TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

MODULO

2^{16}

ADDRESS - ALWAYS 16 BITS (ON CHAR. DIGIT BOUND)

BINARY TO DECIMAL

DECIMAL TO BINARY

TABLE OF CONTENTS

TABLE OF CONTENTS

PREFACE

This specification provides the complete, comprehensive description of the system structure; of the arithmetic, logical, string, branching, state switching and input/output (I/O) operations; and of the interruption and fault handling mechanisms that comprise the MS-3 S-Machine.

RELATED SPECIFICATIONS

The information contained in the following documents is pertinent to this specification.

MS-3 Architecture Notes #1 - #12.

1 GENERAL DESCRIPTION

The system is designed to operate with a Master Control Program (MCP) that coordinates and executes all I/O intructions, handles exception conditions, and supervises scheduling and execution of multiple programs (i.e., time multiplexing of processes).

A program consists of one or more source modules. Source modules consist of symbolic descriptions of algorithms (procedures) and data areas (module "own", process global and procedure local). Source modules are compiled into object modules. A process is the execution of a program and is characterized by a sequential flow of control through invoked object modules.

MS-3 provides facilities for efficient switching from one process to another, traversing through modules, relocation of processes in processor memory and for storage protection.

2 ARCHITECTURAL CHARACTERISTICS

Some of the architectural characteristics of the MS-3 S-Machine are:

Decimal addressing to the digit.

Decimal arithmetic.

Addressable S-registers.

Indirect addressing.

Facilities for modular programming.

COBOL type string operations on 4-bit and 8-bit units.

FORTRAN type storage unit oriented operations.

Storage protection via segmentation.

3 SYSTEM STRUCTURE

3.1 PROCESS AND MODULE ADDRESSING ENVIRONMENTS

An environment consists of logical storage partitions called segments. The symbolic name of a segment is called a segment descriptor. Segment descriptors are contained in either an environment segment dictionary or the System Segment Dictionary.

The segments of an environment may be addressed relative to the base of the environment segment dictionary (called local addressing) or they may be addressed relative to the base of system memory (called global addressing).

An addressing environment is characterized (defined) by its environment segment dictionary.

See Figure 3-1 for the logical representation of an environment.

3.1 PROCESS AND MODULE ADDRESSING ENVIRONMENTS (Continued)

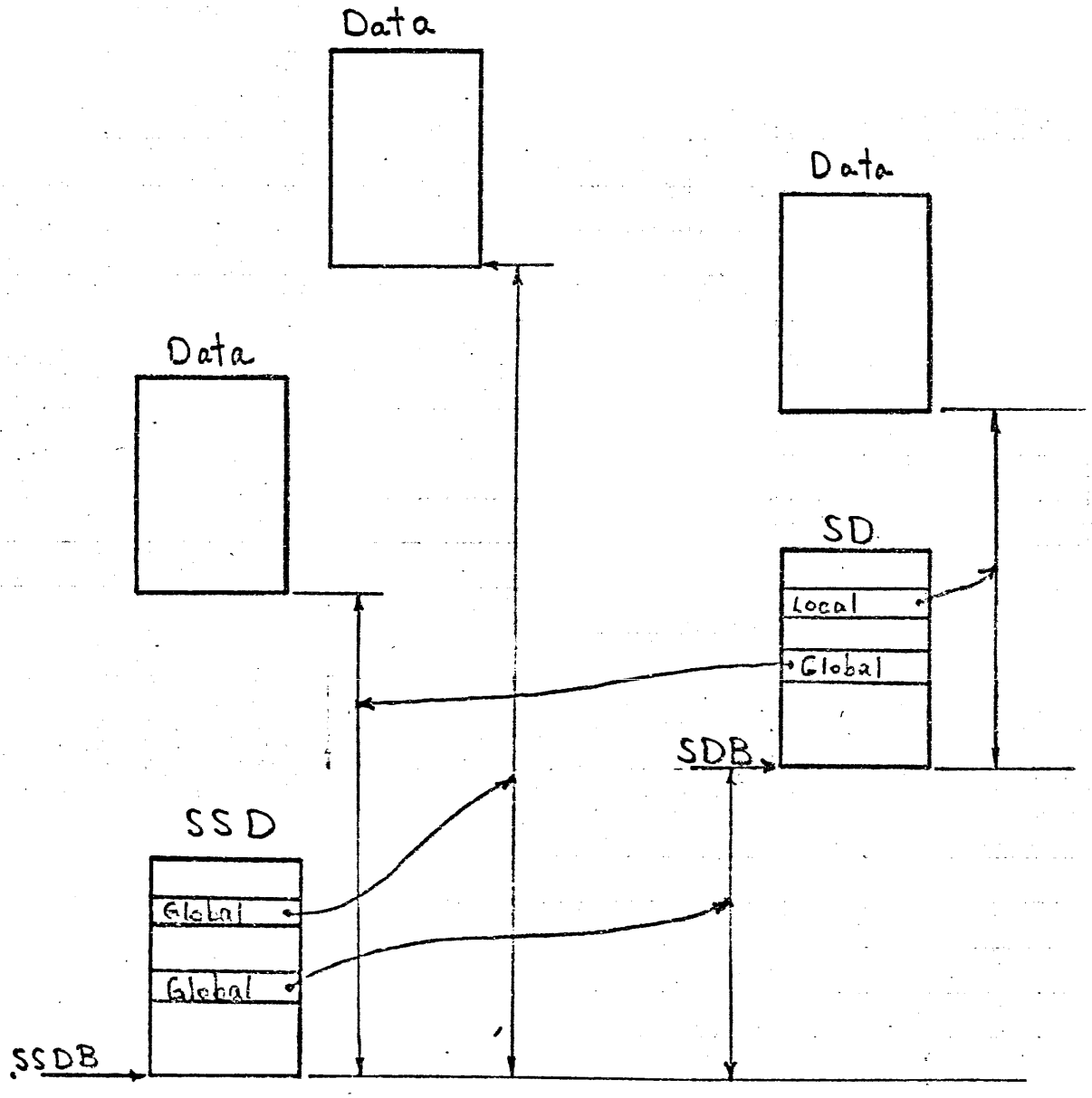


FIGURE 3-1. The Logical Representation Of An Environment

3.1.1 Process Environments

Each process has associated with it a process static environment and a process dynamic environment.

The process static environment consists of segments which are global to the process. These include a process stack.

The process dynamic environment consists of segments of data which have been passed from modules other than the current one.

3.1.2 Module Code (Read Only) Environment

A code environment contains segments of code and read-only data for a module.

3.1.3 Module Data Environment

A data environment contains the static, as opposed to parametric, data segments of a module.

3.2 SYSTEM ADDRESSING STRUCTURE

3.2.1 System Segment Dictionary

The System Segment Dictionary (SSD) defines and protects addressable areas of memory via descriptors. This SSD serves as a global collection of descriptors which primarily, but not exclusively, define environment segment dictionaries. The SSD base (SSDB) is the first (lowest) valid address discovered by the processor at initial load time. The SSDB value is known by the S-Machine at all times.

3.2.2 Segment Dictionaries

Segment dictionaries characterize environments. An environment segment dictionary either locates or defines and protects addressable areas of memory via segment descriptors. Environment segment dictionaries are described by entries in the SSD, thus, the base of an environment segment dictionary is SSDB relative.

3.2.3 Segment Descriptors

Segment descriptors are word addressed (i.e., on modulo 10 digit boundaries). Segment descriptors contain segment dictionary base (SDB) relative addresses (modulo 100 digits) or SSDB relative addresses or SSD indexes.

Segment descriptors are ten digits long and have formats defined by the following fields:

1) descriptor type

The descriptor type field is two bits long. The descriptor type may designate that the descriptor is:

a) a local (segment dictionary) base relative address (LOCAL)

b) a global (SSD) base relative address (GLOBAL)

c) a copy descriptor (an address relative to a SSD entry) (COPY)

d) a miscellaneous format (MISC)

2) usage type

The usage type field is two bits long and the values depend on the descriptor type.

If the descriptor type is LOCAL or GLOBAL then the usage type may designate that the area the descriptor defines is:

a) Read Only data

Access to the described area is limited to a read operation. Attempts to access the area in some other way will result in an access fault (protection violation).

b) Read/Write data

The area may be read from or written into. Other forms of access will generate a fault.

3.2.3 Segment Descriptors (Continued)

c) Execute Only code

Data (code) within the area may only be executed. Read or write access will generate a fault.

d) Read/Execute code

Similar to Execute Only but only a write access will generate a fault. This allows Read Only data in a code segment.

If the descriptor type is COPY then the usage type may designate that the descriptor points to:

a) a segment descriptor of any type

b) a segment dictionary address

c) a module code segment address (is the first word of a module entry point pair)

If the descriptor type is MISC then the usage type may designate:

a) an absent memory area (rest of descriptor is software defined)

b) a module entry point data address and code offset (the second word of an entry point pair)

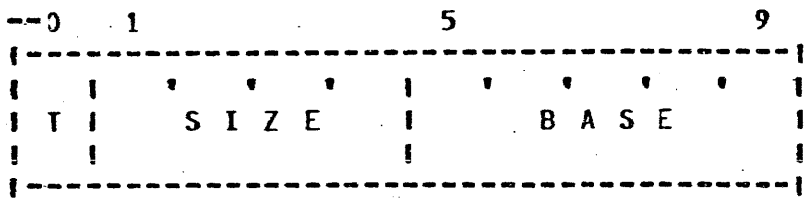
Table 3-1 designates the codes for the various type combinations.

