPTION
OP CODE	47:8	59 = ON-LINE TEST. 99 = TEST.
UNIT	39:4	Unit designate (U = 0 - F designates disk pack drives 1 - 16).
VI	24:1	1 = Selected drive is taken off-line for disk pack removal.

IOCW Information

OPERATION	IOCW 47	44	43	42	41	40	CDL OP CODE
Write	0	0	0	0	1	0	50
Read	0	1	0	0	1	Ø	51
Initialize	0	0	0	0	1	0	56
Verify	0	1	0	0	1	Ø	57
Relocate	0	0	1	0	1	0	58
On-line test	0	0	0	0	0	0	59
Test	0	0	0	0	0	0	99

Ø = Either zero or one.

Result Descriptor Format



41030

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
SBE	13:1	SINGLE BIT ERROR. Indicates that single bit error correction has occurred (2801).
DRB	12:1	DRIVE BUSY (1001). SPEED ERROR (INITIALIZE) (1081). ADDRESS POSITION ERROR (1101). TRANSMISSION PARITY ERROR (1181). SEEK TIME-OUT (1801).
DSK	11:1	DRIVE SEEK ERROR (0801). INFO. BUSS WRITE PARITY ERROR (0881). SECTOR TIME-OUT (0901). SEEK INITIATED (0A01).
WLK	10:1	WRITE LOCK-OUT (0501).
DRS	9:1	DRIVE SEEKING (0201). 1st R/D AFTER DRIVE IS READY (0301).
APE	8:1	ADDRESS PARITY ERROR (0101). MEMORY ACCESS ERROR (0181).

FIELD		DESCRIPTION
DER	7:1	DATA ERROR or MEMORY PARITY ERROR (0081). CONTROL CLEARED DURING OPERATION (0089).
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit is not set (DSE), bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY (0009). DRIVE NOT READY (0009). NON-EXISTENT ADDRESS (0009). UN-SAFE CONDITION (0009). F.P.M. OP CODE (0009).
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

DISK PACK

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operators

WRITE (OP CODE 50)

Information is written from memory onto disk at the designated file address. Partial sector writes result in a fill of bit zeros in the remainder of the sector.

READ (OP CODE 51)

Information is read from the selected file address into the main memory address. A partial sector read results in the termination of data transfer, but control release does not occur until completion of the total sector read. The controller times out if the read operation is not completed.

INITIALIZE (OP CODE 56)

The controller writes addresses and gaps on all tracks, starting with an index on the track of the designated address. The controller may also write a test pattern consisting of all zeros.

VERIFY (OP CODE 57)

The controller reads from the disk-pack and checks for address errors and information parity errors. The verification of disk sectors begins with the first sector following the index on the track specified by the disk file address and continues through the entire track, cylinder, or disk-pack. The sector positions are verified by counting from the index on each track.

The BCD file address (normal or spare) containing the detected error(s) is reported back between the begin and end memory address (or between the begin address plus 6 bytes and the end address if V_4 is 1). Address parity is checked for all relocated sectors.

RELOCATE (OP CODE 58)

The controller flags the sector address field of the unusable sector with an error configuration and rewrites this address in the spare sector. The original address field and the relocated address field are located by counting from the index on each track.

NOTE

An error configuration is one byte of binary "ones" with clock pulses omitted, followed by one byte of binary "ones" (with clock pulses), followed by one byte of binary "ones" with clock pulses omitted.

The error configuration is written into the address field of the designated sector. The standard test data pattern is written into the relocated sector data field. If full-track format is specified, the address is flagged but no relocation takes place.

TEST (OP CODE 99)

The controller returns an appropriate result descriptor indicating the type of drive accessed, the peripheral control type, and (in a shared system configuration), the number of the assigned processor.

Exception Conditions

SINGLE BIT ERROR CORRECTION (SBE)

A single bit error in a 90-byte data block is detected and corrected during a read operation.

DISK-PACK DRIVE BUSY (TIME-OUT) (DRB)

The operation is terminated if the drive accessed is being used by another controller as an operation is initiated and the drive does not become available within 50 milliseconds.

SPEED ERROR (1081)

The operation is terminated if there is a failure to write a full track of information between index pulses during an initialize operation.

ADDRESS POSITION ERROR (VERIFY) (1101)

A valid address is found at the wrong position relative to the index during a verify operation.

DISK PACK

TRANSMISSION PARITY ERROR (1181)

The operation is terminated at the end of the sector being processed if a parity error is detected while data is being transmitted during a write operation.

SEEK TIME-OUT (1801)

The controller terminates the operation when an accessed drive fails to complete the initiated seek within one second.

SEEK ERROR (DSK)

Bit 8 is set and the operation is terminated if a seek is initiated which positions the heads over the wrong cylinder during a read or write operation (as verified by the controller reading the address field from the track on the cylinder in question). Bit 8 is set and the operation is not terminated if a seek is initiated which positions the heads over the wrong cylinder during a verify operation.

SECTOR TIME-OUT (0901)

The operation is terminated if a specified sector address, or spare sector to which that address has been relocated, cannot be found during a read, write, or relocate operation.

WRITE-LOCKOUT (0501)

A write, initialize, or relocate operation is not initiated if the drive to be accessed is in the write lockout state. (The operator failed to press the WRITE ENABLE button on the drive unit.) On a test operation, if the drive is detected to be in a write lockout state, the operation will be executed and bits 5 and 7 will be set in the test result descriptor.

FIRST ACTION (0301)

At the beginning of any operation, if it is detected that this is the first command to a drive that has been powered up, the operation is not initiated.

MEMORY INTERFACE PARITY ERROR (0081)

The controller terminates the operation at the end of the sector being processed if an information parity error is detected on the memory or I/O Processor interface.

CONTROL CLEARED (0089)

If the disk pack controller is cleared during any operation, the operation in process is terminated. The termination is instantaneous and if the controller was writing to the disk at the time Clear was detected, it could destroy the information on the track.

General Information

Each disk pack contains 11 disks and 20 recording surfaces, each surface accessed by an individual arm from the actuator. Each disk surface contains 406 tracks. (See figure IV-3-1).

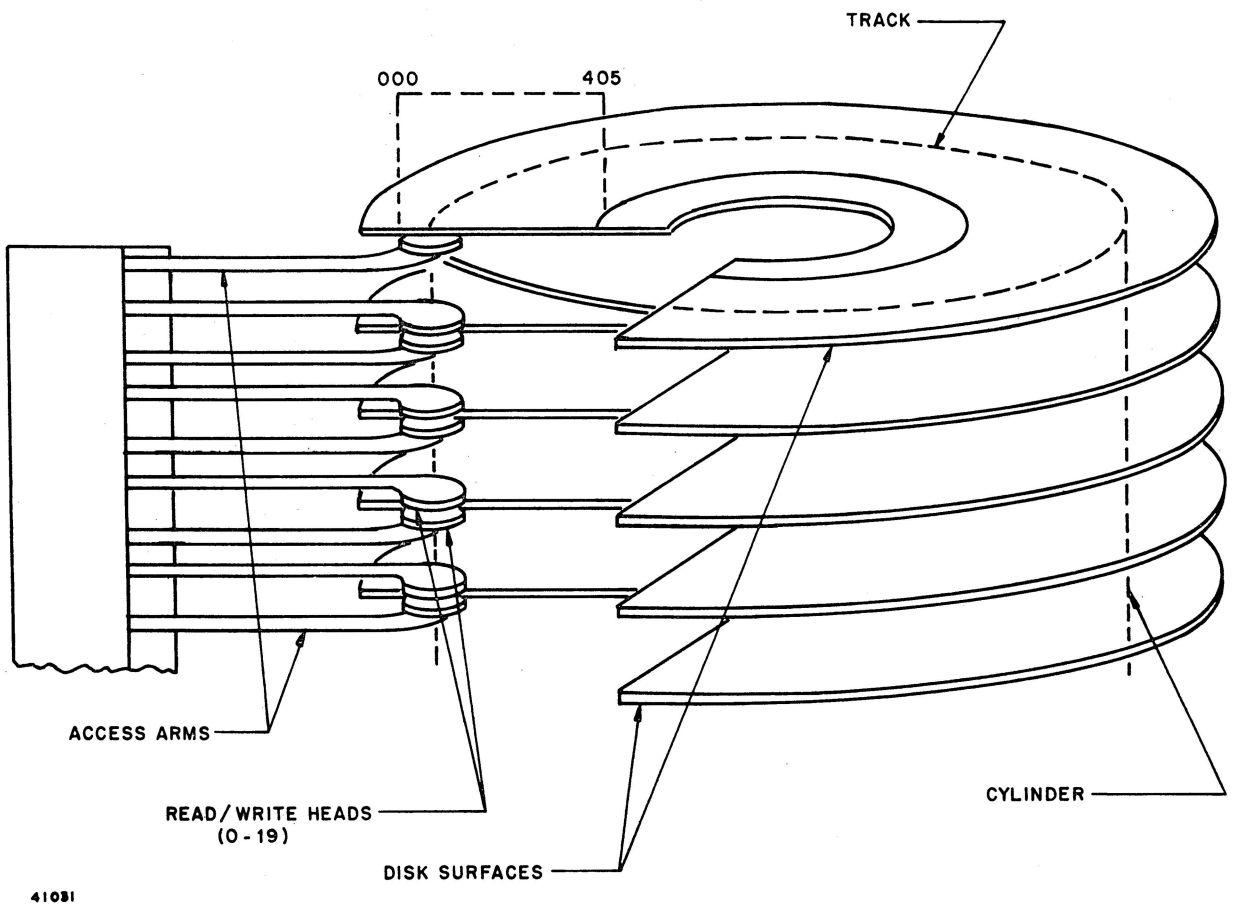


Figure IV-3-1. Disk-Pack Recording Surfaces

The data transfer is bit-serial. The maximum byte capacity, transfer rate, and other pertinent information for the various disk-pack subsystems are presented in table IV-3-1.

