CP 1720-5592

DETAILED DESIGN SPECIFICATION

FOR

B8502 CENTRAL PROCESSOR MODULE

M AND E NO.
1705-2010

OCTOBER 20, 1969

Burroughs Corporation

Defense, Space and Special Systems Group

PAOLI, PA

CP 1720-5592

DETAILED DESIGN SPECIFICATION

FOR

B8502 CENTRAL PROCESSOR MODULE

M AND E NO.
1705-2010

OCTOBER 20, 1969

Burroughs Corporation

Defense, Space and Special Systems Group

PAOLI, PA

Burroughs Corporation

LARGE COMPUTER SYSTEMS ORGANIZATION
PAOLI, PA.

CP 1720-5592

Detailed Design Specification for B8502
Central Processor Module

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
AA	20 Sept. 68	Original Issue		
AB	14 July 69	This specification CP 1720-5592 supersedes the original issue of this specification dated September 1968, changes title, and incorporates complete revision of original issue to reflect changed design concepts authorized under ECN 2247	F.V. Rehnquist Fred T Reynard	L.W. Beers Blair Thompson Released by: H. Hayman/LB
AC	20 Oct. 69	This issue of specification 1720-5592 supersedes revision AB of this specification dated July 1969 and incorporates changes and additions under ECN 2366 Many minor corrections, clarifications, paragraph numbering and title changes have been made. The major changes are as follows: 3.3 Interpreter Unit: Revised literal operator 3.3.2 Kernel Section: Revised descriptor representation to include attribute and structure expression representation. Revised and update name representation to include N-Base names, program names and top of name stack names. Revised descriptor evaluation techniques including attribute collection. Revised local buffer map in order to allow more descriptors to be captured locally.	F.V. Rehnquist Fred T Reynard	L.W. Beers Blair Thompson Released by: H. Hayman/LB

CP 1720-5592

Detailed Design Specification for B8502
Central Processor Module

REVISIONS

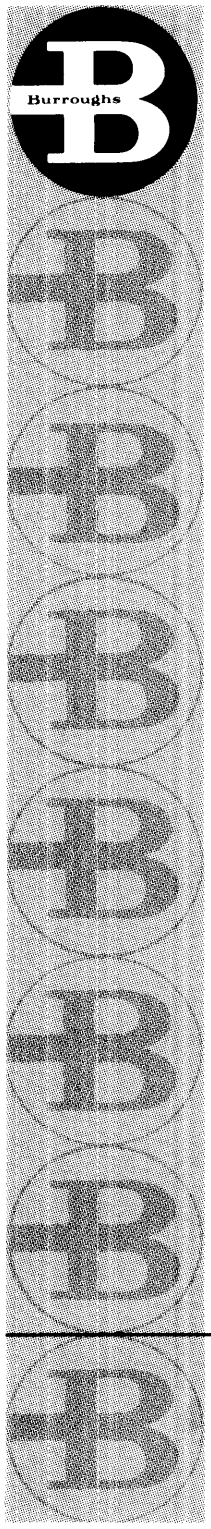
REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
AC (Cont)	20 Oct. 69	3.3.3 Structure Buffering Section: Updated resource structure representation. Updated procedure structure representation to include revised procedure display. Updated coroutine structure representation. 3.4 Arithmetic Unit: A. Residue checking Deleted 3.4.7 AU Residue Checking. Added 3.4.6 Field Section - This is a detailed description of that portion of residue checking which is oriented towards operands (fields) rather than registers. Extended details of 3.4.1 thru 3.4.5 to include residue checking of related registers. B. Format Added 3.4.7 Format Section. C. Control Revised contents of 3.4.8 Au Data Control (formerly 3.4.6). Added 3.4.9 Control Hierachy.		

CP 1720-5592

Detailed Design Specification for B8502
Central Processor Module

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
AC (Cont)	20 Oct. 69	Appendix A - Algorithms were changed in order to: 1. Simplify existing algorithms. 2. Remove program from self identifying structures. 3. Have a new link list structure with two new operations, sequence and operation.		



CP 1720-5592

DETAILED DESIGN SPECIFICATION

FOR

B8502 CENTRAL PROCESSOR MODULE

M AND E NO.
1705-2010

OCTOBER 20, 1969

Burroughs Corporation

Defense, Space and Special Systems Group

PAOLI, PA

TABLE OF CONTENTS

1.0 SCOPE. . . . . PAGE 1
2.0 APPLICABLE DOCUMENTS. . . . . PAGE 1
3.0 IMPLEMENTATION CONCEPTS. . . . . PAGE 1
3.1 FUNCTIONAL PERFORMANCE CHARACTERISTICS. . . . . PAGE 2
3.2 MAJOR COMPONONTS. . . . . PAGE 4
3.2.1 INTERPRETER UNIT. . . . . PAGE 4
3.2.2 ARITHMETIC UNIT . . . . . PAGE 5
3.2.3 MEMORY INTERFACE UNIT . . . . . PAGE 6
3.3 INTERPRETER UNIT (IU) . . . . . PAGE 6
3.3.1 PROGRAM SECTION . . . . . PAGE 8
3.3.1.1 PROGRAM FLOW AND REPRESENTATION . . . . . PAGE 10
3.3.1.2 PROGRAM BUFFER. . . . . PAGE 12
3.3.1.3 PROGRAM BARREL. . . . . PAGE 15
3.3.1.4 PROGRAM CONTROLS. . . . . PAGE 15
3.3.2 KERNEL SECTION . . . . . PAGE 16
3.3.2.1 DESCRIPTOR REPRESENTATION. . . . . PAGE 16
3.3.2.1.1 DESCRIPTOR ATTRIBUTES . . . . . PAGE 16
3.3.2.1.2 STRUCTURE EXPRESSION. . . . . PAGE 19
3.3.2.2 NAMES . . . . . PAGE 22
3.3.2.3. DESCRIPTOR EVALUATION. . . . . PAGE 24
3.3.2.4 KERNEL HARDWARE. . . . . PAGE 26
3.3.2.4.1 ATTRIBUTE STACKS. . . . . PAGE 28
3.3.2.4.2 DESCRIPTOR EXECUTION REGISTER . . . . . PAGE 29
3.3.2.4.3 DESCRIPTOR IMPLODE-EXPLODE MECHANISM. . . . PAGE 29
3.3.2.4.4 PROGRAM/DESCRIPTOR CONTROL REGISTER . . . . PAGE 29
3.3.2.4.5 STRUCTURE AND DESCRIPTOR BUFFER . . . . . PAGE 29
3.3.3 STRUCTURE BUFFERING SECTION . . . . . PAGE 32
3.3.3.1 RESOURCE CONTROL STRUCTURE. . . . . PAGE 32
3.3.3.2 PROCEDURE CONTROL STRUCTURE . . . . . PAGE 37
3.3.3.3 COROUTINE CONTROL STRUCTURE . . . . . PAGE 41
3.3.3.4 PROGRAM/DESCRIPTOR CONTROL STRUCTURE. . . . . PAGE 45
3.3.3.5 DESCRIPTOR BUFFERS. . . . . PAGE 48
3.3.4 INTERRUPT SECTION . . . . . PAGE 48
3.4 ARITHMETIC UNIT (AU). . . . . PAGE 51
3.4.1 STORAGE SECTION . . . . . PAGE 52
3.4.2 BARREL SECTION. . . . . PAGE 55
3.4.3 ADDER SECTION . . . . . PAGE 58
3.4.4 EXPONENT SECTION. . . . . PAGE 61
3.4.5 LENGTH SECTION. . . . . PAGE 63
3.4.6 FIELD SECTION . . . . . PAGE 67
3.4.7 FORMAT SECTION. . . . . PAGE 70
3.4.8 AU DATA CONTROL . . . . . PAGE 74
3.4.9 CONTROL HIERARCHY . . . . . PAGE 77
3.5 MEMORY INTERFACE UNIT (MIU) . . . . . PAGE 80
3.5.1 FUNCTIONAL OPERATION. . . . . PAGE 82
3.5.2 FUNCTIONAL COMPONENTS . . . . . PAGE 86
4.0 PHYSICAL CHARACTERISTICS. . . . . PAGE 87
5.0 POWER REQUIREMENTS . . . . . PAGE 87
6.0 OPERATING ENVIRONMENT REQUIREMENTS . . . . . PAGE 88

7.0 CABLING REQUIREMENTS . . . . . PAGE 88
8.0 CONTROL PANEL REQUIREMENTS . . . . . PAGE 88
9.0 QUALITY ASSURANCE PROVISIONS. . . . . PAGE 88
9.1 RELIABILITY AND MAINTAINABILITY . . . . . PAGE 88
9.1.1 MEAN TIME BETWEEN FAILURES (MTBF) . . . . . PAGE 88
9.1.2 MEAN TIME TO REPAIR (MTTR). . . . . PAGE 89
9.2 QUALITY CONTROL . . . . . PAGE 89
9.2.1 INSPECTION RESPONSIBILITY . . . . . PAGE 89
9.2.2 CLASSIFICATION OF TESTS . . . . . PAGE 89
9.2.3 TEST CONDITIONS . . . . . PAGE 89
9.2.3.1 ENVIRONMENTAL . . . . . PAGE 89
9.2.3.2 TEST POWER. . . . . PAGE 90
(STANDARD AND SPECIAL - TO BE SUPPLIED). . . . . PAGE 90
9.2.4 TEST PROCEDURES(S). . . . . PAGE 90
9.2.5 TEST METHODS. . . . . PAGE 90
10.0 PREPARATION FOR DELIVERY. . . . . PAGE 90
10.1 PRESERVATION, PACKAGING AND PACKING. . . . . PAGE 90
APPENDIX A. . . . . A- 1
TABLE A-1 STANDARD MNEMONICS. . . . . A- 2
TABLE A-2 STANDARD MICRO-OPERATORS. . . . . A- 3
VECTOR. . . . . A- 4
CONSTRUCT. . . . . A- 4
FIELD . . . . . A- 5
CONSTRUCT. . . . . A- 5
VARIABLE FIELD. . . . . A- 6
CONSTRUCT. . . . . A- 6
STACK . . . . . A- 7
ENTER. . . . . A- 7
REMOVE . . . . . A- 8
CONSTRUCT. . . . . A- 9
STACK-VECTOR. . . . . A- 10
ENTER. . . . . A- 10
REMOVE . . . . . A- 11
CONSTRUCT. . . . . A- 12
QUEUE . . . . . A- 13
ENTER. . . . . A- 13
REMOVE . . . . . A- 14
CONSTRUCT. . . . . A- 15
LINKED LIST . . . . . A- 16
ENTER. . . . . A- 16
REMOVE . . . . . A- 17
CONSTRUCT. . . . . A- 18
SEQUENCE . . . . . A- 18
RESET. . . . . A- 18
PUSHDOWN. . . . . A- 19
ENTER. . . . . A- 19
REMOVE . . . . . A- 20
CONSTRUCT. . . . . A- 21
VARIABLE QUEUE. . . . . A- 22
ENTER. . . . . A- 22
REMOVE . . . . . A- 23

CONSTRUCT. . . . . A- 24
PUSHDOWN-STACK. . . . . A- 25
ENTER. . . . . A- 25
REMOVE . . . . . A- 26
PUSHDOWN-PUSHDOWN . . . . . A- 27
ENTER. . . . . A- 27
REMOVE . . . . . A- 28
CALL. . . . . A- 29
FIN . . . . . A- 30
APPENDIX B. . . . . B- 1
PROGRAM OPERATOR ALGORITHMS . . . . . B- 1
STRUCTURE OPERATORS. . . . . B- 2
CONSTRUCT (NAME) . . . . . B- 2
ENTER (NAME) . . . . . B- 4
REMOVE (NAME). . . . . B- 7
REFERENCE OPERATORS . . . . . B- 9
NAME . . . . . B- 9
VALUE. . . . . B- 10
STORE. . . . . B- 11
EXECUTE. . . . . B- 12
DESCRIPTOR OPERATORS. . . . . B- 13
LOAD . . . . . B- 13
DESCRIBE (NAME). . . . . B- 14
ALLOCATE (NAME 1, NAME 2). . . . . B- 15
BIND (NAME 1, NAME 2). . . . . B- 16
SHORTEN (NAME) . . . . . B- 17
FINAL COMBINE. . . . . B- 19
STACK OPERATORS . . . . . B- 20
DUPLICATE. . . . . B- 20
DELETE . . . . . B- 21
EXCHANGE . . . . . B- 22
VSS TO NSS . . . . . B- 23
NSS TO VSS . . . . . B- 24
PROGRAM CONTROL OPERATORS . . . . . B- 25
LOOP (NAME). . . . . B- 25
LOOP TEST. . . . . B- 26
BRANCH . . . . . B- 27
BRANCH CONDITIONAL . . . . . B- 28
HALT . . . . . B- 29
NO-OP. . . . . B- 30
PROCEDURE CONTROL OPERATORS . . . . . B- 31
SLICE. . . . . B- 31
UNSLICE. . . . . B- 32
PROCEDURE RETURN . . . . . B- 33
COROUTINE OPERATORS . . . . . B- 34
COROUTINE ACTIVATE (NAME). . . . . B- 34
COROUTINE CALL (NAME). . . . . B- 35
COROUTINE END. . . . . B- 37
PROCESS CALL OPERATORS. . . . . B- 39
PROCESS PARAMETER (NAME) . . . . . B- 39
PROCESS PARAMETER INDIRECT (NAME). . . . . B- 40

PROCESS CALL . . . . . B- 41
PROCESS END. . . . . B- 42
MISCELLANEOUS OPERATORS . . . . . B- 43
LITERAL. . . . . B- 43
SET INTERRUPT. . . . . B- 44
PROCESS FAULT CHECK. . . . . B- 45
SYSTEM FLAG CHECK. . . . . B- 47
GET V SPACE (NAME) . . . . . B- 48
IMP CALL . . . . . B- 49
IMP RETURN . . . . . B- 50
LOAD ARITHMETIC VARIANT REGISTER (AVR) . . . . . B- 51
SET ATTRIBUTES . . . . . B- 52
APPENDIX C. . . . . C- 1
GLOSSARY. . . . . C- 1

CP 1720-5592

1.0 SCOPE

THIS SPECIFICATION ESTABLISHES THE FUNCTIONAL REQUIREMENTS FOR THE B8502 CENTRAL PROCESSOR MODULE (CPM) WHICH FORMS A PART OF THE B8502 DATA PROCESSING SYSTEM. THE CPM IS THE HIGH-SPEED COMPUTATION CENTER OF THE B8502 DATA PROCESSING SYSTEM.

2.0 APPLICABLE DOCUMENTS

THE FOLLOWING DOCUMENTS OF THE LATEST ISSUE IN EFFECT FORM A PART OF THIS SPECIFICATION TO THE EXTENT SPECIFIED HEREIN:

SPECIFICATIONS

BURROUGHS CORPORATION

LARGE COMPUTER SYSTEM ORGANIZATION PAOLI, PENNSYLVANIA 19301

1720 2045	B8500 INFORMATION PROCESSING SYSTEM, GENERAL SPECIFICATION FOR
CP 1760-0008	B8512 INPUT/OUTPUT MODULE, LOGIC SPECIFICATION FOR
CP 1720-5576	B8515 MEMORY MODULE, DETAILED DESIGN SPECIFICATION FOR

PASADENA MANUFACTURING AND ENGINEERING DIVISION LOS ANGELES, CALIFORNIA 90021

A1123-9134	MUL MEMORY CELL

3.0 IMPLEMENTATION CONCEPTS.

THE 8502 CENTRAL PROCESSOR MODULE (CPM) SHALL PROVIDE A HARDWARE IMPLEMENTATION OF THE INTERPRETER AND ARITHMETIC SECTIONS OF THE GENERAL SPECIFICATION FOR THE B8500 INFORMATION PROCESSING SYSTEM. THE HARDWARE IMPLEMENTATION EMPHASIZES SYSTEM PERFORMANCE BY LOCAL ASSOCIATIVE BUFFERING OF PROGRAM, DESCRIPTORS AND DATA, AND BY HARD-WIRED ALGORITHMS FOR ENTERING, REMOVING AND LOCATING INFORMATION IN DATA AND PROGRAM STRUCTURE AND BY THE SELECTION OF A SET OF PROGRAM OPERATORS DESIGNED TO EXPLOIT THE COMMON ATTRIBUTES OF HIGHER LEVEL LANGUAGES. THE ARITHMETIC SECTION SHALL BE CAPABLE OF PERFORMING ARITHMETIC ON VARIABLE LENGTH OPERANDS, AND ON OPERANDS OF VARYING REPRESENTATION. THIS APPROACH WILL ACHIEVE

SIGNIFICANT SPEED WITHOUT SACRIFICING EFFICIENT THROUGHPUT.

3.1 FUNCTIONAL PERFORMANCE CHARACTERISTICS

THE MAIN FUNCTIONS OF THE CPM SHALL BE TO ACTIVATE AND DEACTIVATE PROCESSES, DIRECT INFORMATION TRANSFERS BETWEEN MODULES, SERVICE INTERRUPTS AND EXECUTE ARITHMETIC CALCULATIONS REQUIRED BY PROGRAM. THESE FUNCTIONS SHALL BE PERFORMED UNDER THE DIRECTION OF THE MASTER CONTROL PROGRAM (MCP). THE CPM MINIMIZES MEMORY ACCESSES BY UTILIZING PHASED FETCHES AND STORES WHERE POSSIBLE, AND BY ASSOCIATIVELY BUFFERING INFORMATION. EXECUTION SPEEDS ARE ENHANCED AND HARDWARE COSTS ARE MINIMIZED BY THE DECENTRALIZATION OF CONTROLS OF FUNCTIONALLY INDEPENDENT SUBSECTIONS WITHIN THE INTERPRETER. FIGURE 1 IS A BLOCK DIAGRAM SHOWING THE OVERALL CONFIGURATION OF THE CPM.

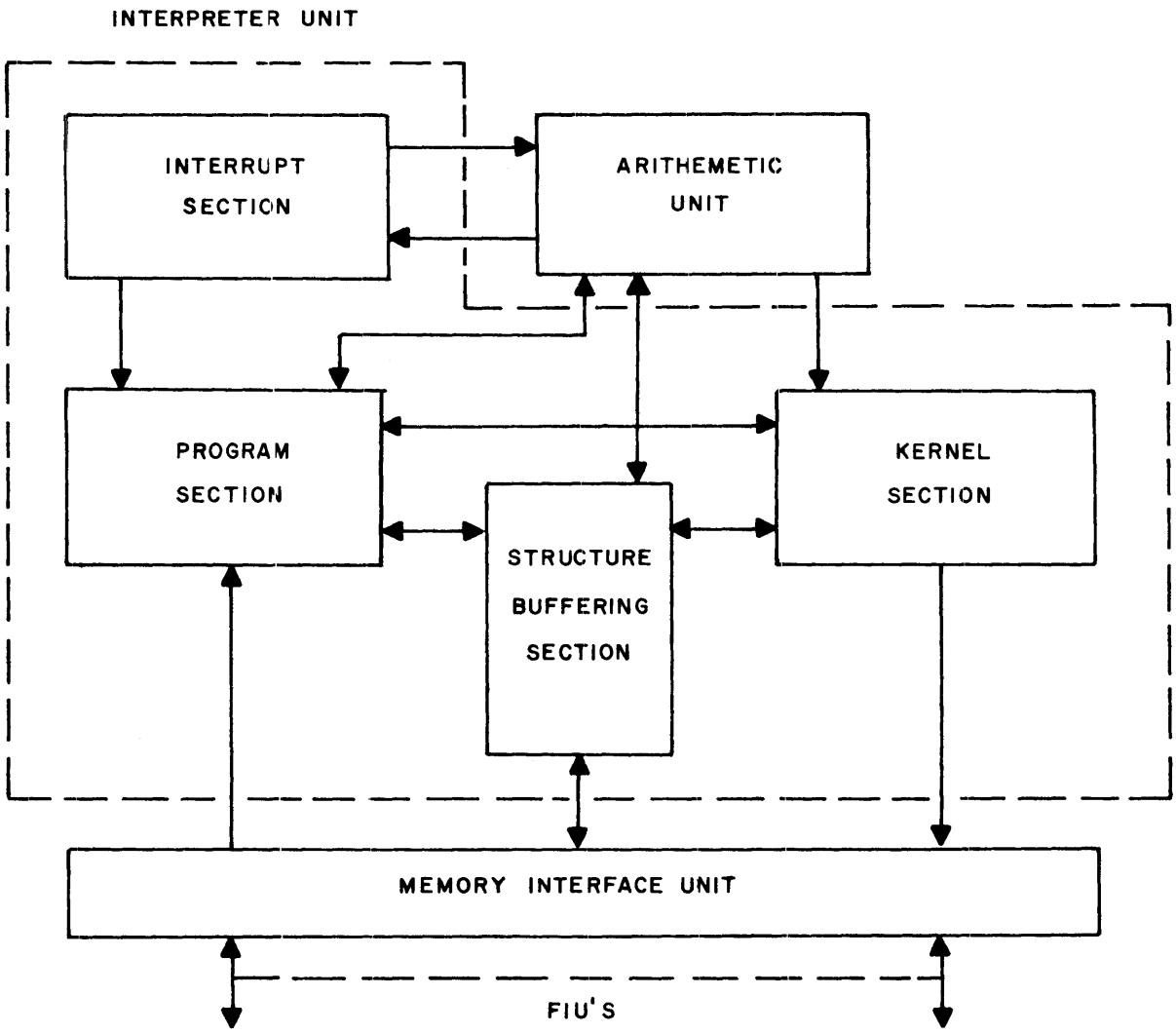


FIGURE 1. CENTRAL PROCESSOR MODULE (CPM), SIMPLIFIED BLOCK DIAGRAM

3.2 MAJOR COMPONONTS.

THE CPM SHALL CONSIST OF THE FOLLOWING THREE FUNCTIONALLY DISTINCT UNITS.

ITEM	NAME	REFERENCE PARAGRAPH
1.	INTERPRETER UNIT	3.2.1, 3.3
2.	ARITHMETIC UNIT	3.2.2, 3.4
3.	MEMORY INTERFACE UNIT	3.2.3, 3.5

3.2.1 INTERPRETER UNIT

AS FURTHER SPECIFIED IN 3.3, THE INTERPRETER UNIT (IU) SHALL CONTROL THE MOVEMENT OF PROGRAM AND DATA, PROVIDE AUTOMATIC MEMORY PROTECTION, RESPOND TO INTERRUPTS, AND CONTROL, EMPTY, AND REPLENISH THE VARIOUS STACKS AND BUFFERS WITHIN THE CPM. THE IU SHALL CONSIST OF THE FOLLOWING SECTIONS:

ITEM	NAME	REFERENCE PARAGRAPH
A.	PROGRAM SECTION	3.3.1
1.	PROGRAM FLOW AND REPRESENTATION	3.3.1.1
2.	PROGRAM BUFFER	3.3.1.2
3.	PROGRAM BARREL	3.3.1.3
4.	PROGRAM AND CONTROLS	3.3.1.4
B.	KERNEL SECTION	3.3.2
1.	DESCRIPTOR REPRESENTATION	3.3.2.1
2.	NAMES	3.3.2.2

3.	DESCRIPTOR EVALUATION	3.3.2.3
4.	KERNEL HARDWARE	3.3.2.4
C.	STRUCTURE BUFFERING SECTION	3.3.3.
1.	RESOURCE CONTROL STRUCTURE	3.3.3.1
2.	PROCEDURE CONTROL STRUCTURE	3.3.3.2
3.	COROUTINE CONTROL STRUCTURE	3.3.3.3
4.	PROGRAM DESCRIPTOR CONTROL STRUCTURE	3.3.3.4
5.	DESCRIPTOR BUFFERS	3.3.3.5
D.	INTERRUPT SECTION	3.3.4

3.2.2 ARITHMETIC UNIT

AS FURTHER SPECIFIED IN 3.4, THE ARITHMETIC UNIT (AU) SHALL PROVIDE A CAPABILITY FOR ARITHMETIC OR LOGICAL OPERATIONS OF EITHER BINARY OR DECIMAL NOTATION AND WITH OPERANDS OF VARIABLE LENGTH. THE AU SHALL CONSIST OF THE FOLLOWING FIVE SECTIONS:

ITEM	NAME	REFERENCE PARAGRAPH
1.	STORAGE SECTION	3.4.1
2.	BARREL SECTION	3.4.2
3.	ADDER SECTION	3.2.3
4.	EXPONENT SECTION	3.4.4
5.	LENGTH SECTION	3.4.5
6.	FIELD SECTION	3.4.6
7.	FORMAT SECTION	3.4.7

3.2.3 MEMORY INTERFACE UNIT

THE MEMORY INTERFACE UNIT (MIU) SHALL COORDINATE COMMUNICATIONS BETWEEN INTERPRETER AND THE FIELD ISOLATION UNIT (FIU) OF THE B8515 MEMORY MODULE AS SPECIFIED IN SPECIFICATION CP 1720-5576. THE MIU SHALL BE CAPABLE OF SERVICING ANY OF UP TO SIXTEEN FIU(S). IN ADDITION, THE MIU SHALL PROVIDE DRIVERS AND RECEIVERS FOR COMMUNICATIONS BETWEEN THE INTERPRETER AND OTHER MODULES ON THE SYSTEM CONTROL BUS. A DESCRIPTION OF THE FUNCTIONAL OPERATION AND FUNCTIONAL COMPONENTS OF THE MIU IS GIVEN IN PARAGRAPHS 3.5, 3.5.1 AND 3.5.2.

3.3 INTERPRETER UNIT (IU)

THE IU SHALL PROVIDE THE PROCESSING CONTROL FOR THE B8502 DATA PROCESSING SYSTEM BY MEANS OF STRUCTURE OPERATORS SPECIFICALLY DESIGNED FOR THE EFFICIENT MANAGEMENT OF DATA AND PROGRAM STRUCTURES, AND BY MEANS OF PROGRAM OPERATORS SELECTED TO ALLOW EASY IMPLEMENTATION OF HIGHER LEVEL LANGUAGES. THE CONTROL INFORMATION SHALL BE DISTRIBUTED, AS REQUIRED, TO THE ARITHMETIC UNIT FOR DATA PROCESSING AND THROUGH THE MIU TO THE MEMORY MODULE FOR DATA AND PROGRAM STORAGE AND RETRIEVAL. IN ADDITION, THE IU SHALL BE COGNIZANT OF INTERRUPTS AND FAULTS THAT HAVE OCCURRED AND WILL BE PREPARED TO PROPERLY SERVICE THEM.

THE IU IS SUBDIVIDED INTO FOUR SECTIONS. THE PROGRAM SECTION FETCHES, INTERPRETS, AND EXECUTES THE PROGRAM OPERATOR IN THE PROGRAM STRING. THE KERNEL SECTION FETCHES, INTERPRETS, EXECUTES, AND UPDATES DESCRIPTORS, REFERRED TO BY NAME IN THE PROGRAM STRING, ACCORDING TO THE PROGRAM OPERATOR BEING EXECUTED. THE STRUCTURE BUFFERING SECTION CONSISTS OF A SET OF LOCAL MEMORIES WHICH BUFFER OFTEN ACCESSED ITEMS IN ORDER TO MINIMIZE LEVEL-1 FETCHES. THE BUFFERING IS BASED ON THE STRUCTURES USED TO DEFINE THE PROCESSOR. THE INTERRUPT SECTION SHALL RECEIVE INTERRUPTS AND FAULTS, EXAMINE THEM AND PASS THE APPROPRIATE FAULT OR INTERRUPT NUMBER TO ACCOMPLISH A CHANGE IN PROGRAM. A DETAILED BLOCK DIAGRAM OF THE IU IS SHOWN IN FIGURE 2.

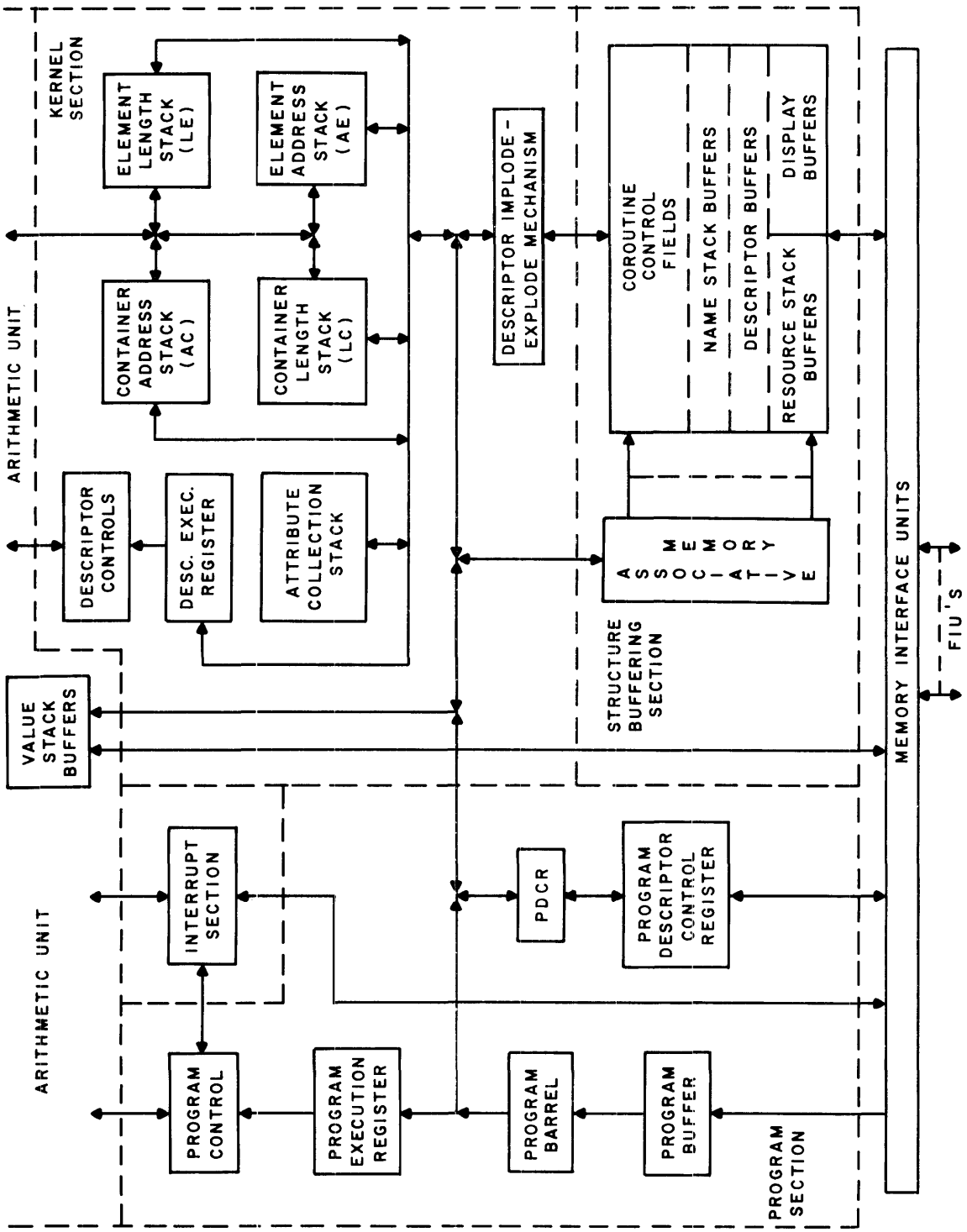


FIGURE 2. INTERPRETER UNIT (IU), DETAILED BLOCK DIAGRAM

3.3.1 PROGRAM SECTION

THIS SECTION DESCRIBES THE FLOW OF PROGRAM FROM LEVEL-1 TO EXECUTION, THE REPRESENTATION OF THE PROGRAM OPERATORS, AND THE HARDWARE ASSOCIATED WITH PROGRAM FETCHING AND EXECUTION. A LISTING OF THE PROGRAM OPERATORS AND A DESCRIPTION OF WHAT EACH OPERATOR DOES APPEARS IN APPENDIX B. THE HARDWARE ASSOCIATED WITH THE PROGRAM SECTION IS SHOWN IN FIGURE 3.

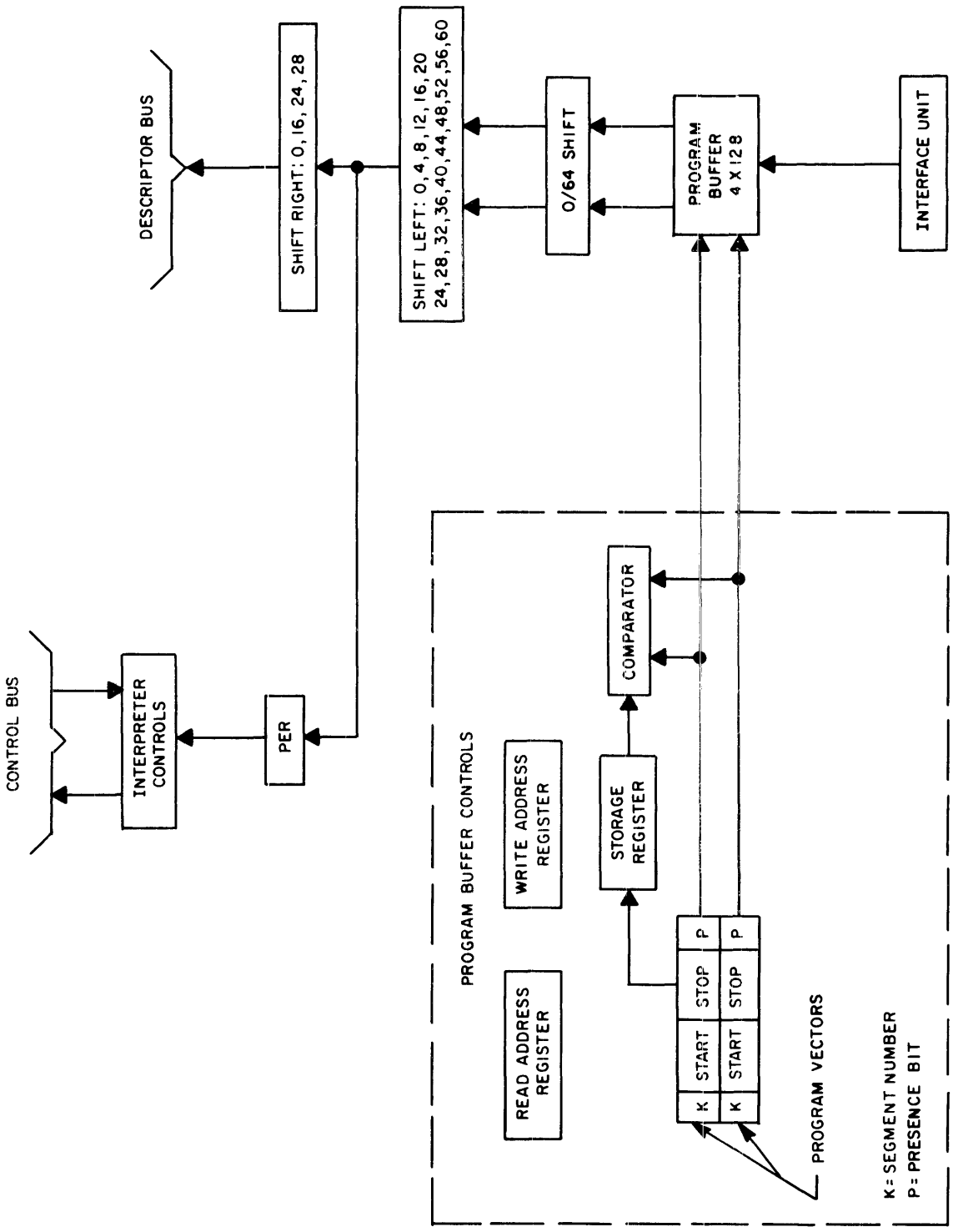


FIGURE 3. PROGRAM SECTION, BLOCK DIAGRAM

3.3.1.1 PROGRAM FLOW AND REPRESENTATION

THE PROGRAM SYLLABLE CURRENTLY BEING EXECUTED IS POINTED TO BY THE CONTENTS OF THE PROGRAM/DESCRIPTOR CONTROL REGISTER (PDCR) THIS PROGRAM SYLLABLE IS PART OF A PROGRAM SEGMENT THAT IS STORED IN THE PROGRAM BUFFER, A LOCAL ASSOCIATIVE MEMORY. THE PROGRAM BUFFER AUTOMATICALLY REFILLS ITSELF WHEN IT SENSES PROGRAM WILL RUN OUT. UPON CHANGES OF DIRECTION IN THE PROGRAM STRING CAUSED EITHER BY PROCEDURE ENTRY OR BRANCHES, THE PROGRAM BUFFER IS CHECKED ASSOCIATIVELY TO SEE IF THE BEGINNING OF THE NEW PROGRAM SEGMENT TO BE EXECUTED IS ALREADY RESIDENT IN THE PROGRAM BUFFER. NESTING AND UNNESTING OF THE PCR FOR PROCEDURE ENTRY AND EXIT AND FOR LOOP CONTROL OPERATORS UTILIZE THE PROGRAM DESCRIPTOR CONTROL STACK (PDCS), ANOTHER LOCAL MEMORY. THE PDCS AUTOMATICALLY LINKS TO LEVEL-1 FOR EMPTYING AND REPLENISHING ITS CONTENTS. THE PROGRAM/ DESCRIPTOR CONTROL REGISTER AND STACK ARE DISCUSSED FURTHER IN PARAGRAPH 3.3.3.4 PROGRAM DESCRIPTOR CONTROL STRUCTURE

PROGRAM OPERATORS ARE EXTRACTED FROM THE PROGRAM STRING BY THE PROGRAM BARREL AND PLACED INTO THE PROGRAM EXECUTION REGISTER (PER). NAMES ARE EXTRACTED FROM THE PROGRAM STRING BY THE PROGRAM BARREL AND PLACED INTO THE ATTRIBUTE STACKS FOR EVALUATION. LITERALS ARE EXTRACTED FROM THE PROGRAM STRING BY THE PROGRAM BARREL AND PLACED INTO THE VALUE STACK OR NAME STACK.

THE PROGRAM OPERATORS ARE, IN THE INTEREST OF CODE COMPACTION, CLASS CODED AS ILLUSTRATED IN FIGURE 4. THE FOUR CLASSES OF PROGRAM OPERATORS ARE AS FOLLOWS:

A. LITERAL OPERATORS

B. AU OPERATORS

C. NAME OPERATORS

D. GENERAL OPERATORS

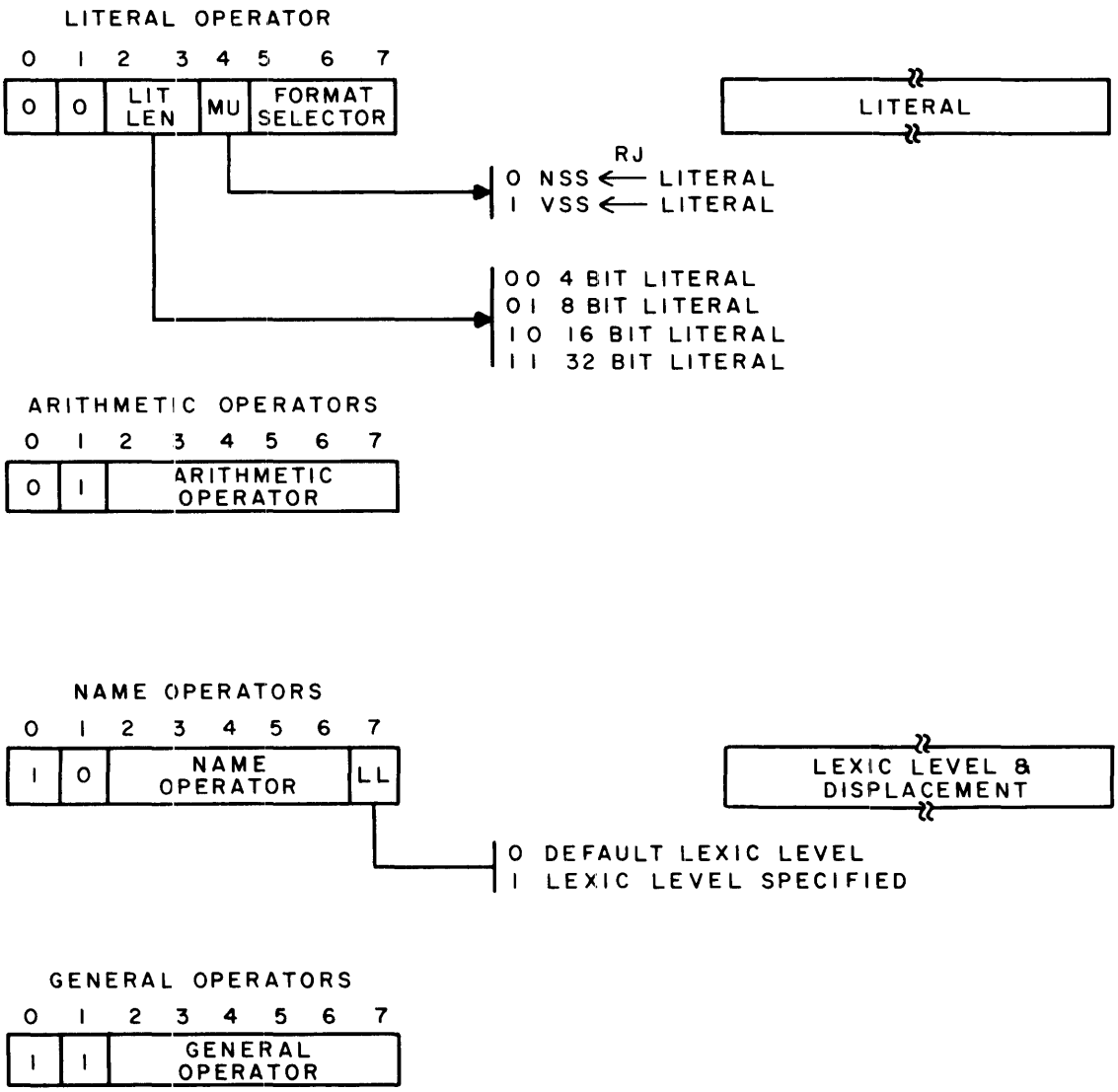


FIGURE 4. PROGRAM OPERATOR REPRESENTATION

EACH CLASS OF OPERATORS STARTS WITH AN EIGHT BIT SYLLABLE. LITERAL AND NAME OPERATORS CAN INCREASE IN FOUR BIT INCREMENTS TO A MAXIMUM SYLLABLE SIZE OF 32 BITS FOR NAME OPERATORS AND 40 BITS FOR LITERAL OPERATORS. THE FIRST TWO BITS OF THE OPERATION CODE DEFINE WHICH CLASS OF OPERATOR THE PROGRAM SYLLABLE CONTAINS. IF THE PROGRAM SYLLABLE CONTAINS A LITERAL OPERATOR, THE NEXT TWO BITS DEFINE THE SIZE OF THE LITERAL. THE LITERAL MAY BE FOUR, EIGHT, 16 OR 32 BITS IN LENGTH. THE NEXT FOUR BIT GROUP OF THE LITERAL SYLLABLE DEFINES THE DESTINATION AND ARITHMETIC FORMAT OF THE LITERAL. THE FIRST BIT OF THIS GROUP DEFINES WHETHER THE LITERAL IS TO BE ENTERED RIGHT JUSTIFIED INTO THE NAME STACK OR INTO THE VALUE STACK. THE REMAINING THREE BITS CONTAIN THE FORMAT SELECTOR WHICH IS USED AS AN INDEX INTO THE ARITHMETIC FORMAT VECTOR. THIS SELECTOR GIVES THE ARITHMETIC FORMAT OF THE LITERAL. THE REMAINDER OF THE PROGRAM SYLLABLE CONTAINS THE LITERAL.

IF THE FIRST TWO BITS OF THE OPERATION CODE DEFINE THAT THE OPERATOR IS A MEMBER OF THE AU OPERATOR CLASS, THE REMAINING SIX BITS OF THE OPERATOR DEFINE THE AU OPERATION TO BE PERFORMED. THESE SIX BITS WILL BE CHECKED BY THE INTERPRETER TO SEE IF THE NECESSARY PROCESSOR ENVIRONMENT IS PRESENT FOR THE AU OPERATION. IF SO, THE OPERATOR WILL THEN BE PASSED TO THE AU FOR EXECUTION.

IF THE FIRST TWO BITS OF THE OPERATION CODE DEFINE THAT THE OPERATOR HAS A NAME ASSOCIATED WITH IT, THEN THE NEXT FIVE BITS DEFINE THE OPERATION TO BE PERFORMED. THE REMAINING BIT DEFINES WHETHER THE NAMED OBJECT IS CONTAINED IN THE TOP NAME STACK SLICE. IF IT IS, THEN THE NEXT EIGHT BITS GIVE THE DISPLACEMENT OF THE OBJECT WITHIN THAT SLICE. DEFAULT LEXIC LEVEL OR A SPECIFIED LEXIC LEVEL WILL BE USED TO FIND THE RELATIVE ADDRESS OF THE NAME. IF A DEFAULT LEXIC LEVEL IS USED, THEN THE NEXT EIGHT BITS DEFINE THE DISPLACEMENT OF THE NAME. OTHERWISE, A NAME FOLLOWS. NAMES ARE DESCRIBED IN PARAGRAPH 3.3.2.2 NAMES.

IF THE FIRST TWO BITS OF THE OPERATION CODE DEFINE THAT THE OPERATOR IS A MEMBER OF THE GENERAL OPERATOR CLASS, THE REMAINING SIX BITS OF THE OPERATION CODE DEFINE WHICH OPERATOR IN THE CLASS IS TO BE EXECUTED.

3.3.1.2 PROGRAM BUFFER

THE PURPOSE OF THE PROGRAM BUFFER SHALL BE TO MINIMIZE MEMORY FETCHES BY PROVIDING PROGRAM ASSOCIATIVELY TO THE PROCESSING MODULE. PRIOR TO INITIATING A MEMORY FETCH, THE ASSOCIATED HARDWARE OF THE PROGRAM BUFFER SHALL EXAMINE THE PROGRAM BUFFER TO DETERMINE IF THE BRANCH ADDRESS OR CONTIGUOUS PROGRAM ADDRESS RESIDES IN THE PROGRAM THE PROGRAM BUFFER SHALL HAVE A MAXIMUM STORAGE BUFFER. CAPABILITY OF 8 WORDS, EACH OF WHICH SHALL BE 64 BITS WIDE.