3.2.3 Segment Descriptors (Continued)

desc type	descr type code	usage type code	symbol	meaning
MISC				
	0 0	0 0	ABS	absent - area not in physical mem
	0 0	0 1	-	reserved
	0 0	1 0	EP2	2nd half of entry point pair
	0 0	1 1	-	reserved
LOCAL				
	0 1	0 0	L/R	local, Read Only (SDB relative)
	0 1	0 1	L/R/W	local, Read/Write
	0 1	1 0	L/E	local, Execute Only
	0 1	1 1	L/E/R	local, Execute/Read Only
GLOBAL				
	1 0	0 0	G/R	global, as above (SSDB relative)
	1 0	0 1	G/R/W	"
	1 0	1 0	G/E	"
	1 0	1 1	G/E/R	"
COPY				
	1 1	0 0	SEG	copy of a segment descriptor
	1 1	0 1	SD	" dictionary desc
	1 1	1 0	EP1	first half of an entry point pair
	1 1	1 1	-	reserved

TABLE 3-1. Segment Descriptor Types.

3.2.3.1 LOCAL/GLOBAL Segment Descriptors



Type Fields (digit 0)

For LOCAL/GLOBAL descriptors the type fields (T = Descriptor Type - eight-bit and four-bit; Usage Type - two-bit and one-bit) specify how to form an address and what operations are permitted upon the data located at that address.

Size Field (digits 1-4)

The size field specifies the length of the described area in units of 100 digits (the two low-order digits are implied and have a value of zero).

0000 means a length of zero.

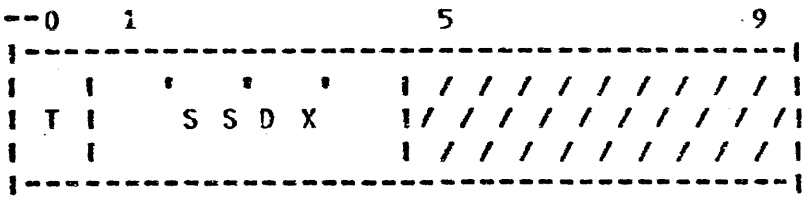
9999'00' is the largest segment size in digits.

Base Field (digits 5-9)

The base field specifies the relative location of the described area. A base represents a seven digit number; the low-order digits are implied and have a value of zero. If the type is LOCAL, the beginning of the area is computed relative to the base of the relevant segment dictionary (i.e., the segment dictionary containing the descriptor). If the type is GLOBAL, the beginning of the area is computed relative to the base of the SSD.

3.2.3.2 COPY Segment Descriptors

SEGMENT DESCRIPTOR ADDRESS



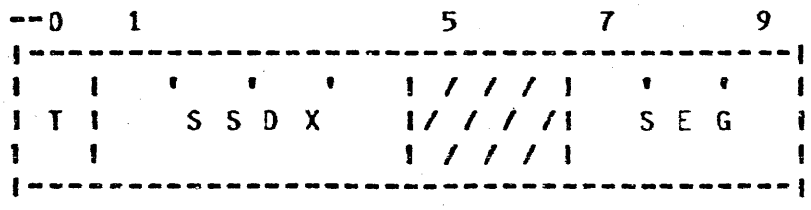
Type Fields (digit 0)

This descriptor type represents a "copy" of a descriptor in the SSD.

SSD Index Field (digits 1-4)

The SSD index (SSDX) field contains a (word) index into the SSD, where the "real" descriptor is to be found. Segment numbers are not applied to this type.

SEGMENT DICTIONARY ADDRESS



Type Fields (digit 0)

This descriptor type represents a "copy" of a descriptor in a segment dictionary.

SSD Index Field (digits 1-4)

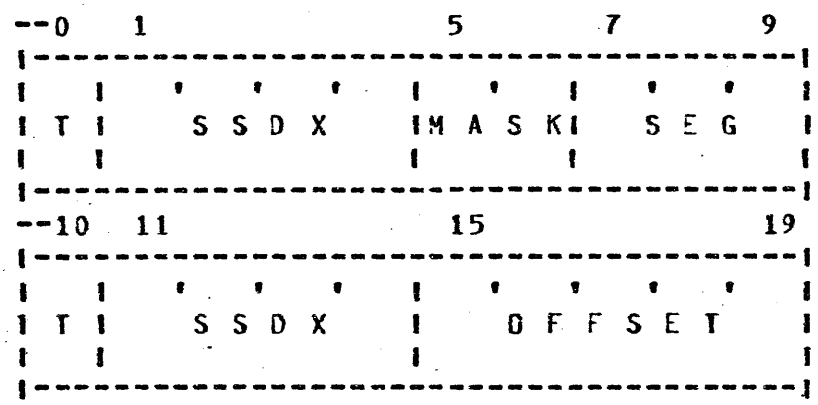
The SSDX field contains a (word) index into the SSD for a descriptor that defines where the segment dictionary of the "real" descriptor is to be found.

Segment Number (digits 7-9)

SEGMENT DICTIONARY ADDRESS (Continued)

The segment number (SEG) is applied to the segment dictionary descriptor to obtain the "real" descriptor.

ENTRY POINT ADDRESS



Entry points have to do with inter-module procedure call and branching. They must occur in pairs.

Type Field (digit 0)

Digit 0 designates an entry-point-1 (EP1) type of descriptor.

SSDX Field (digits 1-4)

This field identifies the descriptor for the next module's code segment dictionary.

MASK Field (digits 5-6)

This field will specify an interrupt and fault mask, etc.

SEG Field (digits 7-9)

This field contains the segment number of the code segment. This segment is relative to the segment dictionary specified by the SSDX field.

ENTRY POINT ADDRESS (Continued)

Type Field (digit 10)

Digit 10 designates an entry-point-2 (EP2) type of descriptor.

SSDX Field (digits 11-14)

This SSDX identifies the descriptor for the next module's data segment dictionary.

Offset Field (digits 15-19)

This field contains the (digit) offset within the code segment of EP1 at which execution will next commence. The size of this field means that only the first 99999 digits of a code segment may be ENTERed (directly) via an entry point pair.

3.3 PROCESSING STRUCTURE

A process is the execution of a prog am and is characterized by the sequential flow of control through invoked object modules. Module invocation is accomplished via the process sequence control instructions (Procedure-call, Branch). Called procedures may be located in the current (intra) module or in some external (inter) module.

The system provides for 1) passing parameters to the called procedure, 2) saving the current environment (as appropriate), 3) acquiring a new environment if a procedure external to the current module is being called, 4) accessing data passed by reference, and 5) restoring the correct environment when the procedure is exited.

Intra/inter-module procedure entry history is maintained in a Process Stack. The information contained in this area includes 1) the type of call (intra/inter module), 2) the instruction pointer (segment number and offset), and 3) the location of the previous history information.

A process is specified by naming a process state record (PSR). Each PSR contains the information necessary to reinitiate the process on the processor.

At any time the addressing range or scope of a module being executed as part of a process is defined directly by three environments, one pseudo environment and a set of addressable registers and indirectly by another environment.

These are respectively:

1) Process static environment. This environment is characterized by the process static segment dictionary.

2) Module data environment. This environment is characterized by the module static data segment dictionary. Associated with this environment is a current data segment which allows a short form of operand address description.

3) Module code environment. This environment is characterized by the module code segment dictionary. Associated with this environment is a current code

3.3 PROCESSING STRUCTURE (Continued)

segment which allows a short form of branch address description.

4) Procedure parameters and local data (this is the pseudo environment). Storage for these is allocated from a Process Stack which is located via the process static segment dictionary.

5) The addressable registers are addressed via a register type operand description and are located via the process state record.

6) Process dynamic environment. This environment is characterized by the process dynamic segment dictionary. Segment descriptors for parameters passed by reference are actually located in the process dynamic environment and are referenced via corresponding operand descriptions in the Process Stack.

See Figure 3-2 for a diagram of the various addressing environments in a process's direct addressing space.