Multisector-Per-Track (Standard) Format

There are 33 sectors/track in the multisector format (table IV-3-2), and information within the track is located by sector address (pack address). When a pack is initialized, addresses are written sequentially, sector by sector, throughout each cylinder in the disk-pack.

The segment size of Burroughs magnetic actuator disk-pack drives can be selected by the user as either 180-bytes or full track of 7470 or 14,940 bytes. When the 180-byte segment or multi-sector mode is selected, inter-record gaps are introduced and, accordingly, as on all disk-packs where the full-track facility is not used, the 121,000,000-byte capacity will be reduced to 95,500,000 bytes; the 242,000,000 byte capacity will be reduced to 174,400,000 bytes.

Spare Sectors

Five sectors are reserved for each cylinder on track 00 (surface under head 00) for relocating data from other sectors in the disk-pack which have been designated as unusable. Data will be relocated between tracks in the same cylinder only.

The method of locating spare sectors so that data is not lost is a software function. If more than five unusable sectors are encountered in one cylinder, the additional unusable sectors are disposed of by software via the XP message.

One format is used to address all sectors which are not spares. This format uses a continuous 6-digit BCD address to designate contiguous sectors beginning at sector 0 (first sector after index) on head 0, cylinder 0 and continuing by sector, head, and cylinder in that order. The BCD file address may be related to the actual disk-pack cylinder head and sector by the following equalities:

a. Burroughs Multi-Sector Format (figure IV-3-2):

   For head 0: (655*cylinder) + sector = file address:
   For other heads: (655*cylinder) + (33*head) + (sector-5) = file address.

b. Single Sector/Track Format (figure IV-3-3):

   For all heads: (20*cylinder) + head - file address.

The other format is the single sector per track format (table IV-3-3). In figure IV-3-3 the entire track is shown first, followed by an enlargement of the data and address fields. There are no spares in this format.

Table IV-3-1. Disk-Pack Subsystem Characteristics

DISK PACK DRIVE STYLE NO.	DESCRIPTION	AVERAGE ACCESS TIME (MS)	AVERAGE LATENCY (MS)	STORAGE CAPACITY PER DISK-PACK DRIVE		DATA TRANSFER RATE	PACK DATA		
				MULTI-SECTOR MODE*	FULL TRACK MODE*		MAX. RECORDING DENSITY (BPI)	TRACK DENSITY (TPI)	DISK-PACK STYLE NO.
B 9484-3	Dual drive with single access disk-pack drive controller (B 7380-1)	30	12.5	95.5	121.0	312.5 KB	2200	200	B 9974-1
B 9485-3	Dual drive with dual access disk-pack drive controller (B 7380-2)	30	12.5	95.5	121.0	312.5 KB	2200	200	B 9974-1
B 9486-3	Dual drive add on without disk-pack drive controller	30	12.5	95.5	121.0	312.5 KB	2200	200	B 9974-1
B 9484-4	Dual drive with single access disk-pack drive controller (B 7383-1)	30	12.5	174.4	242.0	625.0 KB	4400	200	B 9974-4
B 9485-4	Dual drive with dual access disk-pack drive controller (B 7383-2)	30	12.5	174.4	242.0	625.0 KB	4400	200	B 9974-4
B 9486-4	Dual drive add on without disk-pack drive controller	30	12.5	174.4	242.0	625.0 KB	4400	200	B 9974-4
B 9486-45	Single drive add on without disk-pack drive controller	30	12.5	87.2	121.0	625.0 KB	4400	200	B 9974-4

*Million eight-bit bytes.

Table IV-3-2. Disk-Pack File Addresses (Burroughs Multi-Sector Format)

CYL. NO.	HEAD 0 SECTOR	HEAD 0 FILE ADDRESS	HEAD 1 SECTOR	HEAD 1 FILE ADDRESS	HEAD 2 SECTOR	HEAD 2 FILE ADDRESS	HEAD 18 SECTOR	HEAD 18 FILE ADDRESS	HEAD 19 SECTOR	HEAD 19 FILE ADDRESS
0	0	000000	0	000028	0	000061	0	000589	0	000622
	1	000001	1	000029	1	000062	1	000590	1	000623
	2	000002	2	000030	2	000063	2	000591	2	000624
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	27	000027	27	000055	27	000088	27	000616	27	000649
Spare Sectors	28	*Addr+1	28	000056	28	000089	28	000617	28	000650
	29	Addr+2	29	000057	29	000090	29	000618	29	000651
	30	Addr+3	30	000058	30	000091	30	000619	30	000652
	31	Addr+4	31	000059	31	000092	31	000620	31	000653
	32	Addr+5	32	000060	32	000093	32	000621	32	000654
1	0	000655	0	000683	0	000716	0	001244	0	001277
	1	000656	1	000684	1	000717	1	001245	1	001278
	2	000657	2	000685	2	000718	2	001246	2	001279
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	27	000682	27	000710	27	000743	27	001271	27	001304
Spare Sectors	28	Addr+1	28	000711	28	000744	28	001272	28	001305
	29	Addr+2	29	000712	29	000745	29	001273	29	001306
	30	Addr+3	30	000713	30	000746	30	001274	30	001307
	31	Addr+4	31	000714	31	000747	31	001275	31	001308
	32	Addr+5	32	000715	32	000748	32	001276	32	001309
405	0	265275	0	265303	0	265336	0	265864	0	265897
	1	265276	1	265304	1	265337	1	265865	1	265898
	2	265277	2	265305	2	265338	2	265866	2	265899
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.
	27	265302	27	265330	27	265363	27	265891	27	265924
Spare Sectors	28	Addr+1	28	265331	28	265364	28	265892	28	265925
	29	Addr+2	29	265332	29	265365	29	265893	29	265926
	30	Addr+3	30	265333	30	265366	30	265894	30	265927
	31	Addr+4	31	265334	31	265367	31	265895	31	265928
	32	Addr+5	32	265335	32	265368	32	265896	32	265929

* Address of last non-spare sector, plus number of spare (1→5) specified in F variant.

Table IV-3-3. Disk-Pack File Addresses (Single Sector/Track Format)

CYL. NO.	HEAD 0	HEAD 1	HEAD 2	HEAD 18	HEAD 19
	FILE ADDRESS	FILE ADDRESS	FILE ADDRESS	FILE ADDRESS	FILE ADDRESS
0	000000	000001	000002	000018	000019
1	000020	000021	000022	000038	000039
2	000040	000041	000042	000058	000059
3	000060	000061	000062	000078	000079
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
405	008100	008101	008102	008118	008119

Spare sectors (sectors 28 through 32) on surface 0 of each cylinder are addressed with the following six-digit patterns:

a. Two digits of undigit 5's (F) to delimit the field.

b. Three BCD digits to indicate the cylinder on which the spare is located.

c. One BCD digit indicating spare sector 1 through 5 as needed.