THE PROGRAM BUFFER SHALL BE ORGANIZED INTO TWO SECTIONS, EACH OF WHICH SHALL CONTAIN FOUR WORDS. EACH SECTION SHALL HAVE THE ABILITY TO BE EITHER WRITTEN INTO OR READ FROM INDEPENDENTLY OF THE OTHER SECTIONS. THE PROGRAM BUFFER (SEE FIGURE 5) SHALL CONSIST OF THE FOLLOWING HARDWARE:

A. WRITE ADDRESS REGISTER (WAR) - A WORD-ORIENTED, MULTI-PURPOSE REGISTER SPECIFYING THE LOCATION IN THE PROGRAM BUFFER WHERE THE BRANCH PROGRAM, THE CONTIGUOUS PROGRAM, OR THE NEW BLOCK OF PROGRAM HAS BEEN STORED.

B. READ ADDRESS REGISTER (RAR) - A BIT-ORIENTED REGISTER SPECIFYING THE LOCATION OF THE NEXT EXECUTABLE INSTRUCTION IN THE PROGRAM BUFFER.

C. STORAGE REGISTER (SR) - RECORDS THE SEGMENT NUMBER AND MEMORY STOP ADDRESS OF THE PROGRAM PRESENTLY BEING EXECUTED.

D. PROGRAM VECTORS, PRESENCE BITS - EACH SECTION OF THE PROGRAM BUFFER SHALL HAVE A CORRESPONDING PROGRAM VECTOR WHICH SHALL RECORD THE SEGMENT NUMBER, MEMORY START ADDRESS, AND MEMORY STOP ADDRESS OF THE PROGRAM STORED IN THAT SECTION, AND A CORRESPONDING PRESENCE BIT WHICH SHALL BE MARKED TRUE WHEN PROGRAM IS STORED IN THAT SECTION.

E. COMPARATOR - THE COMPARATOR SHALL EXAMINE THE PROGRAM VECTORS TO DETERMINE IF THE BRANCH ADDRESS OR THE CONTIGUOUS PROGRAM ADDRESS RESIDES WITHIN THE PROGRAM BUFFER.

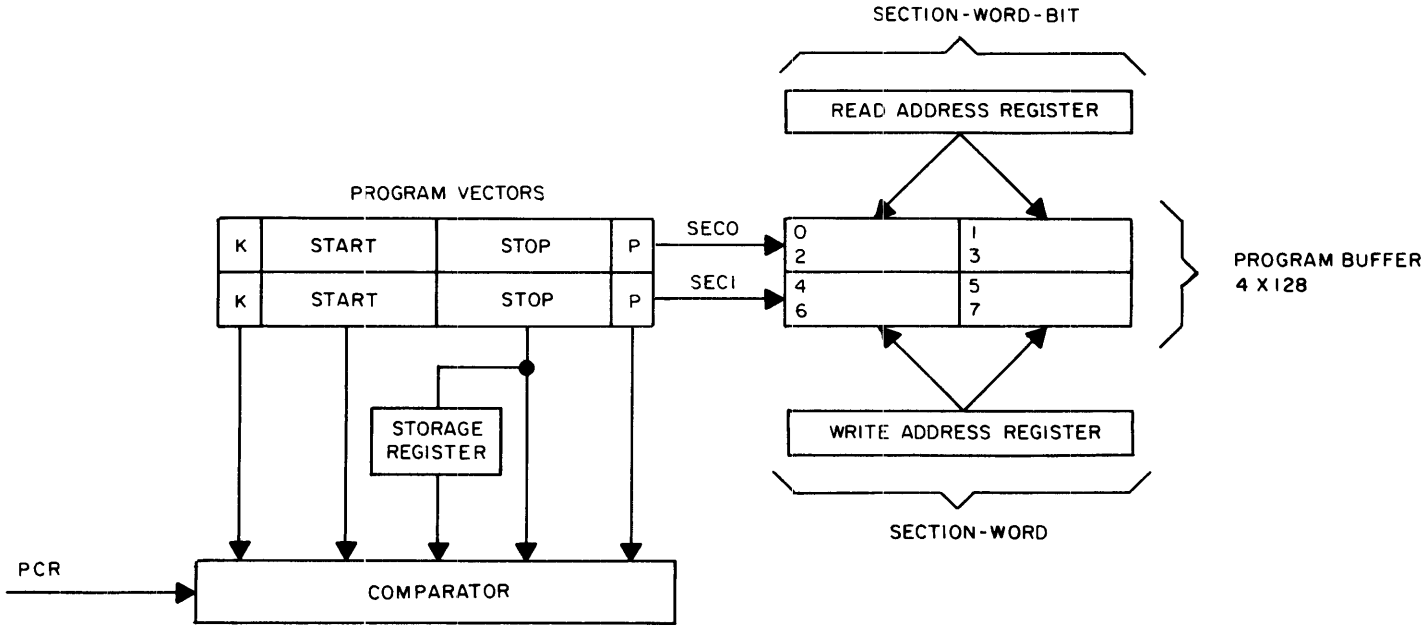
TO ACHIEVE ITS PURPOSE THE PROGRAM BUFFER SHALL UTILIZE SEVERAL HARD ROUTINES. THESE ROUTINES ARE AS FOLLOWS:

A. WAR ROUTINE - THIS ROUTINE SHALL UPDATE THE WRITE ADDRESS REGISTER TO REFLECT THE LOCATION IN THE PROGRAM BUFFER WHERE THE NEW BLOCK OF PROGRAM IS TO BE STORED.

B. FETCH POINT SCAN ROUTINE - THIS ROUTINE SHALL SCAN THE READ ADDRESS WORD REGISTER UNTIL THE REGISTER IS EQUAL TO OR IS GREATER THAN THE FETCH POINT. WHEN EITHER CONDITION EXIST, THE DECISION TO INITIATE OR NOT TO INITIATE A MEMORY FETCH IS MADE.

C. CONTIGUOUS PROGRAM SEARCH ROUTINE - THIS ROUTINE SHALL DETERMINE IF SUFFICIENT PROGRAM RESIDES IN THE PROGRAM BUFFER TO CONTINUE PROGRAM EXECUTION UPON COMPLETION OF PROGRAM IN THE PRESENT SECTION.

D. BRANCH ADDRESS SEARCH ROUTINE - THIS ROUTINE SHALL DETERMINE IF THE BRANCH ADDRESS RESIDES WITHIN THE PROGRAM BUFFER PRIOR TO INITIATING A MEMORY FETCH.



NOTE:

* K=SEGMENT NUMBER
  P=PRESENCE BIT

FIGURE 5. PROGRAM BUFFER

3.3.1.3 PROGRAM BARREL

THE PROGRAM BARREL SHALL PERFORM THE FOLLOWING FUNCTIONS:

A. ALIGNMENT OF INPUT FROM PROGRAM BUFFER (INPUT SHALL CONSIST OF TWO 64-BIT WORDS).

B. SELECTION AND ISOLATION OF AN 8-BIT OPERATION CODE.

C. SELECTION AND ISOLATION OF A VARIABLE LENGTH LITERALS AND NAMES.

D. FAN OUT OF SHIFTER OUTPUTS TO ALL NATURAL DATA DESTINATIONS.

3.3.1.4 PROGRAM CONTROLS

THE PROGRAM CONTROLS SHALL CONSIST OF DECODING AND ENCODING MECHANISMS, CONTROL MECHANISMS, AND TIMING MECHANISMS THAT ARE NEEDED TO PERFORM THE FOLLOWING FUNCTIONS:

A. TO DETERMINE THE CLASS OF OPERATOR SPECIFIED BY THE PROGRAM SYLLABLE.

B. TO DETERMINE THE LITERAL SIZE SPECIFIED BY THE LITERAL OPERATOR.

C. TO TRANSLATE A NAME, SPECIFIED BY THE NAME OPERATOR, INTO A TERMINAL REFERENCE.

D. TO DETERMINE THE OPERATION TO BE PERFORMED AS SPECIFIED BY A NAME OPERATOR OR A GENERAL OPERATOR.

E. TO PASS THE ARITHMETIC OPERATION FIELD OF THE ARITHMETIC OPERATOR TO THE ARITHMETIC UNIT FOR EXECUTION.

F. TO INSURE THAT THE NECESSARY PROCESSOR ENVIRONMENT IS PRESENT PRIOR TO THE EXECUTION OF THE PROGRAM SYLLABLE.

G. TO INTERACT WITH THE INTERRUPT SECTION, ARITHMETIC CONTROLS, AND DESCRIPTOR CONTROLS TO INSURE THAT THE PROPER SUBSEQUENCE OF OPERATIONS IS PERFORMED.

3.3.2 KERNEL SECTION

THIS SECTION DESCRIBES THE REPRESENTATION OF DESCRIPTORS. THE EVALUATION OF DESCRIPTORS AND THE HARDWARE ASSOCIATED WITH DESCRIPTOR EVALUATION.

3.3.2.1 DESCRIPTOR REPRESENTATION.

DESCRIPTORS ARE A MEANS OF CALCULATING A REFERENCE TO AN ELEMENT. A DESCRIPTOR CAN REFERENCE PROGRAM SEGMENTS, DATA OBJECTS, LOCKED DATA FIELDS OR ANOTHER DESCRIPTOR. THE CHARACTERISTICS OF THE REFERENCED ELEMENTS ARE DEFINED AS INTERPRETER ATTRIBUTES. DESCRIPTORS ALSO SPECIFY WHETHER THE ELEMENT REFERENCED CAN BE STORED OR FETCHED. THIS PROTECTION CAPABILITY IS DEFINED BY THE ACCESS ATTRIBUTES OF THE DESCRIPTOR. A PARTICULAR ELEMENT MAY ALSO BE NESTED WITHIN SEVERAL STRUCTURES. EACH STRUCTURE WHICH MUST BE ACCESSED IS SPECIFIED BY A STRUCTURE EXPRESSION. THE STRUCTURE EXPRESSION CONTAINS A STRUCTURE TYPE FIELD WHICH DEFINES THE STRUCTURE AND STRUCTURE PARAMETER FIELDS WHICH GIVE THE PARAMETERS NECESSARY FOR ACCESSING THE STRUCTURE. A DESCRIPTOR CONTAINS AS MANY STRUCTURE EXPRESSIONS AS NECESSARY TO OBTAIN THE DESIRED ELEMENT.

3.3.2.1.1 DESCRIPTOR ATTRIBUTES

THE FIRST TWO BITS (R AND W) OF EACH DESCRIPTOR (SEE FIGURE 6) ARE ACCESS FAULT PERMISSION BITS. BITS R AND W DEFINE READ ACCESS FAULTS AND WRITE ACCESS FAULTS, RESPECTIVELY. IF THE R BIT IS SET DURING A FETCH TO LEVEL-1, THE READ FAULT PROCEDURE NAMED IN THE DESCRIPTOR IS CALLED. LIKEWISE, IF THE W BIT IS SET DURING A STORE TO LEVEL-1, THE STORE FAULT PROCEDURE NAMED IN THE DESCRIPTOR IS CALLED.

THE NEXT TWO BITS OF THE DESCRIPTOR START THE INTERPRETER ATTRIBUTE FIELD. THESE BITS DEFINE WHETHER THE DESCRIPTOR POINTS TO PROGRAM (0,1), DATA (1,0), LOCKED DATA (1,1) OR ANOTHER DESCRIPTOR (0,0). THE INTERPRETER ATTRIBUTE FIELD DESCRIBING DATA OR LOCKED DATA IS COMPLETED WITH THE MU BIT AND THE FORMAT SELECTOR FIELD. IF THE MU BIT IS SET, THE DATA ELEMENT REFERENCED RESIDES IN THE VALUE STACK. THE FORMAT SELECTOR FIELD IS A THREE-BIT INDEX INTO THE ARITHMETIC FORMAT VECTOR. THE SELECTED ENTRY IN THIS VECTOR DEFINES THE FORMAT OF THE DATA ELEMENT. A FORMAT SELECTOR INDEX OF VALUE ZERO DEFINES A NULL OR DEFAULT FORMAT; THAT IS, USE THE MOST RECENTLY DEFINED FORMAT.

IF THE DESCRIPTOR POINTS TO PROGRAM, THEN A PARAMETER BIT (P) A

FUNCTION BIT (F) A LEXIC LINK LENGTH FIELD AND A LEXIC LINK FIELD ARE NECESSARY TO COMPLETE THE INTERPRETER ATTRIBUTE FIELD. THE P BIT IS USED TO INDICATE THAT THE PROGRAM BEING ENTERED HAS PARAMETERS. THE F BIT IS USED TO INDICATE THAT THE PROGRAM BEING ENTERED LEAVES A RESULT. THE LEXIC LINK LENGTH FIELD DEFINES THE LENGTH OF THE LEXIC LINK FIELD WHICH IS 4, 8, 12 OR 16 BITS. THE LEXIC LINK IS AN INDEX TO THE DISPLAY CONTROL WORD (DCW) OF THE NAME STACK SLICE IN WHICH THE PROGRAM HAS BEEN DECLARED. THIS INFORMATION IS USED FOR DISPLAY UPDATE.

IF THE DESCRIPTOR POINTS TO ANOTHER DESCRIPTOR, THEN THE INTERPRETER ATTRIBUTE FIELD CONTAINS ONLY TWO BITS. READ FAULT PROCEDURE NAMES AND WRITE FAULT PROCEDURE NAMES IF REQUIRED, AS INDICATED BY THE ACCESS PERMISSION FAULT BITS, FOLLOW THE INTERPRETER ATTRIBUTE FIELD. THE REMAINDER OF THE DESCRIPTION CONTAINS STRUCTURE EXPRESSIONS.

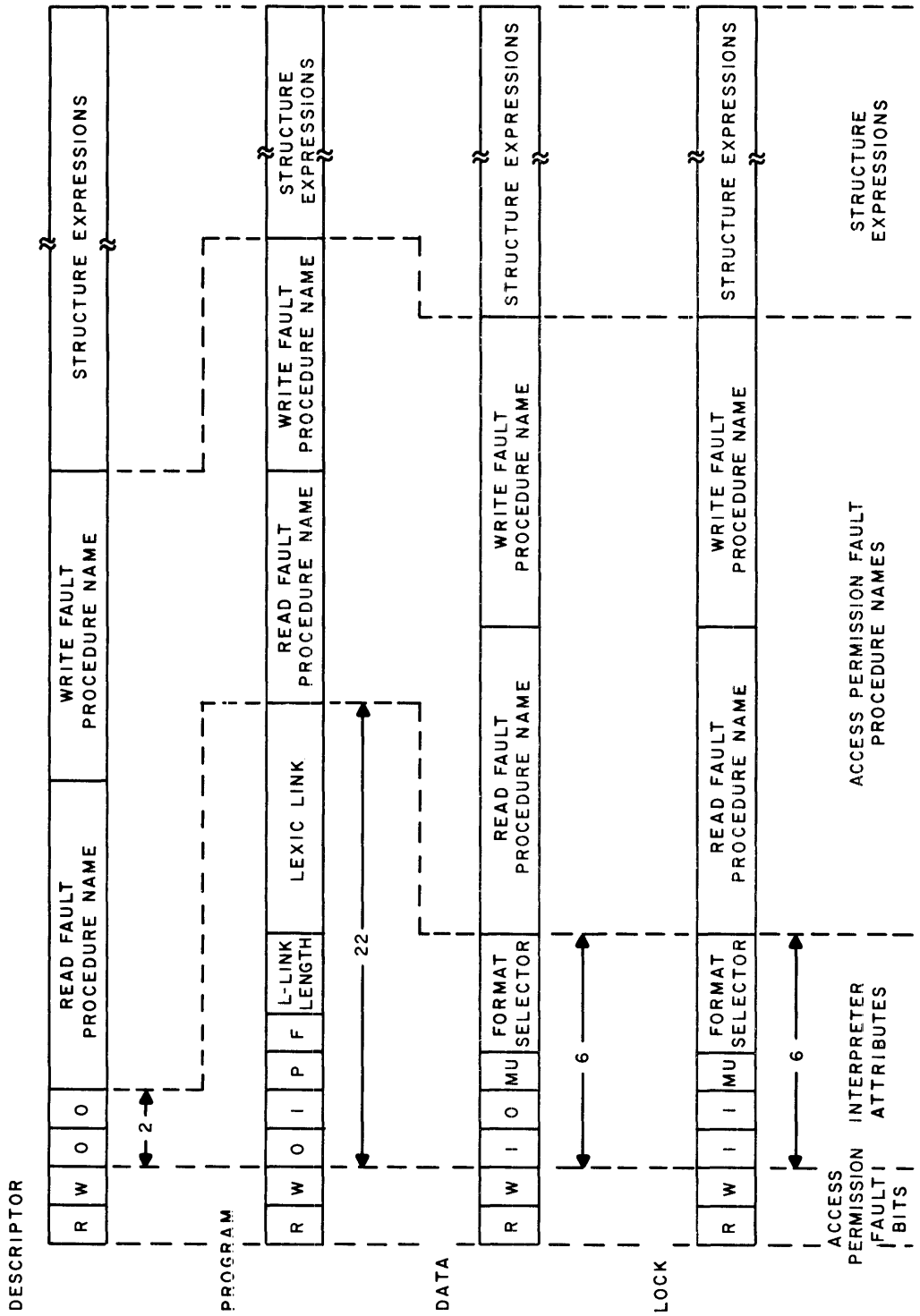


FIGURE 6. DESCRIPTOR, REPRESENTATION

3.3.2.1.2 STRUCTURE EXPRESSION

THE FIRST FOUR BITS OF THE STRUCTURE EXPRESSION FIELD (SEE FIGURE 7) IN THE DESCRIPTOR CONTAIN AN ADDRESS FIELD LENGTH (AFL) AND AN ALLOCATE BIT (A). THE AFL DEFINES THE SIZE (IN BITS) OF ALL STRUCTURE EXPRESSION PARAMETER FIELDS NOT PREDETERMINED BY STRUCTION EXPRESSION TYPES. THE REMAINDER OF THE STRUCTURE EXPRESSION FIELD CONTAINS STRUCTURE EXPRESSIONS SIMILAR TO THOSE ILLUSTRATED IN FIGURE 8. TWO STRUCTURE EXPRESSION TYPES (SEGMENT NUMBER AND CALL) HAVE STRUCUTRE EXPRESSION PARAMETERS OF PREDETERMINED SIZE. SEGMENT NUMBER ALWAYS HAS AN 8 BIT INDEX INTO THE RESOURCE STACK AS ITS PARAMETER. CALL ALWAYS HAS A NAME AS ITS PARAMETER. NAMES ARE DESCRIBED IN SECTION 3.3.2.2. THE LAST STRUCTURE EXPRESSION IS ALWAYS A DELIMITER. FIN.

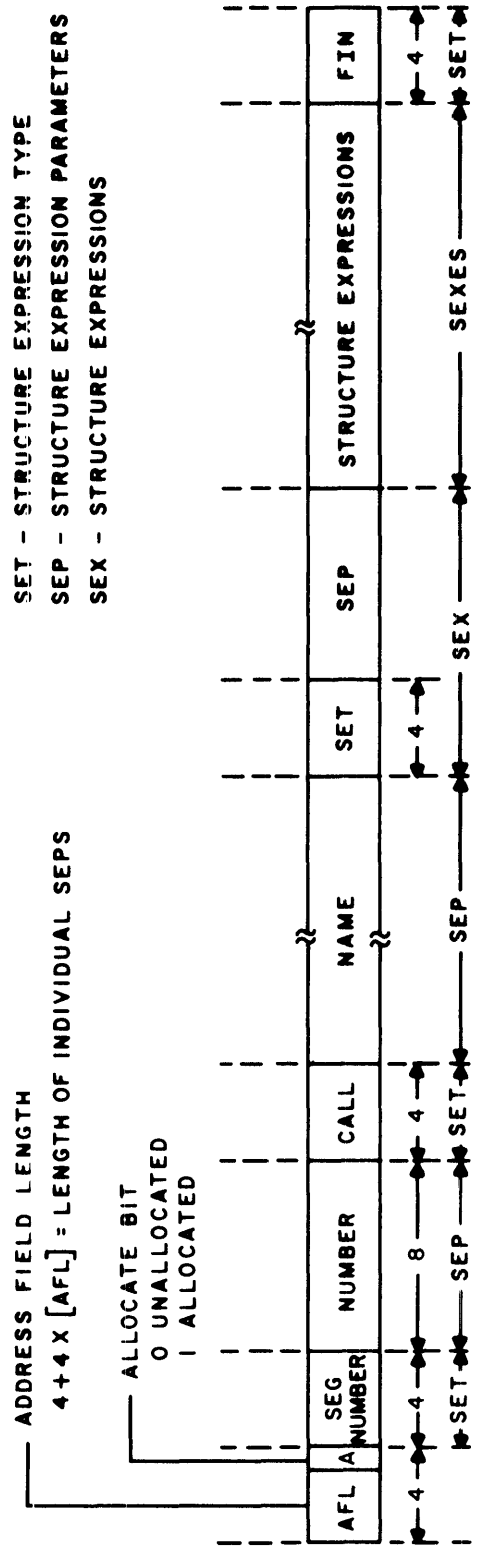


FIGURE 7. TYPICAL STRUCTURE EXPRESSION FIELD REPRESENTATION

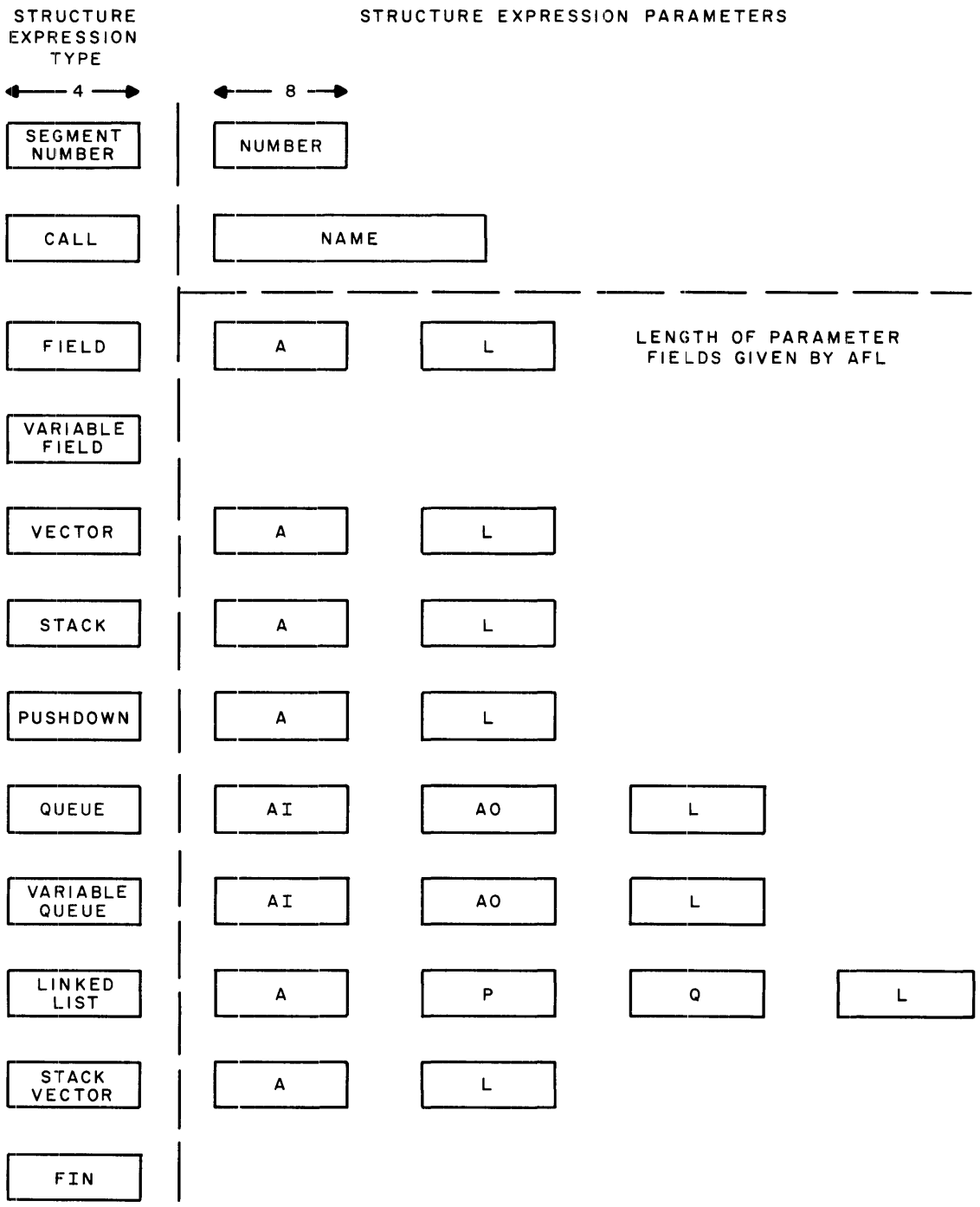


FIGURE 8, STRUCTURE EXPRESSSION PARAMETER FIELDS

3.3.2.2 NAMES

DESCRIPTORS TO BE EVALUATED ARE REFERENCED BY NAME. A NAME IS A SYMBOLIC REPRESENTATION OF THE LOCATION OF A DESCRIPTOR. THERE ARE SEVEN BASIC TYPES OF NAMES AS ILLUSTRATED IN FIGURE 9:

A. THE COROUTINE RELATIVE NAME, WHICH CONSISTS OF 3 INDICES: A STACK NUMBER (SNR), LEXIC LEVEL (LL), AND DISPLACEMENT (D).

B. THE DISPLAY RELATIVE NAME, WHICH CONSISTS OF 2 INDICES: A LEXIC LEVE (LL) AND DISPLACEMENT (D).

C. THE SLICE RELATIVE NAME, WHICH CONSISTS OF ONE INDEX: A DISPLACEMENT (D).

D. THE PROGRAM RELATIVE NAME, WHICH CONSISTS OF ONE INDEX: A DISPLACEMENT (D).

E. THE N-BASE RELATIVE NAME, WHICH CONSISTS OF 2 INDICES: A STACK NUMBER (SNR) AND DISPLACEMENT (D).

F. THE TOP OF NAME STACK NAME, WHICH HAS NO INDICES.

NAME TYPE	NAME TYPE CODE	STACK NUMBER	LEXIC LEVEL	DISPLACEMENT
COROUTINE RELATIVE	4	4	8	8
COROUTINE RELATIVE	4	4	4	12
DISPLAY RELATIVE	4		8	8
DISPLAY RELATIVE	4		4	12
SLICE RELATIVE	4			8
SLICE RELATIVE	4			12
PROGRAM RELATIVE	4			8
PROGRAM RELATIVE	4			16
N-BASE RELATIVE	4			8
N-BASE RELATIVE	4			16
COROUTINE N-BASE RELATIVE	4	4		8
COROUTINE N-BASE RELATIVE	4	4		16
TOP OF NAME STACK	4			

FIGURE 9. NAME REPRESENTATION

THE COROUTINE RELATIVE NAME REQUIRES THREE INDICES. THE FIRST INDEX SNR IS USED TO FIND THE PROPER LEVEL OF COROUTINES IN THE COROUTINE DISPLAY STACK. THE COROUTINE ENTRY THAT IS REFERENCED CONTAINS A DESCRIPTION OF THE DISPLAY. THE LL INDEX IS USED TO FIND AN ENTRY IN THE DISPLAY WHICH CONTAINS THE DESCRIPTON OF A SLICE WITHIN THE NAME STACK. THE D INDEX IS USED TO FIND AN ENTRY IN THE NAME STACK SLICE WHICH CONTAINS THE REFERENCED DESCRIPTOR.

SINCE MANY NAMES THAT ARE REFERENCED WILL BE IN THE ACTIVE NAME STACK, THE DISPLAY RELATIVE NAME CONTAINING TWO INDICES WILL BE AVAILABLE. THE TWO INDICES ARE LL, D. THE LL INDEX IS USED TO FIND AN ENTRY IN THE DISPLAY STACK WHICH CONTAINS THE DESCRIPTION OF A SLICE WITHIN THE NAME STACK. THE D INDEX IS USED TO FIND AN ENTRY IN THE NAME STACK SLICE WHICH CONTAINS THE REFERENCED DESCRIPTOR.

THE SLICE RELATIVE NAME CONSISTING OF JUST A D INDEX IS USED TO ADDRESS LOCALS AND PARAMETERS OF AN ACTIVE PROCEDURE. THE D INDEX IS USED TO FIND AN ENTRY IN THE TOP SLICE OF THE NAME STACK. THIS ENTRY CONTAINS THE REFERENCED DESCRIPTOR.

IF THE NAME IS A PROGRAM RELATIVE NAME, THEN A DISPLACEMENT INDEX IS USED TO FIND THE LOCATION OF THE REFERENCED DESCRIPTOR. THIS LOCATION IS RELATIVE TO THE ACTIVE PROGRAM OR DESCRIPTOR CONTAINER.

IF THE NAME IS AN N-BASE RELATIVE NAME, THEN A DISPLACEMENT INDEX IS USED TO FIND THE LOCATION OF THE REFERENCED DESCRIPTOR. THIS LOCATION IS RELATIVE TO THE BASE OF THE ACTIVE NAME STACK.

IF THE NAME IS A COROUTINE N-BASE RELATIVE NAME THEN TWO INDICES ARE REQUIRED. THE FIRST INDEX SNR IS USED TO FIND AN ENTRY IN THE COROUTINE DISPLAY STACK. THIS ENTRY REFERENCES A COROUTINE CONTROL FIELD. THE D INDEX IS USED TO FIND THE LOCATION OF THE REFERENCE DESCRIPTOR. THIS LOCATION IS RELATIVE TO THE BASE OF THE NAME STACK WHICH IS SPECIFIED IN THE COROUTINE CONTROL FIELD.

IF THE NAME IS A TOP OF NAME STACK NAME, THEN THE DESCRIPTOR REFERENCED WILL BE FOUND IN THE TOP ENTRY IN THE NAME STACK.

THE LEXIC LEVEL AND DISPLACEMENT FIELDS VARY IN SIZE TO ALLOW FOR CODE COMPACTION WHEREVER POSSIBLE.

3.3.2.3. DESCRIPTOR EVALUATION

DESCRIPTORS ARE REFERENCED BY NAME. AFTER THE NAME OF THE DESCRIPTOR HAS BEEN EVALUATED AND THE LOCATION OF THE DESCRIPTOR CALCULATED, THE DESCRIPTOR IS FETCHED FROM LEVEL-1 AND PLACED IN THE DESCRIPTOR BUFFER FOR EVALUATION. THERE ARE THREE MODES FOR DESCRIPTOR EVALUATION: CONSTRUCT, ENTER AND REMOVE. CONSTRUCT

CALCULATES A REFERENCE TO AN ELEMENT IN A STRUCTURE. ENTER ALLOCATES SPACE FOR AN ELEMENT IN A STRUCTURE. REMOVE DEALLOCATES AN ELEMENT SPACE IN A STRUCTURE. IF A STRUCTURE IS SUCH THAT SPACE CANNOT BE ALLOCATED OR REMOVED (E.G. FIELD, VECTOR) THEN A CONSTRUCT IS PERFORMED. THERE ARE TWO ADDITIONAL MODES; RESET AND SEQUENCE WHICH ARE ONLY AVAILABLE FOR LIST STRUCTURES. APPENDIX A CONTAINS THE ALGORITHMS FOR CONSTRUCT, ENTER AND REMOVE OF EACH OF THE AVAILABLE STRUCTURES.

THERE ARE TWO COMPOSITE STRUCTURES, PUSHDOWN-STACK AND PUSHDOWN-PUSHDOWN THAT ARE KNOWN TO THE HARDWARE, BUT NON-SPECIFIABLE TO THE SOFTWARE. THAT IS, THERE ARE CERTAIN KNOWN STRUCTURES THAT WILL BE CLASSIFIED AS PUSHDOWN-STACK OR PUSHDOWN-PUSHDOWN. (I.E. NAME STACK, RESOURCE STACK AND VALUE STACK) THAT WILL BE MANAGED BY THE HARDWARE AND USED TO STRUCTURE THE CENTRAL PROCESSOR. ALGORITHMS FOR CONSTRUCT, ENTER AND REMOVE ON THESE STRUCUTRES ARE ALSO INCLUDED IN APPENDIX A.

AFTER THE DESCRIPTOR IS LOADED IN THE DESCRIPTOR BUFFER, THE FIRST FOUR BITS OF THE DESCRIPTOR ARE EXTRACTED AND EXAMINED USING THE DESCRIPTOR IMPLODE-EXPLODE MECHANISM. THE REMAINDER OF THE INTERPRETER ATTRIBUTE FIELD IS EXTRACTED AND PLACED IN THE APPROPRIATE FIELDS OF THE PROGRAM/DESCRIPTOR CONTROL REGISTER. INTERPRETER ATTRIBUTES ARE NESTED DURING DESCRIPTOR CALLS SO THAT THE ATTRIBUTES OF THE FIRST DESCRIPTOR REFERENCED DURING THE EVALUATION PROCESS ARE USED. INTERPRETER ATTRIBUTES OF TYPE DESCRIPTOR IMPLY AN AUTOMATIC BRANCH TO THE LOCATION CALCULATED AT THE END OF THE DESCRIPTOR EVALUATION IN ORDER TO CONTINUE THE EVALUATION PROCESS. THEREFORE AT BRANCH TIME, THE INTERPRETER ATTRIBUTES ARE OVERWRITTEN IN THE PDCR.

IF ACCESS FAULT PROCEDURES ARE DEFINED, THEY ARE EXTRACTED FROM THE DESCRIPTOR AND PLACED IN THE ATTRIBUTE COLLECTION STACK. IF NEW ACCESS PERMISSION FAULT PROCEDURES ARE ENCOUNTERED DURING DESCRIPTOR EVALUATION, THE CONTENTS OF THE ATTRIBUTE COLLECTION STACK ARE OVERWRITTEN WITH THE NEW PROCEDURE NAME. IF A DESCRIPTOR WHICH WILL NOT INVOKE AN ACCESS FAULT PROCEDURE IS ENCOUNTERED DURING EVALUATION, THEN THE EXISTING FAULT PROCEDURE NAMES IN THE ATTRIBUTE COLLECTION STACK ARE RETAINED. THE ACCESS PERMISSION FAULT WILL BE INVOKED DURING FETCH AND STORE OPERATIONS TO LEVEL-1 STORAGE. IF THE INTERPRETER ATTRIBUTE TYPE OF A DESCRIPTOR IS A DESCRIPTOR, THE ACCESS PERMISSION FAULT PROCEDURES ARE INVOKED AT BRANCH TIME. ALSO, THE ATTRIBUTE COLLECTION STACK IS CLEARED OF THE ACCESS PERMISSION FAULT PROCEDURES AT BRANCH TIME. AFTER THE DESCRIPTOR ATTRIBUTE FIELDS HAVE BEEN EVALUATED, THE FIRST FOUR BITS OF THE STRUCTURE EXPRESSION FIELD ARE EXTRACTED FROM THE DESCRIPTOR AND PLACED IN THE PROGRAM/DESCRIPTOR CONTROL REGISTER. THE FIRST THREE BITS ARE THE AFL FIELD WHICH DEFINES THE LENGTH OF THE STRUCTURE EXPRESSION PARAMETER FIELDS. THE FOURTH BIT IS AN ALLOCATE BIT. IF THIS BIT IS OFF, THE DESCRIPTOR EVALUATION STOPS AND AN ALLOCATE FAULT IS SET.

THE STRUCTURE EXPRESSION FIELDS ARE EVALUATED FOR TERMINAL REFERENCES. THE FIRST FOUR BITS OF EACH STRUCTURE EXPRESSION FIELD ARE PLACED IN THE DESCRIPTOR EXECUTION REGISTER (DER.) THE STRUCTURE PARAMETER FIELDSARE EXTRACTED AND PLACED IN THE APPROPRIATE ATTRIBUTE STACKS. IF THE STRUCTURE EXPRESSION TYPE IS A SEGMENT NUMBER, THE PROPER ENTRY IN THE ATTRIBUTE COLLECTION STACK IS EXAMINED TO SEE IF IT IS EMPTY OR FULL. IF IT IS EMPTY, THE NEXT EIGHT-BIT FIELD CONTAINING THE NUMBER ITSELF IS EXTRACTED FROM THE STRUCTURE EXPRESSION AND PLACED IN THE ATTRIBUTE COLLECTION STACK. IF THE ENTRY IS FULL, THE SEGMENT NUMBER IS IGNORED. IF THE STRUCTURE EXPRESSION TYPE IS CALL, THE NEXT FOUR BITS DEFINE THE NAME TYPE AND ARE EXTRACTED FROM THE STRUCTURE EXPRESSION. THE REMAINDER OF THE NAME IS EVALUATED. THE PDCR OF THE CURRENT DESCRIPTOR IS NESTED IN THE PROGRAM/DESCRIPTOR CONTROL STACK AND THE REFERENCE OF THE NEW DESCRIPTOR IS PLACED IN THE PDCR. THE DESCRIPTOR EVALUATION PROCESS IS THEN REPEATED. THE REMAINDER OF THE STRUCTURE EXPRESSION TYPES ARE EVALUATED AS DESCRIBED IN APPENDIX A.

UPON REACHING "FIN", THE PDCS IS POPPED AND THE CONTENTS OF THE PDCR ARE EXAMINED. IF THE DESCRIPTOR IN THE PDCR INDICATES THAT A DESCRIPTOR IS BEING EVALUATED, THEN THAT DESCRIPTOR IS FETCHED FROM LEVEL-1 AND THE EVALUATION OF THAT DESCRIPTOR RESUMES AT THE LOCATION SPECIFIED BY THE PDCR. OTHERWISE, THE CONTENTS OF THE ATTRIBUTE COLLECTION STACKS, THE ATTRIBUTE STACKS, AND THE INTERPRETER ATTRIBUTES ARE LINKED TOGETHER TO FORM A TERMINAL REFERENCE IN THE NAME STACK TO THE OBJECT BEING REFERENCED. THE NEXT PROGRAM OPERATOR IS THEN EXECUTED.

3.3.2.4 KERNEL HARDWARE.

THE KERNEL HARDWARE PERFORMS THE DESCRIPTOR EVALUATION FUNCTIONS. THE KERNEL HARDWARE IS ILLUSTRATED IN FIGURE 10. THE KERNEL HARDWARE CONSISTS OF THE FIVE ATTRIBUTE STACKS, THE DESCRIPTOR IMPLODE-EXPLODE MECHANISM, THE PROGRAM/DESCRIPTOR CONTROL REGISTER, THE DESCRIPTOR EXECUTION REGISTER AND CONTROLS. THE KERNEL RECEIVES DATA FROM THE STRUCTURE DESCRIPTOR BUFFERS, VALUE STACK BUFFERS, PROGRAM BARREL AND THE ARITHMETIC UNIT. THE KERNEL SENDS DATA TO THE STRUCTURE AND DESCRIPTOR BUFFERS AND TO THE ARITHMETIC UNIT.

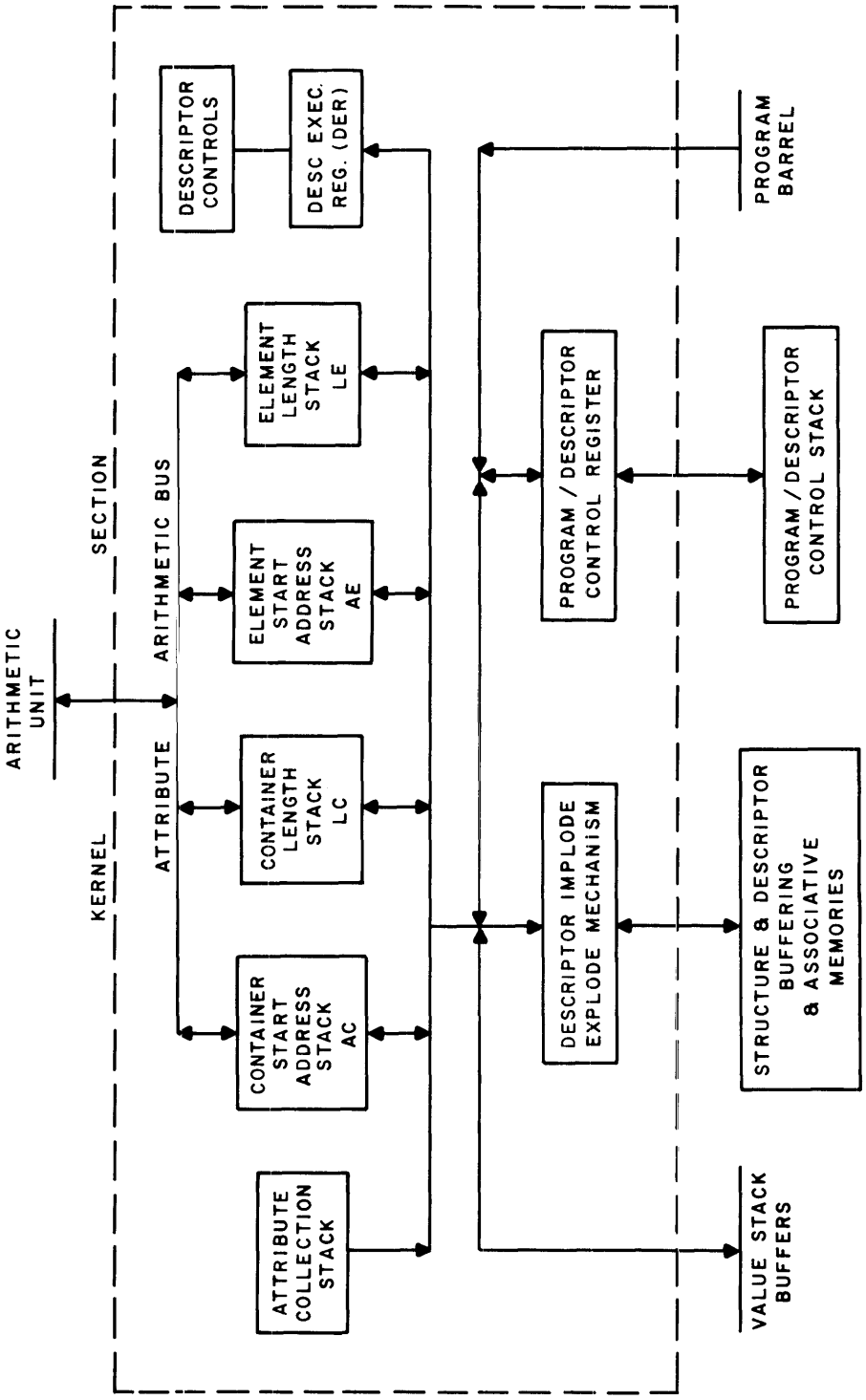


FIGURE 10. KERNEL, BLOCK DIAGRAM

3,3,2,4,1 ATTRIBUTE STACKS.

THE ATTRIBUTE STACKS ARE USED TO COLLECT ACCESS PERMISSION ATTRIBUTES, SEGMENT NUMBERS, FORMAT SELECTORS, AND TO PROVIDE WORKING STORAGE TO HOLD THE STRUCTURE EXPRESSION PARAMETERS DURING STRUCTURE EXPRESSION EVALUATION. THE ATTRIBUTE STACKS ARE DIVIDED INTO AN ATTRIBUTE COLLECTION STACK AND FOUR ATTRIBUTE ARITHMETIC STACKS.

THE ATTRIBUTE COLLECTION STACK (SEE FIGURE 11) CONSISTS OF FOUR WORDS, EACH OF WHICH IS 16 BITS WIDE. THE FIRST WORD CONTAINS ACCESS PERMISSION BITS, SEGMENT NUMBER AND FORMAT SELECTOR EMPTY/FULL BITS, DEFAULT FORMAT SELECTOR AND NAME TYPE FIELD, AND STACK NUMBER FIELD OF THE READ FAULT PROCEDURE NAME. THE SECOND WORD CONTAINS THE LEXIC LEVEL AND DISPLACEMENT FIELDS OF THE READ FAULT PROCEDURE NAME. THE THIRD WORD CONTAINS DEFAULT SEGMENT NUMBER, NAME TYPE FIELD AND STACK NUMBER FIELD OF THE WRITE FAULT PROCEDURE NAME. THE LAST WORD IN THE ATTRIBUTE COLLECTION STACK CONTAINS THE LEXIC LEVEL AND DISPLACEMENT FIELDS OF THE WRITE FAULT PROCEDURE NAME.

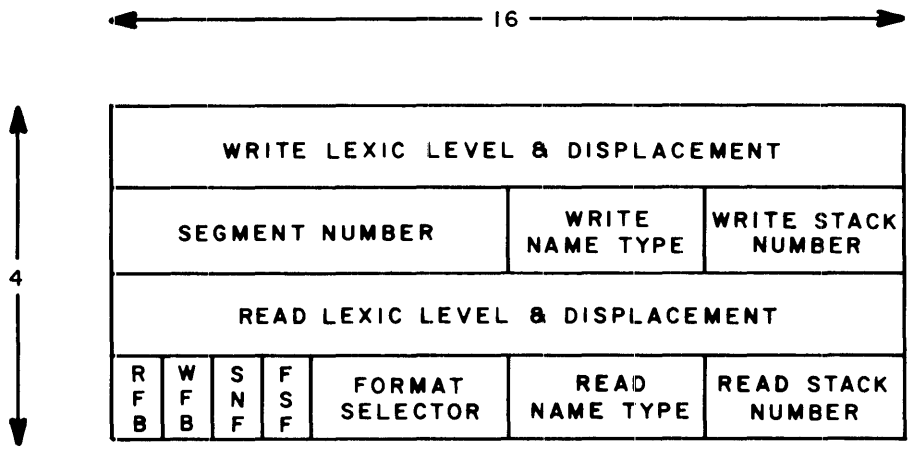


FIGURE 11. ATTRIBUTE COLLECTION STACK

THE OTHER FOUR STACKS ARE USED FOR STRUCTURE EXPRESSION PARAMETER MANIPULATION. EACH STACK CONSISTS OF FOUR WORDS, EACH THIRTY TWO BITS LONG. THESE STACKS INTERFACE WITH THE ARITHMETIC UNIT FOR ALL CALCULATIONS. THEY ALSO UTILIZE AND MODIFY THE STRUCTURE EXPRESSIONS IN THE STRUCTURE AND DESCRIPTOR BUFFERS, AND THEY RECEIVE PARAMETERS FROM THE VALUE STACK AND PROGRAM BARREL. THE STACKS MAY BE MANIPULATED INDIVIDUALLY. IN GENERAL, TWO OF THE STACKS HOLD CONTAINER INFORMATION (STARTING ADDRESS AND LENGTH)

WHILE THE REMAINING TWO STACKS HOLD ELEMENT INFORMATION (STARTING ADDRESS AND LENGTH). DURING EVALUATION THE STACKS WILL HOLD INTERMEDIATE VALUES, SUCH AS CONTAINERS FOR LENGTH INFORMATION IN SELF IDENTIFYING STRUCTURES. AT THE END OF EVERY STRUCTURE TYPE EVALUATION, THE ELEMENT STACKS WILL BE EMPTY WHILE THE CONTAINER STACKS WILL HAVE A PARTIAL REFERENCE TO THE OBJECT OF THE DESCRIPTOR. THE PARTIAL REFERENCE IS A CONTAINER ADDRESS AND LENGTH CORRESPONDING TO THE POINT UP TO WHICH THE DESCRIPTOR HAS BEEN EVALUATED.