3.3 PROCESSING STRUCTURE (Continued)

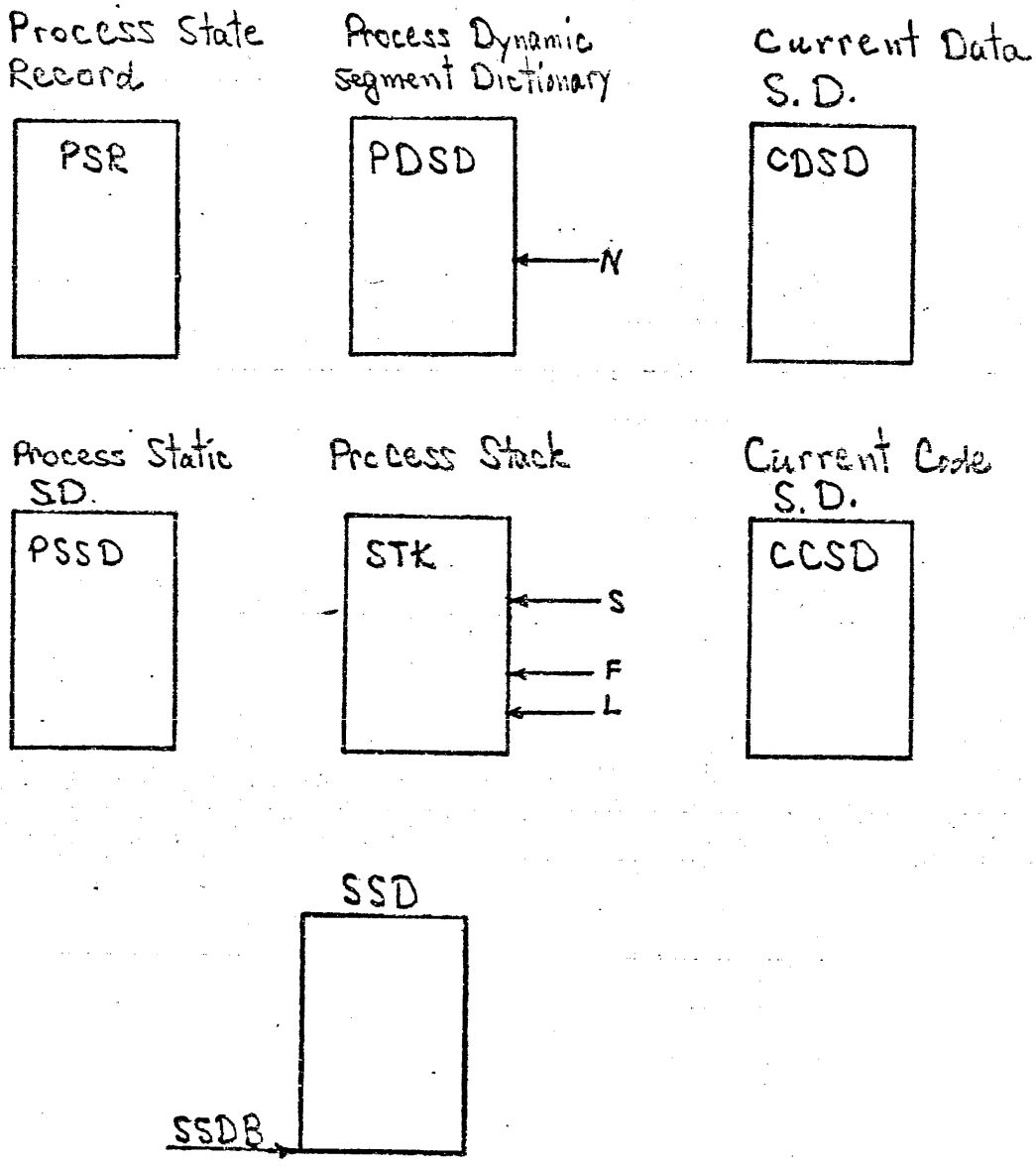


FIGURE 3-2. Process Direct Addressing Space.

3.3.1 Process Stack

By definition, segment zero (the zero'th entry) of the process static segment dictionary defines the Process Stack area. Space for history linkage, parameters and local variables (temporary storage) is allocated in a LIFO manner in this stack. Addressing is relative to the last information deposited by the Mark-stack instruction. This information is termed a Mark Stack Control Word (MSCW). Local (L), historical (F) and top-of-stack (S) pointers are associated with the Process Stack. L, F and S are manipulated by the Mark-stack (MKST), Procedure-call (ENTER) and Procedure-return (EXIT) operators. The S pointer is also manipulated via the TOS operand type.

Bounds checking on the Process Stack and the process dynamic segment dictionary must be performed whenever their stack pointers are changed. Such checking should use an equal condition only. Thus the fault-handling procedure will not get bounds checks.

3.3.2 Process State Record

The Process State Record (PSR) is not addressable by the process. This area contains all the variables that are needed for process initiation.

3.3.3 Process Static Segment Dictionary

The process static segment dictionary is a process wide segment dictionary (referenced by environment number one in an operand description - see section 3.4.2). The process static segment dictionary does not change during the life of the process. This segment dictionary describes areas global to the process.

3.3.4 Module Static Data Segment Dictionary

A module static data segment dictionary defines data areas (referenced by environment number two in an operand description) of current interest. The module static data segment dictionary is dynamic in that the current segment dictionary may be changed via the ENTER, EXIT or inter module branch instructions.

3.3.5 Module Code Segment Dictionary

A module code segment dictionary defines those code segments (referenced by environment number three in a transfer of control intruction) of current interest. The module code segment dictionary is dynamic in the same fashion as the module data segment dictionary.

3.3.6 Dynamic (Procedure Local) Data And Parameters

Dynamic data and procedure parameters are placed in the Process Stack along with the procedure history. These variables are referenced by environment number zero and no segment number in an operand description.

3.3.7 Process Dynamic Segment Dictionary

The process dynamic segment dictionary is a segment dictionary which is not directly accessible by a process, but is used to contain the address of parameters passed by reference. This segment dictionary is referenced by an environment number of zero in an operand description (which contains a segment number) that was constructed in the Process Stack by a Push-name (STUFF) instruction. The

3.3.7 Process Dynamic Segment Dictionary (Continued)

space allocation within this segment dictionary operates in a stack or LIFO manner. The addressing is relative to its base. The MKST and STUFF instructions cause the name (N) pointer to be manipulated. The process dynamic segment dictionary is not addressable via the process static segment dictionary.

3.3.8 Addressable S-registers

The S-machine provides sixteen addressable registers for use as index registers or accumulators. Each register is ten digits long. Double length operands may extend over two registers.

3.4 INSTRUCTION FORMATS

Instructions are composed of an Operator Description followed by zero, one, two or three Operand Descriptions. Instructions consist of byte modulo syllables which must begin on an even address.

3.4.1 Operator Description

The first byte of an instruction contains the Operator Description which is encoded and defines the operation (Op) to be performed. All unassigned Op codes are reserved and will cause a program fault if encountered. Each Op code implies the number of Operand Descriptions which follow.

3.4.2 Operand Description

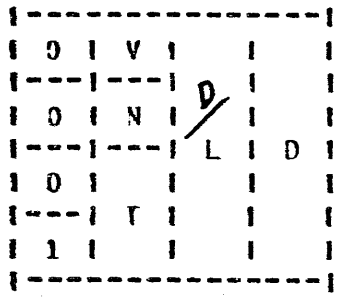
Each Operand Description consists of a Primary Syllable, optionally followed by one or more Extension Syllables. The Primary Syllable may vary in length from one to four bytes. Each Extension Syllable is two bytes long.

3.4.2.1 Primary Syllable

The Primary Syllable may be one of four categories: Literal (LIT), Register (REG), Top-Of-Stack (TOS) and Reference (REF). The primary syllables for LIT, REG and TOS are two bytes long. The primary syllable for REF may vary in length from one to four bytes. The first digit of each primary syllable specifies which category the operand belongs to, as well as giving further formatting information for the REF category.

The formats of the Primary Syllables for Literal, Register, Top-Of-Stack and Reference Operand Descriptions are as follows:

LITERAL (LIT)



The first digit specifies a literal operand.

V specifies variable or fixed-length operand.

N specifies numeric or string operand.

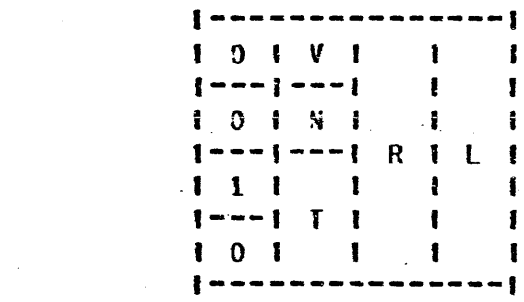
T is the type, selecting the appropriate ro from the table in section 3.5.

L is the length of the literal in digits.

D is the first digit of data in the literal.

The number of extension syllables which follow is determined by the formula ((L-1) DIV 4).

REGISTER (REG)



The first digit indicates that this is a register operand.

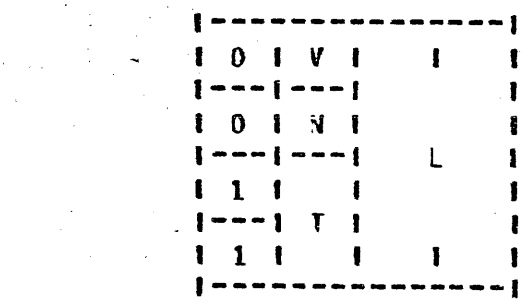
V, N and T are the same as for LIT.

L is the length of the operand in digits.

R specifies which one of the 16 general-purpose registers contains the operand.

No Extension Syllables follow.

TOP-OF-STACK (TOS)



The first digit indicates that this is a top-of-stack operand.

V, N and T are the same as for LIT.

L is the length of the operand in digits.