UNDIGIT 5	SPARE SECTOR	UNDIGIT 5

TRACK FORMAT (MULTI-SECTOR MODE)

COMPLETE TRACK

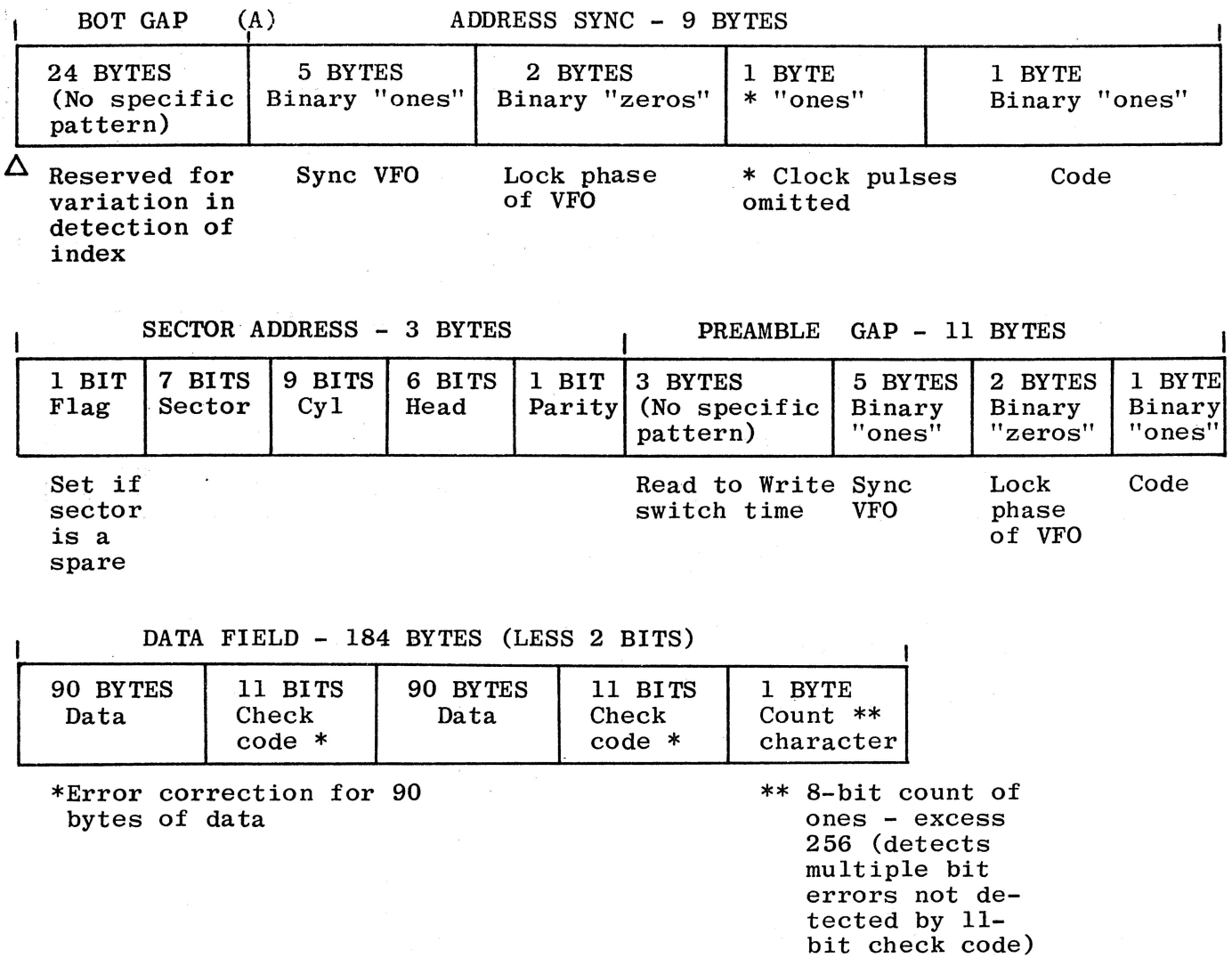


Figure IV-3-2. Standard Format (Sheet 1)

TRACK FORMAT (MULTI-SECTOR MODE) (Cont)

POSTAMBLE GAP - 24 BYTES		(B) EOT GAP - 42 to 353 BYTES	
9 BYTES Binary "ones"	15 BYTES Binary "ones"	0 to 311 BYTES (No specific pattern)	42 BYTES Spare
Reserved for space variation in writing data	Reserved for Write to Read switch time	Reserved for speed variation (± 2%)	

Note: (A) through (B) is repeated 32 times except no postamble on last sector.

ADDRESS FIELD OF SECTOR FLAGGED AS AN ERROR SECTOR (PLUS LAST TWO BYTES OF ADDRESS SYNC)

(PARTIAL) ADDRESS SYNC			
1 BYTE *Binary "ones"	1 BYTE Binary "zeroes"	1 BYTE *Binary "ones"	2 BYTES Binary "zeroes"
*Clock pulses omitted	Code byte changed to "zeroes"		

ADDRESS FIELD OF SECTOR WITH RELOCATED SECTOR INFORMATION

SECTOR ADDRESS FIELD				
1 BIT Flag	7 BITS Sector	9 BITS CYL	6 BITS Head	1 BIT Parity
Bit is a "zero"				

ADDRESS FIELD OF UNUSED SPARE SECTOR

SECTOR ADDRESS FIELD				
1 BIT Flag	7 BITS Sector	9 BITS CYL	6 BITS Head	1 BIT Parity
Bit is a "one"				

Figure IV-3-2. Standard Format (Sheet 2)

TRACK FORMAT (SINGLE SECTOR/TRACK FORMAT)

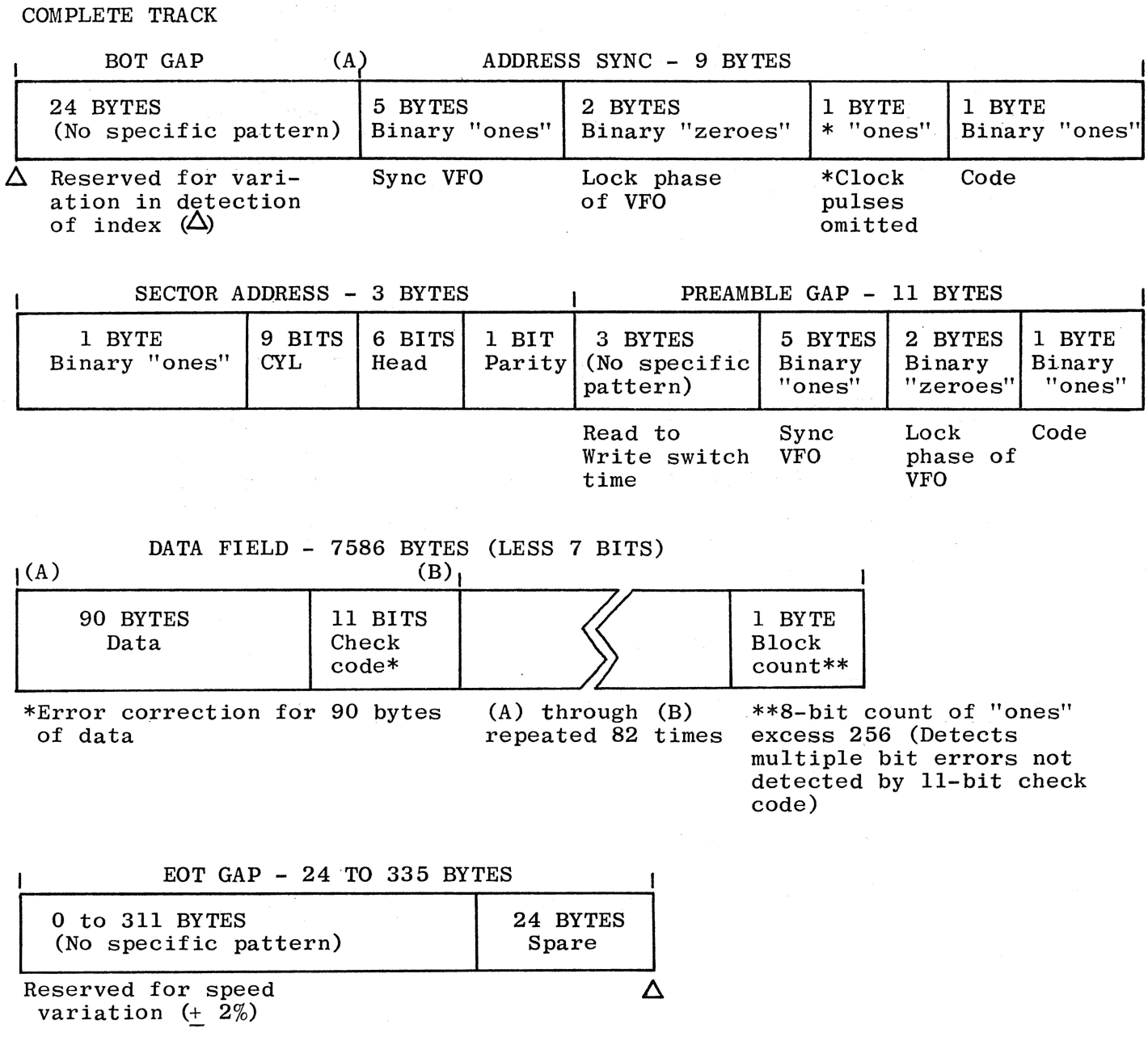


Figure IV-3-3. Single Sector For Track Format

LINE PRINTER

Printer Control

Two basic line printers are available for use on the B 7700 system. The B 9242-11 prints 860 lines per minute (LPM) and the B 9243-11, 1100 LPM. Both printers are available with either 120 or 132 print positions. OCR printers are also available with printing speeds of 725 LPM and 900 LPM. All printers have vertical skipping and end-of-page formatting controlled by a punched paper tape and include the forms self-align feature. The B 7240 Line Printer Control connects the printer to the I/O Module. Translators in the I/O Module convert internal BCL, ASCII, or EBCDIC into external BCL for transmission to the printer control.