3.3.2.4.2 DESCRIPTOR EXECUTION REGISTER

THE DESCRIPTOR EXECUTION REGISTER (DER) RETAINS THE CURRENT DESCRIPTOR STRUCTURE EXPRESSION TYPE FIELD IN ORDER THAT IT MAY BE USED WITH INFORMATION FROM THE INTERPRETER CONTROL SECTION IN DETERMINING THE ALGORITHM THAT IS TO BE USED BY THE DESCRIPTOR CONTROL SECTION IN THE CURRENT STRUCTURE EXPRESSION. THE DESCRIPTOR EXECUTION REGISTER IS FOUR BITS IN LENGTH.

3.3.2.4.3 DESCRIPTOR IMPLODE-EXPLODE MECHANISM

THE DESCRIPTOR IMPLODE-EXPLODE MECHANISM SERVES TWO FUNCTIONS; IT IS USED TO UNPACK FIELDS IN DESCRIPTORS AND PRESENT EACH FIELD TO ITS APPROPRIATE DESTINATION. IT IS ALSO USED TO UPDATE AND REPACK FIELDS FROM VARIOUS SOURCES TO FORM AND UPDATE DESCRIPTORS.

3.3.2.4.4 PROGRAM/DESCRIPTOR CONTROL REGISTER

THE PROGRAM/DESCRIPTOR CONTROL REGISTER IS DISCUSSED IN PARAGRAPH 3.3.3.4 PROGRAM/DESCRIPTOR CONTROL STRUCTURE.

3.3.2.4.5 STRUCTURE AND DESCRIPTOR BUFFER

THE STRUCTURE AND DESCRIPTOR BUFFER AND ASSOCIATIVE MEMORY (SEE FIGURE 12), WHILE NOT PART OF THE KERNEL HARDWARE, PROVIDE THE KERNEL WITH THE DESCRIPTORS THAT ARE TO BE EVALUATED. THE BUFFER IS A 32 WORD BY 128-BIT LOCAL MEMORY. THE BUFFER IS DIVIDED INTO FIVE AREAS; COROUTINE CONTROL FIELD BUFFER, NAME STACK BUFFERS, DESCRIPTOR BUFFERS, RESOURCE STACK BUFFERS, AND DISPLAY BUFFERS. THE DESCRIPTOR RESOURCE STACK AND DISPLAY BUFFERS HAVE AN ASSOCIATIVE MEMORY IN ORDER TO QUICKLY REFERENCE CAPTURED ENTRIES. THE COROUTINE CONTROL FIELD ENTRIES AND NAME STACK ENTRIES HAVE THEIR LEVEL-1 ADDRESSES STORED IN THE ASSOCIATIVE MEMORY FOR QUICK

UPDATE. A DETAILED DESCRIPTION OF THE DIFFERENT STRUCTURES AND HOW THEY ARE USED IS GIVEN IN PARAGRAPH 3.3.3 STRUCTURE BUFFERING SECTION.

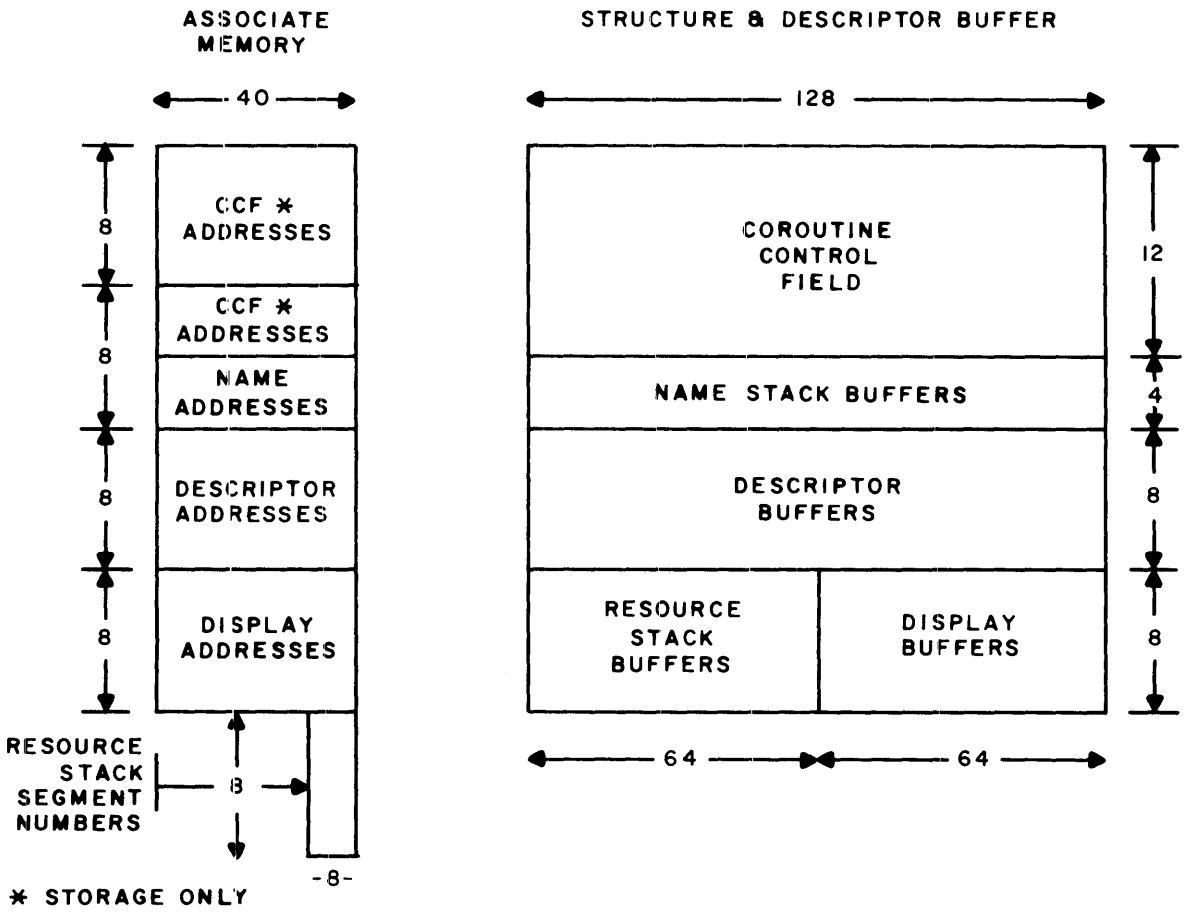


FIGURE 12. STRUCTURE AND DESCRIPTOR BUFFER AND ASSOCIATIVE MEMORY

3.3.3 STRUCTURE BUFFERING SECTION

STRUCTURES THAT DEFINF THE PROCESSOR ARE AS FOLLOWS:

A. RESOURCE CONTROL STRUCTURE

B. PROCEDURE CONTROL STRUCTURE

C. COROUTINE CONTROL STRUCTURE

D. PROGRAM CONTROL STRUCTURE

THE CPM HAS LOCAL BUFFERING FOR THESE STRUCTURES IN ORDER TO REDUCE MEMORY ACCESSES AND ENHANCE OPERATING SPEEDS. THE MECHANISMS AND INTERACTIONS OF THESE STRUCTURES ARE DISCUSSED IN THE GENERAL SPECIFICATION FOR THE B8500 SYSTEM (CP 1720-2045).

3.3.3.1 RESOURCE CONTROL STRUCTURE

THE RESOURCE CONTROL STRUCTURE DEFINES THE RESOURCES ALLOTTED TO A SPECIFIC PROCESS. THESE RESOURCES INCLUDE THE FOLLOWING:

A. SEGMENT CONTAINERS IN LEVEL-1

B. SEGMENT CONTAINERS IN LEVEL-2

C. DEVICES IN LEVEL-3

D. PROCESSOR TIME

E. FAULT MASKS

H. PROCESS ENVIRONMENT DESCRIPTION

ALL THE RESOURCES FOR A GIVEN PROCESS ARE DEFINED IN A SLICE OF THE RESOURCE STACK. A SLICE IS TREATED AS A VECTOR. THE INDIVIDUAL RESOURCE ENTRIES IN A SLICE ARE ACCESSED BY USING THE SEGMENT NUMBER FIELD IN THE DESCRIPTOR STRUCTURE-EXPRESSION AS AN INDEX. THE LEVEL-1 RESOURCE STACK MAP IS ILLUSTRATED IN FIGURE 13.

INDIVIDUAL RESOURCE FNTRIES ARE USED TO SET PROCESS-ENVIRONMENT AT PROCESS CALL, STORE PROCESS ENVIRONMENT AT PROCESS EXIT, PROVIDE STORAGE LEVEL CONTROL, AND TO SUPPLY ABSOLUTE CONTAINERS DURING FINAL COMBINE. THE ABSOLUTE CONTAINERS ARE NEEDED IN ORDER TO CALCULATE ABSOLUTE ADDRESSES. THE FORMAT OF EACH TYPE OF RESOURCE STACK ENTRY IS ILLUSTRATED IN FIGURE 14.

THE LEVEL-1 RESOURCE STACK SLICE OF AN ACTIVE PROCESS MAPS INTO THE

PROCESSOR AS ILLUSTRATED IN FIGURE 15 AND AS DESCRIBED BELOW:

THE FIRST ENTRY, WHICH CONTAINS THE PROCESS ENVIRONMENT DESCRIPTOR, IS PLACED IN THE FIRST ENTRY OF THE RESOURCE STACK BUFFER (RSB). HARDWARE STATE INFORMATION AND THE CONDITIONAL HALT REGISTER, WHICH ARE PART OF THE PROCESS ENVIRONMENT, ARE ESTABLISHED IN THE PROCESSOR.

THE NEXT THREE ENTRIES, WHICH CONTAIN PROCESSOR STATE INFORMATION, ARE PLACED INTO THE APPROPRIATE REGISTERS IN THE INTERRUPT SECTION.

THE REMAINING ENTRIES, WHICH ARE LEVEL-1 CONTAINERS, LEVEL-2 CONTAINERS, AND LEVEL-3 DEVICE NUMBERS, ARE CAPTURED UPON ACCESS IN THE RSB. THE RSB IS A LOCAL MEMORY WITH EIGHT ENTRIES, EACH OF WHICH IS 64 BITS WIDE. FOR EACH ENTRY IN THE RSB, THERE IS A CORRESPONDING ENTRY IN THE ASSOCIATIVE MEMORY OF THE RSB. RSB AND THE PROCESS STATE IS SET. THE REMAINING ENTRIES IN THE RSB AT PROCESS CALL TIME, THE FIRST ENTRY IS LOADED INTO THE ARE MARKED EMPTY. UPON EACH MEMORY ACCESS, THE SEGMENT NUMBER OF THE TERMINAL REFERENCE IS CHECKED AGAINST THE ASSOCIATIVE MEMORY TO SEE WHETHER THE RESOURCE STACK ENTRY, WHICH IS SPECIFIED BY THE SEGMENT NUMBER, IS CAPTURED IN THE LOCAL BUFFER. IF IT IS, THE PROPER LOCATION WITHIN THE LOCAL BUFFER IS ACCESSED. IF IT IS NOT, THE RESOURCE STACK ENTRY IS FETCHED FROM LEVEL-1 AND PLACED INTO THE FIRST EMPTY LOCATION IN THE RSB. THE PROPER ENTRY IS THEN PLACED IN THE ASSOCIATIVE MEMORY OF THE RSB. IF ALL LOCATIONS IN THE RSB ARE FULL, THEN THE OLDEST ENTRY (EXCLUDING PROCESS ENVIRONMENT INFORMATION) IN THE RSB IS OVERWRITTEN WITH THE RESOURCE STACK ENTRY AND THE PROPER ASSOCIATIVE MEMORY LOCATION IS UPDATED. SEGMENT NUMBER REFERENCES TO SEGMENT NUMBERS 0 THRU 3 ARE ILLEGAL AND WILL RESULT IN A FAULT.

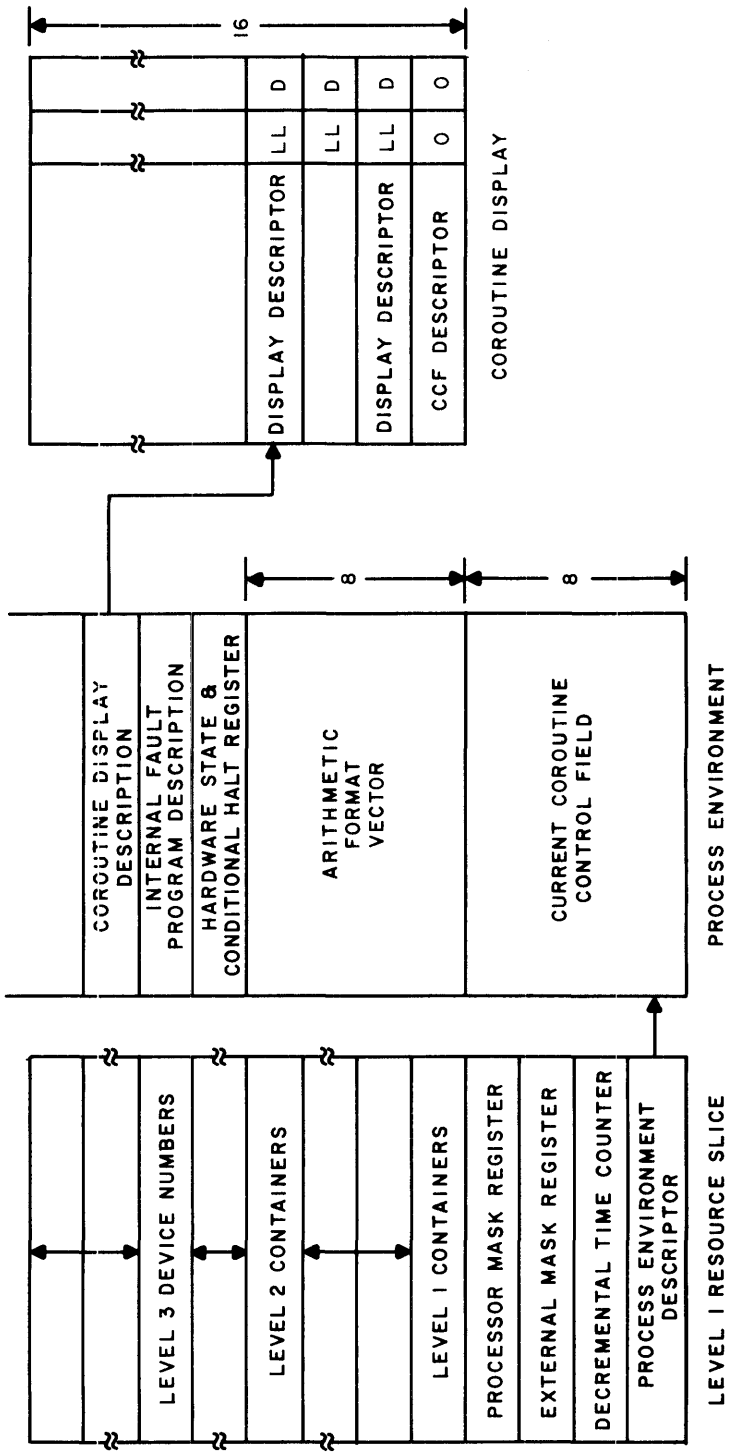
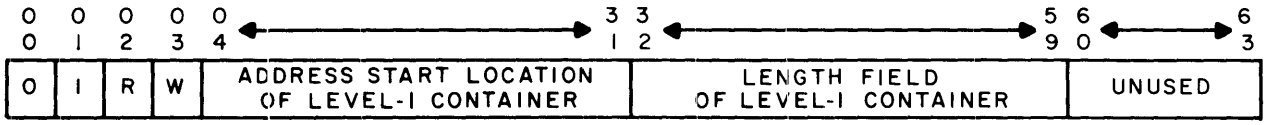
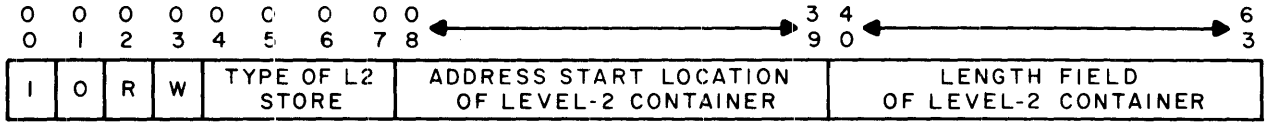


FIGURE 13. LEVEL-1 RESOURCE STACK MAP

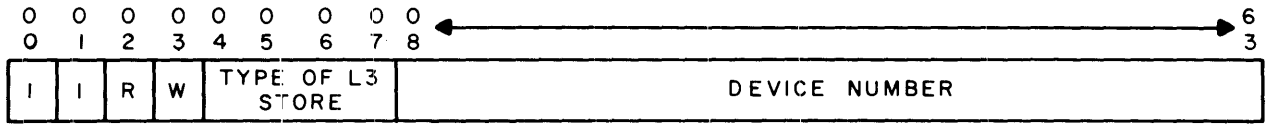
LEVEL-I STORAGE CONTAINER



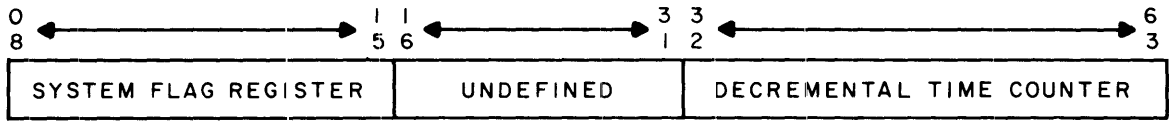
LEVEL-2 STORAGE CONTAINER



LEVEL-3 STORAGE CONTAINER



DECREMENTAL TIME COUNTER & SYSTEM FLAG REGISTER



PROCESS ENVIRONMENT DESCRIPTION

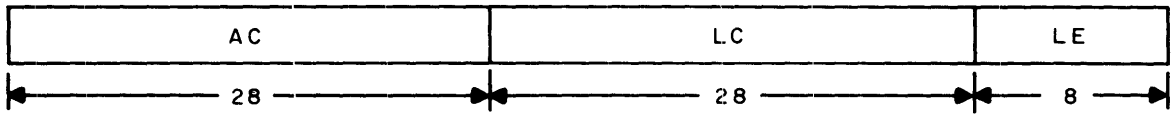


FIGURE 14. RESOURCE STACK ENTRIES, REPRESENTATION

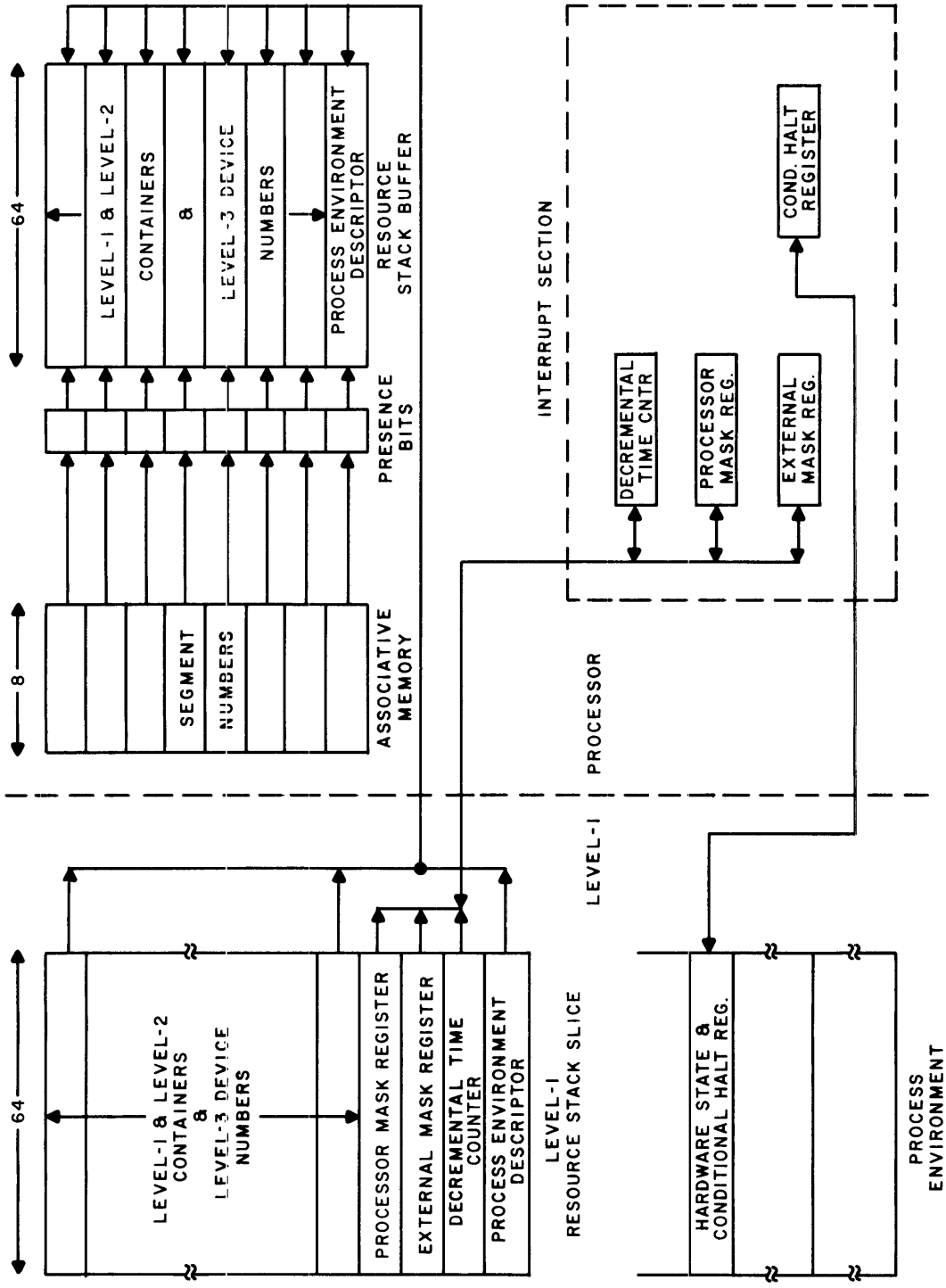


FIGURE 15. LOADING OF RESOURCE STACK SLICE INTO PROCESSOR

3.3.3.2 PROCEDURE CONTROL STRUCTURE

THE PROCEDURE CONTROL STRUCTURE SHALL PROVIDE THE MECHANISMS FOR ADDRESS CALCULATIONS THAT ARE REQUIRED BY HIGHER-LEVEL LANGUAGES. METHODS OF PROCESSING PARAMETERS TO PROCEDURES AND FUNCTIONS, SPACE ALLOCATIONS FOR LOCAL VARIABLES USED IN PROCEDURES, AND FUNCTIONS AND BLOCKS WILL BE AVAILABLE USING THE PROCEDURE CONTROL STRUCTURE. MEANS OF ADDRESSING THESE PARAMETERS AND LOCAL VARIABLES WILL BE MECHANIZED.

THE PROCEDURE CONTROL STRUCTURE (SEE FIGURE 16) SHALL CONSIST OF THREE INTERRELATED STACKS: NAME STACK, DISPLAY STACK AND VALUE STACK. THE INTERRELATION OF THESE STACKS IS EVIDENT AT PROCEDURE CALL AND RETURN WHEN ADDRESSING ENVIRONMENT OF THE INVOKED PROCEDURE MUST BE ESTABLISHED.

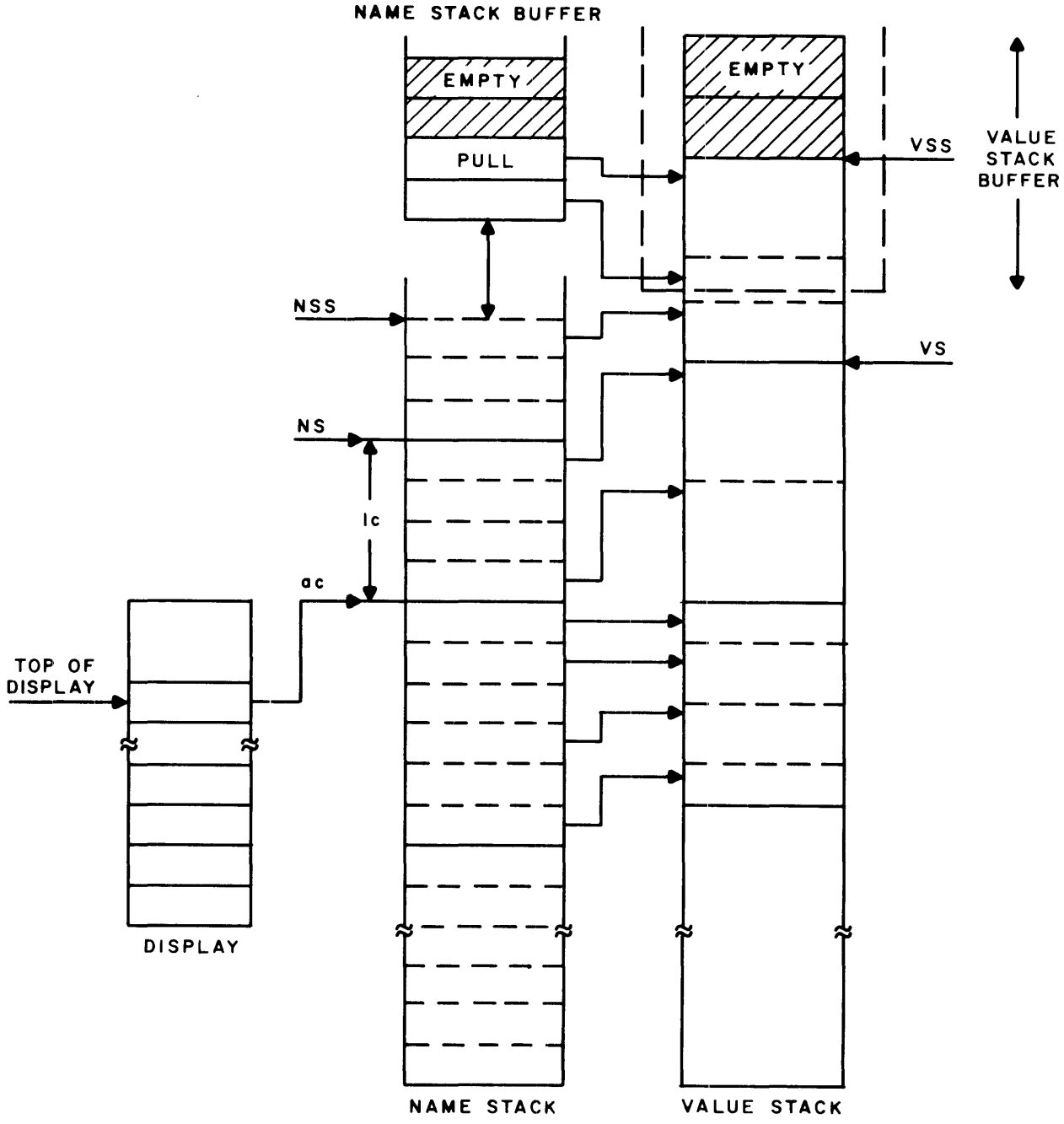


FIGURE 16. PROCEDURE CONTROL STRUCTURE

THE NAME STACK CONTAINS DESCRIPTIONS OF PARAMETERS AND LOCALS REQUIRED AT VARIOUS PROCEDURE, FUNCTION AND BLOCK LEVELS. SLICES ARE BUILT IN THE NAME STACK SO THAT PARAMETERS AND LOCALS MAY BE ADDRESSED BY NAME. EACH SLICE CONTAINS DESCRIPTIONS OF PARAMETERS FOR A GIVEN PRECEDING FUNCTION OR DESCRIPTIONS OF LOCALS FOR A GIVEN BLOCK. EACH SLICE IS DEFINED AS A LEXIC LEVEL. A DESCRIPTION OF EACH SLICE IS CONTAINED IN THE DISPLAY. A TYPICAL NAME CONSISTS OF A LEXIC LEVEL AND DISPLACEMENT; THAT IS, AN INDEX INTO THE DISPLAY WILL LOCATE THE PROPER NAME STACK SLICE AND AN INDEX INTO THE NAME STACK SLICE WILL LOCATE THE PROPER DESCRIPTION IN THE NAME STACK. SLICES CAN BE CREATED AND DESTROYED BY PROCEDURE OPERATORS OR BY PROCEDURE CALL AND RETURN. ENTRIES IN THE NAME STACK AREA BETWEEN THE TOP OF THE STACK AND THE TOPMOST SLICE ARE USED FOR EXPRESSION EVALUATION. THESE ENTRIES ARE ONLY ADDRESSABLE ON A LAST-IN-FIRST-OUT BASES. THE TOP FOUR ENTRIES IN THE EXPRESSION EVALUATION AREA MAY BE BUFFERED IN A LOCAL MEMORY FOR FAST ACCESS TO LEVEL-1 STORAGE. THE LOCAL MEMORY IS 4 WORDS BY 128 BITS. THE AMOUNT BUFFERED IS DYNAMICALLY CONTROLLED ON A USAGE BASIS. THE SIZE OF THIS MEMORY RESTRICTS THE WIDTH OF THE NAME STACK TO 128 BITS.

THE DISPLAY CONTAINS DESCRIPTIONS OF NAME STACK SLICES. THESE DESCRIPTIONS ARE ENTERED INTO THE DISPLAY BY PROCEDURE CALL OR BY THE SLICE OPERATOR. THESE DESCRIPTIONS ARE REMOVED FROM THE DISPLAY BY PROCEDURE RETURN OR BY THE UNSLICE OPERATOR. EACH ENTRY IN THE DISPLAY THAT IS ACCESSED WILL BE CHECKED TO SEE IF IT IS CAPTURED IN THE LOCAL ASSOCIATIVE MEMORY OF THE DISPLAY. IF IT IS NOT CAPTURED, THEN THIS ENTRY IS FETCHED FROM LEVEL-1 AND REPLACES THE OLDEST ENTRY IN THE LOCAL MEMORY. THE LOCAL MEMORY IS 8 WORDS BY 64 BITS. A DISPLAY ENTRY IS ILLUSTRATED IN FIGURE 17.

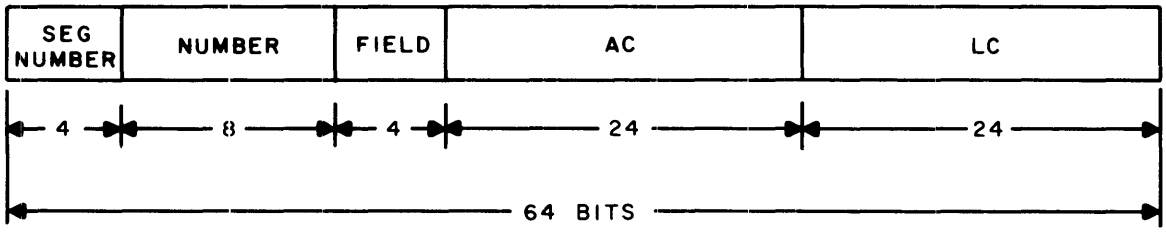


FIGURE 17. DISPLAY ENTRY REPRESENTATION

SINCE THE NAME STACK WIDTH IS FIXED IN SIZE, THE VECTOR STRUCTURE OF THE DISPLAY IS IMPLIED. THE S-EXPRESSION PARAMETERS OF TYPE "FIELD" DEFINES THE CONTAINER OF THE SLICE.

THE SLICES IN THE NAME STACK ARE SEPARATED BY DISPLAY CONTROL WORDS (DCW). THE DCW IS ILLUSTRATED IN FIGURE 18. THE DCW CONSISTS OF A 28-BIT LENGTH FIELD WHICH CONTAINS THE LENGTH OF THE SLICE BELOW THE DCW, AN 8-BIT LEXIC LEVEL FIELD WHICH CONTAINS THE LEXIC LEVEL OF THE SLICE BELOW THE DCW, AND A 64-BIT DISPLAY FIELD WHICH CONTAINS A DESCRIPTION OF THE SLICE WHICH INVOKED THE SLICE BELOW THE DCW. THE DCWS ARE CREATED DURING THE SLICE OPERATOR AND THE EXECUTE OPERATOR. DURING SLICE, THE CONTENTS OF THE TOP OF THE DISPLAY ARE PLACED IN THE DISPLAY FIELD OF THE DCW. THE SLICE DESCRIPTOON IS CREATED AND ENTERED INTO THE DISPLAY. THE LEXIC LEVEL OF THE SLICE IS PLACED IN THE LEXIC LEVEL OF THE DCW. THE LENGTH OF THE SLICE IS PLACED IN THE LENGTH FIELD OF THE DCW.

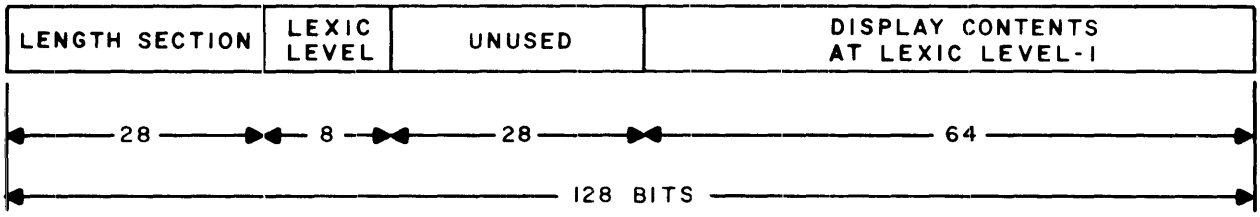


FIGURE 18. DISPLAY CONTROL WORD REPRESENTATION

DURING EXECUTE, THE LEXIC LINK FIELD OF THE PROGRAM/DESCRIPTOR CONTROL REGISTER IS USED AS AN INDEX INTO THE NAME STACK (BASE RELATIVE) TO FIND THE DCW OF THE SLICE IN WHICH THE PROGRAM WAS DECLARED. THIS DCW IS BROUGHT TO THE TOP OF THE NAME STACK. ITS LEXIC LEVEL FIELD IS INCREASED BY ONE TO GIVE THE LEXIC LEVEL OF THE ENTERED PROCEDURE. THIS DISPLAY FIELD IS SAVED, AND THE LENGTH AND DISPLAY FIELDS ARE UPDATED AS IN THE SLICE OPERATOR. THE DISPLAY IS UPDATED BY COMPARING THE SAVED DISPLAY FIELD WITH THE LL-1 ENTRY IN THE DISPLAY. IF THEY ARE THE SAME THE DISPLAY UPDATE IS FINISHED. IF NOT, THE CONTENTS OF THE SAVED DISPLAY FIELD IS USED TO CALCULATE THE LOCATION OF THE DCW OF THE INVOKING SLICE. THE COMPARISON IS REPEATED WITH EACH DCW UNTIL THE DISPLAY UPDATE IS FINISHED.

DCW(S) ARE REMOVED UPON UNSLICE AND PROCEDURE RETURN . UNSLICE DOES NOT REQUIRE A DISPLAY UPDATE BUT PROCEDURE RETURN DOES. THE DCW OF THE UNCOVERED SLICE CONTAINS THE LEXIC LEVEL AT WHICH THE UPDATE STARTS. THE UPDATE PROCEDURE IS AS STATED ABOVE.

THE VALUE STACK STORES ARITHMETIC OPERANDS THAT ARE ABOUT TO BE USED OR THAT ARE THE RESULT OF A COMPUTATION. EACH ENTRY IN THE VALUE STACK IS REFERENCED BY A DATA DESCRIPTOR IN THE NAME STACK. VALUES MAY BE EXPLICITLY NAMED. THE NAME REFERENCES A DESCRIPTOR IN THE NAME STACK. IN TURN, THIS DESCRIPTOR DEFINES THE DESIRED ENTRY IN THE VALUE STACK. ARITHMETIC OPERATORS WHICH REQUIRE VALUES, CAUSE THE TOP OF THE NAME STACK TO BE EXAMINED TO SEE IF IT REFERENCES A VALUE. PROGRAM OPERATORS, WHICH AFFECT THE CONTENTS OF THE NAME STACK, WILL ALSO AFFECT THE CONTENTS OF THE VALUE STACK IF THE NAME STACK ENTRIES REFERENCE THE VALUE STACK.

THE VALUE STACK HAS SLICES THAT ARE CREATED AND DESTROYED CONCURRENTLY WITH NAME STACK SLICES. THESE SLICES CONTAIN OPERANDS, CONSTANTS AND PARTIAL RESULTS OF PROGRAM EXECUTION AT VARIOUS LEXIC-LEVELS.

ANY OR ALL OF THE TOP FOUR ENTRIES IN THE VALUE STACK MAY BE CAPTURED IN THE VALUE STACK BUFFERS OF THE ARITHMETIC UNIT. THE BUFFERS ARE A LOCAL MEMORY OF 4 WORDS BY 256 BITS. THE WORD SIZE OF 256 BITS LIMITS THE SIZE OF A SINGLE OPERAND FOR AN ARITHMETIC OPERATION TO 256 BITS. THE VALUE STACK BUFFERS LINK AUTOMATICALLY TO THE VALUE STACK IN LEVEL-1 STORAGE.

3.3.3.3 COROUTINE CONTROL STRUCTURE

THE COROUTINE CONTROL STRUCTURE SHALL CONTROL ALL ROUTINES THAT CAN EXIST CONCURRENTLY BUT MUST BE RUN CONSECUTIVELY. EACH COROUTINE IS DEFINED BY A PROCEDURE CONTROL STRUCTURE AND A PROGRAM CONTROL STRUCTURE WHICH ARE NAMED IN THE NAME STACK OF THE PARENT STRUCTURE DESCRIPTORS ARE IN CONSECUTIVE LOCATIONS IN THE NAME STACKS. STRUCTURE. THE PROCEDURE CONTROL STRUCTURE AND PROGRAM CONTROL THIS GROUP OF CONSECUTIVE LOCATIONS IS CALLED THE COROUTINE CONTROL FIELD (CCF). THE CCF FOR THE ROUTINE CURRENTLY BEING EXECUTED IS CONTAINED IN THE DESCRIPTOR BUFFER (A LOCAL MEMORY).

THE COROUTINE CONTROL STRUCTURE (SEE FIGURE 19) SHALL HAVE A COROUTINE DISPLAY DESCRIPTION WHICH SHALL RESIDE IN A FIXED LOCATION OF THE PROCESS INVIRONMENT VECTOR. THE COROUTINE DISPLAY DESCRIPTION DEFINES THE COROUTINE DISPLAY (CD), WHICH IS A STACK VECTOR. THE TOP ENTRY IN THE CD DEFINES AN ACTIVE COROUTINE. THE TOP ENTRY SHALL CONTAIN A DESCRIPTION OF THE PARENTS DISPLAY AND A NAME (I.E. LEXIC LEVEL AND DISPLACEMENT) WHICH, WHEN APPLIED TO THE PARENTS DISPLAY, FINDS THE CCF OF THE ACTIVE COROUTINE. THE REMAINING ENTRIES IN THE CD DEFINE THE ANCESTRY (I.E. SON PARENT LINKS) OF ACTIVE COROUTINES.

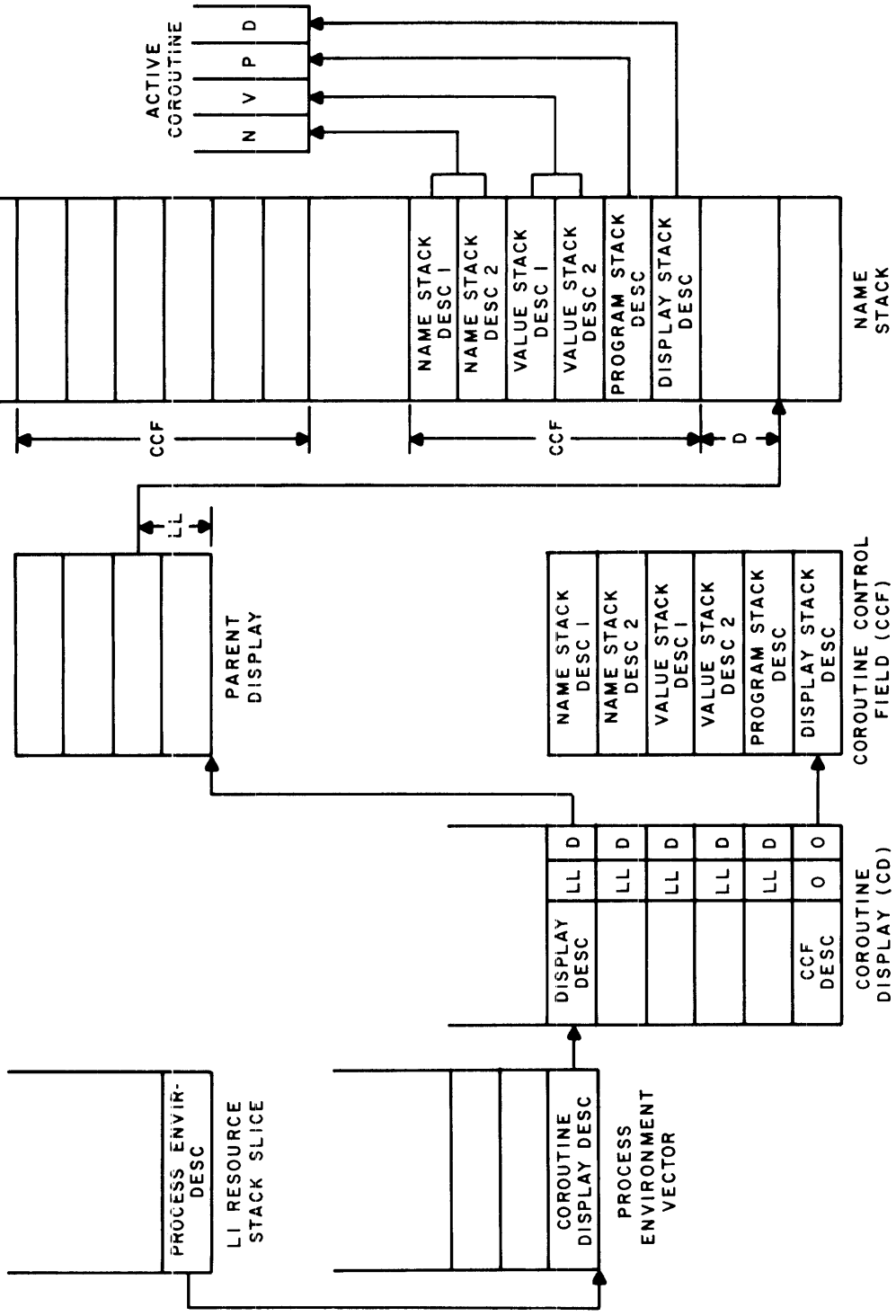


FIGURE 19. COROUTINE CONTROL STRUCTURE

A COROUTINE CAN BE INVOKED BY THE COROUTINE CALL OPERATOR. THE COROUTINE CALL OPERATOR HAS THE NAME OF THE CCF OF THE COROUTINE THAT IS TO BE INVOKED. THIS NAME REPLACES THE EXISTING NAME IN THE TOP ENTRY OF THE CD. THE HARDWARE RESTORES THE CCF OF THE EXISTING COROUTINE IN THE NAME STACK OF THE PARENT. THE NEW CCF IS NOW CAPTURED IN THE DESCRIPTOR BUFFER STRUCTURE.

THE COROUTINE ACTIVATE OPERATOR ESTABLISHES A NEW FAMILY OF COROUTINES BY PLACING A NEW ENTRY ON TOP OF THE CD. THE COROUTINE END OPERATOR REMOVES THE CURRENT FAMILY OF COROUTINES BY REMOVING THE TOP ENTRY IN THE CD.

THE COROUTINE CONTROL FIELD BUFFER (SEE FIGURE 20) CONSIST OF A LOCAL MEMORY OF 12 WORDS BY 128 BITS AND AN ASSOCIATIVE MEMORY OF 12 WORDS BY 40 BITS.

THE FUNCTION OF THE COROUTINE CONTROL FIELD BUFFER IS TO CONTAIN THE CCF OF THE CURRENT COROUTINE, AND THE DESCRIPTIONS OF THE RESOURCE STACK AND THE COROUTINE DISPLAY. THE DESCRIPTIONS ARE STRUCTURE INFORMATION THAT IS REFERENCED BY THE PROGRAM OPERATOR. THAT IS, THE STRUCTURES THAT ARE USED BY THE PROGRAM OPERATORS.

THE ASSOCIATIVE MEMORY CONTAINS THE LEVEL-1 ADDRESS OF EACH DESCRIPTOR CONTAINED IN THE BUFFER IN ORDER THAT EACH UPDATEED DESCRIPTOR CAN BE RESTORED QUICKLY TO LEVEL-1 STORAGE.

THE CCF IS STORED IN THE DESIGNATED ENVIRONMENT AT PROCESS RETURN AND RESTORED FROM THIS AREA AT PROCESS CALL.