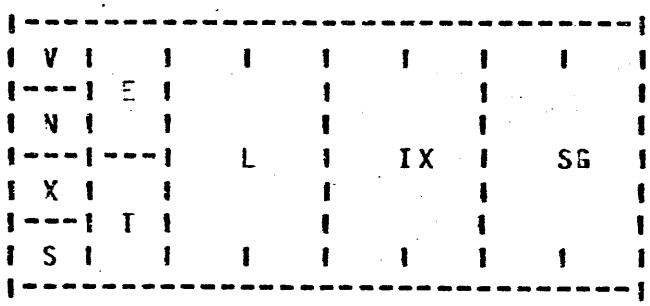
The address is assumed to be the current top of the procedure call stack. Use of this form causes the TOS

TOP-OF-STACK (TOS) (Continued)

pointer to be incremented past the end of the operand (TOS := TOS + L).

No Extension Syllables follow.

REFERENCE (REF)



Both V and N may not be zero.

Each of L, IX and SG may or may not be present. Thus eight possible formats exist, of one, two, three or four bytes in length.

X specifies the presence of the IX Field.

S specifies the presence of the SG Field.

V specifies the presence of the L Field as well as specifying whether the operand is variable or fixed length.

N, T and L are the same as for TOS.

E specifies the Environment (see section 3.1).

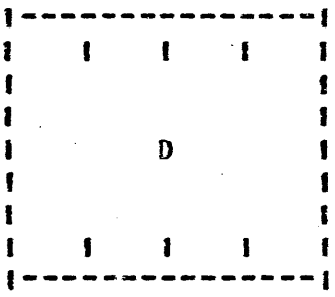
IX specifies the indexing or indirection (see section 3.4.2.4).

SG specifies the segment number.

The number of Extension Syllables is variable (see section 3.4.2.2).

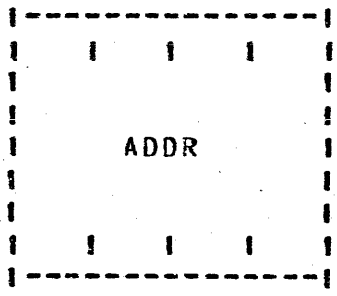
3.4.2.2 Extension Syllables

Each Extension Syllable for a Literal operand is of the form:



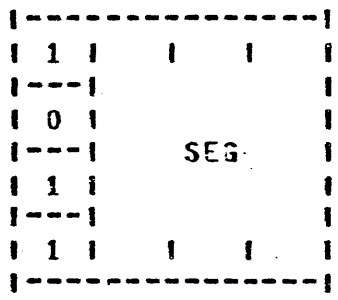
Extensions for Reference operands may be of type Address, Segment, Offset, Length or Extension. Extension Syllables are fetched repeatedly until an Address syllable is encountered.

ADDRESS (ADDR)



If the first digit of a Reference Extension is 0 through 9, the type is Address, and contains a full four digits of address specification. This address is relative to the context established by the rest of the Operand Description.

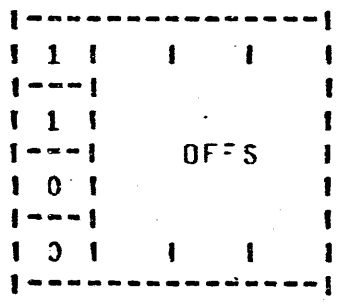
SEGMENT (SEG) EXTENSION



A first digit of hex 'B' specifies that the Extension is a Segment type.

SEG field specifies the segment number and overides the S field of the Primary Syllable.

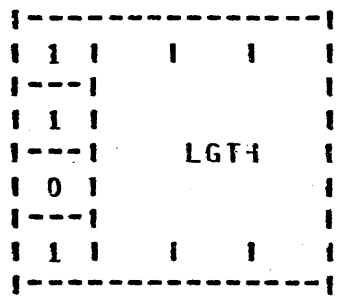
OFFSET (OFFS) EXTENSION



A first digit of hex 'C' specifies that the Extension is an Offset type.

OFFS field specifies three digits of offset to be added to the address after all indirection has taken place.

LENGTH (LGTH) EXTENSION

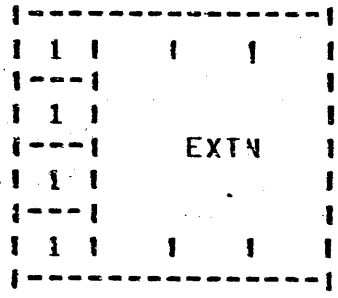


A first digit of hex 'D' specifies that the Extension is a Length type.

LGTH field specifies three digits of operand length.

Values of V and N are set to V = 1, N = 0 regardless of the values specified in the Primary Syllable. The value of the LGTH field overrides the L field of the Primary Syllable.

FIELD EXTENSION (EXTN)



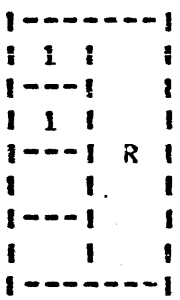
A first digit of hex 'F' specifies that the Extension Syllable is a Field Extension type.

EXTN field specifies three digits which are to be used as the high-order digits of the Extension Syllable which immediately follows this syllable.

EXTN may precede the ADDR, OFFS, or LGTH syllables.

3.4.2.3 Indirect Length

Indirect Length may be specified in the L field of either a TOS or REF operand. The format is as follows:

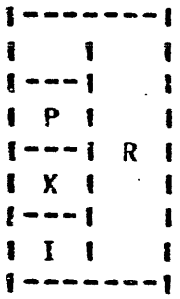


The two high-order bits of the first digit specify that length indirection is to be used.

R specifies one of the general purpose registers which will contain the actual length to be used. The entire register is used (excluding sign), but the length is checked against the maximum allowed for the specified data type.

3.4.2.4 Indirect Addressing And Indexing (IX Field)

The presence of an IX field in the Primary Syllable allows specification of indirect addressing and/or indexing. The IX field is formatted as follows:



I = 1 specifies that indirect addressing is to be applied.

X = 1 specifies that indexing is to be applied.

R specifies one of the general-purpose registers which is to be used for an index value.

3.4.2.4 Indirect Addressing And Indexing (IX Field) (Continued)

P specifies pre-indexing or post-indexing. If P = 0 (pre-indexing) the contents of the specified register are added to the address of the operand before any indirect address fetch takes place. If P = 1 (post-indexing) the contents of the specified register are added to the final address of the operand after all indirection has been resolved.

When indirect addressing is specified, the operand address references a memory location from which an operand description is to be fetched (Primary Syllable and Extension Syllables). The specified location must be an even address. All formats and requirements for Operand Descriptions in an instruction apply to indirectly fetched Operand Descriptions as well.

3.5 DATA TYPES

The following data types are primitive to the S-Machine. The column of the table is selected based on the V and N bits of an operand description and the row is specified by the type field of the same operand description.

Type Code	String	Variable Numeric	Fixed Numeric
	V=1,N=0	V=1,N=1	V=0,N=1
0	HX	UN	RS
1	*	SN	FI
2	CH	UD	RD
3	*	SD	null

Where * means illegal type.

General characteristics are:

String

Left justification, padding with blanks, right truncation and no data checking.

Variable Numeric

Right justification, padding with zeros, no store on overflow and data is validated. The maximum allowable field length for arithmetic operations is twenty units.

Fixed Numeric

No store on overflow.

3.5 DATA TYPES (Continued)

The data type may be "null" in any of several cases:

a) The operator requires only an address.

b) The type and length is specified at a prior or subsequent level in a chain of indirect addresses.

c) The type and length of a previous operand is to be used.

The following subsets of the data types are referenced collectively in this document:

string (S)	= HX, CH
numeric (N)	= UN, UD, SN, SD, FI, RS, RD
floating-point (R)	= RS, RD
integer (I)	= UN, UD, SN, SD, FI
signed-integer	= SN, SD, FI
unsigned-integer	= UN, UD
eight-bit	= UD, SD, CH
four-bit	= UN, SN, FI, RS, RD, HX
eight-bit numeric	= UD, SD
four-bit numeric	= UN, SN, FI, RS, RD
four-bit integer	= UN, SN, FI

3.5.1 Unsigned Numeric (UN)

Four-bit units which may contain only BCD values 0 through 9. Used only for arithmetic operations, the validity of the BCD digits is checked, the number is considered to be right justified and there is no store on an overflow condition. The maximum length is 20 digits.

3.5.2 Signed Numeric (SN)

Four-bit units where the leftmost unit is interpreted as: C = plus, D = minus, other values are illegal. The remaining units are the same as UN. The maximum length is 20 digits plus the sign digit (21 digits total).

3.5.3 Unsigned Display (UD)

Eight bit units which may only contain EBCDIC codes for zero through nine, except that leading blanks are treated as zeros. Used only for arithmetic operations, the units are validated to be "F0" through "F9", the number is considered to be right justified and there is no store on an overflow condition. The maximum length is 20 units.

3.5.4 Signed Display (SD)

The same as UD, except that the high-order four bits (zone) of the leftmost unit is treated the same as the leftmost unit in SN. The maximum length is 20 units.

3.5.5 Hexadecimal (HX)

Four-bit units. Used only for string manipulation operations, all bit combinations are valid, the contents are left justified in the field and truncation occurs from the right. The maximum length is memory size.