CDL Word Format

	47	43	S 39	U 35	S 31	C 27	23	19	15	11	7	3
O_{50}	OP 46	42	PA 38	N 34	KI 30	HA 26	22	18	14	10	6	2
O_{49}	CODE 45	41	CE 37	I 33	P 29	NN 25	21	17	13	9	5	1
O_{48}	44	40	36	T 32	TO 28	EL 24	20	16	12	8	4	0

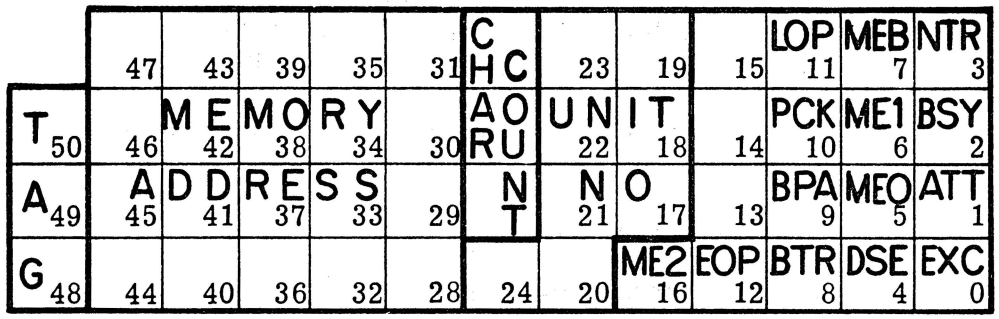
41032

FIELD		DESCRIPTION
OP CODE	47:8	10 = WRITE 11 = SKIP 99 = Test
SPACE	39:4	Space: 0, 1, or 2
UNIT	35:4	Unit Designate for printer exchange: 0 or 1.
SKIP TO CHANNEL	31:8	Skip to channel: 01 - 11 00 = NO SKIP

IOCW Information

OPERATION	IOCW 47	43	42	41	CDL OP CODE
BCL EXTERNAL (No translate)	0	0	0	0	10
BCL (From INT. BCL)	0	0	1	0	10
BCL (From ASCII)	1	0	1	0	10
BCL (From EBCDIC)	0	0	1	1	10
SPACE	0	1	0	0	11
SKIP	0	1	0	0	11

Result Descriptor Format



41034

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and MEO are encoded to report various errors detected by the MIU.
EOP	12:1	END OF PAGE (1001).
LOP	11:1	LOW PAPER (0801).
PCK	10:1	PRINT CHECK/CODE PARITY (0481)
BPA	9:1	BUFFER PARITY ERROR (0281)
BTR	8:1	BIT TRANSFER ERROR (0181)
MEB	7:1	Miscellaneous error bit which is used in conjunction with PCK, BPA, and BTR.
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and MEO are encoded to report various errors detected by the MIU.
MEO	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and MEO are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table:

15:1	DVE	DEVICE DETECTED ERROR.
14:1	CDE	CDL CHARACTER ERROR.
12:1	CME	COMBINATION ERRORS.
11:1	CTE	COUNTER ERROR.
10:1	IFE	INTERFACE ERROR.

FIELD		DESCRIPTION
		9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	LIA ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operation

PRINT (OP 10)

Print one line on the line printer. The length of the line is determined by the number of printer columns (120 or 132) or by printing the specified number of words. Spacing or skipping takes place after printing.

When the 6-bit frame size is selected and control word bit 47 is false and control word bit 42 is true, BCL Internal code is converted to BCL code by a translator in the IOM. When the 8-bit frame size is selected and control word bit 42 is true, EBCDIC is converted to BCL code; when control word bits 47 and 42 are true, ASCII is converted to BCL code.

LINE PRINTER

SPACE (OP 11)

Space as specified by CDL word bits 37:2 (bits 31:8 must be zero).

00	No space
01	Single space
1x	Double space

SKIP (OP 11)

Skip as specified by CDL word bits 31:8 (skip channels 01-11).

TEST (OP 99)

Test the status of the unit and return a result descriptor.

Error Termination

The end-of-page bit is set if the current descriptor does not specify a skip and there is a punch in channel 12 of the forms loop. The current descriptor is executed.

BUFFERED PRINTER CONTROL NO. 2

A result descriptor is returned when the printer buffer has been loaded. The print check error bit refers to the line of print associated with the prior descriptor.

MAGNETIC TAPE

Tape Subsystem

A magnetic tape subsystem can include up to four tape controls and up to sixteen magnetic tape transports. Within a single tape subsystem, all magnetic tape transports must be used at the same speed; and all controls must be of the same type.

TAPE EXCHANGES, FREE-STANDING UNITS

A magnetic tape exchange is required when more than one control is used or more than 6 magnetic tape transports are used.

Magnetic Tape Exchange No. 2

This magnetic tape exchange provides the facility for either of two compatible magnetic tape controls to communicate with any of ten magnetic tape transports (all 7-track or all 9-track):

An exchange extension adapter is required for each group of two tape units. An adapter/magnetic tape exchange is required for each ready-status cable (1 or 2) that is connected to an IOM.

Magnetic Tape Exchange No. 1

This magnetic tape exchange provides the facility of communication between any of four NRZ magnetic tape controls and any of 16 free standing magnetic tape transports.

The basic exchange is equipped for one control unit and no tapes.

TAPE EXCHANGE, CLUSTER UNITS

A cluster is not connected to an external exchange. It can include, as an option, 2x adapter(s) enabling it to operate with two controls. A 2X master cluster provides 2 x 4 capability. The addition of a 2X slave provides 2 x 8 capability.

The controls of the same types (number of heads, and NRZ or PE) can operate with one master cluster (and its matching slave).

2 7-track NRZ, or

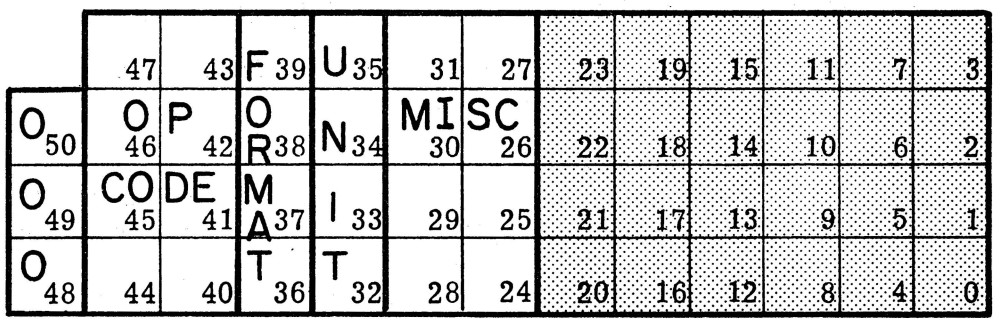
2 9-track NRZ, only, or

2 9-track PE only.

Use, if any, of a cluster with two types of stations with the matching controls is permitted. The NRZ controls and stations must be treated as a separate sub-system from phase-encoded (P.E.) stations and controls associated with the same cluster (i.e., separate groups of unit designate numbers).

MAG TAPE

CDL Word Format



41035

FIELD		DESCRIPTION
OP CODE	47:8	01 = REWIND 02 = READ FORWARD 03 = READ BACKWARD 04 = ERASE 06 = WRITE 08 = SPACE FORWARD 09 = SPACE BACKWARD 99 = TEST
FORMAT	39:4	0 = 800 BPI, Even parity (Note 1) 1 = 800 BPI, Odd parity 2 = 556 BPI, Even parity (Note 2) 3 = 556 BPI, Odd parity (Note 2) 4 = 200 BPI, Even parity (Note 1) 5 = 200 BPI, Odd parity 7 = 1600 BPI, Odd parity (Note 3) 8 = Unit selected density, even parity. 9 = Unit selected density, odd parity.
UNIT	35:4	Unit designate for the tape exchange.

NOTES

1. It is not possible to have even parity on 9-track tape.
2. 556 BPI is only used on 7-track tape.
3. 1600 BPI is only used with Magnetic Tape Control No. 5.