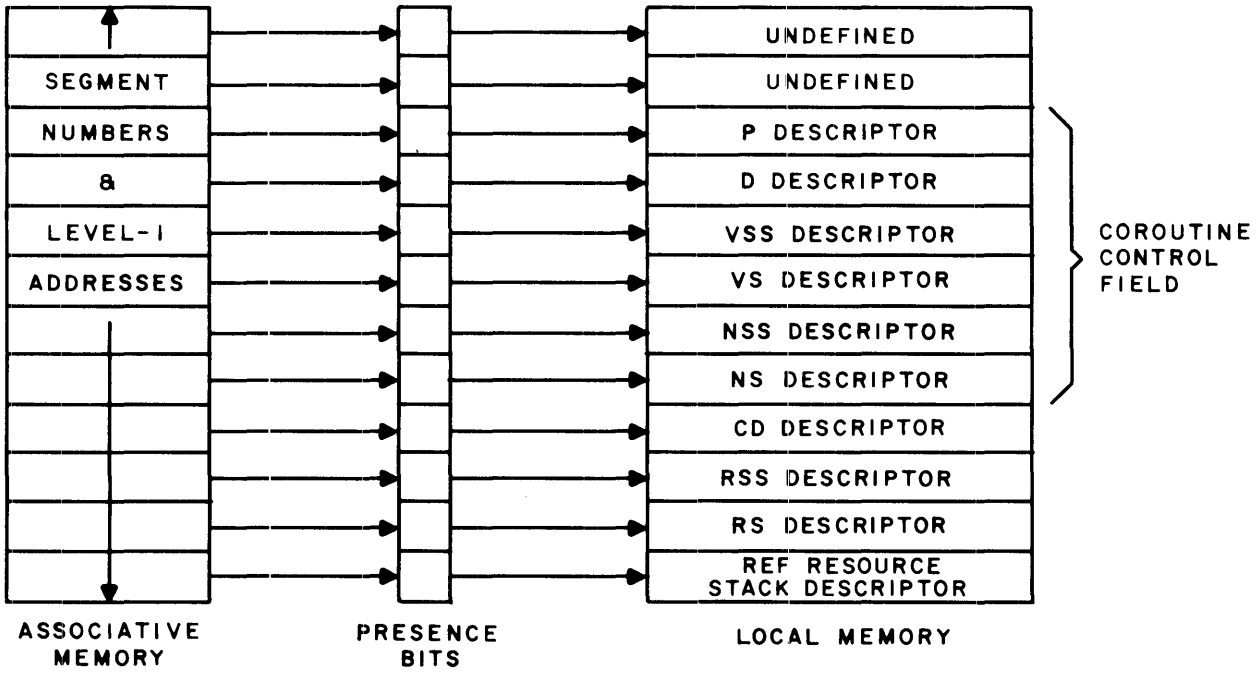


FIGURE 20. COROUTINE CONTROL FIELD BUFFER

3.3.3.4 PROGRAM/DESCRIPTOR CONTROL STRUCTURE

THE PROGRAM/DESCRIPTOR CONTROL STRUCTURE (SEE FIGURE 21 CONSISTS OF A ~~106~~ 96 BIT PROGRAM/DESCRIPTOR CONTROL REGISTER (PDCR) AND AN 8 WORD BY ~~106~~ 96 BIT PROGRAM/DESCRIPTOR CONTROL STACK (PDCS) WITH A LINK TO LEVEL-1. THE STRUCTURE RETAINS BOTH PROGRAM EXECUTION AND DESCRIPTOR EVALUATION HISTORY. ENTRY INTO A SUBROUTINE, PROCEDURE, FUNCTION, OR LOOP CAUSES THE PROGRAM EXECUTION INFORMATION IN THE PDCR TO BE PUSHED INTO PDCS. THE ENTRY IS THEN RECORDED IN PDCR. A PROGRAM BRANCH REPLACES THE PRESENT INFORMATION IN THE PDCR WITH A DESCRIPTION OF THE BRANCH. DURING DESCRIPTOR EVALUATION, A STRUCTURE EXPRESSION OF TYPE "CALL" CAUSES THE PDCR TO BE PUSHED INTO PDCS. THE CALLED DESCRIPTION IS PLACED IN PDCR. SINCE DESCRIPTOR EVALUATION NEVER CHANGES PROGRAM HISTORY, THE DESCRIPTOR EVALUATION HISTORY WILL ALWAYS BE ON TOP OF THE PROGRAM EXECUTION HISTORY IN THE PDCS.

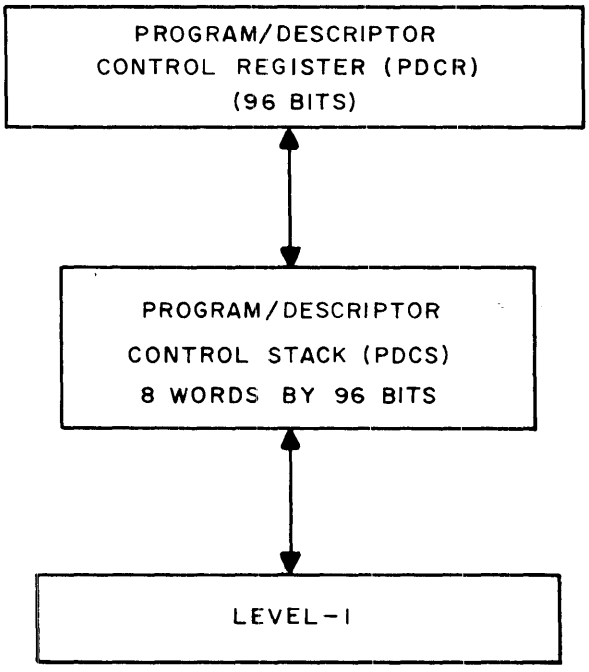


FIGURE 21. PROGRAM/DESCRIPTOR CONTROL STRUCTURE

THE PDCR REPRESENTATION (SEE FIGURE 22) IS AS FOLLOWS. THE FIRST BIT OF THE PDCR DEFINES THE OBJECT THAT IS REFERENCED BY THE PDCR AS EITHER A PROGRAM OR DESCRIPTOR. IF THE M + E OBJECT IS A DESCRIPTOR, THE NEXT THREE BITS CONTAIN THE AFL FIELD OF THE DESCRIPTOR. IF THE OBJECT IS PROGRAM THESE THREE BITS ARE UNUSED. THE NEXT 22 BITS ARE THE INTERPRETER ATTRIBUTES OF THE OBJECT. FOLLOWING THE INTERPRETER ATTRIBUTES IS AN 8-BIT SEGMENT NUMBER AND A 48-BIT DESCRIPTION OF THE CONTAINER OF THE OBJECT. THE LAST 24-BITS OF THE PDCR IDENTIFY THE ADDRESS FIELD WHICH POINTS TO THE NEXT PROGRAM OPERATOR TO BE EXECUTED OR DESCRIPTOR ELEMENT TO BE EVALUATED.

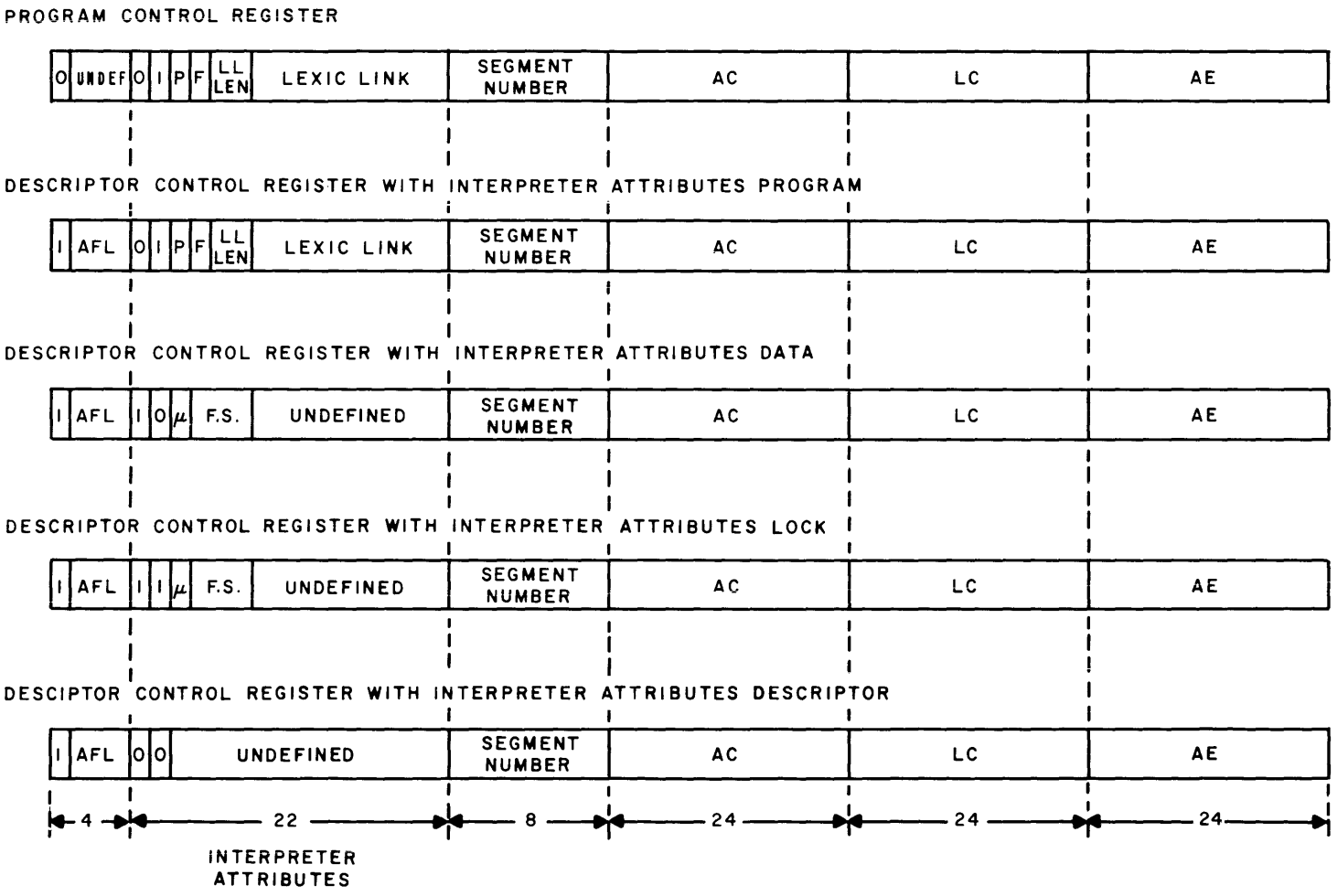


FIGURE 22. PROGRAM/DESCRIPTOR CONTROL REGISTER REPRESENTATION

3.3.3.5 DESCRIPTOR BUFFERS

THE DESCRIPTOR BUFFER CONSISTS OF A LOCAL MEMORY OF 8 WORDS BY 128 BITS AND AN ASSOCIATIVE MEMORY OF 8 WORDS BY 40 BITS. THE FUNCTION OF THE DESCRIPTOR BUFFER IS TO PROVIDE ASSOCIATIVE STORAGE FOR DESCRIPTORS THAT ARE BEING CURRENTLY EVALUATED AND THAT ARE OFTEN ACCESSED. THE ASSOCIATIVE MEMORY CONTAINS THE SEGMENT NUMBER AND CONTAINER LOCATION, AND LENGTH FIELDS THAT RESULT FROM THE EVALUATION OF A DESCRIPTORS NAME. WHEN THE NAME OF A DESCRIPTOR HAS BEEN REDUCED TO A TERMINAL REFERENCE, THIS REFERENCE IS CHECKED AGAINST THE ASSOCIATIVE MEMORY TO SEE IF THE DESCRIPTOR IS CAPTURED IN THE DESCRIPTOR BUFFER. IF SO, THEN THE CONTENTS OF THE PROPER ENTRY IN THE DESCRIPTOR BUFFER ARE USED. IF THE DESCRIPTOR IS NOT CONTAINED IN THE DESCRIPTOR BUFFER, THEN THE DESCRIPTOR IS BROUGHT INTO THE FIRST EMPTY LOCATION OF THE DESCRIPTOR BUFFER. IF ALL ENTRIES ARE FULL, THE OLDEST ENTRY IN THE DESCRIPTOR BUFFER IS OVERWRITTEN. IF A DESCRIPTOR IS TOO LARGE TO TO BE CONTAINED IN ONE ENTRY OF THE DESCRIPTOR BUFFER, AS MANY ENTRIES AS NEEDED ARE USED TO HOLD THE DESCRIPTOR. EACH ADDRESS IN THE ASSOCIATIVE MEMORY CONTAINS THE ADDRESS OF ITS CORRESPONDING ENTRY IN THE BUFFER. EVALUATED DESCRIPTORS THAT HAVE BEEN UPDATED ARE RESTORED TO LEVEL-1 STORAGE.

3.3.4 INTERRUPT SECTION

THE INTERPRETER INTERRUPT SECTION (SEE FIGURE 23) SHALL RECEIVE EXTERNALLY GENERATED INTERRUPTS AND EXTERNALLY OR INTERNALLY GENERATED FAULTS AND SHALL EXAMINE THE FAULTS IN ACCORDANCE WITH A PROGRAMMABLE SET OF MASKS. THE PROGRAM SECTION SHALL BE NOTIFIED OF INTERRUPTS AND UMMASKED FAULTS IN ORDER TO ACCOMPLISH CHANGES IN THE PROGRAM BEING EXECUTED. THEN THE APPROPRIATE INTERRUPT OR FAULT HANDLING ROUTINE MAY BE CALLED. THE INTERPRETER INTERRUPT SECTION SHALL ALSO INFORM THE PROGRAM SECTION WHEN A CONDITIONAL HALT SITUATION IS REACHED.

SYSTEM FLAG INTERRUPTS OCCUR WHEN SOME PROCESSOR OR I/O MODULE IN THE OPERATING SYSTEM SETS A BIT IN THE PROCESSORS SYSTEM FLAG REGISTER (SFR). THE PRESENCE OF A FLAG BIT INDICATES TO THE PROCESSOR THAT ANOTHER PROCESSOR OR I/O MODULE REQUIRES ASSISTANCE. THE BITS OF THE SYSTEM FLAG REGISTER ARE INDIVIDUALLY RESETTABLE. THIS REGISTER IS NOT MASKABLE.

PROCESS FAULTS, WHICH ARE RECORDED IN THE PROCESS FAULT REGISTER (PFR), RESULT FROM CONDITIONS DETECTED BY THE PROCESSOR DURING THE CURRENT EXECUTION. PROCESS FAULTS INCLUDE: ARITHMETIC UNIT DETECTED FAULTS SUCH AS OVERFLOW, DIVIDE BY ZERO, ILLEGAL FORMAT, ETC.; INTERPRETER DETECTED FAULTS SUCH AS ADDRESS OUT OF BOUNDS, ILLEGAL OPERATOR, DECREMENTAL TIMER RUN OUT, ETC.; HARDWARE

MALFUNCTION FAULTS SUCH AS PARITY ERROR, IMPROPER RESIDUE, NO ACCESS TO MEMORY, ETC AND MEMORY MODULE FAULTS. THE BITS OF THE PROCESS FAULT REGISTER ARE INDIVIDUALLY RESETTABLE. FAULTS RECORDED IN THE PROCESS FAULT REGISTER ARE MASKABLE. FOR THIS PURPOSE TWO MASK REGISTERS ARE PROVIDED. THE EXTERNAL MASK REGISTER (EMR) IS USED TO DETERMINE WHETHER RESPONSE TO A FAULT IS THE RESPONSIBILITY OF THE CURRENT PROCESS OR OF ITS PARENT PROCESS. IF THE RESPONSIBILITY FALLS ON THE CURRENT PROCESS, THE PROCESS MASK REGISTER (PMR) IS USED TO DETERMINE WHETHER THE PROCESS WILL ACT ON THE FAULT OR IGNORE IT. FLAG AND MASKED FAULT INFORMATION SIGNALS ARE SENT TO THE ARITHMETIC UNIT FOR BIT ENCODING.

THE CONTENTS OF THE CONDITIONAL HALT REGISTER (CHR) ARE CONTINUALLY COMPARED WITH THE PROGRAM CONTROL REGISTER. THIS ALLOWS CONDITIONAL HALTING OF THE PROCESSOR AT ANY DESIRED VALUE OF THE PROGRAM CONTROL REGISTER.

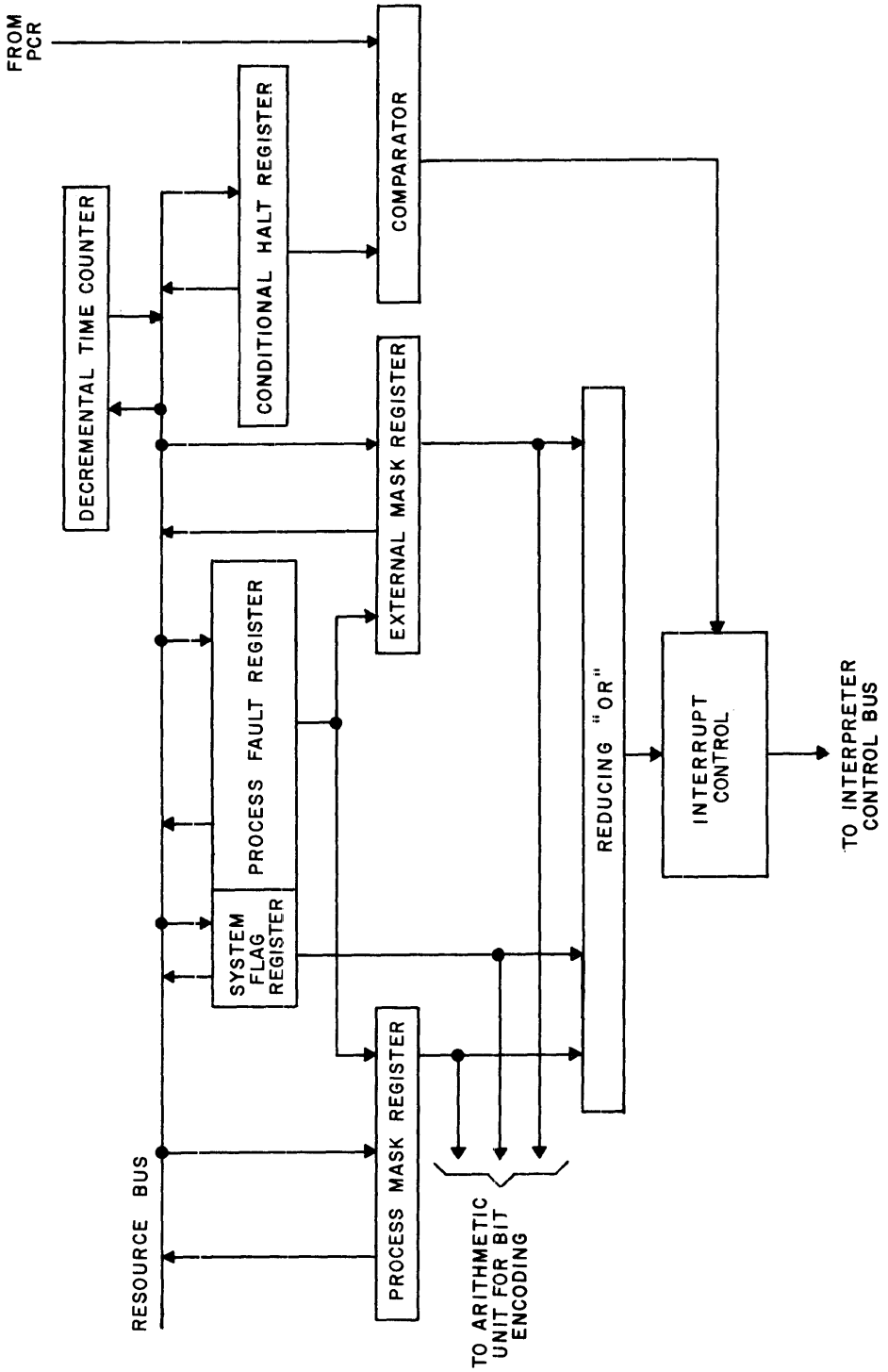


FIGURE 23. INTERPRETER INTERRUPT SECTION

3.4 ARITHMETIC UNIT (AU)

THE AU SHALL CONTAIN THE HARDWARE NEEDED TO EXECUTE ARITHMETIC, LOGICAL, AND CHARACTER OPERATIONS. IN ADDITION TO THE LOGIC THAT PERFORMS THE ACTUAL CALCULATIONS, THE AU SHALL BE PROVIDED WITH LOCAL OPERAND STORAGE, THE MEANS OF ASSEMBLING AND POSITIONING DATA WITHIN THE UNIT, A CONTROL FOR VARIABLE LENGTH AND FLOATING POINT OPERATIONS, ERROR DETECTION HARDWARE, AND THE CAPABILITY OF HANDLING VARIOUS FORMATS.

THE INDIVIDUAL SECTIONS OF THE AU (SEE FIGURE 24) WILL BE CONNECTED BY EITHER VALUE DATA PATHS OR LENGTH DATA PATHS. THE VALUE DATA PATHS SHALL ALLOW MOVEMENT OF OPERANDS WHILE THE LENGTH DATA PATHS SHALL ONLY BE CONCERNED WITH MOVEMENT OF LENGTH INFORMATION NEEDED FOR VARIABLE LENGTH AND FLOATING POINT OPERATIONS. A DESCRIPTION OF EACH SECTION OF THE AU IS PRESENTED UNDER THE FOLLOWING HEADINGS:

A. STORAGE SECTION

B. BARREL SECTION

C. ADDER SECTION

D. EXPONENT SECTION

E. LENGTH SECTION

F. FIELD SECTION

G. FORMAT SECTION

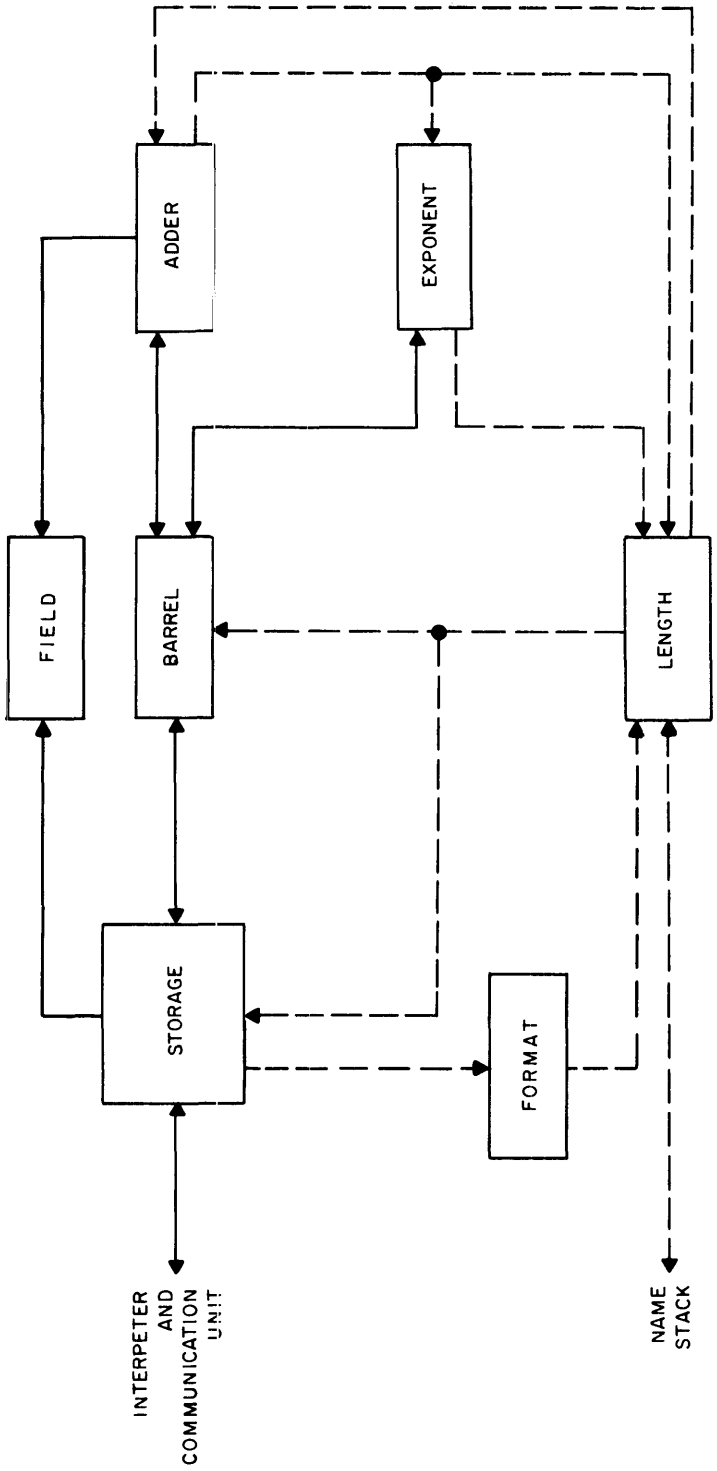


FIGURE 24. ARITHMETIC UNIT DATA PATHS

3.4.1 STORAGE SECTION

THE STORAGE SECTION SHALL PROVIDE THE LOCAL BUFFERING OF OPERANDS IN THE SAME FORM AS THEY ARE RECEIVED FROM OR SENT TO THE MAIN MEMORY AND THE INTERPRETER. THE STORAGE SECTION SHALL HOLD THE TOP ONE, TWO, THREE, OR FOUR VALUE STACK OPERANDS

THE STORAGE SECTION SHALL CONSIST OF TWO FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 25, AND OPERATED AS DESCRIBED BELOW.

A. THE BUFFER STORAGE AREA SHALL BE A 64-BIT WIDE MEMORY WHICH WILL BE ALLOCATED AS FOLLOWS:

1. FOUR FOUR-WORD ZONES, EACH OF WHICH CONTAINS 256 BITS, CAN BE ALLOCATED TO BUFFER THE FOUR TOP OPERANDS OF THE VALUE STACK (A, B, C, AND D). AS AN OPERAND CHANGES ITS POSITION IN THE VALUE STACK, THE ZONE OF THIS OPERAND IS ALLOCATED TO ITS NEW POSITION TO AVOID MOVING THE OPERAND. FOR INSTANCE AT THE END OF A DYADIC AU OPERATION, THE ZONE ALLOCATED AS "C" BECOMES "B".

2. ONE FOUR-WORD ZONE IS PERMANENTLY ASSIGNED THE LOCATION WHERE THE RESULT IS ASSEMBLED.

3. ONE FOUR-WORD ZONE IS PERMANENTLY ASSIGNED TO BE THE WORK AREA FOR SUCH OPERATIONS AS MULTI-PRECISION MULTIPLY AND DIVIDE.

4. ONE WORD IS ASSIGNED TO INTERFACE WITH THE OPERAND STORAGE WHEN MULTIPLE BARREL OPERATIONS ARE REQUIRED.

5. ONE WORD IS ASSIGNED TO INTERFACE WITH THE ADDER WHEN MULTIPLE BARREL OPERATIONS ARE REQUIRED.

6. TWO WORDS ARE ASSIGNED TO HOLD THREE INTERMEDIATE EXPONENTS (MAXIMUM LENGTH OF EACH IS 32 BITS).

B. THE B-REGISTER SHALL BE A 64-BIT WIDE FLIP-FLOP REGISTER WHICH PRIMARILY FUNCTIONS AS AN INTERFACE BETWEEN THE BUFFER STORAGE AREA AND ITS SOURCES AND DESTINATIONS: THE BARREL SECTION, THE INTERPRETER, AND THE COMMUNICATIONS UNIT. IN ADDITION, THE B-REGISTER IS USED TO PROVIDE INPUTS DIRECTLY TO THE ADDER SECTION FOR MULTIPLY AND CONVERSION, AND ASSEMBLE OUTPUTS FROM THE ADDER SECTION FOR MULTIPLY, DIVIDE AND CONVERSION.

THE STORAGE SECTION SHALL ALSO INCLUDE SIX TWO-BIT RESIDUE REGISTERS THAT WILL BE USED TO STORE THE RESIDUE VALUES FOR THE FOLLOWING ELEMENTS OF THE STORAGE SECTION: THE B-REGISTER, THE STORAGE INTERFACE WORD, THE ADDER INTERFACE WORD AND THE THREE INTERMEDIATE EXPONENTS.

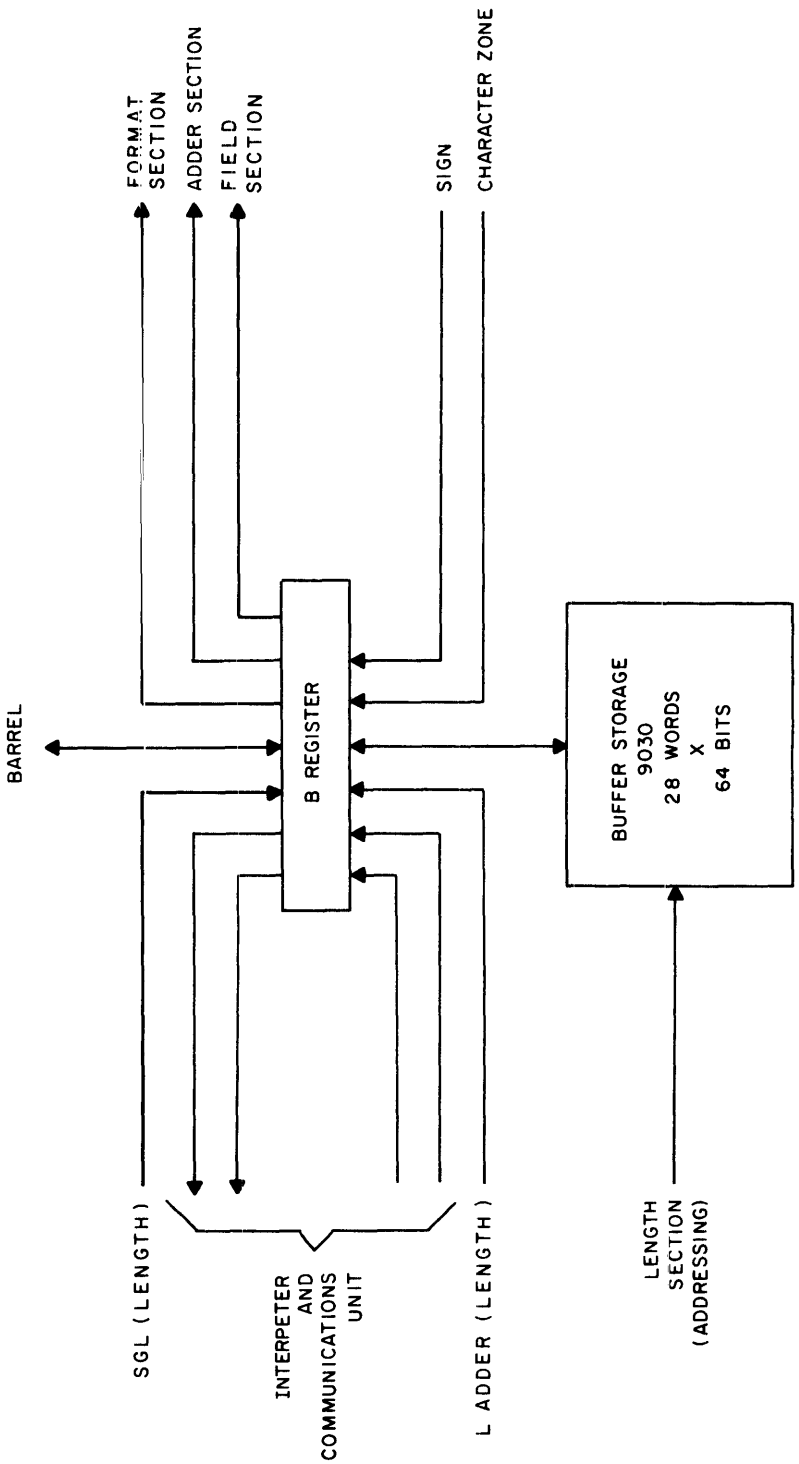


FIGURE 25. STORAGE SECTION, BLOCK DIAGRAM

3.4.2 BARREL SECTION

THE BARREL SECTION SHALL LINK THE STORAGE SECTION WITH OTHER SECTIONS OF THE AU. IT SHALL EXECUTE THE POSITIONING, THE SEGMENTING, AND THE ASSEMBLING OF DATA AS REQUIRED FOR EACH PARTICULAR OPERATION.

THE BARREL SECTION SHALL CONSIST OF SEVEN FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 26, AND OPERATED AS DESCRIBED BELOW:

A. THE MAIN BARREL SHALL BE A 64-BIT END-AROUND SHIFTING MECHANISM BY WHICH NEARLY ALL VALUE DATA TRANSFERS BETWEEN THE ADDER AND STORAGE SECTION ARE ACCOMPLISHED. IT WILL EXTRACT DATA BY COMBINING THE PROPER SHIFT WITH MASKING ON THE OUTPUT OF THE BARREL AND WILL INSERT DATA BY COMBINING THE PROPER SHIFT WITH MASKING ON THE INPUT OF THE BARREL. THE MAIN BARREL SHALL ALSO BE USED TO NORMALIZE MANTISSAS AND TO SHIFT MANTISSAS RIGHT TO INSERT LEADING ZEROS.

B. THE EXPONENT BARREL SHALL BE A 16-BIT END-AROUND SHIFTING MECHANISM BY WHICH ALL VALUE DATA TRANSFERS BETWEEN THE EXPONENT AND STORAGE SECTIONS ARE ACCOMPLISHED. IT USES MASKING IN THE SAME MANNER AS THE MAIN BARREL.

C. THE SHIFT REGISTER SHALL BE SIX BITS WIDE AND WILL BE USED TO CONTAIN THE AMOUNT BETWEEN 0 AND 63 BY WHICH DATA WILL BE SHIFTED IN THE MAIN BARREL. THIS REGISTER SHALL ALSO DETERMINE THE SHIFT IN THE EXPONENT BARREL.

D. THE MAIN MASK REGISTER SHALL CONTAIN THE MASK IN A PARTIALLY DECODED FORM FOR THE MAIN BARREL. THIS DECODE RESULTS IN THREE REGISTERS.

1. THE BIT MASK REGISTER SHALL BE 16 BITS WIDE AND WILL BE USED TO CONTAIN A MASKING PATTERN OF ADJACENT ONES AND ZEROS WHICH WILL BE APPLIED TO A PORTION OF THE MAIN BARREL.

2. THE SELECT MASK REGISTER SHALL BE FOUR BITS WIDE AND WILL BE USED TO DETERMINE WHERE THE BIT MASK WILL BE APPLIED.

3. THE ALLOW MASK REGISTER SHALL BE FOUR BITS WIDE AND WILL BE USED TO DETERMINE WHICH PARTS OF THE MAIN BARREL WILL RECEIVE 16 ONES AS A MASK.

E. THE EXPONENT MASK REGISTER SHALL CONTAIN THE MASK AMOUNT FOR THE EXPONENT BARREL.

F. THE MM AND ML REGISTERS SHALL BE FIVE BITS AND SIX BITS WIDE, RESPECTIVELY, AND WILL BE USED TO SAVE THE SHIFT AND MASK AMOUNTS REQUIRED FOR PLACING THE RESULTS OF A FLOATING POINT OPERATION INTO THE STORAGE SECTION.

THE BARREL SECTION SHALL ALSO INCLUDE FOUR TWO-BIT RESIDUE REGISTERS THAT WILL BE USED TO STORE THE RESIDUE VALUES FOR THE FOLLOWING ELEMENTS OF THE BARREL SECTION: THE SHIFT REGISTER, THE MAIN MASK REGISTER, THE MM REGISTER AND THE ML REGISTER.

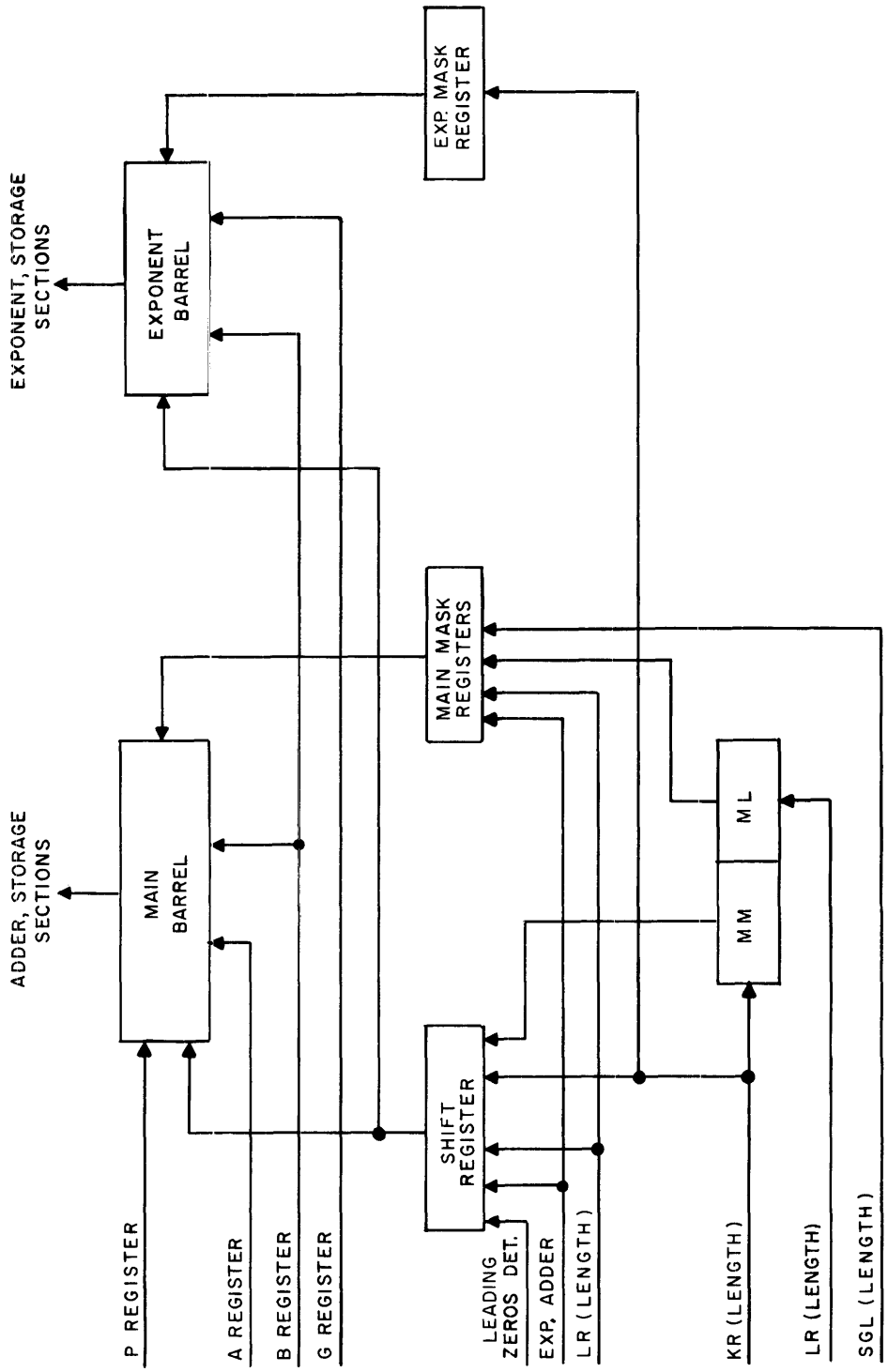


FIGURE 26. BARREL SECTION, BLOCK DIAGRAM

3.4.3 ADDER SECTION

THE ADDER SECTION SHALL PERFORM THE ARITHMETIC AND LOGICAL COMBINATIONS ON ALL VALUES EXCEPT FOR THE EXPONENT CALCULATIONS REQUIRED FOR FAST FLOATING POINT OPERATIONS. THE ADDER SECTION SHALL ALSO PROVIDE LENGTH DATA TO THE LENGTH AND EXPONENT SECTIONS.

THE ADDER SECTION SHALL CONSIST OF 11 FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 27, AND OPERATED AS DESCRIBED BELOW:

A. THE MAIN ADDER SHALL CONSIST OF A THREE-INPUT BINARY ADDER, WHICH WILL HAVE A DECIMAL CAPABILITY. THE ADDER SHALL BE 64 BITS WIDE PLUS AN EXTENSION WHICH WILL BE NECESSARY FOR MULTIPLY, DIVIDE, AND CONVERSION OPERATIONS.

B. THE LEADING ZEROS DETECTOR SHALL ENCODE THE NUMBER OF LEADING ZEROS OF AN OPERAND IN THE A-REGISTER.

C. THE BINARY AND DECIMAL MULTIPLY DECODERS SHALL BE SIX-BIT AND FOUR-BIT (ONE DECIMAL DIGIT) LOOKAHEAD DECODERS, RESPECTIVELY, WHICH SELECT THE NEXT MULTIPLES TO BE ADDED TO THE PARTIAL PRODUCT.

D. THE BINARY DIVIDE DECODER SHALL DERIVE AN APPROXIMATION OF THE NEXT TWO QUOTIENT BITS TO BE PROCESSED WHILE THE PRESENT TWO BITS (WHICH THE DECODER SELECTED DURING THE PREVIOUS ADDER CYCLE) ARE BEING VERIFIED OR CORRECTED.

E. THE A-REGISTER SHALL BE 64 BITS WIDE PLUS AN EXTENSION. THIS REGISTER SHALL FUNCTION AS AN INPUT AND THE ACCUMULATION REGISTER FOR THE ADDER. THE A-REGISTER IS ALSO THE ONLY REGISTER FROM WHICH DATA CAN BE REMOVED FROM THE ADDER SECTION.

F. THE C-REGISTER SHALL BE 64 BITS WIDE PLUS AN EXTENSION AND WILL NORMALLY BE THE SECOND INPUT TO THE ADDER. IN MULTIPLY, IT SHALL RECEIVE SMALL MULTIPLES OF THE MULTIPLICAND.

G. THE D-REGISTER SHALL BE 64 BITS WIDE PLUS AN EXTENSION. THIS REGISTER SHALL FEED THE DECIMAL CORRECTION TO THE ADDER FOR DECIMAL OPERATIONS. IN BINARY MULTIPLY, THE LARGE MULTIPLES OF THE BINARY MULTIPLICAND WILL BE SENT TO THE D-REGISTER FOR PRESENTATION TO THE ADDER.

H. THE E-REGISTER SHALL BE 64 BITS WIDE AND WILL BE USED TO HOLD THE MULTIPLICAND IN MULTIPLY AND THE DIVISOR IN DIVIDE.

I. THE F-REGISTER SHALL BE 64 BITS WIDE PLUS AN EXTENSION. THIS REGISTER SHALL BE USED TO HOLD THREE TIMES THE MULTIPLICAND OR DIVISOR IN BINARY OPERATIONS OR TWO TIMES THE MULTIPLICAND OR DIVISOR IN DECIMAL OPERATIONS. IN MULTIPRECISION ADD AND IN CONVERSION THE F-REGISTER BUFFERS THE ADDER OUTPUT.

J. THE P-REGISTER SHALL BE SIX BITS WIDE AND WILL ESSENTIALLY BE USED AS AN INTERFACE BETWEEN THE ADDER SECTION AND THE B-REGISTER. IN MULTIPLY, THE P-REGISTER SHALL RECEIVE THOSE BITS OF PRODUCT OR PARTIAL PRODUCT WHICH THE ADDER SECTION HAS FINISHED PROCESSING AT THAT TIME AND SEND THEM TO THE B-REGISTER. IT SHALL ALSO PERFORM THE SAME OPERATION FOR QUOTIENT BITS IN DIVIDE AND FOR OVERFLOW BITS IN CONVERSION.

K. THE RM REGISTER SHALL BE A DUPLICATION OF SIX OF THE MOST SIGNIFICANT BITS IN THE A-REGISTER. THE PURPOSE OF THE RM REGISTER WILL BE TO INPUT PART OF THE REMAINDER TO THE BINARY DIVIDE DECODE.

THE ADDER SECTION SHALL ALSO INCLUDE FIVE TWO-BIT RESIDUE REGISTERS THAT WILL BE USED TO STORE THE RESIDUE VALUE FOR THE FOLLOWING ELEMENTS OF THE ADDER SECTION: THE A-REGISTER, THE SUM OF THE C-REGISTER AND D-REGISTER, THE E-REGISTER, THE F-REGISTER AND THE P-REGISTER.

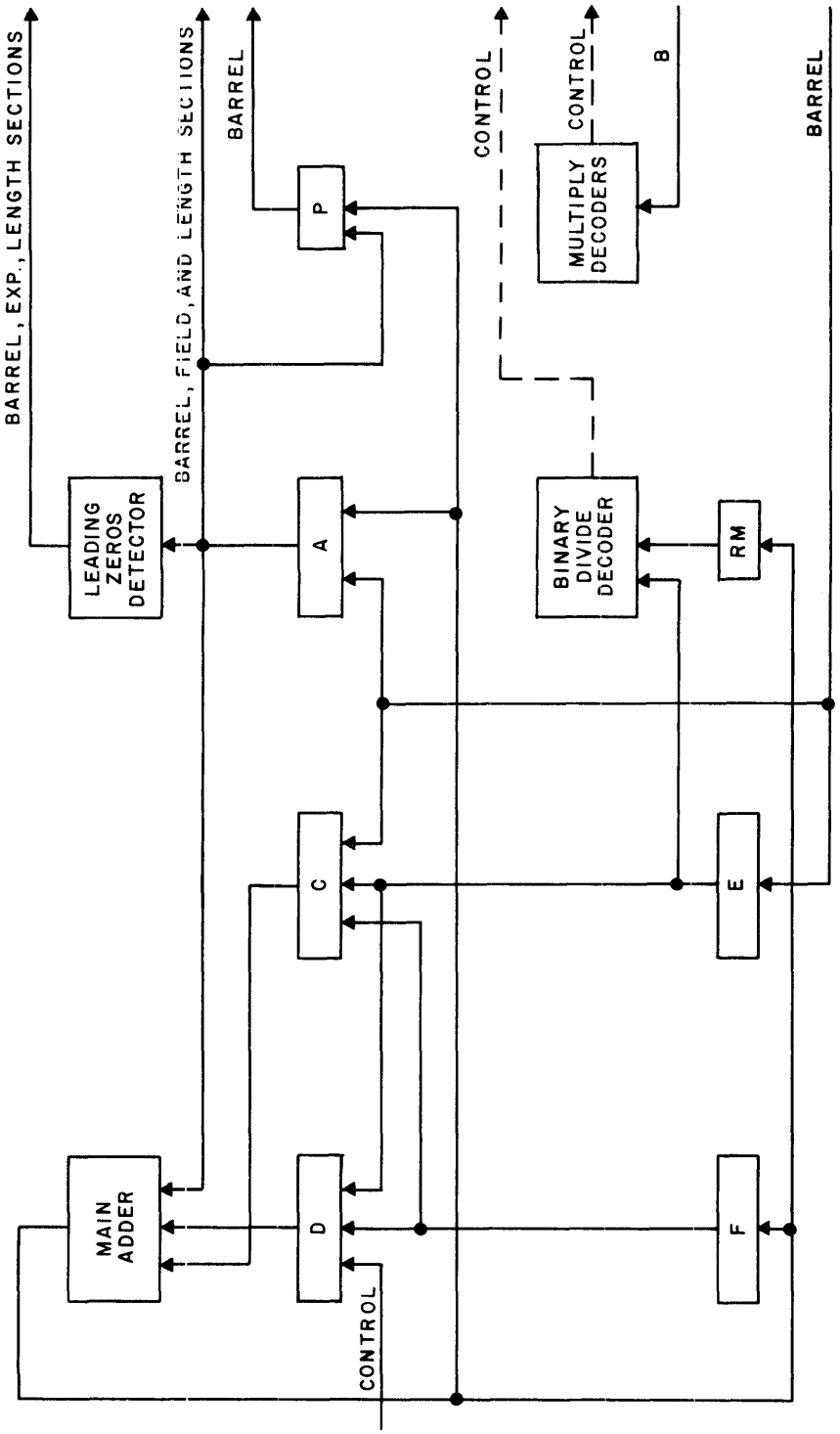


FIGURE 27. ADDER SECTION, BLOCK DIAGRAM

3.4.4 EXPONENT SECTION

THE EXPONENT SECTION SHALL EXECUTE THE EXPONENT CALCULATIONS REQUIRED FOR FAST FLOATING POINT OPERATIONS.