3.5.6 Character (CH)

Eight-bit units. Has the same characteristics as HX.

3.5.7 Integer (FI)

The same as SN except that the number of units (length) is fixed at 9 digits plus the sign digit (10 digits total - the same size as the index registers).

3.5.8 Single Precision Real (RS)

This is a unit with one bit signs for the mantissa and exponent (0 = plus, 1 = minus), a six bit exponent (binary power of 10) and the next eight digits are a BCD mantissa. The decimal point is assumed to be to the left of the mantissa. The mantissa is validated to ensure that it contains BCD digits only. The total length is ten digits.

3.5.9 Double Precision Real (RD)

The same as RS except that the mantissa is 18 BCD digits. The total length is 20 digits.

3.6 EXCEPTION MECHANISMS

3.6.1 Fault Structure

Faults are defined to be exception conditions which can be blamed on the currently running process. All are handled by:

1) Generating a MSCW.

2) Inserting one and possibly two parameters into the STK. The first parameter identifies the fault. The second parameter is specific to the kind of fault (see the description of the Absent fault).

3) Performing an ENTER to a standard entry point for the type of fault (e.g., PSSD, 3).

3.6.1.1 Absent Descriptor Fault

This fault occurs if fetch trips across a segment descriptor with a usage type of Absent.

The parameters are:

1) An indication that this is an absent descriptor fault.

2) A COPY descriptor which identifies the offensive segment dictionary entry (this goes in the STK, not the PDSD).

3.6.2 Interrupt Structure

See section 4.1.7.

3.7 INPUT/OUTPUT STRUCTURE

See section 4.1.7.

4 INSTRUCTION SET

The following abbreviations are used for operands in instruction descriptions:

O1, O2 ...	any operands
S1, S2 ...	string operands
N1, N2 ...	numeric operands
R1, R2 ...	real operands
T1, T2 ...	table operands
A1, A2 ...	address operands

Operand types are checked to validate that they are in the appropriate subset allowed for an instruction.

Table 4-1 describes the conversions which take place between operands of different data types. When on operand is a sending (SND) field and one a receiving (RCV) field, the sending operand type is converted into the receiving operand type. When two operands are both sending fields, the internal type conversion for an operation is made as follows:

a) data type precedence is UN, UD, FI, SN, SD, RS, RD, HX, CH

b) the operand whose type is to the left is converted to the type of the other operand.

Instructions which use different types for special purposes (bit manipulation, exponent manipulation, shifts, etc.) are treated as specified in the instruction description.

No receiving field may overlap any sending field.

The codes used in Table 4-1 have the following meanings:

O = overflow protection (no store)
F = float
N = normalize
R = right justify
L = left justify, right truncation
T = truncate fractional part
B = pad with blanks
Z = pad with zeros
S = sign is lost

4 INSTRUCTION SET (Continued)

4 = zones are stripped
8 = zones are set to 'F'
+ = sign is set to plus
C = BCD check on appropriate fields

4 INSTRUCTION SET (Continued)

* R / * C / S ± V / N * / D *	HX	CH	UN	SN	UD	SD	FI	RS	RD
HX	L Z	L B8	0 RZ C	0 RZ+ C	0 8 RZ C	0 8 RZ+ C	0 RZ+ C	0 N Z+ C	0 F N Z+ C
CH	L 4 Z	L 4 B	0 RZ C	0 RZ+ C	0 RZ C	0 RZ+ C	0 4 RZ+ C	0 F 4 N Z+ C	0 F4 N Z+ C
UN	L Z	L B8	0 RZ C	0 RZ C	0 8 RZ C	0 RZ+ C	0 RZ+ C	0 F N Z+ C	0 F N Z+ C
SN	LS Z	LS B8	0 S RZ C	0 RZ C	0 S 8 RZ C	0 8 RZ C	0 RZ C	0 F N Z C	0 F N Z C

TABLE 4-1. The Implied Type Conversions For Move Operations.

4 INSTRUCTION SET (Continued)

* R / * C / S * V / N * / D *	HX	CH	UN	SN	UD	SD	FI	RS	RD
UD	L 4 B Z	L	0 4 RZ C	0 4 RZ+ C	0 RZ C	0 RZ+ C	0 4 RZ+ C	0 F 4 N Z+ C	0 F 4 N Z+ C
SD	LS 4 B Z	LS	0 S 4 RZ C	0 4 RZ C	0 S RZ C	0 RZ C	0 4 RZ C	0 F 4 N Z C	0 F 4 N Z C
FI	LS B8 Z	LS	0 S RZ C	0 RZ C	0 S 8 RZ C	0 8 RZ C	0 RZ C	0 F N Z C	0 F N Z C
RS	LS T B8 Z	LS T	0 S T RZ C	0 T RZ C	0 S T 8 RZ C	0 8 RZ C	0 RZ C	0 N Z C	0 N Z C
RD	LS T B8 Z	LS T	0 S T RZ C	0 T RZ C	0 S T 8 RZ C	0 8 RZ C	0 RZ C	0 N Z C	0 N Z C

TABLE 4-1 (continued). Implied Type Conversions.

4.1 PROCESS SEQUENCE CONTROL

A Process Stack (STK) is maintained as segment zero of the process static segment dictionary. Within this segment, the S-Machine provides a local (pseudo) environment. Procedure call and return operations change the local environment. Two S-Machine registers are used to define this local environment.

The Local (L) register identifies the base of the currently valid local environment. The L register value is always even (i.e., a byte address).

PUSH TOS

The Stack (S) register identifies the first available cell in the Process Stack. It is a digit address.

A third S-Machine register, the History (F - a historical name) register is used in procedure call to reference the most recent history information.

Figure 4-1 illustrates the partioning of the Process Stack.

4.1 PROCESS SEQUENCE CONTROL (Continued)

:

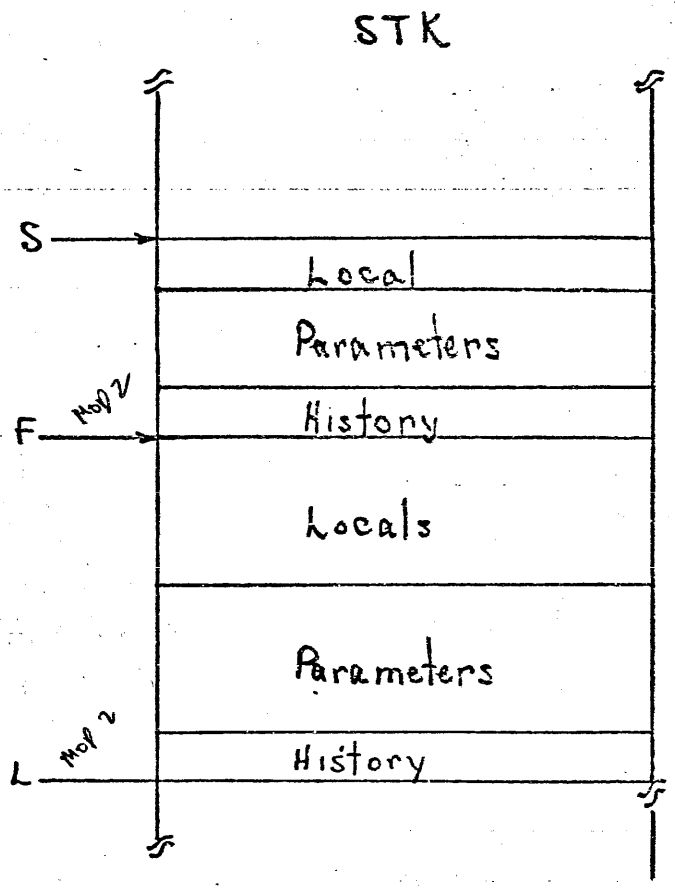


FIGURE 4-1. The Process Stack.

HISTORY

The information pointed to by the L and F registers is ten digits (one word) of history information. This information is placed in the STK by the Mark-stack (MKST) operator as the first instruction in the procedure call sequence. The history word allows the S-Machine to recover the proper (previous) state following a Procedure-return (EXIT) instruction.

PARAMETERS

The next area in the local environment contains any parameters passed to the procedure. Such parameters are placed in the stack by operations which specify the top-of-stack type destination operand. Parameters may also cause information to be placed in the process dynamic segment dictionary (PDSD).

LOCALS

The third area in a local environment may contain local storage for the procedure.

THE PROCESS DYNAMIC SEGMENT DICTIONARY

In order to pass parameters by reference (address) in a secure fashion, the PDSD is used. The PDSD operates in a stack like manner; information is entered as a result of the Push-name (STUFF) and Procedure-call (ENTER) operators and is removed by the EXIT operator. The PDSD is word oriented and data is always on word (modulo 10 digit) boundaries. The S-Machine maintains a Name (N) register to the first free word in the PDSD. Figure 4-2 illustrates the PDSD.