FIELD		DESCRIPTION
MISC	31:8	REWIND, ERASE, or TEST: = 0
	31:8	SPACE FORWARD or SPACE BACKWARD: = the number of records.
	31:4	READ or WRITE: 1, 2, or 3 = Maintenance variants on READ 2 = Write a tape mark record on a WRITE. 4 = EBCDIC/BCL translation on 7-track tape only.
	27:4	WRITE: = 0
	27:4	READ on 9-track PE tape: If bit 27 = 1, then the other 3 bits (26:3) specify tracks to be corrected via the CRC adapter.

IOCW Information

OPERATION	IOCW 47	44	43	42	41	39	CDL OP CODE
Read BINARY (6-bit to 6-bit)	0	1	0	0	0	0/1	02/03
7 Read BCL into INT. BCL	0	1	0	1	0	0/1	02/03
Read BCL into EBCDIC	0	1	0	1	1	0/1	02/03
T Read BCL into ASCII	1	1	0	1	0	0/1	02/03
R Write BINARY (6-bit to 6-bit)	0	0	0	0	0	x	06
A Write BCL from INT. BCL	0	0	0	1	0	x	06
C Write BCL from EBCDIC	0	0	0	1	1	x	06
K Write BCL from ASCII	1	0	0	1	0	x	06
ERASE	0	0	1	x	0	x	04
9 Read (8-bit to 8-bit)	0	1	0	0	1	0/1	02/03
Read EBCDIC into ASCII	1	1	0	0	1	0/1	02/03
T Read ASCII into EBCDIC	1	1	0	1	1	0/1	02/03
R Write (8-bit to 8-bit)	0	0	0	0	1	x	06
A Write EBCDIC from ASCII	1	0	0	0	1	x	06
C Write ASCII from EBCDIC	1	0	0	1	1	x	06
K ERASE	0	0	1	x	1	x	04
B REWIND	0	0	1	x	x	1	01
O SPACE	0	1	1	x	x	0/1	08/09
T Write Tapemark	0	0	x	x	x	x	06
H							

x = Not used.

Result Descriptor Format

	47	43	39	35	31	C	23	19	TIM 15	MEB 11	MAE 7	NTR 3
T 50	46	ME 42	MO 38	RY 34	30	CO HU	22	IT 18	REW 14	SBK 10	MEI 6	BSY 2
A 49	45	AD 41	DR 37	ESS 33	29	AN RT	N 21	O 17	NPO 13	WLK 9	MEO 5	ATT 1
G 48	44	40	36	32	28	24	20	ME2 16	12	EOT 8	DSE 4	EXC 0

41036

FIELD		DESCRIPTION
ME2	16:1	MEMORY RELATED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
TIM	15:1	BLANK TAPE TIMEOUT (8001). BEGINNING OF TAPE & BLANK TAPE TIMEOUT (8101)
REW	14:1	REWINDING (4001).
NPO	13:1	NON-PRESENT OPTION (2001).
MEB	11:1	Miscellaneous error bit which is used in conjunction with SBK, WLK, and EOT. On a tape parity error bits 7, 10, and 11 are set. Test bits 10 and 11, define the tape density switch setting as follows:

Bit 11	Bit 10	DENSITY
0	0	800
1	0	200
0	1	555
1	1	1600

FIELD		DESCRIPTION
SBK	10:1	SHORT BLOCK (0401) PARITY ERROR (0C81) PARITY ERROR & END OF TAPE (0D81) END OF FILE & PARITY ERROR (0E81)
WLK	9:1	WRITE LOCKOUT or END OF FILE (0201).
EOT	8:1	END OF TAPE or BEGINNING OF TAPE (0101).
MAE	7:1	MEMORY ACCESS ERROR (0081).
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE), is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION. LONG BLOCK (0001).

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operations

REWIND (OP 01)

Rewind the designated tape unit. The control is released and a result descriptor returned after rewind is initiated.

READ OP 02 (FORWARD) OR OP 03 (REVERSE)

Read a record from the designated tape unit. The operation is terminated by detection of an interrecord gap. Information transfer is terminated after reading the specified number of words or by sensing an internal DSU error.

ERASE (OP 04)

Erase in the forward direction on the designated tape unit. The operation is terminated by erasing the number of words specified. No memory cycles are used.

WRITE (OP 06)

Write a record on the designated tape unit. The operation is terminated by writing the specified number of words or a delimiter in the data stream.

WRITE TAPE MARK (OP 06)

Write a tape mark record on the unit designated, when V = 2.

SPACE (OP 08 (FORWARD): OP 09 (REVERSE))

Space 1 to 100 records as specified by the BCD value of NN of the CDL word. If bits NN are all 0's, space 100 records.

TEST (OP 99)

Test the status of the designated unit and return a result descriptor.

BCL ALPHA OPERATION (7-TRACK TAPE WITH EVEN PARITY)

When the 6-bit frame size and even parity are selected, BCL internal code is converted to BCL code on write, and BCL code is converted to BLC Internal Code on read. The BCL "?" code is written (001111).

All above operations are performed by the control.

Exception Conditions

End of tape does not terminate an operation. The end-of-tape bit is set in the result descriptor after the operation is completed.

On read operations, when a vertical parity error is detected and the 6 bit frame size is selected, a BCL "?" code is stored by the control in memory in place of the code in error.

CRC CORRECTION (9-TRACK, 800 BPI ONLY)

CDL bits V enables CRC correction. The 3 LSB's define the track to be corrected (0-7). The parity track cannot be corrected.

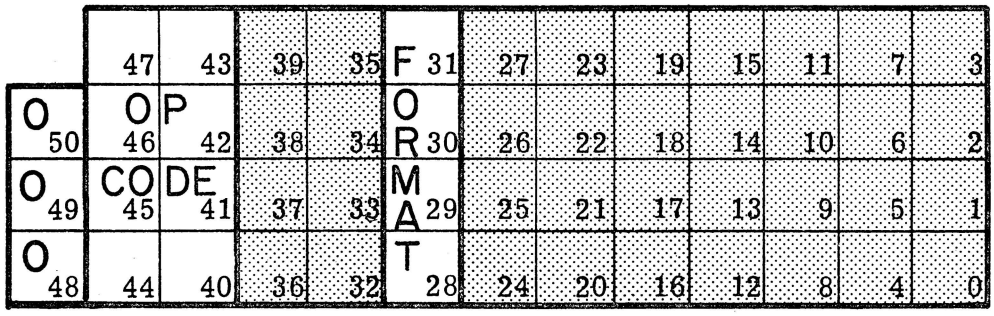
PAPER TAPE PUNCH

Paper Tape Punch Control

The B 9220 Paper Tape Punch is capable of punching a standard paper tape format in either BCL or Baudot code. The punch accommodates 5-, 6-, 7-, or 8-channel tape at a maximum rate of 100 characters per second, punching 10 characters to the inch. Standard tape widths of 11/16, 7/8, and 1 inch may be used in either the oiled paper tape, metalized mylar tape, or laminated mylar tape.

Each paper tape I/O control, reader or punch, can accommodate only one paper tape unit. The controls are the small-size controls which can be set into a PCC cabinet as either a right hand or a left hand control.

CDL Word Format



41037

FIELD		DESCRIPTION
OP CODE	47:8	48 = WRITE 49 = PUNCH LEADER 99 = TEST
FORMAT	31:4	0 = 7 Bits of data, even parity bit generated by control. 1 = 6 bits of data, odd parity bit generated by control. 2 = 8 bits, no parity.

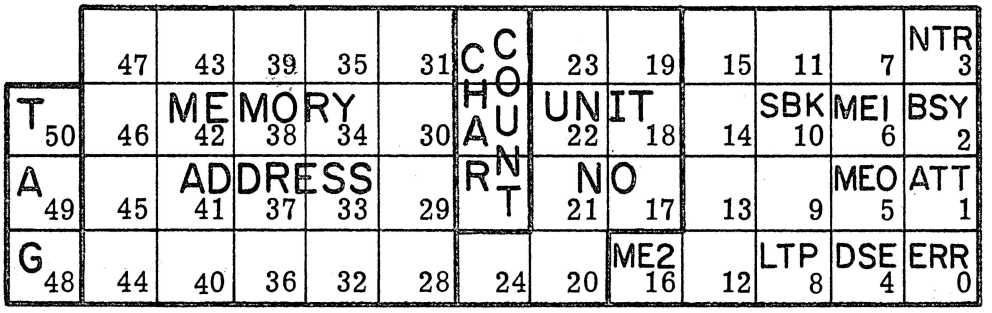
IOCW Information

OPERATION	IOCW 47	43	42	41	CDL OP CODE
WRITE BINARY (6 bit from 6 bit)	0	0	0	0	48
WRITE (8 bit from 8 bit)	0	0	0	1	48
WRITE EBCDIC (From ASCII)	1	0	1	1	48
WRITE ASCII (From EBCDIC)	1	0	0	1	48
WRITE BCL (From INT. BCL)	0	0	1	0	48
WRITE BCL (From ASCII)	1	0	1	0	48
WRITE BCL (From EBCDIC)	0	0	1	1	48
PUNCH LEADER	0	1	0	0	49

NOTE

The actual format punched is dependent on the IOCW, CDL, and Translator board in the punch.