THE EXPONENT SECTION SHALL CONSIST OF THREE FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 28, AND OPERATED AS DESCRIBED BELOW.

A. THE G-REGISTER SHALL BE 12 BITS WIDE AND WILL BE USED AS A PRIMARY INPUT REGISTER AND AN OUTPUT REGISTER OF THE EXPONENT SECTION.

B. THE H-REGISTER SHALL BE 12 BITS WIDE AND WILL BE USED AS THE SECONDARY INPUT REGISTER.

C. THE EXPONENT ADDER SHALL BE 12 BITS WIDE AND WILL BE USED TO DEVELOP THE SUM OF THE G AND H REGISTER OUTPUTS.

THE EXPONENT SECTION SHALL ALSO INCLUDE TWO TWO-BIT RESIDUE REGISTERS THAT WILL BE USED TO STORE THE RESIDUE VALUES FOR THE FOLLOWING ELEMENTS OF THE EXPONENT SECTION: THE G-REGISTER AND H-REGISTER.

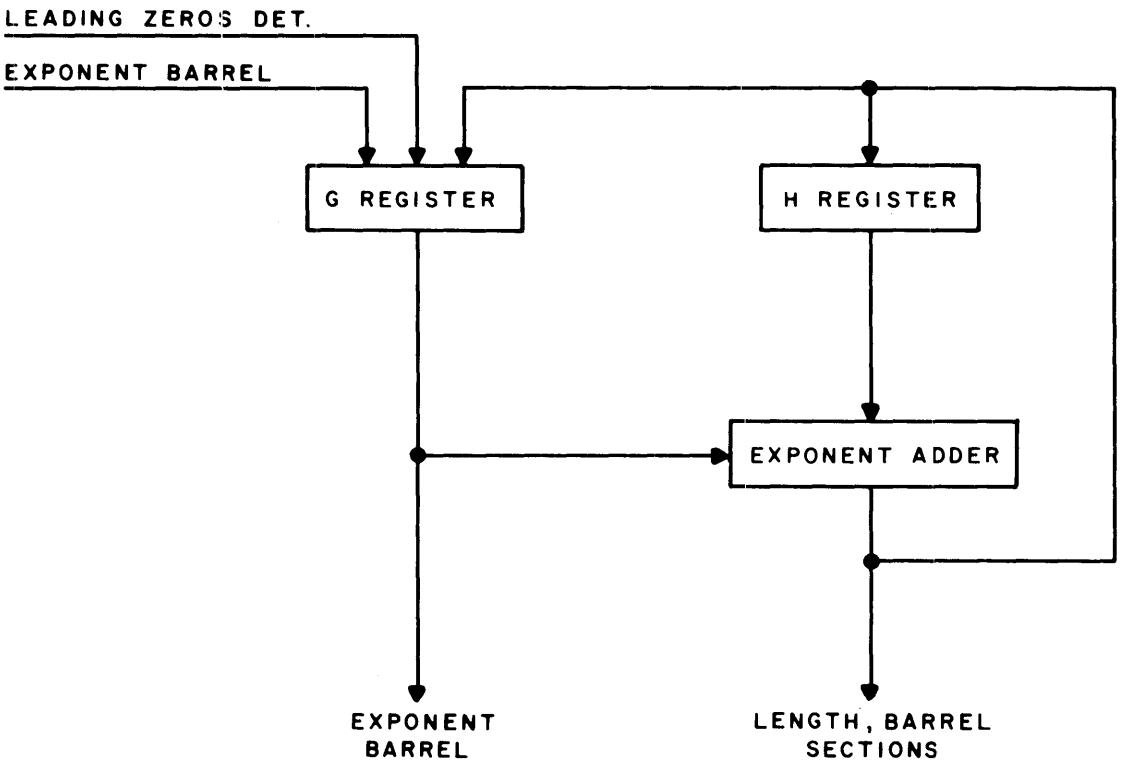


FIGURE 28. EXPONENT SECTION, BLOCK DIAGRAM

3.4.5 LENGTH SECTION

THE LENGTH SECTION SHALL PROCESS LENGTH DATA IN ORDER TO DEFINE THE TRANSFER OF VALUE DATA BETWEEN THE STORAGE SECTION AND THE ADDER OR EXPONENT SECTIONS. IN ORDER TO ACCOMPLISH THIS, THE LENGTH SECTION SHALL INCLUDE AN INITIAL DEFINITION OF THE VALUE DATA AND THE MEANS TO PROCESS THIS DEFINITION.

THE OPERAND (WHICH IS THE DATA IN THE STORAGE SECTION) IS DEFINED BY THE CORRESPONDING NAME STACK AND FORMAT ELEMENTS. AN EXPANDED COPY OF THIS DEFINING INFORMATION, INCLUDING THE LOCATIONS AND LENGTHS OF THE FIELDS WITHIN THE OPERAND, IS KEPT IN THE LENGTH SECTION AS PERMANENT DATA WHICH IS NOT ALTERED UNTIL THE OPERAND IS REMOVED FOR THE STORAGE.

THE FIELD OF VALUE DATA IN THE ADDER OR EXPONENT SECTION DOES NOT REQUIRE EXPLICIT DEFINITION BY THE LENGTH SECTION IF THE DATA IS THE SAME IN BOTH THE ADDER AND STORAGE, AND IS EITHER LEFT-JUSTIFIED OR RIGHT-JUSTIFIED IN THE ADDER. IF THESE CONDITIONS ARE NOT MET, THEN THE ADDER FIELD LENGTH MUST BE KNOWN TO THE LENGTH SECTION. IF THE ADDER FIELD IS NOT JUSTIFIED, THEN THE AMOUNT BY WHICH IT IS SHIFTED MUST ALSO BE SAVED IN THE LENGTH SECTION.

PROCESSING THE STORAGE AND ADDER VALUE DATA REQUIRES CURRENT LOCATION AND CURRENT LENGTH INFORMATION. THESE ARE KNOWN RESPECTIVELY, AS POINTERS AND COUNTERS. IF A NON-ZERO SHIFT AMOUNT WAS DEVELOPED FOR A TRANSFER, THEN IT MUST ALSO BE PROCESSED.

THE LENGTH SECTION SHALL CONSIST OF 20 FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 29, AND OPERATED AS DESCRIBED BELOW.

ALL 20 COMPONENTS SHALL BE TWELVE BITS WIDE. THE TWELVE BITS SHALL INCLUDE THE FOLLOWING INFORMATION: TEN BITS OF ACTUAL LENGTH DATA (IN SIGNED, TWO(S) COMPLEMENT NOTATION) AND TWO BITS OF RESIDUE.

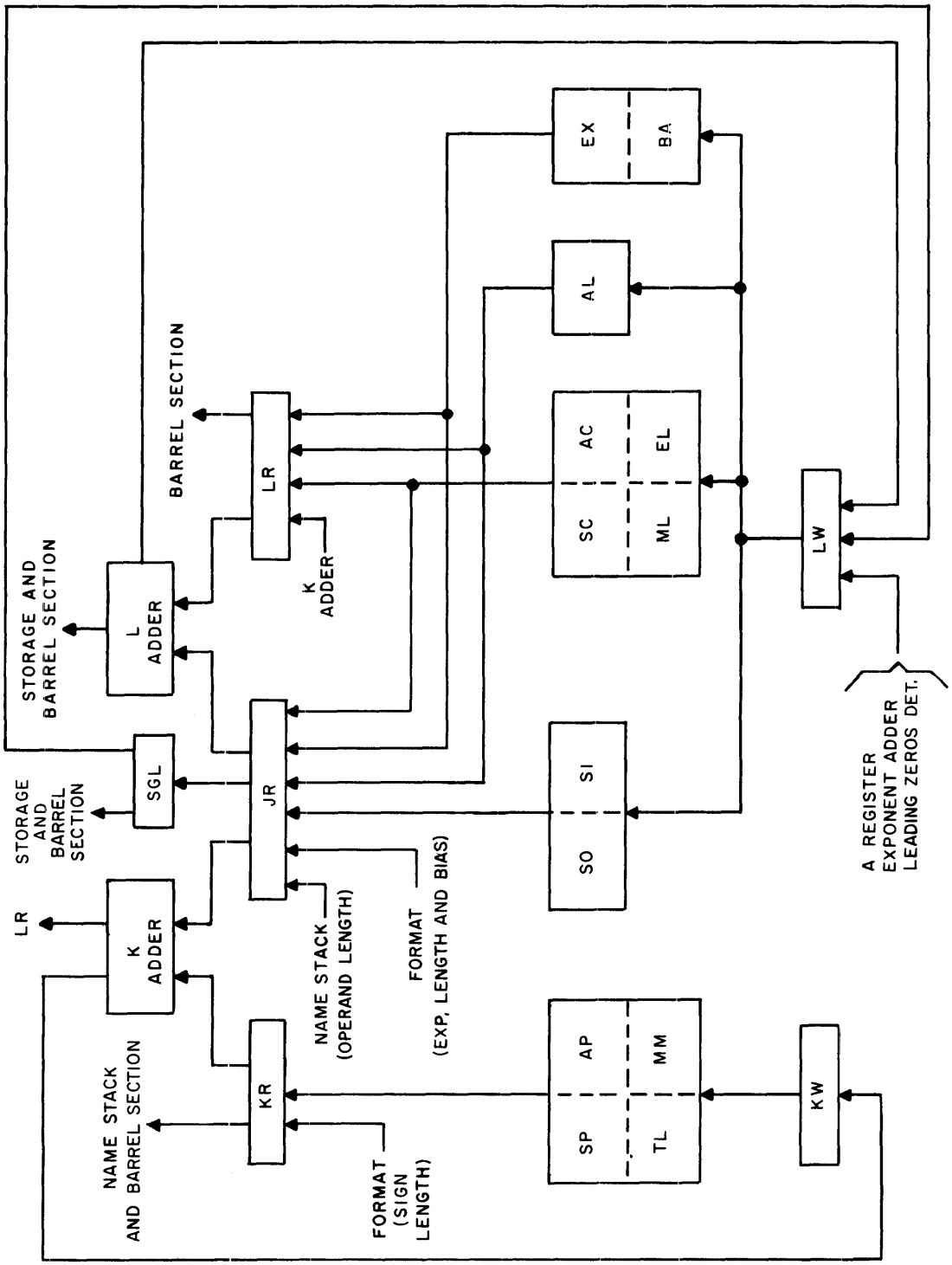


FIGURE 29. LENGTH SECTION, BLOCK DIAGRAM

A. PERMANENT STORAGE

EACH OF THE FOLLOWING STORAGE GROUPS SHALL CONTAIN SIX WORDS. THE SIX WORDS SHALL PROVIDE SPACE FOR PERMANENT LENGTH DATA FOR THE FOUR TOP OPERANDS IN THE VALUE STACK (A,B,C, AND D), AND FOR THE TWO AU OPERAND SPACES (R AND W).

THE LENGTH DATA OF THE FOUR TOP VALUE STACK OPERANDS IS NOT MOVED AS THE CORRESPONDING OPERANDS CHANGE THEIR POSITIONS IN THE VALUE STACK; INSTEAD THE ALLOCATION OF THESE FOUR WORDS CHANGES SO THAT THE LENGTH DATA IS ASSOCIATED WITH THE PROPER OPERAND POSITION IN THE VALUE STACK.

1. TOTAL LENGTH (TL) - THIS GROUP SHALL CONTAIN THE SAME OPERAND LENGTH DATA AS THE CORRESPONDING DESCRIPTIONS IN THE NAME STACK. ITS PRIMARY PURPOSE SHALL BE TO PROVIDE THE LOCATION OF THE LEAST SIGNIFICANT BIT OF THE OPERAND IN THE CORRESPONDING VALUE STACK BUFFER.

2. MANTISSA MOST SIGNIFICANT BIT (MM) - THIS GROUP SHALL CONTAIN THE LOCATION OF THE MOST SIGNIFICANT BIT OF THE MANTISSA OF A FLOATING POINT NUMBER, OR THE LOCATION OF THE MOST SIGNIFICANT BIT OF THE MAGNITUDE OF A FIXED POINT NUMBER.

3. MANTISSA LENGTH (ML) - THIS GROUP SHALL CONTAIN THE LENGTH OF THE MANTISSA OF A FLOATING POINT NUMBER, OR THE LENGTH OF THE MAGNITUDE OF A FIXED POINT NUMBER.

4. EXPONENT LENGTH (EL) - THIS GROUP SHALL CONTAIN THE LENGTH OF THE EXPONENT OF A FLOATING POINT NUMBER.

5. BIAS (BA) - THIS GROUP SHALL CONTAIN THE BIAS OF THE CORRESPONDING OPERAND.

B. TEMPORARY STORAGE

EACH OF THE FOLLOWING STORAGE GROUPS SHALL CONTAIN FOUR WORDS. THE FOUR WORDS SHALL PROVIDE SPACE FOR TEMPORARY LENGTH DATA FOR TWO VALUE STACK INPUTS TO AN OPERATION (A AND B), THE RESULT OF THE OPERATION, AND THE WORK AREA. THIS TEMPORARY LENGTH DATA IS NOT SAVED AFTER THE OPERATION IS COMPLETED. THE ALLOCATION OF THE FOUR WORDS TO THE INPUTS, RESULT, AND WORK AREA WILL NOT CHANGE.

1. STORAGE POINTER (SP) - THIS GROUP SHALL CONTAIN THE LOCATION OF THE NEXT BIT TO BE PROCESSED IN THE CORRESPONDING OPERAND IN THE VALUE STACK.

2. ADDER POINTER (AP) - THIS GROUP SHALL CONTAIN THE LOCATION (WITH RESPECT TO ONE END OF THE TOTAL FIELD) OF THE NEXT BIT TO BE PROCESSED IN THIS ADDER INPUT OR OUTPUT.

3. STORAGE COUNTER (SC) - THIS GROUP SHALL CONTAIN THE NUMBER OF BITS STILL TO BE PROCESSED IN THE CORRESPONDING OPERAND FIELD IN THE VALUE STACK.

4. ADDER COUNTER (AC) - THIS GROUP SHALL CONTAIN THE NUMBER OF BITS STILL TO BE PROCESSED IN AN ADDER INPUT OR OUTPUT.

5. ADDER LENGTH (AL) - THIS GROUP SHALL CONTAIN THE INITIAL LENGTH OF AN ADDER INPUT OR OUTPUT. IT IS UNLIKE THE OTHER TEMPORARY STORAGE GROUPS IN THAT IT IS NOT UPDATED AS THE FIELD IS PROCESSED (THAT UPDATING OCCURS IN AP AND AC).

6. SHIFT STORAGE GROUPS 0 AND 1 (S0 AND S1) - THESE GROUPS SHALL BE THE ALTERNATING STORAGE SPACES FOR THE SHIFT DATA WHICH ALIGNS VALUE DATA AS IT IS MOVED BETWEEN THE STORAGE AND ADDER SECTIONS.

7. EXPONENT (EX) - THIS GROUP SHALL CONTAIN THE EXPONENT CALCULATED FOR FIXED POINT ARITHMETIC.

C. PROCESSING UNITS.

1. LOCATION READ REGISTER (KR) - THIS FLIP-FLOP REGISTER SHALL HOLD THE LOCATION CURRENTLY BEING PROCESSED WITHIN THE CORRESPONDING OPERAND.

2. ADJUSTMENT READ REGISTER (JR) - THIS REGISTER SHALL HOLD THE LENGTH OF NEXT SEGMENT OF DATA TO BE PROCESSED.

3. LENGTH READ REGISTER (LR) - THIS FLIP-FLOP REGISTER SHALL HOLD THE LENGTH OF THE PART OF THE CORRESPONDING OPERAND STILL TO BE PROCESSED.

4. LOCATION ADDER (K ADDER) - THIS ADDER SHALL DEVELOP KR + JR (FOR LEFT TO RIGHT PROCESSING) OR KR - JR (FOR RIGHT TO LEFT PROCESSING).

5. LENGTH ADDER (L ADDER) - THIS ADDER SHALL DEVELOP LR - JR.

6. SEGMENT LENGTH (SGL) - THIS COUNTER SHALL DEVELOP JR (NOT) IF JR IS ONE(S) COMPLEMENT OR JR (NOT) + 1 IF JR IS TWO(S) COMPLEMENT.

D. WRITE REGISTERS.

1. LOCATION WRITE REGISTER (KW) - THIS FLIP-FLOP REGISTER SHALL HOLD LOCATION DATA WHILE THE LOCATION DATA IS WRITTEN INTO SP, AP, TL, OR MM.

2. LENGTH WRITE REGISTER (LW) - THIS FLIP-FLOP REGISTER SHALL HOLD LENGTH DATA WHILE THE LENGTH DATA IS WRITTEN INTO SC, AC, ML, EL, OR AL; OR IT SHALL HOLD SHIFT DATA WHILE THE SHIFT DATA IS WRITTEN INTO S0 OR S1; OR IT SHALL HOLD EXPONENT DATA WHILE THE EXPONENT DATA IS WRITTEN INTO BA OR EX.

3.4.6 FIELD SECTION

THE FIELD SECTION SHALL PROVIDE THE MEANS TO OBTAIN RESIDUE CHECKING ON FIELDS AS THEY ARE USED IN THE AU. THE FIELDS, WHICH CAN HAVE RESIDUE CHECKING, INCLUDE ALL STORAGE FIELDS (WHERE A FIELD MAY BE THE ENTIRE OPERAND, THE EXPONENT, OR THE MANTISSA), AND ALL ADDER FIELDS UPON WHICH ARITHMETIC OPERATIONS ARE PERFORMED. STORAGE FIELD CHECKING PROVIDES THE ONLY MEANS FOR CHECKING THE OPERAND STORAGE AND THE EXTRACTION AND INSERTION OF OPERAND SEGMENTS FROM AND TO STORAGE. ADDER FIELD CHECKING PROVIDES THE ONLY RESIDUE CHECK ON THE ADDER CONTROLS.

THE FIELD SECTION SHALL CONSIST OF 22 FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 30, AND OPERATED AS DESCRIBED BELOW.

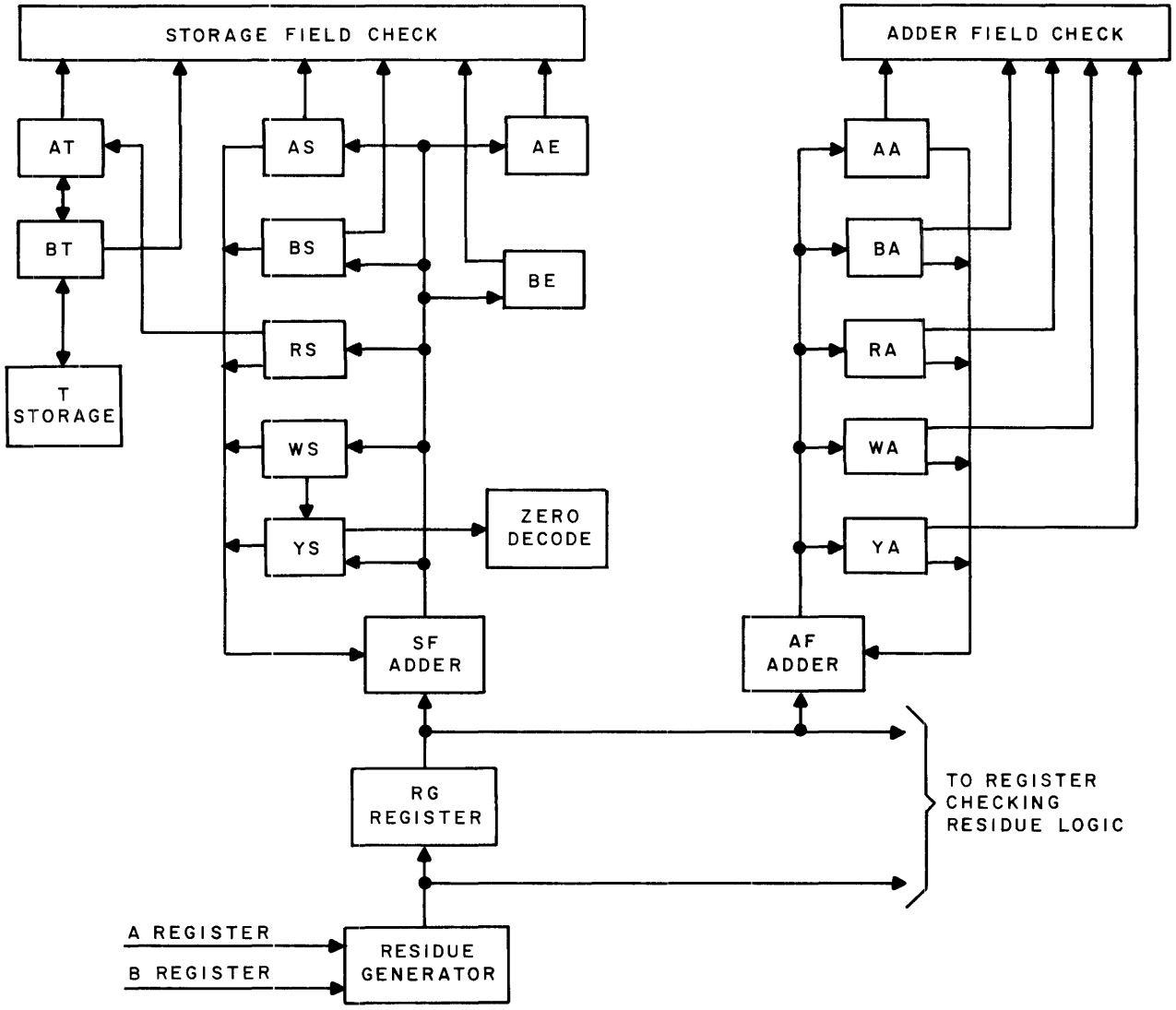


FIGURE 30. FIELD SECTION, BLOCK DIAGRAM

A. THE RESIDUE GENERATOR SHALL GENERATE A THREE-BIT RESIDUE DECODE FROM A 64-BIT INPUT. THIS INPUT IS SELECTED FROM EITHER THE A-REGISTER OR THE B-REGISTER.

B. THE RG REGISTER SHALL BE A 3-BIT WIDE FLIP-FLOP REGISTER WHICH STORES THE OUTPUT OF THE RESIDUE GENERATOR.

C. THE STORAGE FIELD ADDER (SF ADDER) SHALL BE A TWO BIT WIDE RESIDUE ADDER WHICH ADDS OR SUBTRACTS RG TO OR FROM ONE OF THE STORAGE FIELD RESIDUE REGISTERS.

D. THE ADDER FIELD ADDER (AF ADDER) SHALL BE TWO BIT WIDE RESIDUE ADDER WHICH ADDS RG TO ONE OF THE ADDER FIELD RESIDUE REGISTERS.

E. THE TOTAL RESIDUE REGISTERS AND STORAGE (AT, BT, AND T STORAGE) SHALL CONTAIN THE TWO-BIT WIDE RESIDUES OF THE VALUE STACK OPERANDS AS THEY APPEAR IN THE STORAGE SECTION.

1. THE A TOTAL RESIDUE REGISTER (AT) CONTAINS THE RESIDUE OF THE A OPERAND.

2. THE B TOTAL RESIDUE REGISTER (BT) CONTAINS THE RESIDUE OF THE B OPERAND.

3. THE TOTAL RESIDUE STORAGE (T STORAGE) CONTAINS THE RESIDUES OF THE C AND D OPERANDS.

F. THE STORAGE FIFLD RESIDUE REGISTERS (AS, BS, RS, WS, AND YS) SHALL BE TWO BIT WIDE FLIP-FLOP REGISTERS WHICH CONTAIN THE RESIDUE OF THE DEVELOPING MANTISSA OR MAGNITUDE FIELD WITHIN AN OPERAND.

1. THE A STORAGE FIELD RESIDUE REGISTER (AS) CONTAINS THE RESIDUE OF THE A MANTISSA OR MAGNITUDE.

2. THE B STORAGE FIELD RESIDUE REGISTER (BS) CONTAINS THE RESIDUE OF THE B MANTISSA OR MAGNITUDE.

3. THE R STORAGE FIELD RESIDUE REGISTER (RS) CONTAINS THE RESIDUE OF THE ENTIRE RESULT.

4. THE W STORAGE FIELD RESIDUE REGISTER (WS) CONTAINS THE RESIDUE OF THE OUTPUT TO THE WORK AREA.

5. THE Y STORAGE FIELD RESIDUE REGISTER (YS) CONTAINS THE RESIDUE OF THE INPUT TO THE WORK AREA.

G. THE EXPONENT RESIDUE REGISTERS (AE AND BE) SHALL BE TWO-BIT WIDE FLIP-FLOP REGISTERS WHICH CONTAIN THE SUM OF THE SIGN AND EXPONENT RESIDUES.

1. THE A EXPONENT RESIDUE REGISTER (AE) CONTAINS THE SUM OF THE SIGN AND EXPONENT RESIDUES OF THE A OPERAND.

2. THE B EXPONENT RESIDUE REGISTER (BE) CONTAINS THE SUM OF THE SIGN AND EXPONENT RESIDUES OF THE B OPERAND.

H. THE ADDER FIELD RESIDUE REGISTERS (AA, BA, RA, WA, AND YA) SHALL BE TWO-BIT WIDE FLIP-FLOP REGISTERS WHICH CONTAIN THE RESIDUE OF THE CORRESPONDING ADDER FIELD.

1. THE A ADDER FIELD RESIDUE REGISTER (AA) CONTAINS THE RESIDUE OF THE A OPERAND ADDER FIELD.

2. THE B ADDER FIELD RESIDUE REGISTER (BA) CONTAINS THE RESIDUE OF THE B OPERAND ADDER FIELD.

3. THE R ADDER FIELD RESIDUE REGISTER (RA) CONTAINS THE RESIDUE OF THE RESULT ADDER FIELD.

4. THE W ADDER FIELD RESIDUE REGISTER (WA) CONTAINS THE RESIDUE OF THE WORK AREA ADDER OUTPUT.

5. THE Y ADDER FIELD RESIDUE REGISTER (YA) CONTAINS THE RESIDUE OF THE WORK AREA ADDER INPUT.

I. THE STORAGE FIELD CHECK SHALL BE A TWO-BIT WIDE ADDER AND COMPARATOR WHICH CHECKS THAT AT = AS + AE AFTER EACH TIME THE A MANTISSA OR MAGNITUDE IS COMPLETELY SCANNED, AND THAT BT = BS + BE AFTER EACH TIME THE B MANTISSA OR MAGNITUDE IS COMPLETELY SCANNED.

J. THE ZERO RECODE CHECKS THAT YS = 0 AFTER EACH TIME THE INPUT FROM THE WORK AREA IS COMPLETELY SCANNED.

K. THE ADDER FIELD CHECK SHALL BE A TWO BIT WIDE ADDER AND COMPARATOR WHICH CHECKS THAT THE APPROPRIATE RELATIONSHIP EXISTS AMONG THE ADDER FIELD REGISTERS AFTER EACH USE OF THE ADDER FOR A COMPLETE FIELD OR GROUP OF FIELDS.

3.4.7 FORMAT SECTION

THE FORMAT SECTION SHALL STORE THREE TYPES OF INFORMATION : THE ARITHMETIC OPERATION VARIANTS , INCLUDING THE OUTPUT LENGTH FOR FIXED LENGTH OPERATIONS ; A COPY OF THE FORMAT TABLE WHICH MAY HOLD UP TO SEVEN OPERAND FORMATS ; AND THE FORMAT INFORMATION THAT IS ASSOCIATED WITH EACH VALUE STACK OPERAND IN THE STORAGE SECTION.

THE FORMAT SECTION SHALL CONSIST OF FOUR FUNCTIONAL COMPONENTS INTERFACED AS SHOWN IN FIGURE 31, AND OPERATED AS DESCRIBED BELOW.

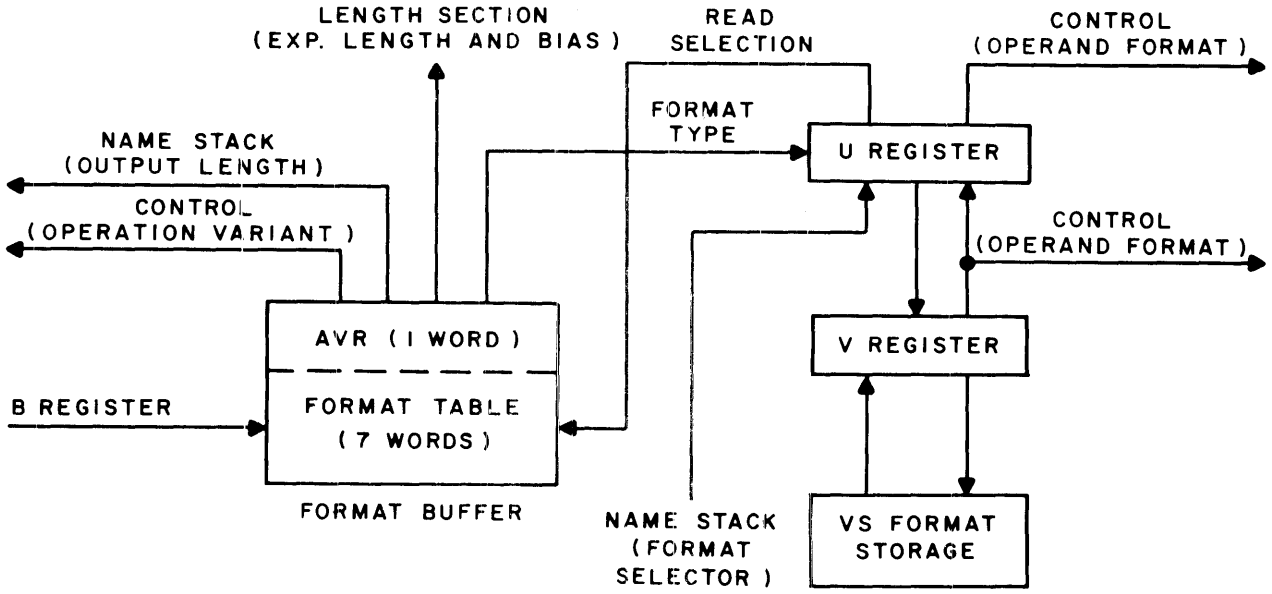


FIGURE 31. FORMAT SECTION, BLOCK DIAGRAM

A. THE FORMAT BUFFER SHALL BE AN EIGHT WORD BY 21 BIT MEMORY WHICH WILL BE ALLOCATED AS FOLLOWS:

1. ONE WORD IS USED FOR THE ARITHMETIC VARIANT REGISTER (AVR). THE FORMAT OF THE AVR IS ILLUSTRATED IN FIGURE 32. THIS WORD IS CONSTANTLY READ DURING ALL ARITHMETIC OPERATIONS.

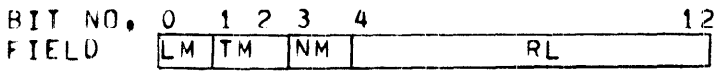


FIGURE 32. AVR FORMAT

THE VARIOUS FIELDS OF THE AVR ARE DEFINED AS FOLLOWS:

LENGTH MODE (LM) FIXED LENGTH (LM=0)
VARIABLE LENGTH (LM=1)

TRUNCATION MODE (TM)	NOT ROUNDED, MAGNITUDE TRUNCATION (TM=0) NOT ROUNDED, ALGEBRAIC TRUNCATION (TM=1) ROUNDED, MAGNITUDE TRUNCATION (TM=2) ROUNDED, ALGEBRAIC TRUNCATION (TM=3)
NORMALIZATION MODE (NM)	UNCONDITIONAL NORMALIZATION (NM=0) CONDITIONAL NORMALIZATION (NM=1)
RESULT LENGTH (RL)	UNSIGNED BINARY NUMBER NOT EXCEEDING 256

2. SEVEN WORDS ARE USED TO COMPRISE THE FORMAT TABLE. THE FORMAT FOR ANY ONE OF THESE WORDS IS ILLUSTRATED IN FIGURE 33. ONE OF THESE WORDS IS SELECTED TO BE READ (BY THE NAME STACK FORMAT SELECTOR LOADED IN THE U-REGISTER) DURING EITHER OF THE FOLLOWING EVENTS: LOADING A NEW OPERAND INTO THE STORAGE SECTION; OR PRIOR TO EXECUTING A TRANSFORM OPERATOR.

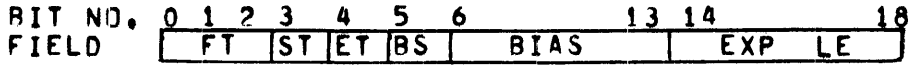


FIGURE 33. WORD FORMAT

THE VARIOUS FIELDS OF THE WORD ARE DEFINED AS FOLLOWS:

FORMAT TYPE (FT)	LOGICAL (FT≠0) BINARY (FT=1) OCTAL (FT=2) DECIMAL (FT=5) ASCII (FT=6) EBCDIC (FT=7)
SIGN TYPE (ST)	UNSIGNED (ST=0) SIGNED (ST=1)
EXPONENT TYPE (ET)	FIXED POINT (ET=0) FLOATING POINT (ET=1)
BIAS SIGN (BS)	BINARY SIGN
BIAS	BINARY MAGNITUDE NOT EXCEEDING 255
EXPONENT LENGTH (EXP LE)	NOT EXCEEDING 30 IF BINARY OR OCTAL NOT EXCEEDING 24 IF DECIMAL

B. THE U-REGISTER SHALL BE A 12-BIT FLIP-FLOP REGISTER WHICH CONTAINS AN ACTIVE VS FORMAT. THE VS FORMAT, WHICH IS ASSEMBLED BY THE HARDWARE, IS ILLUSTRATED IN FIGURE 34. VS FORMATS ARE ASSEMBLED IN THE U-REGISTER DURING EACH FETCH TO THE VALUE STACK AND AT THE START OF EACH TRANSFORM OPERATOR. DURING THE EXECUTION OF A TRANSFORM OPERATION, THE U-REGISTER WILL HOLD THE RESULT FORMAT. FOR ALL OTHER ARITHMETIC OPERATING, THE U-REGISTER WILL HOLD THE FORMAT OF A (THE TOP OPERAND IN THE VALUE STACK).

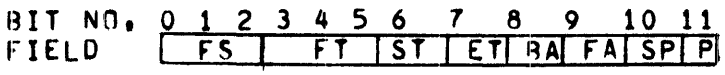


FIGURE 34. VS FORMAT

THE VARIOUS FIELDS OF THE VS FORMAT ARE DEFINED AS FOLLOWS:

FORMAT SELECTOR (FS)	FORMAT TABLE ADDRESS (NUMBERS 1 THRU 7)
FORMAT TYPE (FT)	REFER TO FIGURE 33
SIGN TYPE (ST)	REFER TO FIGURE 33
EXPONENT TYPE (ET)	REFER TO FIGURE 33
BIAS PRESENT BIT (BA)	NO BIAS (BA=0) BIAS (BA=1)
FAST ALGORITHM (FA)	GENERAL ALGORITHM REQUIRED (FA=0) FAST ALGORITHM POSSIBLE (FA=1)
SINGLE PRECISION (SP)	NOT SINGLE PRECISION (SP=0) (1) LOGICAL OPERAND LENGTH EXCEEDING 32 BITS (2) ARITHMETIC OPERAND LENGTH EXCEEDING 64 BITS SINGLE PRECISION (SP=1) (1) LOGICAL OPERAND LENGTH NOT EXCEEDING 32 BITS (2) ARITHMETIC OPERAND LENGTH NOT EXCEEDING 64 BITS
PARITY BIT (P)	VS FORMAT HAS ODD PARITY

C. THE V-REGISTER SHALL BE A 12-BIT FLIP-FLOP REGISTER WHICH CONTAINS AN ACTIVE VS FORMAT. DURING THE EXECUTION OF TRANSFORM, THE V-REGISTER WILL HOLD THE INPUT FORMAT; DURING ALL OTHER ARITHMETIC OPERATIONS, IT WILL HOLD THE FORMAT OF B (THE SECOND OPERAND IN THE VALUE STACK).

D. THE VS FORMAT STORAGE SHALL BE A 12-BIT WIDE MEMORY WHICH MAY CONTAIN UP TO FOUR ACTIVE VS FORMATS. THE VS FORMATS FOR C AND D (THE THIRD AND FOURTH OPERANDS IN THE VALUE STACK) WILL NORMALLY BE IN THE VS FORMAT STORAGE WHENEVER THE CORRESPONDING OPERANDS ARE IN THE STORAGE SECTION.

THE FORMAT SECTION SHALL ALSO INCLUDE RESIDUE CHECKING ON THE FORMAT BUFFER BY INCLUDING TWO-RESIDUE BITS WITHIN EACH 21-BIT WORD. THE U-REGISTER, V-REGISTER, AND EACH WORD OF THE VS FORMAT STORAGE SHALL INCLUDE A PARITY BIT WHICH WILL ALLOW PARITY CHECKING ON ALL VS FORMATS.

3.4.8 AU DATA CONTROL

THE CONTROL OF THE AU DATA HARDWARE IS ACCOMPLISHED BY ORGANIZING THE AU DATA HARDWARE AND THEIR IMMEDIATE CONTROLS INTO 14 SUB-UNITS. EACH SUB-UNIT IS COMPRISED OF DATA AND/OR MICRO-COMMAND REGISTERS. A SUB-UNIT HAS TWO DISTINCT CHARACTERISTICS. FIRST, IT DOES NOT SHARE ANY OF ITS REGISTERS WITH ANOTHER SUB-UNIT. SECONDLY, IT CAN NOT BE SEPARATED INTO NEW SUB-UNITS. EACH SUB-UNIT CAN PERFORM A TASK WITHOUT INTFRFERING WITH ANOTHER SUB-UNIT EVEN THOUGH ALL THE SUB-UNITS MAY BE BUSY DOING DIFFERENT PARTS OF AN OPERATION AT THE SAME TIME . EACH SUB-UNIT CAN BE ACCESSED BY ONE OR TWO SUB-UNITS. AT TIMES, A SOURCE SUB-UNIT MUST WAIT UNTIL THE DESTINATION SUB-UNIT IS AVAILABLE, OR UNTIL AN OUTPUT READY CONDITION IS INDICATED FROM SOME SUB-UNIT.

THE ORGANIZATION OF THE 14 SUB-UNITS IS ILLUSTRATED IN FIGURE 35. THE FUNCTIONS AND CONTENTS OF EACH SUB-UNIT ARE DESCRIBED IN PARAGRAPHS A THRU N.

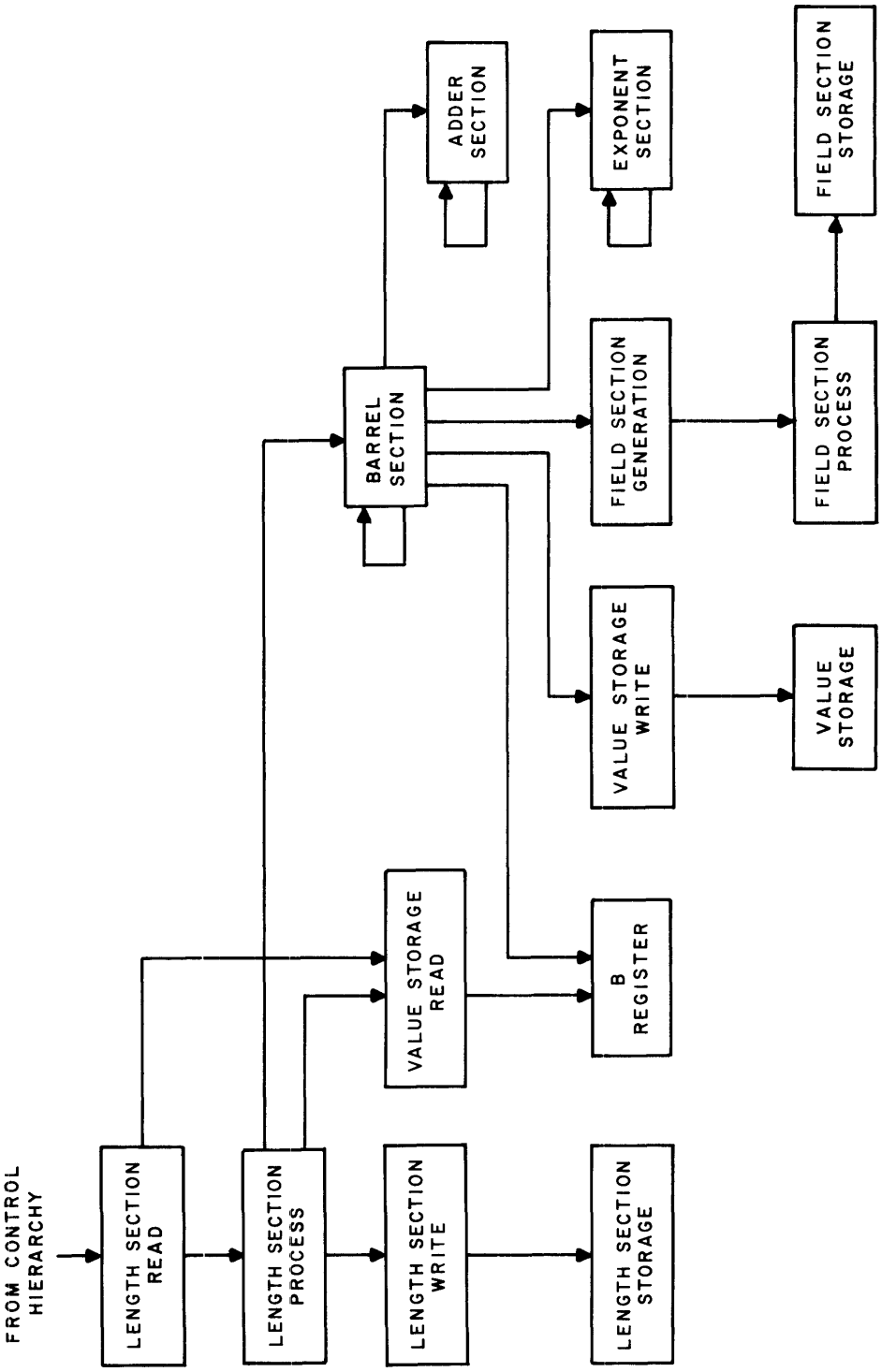


FIGURE 35. SUB-UNIT ORGANIZATION, BLOCK DIAGRAM

A. THE LENGTH SECTION READ SUB-UNIT DEFINES THE NEXT LENGTH EXPRESSION TO BE EVALUATED. IT IS COMPRISED OF A MICRO-COMMAND REGISTER AND READ ADDRESS REGISTERS.

B. THE LENGTH SECTION PROCESS SUB-UNIT PROCESSES A LENGTH EXPRESSION. IT IS COMPRISED OF A MICRO-COMMAND REGISTER, THE JR, KR, AND LR LENGTH DATA REGISTERS AND ASSOCIATED LENGTH ADDER CONTROL FLIP-FLOPS.

C. THE LENGTH SECTION WRITE SUB-UNIT WRITES THE RESULT OF A LENGTH EXPRESSION. IT IS COMPRISED OF A WRITE ADDRESS REGISTER AND THE KW AND LW LENGTH DATA REGISTERS.

D. THE LENGTH SECTION STORAGE SUB-UNIT CONTAINS ALL LENGTH DATA NOT CURRENTLY BEING PROCESSED. IT IS COMPRISED OF THE LENGTH SECTION MEMORY.

E. THE VALUE STORAGE READ SUB-UNIT READS THE VALUE STORAGE. IT IS COMPRISED OF A READ ADDRESS REGISTER.

F. THE B REGISTER SUB-UNIT HOLDS A 64-BIT VALUE. IT IS COMPRISED OF THE B-REGISTER AND AN OUTPUT READY FLIP-FLOP.

G. THE VALUE STORAGE WRITE SUB-UNIT WRITES INTO THE VALUE STORAGE. IT IS COMPRISED OF A WRITE ADDRESS REGISTER.

H. THE VALUE STORAGE SUB-UNIT CONTAINS ALL VALUE DATA NOT CURRENTLY BEING PROCESSED. IT IS COMPRISED OF THE STORAGE SECTION MEMORY.

I. THE BARREL SECTION SUB-UNIT EXTRACTS AND INSERTS VALUES. IT IS COMPRISED OF A MICRO-COMMAND REGISTER, A REPETITION COUNTER, A SHIFT REGISTER, MASK REGISTERS, AND BARREL SELECTION FLIP-FLOPS.

J. THE FIELD SECTION GENERATION SUB-UNIT GENERATES RESIDUES OF VALUES. IT IS COMPRISED OF A MICRO-COMMAND REGISTER AND RESIDUES SELECTION FLIP-FLOPS.

K. THE FIELD SECTION PROCESS SUB-UNIT UPDATES FIELD RESIDUE REGISTERS. IT IS COMPRISED OF A MICRO-COMMAND REGISTER AND THE RG REGISTER.

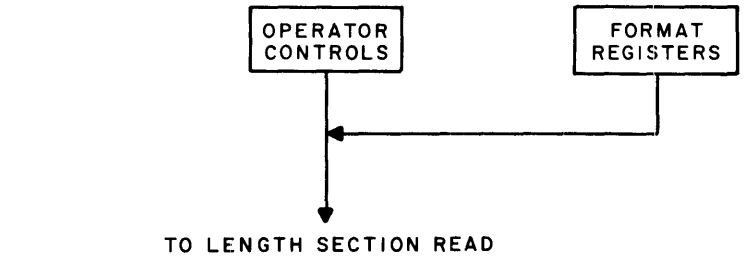
L. THE FIELD SECTION STORAGE SUB-UNIT STORES FIELD RESIDUES. IT IS COMPRISED OF THE FIELD RESIDUE REGISTERS.

M. THE ADDER SECTION SUB-UNIT ADDS AND SUBTRACTS SEGMENTS, MULTIPLIES SEGMENTS BY DIGITS, DEVELOPS QUOTIENT DIGITS, AND CONVERTS DIGITS AND ACCUMULATES THEM TO CONVERTED SEGMENTS. IT IS COMPRISED OF A MICRO-COMMAND REGISTER, SEQUENCING FLIP-FLOPS, THE A,C,D,E,F,P, AND RM REGISTERS, THE ASSOCIATED ADDER CONTROL FLIP-FLOPS, AND TWO OUTPUT READY FLIP-FLOPS.