THE PROCESS DYNAMIC SEGMENT DICTIONARY (Continued)

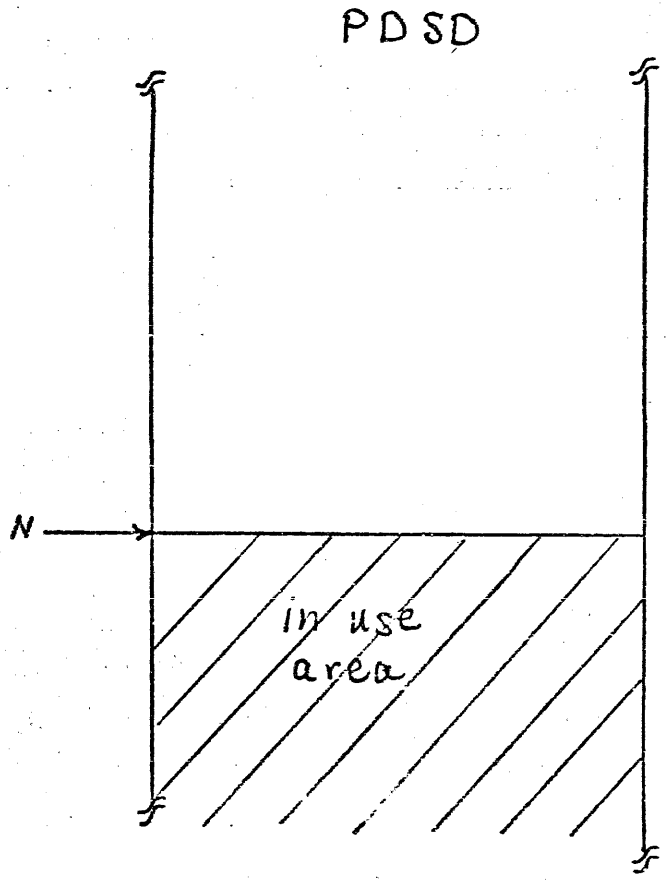


FIGURE 4-2. The Process Dynamic Segment Dictionary.

PROCEDURE CALL SEQUENCE

The normal procedure call consists of:

1) Executing a MKST operator to record appropriate history information (F and S will be changed).

2) Build parameters in the Process Stack (S and possibly N will be changed).

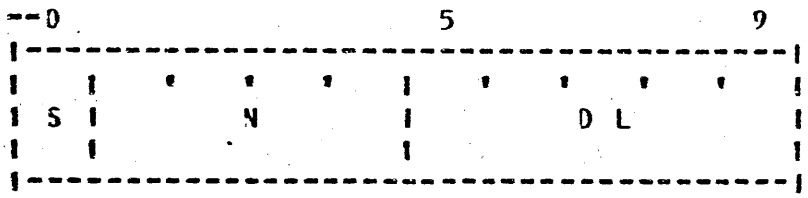
3) Execute an ENTER operator, specifying where to transfer control. ENTER will record necessary return information and establish a new local environment.

4.1.2 Mark-stack

MKST

The MKST operator causes a word (10 digits) of history information to be entered into the Process Stack. The S and F registers are changed.

STK History Word (Markstack Control Word - MSCW) Format



S = State Bits

The eight-bit contains the Mark Stack Flag (MSF). The MSF is set by a MKST operator and reset by an ENTER operator. This flag is used during an EXIT operation.

The four, two and one-bits are reserved.

N = PDSD Pointer

This four digit field contains the value of the N register at the instant of the MKST operation. (Since N is a word address, useful PDSDs are limited to 99990 digits in size.)

4.1.2 Mark-stack (Continued)

DL= Dynamic Link

This link tells how to find the immediately preceding MSCW and how to reset S and F. Since MSCWs are to be on an even address, but S might be odd at the time the MKST is executed, some adjustment is performed. The new value of F is to be S rounded up to a byte boundary (i.e., S plus the low order bit of S - call it Sr). Sr-F is stored in the DL field. The low order bit of S also has to be kept (TBS). Local environments are thus limited to 99998 digits.

The sequence of steps is:

1) Lay down the MSCW at Sr.

2) F := Sr

   S := F + 10

3) MSF is set

The PDSD is not affected. Figure 4-3 illustrates the mark stack sequence.

4.1.2 Mark-stack (Continued)

:

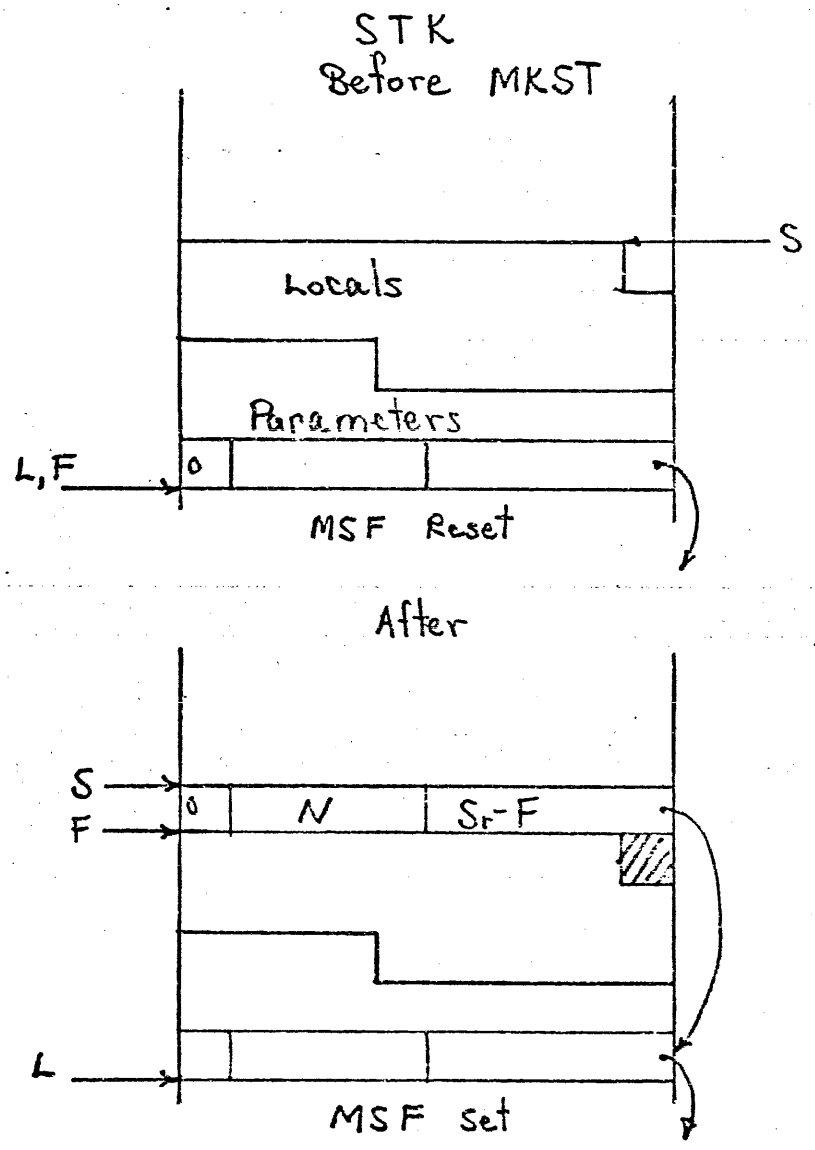


FIGURE 4-3. The Mark Stack Sequence.

4.1.2 Mark-stack (Continued)

Parameter Building Notes:

The F register always references the most recent MSCW; addressing, however, is performed relative to the L register. Thus data items within the current local environment are addressed as before and may be passed as parameters.

Parameters may be placed in the Process Stack above the history information by using the Top-Of-Stack address type as the destination operand. S is adjusted and checked for stack overflow.

If parameters are passed by reference, data may also be entered into the PDSD and N will be adjusted.

Figure 4-4 illustrates a repeated MKST operation and parameter insertion before an ENTER operation is performed. This situation would arise naturally in case of a procedure call, one of whose parameters was the result of a function call.

4.1.2 Mark-stack (Continued)

:

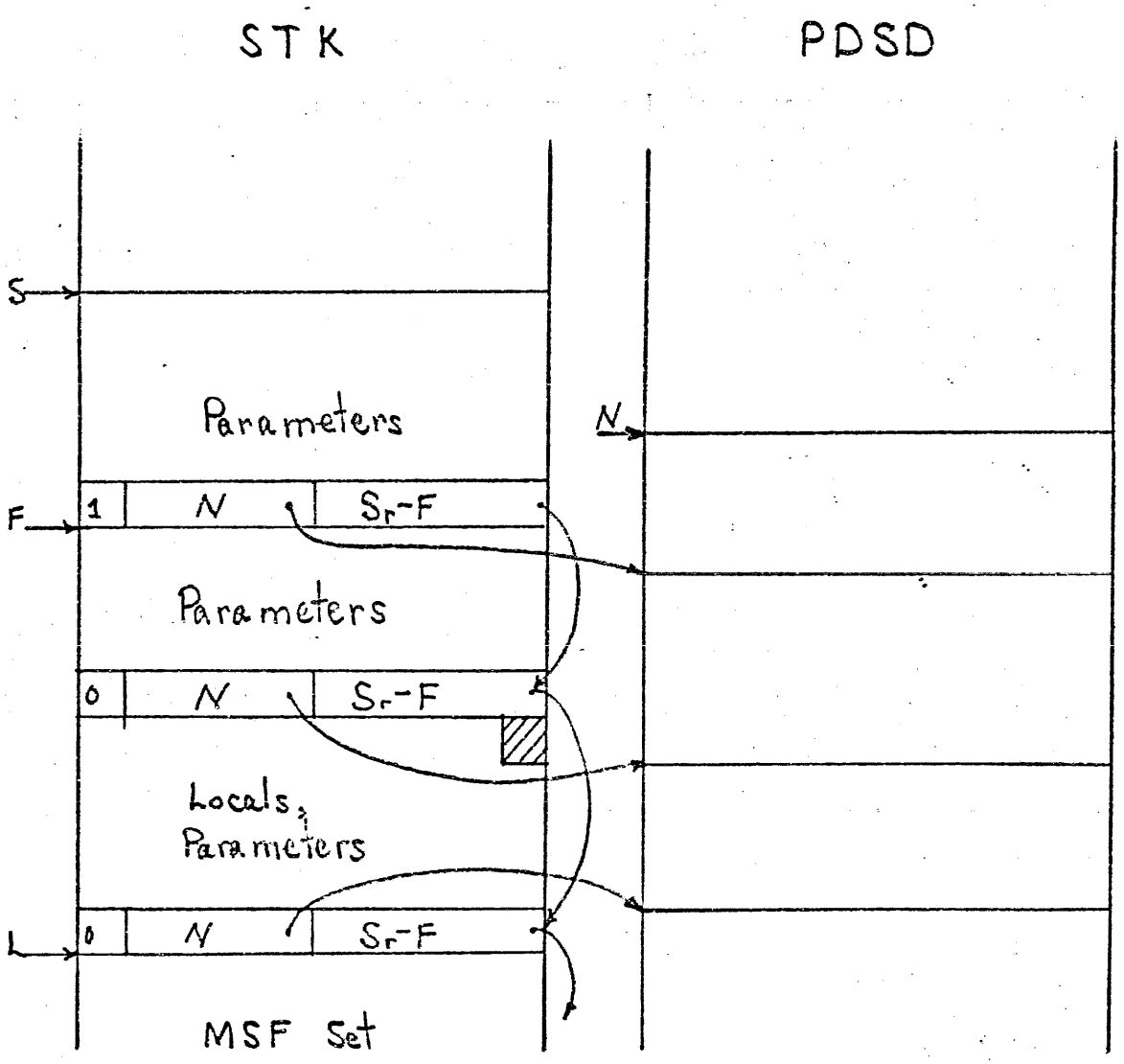


FIGURE 4-4. A Process Stack With Un-Entered MSCWs.

4.1.2 Procedure-call

ENTER A1

An ENTER operator is executed after any procedure parameters have been placed in the STK to:

1) Record the necessary return information.

2) Adjust the local environment and, if necessary, the module data and code environments.

3) Transfer control to the specified address (A1).

The ENTER operator's single operand specifies the location to which execution control is to be transfered.

The sequence of operations is:

1) Operand fetch.

2) Return information storage.

3) Environment updating.

4) Transfer of control.

Operand Fetch

A transfer address may be specified in one of two ways;

a) by a simple address (segment number, offset)

b) by an entry point pair.

Both of these forms might be preceded by various amounts of indirection. The net result of operand fetch will be a code segment and offset. If an entry point is discovered, new code and data environments are specified as well.

Return Information

After operand fetch, two words of return information are deposited in the PDSD at N and N+1. N is set to N+2. The format of these Re-entry Control Words (RCWs) is exactly that of an entry point pair (segment descriptor types EP1

4.1.2 Procedure-call (Continued)

and EP2) containing information appropriate to the current state of the S-Machine. Namely, the code and data segment dictionary SSDXs and the segment number and offset within the code stream of the next instruction following the ENTER.

Environment Updating

If in the course of operand fetch, an entry point descriptor was encountered, the code and data environments are changed to those specified in the entry point. The final sequence before transfering control is:

1) L := F

2) MSF is reset

Transfer Of Control

Control passes to the address generated as the result of fetch (i.e., this address identifies the next code to fetch). Figure 4-5 illustrates the Process Stack state following an ENTER operation.

4.1.2 Procedure-call (Continued)

:

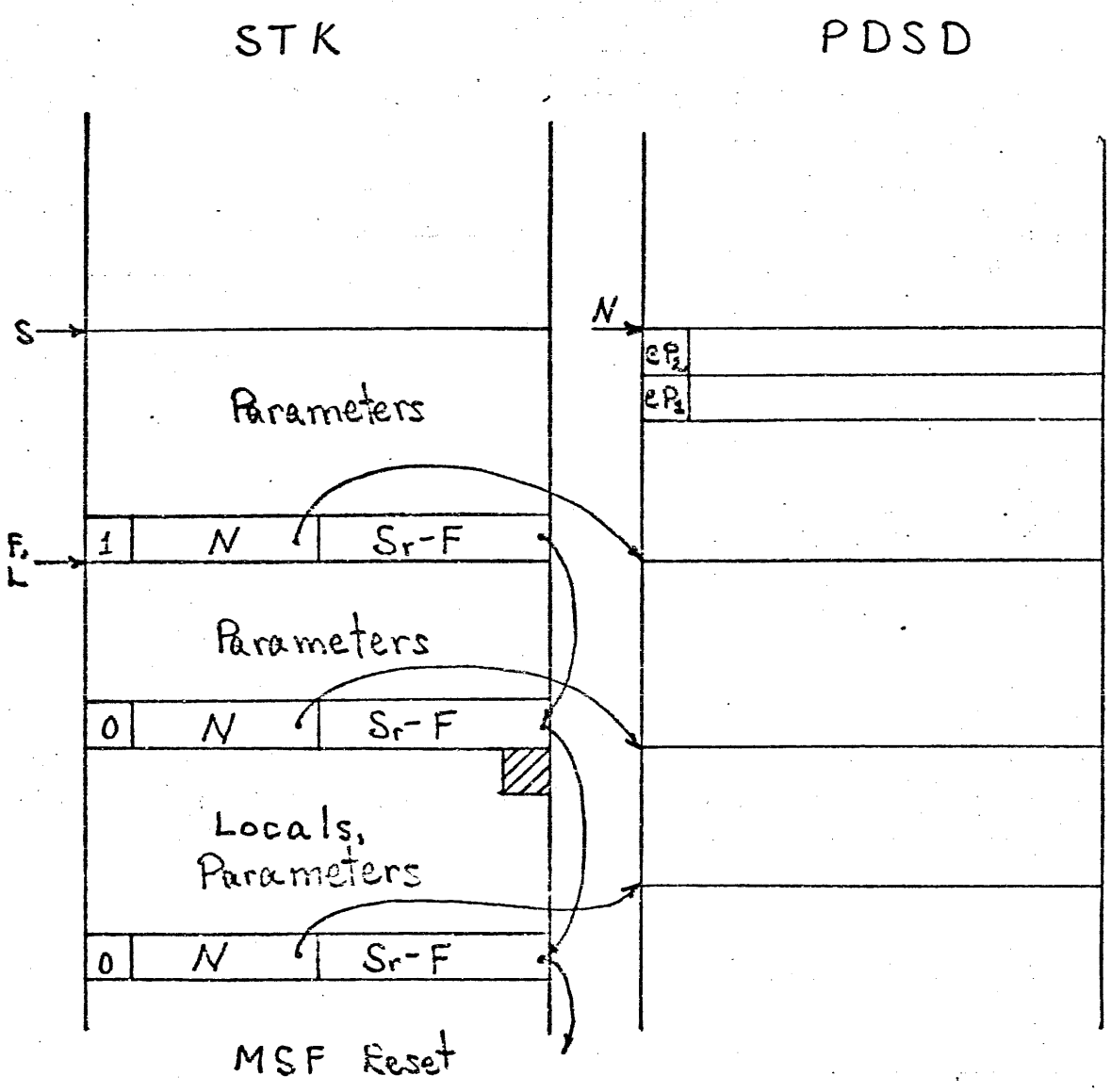


FIGURE 4-5. The Process Stack After An ENTER Operation.

4.1.3 Procedure-return

EXIT

It is EXIT's responsibility to restore the state of the caller.

a) The RCWs at N-2 and N-1 are evaluated to:

1) establish the new code address

2) determine if the code and data environments must be changed (i.e., if the SSDXs in the RCWs differ from the current processor values)

Note: The state of the S-Machine should not be changed until this evaluation is complete. Absent descriptors might be encountered in the process. It is necessary to ensure that the proper machine state is available. The same concept applies during ENTER operand fetches.

b) The appropriate state of the S-Machine is restored from EP1 and EP2 (i.e., the RCWs).

c) The appropriate state of the STK and PDSD is restored as follows:

1) the word pointed to by the L register is accessed (i.e., this is the MSCW corresponding to the current local environment)

2) the N (PDSD) pointer is set to the value in the MSCW

3) F := L - MSCW.DL

   S := F - (odd/even bit)

4) L := L - MSCW.DL

   if the MSCW.MSF bit is set then pick up the MSCW now pointed to by L and repeat step four

4.1.3 Procedure-return (Continued)

d) Control is passed to the code address generated from EP1 and EP2.

Figure 4-6 illustrates the state of the STK and PDSD after this operation.