Result Descriptor Format



41039

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
SBK	10:1	SHORT BLOCK (0401).
LTP	8:1	LOW TAPE (0101).
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
MEO	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

PAPER TAPE PUNCH

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operations

WRITE (OP 48)

Punch one record on the paper tape punch. The operation is terminated by punching the specified number of words or by punching a control code. BCL Internal Code, ASCII, or EBCDIC is converted to BCL code by translators in the IOM. BCL code is converted to BCL paper Tape Code by a translator in the punch.

If bits 31:4 in the CDL word contain a zero, the eighth bit of the character is ignored, and an even parity bit is sent on the eighth line from the control to the paper tape punch. Odd parity is punched on the paper tape.

If bits 31:4 in the CDL word contain a one, the EBCDIC/BCL translator is enabled. The control will terminate the operation when the delimiter (1000 0000) is detected. The delimiter is not sent to the punch. The seventh bit received from memory is ignored, a zero bit is forced on the seventh line, and an odd parity bit is sent on the eighth line. The BCL/BCL paper tape code translator in the paper tape punch is enabled to complete the data transfer. Odd parity is punched on the paper tape.

If bits 31:4 in the CDL word contain a two, the control sends all eight bits received from memory to the paper tape punch. Parity is neither generated not sent.

PUNCH LEADER (OP 49)

Punch the specified number of characters. When the 8-bit frame size is selected, all one characters are punched for each character spaced. When the 6-bit frame size is selected, each character spaced is an all zeros character.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

Exception Conditions

When the number of words specified are not written due to termination by the punch, the incomplete record bit is set by the control in the result descriptor.

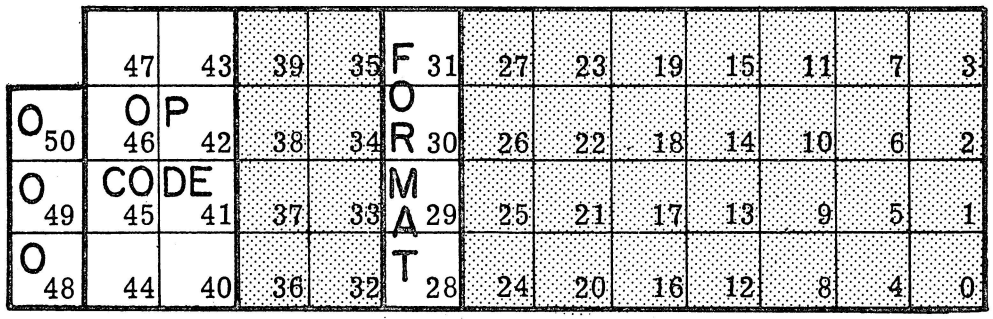
If a memory parity error is encountered during a write operation, the operation is terminated without punching any erroneous characters.

PAPER TAPE READER

Paper Tape Reader Control

The B 7120 Paper Tape Reader Control is used with the paper tape reader. The B 9120 Paper Tape Reader is capable of reading punched paper tape at a rate of 1000 characters per second and metalized mylar tape or fanfold tape at a rate of 500 characters per second. Baudot and BCL to EBCDIC code translation is automatic. All other codes are read directly into memory and may be translated programmatically. The reader can accommodate 5-, 6-, 7-, or 8-channel tape as selected by the operator. Tape widths of 11/16, 7/8, or 1 inch are interchangeable.

CDL Word Format



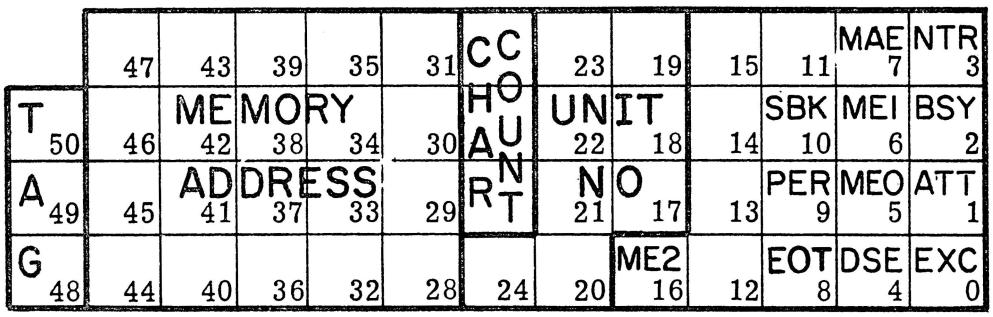
41040

FIELD		DESCRIPTION
OP CODE	47:8	40 = READ 41 = SPACE FORWARD 43 = SPACE BACKWARD 47 = REWIND 99 = TEST
FORMAT	31:4	0 = Read 7 LSB's (controller checks parity). 1 = Read 6 LSB's (controller checks parity). 2 = 8 bits, no parity.

IOCW Information

OPERATION	IOCW 47	43	42	41	39	CDL OP CODE
READ BINARY (6 bit to 6 bit)	0	0	0	0	0	40
READ (8 bit to 8 bit)	0	0	0	1	0	40
READ EBCDIC (EBCDIC to ASCII)	1	0	0	1	0	40
READ BCL (EXT BCL to INT BCL)	0	0	1	0	0	40
READ BCL (EXT BCL to ASCII)	1	0	1	0	0	40
READ BCL (EXT BCL to EBCDIC)	0	0	1	1	0	40
READ ASCII (ASCII to EBCDIC)	1	0	1	1	0	40
SPACE FORWARD	0	1	0	0	0	41
SPACE BACKWARD	0	1	0	0	1	43
REWIND	0	1	0	0	1	47

Result Descriptor Format



41042

FIELD		DESCRIPTION
ME2	16:1	MEMORY RELATED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
SBK	10:1	SHORT BLOCK (0401).
PER	9:1	PARITY ERROR (0281).
EOT	8:1	END OF TAPE on Read and Space Forward. BEGINNING OF TAPE on Rewind and Space Backward.
MAE	7:1	MEMORY ACCESS ERROR (0081).
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.

FIELD		DESCRIPTION
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operations

READ (OP 40)

Read one record from the paper tape reader. The operation is terminated by reading the specified number of words, reading a control code, or encountering end of tape. BCL code is converted to BCL Internal Code, ASCII, or EBCDIC by translators in the IOM. The reader parity-error line is monitored.

NOTE

Control codes are defined by switches on the paper tape reader.

If bits 31:4 in the CDL word contain a zero, only the least significant seven bits of the eight bits received from the reader, together with a high order zero bit are sent to the IOM for possible translation.

If bits 31:4 in the CDL word contain a one, the eight bits received from the reader are transferred to the IOM via the BCL/EBCDIC translator; however, the translator will ignore the two most significant bits. The BCL paper tape code/BCL internal translator in the reader is enabled.

If bits 31:4 in the CDL word contain a two, the eight bits received from the reader are transferred to the IOM.

FORWARD SPACE (OP 41)

Space forward the number of words specified unless stopped by end of tape or a control code.

BACKWARD SPACE (OP 43)

Space backward the number of words specified unless stopped by beginning of tape or a control code.

REWIND (OP 47)

Rewind to beginning of tape.

TEST (OP 99)

Test the status of the unit and return a result descriptor.

Exception Conditions

When the number of words specified is not read or spaced due to termination by the reader, the incomplete record bit is set by the control in the result descriptor.

Parity errors and memory access errors are reported at the end of an operation.

In the event of a memory access error and the termination of an operation by word count or end of tape, the number of characters spaced is not known. Special-error retry procedures may be required.

SINGLE LINE CONTROL

Burroughs Terminal Computer Model TC500

Burroughs Input and Display Terminal Model B 9352

Up to 8 of the above units in any combination can be serviced. It is required that the TC500 have the proper "firmware" microprogram to operate in accordance with the communications procedures outlined below. The single line control, figure IV-3-4, performs the following functions:

a. Message heading insertion for messages transmitted to a terminal.

b. Message heading deletion for messages received from the terminal.

c. Generation of vertical and longitudinal parity (even).

d. Checking of vertical and longitudinal parity (even).

e. Code conversion between EBCDIC and ASCII (7-bit).

f. Generation of input request interrupts.

Read and write message formats are defined in figures IV-3-5 and IV-3-6.