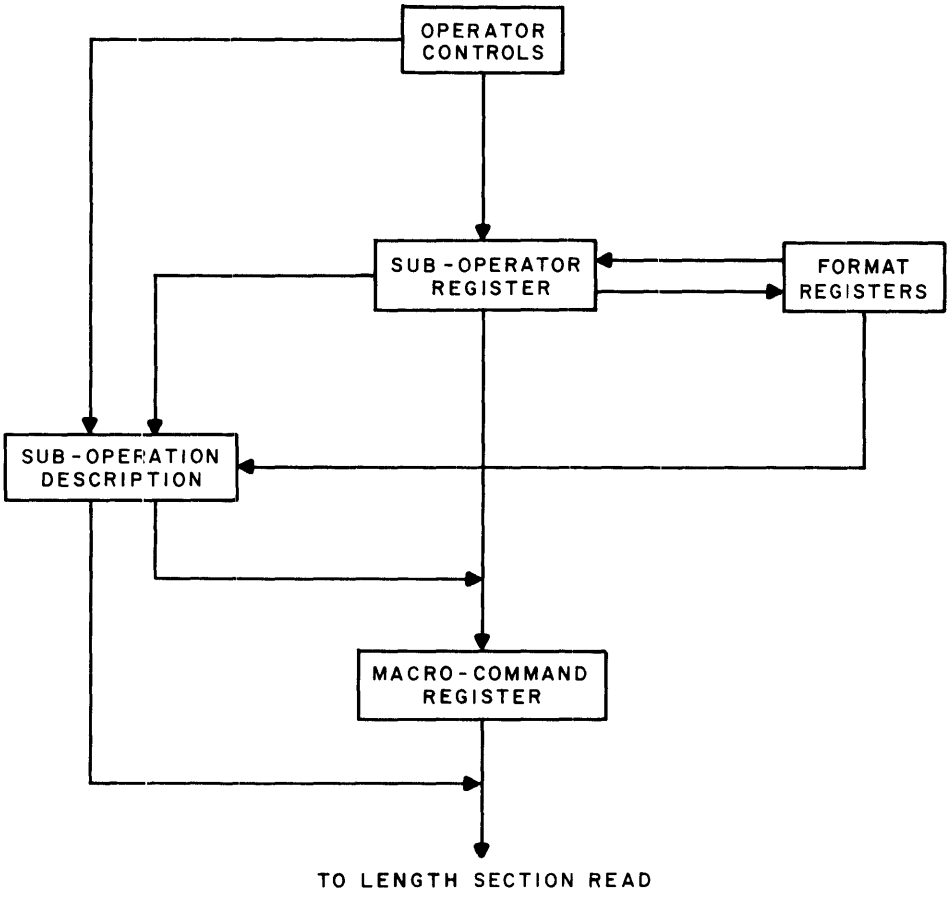
N. THE EXPONENT SECTION SUB-UNIT ADDS AND SUBTRACTS EXPONENTS FOR FAST FLOATING-POINT ALGORITHMS. IT IS COMPRISED OF A MICRO-COMMAND REGISTER, SEQUENCING FLIP-FLOPS, THE G AND H REGISTERS, AND TWO OUTPUT READY FLIP-FLOPS.

3.4.9 CONTROL HIERARCHY

THE CONTROL HIERARCHY (SEE FIGURE 36) FOR AU OPERATORS HAS TWO MODES. THE SIMPLER OF THESE IS THE FAST ALGORITHM MODE. FAST ALGORITHMS ARE DESIGNED FOR OPERANDS SATISFYING RESTRICTIVE FORMAT CONDITIONS, SO THAT MANY ASPECTS OF AN ARITHMETIC OPERATION WILL NOT REQUIRE EXPLICIT DEFINITION. IN THESE OPERATIONS, MICRO-COMMANDS ARE ISSUED DIRECTLY FROM THE OPERATOR REGISTER (INFLUENCED BY THE ARITHMETIC VARIANT AND FORMAT REGISTERS). TO THE LENGTH SECTION READ REGISTER.



FAST MODE



GENERAL MODE

FIGURE 36. CONTROL HIERARCHY, BLOCK DIAGRAM

THE MORE COMPLEX MODE IS THE GENERAL ALGORITHM MODE, WHICH ONLY ASSUMES THAT THE OPERANDS HAVE LEGAL FORMATS. THE GREATER COMPLEXITY SUGGESTS MORE LEVELS OF CONTROL, IN ORDER TO MINIMIZE THE AMOUNT OF UNIQUE HARDWARE REQUIRED FOR A SINGLE OPERATOR-FORMAT COMBINATION.

THE FIRST LEVEL BELOW THE OPERATOR IS CALLED THE SUB-OPERATOR. THIS LEVEL IS RESPONSIBLE FOR ACCOMPISHING A SINGLE, COMPLETE ACTION OF CONVERTING A FIELD, OR COMBINING TWO OR MORE FIELDS, OR SOMETHING ELSE OF EQUIVALENT SCOPE. SOMF SUB-OPERATORS ARE UNIQUE TO CERTAIN OPERATORS, OTHERS MIGHT BE USED BY ANY OPERATOR.

THE SECOND LEVEL BELOW THE OPERATOR IS CALLED THE MACRO-COMMAND. THIS LEVEL IS RESPONSIBLE FOR ACCOMPLISHING A SINGLE, COMPLETE ACTION OF OPERATING ON A SEGMENT FROM A FIELD, OR DEVELOPING A LENGTH VALUE WHICH DEFINES A FIELD, OR SOMETHING ELSE OF EQUIVALENT SCOPE. MOST MACRO-COMMANDS ARE USED BY MANY SUB-OPERATORS.

THE MICRO-COMMAND LEVEL IS USED IN BOTH FAST AND GENERAL ALGORITHMS. A MICRO-COMMAND IS RESPONSIBLE FOR SINGLE, COMPLETE USE OF A SUB-UNIT OF THE AU, SUCH AS DEVELOPING A SUM IN THE LENGTH ADDER, A SUM IN THE MAIN ADDER, OR A DIGIT OF QUOTIENT IN THE MAIN ADDER.

IN THE GENERAL ALGORITHM MODE, EACH OPERATOR CAUSES A SUCCESSION OF SUB-OPERATORS (WHICH IS MODIFIED IF THE FORMATS ARE NOT THE SAME). DURING EACH SUB-OPERATOR, THE OPERATOR REGISTER, ARITHMETIC VARIANT REGISTER, FORMAT (U AND V) REGISTERS, AND THE SUB-OPERATION REGISTER ARE DECODED TO DEVELOP A SUB-OPERATION DESCRIPTION DECODE. THIS DECODE DEFINES THE REQUIREMENTS FOR DATA POSITIONING, RESULT LENGTH, EXPONENT BASE ADJUSTMENT, AND OTHER VARIANTS OF THE SUB-OPERATION.

EACH SUB-OPERATION, AS MODIFIED BY THE SUB-OPERATION DESCRIPTION, RESULTS IN A SEQUENCE OF MACRO-COMMANDS. EACH MACRO-COMMAND, AS MODIFIED BY THE SUB-OPERATION DESCRIPTION, RESULTS IN A SEQUENCE OF MICRO-COMMANDS. THE MICRO-COMMANDS START IN THE LENGTH SECTION READ REGISTER AND STEP THROUGH THE APPROPRIATE SUB-UNITS.

3.5 MEMORY INTERFACE UNIT (MIU)

THE MIU (SEE FIGURE 37) SHALL PERFORM ALL TRANSFERS BETWEEN THE INTERPRETER AND ANY OF UP TO A MAXIMUM OF 16 MEMORY MODULES. THE MIU SHALL HANDLE ALL DATA TRANSFERS AS FIELD-ORIENTED OPERATIONS AND SHALL MANAGE THE MEMORY ACCESS REQUESTS BY THE FUNCTIONAL ELEMENTS OF THE INTERPRETER ON A PREASSIGNED PRIORITY BASIS. THE ACCESS PRIORITY ASSIGNMENT SHALL BE SPECIFIED BY THE INTERPRETER AND SHALL TYPICALLY INVOLVE THE FOLLOWING ELEMENTS:

A. DISPLAY

B. RESOURCE STACK SLICE

C. NAME STACK BUFFER

D. PROGRAM CONTROL STACK

E. VALUE STACK BUFFER

F. DESCRIPTION BUFFER

G. PROGRAM BUFFER

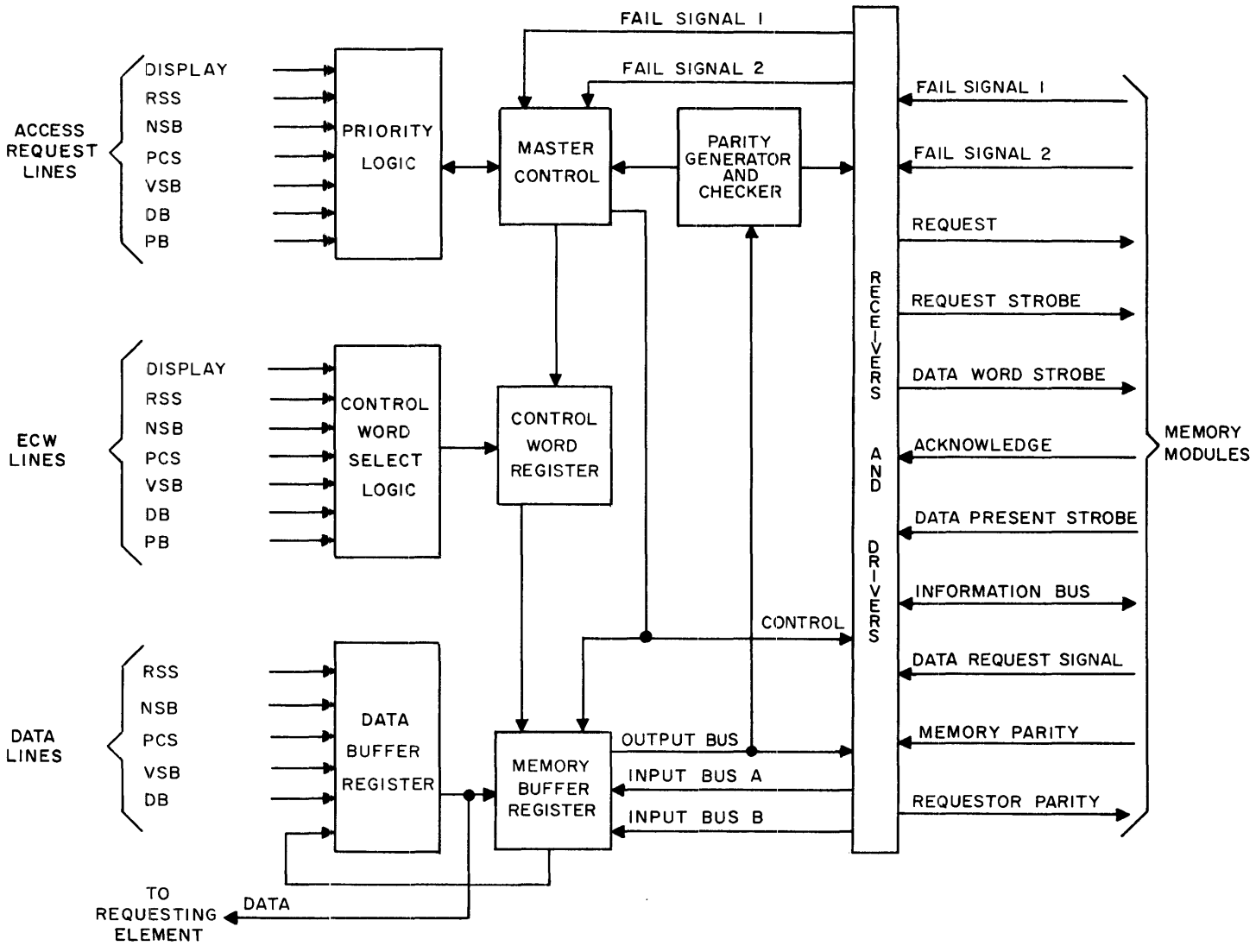


FIGURE 37. MEMORY INTERFACE UNIT (MIU), BLOCK DIAGRAM

3.5.1 FUNCTIONAL OPERATION

WHEN A FUNCTIONAL ELEMENT OF THE INTERPRETER REQUIRES THE SERVICES OF THE MIU, IT SHALL BE REQUIRED TO RAISE ITS "ACCESS REQUEST" LINE TO THE MIU AND PLACE AN ELEMENT CONTROL WORD (ECW) AS DESCRIBED AND ILLUSTRATED IN FIGURE 38, ON ITS ECW LINE TO THE MIU. WHEN THE REQUESTING ELEMENT HAS PRIORITY, THE MIU SHALL LOAD THE ECW INTO ITS CONTROL WORD REGISTER, AND DETERMINE WHICH OF THE FOLLOWING OPERATIONS IS SPECIFIED.

A. A SINGLE WORD (FIELD LENGTH < 64 BITS) STORE OPERATION.

B. A MULTIPLE WORD (FIELD LENGTH > 64 BITS) STORE OPERATION.

C. FETCH OPERATION.

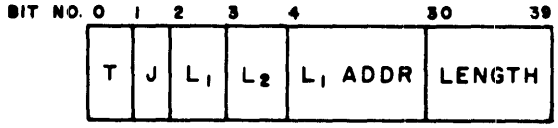


FIGURE 38. ELEMENT CONTROL WORD (ECW) FORMAT

THE VARIOUS FIELDS OF THE UNIT CONTROL (ECW) SHALL BE DEFINED AS FOLLOWS:

TYPE (T) BIT	IDENTIFIES THE SERVICE REQUEST AS A FETCH (T=0) OR STORE (T=1) OPERATION.
JUSTIFICATION (J) BIT	IDENTIFIES THE JUSTIFICATION REQUIRED OF A SINGLE WORD FETCH OR STORE OPERATION. RIGHT JUSTIFICATION (WHERE THE LEAST SIGNIFICANT BIT TRANSFERRED IS PLACED IN LEAST SIGNIFICANT BIT POSITION) SHALL BE INDICATED BY J=1.
LOCK (L) BITS	IDENTIFIES THE TYPE OF FETCH OPERATION TO BE PERFORMED AS FOLLOWS: CODE 00 - FETCH. THIS CODE SHALL CAUSE THE MEMORY TO SEND WHATEVER

FIELD IS SPECIFIED BY THE REQUESTING ELEMENT.

CODE 01 - LOCKOUT FETCH. THIS CODE SHALL CAUSE THE MEMORY TO TRANSFER WHATEVER FIELD IS SPECIFIED BY THE REQUESTING ELEMENT AND SHALL THEN DECLARE THE FIELD LOCKED. A LOCKED FIELD IS DEFINED AS ANY FIELD WHOSE STARTING BIT (IDENTIFIED BY THE L1 ADDRESS FIELD OF THE ECW) IS IN THE ONE STATE.

CODE 11 - ILLEGAL

NOTE

IT SHALL BE THE RESPONSIBILITY OF THE REQUESTING ELEMENT TO KNOW THE STATE OF THE FIELD IT IS REQUESTING.

L1 ADDRESS FIELD	IDENTIFIES THE ABSOLUTE LEVEL-1 STARTING BIT POSITION INVOLVED IN THE TRANSFER.
LENGTH FIELD	SPECIFIES THE TOTAL LENGTH OF THE FIELD BEING TRANSFERRED, IN BITS.

UPON DETERMINING THE TYPE OF OPERATION REQUESTED, THE MIU SHALL CONSTRUCT A MEMORY CONTROL WORD (MCW) AS DESCRIBED AND ILLUSTRATED IN FIGURE 39. THE MIU SHALL THEN BE REQUIRED TO PERFORM ONE OF THE OPERATIONS LISTED BELOW, AS APPLICABLE.

A. IF A SINGLE WORD STORE OPERATION WAS SPECIFIED, THE MIU SHALL RAISE ITS REQUEST LINES TO THE SPECIFIED MEMORY MODULE, AND THEN ALTERNATELY TRANSMIT THE MCW AND THE DATA TO BE STORED TO THE ADDRESSED MEMORY MODULE. THE MIU SHALL CONTINUE TO TRANSMIT THE MCW FOLLOWED BY THE DATA TO BE STORED, UNTIL AN ACKNOWLEDGE SIGNAL IS RECEIVED FROM THE MEMORY MODULE.

B. IF A MULTIPLE WORD STORE OPERATION IS SPECIFIED, THE MIU SHALL RAISE ITS REQUEST LINES TO THE APPLICABLE MEMORY MODULE, AND THEN SEND THE MCW TO THE MEMORY MODULE. WHEN THE MEMORY MODULE ACKNOWLEDGES THE MCW-S PRESENCE, THE MIU COMENCE THE DATA TRANSFER UNDER THE CONTROL OF THE DATA REQUEST SIGNAL.

C. IF A FETCH OPERATION IS SPECIFIED, THE MIU SHALL RAISE ITS REQUEST LINES AND SEND THE MCW TO THE APPLICABLE MEMORY MODULE. WHEN THE MEMORY MODULE ACKNOWLEDGES THE MCW-S PRESENCE, THE MIU

SHALL ENABLE ITS INFORMATION BUS RECEIVER CIRCUITS. INFORMATION FROM THE MEMORY WILL NOW BE ACCEPTED BY THE MIU. HOWEVER, THE MEMORY SHALL BE REQUIRED TO TRANSMIT TO THE MIU A DATA PRESENT STROBE PULSE TO CAUSE THE INFORMATION PRESENT ON THE INFORMATION BUS TO BE TRANSFERRED TO AND DETECTED BY THE REQUESTING ELEMENT. THE DATA PRESENT STROBE PULSE SHALL BE REQUIRED FOR EACH WORD TRANSFERRED FROM MEMORY TO A REQUESTING ELEMENT.

IF EITHER A FETCH OR STORE OPERATION REQUIRES THE INVOVEMENT OF MORE THAN ONE MEMORY MODULE, THE MIU SHALL BE REQUIRED TO CONSTRUCT AN MCW FOR EACH MEMORY MODULE INVOLVED. IN THIS CASE, THE MIU SHALL CONSTRUCT AN UPDATED MCW, AND THEN INITIATE AND CONCLUDE THE DATA TRANSFER WITH THE SECOND MEMORY MODULE. IF THE SIX LEAST SIGNIFICANT BITS OF THE L1 ADDRESS FIELD IN THE ORIGINAL MCW WERE ALL ZEROS THE UPDATED MCW SHALL BE REQUIRED TO HAVE A MODIFIED L1 ADDRESS FIELD WHICH POINTS TO THE FIRST BIT POSITION OF THE NEW MEMORY MODULE AND A NEW LENGTH FIELD WHICH REFLECTS THE NUMBER OF BITS REMAINING TO BE TRANSFERRED. IF THE SIX LEAST SIGNIFICANT BITS OF THE ORIGINAL L1 ADDRESS FIELD WERE NOT EQUAL TO ZERO THE UPDATED MCW SHALL BE REQUIRED TO HAVE ITS LINK (L) BIT SET; A MODIFIED L1 ADDRESS FIELD WHOSE SIX LEAST SIGNIFICANT DIGITS ARE IDENTIFICAL TO THOSE IN THE ORIGINAL MCW, BITS 18 THRU 33 SHALL BE ALL ONES, AND BITS 14 THRU 17 SHALL REFLECT THE NEW MEMORY MODULE NUMBERS; AND A MODIFIED LENGTH FIELD WHICH SHALL REFLECT THE NUMBER OF WORDS REMAINING TO BE TRANSFERRED PLUS ONE, WHICH IS REQUIRED TO REFLECT THE LINK OPERATION REQUIRED OF THE MEMORY.

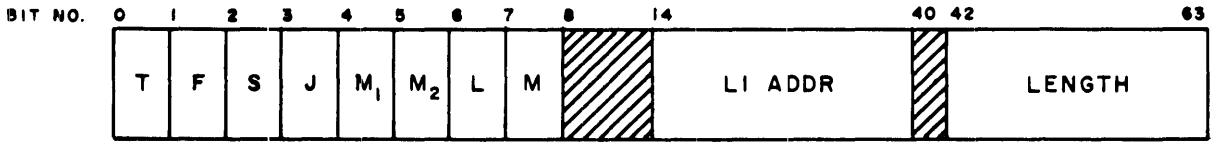


FIGURE 39. MEMORY CONTROL WORD (MCW) FORMAT

THE VARIOUS FIELDS OF THE MEMORY CONTROL WORD (MCW) SHALL BE DEFINED AS FOLLOWS:

TYPE (T) BIT	SAME AS DEFINED FOR ECW.
SPECIFIER (S) BIT	IDENTIFIES A STORE OPERATION AS EITHER A SINGLE WORD STORE (S=1) OR MULTIPLE WORD STORE (S=0) OPERATION. ALSO IDENTIFIES A FETCH OPERATION WHICH IS REQUESTING THAT THE MEMORY FATL REGISTER BE READ AND THEN CLEARED (S=1).

JUSTIFICATION (J) BIT	SAME AS DEFINED FOR ECW.
MODIFY BITS M1 AND M2	SAME AS DEFINED FOR LOCK (L) BITS IN ECW.
LINK (L) BIT	WHEN PRESENT, INDICATES THAT THE FIELD BEING TRANSFERRED IS CONTAINED IN MORE THAN ONE MEMORY MODULE AND THAT ITS STARTING MEMORY ADDRESS WAS NOT THE BEGINNING OF A MEMORY WORD BOUNDARY (ZERO OR A MULTIPLE OF 64). THIS BIT IS REQUIRED TO BE IN THE TRUE STATE ONLY WHEN FETCHING OR STORING A FIELD ACROSS A MEMORY BOUNDARY AND MORE THAN ONE MEMORY MODULE IS INVOLVED IN THE TRANSFER. WHEN THIS SITUATION ARISES, THE LINK BIT MUST BE IN THE TRUE STATE WHEN THE UPDATED MCW IS SENT TO THE SECOND MEMORY MODULE.
MODE (M) BIT	INDICATES WHEN PRESENT, THAT THE MEMORY SHALL BE OPERATED IN A DEFINED PATTERN (E.G., ONE WORD EVERY TWO CLOCKS) AS CONTROLLED BY THE MEMORY.
L1 ADDRESS FIELD	SAME AS DEFINED FOR ECW.
LENGTH FIELD	SAME AS DEFINED FOR ECW. THE MIU SHALL ALSO BE REQUIRED TO DETECT AND REPORT TO THE INTERPRETER, MIU-DETECTED ERRORS AND MEMORY-DETECTED ERRORS.

MIU-DETECTED ERRORS SHALL BE AS FOLLOWS:

A. NO ACCESS TO MEMORY - THIS ERROR CONDITION SHALL BE DECLARED IF THE MIU RECEIVES NO RESPONSE FROM THE REQUESTED MEMORY MODULE FOR A PERIOD OF 25 MICROSECONDS. NO RESPONSE FROM MEMORY SHALL BE DECLARED IF AN ACKNOWLEDGE SIGNAL IS NOT RECEIVED FROM A MEMORY MODULE OR WHEN COMPLETE DATA IS NOT TRANSFERRED BY A MEMORY MODULE.

B. DISPARITY - THIS ERROR CONDITION SHALL BE DECLARED IF A FETCH OF DATA FROM MEMORY IS RECEIVED BY THE MIU WITH INCORRECT PARITY, OR IF A DATA TRANSFER FROM THE INTERPRETER IS RECEIVED BY THE MIU WITH INCORRECT PARITY. IF A NO ACCESS TO MEMORY OR PARITY ERROR IS DETECTED, THE PROCESSOR ERROR REGISTER SHALL BE LOADED AS DESCRIBED IN PARAGRAPH 3.5.2(I).

THERE SHALL BE TWO CLASSIFICATIONS OF MEMORY-DETECTED ERRORS WHICH WILL BE REPORTED TO THE MIU: UNCORRECTABLE AND CORRECTABLE. THESE TWO TYPES OF ERRORS SHALL BE REPORTED TO THE MIU AS FAIL SIGNAL 1 AND FAIL SIGNAL 2, RESPECTIVELY; HOWEVER, THE MIU SHALL SEND ONLY ONE FAIL SIGNAL TO THE INTERPRETER. THE FOLLOWING OPERATIONS WILL TAKE PLACE WHEN THE MEMORY SENDS A FAIL SIGNAL TO THE MIU.

A. FAIL SIGNAL 1 (UNCORRECTABLE ERROR CONDITION) - IF THIS TYPE OF ERROR CONDITION IS REPORTED BY A MEMORY MODULE WHILE AN MIU OPERATION IS IN PROGRESS, THE MIU OPERATION SHALL BE TERMINATED AND THE INTERPRETER NOTIFIED OF THIS ACTION. IF THE ERROR IS REPORTED DURING A TIME WHEN AN MIU OPERATION IS NOT IN PROGRESS WITH THE REPORTING MEMORY MODULE, THE MIU SHALL RECORD THE FAILURE BUT IT WILL COMPLETE THE CURRENT OPERATION.

B. FAIL SIGNAL 2 (CORRECTABLE ERROR CONDITION) - THIS TYPE OF ERROR SIGNAL SHALL CAUSE THE MIU TO NOTIFY THE INTERPRETER OF THE CONDITION, AND OPERATION WILL PROCEED AS USUAL.

3.5.2 FUNCTIONAL COMPONENTS

THE MIU SHALL CONSIST OF NINE FUNCTIONAL COMPONENTS INTERFACED AS ILLUSTRATED IN FIGURE 27, AND OPERATED AS DESCRIBED BELOW.

A. PRIORITY LOGIC - THIS SECTION SHALL BE RESPONSIBLE FOR GRANTING THE SERVICES OF THE MIU TO THE HIGHEST PRIORITY REQUESTING ELEMENT.

B. CONTROL WORD SELECT LOGIC - THIS SECTION SHALL BE RESPONSIBLE FOR ROUTING THE ECW OF THE REQUESTING ELEMENT TO THE CONTROL WORD REGISTER, AS DIRECTED BY THE PRIORITY LOGIC.

C. CONTROL WORD REGISTER - THIS SHALL BE A 64-BIT REGISTER, AND WILL BE USED TO STORE THE ECW DURING ITS EXECUTION AND UPDATING BY THE MASTER CONTROL SECTION.

D. MASTER CONTROL - THIS SECTION SHALL CONTAIN THE CONTROL LOGIC NECESSARY TO EXECUTE ALL MIU OPERATIONS, INCLUDING THE CONTROLS REQUIRED TO COMPLETE RECEIVER AND DRIVER PATHS.

E. MEMORY BUFFER REGISTER - THIS SHALL BE A 64-BIT REGISTER, AND WILL BE USED TO BUFFER ALL INPUT AND OUTPUT DATA TO AND FROM THE MEMORY VIA THE INFORMATION BUS.

F. DATA BUFFER REGISTER - THIS SHALL BE A 64-BIT REGISTER, AND WILL BE USED TO BUFFER ALL DATA TRANSFERS BETWEEN THE REQUESTING ELEMENT OF THE INTERPRETER AND THE MIU. THIS REGISTER SHALL ALSO BE USED FOR LINK TRANSFER OPERATIONS WHICH NECESSITATE THE COMBINING OF DATA FIELDS.

G. PARITY GENERATOR AND CHECKER - THIS SECTION SHALL BE REQUIRED TO GENERATE PARITY FOR ALL WORDS BEING TRANSFERRED TO MEMORY, AND CHECK THE PARITY OF WORDS BEING FETCHED FROM MEMORY.

H. RECEIVERS AND DRIVERS - THERE SHALL BE SIXTEEN DISCRETE GROUPS OF RECEIVER AND DRIVER CIRCUITS IN THE MIU ONE GROUP PER MEMORY MODULE INTERFACE. THE STATE OF THESE GROUPS SHALL BE DETERMINED BY MASTER CONTROL, AND ONLY ONE GROUP SHALL BE ACTIVE AT ANY ONE TIME.

I. PROCESSOR ERROR REGISTER (PER) SHALL BE A 64-BIT REGISTER AND WILL BE USED TO FACILITATE RECOVERY FROM ERROR CONDITIONS INVOLVING LEVEL-1 REFERENCES BY CAPTURING, FOR SUBSEQUENT ANALYSIS, ALL AVAILABLE CONTROL INFORMATION RELATING TO THE REFERENCE CAUSING THE INTERRUPT. THE PER CAN BE PROGRAMMATICALLY BROUGHT TO THE TOP OF THE VALUE STACK ONCE THE PER IS LOADED WITH ERROR INFORMATION, IT CANNOT BE LOADED AGAIN UNTIL IT IS CLEARED; CLEARING PER IS DONE BY FETCHING IT. THE PER IS NEVER LOADED UNLESS AN ACTUAL INTERRUPT IS GOING TO OCCUR

4.0 PHYSICAL CHARACTERISTICS

(TO BE SUPPLIED)

5.0 POWER REQUIREMENTS

A 120/208 VAC PLUS OR MINUS 10%, THREE-PHASE, FOUR-WIRE, 60-CPS PLUS OR MINUS 5% SOURCE CAPABLE OF SUPPLYING 20.1 KVA IS REQUIRED BY THE CPM. THE FOLLOWING DC POWER IS INTERNALLY SUPPLIED BY THE CPM.

NOMINAL VOLTAGE	MAXIMUM CURRENT REQUIRED (CALCULATED)
+4.8 VOLTS	1,105 AMPS
-2.0 VOLTS	585 AMPS
-6.0 VOLTS	65 AMPS

6.0 OPERATING ENVIRONMENT REQUIREMENTS

THE CPM SHALL PERFORM ITS INTENDED FUNCTIONS AND SATISFY THE RELIABILITY AND MAINTAINABILITY REQUIREMENTS SPECIFIED HEREIN, WHEN OPERATED UNDER THE FOLLOWING ENVIRONMENTAL CONDITIONS:

	MINIMUM	MAXIMUM	RATE OF CHANGE
AMBIENT TEMPERATURE (DEGREES F)	65	80	15 DEGREES PER 0.25 HR. MAX
RELATIVE HUMIDITY		%	90%
ALTITUDE	---	10,000 FT	---

UNDER THESE CONDITIONS, THE CPM WILL DISSIPATE APPROXIMATELY 39,800 BTU(S) OF HEAT PER HOUR.

7.0 CABLING REQUIREMENTS

(TO BE SUPPLIED)

8.0 CONTROL PANEL REQUIREMENTS

THE CONTROLS AND INDICATORS NEEDED TO MANUALLY OPERATE AND MONITOR THE CPM WILL BE CONTAINED ON AND BE PART OF THE SYSTEM MAINTENANCE DIAGNOSTIC PROCESSOR.

9.0 QUALITY ASSURANCE PROVISIONS

9.1 RELIABILITY AND MAINTAINABILITY

9.1.1 MEAN TIME BETWEEN FAILURES (MTBF)

THE MTBF FOR THE CPM AS CALCULATED USING THE STANDARD COMPONENT FAILURE RATES SPECIFIED IN MIL-HDBK-217A AS APPLICABLE TO THE CPU PARTS APPLICATION, IS 486 HOURS.

9.1.2 MEAN TIME TO REPAIR (MTTR)

THE ESTIMATE MTTR FOR THE CPM BASED ON USING THE MAINTENANCE DIAGNOSITC PROCESSOR TO LOCATE A SINGLE FAULTY COMPONENT OR INTEGRATED CIRCUIT MODULE, AND THE SUBSEQUENT REPLACEMENT OF THE FAULTY COMPONENT, IS 1.0 HOUR.

9.2 QUALITY CONTROL

9.2.1 INSPECTION RESPONSIBILITY

THE SUPPLIER IS RESPONSIBLE FOR THE PERFORMANCE OF ALL INSPECTIONS AND TESTS SPECIFIED HEREIN. EXCEPT AS NOTED IN THE CONTRACT OR ORDER, THE SUPPLIER MAY USE HIS OWN OR ANY OTHER INSPECTION FACILITIES AND SERVICES ACCEPTABLE TO THE PROCURING ACTIVITY. INSPECTION RECORDS OF EXAMINATIONS AND TESTS SHALL BE COMPLETE AND AVAILABLE TO THE PROCURING ACTIVITY. THE PROCURING ACTIVITY RESERVES THE RIGHT TO PERFORM ANY OF THE INSPECTIONS AND TESTS SPECIFIED HEREIN WHEN SUCH INSPECTIONS ARE DEEMED NECESSARY TO ASSURE THAT THE PRODUCTS AND SERVICES CONFORM TO THE PRESCRIBED REQUIREMENTS.

9.2.2 CLASSIFICATION OF TESTS

THESE SHALL BE CLASSIFIED AS QUALITY CONFORMANCE TESTS.

9.2.3 TEST CONDITIONS

9.2.3.1 ENVIRONMENTAL

(TEMPERATURE, PRESSURE, AND HUMIDITY FOR TESTING - TO BE SUPPLIED)

9.2.3.2 TEST POWER

(TO BE SUPPLIED)

9.2.3.3 TEST EQUIPMENT

(STANDARD AND SPECIAL - TO BE SUPPLIED)

9.2.4 TEST PROCEDURES(S)

(REFERENCE(S) TO BE SUPPLIED)

9.2.5 TEST METHODS

(TO BE SUPPLIED)

10.0 PREPARATION FOR DELIVERY

10.1 PRESERVATION, PACKAGING AND PACKING

THE EQUIPMENT SHALL BE PRESERVED AND PACKAGED ACCORDING TO THE SUPPLIER(S) BEST COMMERCIAL PRACTICE FOR SAFE DELIVERY BY COMMON CARRIER.

APPENDIX A

STRUCTURE EXPRESSION EVALUATION ALGORITHMS

APPENDIX A REPRESENTS THE ALGORITHMS USED BY THE INTERPRETER FOR THE EVALUATION OF DESCRIPTOR STRUCTURE EXPRESSIONS. ALSO INCLUDED ARE ILLUSTRATIONS OF THE ATTRIBUTE STACK AND THE STRUCTURE EXPRESSION BEFORE, DURING, AND AFTER EVALUATION.

STANDARD MNEMONICS USED IN THE ALGORITHMS ARE DEFINED IN TABLE A-1. IN THE ALGORITHMS A NUMERICAL SUFFIX USED ON THE RECEIVING SIDE OF AN EXPRESSION DENOTES A WRITE OVER. THE VALUE OF THE SUFFIX INDICATES DEPTH INTO A STACK. THE ABSENCE OF A SUFFIX IMPLIES A PUSH OR A POP OF THE STACK.

STANDARD MICRO-OPERATORS USED IN THE ALGORITHMS ARE DEFINED IN TABLE A-2. THE DEFINITIONS ARE IN BACKUS NAUER FORM.

IN THE ILLUSTRATIONS OF THE ATTRIBUTE STACK, A NUMBER TO THE LEFT INDICATES THAT THE CORRESPONDINGLY NUMBERED MICRO-OPERATOR CAUSED AN ENTRY TO THE ILLUSTRATION ON THAT LINE. A NUMBER ON THE RIGHT INDICATES A DELETION.

IN THE ILLUSTRATIONS OF THE STRUCTURE EXPRESSION A NUMBER INDICATES A CHANGE CAUSED BY THE CORRESPONDINGLY NUMBERED MICRO-OPERATOR. ALL EVALUATIONS RESULT IN A VALUE IN EACH OF THE ATTRIBUTE CONTAINER FIELDS. CONSEQUENTLY ALL ATTRIBUTE STACK ILLUSTRATIONS BEGIN WITH ONE VALUE IN EACH OF THE TWO CONTAINER STACKS.

EVALUATION OF ALL STRUCTURE EXPRESSIONS EXCEPT THE LAST IS DONE WITH THE CONSTURCT OPERATOR. STURCTURE EXPRESSIONS PERTAINING TO DATA STRUCTURES WHICH HAVE NO PROVISION FOR ALLOCATION OR DEALLOCATION OF SPACE ARE ALWAYS EVALUATED BY THE CONSTRUCT OPERATOR.

WHEN THE STRUCTURE EXPRESSION IS OF TYPE SEGMENT NUMBER, THE NUMBER FIELD IS USED AS AN INDEX INTO THE RESOURCE STACK SLICE TO LOCATE THE APPROPRIATE STORAGE LEVEL CONTAINER OR DEVICE.

A STRUCTURE EXPRESSION OF TYPE FIN INDICATES THE END OF A DESCRIPTOR STRUCTURE EXPRESSION STRING. IT RESULTS IN A TRANSFER OF THE CONTAINER FIELDS FROM THE ATTRIBUTE STACK TO THE TOP OF THE NAME STACK.

PORTIONS OF THE ALGORITHMS ARE WRITTEN IN PSEUDO-ALGOL FOR CLARITY.

TABLE A-I STANDARD MNEMONICS

AS - ATTRIBUTE STACK

CSPR - CONSTRUCT PROGRAM REGISTER

DPCR - DESCRIPTOR PROGRAM COUNT REGISTER

k - THE SIZE (IN BITS) OF A PROGRAM FIELD USED IN A SELF IDENTIFYING STRUCTURE

SEP - STRUCTURE EXPRESSION PARAMETERS

SET - STRUCTURE EXPRESSION TYPE

VSS - VALUE STACK SLICE (TOP ELEMENT OF THE VALUE STACK)

X - ARBITRARY SYMBOL OR SYMBOLS

[ ] - CONTENTS OF THE OBJECT NAMED INSIDE

< > - EXPLANATION OF A PORTION OF THE MICRO-SEQUENCE

. - DENOTES FIELD SPECIFIER STRING FOLLOWS

a_c - CONTAINER ADDRESS FIELD

lc - CONTAINER LENGTH FIELD

a_e - ELEMENT ADDRESS FIELD

le - ELEMENT LENGTH FIELD

THE REMAINING FIELDS ARE DEFINED BY THE DATA STRUCTURE DIAGRAMS

TABLE A-2 STANDARD MICRO-OPERATORS

<CALL> :: = <DPCR.a_c, lc ← ASO.a_c, lc> <DELETE AS.a_c, lc >

<COMBINE> :: = <ASO.a_c ← ASO.a_c + ASO.a_e > <ASO.lc ← ASO.le
<DELETE AS.a_e, le> <BOUNDS CHECK>

<DELETE AS.X> :: = <AS.X ← EMPTY>

<DUPLICATE AS.X> :: = < AS.X ← ASO.X>

<EVALUATE> :: = <LOCATE AN ELEMENT OF THE NAME STACK VIA THE DISPLAY>

<FAULT (X)> :: = <SET BIT X IN THE PROCESSOR INTERRUPT SECTION>

<FETCH TO X> :: = < X ← CONTENTS (AS . a_c , lc)> < DELETE AS . a_c , lc >

<MOVE (X)> :: = <CONTAINER (ASO.a_c, lc) ← [X] > <DELETE AS.a_c, lc >

<NEGATIVE SEQUENCE > :: = < ASO.a_e ← ASO.a_e - ASO.le > <SEP.X ← ASO.a_e>

<NEGATIVE SEQUENCE AS.a_e> :: = <ASO.a_e ← ASO.a_e - ASO.le>

<POSITIVE MOD SEQUENCE SEP.X > :: = <SEP.X ← (ASO.a_e + ASO.le) MOD ASO.lc >

<POSITIVE SEQUENCE > :: = < ASO.a_e ← ASO.a_e + ASO.le> <SEP.X ← ASO.a_e + ASO.le >

<POSITIVE SEQUENCE ASO.a_e > :: = < ASO.a_e ← ASO.a_e + ASO.le>

<POSITIVE SEQUENCE SEP.X > :: = < SEP.X ← ASO.a_e + ASO.le >

<R> :: = < END OF MICRO-OPERATOR STRING>

<REMOVE> :: = (SEE APPENDIX B)

<RET> :: = <DPCRO.a_c, lc ← EMPTY > <POP DPCR STACK >

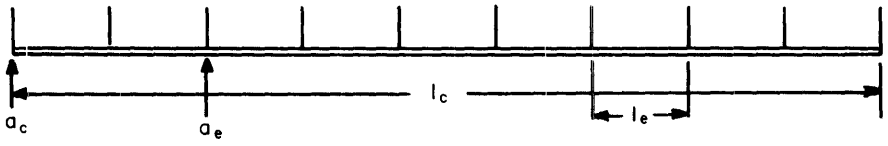
<RTS> :: = < AS.a_c, lc ← DPCRO.a_c, lc > <RET>

<SELECT> :: = <ASO.a_e ← ASO.a_e * ASO.lc

< STORE FROM X > :: = < CONTENTS (AS.a , lc) ← CONTENTS (X) > < DELETE AS.a , lc >

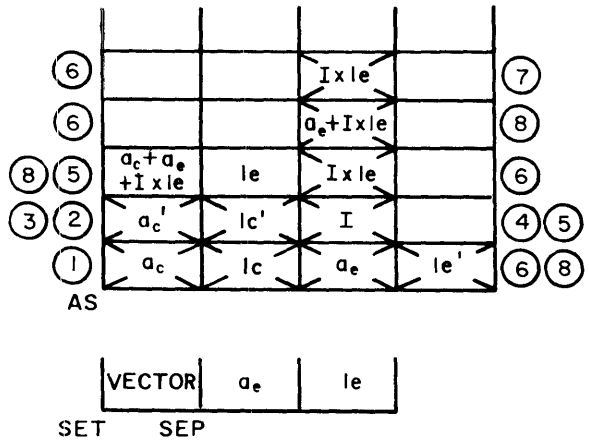
< UPDATE X > :: = < RETURN LOCAL COPY OF X TO ITS LEVEL ONE LOCATION >

VECTOR

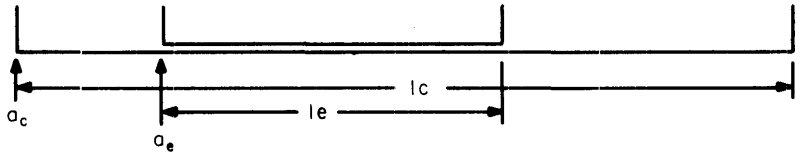


CONSTRUCT

1. AS. a_e, le ← SEP. a_e, le
2. AS. a_e ← [VSS] <INDEX I>
3. REMOVE (VSS) <NOT A MICRO-OPERATOR>
4. DELETE AS. a_c, lc
5. SELECT
6. ASI. a_e ← ASI. a_e + ASO. ae
7. DELETE ASO. a_e
8. COMBINE
9. R

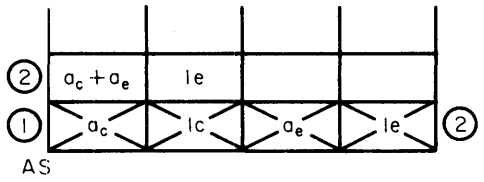


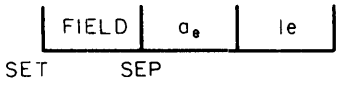
FIELD



CONSTRUCT

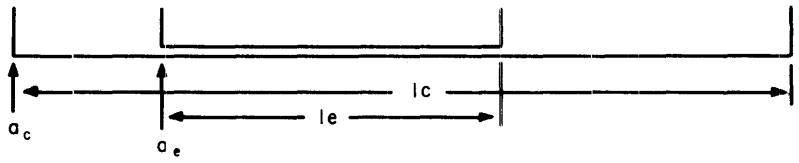
1. AS.a_e, le ← SEP.a_e, le
2. COMBINE
3. R





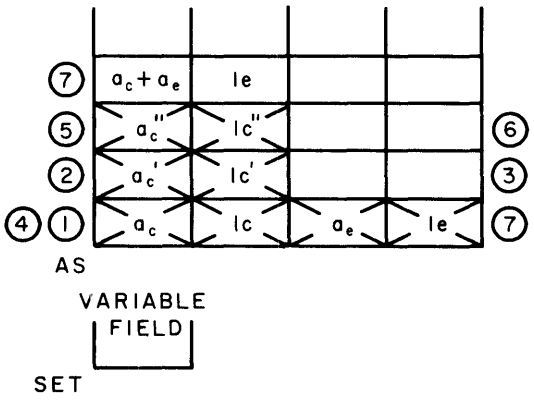
CP 1720-5592

VARIABLE FIELD

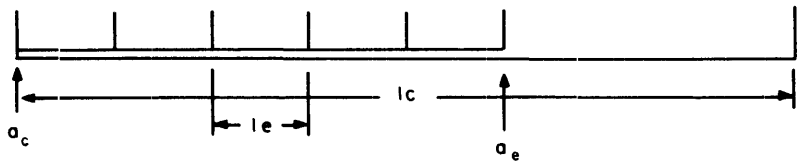


CONSTRUCT

1. AS.a_e ← [VSS] < PARAMETER a_e >
2. REMOVE (VSS) < NOT A MICRO-OPERATOR >
3. DELETE AS.a_c, lc
4. AS.le ← [VSS] < PARAMETER le >
5. REMOVE (VSS) < NOT A MICRO-OPERATOR >
6. DELETE AS.a_c, lc
7. COMBINE
8. R