CDL Word Format

	47	43	U 39	35	31	27	23	19	15	11	7	3
O 50	O 46	P 42	N 38	34	30	26	22	18	14	10	6	2
O 49	CO 45	DE 41	I 37	33	29	25	21	17	13	9	5	1
O 48	44	40	T 36	32	28	24	20	16	12	8	4	0

41043

FIELD		DESCRIPTION
OP CODE	47:8	32 = READ 34 = WRITE 99 = TEST
UNIT	39:4	Unit number: 0 - 7

IOCW Information

OPERATION	IOCW 47	44	42	41	CDL OP CODE
READ (EBCDIC to EBCDIC)	0	1	0	1	32
READ (EBCDIC to ASCII)	1	1	0	1	32
WRITE (EBCDIC to EBCDIC)	0	0	0	1	34
WRITE (ASCII to EBCDIC)	1	0	1	1	34

Result Descriptor Format

	47	43	39	35	31	CHC	23	19	TIM 15	OVF 11	MAE 7	NTR 3
T 50	46	ME 42	MO 38	RY 34	30	AO RU	UN 22	IT 18	BID 14	CMG 10	MEI 6	BSY 2
A 49	45	AD 41	DRE 37	SS 33	29	NT	NO 21	17	TC5 13	PER 9	MOE 5	ATT 1
G 48	44	40	36	32	28	24	20	ME2 16	CON 12	8	DSE 4	EXC 0

41045

FIELD		DESCRIPTION
ME2	16:1	MEMORY REPORTED ERROR BIT 2. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
TIM	15:1	TIMEOUT (8001).
BID	14:1	Terminal unit is B 9353 (Test only).
TC5	13:1	Terminal unit is TC500 (Test only).
CON	12:1	Terminal unit is B 9352 (Test only).
OVF	11:1	OVERFLOW (0801).
CMG	10:1	CONTROL MESSAGE (0401).
PER	9:1	PARITY ERROR (0281).
MAE	7:1	MEMORY ACCESS ERROR (0081).
ME1	6:1	MEMORY REPORTED ERROR BIT 1. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.
ME0	5:1	MEMORY REPORTED ERROR BIT 0. ME2, ME1, and ME0 are encoded to report various errors detected by the MIU.

SINGLE LINE CONTROL

FIELD		DESCRIPTION
DSE	4:1	DATA SERVICE ERROR. This bit, when set, indicates that bits 15:9 contain a description of a DSU detected error as indicated in the following table: 15:1 DVE DEVICE DETECTED ERROR. 14:1 CDE CDL CHARACTER ERROR. 12:1 CME COMBINATION ERROR. 11:1 CTE COUNTER ERROR. 10:1 IFE INTERFACE ERROR. 9:1 BSE BUSS PARITY ERROR. 8:1 CPE CONTROL PARITY ERROR. 7:1 DPE DATA PARITY ERROR. If this bit (DSE) is not set, bits 15:9 contain information reported by the PC.
NTR	3:1	NOT READY TO RECEIVE.
BSY	2:1	CHANNEL BUSY ON INITIATE.
ATT	1:1	SOFTWARE ATTENTION.
EXC	0:1	EXCEPTION.

MEMORY REPORTED ERRORS

ME2	ME1	ME0	DESCRIPTION
0	1	1	STORE DISPARITY.
0	1	0	L1A ADDRESS RESIDUE ERROR.
1	1	1	MEMORY DETECTED ERROR.
1	0	1	NO ACCESS TO MEMORY.
1	1	0	FETCH DISPARITY.
1	0	0	MEMORY PROTECT ERROR.
0	0	1	MEMORY DETECTED ERROR.

Operation

READ (OP 32)

Read an input message from the designated terminal unit until an "End of Text" character is detected or until the area descriptor word count is exhausted, whichever comes first. An ASCII & EBCDIC translator is located in the peripheral control.

The IOM either stores the data as received or translates to ASCII.

NOTE

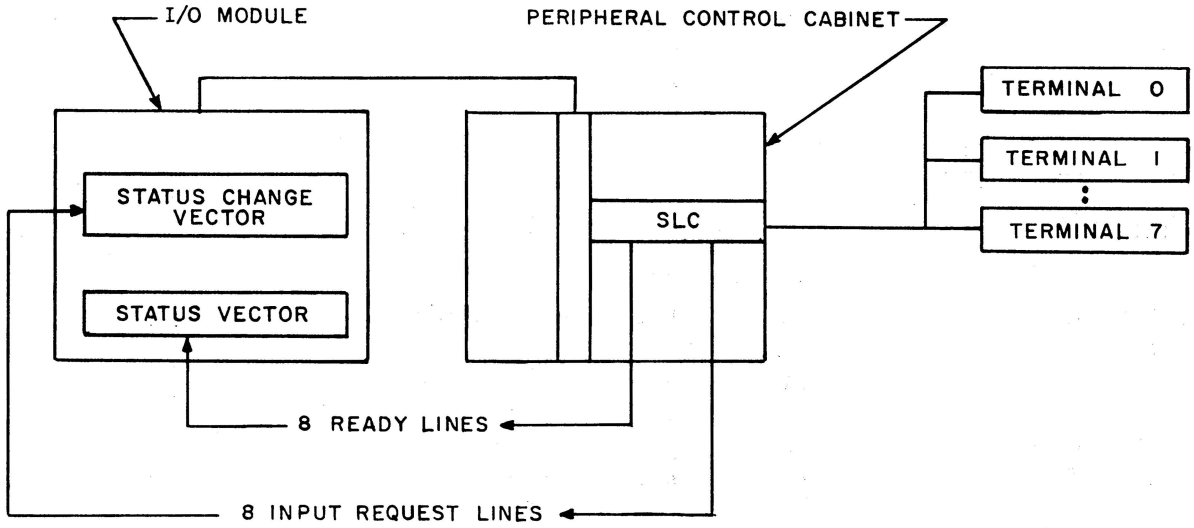
Each of the 8 terminals which can be connected to a Single Line Control is assigned a system unit designate number. The unit designates must be sequential.

WRITE (OP 34)

Send a message to the designated terminal unit until an "End of Text" character is detected in the message or until the area descriptor word count is exhausted, whichever comes first. The IOM either transmits data as received, or does an ASCII to EBCDIC translation. An EBCDIC to ASCII translator is enabled in the control.

TEST (OP 99)

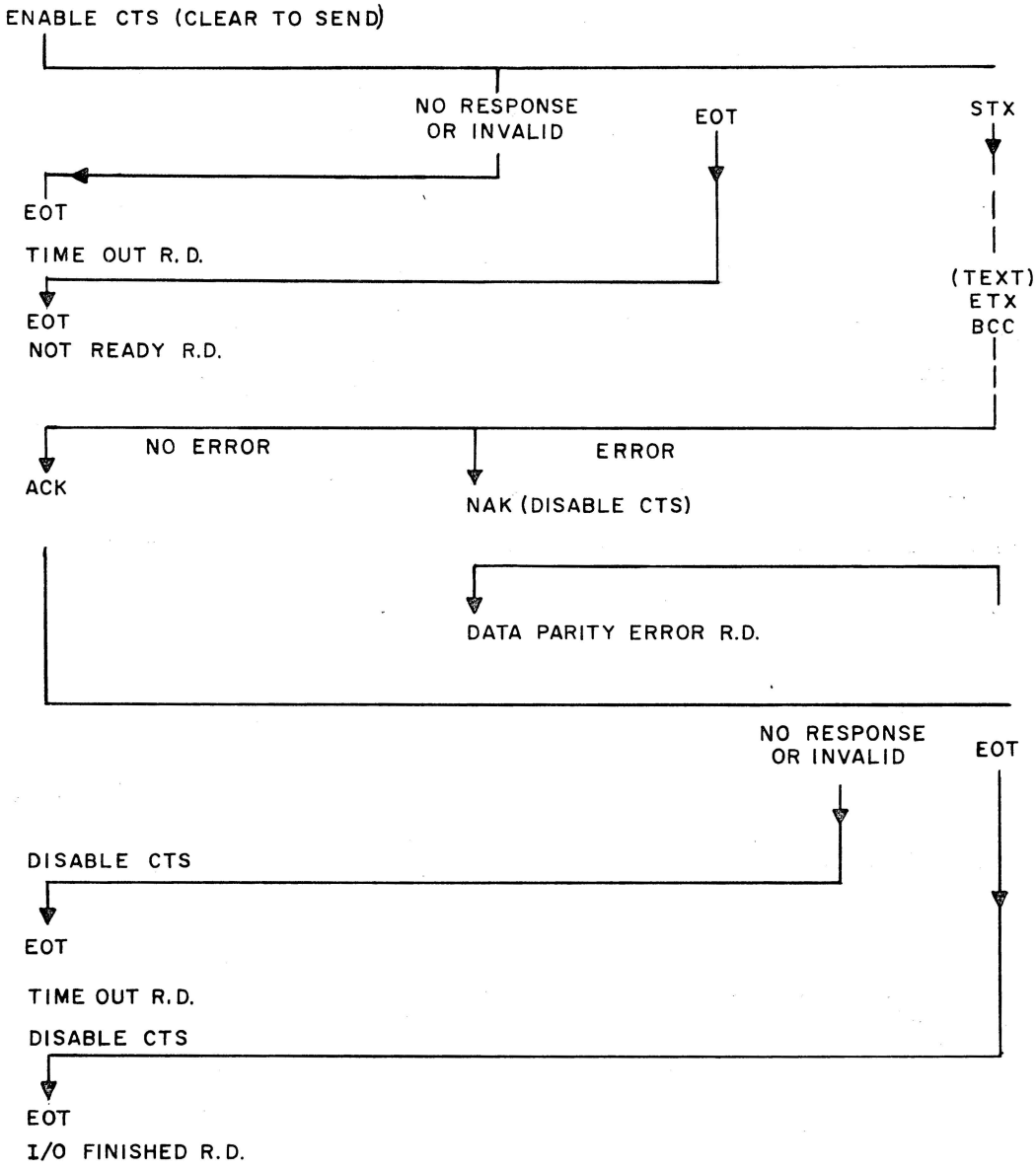
Return a result descriptor indicating the type of the designated terminal unit. If no type bit is returned, it means there is no terminal unit connected to the Single Line Control with that unit designation.