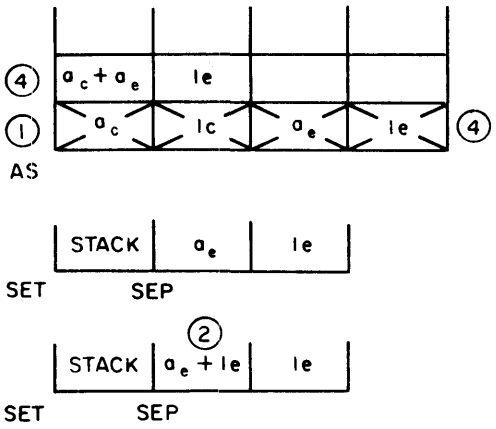


STACK



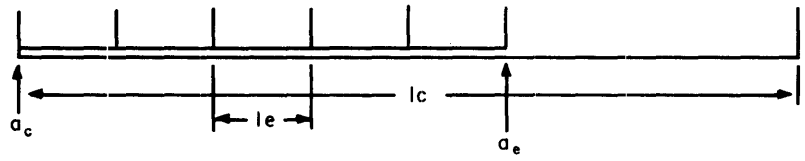
ENTER

1. AS. a_e, le ← SEP. a_e, le
2. POSITIVE SEQUENCE SEP. a_e
3. UPDATE SEP. a_e
4. COMBINE
5. R



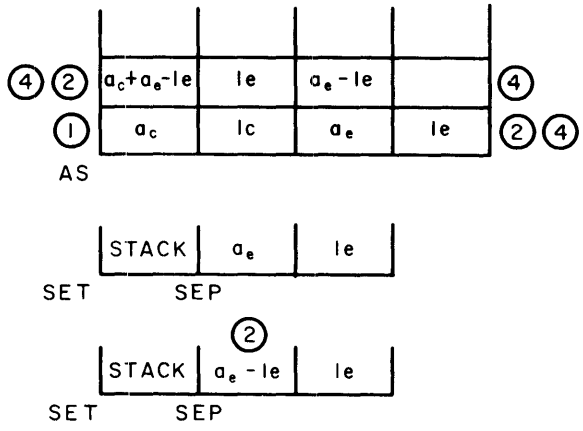
CP 1720-5592

STACK

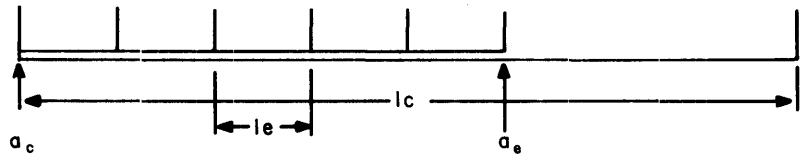


REMOVE

1. AS.a_e, le ← SEP.a_e, le
2. NEGATIVE SEQUENCE
3. UPDATE SEP.a_e
4. COMBINE
5. R

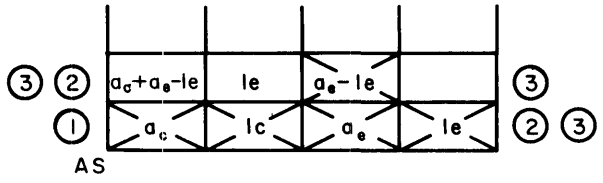


STACK

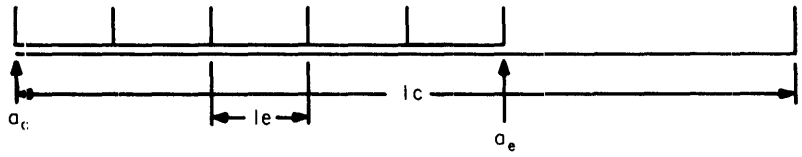


CONSTRUCT

1. AS.a_e, le ← SEP.a_e, le
2. NEGATIVE SEQUENCE AS.a_e
3. COMBINE
4. R

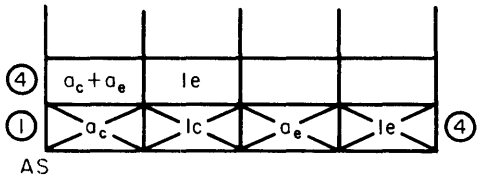


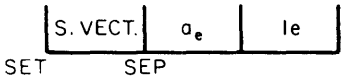
STACK-VECTOR

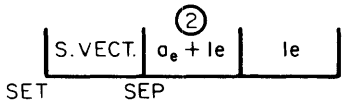


ENTER

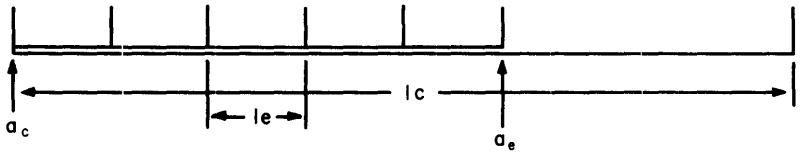
1. AS. a_e, le ← SEP. a_e, le
2. POSITIVE SEQUENCE SEP. a_e
3. UPDATE SEP. a_e
4. COMBINE
5. R





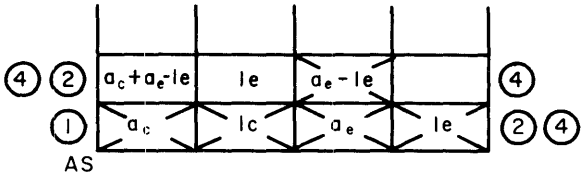


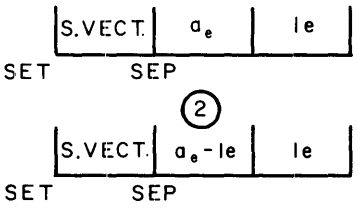
STACK - VECTOR



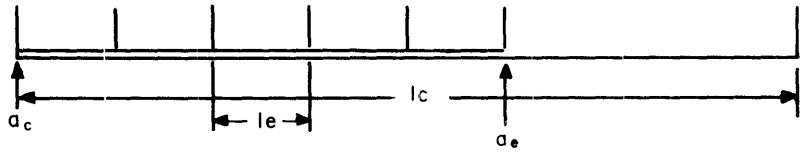
REMOVE

1. AS. a_e , le ← SEP. a_e , le
2. NEGATIVE SEQUENCE
3. UPDATE SEP. a_e
4. COMBINE
5. R



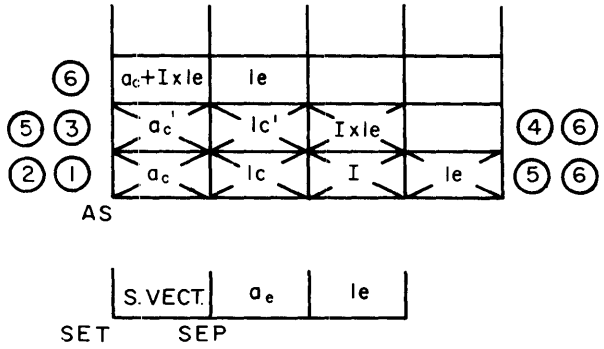


STACK-VECTOR

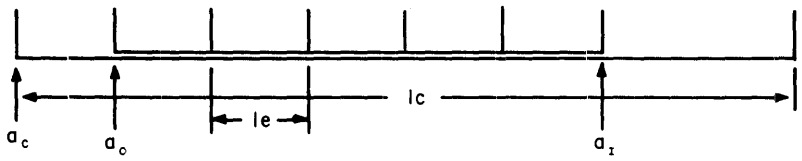


CONSTRUCT

1. AS. le ⟵ SEP. le
2. AS. a_e ⟵ [VSS] <INDEX I>
3. REMOVE (VSS) <NOT A MICRO-OPERATOR>
4. DELETE AS. a_c, lc
5. SELECT
6. COMBINE
7. R

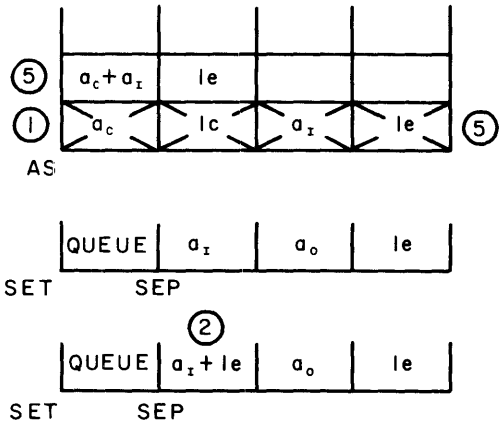


QUEUE

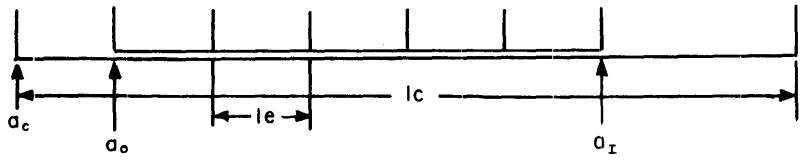


ENTER

1. AS. a_e, le ← SEP. a_I, le
2. POSITIVE MOD SEQUENCE SEP. a_I
3. IF SEP. a_I = SEP. a_o THEN FAULT (FULL)
4. UPDATE SEP. a_I
5. COMBINE
6. R

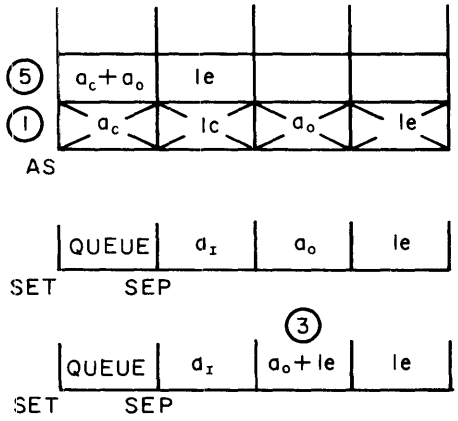


QUEUE

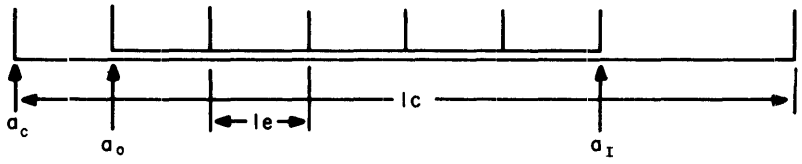


REMOVE

1. AS. a_c, le ⟵ SEP. a_o, le
2. IF SEP. a_o = SEP. a_I THEN FAULT (EMPTY)
3. POSITIVE MOD SEQUENCE SEP. a_o
4. UPDATE SEP. a_o
5. COMBINE
6. R

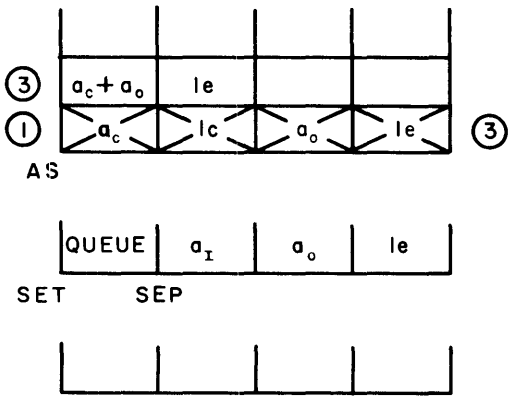


QUEUE

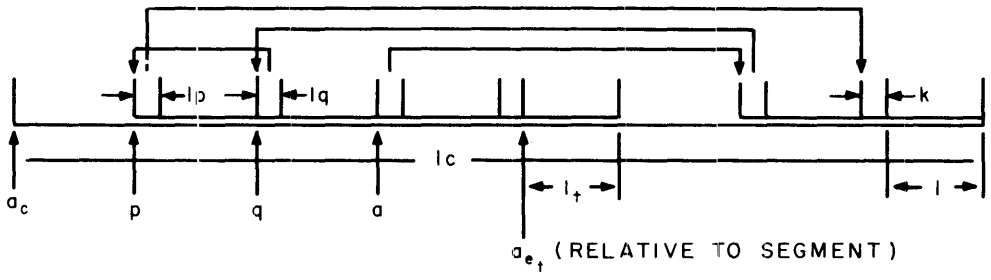


CONSTRUCT

1. AS.a_e,le ← SEP.a_o,le
2. IF SEP.a_o = SEP.a_I THEN FAULT (EMPTY)
3. COMBINE
4. R

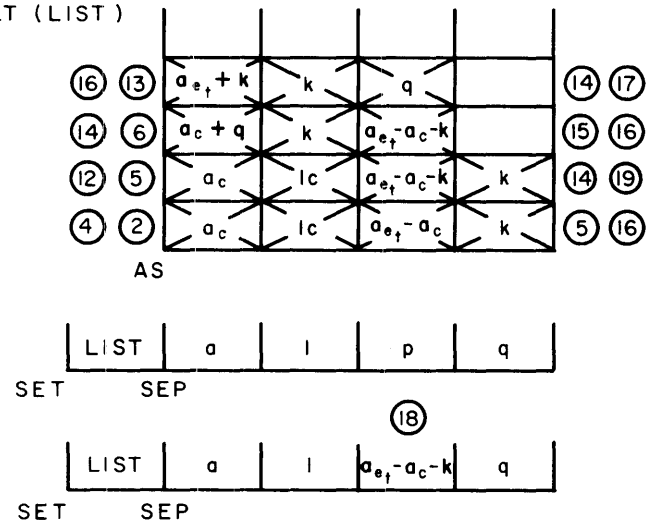


LINKED LIST

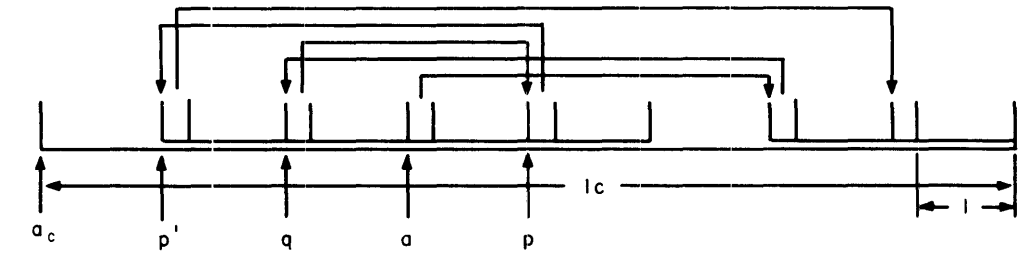


ENTER

```
1.  IF (NSO.SEG # ≠ ACS.SEG #) OR
    (NSO.l_et ≠ SEP.l) THEN FAULT (LIST)
2.  AS.a_e ← NSO.a_et − AS.a_c
3.  IF AS.a_e + SEP.l > AS.lc THEN FAULT (LIST)
4.  AS.le ← k
5.  NEGATIVE SEQUENCE AS.a_e
6.  DUPLICATE AS.a_e
7.  IF (SEP.a = - ) OR (SEP.q = -1) THEN
      BEGIN
8.      COMBINE
9.      STORE FROM SEP.a
10.     SEP.a ← AS.a_e
11.     SEP.q ← -1
      END
    ELSE
      BEGIN
12.     DUPLICATE AS.a_c, lc, le
13.     AS.a_e ← SEP.q
14.     COMBINE
15.     STORE FROM AS.a_e
16.     COMBINE
17.     STORE FROM SEP.p
      END
18. SEP.p ← AS.a_e
19. DELETE AS.a_e
20. R
```

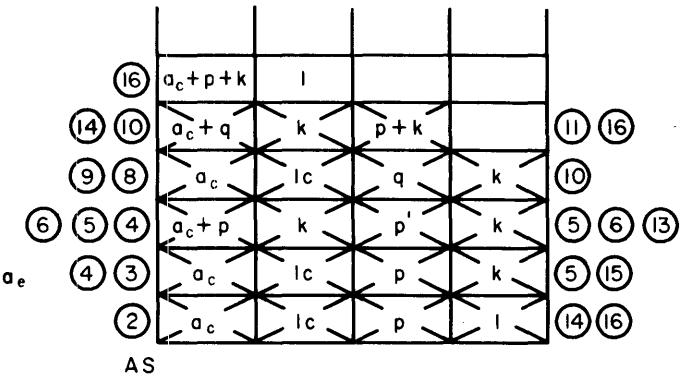


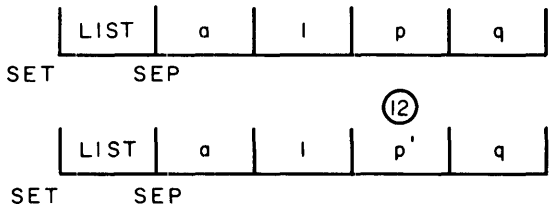
LINKED LIST



REMOVE

1. IF (SEP.a = -1) OR (SEP.p = -1)
   THEN FAULT (LIST)
2. AS.a_e,le ⟵ SEP.p,1
3. AS.le ⟵ k
4. DUPLICATE AS.a_c,lc,a_e,le
5. COMBINE
6. FETCH TO AS.a_e
7. IF SEP.p = SEP.a THEN SEP.a ⟵ AS.a_e
   ELSE
   BEGIN
8. DUPLICATE AS.a_c,lc,le
9. AS.a ⟵ SEP.q
10. COMBINE
11. STORE FROM AS.a_e
    END
12. SEP.p ⟵ AS.a_e
13. DELETE AS.a_e
14. POSITIVE SEQUENCE AS.a_e
15. DELETE AS.le
16. COMBINE
17. R



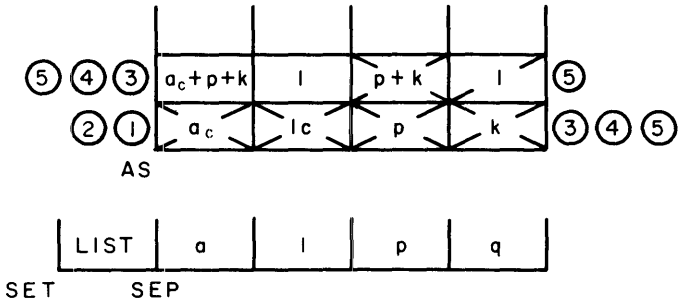


CP 1720-5592

LINKED LIST

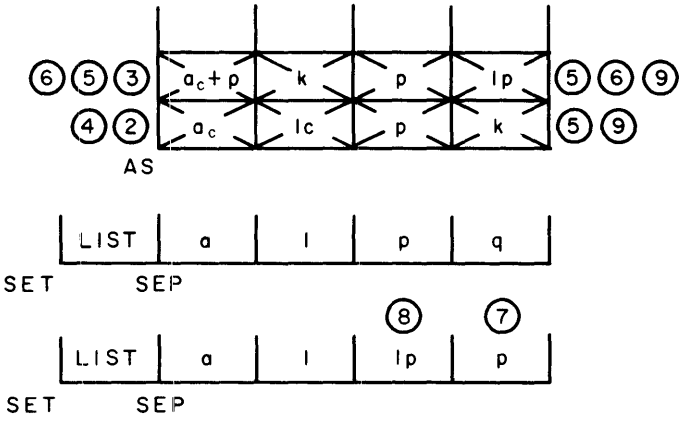
CONSTRUCT

1. AS.a ← SEP.p
2. AS.le ← k
3. POSITIVE SEQUENCE AS.a_e
4. ASO.le ← SEP.l
5. COMBINE
6. R



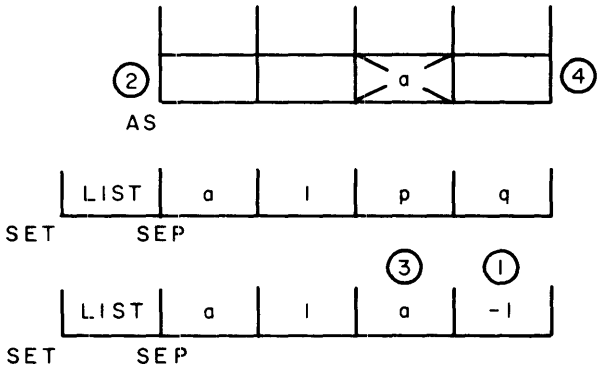
SEQUENCE

1. IF (SEP.a = -1) OR (SEP.p = -1) THEN FAULT (LIST)
2. AS.a_e ← SEP.p
3. DUPLICATE AS.a_e
4. AS.le ← k
5. COMBINE
6. FETCH TO AS.le
7. SEP.q ← AS.a_e
8. SEP.p ← AS.le
9. DELETE AS.a_e, le
10. R

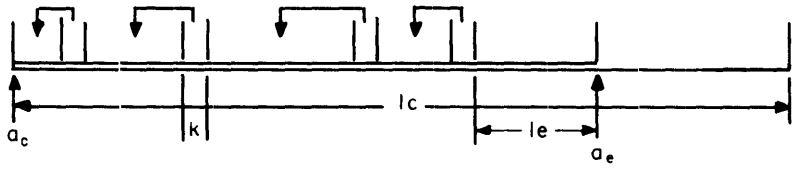


RESET

1. SEP.q ← -1
2. AS.a_e ← SEP.a
3. SEP.p ← AS.a_e
4. DELETE AS.a_e

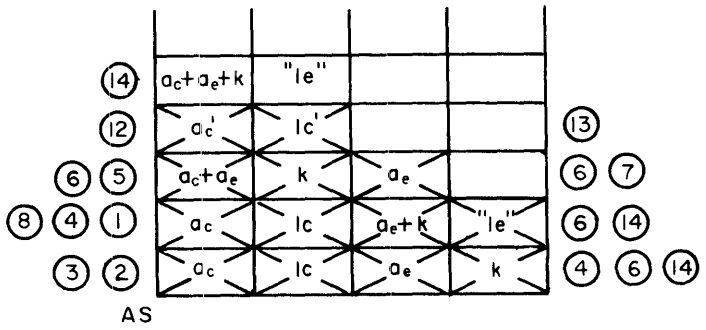


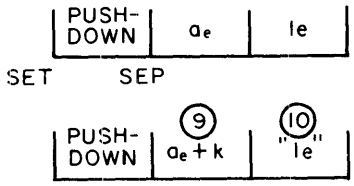
PUSHDOWN



ENTER

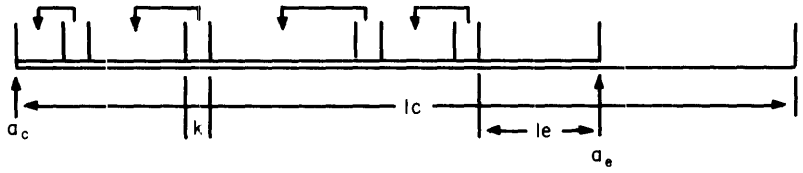
1. DUPLICATE AS. a_c, lc
2. AS. a_e ← SEP. a_e
3. AS. le ← k
4. POSITIVE SEQUENCE AS. a_e
5. AS. a_e ← SEP. a_e
6. COMBINE
7. STORE FROM SEP. le
8. AS. le ← [VSS] <PARAMETER "le">
9. POSITIVE SEQUENCE SEP. a_e
10. SEP. le ← AS. le
11. UPDATE SEP. a_e, l_e
12. REMOVE (VSS) <NOT A MICRO-OPERATOR>
13. DELETE AS. a_c, lc
14. COMBINE
15. R





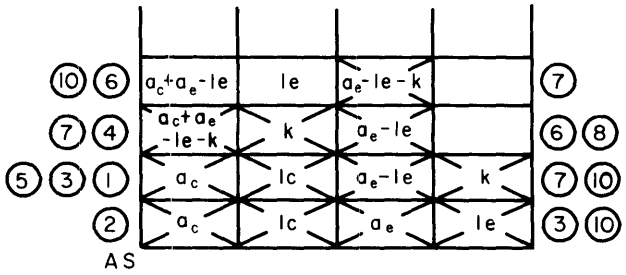
CP 1720-5592

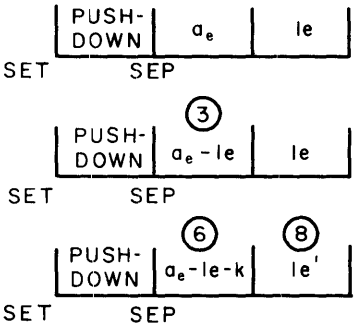
PUSHDOWN



REMOVE

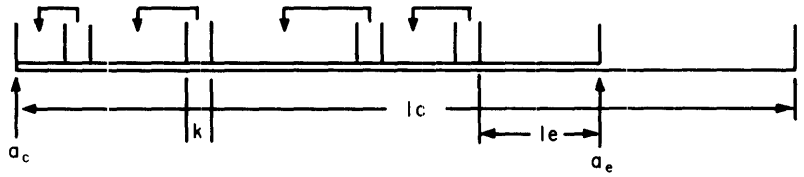
1. DUPLICATE AS. a_c, lc
2. AS. a_e, le ← SEP. a_e, le
3. NEGATIVE SEQUENCE
4. DUPLICATE AS. a_e
5. AS. le ← k
6. NEGATIVE SEQUENCE
7. COMBINE
8. FETCH TO SEP. le
9. UPDATE SEP. a_e, le
10. COMBINE
11. R





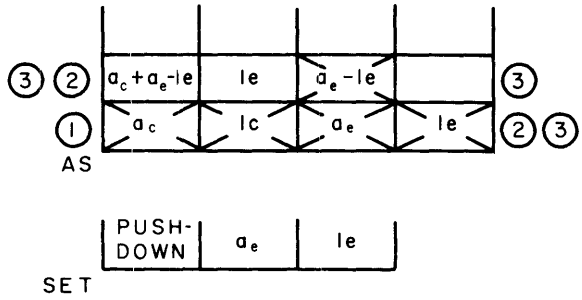
CP 1720-5592

PUSHDOWN

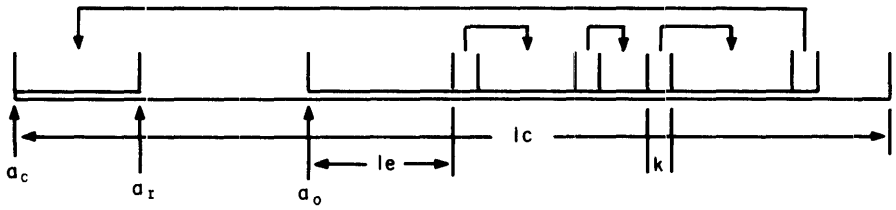


CONSTRUCT

1. AS. a_e, le ← SEP. a_e, le
2. NEGATIVE SEQUENCE AS. a_e
3. COMBINE
4. R

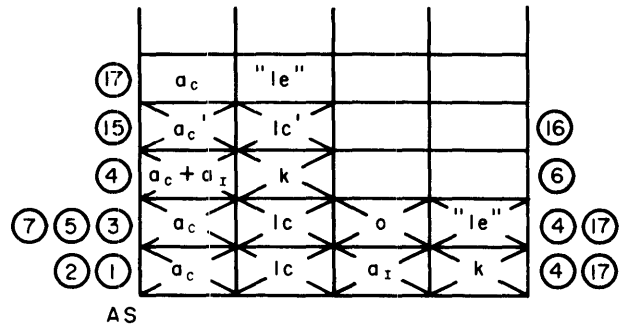


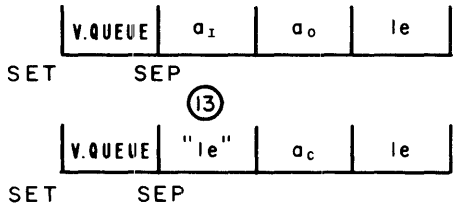
VARIABLE QUEUE

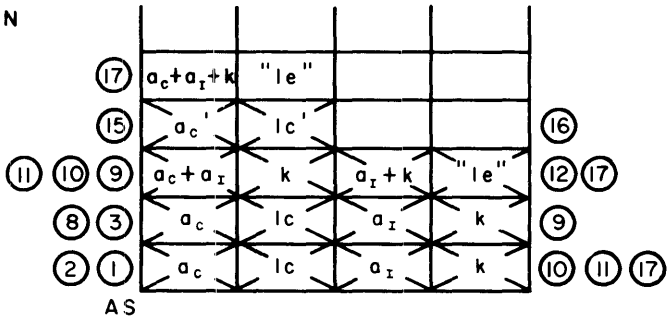


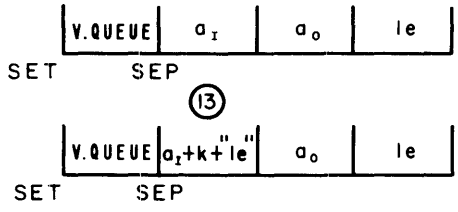
ENTER

```
1.  AS.ae ← SEP.aI
2.  AS.le ← k
3.  DUPLICATE AS.ac, lc
    IF (AS.ae ≥ SEP.ao) AND
    (AS.ae+k+[VSS]+k>AS.lc) THEN
      BEGIN
        IF [VSS] ≥ SEP.ao THEN
          FAULT (FULL)
        ELSE
          BEGIN
4.          COMBINE
5.          AS.le ← [VSS]
6.          STORE FROM AS.le
7.          AS.ae ← 0
          END;
      END
    ELSE
      BEGIN
        IF AS.ae+k+[VSS] ≥ SEP.ao THEN
          FAULT (FULL)
        ELSE
          BEGIN
8.          DUPLICATE AS.ae, le
9.          COMBINE
10.         POSITIVE SEQUENCE AS.ae
11.         ASO.le ← [VSS]
12.         STORE FROM AS.le
          END;
      END;
13. POSITIVE SEQUENCE SEP.aI
14. UPDATE SEP.aI
15. REMOVE (VSS) <NOT A MICRO-OPERATOR>
16. DELETE AS.ac, lc
17. COMBINE
18. R
```

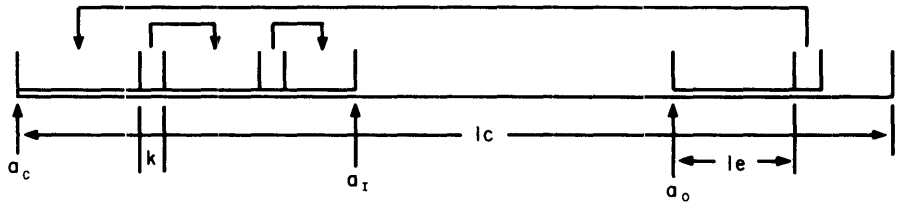






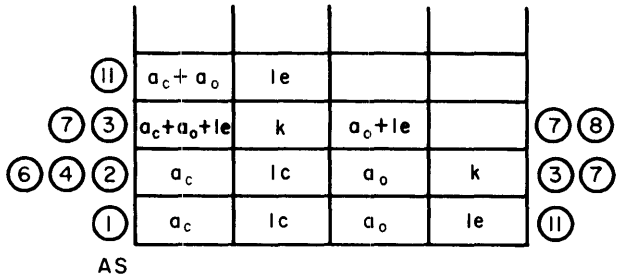


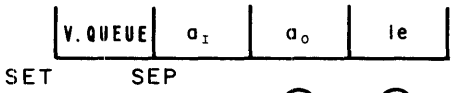
VARIABLE QUEUE

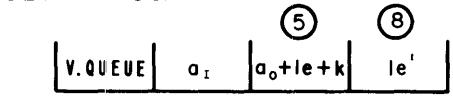


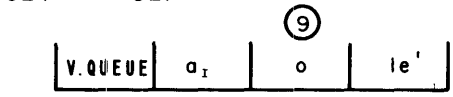
REMOVE

1. AS.a_e, le — SEP.a_o, le
   IF SEP.a_I = AS.a_e THEN
   FAULT (EMPTY)
2. DUPLICATE AS.a_e
3. POSITIVE SEQUENCE AS.a_e
4. AS.le — k
5. POSITIVE SEQUENCE SEP.a_c
   IF SEP.$a_o \neq$ SEP.a_I THEN
   BEGIN
6. DUPLICATE AS.a_c, lc
7. COMBINE
8. FETCH TO SEP.le
   IF SEP.le > AS.lc - SEP.a_o THEN
9. SEP.a_o — o ;
   END
   ELSE
10. DELETE AS.a_e, le ;
11. COMBINE
12. R

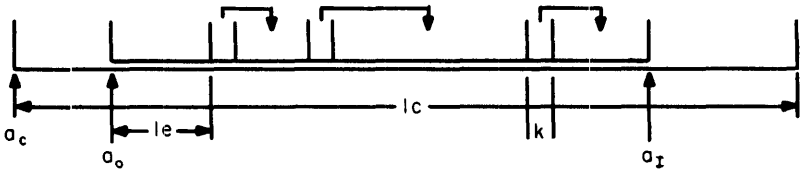






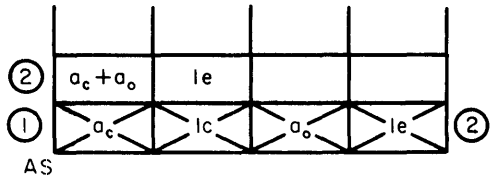


VARIABLE QUEUE

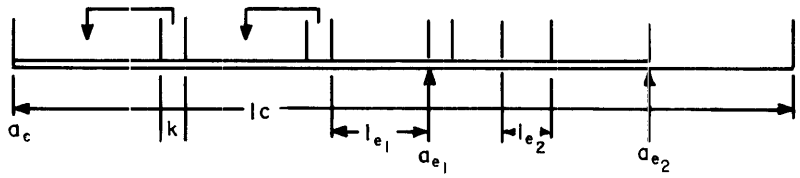


CONSTRUCT

1. AS. a_e, le ⟵ SEP. a_o, le
2. COMBINE
3. R

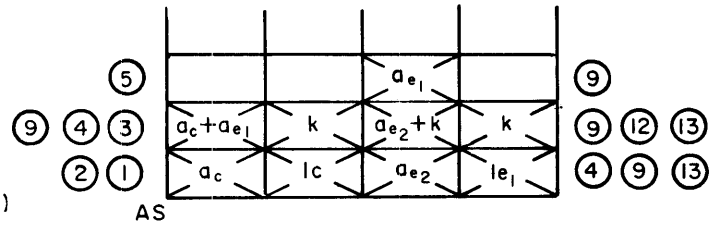


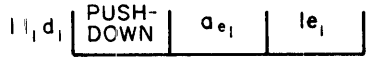
PUSHDOWN STACK

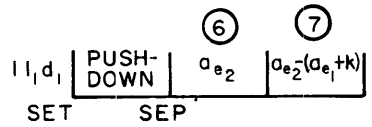


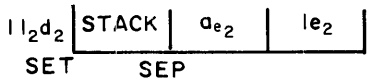
ENTER

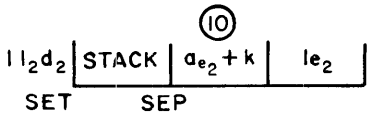
1. AS.a_e ← SEP.a_{e_2}
2. AS.le ← le_1
3. AS.le ← k
4. POSITIVE SEQUENCE AS.a_e
5. AS.a_e ← SEP.a_{e_1}
6. SEP.a_{e_1} ← SEP.a_{e_2}
7. SEP.l_{e_1} ← SEP.a_{e_2} − (AS.a_e + AS.le)
8. UPDATE SEP.a_{e_1}, le_1
9. COMBINE
10. SEP.a_{e_2} ← AS.a_e
11. UPDATE SEP.a_{e_2}
12. STORE FROM AS, le
13. DELETE AS.a_e, le
14. R





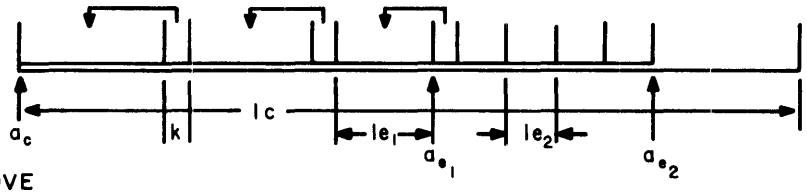






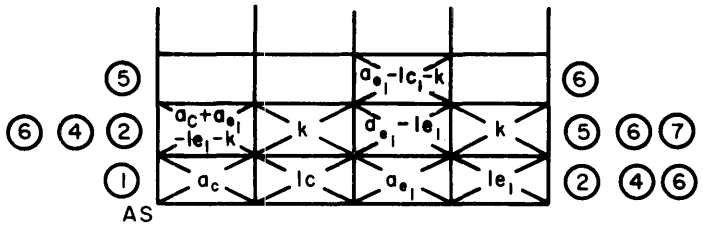
CP 1720-5592

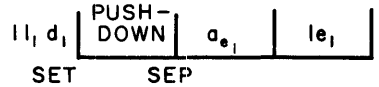
PUSHDOWN STACK

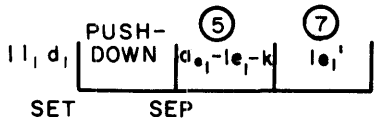


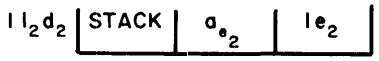
REMOVE

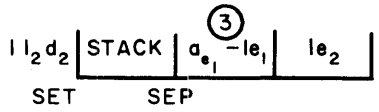
1. AS.a_{e_1}, le ← SEP.a_{e_1}, le_1
2. NEGATIVE SEQUENCE AS.a_e
3. SEP.a_{e_2} ← AS.a_e
4. ASO.le ← k
5. NEGATIVE SEQUENCE
6. COMBINE
7. FETCH TO SEP. le
8. UPDATE SEP. a_{e_2}
9. UPDATE SEP. a_{e_1} , le_1
10. R



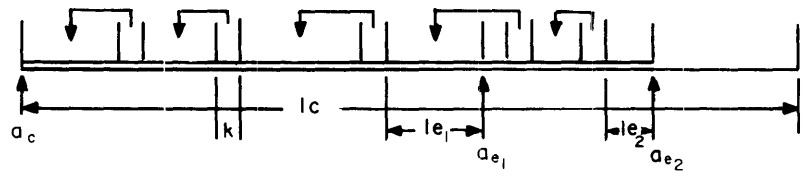






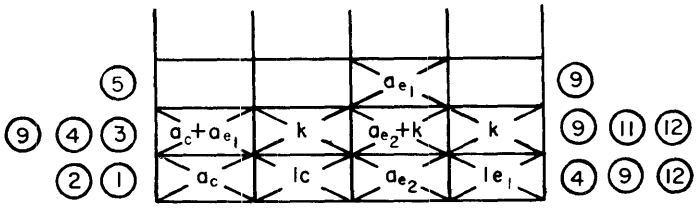


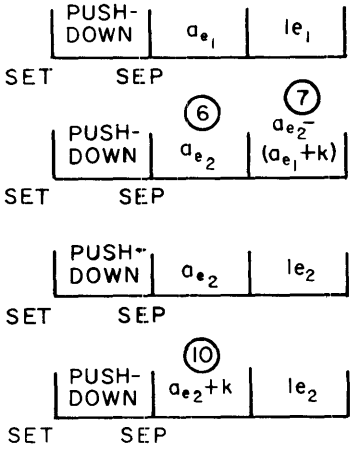
PUSHDOWN - PUSHDOWN



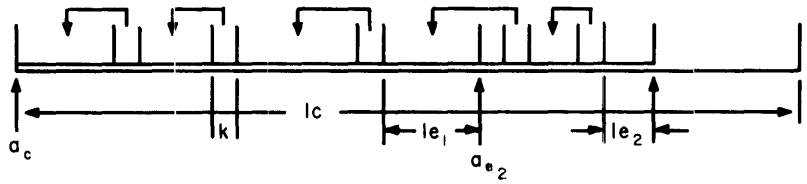
ENTER

1. AS. $a_e \leftarrow$ SEP. a_{e_2}
2. AS. $le \leftarrow le_1$
3. AS. $le \leftarrow k$
4. POSITIVE SEQUENCE AS. a_e
5. AS. $a_e \leftarrow$ SEP. a_{e_1}
6. SEP. $a_{e_1} \leftarrow$ SEP. a_{e_2}
7. SEP. $le_1 \leftarrow$ SEP. $a_{e_2} - (AS.a_e + AS.le)$
8. UPDATE SEP. a_{e_1}, le_1
9. COMBINE
10. SEP. $a_{e_2} \leftarrow$ AS. a_e
11. UPDATE SEP. a_{e_2}
12. STORE FROM AS. le
13. DELETE AS. a_e, le
14. R



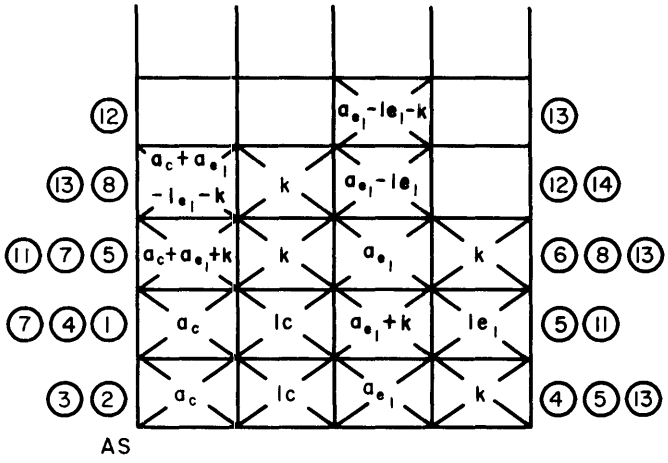


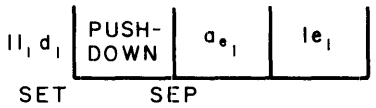
PUSHDOWN - PUSHDOWN

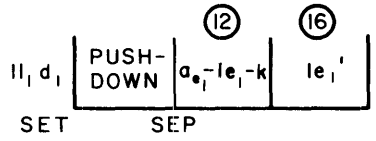


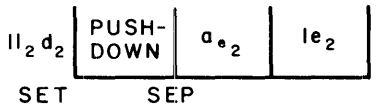
REMOVE

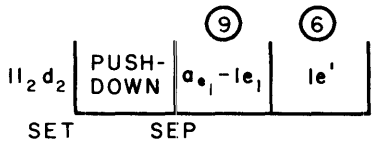
1. DUPLICATE AS.a_c, lc
2. AS.a_e ← SEP.a_{e_1}
3. AS.le ← k
4. POSITIVE SEQUENCE AS.a_e
5. COMBINE
6. FETCH TO SEP.le_2
7. AS.a_e, le ← SEP.a_{e_1}, le_1
8. NEGATIVE SEQUENCE AS.a_e
9. SEP.a_{e_2} ← AS.a_e
10. UPDATE SEP.a_{e_2}, le_2
11. ASO. le ← k
12. NEGATIVE SEQUENCE
13. COMBINE
14. FETCH TO SEP.le_1
15. UPDATE SEP.a_{e_1}, le_1
16. R







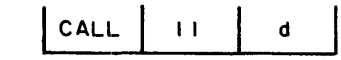




CALL

CONSTRUCT

1. AS.a_e, le ← SEP.11, d
2. EVALUATE
3. CALL <MICRO-OPERATOR>
4. R



SET SEP

FIN

CONSTRUCT

1. IF PDCR STACK IS EMPTY THEN
   RETURN TO PROGRAM
   ELSE
   POP A VALUE TO THE PDCR
2. R

APPENDIX B

PROGRAM OPERATOR ALGORITHMS

APPENDIX B CONTAINS THE ALGORITHMS USED BY THE INTERPRETER FOR EXECUTION OF THE PROGRAM OPERATORS.

EACH OPERATOR ALGORITHM STARTS WITH AN ENVIRONMENT AT ENTRY TO SHOW THE STATE OF THE INTERRETER JUST PRIOR TO START OF OPERATOR EXECUTION.

THEN THE PURPOSE OF EACH OPERATOR IS GIVEN, FOLLOWED BY A DESCRIPTION OF WHAT THE OPERATOR DOES.

THE INSTRUCTION FLOW GIVING DETAILED EXECUTION OF THE OPERATOR FOLLOWS THE DESCRIPTION. THE INSTRUCTION FLOWS ARE WRITTEN IN PSEUDO-ALGOL FOR CLARITY.

STRUCTURE OPERATORS

CONSTRUCT (NAME)

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO BUILD A REFERENCE.

DESCRIPTION

CONSTRUCT FETCHES THE NAMED DESCRIPTOR AND BUILDS A REFERENCE BY EXECUTING THE DESCRIPTION. THE EXECUTION SEQUENCE DEPENDS ON THE STRUCTURE EXPRESSION TYPES AND IS GIVEN IN APPENDIX A.

INSTRUCTION FLOW

```
L1:  IF ACCESS ATTRIBUTES SPECIFIED THEN

         ACCESS COLLECTION STACK ← ACCESS ATTRIBUTES;

     IF INTERPRETER ATTRIBUTES SPECIFIED THEN

         ACCESS COLLECTION STACK ← INTERPRETER ATTRIBUTES

     IF ALLOCATE BIT ≠ 1 THEN

         SET ALLOCATE FAULT
         END;

     IF S-EXPRESSION TYPE = CALL THEN
```

```
        L2:  PUSH DPCS
             DPCR ← EVALUATE (NAME)
             FETCH DESCRIPTOR
             GO TO L1.
             END;

    IF S-EXPRESSION TYPE = SEGMENT NUMBER THEN

        IF DEFAULT SEGMENT NUMBER SPECIFIED
              GO TO NEXT S-EXPRESSION TYPE
        ELSE AC,LC ← RSB (SEG #)

    IF DEFAULT SEGMENT NUMBER NOT SPECIFIED THEN
        SET S-EXPRESSION FAULT
        END

    IF S-EXPRESSION TYPE ≠ END, AND ≠ CALL THEN

        CONSTRUCT (S-EXP TYPE);

    IF S-EXPRESSION TYPE = CALL THEN

        GO TO L2;

    IF S-EXPRESSION TYPE = FIN THEN

        IF INTERPRETER ATTRIBUTE = DESCRIPTOR THEN

            DPCR ← AC,LC
            FETCH DESCRIPTOR
            GO TO L1;

        ELSE IF DPCS ≠ EMPTY THEN

            POP DPCS
            FETCH DESCRIPTOR
            GO TO L1

        ELSE

    NSO ← ACCESS ATTRIBUTES
    NSO ← INTERPRETER ATTRIBUTES
    NSO ← SEGMENT NUMBER
    NSO ← FIELD EXPRESSION TYPE
    NSO ← AC,LC
    NSO ← FIN

            END
```

ENTER (NAME)

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO ALLOCATE SPACE IN A STRUCTURE AND BUILD A REFERENCE TO THAT SPACE.

DESCRIPTION

ENTER FETCHES THE NAMED DESCRIPTOR, FINDS THE INNERMOST STRUCTURE, ALLOCATES A FIELD IN THE INNERMOST STRUCTURE, AND BUILDS A TERMINAL REFERENCE TO THE NEWLY ALLOCATED FIELD IN THE TOP OF THE NAME STACK.

INSTRUCTION FLOW

```
L1:  IF ACCESS ATTRIBUTES SPECIFIED THEN

         ACCESS COLLECTION STACK ← ACCESS ATTRIBUTES;

     IF INTERPRETER ATTRIBUTES SPECIFIED THEN

         ACCESS COLLECTION STACK ← INTERPRETER ATTRIBUTES

     IF ALLOCATE BIT ≠ 1 THEN

         SET ALLOCATE FAULT
         END;

     IF S-EXPRESSION TYPE = CALL THEN

         PUSH DPCS
         DPCR ← EVALUATE (NAME)
         FETCH DESCRIPTOR
         GO TO L1.
         END;
```

```
      IF S-EXPRESSION TYPE SEGMENT NUMBER THEN

          IF DEFAULT SEGMENT NUMBER SPECIFIED

              GO TO NEXT S-EXPRESSION TYPE

          ELSE AC,LC ← RSB (SEG #)

      IF DEFAULT SEGMENT NUMBER NOT SPECIFIED THEN

          SET S-EXPRESSION FAULT
          END

L2:   IF S-EXP TYPE I ≠ CALL AND S-EXP TYPE I + 1 ≠ FIN THEN

          CONSTRUCT (S-EXP TYPE)
          INCREMENT DPCR
          GO TO L2;

      IF S-EXP TYPE I = CALL AND S-EXP TYPE I + 1 ≠ FIN THEN

          PUSH DPCR
          DPCR ← EVALUATE (LL,D)
          FETCH DESCRIPTOR
          GO TO L1;

      IF S-EXP TYPE I = [CALL OR VECTOR OR FIELD OR VARIABLE FIELD]
          AND S-EXP TYPE I +1 = FIN
          AND DPCS = EMPTY
          AND INTERPRETER TYPE ≠ DESCRIPTOR THEN

          FAULT;

      IF S-EXP TYPE = FIN AND DPCS ≠ EMPTY THEN

          POP DPCS
          FETCH DESCRIPTOR
          GO TO L1;

      IF S-EXPRESSION TYPE = FIN AND INTERPRETER TYPE = DESCRIPTOR
      THEN

          DPCR ← AC,LC
          FETCH DESCRIPTOR
          GO TO L1;
```

```
ENTER (S-EXP TYPE)
NSO ← ACCESS ATTRIBUTES
NSO ← INTERPRETER ATTRIBUTUTES
NSO ← SEGMENT NUMBER
NSO ← FIELD = [S-EXPRESSION TYPE]
NSO ← AC,LC
NSO ← FIN =[S-EXPRESSION TYPE]

END
```

CP 1720-5592

REMOVE (NAME)

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO DEALLOCATE SPACE IN A STRUCTURE AND BUILD A REFERENCE TO THAT SPACE.