NOTE:
MAXIMUM CABLE LENGTH BETWEEN THE SINGLE LINE CONTROL AND A TERMINAL UNIT IS 50 FEET. THE MESSAGE FORMAT IS A SUBSET OF THE BURROUGHS COMMUNICATION PROCEDURE STANDARD, STANDARD 1284-9006. DATA TRANSMISSION IS ASYNCHRONOUS WITH EACH CHARACTER TRANSMITTED CONSISTING OF 10 BITS AS FOLLOWS: A "0" START BIT, 7 INFORMATION BITS (LEAST SIGNIFICANT BIT FIRST), AN EVEN PARITY AND A "1" STOP BIT.

40197

Figure IV-3-4. Single Line Control Configuration



NOTE:

1. STX IS NOT STORED IN THE B7700 MEMORY
2. THE NAK SENT TO THE TC500 OR B 9352 IN THE EVENT OF AN ERROR, CONDITIONS THE TC500 OR B 9352 TO RETRANSMIT THE SAME MESSAGE IN RESPONSE TO THE NEXT READ COMMAND.

40198

Figure IV-3-5. Read Message Format TC500 and B 9352

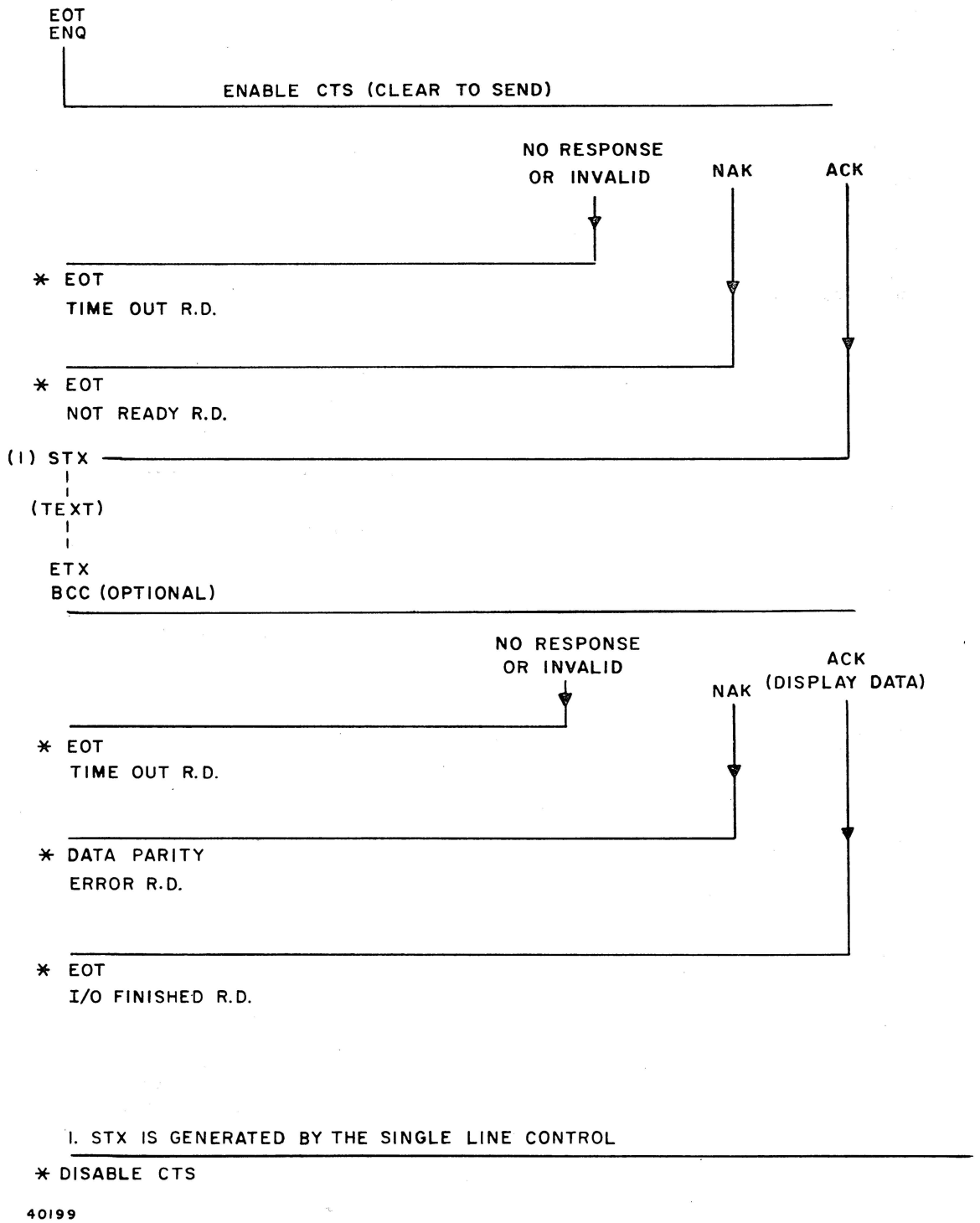


Figure IV-3-6. Write Message Format TC500 and B 9352

SINGLE LINE CONTROL

One adapter is required for each terminal unit connected to a Single Line Control. This adapter is comprised of one level changer card and one cable.

Data transmission rates of either 1200 or 2400 bits per second can be selected by implementation of the appropriate adapter in the peripheral control. The adapters used for this purpose are Adapter-SLC Timer 2400 and Adapter-SLC Timer 1200.

Either of these pluggable adapters may be installed by a field engineer. The transmission line rate must be the same for all terminals on the same Single Line Control. The rate for the TC500 is limited to 1200 bits per second.

Plug-ons with jumpers in the peripheral control are required to identify the terminal device type associated with each of the 8 possible terminal units. If no device is connected to a particular interface, the corresponding jumper should be removed from the plug-on. See Instructions - Peripheral Control Special No. 1639 7049 for details on jumper locations.

The control receives the receive ready status from the terminals and sends this information to the appropriate status vector in the IOM via a belted coaxial cable. Input request interrupts initiated by the terminals are sent via individual coaxial cables (1 to 8) to the related IOMS status change vector.

The control contains a one character buffer which receives and sends bit serial information from/to the terminal unit. Transfer of data from and to the IOM is character serial (8 bits).

The control recognizes the "ETX" code (0000011) in an input message which is translated and transferred to the IOM as 0000 0011. Detection of the end of text code terminates the input and stores a result descriptor.

The control recognizes the "ETX" code (0000 0011) in an output message. The end of text code terminates the output and a result descriptor is stored.

If the terminal is in the transmit mode when the control is initiated to do an output, a result descriptor is returned with the not-ready bit set.

There are two delay multi's in the Single Line Control. One multi (DLTO) monitors the line "Turn around" time such that if a terminal does not respond within a time limit, the time out flip-flop (TMOF) is set and a result descriptor generated. If an invalid response is received, the same result occurs without waiting for the multi to time out. The second multi monitors the time elapsed between characters being received from a terminal. If excessive time elapses, the multi (CHTO) times out, setting the time out flip-flop (TMOF) and a result descriptor is generated.

On a read, if the Single Line Control has not gained access to the IOM in time to store a character before the next character is received from the terminal, a memory access error is flagged in the result descriptor, the character is lost, and reading continues. On a write, if the Single Line Control has not gained access to the IOM to fetch another character by the time required to maintain line transmission rates, a memory access error is flagged in the result descriptor, an all zeros (NULL) code is sent to the terminal in place of the character, and writing continues.