DESCRIPTION

REMOVE FETCHES THE NAMED DESCRIPTOR, FINDS THE INNERMOST STRUCTURE, DEALLOCATES A FIELD IN THE INNERMOST STRUCTURE AND BUILDS A TERMINAL REFERENCE TO THE NEWLY REALLOCATED FIELD IN THE TOP OF THE NAME STACK.

INSTRUCTION FLOW

```
L1:  IF ACCESS ATTRIBUTES SPECIFIED THEN

         ACCESS COLLECTION STACK ← ACCESS ATTRIBUTES;

     IF INTERPRETER ATTRIBUTES SPECIFIED THEN

         ACCESS COLLECTION STACK ← INTERPRETER ATTRIBUTES

     IF ALLOCATE BIT ≠ 1 THEN

         SET ALLOCATE FAULT
         END;

     IF S-EXPRESSION TYPE = CALL THEN

              PUSH DPCS
              DPCR ← EVALUATE (LL,D)
              FETCH DESCRIPTOR
              GO TO L1.
              END;

     IF S-EXPRESSION TYPE SEGMENT NUMBER THEN

              IF DEFAULT SEGMENT NUMBER SPECIFIED
```

```
                GO TO NEXT S-EXPRESSION TYPE

            ELSE AC,LC ← RSB (SEG #)

    IF DEFAULT SEGMENT NUMBER NOT SPECIFIED THEN

            SET S-EXPRESSION FAULT
            END

L2: IF S-EXP TYPE I ≠ CALL AND S-EXP TYPE I + 1 ≠ FIN THEN

        CONSTRUCT (S-EXP TYPE)
        INCREMENT DPCR
        GO TO L2;

    IF S-EXP TYPE I = CALL AND S-EXP TYPE I + 1 + FIN THEN

        PUSH DPCR
        DPCR ← EVALUATE (LL,D)
        FETCH DESCRIPTOR
        GO TO L1;

    IF S-EXP TYPE I = [CALL OR VECTOR OR FIELD OR VARIABLE FIELD]
        AND S-EXP TYPE I+1 = FIN AND DPCS = EMPTY
        AND INTERPRETER ATTRIBUTE ≠ DESCRIPTOR THEN

        FAULT;

    IF S-EXP TYPE I = FIN AND DPCS ≠ EMPTY THEN

        POP DPCS
        FETCH DESCRIPTOR
        GO TO L1;

    IF S-EXP TYPE = FIN AND INTERPRETER ATTRIBUTES = DESCRIPTOR
    THEN

        DPCR ← AC,LC
        FETCH DESCRIPTOR
        GO TO L1;

    REMOVE (S-EXP TYPE)

    NSO ← ACCESS ATTRIBUTES
    NSO ← INTERPRETER ATTRIBUTES
    NSO ← SEGMENT NUMBER
    NSO ← FIELD -[S-EXPRESSION]
    NSO ← AC,LC
    NSO ← FIN [S-EXPRESSION]

    END
```

REFERENCE OPERATORS

NAME

ENVIRONMENT AT ENTRY

ENTRY IN NSO

PURPOSE

NAME IS UTILIZED TO PRODUCE A REFERENCE TO A FIELD OR PROGRAM.

DESCRIPTION

THE NAME OPERATOR EVALUATES THE INTERPRETER ATTRIBUTES. IF THE INTERPRETER ATTRIBUTE IS DATA, THE DESCRIPTOR IS LEFT IN NSO. IF THE INTERPRETER ATTRIBUTE IS PROGRAM, A FUNCTION CALL IS MADE.

INSTRUCTION FLOW

```
IF INTERPRETER ATTRIBUTES = PROGRAM THEN
      IF INTERPRETER ATTRIBUTES ≠ FUNCTION THEN
          FAULT
      ELSE EXECUTE
          PROCEDURE RETURN;
END
```

CP 1720-5592

VALUE

ENVIRONMENT AT ENTRY

ENTRY IN NSO

PURPOSE

VALUE IS USED TO PLACE AN OPERAND IN THE VALUE STACK.

DESCRIPTION

THE VALUE OPERATOR EVALUATES THE INTERPRETER ATTRIBUTES. IF THE INTERPRETER ATTRIBUTE IS DATA, THE DATA IS FETCHED AND ENTERED IN THE VALUE STACK. IF THE INTERPRETER ATTRIBUTE IS PROGRAM, A PROCEDURE CALL IS MADE.

INSTRUCTION FLOW

```
IF INTERPRETER ATTRIBUTES = PROGRAM THEN

      IF INTERPRETER ATTRIBUTES ≠ FUNCTION THEN
      FAULT
      ELSE EXECUTE

          PROCEDURE RETURN;

IF INTERPRETER ATTRIBUTES = DATA OR LOCK THEN

      VS ← [NSO]
      NSO ← MU*
      END

ELSE SET FAULT

      END;
```

STORE

ENVIRONMENT AT ENTRY

ENTRY IN NSO
MU OR MU* IN NS1

PURPOSE

STORE IS USED TO STORE A VALUE IN LEVEL-1.

DESCRIPTION

THE INTERPRETER ATTRIBUTES ARE EVALUATED. IF THE ATTRIBUTE IS PROGRAM THEN A FUNCTION CALL IS MADE. A DATA DESCRIPTOR IS RETURNED AS A RESULT OF THE FUNCTION CALL. THE FORMAT OF THE VALUE IS TRANSFORMED TO THE FORMAT OF THE STORED FIELD. THE VALUE IS THEN STORED IN THE FIELD POINTED TO BY THE DATA DESCRIPTOR.

INSTRUCTION FLOW

```
IF INTERPRETER ATTRIBUTES = PROGRAM THEN

    IF INTERPRETER ATTRIBUTE ≠ FUNCTION THEN

        FAULT

    ELSE EXECUTE

        PROCEDURE RETURN

    IF INTERPRETER ATTRIBUTES = DATA OR LOCK THEN

        IF NSO FORMAT SEL = NS1. FORMAT SEL THEN

            AU TRANSFORM;
            [NSO] ← VSO
            VS OFF
            NSO OFF
            NS1 OFF
            END

        ELSE FAULT

            END;
```

CP 1720-5592

EXECUTE

ENVIRONMENT AT ENTRY

ENTRY IN NSO

PURPOSE

EXECUTE IS USED TO PROGRAMMATICALLY CALL PROCEDURES.

DESCRIPTION

THE EXECUTE OPERATOR EVALUATES THE INTERPRETER A PROCEDURE CALL IS MADE. OTHERWISE THE REFERENCE IS LEFT ON THE TOP OF NS. A PROCEDURE CALL ENTERS THE PROGRAM DESCRIPTOR IN THE PROGRAM CONTROL STACK. IF THE PROCEDURE HAS PARAMETERS A SLICE IS CREATED AND THE REFERENCE ENTERED IN NSS. THE SLICE REFERENCE IS ALSO STORED IN THE DISPLAY AT THE LEXIC LEVEL SPECIFIED BY THE CALLED PROGRAM. THE DISPLAY IS UPDATED.

INSTRUCTION FLOW.

```
IF INTERPRETER ATTRIBUTES = PROGRAM THEN

      PUSH PCS
      PCR ← NSO
      NSO OFF

IF PCR.P=1 THEN

      ENTER V
      NSO OFF
      ENTER N
      DUPLICATE
      CONSTRUCT DISPLAY (PCR,LL)
      STORE
      UPDATE DISPLAY;

      END;

END
```

DESCRIPTOR OPERATORS

LOAD

ENVIRONMENT AT ENTRY

NSO FULL

PURPOSE

GENERAL

DESCRIPTION

LOAD FINDS A DESCRIPTOR IN NSS. THE INTERPRETER ATTRIBUTES ARE EVALUATED IN ORDER TO DETERMINE SUBSEQUENT ACTION. IF THE INTERPRETER ATTRIBUTFS ARE PROGRAM OR DATA, THE DESCRIPTOR REMAINS IN NSS. IF THE INTERPRETER ATTRIBUTE IS DESCRIPTOR, A REFERENCE IS BUILT BY EXECUTING THE DESCRIPTOR. THE CONTENTS OF THE FIELD SPECIFIED BY THE REFERENCE REPLACE THE GIVEN DESCRIPTION.

INSTRUCTION FLOW

```
EVALUATE DESCRIPTOR IN NSO
IF INTERPRETER ATTRIBUTE IS DESCRIPTOR THEN

     FETCH DESCRIPTOR

ELSE END
```

CP 1720-5592

DESCRIBE (NAME)

ENVIRONMENT AT ENTRY

NSO FULL

PURPOSE

TO GENERATE A DESCRIPTION OF A DESCRIPTION.

DESCRIPTION

DESCRIBE WILL SET THE ACCESS ATTRIBUTES TO THE CLEARED STATE; THE INTERPRETER ATTRIBUTES WILL BE SET TO DESCRIPTOR. THE STRUCTURE EXPRESSION GENERATED WILL BE A FIELD WITH THE AE AND LE OF THE NAME PASSED TO THE DESCRIBE OPERATOR AS A PARAMETER FOLLOWED BY AN END. THE GENERATED DESCRIPTION IS LEFT IN NSS.

INSTRUCTION FLOW

```
NS.  ACCESS ATTRIBUTES ← CLEAR
NSO.  INTERPRETER ATTRIBUTES ← DESCRIPTOR
NSO.  S-EXP ← SEG NUMBER
EVALUATE (NAME)
NS1. S-EXP ← NSO (REF)
DELETE
NSO.  S-EXP ← FIN
END
```

ALLOCATE (NAME 1, NAME 2)

ENVIRONMENT AT ENTRY

NSO, NS1 FULL

PURPOSE

TO ALLOCATE SPACE TO AN UNALLOCATED STRUCTURE.

DESCRIPTION

ALLOCATE RECEIVES THE NAME OF AN ALLOCATED STRUCTURE AND THE NAME OF AN UNALLOCATED STRUCTURE. THE NAMED DESCRIPTION OF THE UNALLOCATED STRUCTURE IS EVALUATED TO SEE IF THE FIRST STRUCTURE EXPRESSION IS A CALL OR A SEGMENT NUMBER, FIELD. IF NEITHER, AN ILLEGAL OPERATION FAULT IS SET. IF IT IS A CALL, THE NAME OF THE ALLOCATED STRUCTURE IS PASSED TO THE CALL. IF IT IS A SEGMENT-NUMBER FIELD, THE NAMED DESCRIPTOR OF THE ALLOCATED STRUCTURE IS EXECUTED IN ENTER MODE. THE RESULTING REFERENCE IS PASSED TO THE FIELD EXPRESSION OF THE UNALLOCATED STRUCTURE.

INSTRUCTION FLOW

```
IF S-EXP1 = SEGMENT NUMBER THEN

          IF S-EXP2 = FIELD THEN

          ENTER (LL2,D2) YIELDS REFERENCE
          FIELD (AE,LE) ← REFERENCE

          ELSE FAULT

ELSE IF S-EXP1 = CALL THEN

          CALL(LL,D) ← (LL2,D2)

     ELSE FAULT

END
```

CP 1720-5592

BIND (NAME 1, NAME 2)

ENVIRONMENT AT ENTRY

NS0,NS1 FULL

PURPOSE

TO MAP AN UNALLOCATED STRUCTURE ONTO AN ALLOCATED SPACE.

DESCRIPTION

THE BIND OPERATOR IS GIVEN THE NAME OF AN ALLOCATED STRUCTURE AND THE NAME OF AN UNALLOCATED STRUCTURE. THE NAMED DESCRIPTION OF THE UNALLOCATED STRUCTURE IS EVALUATED TO SEE IF THE FIRST STRUCTURE EXPRESSION IS A CALL OR A SEGMENT NUMBER, FIELD. IF IT IS NEITHER, AN ILLEGAL OPERATION FAULT IS SET. IF IT IS A CALL, THE NAME OF THE ALLOCATED STRUCTURE IS PASSED TO THE CALL. IF IT IS A SEGMENT-NUMBER-FIELD, THE NAMED DESCRIPTOR OF THE ALLOCATED STRUCTURE IS EXECUTED IN CONSTRUCT MODE. THE RESULTING REFERENCE IS PASSED TO THE FIELD EXPRESSION OF THE UNALLOCATED STRUCTURE. THE DESCRIPTION OF THE NEWLY ALLOCATED STRUCTURE REMAINS IN NSS.

INSTRUCTION FLOW

```
IF S-EXP1 = SEGMENT NUMBER THEN

          IF S-EXP 2 = FIELD THEN

              CONSTRUCT (LL2,D2) YIELDS REFERENCE
              FIELD (AE,LE) ← REFERENCE

          ELSE FAULT

ELSE IF S-EXP2 = CALL THEN

          CALL (LL,D) ← (LL2,D2)

     ELSE FAULT

END
```

CP 1720-5592

SHORTEN (NAME)

ENVIRONMENT AT ENTRY

NSO FULL

PURPOSE

TO REDUCE THE SIZE OF A DESCRIPTOR.

DESCRIPTION

THE SHORTEN OPERATOR FETCHES THE NAMED DESCRIPTOR. SHORTEN EXECUTES THE FIRST TERM OF THE STRUCTURE EXPRESSION THAT IS NOT SEGMENT NUMBER OR FIELD, AND REPLACES THAT TERM WITH THE DERIVED FIELD EXPRESSION. CONSECUTIVE FIELD EXPRESSIONS ARE COMBINED. THE OPERATOR YIELDS A NEW DESCRIPTOR IN NSS.

INSTRUCTION FLOW

NOTE

S = S-EXPRESSION
N1 = N PRIME
N11 = N DOUBLE PRIME

```
L1: IF S(N) = SEGMENT NUMBER THEN
                    N ← N + 1
                    GO TO L1
    ELSE IF S(N) = FIELD THEN
                IF S(N+ 1) = FIELD THEN
                        NSO, S(N) ← COMBINE (S(N),S(N+ 1))
                        GO TO L2
```

```
                    ELSE IF S (N + 1) = SEGMENT THEN FAULT

                        ELSE S (N1) ← CONSTRUCT (NAME N + 1)
                        NSO, S (N) ← COMBINE (S(N), S (N1))
                        GO TO L2

              ELSE S (N1) ← CONSTRUCT (NAME N)
              IF S (N + 1) = SEGMENT NUMBER THEN FAULT
              ELSE IF S (N +1) = FIELD THEN

                        NSO,  S(N) ← COMBINE (S (N1), S (N + 1))
                        GO TO L2

                   ELSE S (N11) ← CONSTRUCT NAME (N + 1)
                   NSO, S (N) ← COMBINE (S(N1), S (N11))

L2: N ← N + 1

      IF S (N) = END THEN GO TO L3
      ELSE S (N) ← S (N + 1)
      GO TO L2

L3: END
```

CP 1720-5592

FINAL COMBINE

ENVIRONMENT AT ENTRY

PRIVILEGED MODE AND TERMINAL REFERENCE IN NSO

PURPOSE

TO CALCULATE THE ABSOLUTE ADDRESS OF THE STRUCTURE.

DESCRIPTION

FINAL COMBINE USES THE SEGMENT NUMBER AS A DISPLACEMENT INTO RL TO FIND THE ABSOLUTE CONTAINER OF THE STRUCTURE. THIS ABSOLUTE CONTAINER IS USED TO CALCULATE THE ABSOLUTE ADDRESS OF THE STRUCTURE. THIS OPERATOR MAY BE EXPLICITLY CALLED ONLY IN PRIVILEGED MODE.

INSTRUCTION FLOW

```
      IF STORE AND WRITE FAULT BIT ON OR
      FETCH AND READ FAULT BIT ON THEN

          EVALUATE (FAULT LL, D)
          EXECUTE

      SAVE L

L1:   NS ← RS [SEG #]

      IF INDIRECT THEN

          NSO ← DESC
          L ← L - 1
          CONSTRUCT (DESC)
          COMBINE
          GO TO L1

      COMBINE
      RESTORE L
      END
```

STACK OPERATORS

DUPLICATE

ENVIRONMENT AT ENTRY

NSO IS FULL

PURPOSE

GENERAL

DESCRIPTION

THE DUPLICATE OPERATOR DUPLICATES THE TOP ENTRY OF THE NAME STACK. IF THE CONTENTS OF THE NSS CONTAIN A REFERENCE TO VSS, THE CONTENTS OF VSS ARE ALSO DUPLICATED.

INSTRUCTION FLOW

```
IF MU IS IN NSO THEN

      POP VSS.

ELSE IF MU* IS IN NSO THEN

      ADJUST VSO TO VS1
      VSO ← VS1
      ADJUST NSO TO NS1
      NSO ← NS1

ELSE ADJUST NSO TO NS1

END
```

CP 1720-5592

DELETE

ENVIRONMENT AT ENTRY

NSO FULL

PURPOSE

GENERAL

DESCRIPTION

THE DELETE OPERATOR DELETES THE TOP ENTRY OF NSS. IF THE TOP ENTRY OF NSS CONTAINS A REFERENCE TO VSS, THE CONTENTS OF VSS ARE ALSO DELETED.

INSTRUCTION FLOW

```
IF MU IS IN NSO THEN

      REMOVE VSS
      NSO OFF

ELSE IF MU* IS IN NSO THEN

      NSO OFF
      VSO OFF

ELSE NSO OFF
END
```

CP 1720-5592

EXCHANGE

ENVIRONMENT AT ENTRY

NSO AND NS1 FULL

PURPOSE

GENERAL

DESCRIPTION

THE EXCHANGE OPERATOR EXCHANGES THE TOP TWO ENTRIES OF THE NAME STACK. IF BOTH ENTRIES OF THE NAME STACK CONTAIN REFERENCES TO THE VALUE STACK, THE TOP TWO ENTRIES OF THE VALUE STACK ARE ALSO EXCHANGED.

INSTRUCTION FLOW

```
IF MU OR MU* IN NSO AND NS1 THEN

      ADJUST TOP TWO ENTRIES TO VSO AND VS1
      VSO ]-*-> VS1
      NSO <-*-> NS1

ELSE NSO <-*-> NS1
END
```

CP 1720-5592

VSS TO NSS

ENVIRONMENT AT ENTRY

MU OR MU* IN NSO

PURPOSE

GENERAL

DESCRIPTION

THIS OPERATOR REMOVES THE TOP ENTRY OF THE VALUE STACK AND ENTERS IT INTO THE TOP OF THE NAME STACK.

INSTRUCTION FLOW

```
ADJUST TO VSO
AU TRANSFORM
NSO ← VSO
VSO OFF
END
```

CP 1720-5592

NSS TO VSS

ENVIRONMENT AT ENTRY

VALUE IN NSO

PURPOSE

GENERAL

DESCRIPTION

THE TOP ENTRY OF THE NAME STACK IS ENTERED INTO THE VALUE STACK AND A REFERENCE TO THE NEW VALUE STACK ENTRY REPLACES THE ORIGINAL IN THE NAME STACK.

INSTRUCTION FLOW

VSO ← NSO
NSO ← "MU*"

PROGRAM CONTROL OPERATORS

LOOP (NAME)

ENVIRONMENT AT ENTRY

CONTROL VARIABLE IN VALUE STACK

PURPOSE

TO BEGIN A PROGRAM LOOP

DESCRIPTION

THE OPERATOR CAUSES A SUBROUTINE CALL ON THE NAMED SUBPROGRAM. THE TOP OF PCS IS DUPLICATED PRIOR TO THE ACTUAL EXECUTION OF THE SUBPROGRAM.

INSTRUCTION FLOW

```
LOOP DESCRIPTION + EVALUATION (NAME)
PUSH PCS
PCS + PCR
PUSH PCS
PCR + LOOP DESCRIPTION
PCS + PCR
END
```

CP 1720-5592

LOOP TEST

ENVIRONMENT AT ENTRY

CONTROL VARIABLE IN USS

PURPOSE

TO DETERMINE IF THE END OF THE LOOP HAS BEEN ATTAINED.

DESCRIPTION

LOOP TEST SHALL TEST THE VALUE OF THE VARIABLE IN THE VALUE STACK. IF IT IS LESS THAN ZERO, PCS, VSS, AND NSS ARE DELETED CAUSING A RETURN TO THE OPERATOR FOLLOWING THE LOOP OPERATOR. IF THE VALUE IS GREATER THAN OR EQUAL TO ZERO, PCS IS DUPLICATED AND EXECUTION RESUMED FROM THE START OF THE LOOP ROUTINE.

INSTRUCTION FLOW

```
IF [VSS] < ZERO THEN

      POP PCS
      PCR ← PCS
      POP PCS
      NSO OFF
      VSO OFF

ELSE PCR←PCS
END
```

CP 1720-5592

BRANCH

ENVIRONMENT AT ENTRY

NSO FULL

PURPOSE

TO CAUSE A CHANGE IN PROGRAM CONTROL

DESCRIPTION

THE BRANCH OPERATOR EVALUATES THE INTERPRETER ATTRIBUTES OF THE DESCRIPTOR IN NSS TO DETERMINE THE ACTION TO BE TAKEN.

INSTRUCTION FLOW

```
IF [NSS,CODE] = 01 THEN

              BEGIN
                   CONSTRUCT (NAME)
                   PCR ← NSO
              END

ELSE IF [NSS,CODE] = 10 THEN

              BEGIN
                   IF MU THEN POP VSS
                   IF NOT MU* THEN CONSTRUCT (NAME) VALUE
                   AS,AC,LC,LE ← PCR,AC,LC,LE
                   AS,AE ← VSO
                   COMBINF
                   PCR,AC,LC ← ASO,AC,LC
              END

ELSE FAULT
```

CP 1720-5592

BRANCH CONDITIONAL

ENVIRONMENT AT ENTRY

BOOLEAN IN VSO
REFERENCE IN NSO
BRANCH DESCRIPTION IN NS1

PURPOSE

TO CAUSE A CHANGE IN PROGRAM CONTROL WHEN A SPECIFIC CONDITION EXISTS.

DESCRIPTION

THE BOOLEAN AT THE TOP OF V IS TESTED. IF IT IS FALSE, A BRANCH IS PERFORMED AFTER THE BOOLEAN AND ITS REFERENCE HAVE BEEN DELETED FROM V AND N. IF THE VALUE IS TRUE, EXECUTION PROCEEDS NORMALLY.

INSTRUCTION FLOW.

```
IF VSO ≠ 0 THEN

          BEGIN
               NSO OFF
               VSO OFF
          END

ELSE NSO OFF
     VSO OFF
     BRANCH
```

HALT

ENVIRONMENT AT ENTRY

NO SPECIFIC ENVIRONMENT

PURPOSE

TO PROVIDE A MEANS OF PROGRAMMATICALLY STOPPING A PROCESSOR.

DESCRIPTION

THE HALT OPERATOR SHALL BE CONTROLLED BY A SWITCH. IF THE SWITCH IS IN THE "HALT" POSITION, THE OPERATOR IS EXECUTED. IF THE SWITCH IS IN THE "CONDITIONAL HALT" POSITION, THE CONTENTS OF THE PCR ARE COMPARED AGAINST THE CONTENTS OF THE CONDITIONAL HALT REGISTER. IF EQUAL, THE OPERATOR IS EXECUTED; OTHER, IT IS TREATED AS A NO/OP. IF THE SWITCH IS IN "NORMAL" POSITION THE OPERATOR IS TREATED AS A NO-OP.

INSTRUCTION FLOW

```
IF SWITCH = NORMAL, GO TO NEXT INSTRUCTION
ELSE IF SWITCH = CONDITIONAL HALT THEN

                    IF [PCR] = [CONDITIONAL HALT REGISTER] THEN
                                                    STOP PROCESSOR
                    ELSE GO TO NEXT INSTRUCTION

ELSE STOP PROCESSOR
END
```

CP 1720-5592

NO-OP

NO SPECIFIC ENVIRONMENT

PURPOSE

DESCRIPTION

NO-OP CAUSES THE PCR TO BE UPDATED AND THE NEXT OPERATOR TO BE EXECUTED.

INSTRUCTION FLOW

END

PROCEDURE CONTROL OPERATORS

SLICE

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO PROGRAMMATICALLY CREATE A LEXIC LEVEL ENTRY.

DESCRIPTION

THE SLICE OPERATOR PERFORMS AN ENTER V, DELETING THE REFFRENCE OBTAINED. IT THEN PERFORMS AN ENTER N AND STORES THE REFERENCE IN THE TOP OF THE NAME STACK AND ALSO IN THE NEXT LEXIC LEVEL IN THE DISPLAY STACK POINTED TO BY THE PROGRAM CONTROL REGISTER LEXIC LEVEL FIELD.

INSTRUCTION FLOW

```
ENTER V
DELETE
ENTER N
DUPLICATE
ENTER DISPLAY (PCR.LL)
STORE
END
```

CP 1720-5592

UNSLICE

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO PROGRAMMATICALLY RFMOVE A LEXIC LEVEL ENTRY.

DESCRIPTION

THE UNSLICE OPERATOR PERFORMS A REMOVE V, DELETING THE REFERENCE IN THE NAME STACK. A REMOVE N IS THEN PERFORMED, AND THE REFERENCE TO THE SLICE REMOVED IS DELETED. A REMOVE DISPLAY IS PERFORMED TO RID THE DISPLAY STACK OF THE LEXIC LEVEL JUST REMOVED. THE REST OF THE DISPLAY IS UPDATED.

INSTRUCTION FLOW

```
REMOVE V
REMOVE N
DELETE
DELETE
REMOVE DISPLAY (PCR,LL)
UPDATE DISPLAY
END
```

PROCEDURE RETURN

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO INVOKE A RETURN FROM A PROCEDURE OR A FUNCTION.

DESCRIPTION

PROCEDURE RETURN REMOVES THE LOCAL VARIABLE SLICES IN THE NAME STACK AND VALUE STACK. IF THE PROCEDURE HAD PARAMETERS THE PARAMETRIC SLICES ARE REMOVED FROM THE NAME STACK AND VALUE STACK. THE DISPLAY IS ALSO UPDATED. IF THE PROCEDURE IS A FUNCTION THEN A POINTER TO THE RESULT RETURNED IS LEFT IN THE TOP OF THE NAME STACK.

INSTRUCTION FLOW

```
 IF PCR.P = 1 THEN

       REMOVE N
       DELETE
       REMOVE V
       DELETE
       REMOVE D
       DELETE;

 IF PCR.F=1 THEN

       DELETE
       REMOVE D;

 UPDATE DISPLAY
 POP PCS
 END
```

COROUTINE OPERATORS

COROUTINE ACTIVATE (NAME)

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO ESTABLISH A SET OF COROUTINES.

DESCRIPTION
COROUTINE ACTIVATE ENTERS THE PRESENT DISPLAY DESCRIPTOR INTO THE COROUTINE STACK. IT THEN DOES A COROUTINE CALL.

INSTRUCTION FLOW

```
NS ← CCF.D
NSS TO VSS
ENTER CD
STORE
COROUTINE CALL
END
```

COROUTINE CALL (NAME)

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO CALL A NEW COROUTINE.

DESCRIPTION

THE ACTIVE COROUTINE CONTROL FIELD IS RETURNED TO THE LOCATION IN THE NAME STACK AND THE NEW COROUTINE CONTROL FIELD IS FETCHED AND PLACED IN THE DESCRIPTOR BUFFER PORTION.

INSTRUCTION FLOW

```
       CONSTRUCT CD
       LOAD
       EVALUATE (LL,D)
       DUPLICATE

 L1:   IF CCF + CCF+6

       THEN
       NS ← CCF

            NSS TO VSS
            EXCHANGE
            STORE
            CCF ← CCF + 1
            + SEQ NS
            GO TO L1
            NS ← (LL,D)

       DELETE
       NSS TO VSS
       ENTER CD
       STORE

       CONSTRUCT LL,D
```

```
IF CCF< CCF + 6 THEN

    NAME
    CCF ← NS
    + SEQ NS
    CCF ← CCF + 1
    GO TO L2

END
```

CP 1720-5592

COROUTINE END

ENVIRONMENT AT ENTRY

NONE

PURPOSE

TO RETURN TO A PARENT COROUTINE.

DESCRIPTION

COROUTINE END STORES THE CURRENT CCF FIELD; AND THEN RESTORES THE CCF FIELD POINTED TO BY THE PARENT CD STACK ENTRY IN THE DESCRIPTOR BUFFER. THE TOP ENTRY OF THE CD STACK IS REMOVED.

INSTRUCTION FLOW

```
       CONSTRUCT CD
       LOAD
       EVALUATE (LL,D)
       DUPLICATE

 L1:   IF CCF < CCF + 6

            NS ← CCF
            NSS TO VSS
            EXCHANGE
            STORE
            CCF ← CCF + 1
            + SEQ NS
            GO TO L1

       DELETE
       REMOVE CD
       NSO OFF
       CONSTRUCT CD
       LOAD
       EVALUATE
       DUPLICATE

 L2:   IF CCF < CCF + 6 THEN
```

```
        NS ← CCF
        NSS TO VSS
        EXCHANGE
        STORE
        CCF ← CCF + 1
        + SEQ NS
        GO TO L2

    DELETE
    END
```

PROCESS CALL OPERATORS

PROCESS PARAMETER (NAME)

ENVIRONMENT AT ENTRY

PRIVILEGED MODE

PURPOSE

TO PASS RESOURCES FROM A PARENT PROCESS TO A CHILD PROCESS.

DESCRIPTION

THE NAME IS EVALUATED GIVING A TERMINAL REFERENCE. THE TERMINAL REFERENCE IS ABSOLUTIZED USING FINAL COMBINE. IF THE STORAGE LEVEL IS LEVEL-1, THE DIRECT BIT IS SET AND THE ABSOLUTIZED REFERENCE IS ENTERED INTO THE RESOURCE STACK.

INSTRUCTION FLOW

```
IF MODE = PRIVILEGED THEN

      CONSTRUCT (NAME)
      FINAL COMBINE

      IF STORE LEVEL = LEVEL-1

          SET DIRECT BIT;

      ENTER RSS
      STORE;

ELSE FAULT;
END
```

CP 1720-5592

PROCESS PARAMETER INDIRECT (NAME)

ENVIRONMENT AT ENTRY

PRIVILEGED MODE

PURPOSE

TO PASS AND/OR CREATE REFERENCES TO EXTERNAL OBJECTS FROM A PARENT PROCESS TO A CHILD PROCESS.

DESCRIPTION

THE NAME IS EVALUATED GIVING A TERMINAL REFERENCE. THE TERMINAL REFERENCE IS ABSOLUTIZED USING FINAL COMBINE. IF THE STORAGE LEVEL IS LEVEL-1, THE DIRECT BIT IS RESET AND THE ABSOLUTIZED REFERENCE IS ENTERED INTO THE RESOURCE STACK.

INSTRUCTION FLOW

```
IF MOD = PRIVILEGED THEN

      CONSTRUCT (NAME)
      FINAL COMBINE

      IF STORAGE LEVEL = LEVEL-1 THEN

          RESET DIRECT BIT;

      ENTER RSS

STORE;
ELSE FAULT;
END
```

CP 1720-5592

PROCESS CALL

ENVIRONMENT AT ENTRY

PRIVILEGED MODE

PURPOSE

TO CREATE A NEW RESOURCE STACK SLICE

DESCRIPTION

A NEW SLICE IS CRFATED IN THE RESOURCE STACK. THE REFERENCE TO THIS SLICE IS ENTERED INTO RL. THE LEVEL POINTER IN RL IS INCREASED BY ONE.

INSTRUCTION FLOW

```
IF MODE = PRIVILEGED THEN

      ENTER R
      ENTER RL
      STORE;

ELSE FAULT
END
```

CP 1720-5592

PROCESS END

ENVIRONMENT AT ENTRY

PRIVILEGED MODE

PURPOSE

TO REMOVE A RESOURCE STACK SLICE.

DESCRIPTION

THE TOP SLICE IN THE RESOURCE STACK IS REMOVED AND THE REFERENCE TO THAT SLICE IN RL IS ALSO DELETED. THE LEVEL POINTER IN RL IS DECREASED BY ONE.

INSTRUCTION FLOW

```
IF MODE = PRIVILEGED THEN

      REMOVE R
      DELETE
      REMOVE RL
      DELETE;

ELSE FAULT
```

MISCELLANEOUS OPERATORS

LITERAL

ENVIRONMENT AT ENTRY

NO SPECIFIC ENVIRONMENT

PURPOSE

GENERAL.

DESCRIPTION

THE LITERAL OPERATOR ENTERS A VALUE FOLLOWING THE OPERATOR IN THE PROGRAM STRING TO THE TOP OF V AND ENTERS A REFERENCE TO IT IN NSS. VALUE LENGTHS MAY BE 4, 8, 16, OR 32 BITS.

INSTRUCTION FLOW

```
PUSH VSS
PUSH NSS
VSO ← "LIT"
NSO ← "MU*"
END
```

CP 1720-5592

SET INTERRUPT

ENVIRONMENT AT ENTRY

CHANNEL NUMBER IN VS0

PURPOSE

SET INTERRUPT ALLOWS FOR INTERMODULE COMMUNICATION.

DESCRIPTION

SET INTERRUPT CAUSES AN INTERRUPT TO BE SET IN THE DEVICE INDICATED BY THE PARAMETER IN VSS (THE CHANNEL NUMBER OF THE UNIT).

INSTRUCTION FLOW

```
INTERRUPT DRIVER (N) ← INTERRUPT      (N = [VSS])
DELETE
END
```

CP 1720-5592

PROCESS FAULT CHECK

ENVIRONMENT AT ENTRY

NO SPECIFIC ENVIRONMENT

PURPOSE

TO DETECT PROCESS FAULTS AND EXECUTE CORRESPONDING ACTION.

DESCRIPTION

THIS OPERATOR SCANS THE PROCESS FAULT REGISTER. IF AN UNMASKED, INTERNAL FAULT IS DETECTED, AN ENTRY INTO THE PROCESSOR FAULT HANDLING SUBROUTINE IS MADE. IF AN UNMASKED EXTERNAL FAULT IS DETECTED, AN IMP CALL IS EXECUTED. WHEN A FAULT IS DETECTED, THE PROCESS FAULT NUMBER IS ENTERED INTO VSS AND ITS REFERENCE IS ENTERED INTO NSS. THE ABSENCE OF A PROCESS FAULT OR A MASKED PROCESS FAULT CAUSES THE NEXT OPERATOR TO BE EXECUTED.

INSTRUCTION FLOW

```
FOR N = 0 STEP 1 UNTIL 64 DO

      IF PF(N) = 1 AND EM(N) = 1 THEN

                                    PUSH NSS
                                    PUSH VSS
                                    ENTER VSS
                                    AU ← [PFR]
                                    VSO ← PROCESS FAULT NUMBER
                                    NSO.IAT ← DATA
                                    NSO.IAD ← MU
                                    RESET PF(N)
                                    IMP CALL
```

CP 1720-5592

```
ELSE IF PF(N) = 1 AND PM(N) = 1 THEN

                                PUSH NSS
                                PUSH VSS
                                ENTER VSS
                                AU ← [PFR]
                                VSO ← PROCESS FAULT NUMBER
                                NSO.IAT ← DATA
                                NSO.IAD ← MU
                                RESET PF(N)
                                EXECUTE PROCESS FAULT HANDLING ROUTI

        ELSE END
```

CP 1720-5592

SYSTEM FLAG CHECK

ENVIRONMENT AT ENTRY

NO SPECIFIC ENVIRONMENT

PURPOSE

TO DETERMINE IF ANY SYSTEM FLAG BITS ARE SET IN THE SYSTEM FLAG REGISTER.

DESCRIPTION

THIS OPERATOR INITIATES A SCAN OF THE SYSTEM FLAG REGISTER. IF A FLAG BIT HAS BEEN SET, THE FLAG NUMBER IS ENTERED INTO VSS, ITS REFERENCE IS ENTERED INTO NSS, AND AN IMP CALL IS EXECUTED. IF NO FLAG BIT WAS SET, THE NEXT OPERATOR IS EXECUTED.

INSTRUCTION FLOW

```
FOR N = 0 STEP 1 UNTIL 16 DO

      IF SF(N) = 1 THEN
                   BEGIN
                        PUSH VSS
                        PUSH NSS
                        ENTER VSS
                        AU ← [SFR]
                        VSO ← SYSTEM FLAG NUMBER
                        NSO.IAT ← DATA
                        NSO.IAD ← MU
                        RESET SF(N)
                        IMP CALL
                    END;

    END.
```

CP 1720-5592

GET V SPACE (NAME)

ENVIRONMENT AT ENTRY
MU OR MU* IN NSO

PURPOSE

TO ALLOCATE SPACE IN V AND CREATE A REFERENCE IN N.

DESCRIPTION

THE OPERATOR USES THE DESCRIPTION OF AN UNALLOCATED FIELD AND PERFORMS AN ENTER IN V OF THAT LENGTH. THE REFERENCE IS ENTERED IN N. IF THE DESCRIPTION IS NOT DATA, A FAULT IS SET.

INSTRUCTION FLOW

```
EVALUATE [NSO]
IF DESCRIPTION IS NOT DATA, SET A FAULT
ELSE ENTER (NAME)
NSO ← "MU*"
END
```

IMP CALL

ENVIRONMENT AT ENTRY

NO SPECIFIC ENVIRONMENT

PURPOSE

IMP CALL IS A MEANS BY WHICH A PROCESS EXECUTES PROCESSOR CONTROL FUNCTIONS. THERE ARE TWO TYPES OF ENTRY INTO IMP: PROGRAMMATICALLY BY IMP CALL AND AUTOMATICALLY BY FAULT. PROGRAMMATIC ENTRY WAITS FOR AU COMPLETE.

DESCRIPTION

IMP CALL CAUSES THE FOLLOWING ACTIONS TO OCCUR. FAULT AND INTERRUPT SENSING IS INHIBITED. A LEVEL POINTER DEFINING THE PROCESS JUST EXITED IS PLACED IN IMPS RESOURCE STACK. THE NUMBER OF THE FAULT OR INTERRUPT BIT THAT CAUSED THE IMP CALL IS PLACED IN VSS. THE PRIVILEGED MODE FLIP-FLOP BIT IS SET. THE PROCESS ENVIRONMENT AT THE BASE OF THE RESOURCE STACK IS ACTIVATED IN THE PROCESSOR.

INSTRUCTION FLOW

```
MODE ← PRIVILEGED
INHIBIT FAULT AND INTERRUPT SENSING
UPDATE FIRST FIVE ENTRIES OF RESOURCE STACK SLICE
PURGE NAME STACK
PURGE PROGRAM CONTROL STACK
PURGE RESOURCE STACK SLICE
RESOURCE STACK SLICE ← BASE RESOURCE STACK SLICE
RESOURCE STACK SLICE ← LEVEL POINTER
PROGRAM CONTROL REGISTER ← IMP ENTRY POINT
END
```

CP 1720-5592

IMP RETURN

ENVIRONMENT AT ENTRY

PRIVILEGED MODE

PURPOSE

THE RESOURCE STACK SLICE. THE PRIVILEGED IMP RETURN ESTABLISHES A PROCESS ENVIRONMENT BY LOADING MODE FLIP-FLOP IS RESET. THIS OPERATOR IS PRIVILEGED.

DESCRIPTION

IMP RETURN CAUSES THE FOLLOWING ACTIONS TO OCCUR. A LEVEL POINTER IS USED TO ACTIVATE THE PROCESS ENVIRONMENT IN THE APPROPRIATE RESOURCE STACK SLICE. FAULT AND INTERRUPT SENSING ARE REINSTATED.

INSTRUCTION FLOW

```
MODE ← NORMAL
REINSTATE FAULT AND INTERRUPT SENSING
UPDATE FIRST FIVE ENTRIES OF RESOURCE STACK SLICE
PURGE NAME STACK
PURGE PROGRAM CONTROL STACK
PURGE RESOURCE STACK SLICE
RESOURCE STACK SLICE ← PROCESS RESOURCE STACK SLICE
PROGRAM CONTROL REGISTER ← PROCESS ENTRY POINT
END
```

LOAD ARITHMETIC VARIANT REGISTER (AVR)

ENVIRONMENT AT ENTRY

VSO CONTAINS CONTROLS FOR AU OPERATORS

PURPOSE

TO ENABLE EXECUTION OF AU OPERATORS.

DESCRIPTION

THE TOP OF VSS CONTAINS THE CONTROLS FOR THE AU OPERATORS TO BE EXECUTED. THE AU TRANSFERS THE CONTENTS OF THE TOP OF VSS TO THE ARITHMETIC VARIANT REGISTER (AVR).

INSTRUCTION FLOW

```
AVR ← [VSO]
DELETE
END
```

CP 1720-5592

SET ATTRIBUTES

ENVIRONMENT AT ENTRY

MU OR MU* IN NSO

PURPOSE

GENERAL.

DESCRIPTION

THE ACCESS ATTRIBUTES IN THE DESCRIPTION IN NSS ARE REPLACED BY THE VALUE FROM VSS.

INSTRUCTION FLOW

```
ADJUST TO VSO
AU TRANSFORM
NSO.ACCESS-FLD ← [VSO]
```

APPENDIX C

GLOSSARY

ABSOLUTE REFERENCE	LEVEL-1 MEMORY ADDRESS.
AC	CONTAINER ADDRESS.
ACCESS ATTRIBUTE	FAULT INDICATOR OR, LEXIC LEVEL, DISPLACEMENT COUPLE.
ADJUST	PERFORM NECESSARY POP AND PUSHES TO CREATE DESIRED ENVIRONMENT.
AE	ELEMENT ADDRESS.
ALLOCATE	BIND A STRUCTURE TO A MORE GLOBAL STRUCTURE.
AS	ATTRIBUTE STACK.
ATTRIBUTE	ACCESS ATTRIBUTE OR, INTERPRETER ATTRIBUTE.
AU TRANSFORM	REPRESENTATION OF INPUT IS CHANGED TO FIT THE FORMAT AND LENGTH SPECIFIED BY THE OUTPUT DESCRIPTION GIVEN IN THE NAME STACK.
BIND	MAP AN UNALLOCATED STRUCTURE ONTO AN ALLOCATED SPACE.
BRANCH DESCRIPTOR	REFERENCE TO NON-RETURNABLE PROGRAM TO EXECUTE.
CCF	COROUTINE CONTROL FIELD.
CD	COROUTINE DISPLAY.

CHANNEL NUMBER	NUMERICAL DESIGNATION OF COMMUNICATION LINK.
CHILD	LESS-GLOBAL PROCESS.
CONTROL VARIABLE	ORIGINALLY SPECIFIED NUMBER DETERMINING NUMBER OF TIMES LOOP IS TO BE PERFORMED.
COROUTINES	ROUTINES WHICH EXIST CONCURRENTLY AND ARE EXECUTED SYNCHRONOUSLY.
D	DISPLACEMENT.
DEALLOCATE	REMOVE A STRUCTURE FROM A MORE-GLOBAL STRUCTURE. THE SPACE IS RETURNED TO UNASSIGNED SPACE.
DISPLAY	STACK-VECTOR WHOSE ELEMENTS DESCRIBE SEGMENTS OF THE NAME STACK
DPCR	DESCRIPTOR PROGRAM CONTROL REGISTER.
DPCS	DESCRIPTOR PROGRAM CONTROL STACK.
EVALUATE	DEVELOP TERMINAL REFERENCE FROM A NAME.
F	FUNCTION.
FAULT	PROCESS DEPENDENT INTERRUPT DETECTED BY THE PROCESSOR AS PART OF EXECUTION.
FETCH	TRANSFER FROM MEMORY TO PROCESSOR.
INDIRECT	METHOD OF ACCESSING NORMALLY INACCESSIBLE ELEMENTS BY SECONDARY ADDRESSING.
INTERRUPT	METHOD FOR THE MODULES OF THE SYSTEM TO COMMUNICATE WITH EACH OTHER.
INTERRUPT DRIVER	DEVICE WHICH CAUSES INTERRUPT BIT TO BE SET.
L	LEVEL - TOPMOST LEVEL OR CURRENT PROCESS ACTIVATED.

LC	CONTAINER LENGTH.
LF	ELEMENT LENGTH.
LL	LEXIC LEVEL.
MU	LEVEL-1 MEMORY.
MU*	BUFFERED STACK.
NORMAL MODE	INTERRUPTABLE STATE OF PROCESSOR, WHILE PERFORMING NON-CONTROL OPERATIN NON-CONTROL OPERATIONS.
NSS	NAME STACK SLICE SEGMENT.
P	PROCEDURE.
PARENT	MORE-GLOBAL PROCESS.
PCR	PROGRAM CONTROL REGISTER.
PCS	PROGRAM CONTROL STACK.
POP	MOVE ONE ENTRY FROM LEVEL-1 TO BUFFER OR, MOVE ONE ENTRY FROM STACK TO REGISTER.
POSITIVE SEQUENCE	NS GOES TO AS, DO POSITIVE SEQUENCE (SEE APPENDIX A), RETURN AS TO NS.
PRIVILEGED MODE	NON-INTERRUPTABLE STATE OF PROCESSOR WHILE PERFORMING CONTROL OPERATIONS.
PUSH	MOVE ONE ENTRY FROM BUFFER TO LEVEL-1 OR, MOVE ONE ENTRY FROM REGISTER TO STACK.
R	RESOURCE STACK.
REFERENCE	DESCRIPTOR.
RESTORE	RETURN TO SAVED VALUE.
RL	REFERENCE TO RESOURCE STACK.
RSB	RESOURCE STACK BUFFER.

RSS	RESOURCE STACK SLICE SEGMENT.
SUBROUTINE CALL	CALL TO RETURNABLE PROGRAM WITHOUT PARAMETERS WHICH DOES NOT RETURN A RESULT.
TERMINAL REFERENCE	LEAST POSSIBLE RELATIVE ADDRESS.
UPDATE	MODIFY ORIGINAL TO AGREE WITH NEW ENVIRONMENT OR, MODIFY COPIES IN LEVEL-1 TO AGREE WITH ENTRIES IN BUFFERED STACK.
V	VALUE STACK.
VSS	VALUE STACK SLICE SEGMENT.
[ ]	CONTENTS OF.