4.1.3 Procedure-return (Continued)

:

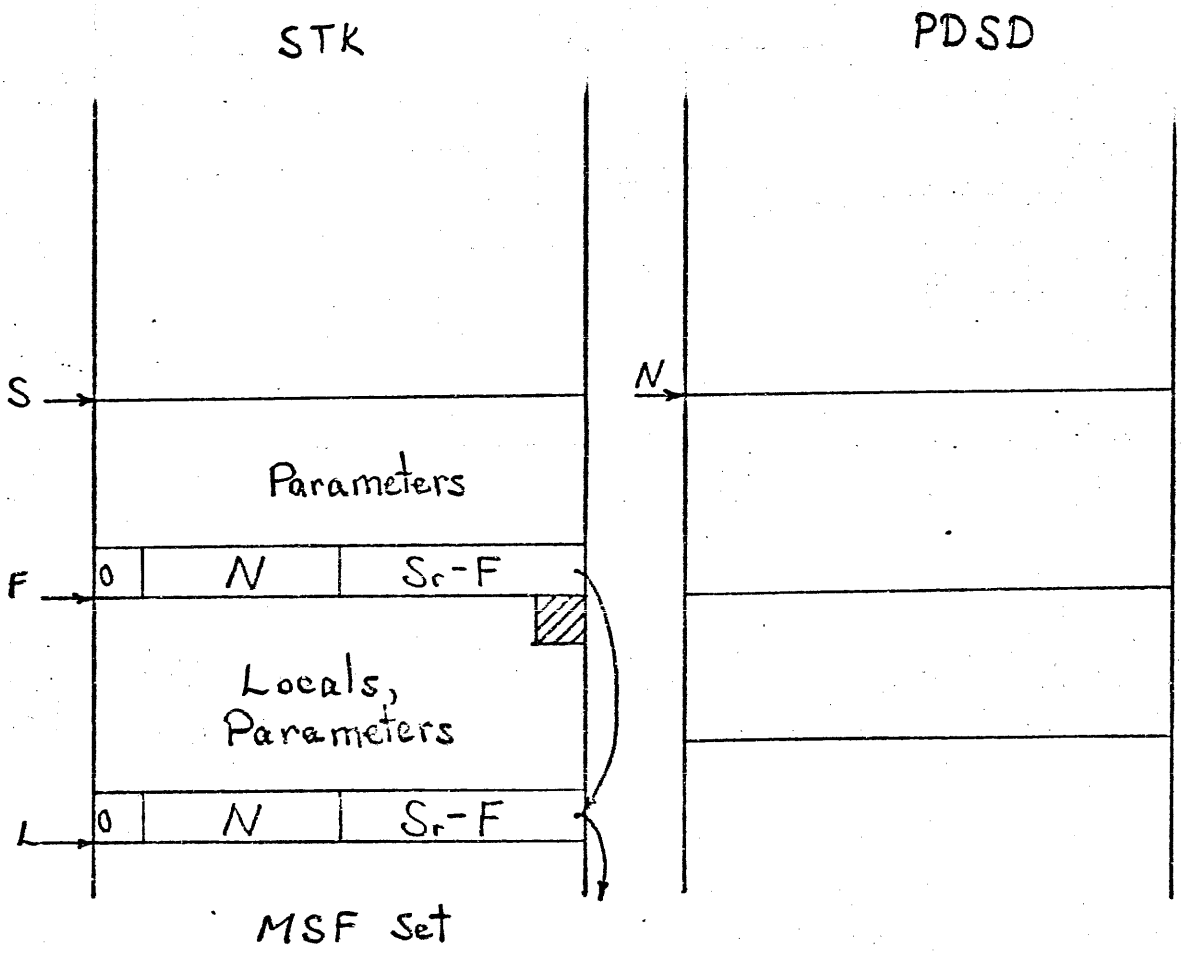


FIGURE 4-6. The Process Stack After An Exit Operation.

4.1.4 Conditional-exit

4.1.5 Branch

BRANCH A1

A branch to an entry point operates just like the corresponding portion of an ENTER operation. In addition a condition may have to be evaluated.

The main difference between ENTER and BRANCH is that no history information is saved for BRANCH.

The sequence of operations is:

1) Operand fetch.

2) Environment updating.

3) Transfer of control.

Operand Fetch

A transfer address may be specified in one of two ways;

a) by a simple address (segment number, offset)

b) by an entry point pair.

Both of these forms might be preceded by various amounts of indirection. The net result of operand fetch will be a code segment and offset. If an entry point is discovered, a new code and data environment are specified as well.

Environment Updating

If in the course of operand fetch, an entry point descriptor was encountered, the code and data environments are changed to those specified in the entry point.

Transfer Of Control

Control passes to the address generated as the result of fetch.

4.1.6 Conditional Branches

4.1.6.1 Branch-LSS

4.1.6.2 Branch-LEQ

4.1.6.3 Branch-EQL

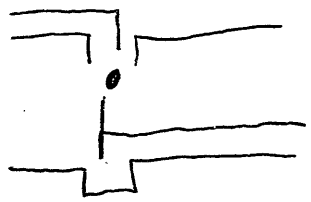
4.1.6.4 Branch-NEQ

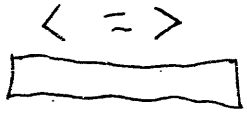
4.1.6.5 Branch-GEQ

4.1.6.6 Branch-GTR

4.1.6.7 Branch-on-overflow

4.1.6.8 Branch-on-no-overflow





4.1.7 Process Control And Coordination And Processor Allocation

For process control and coordination and processor allocation there exists a Master Controller (MC). The MC coordinates process execution via a bit vector designated the Ready Vector. The bits in this Ready Vector are uniquely assigned, in priority order, to Primary and Supplemental Processes.

A Primary Process is one which specifies the execution of a sequence of processor instructions. A Primary Process has a Process State Record (PSR) which contains (directly and/or by reference) the environment in which those instructions are executed.

A Supplemental Process is primarily a queuing device. A Supplemental Process has a Control Block (CB) whose normal usage is to record the occurance of an external event (such as the completion of an I/O process or the timeout of the interval timer), so that a Primary Process may synchronize itself with reality.

There also exists a Process-Control Vector (PCV), indexed in parallel with the Ready Vector, which locates the Process Record or Control Block, identifies the process type and gives the Running Priority of the process.

The processor has the following registers accessible by the MC:

PRI The index in the Ready Vector and PCV of the process currently being executed.

RPR The Running Priority of the process currently being executed. (This is not necessarily the same value as in PRI.)

4.1.7.1 Master Controller Commands

The MC accepts the following commands from the processor:

1) Activate Process n.

2) Wait.

3) Yield To Process n.

The MC also accepts the "Activate Process n" command from the I/O Subsystem and from the Interval Timer or, if the MC is a part of the I/O Subsystem, it may cause this action itself.

ACTIVATE-PROCESS

START N1

The Process-Number, N1, is compared to the processor's RPR register. If N1 is not greater than RPR, bit N1 is set in the Ready Vector and the operation is complete. Otherwise, the processor is commanded to Go-to-idle, the bit in the Ready Vector for the processor's current process (given by the PRI register) is set, the value N1 is placed into PRI, and the processor is commanded to commence execution of that process.

WAIT

WAIT

The processor is commanded to Go-to-idle. The Ready Vector is examined, starting at the bit corresponding to the value in RPR and proceeding towards bit zero, until a bit is found to be ON or it is determined that all bits are OFF. In the latter case, RPR and PRI are set to "-1" and the operation is complete. Otherwise, the index of the highest-numbered bit that is ON is determined and placed into PRI, the bit is reset, and the processor is commanded to commence execution of that process.

YIELD-TO-PROCESS

YIELDTO N1

The processor is commanded to Go-to-idle. If N1 is not less than the value in RPR, then N1 is placed into PRI, and the processor is commanded to commence execution of that process, and the operation is complete. Otherwise, the bit in the Ready Vector for process N1 is set, and the Ready Vector is examined, starting at the bit corresponding to the value in RPR and proceeding towards bit zero, until a bit is found that is on. The number of that bit (which is not less than N1) is placed into PRI, the bit is reset, and the processor is commanded to commence execution.

4.1.7.2 Processor Commands

The MC can cause the processor to

1) Leave its current process and idle.

2) Execute the process identified by PRI.

3) Set PRI.

GO-TO-IDLE

IDLE

The Go-to-idle command causes the processor to update the PSR of the process being executed and wait for a further command.

EXECUTE-PROCESS

EXECUTE

The processor reacts to the Execute-process command by accessing the PCV entry for the process indexed by PRI, setting RPR to the value in the PCV entry, and identifying the Process Type.

EXECUTE-PROCESS (Continued)

If it is a Primary Process, its PSR is absorbed and executed; otherwise it is a Supplemental Process and its CB is accessed. A "released" bit is set and a "notify" bit is examined. If the "notify" bit is ON, a Yield-to-process n is executed, where the contents of an "owner" field supplies the value for n. If the "notify" bit is OFF, a Wait is executed.

SET-PRI

SETPRI

The Set-PRI command causes the processor to insert a value into its PRI register. This command must occur only after a Go-to-idle command or at processor initialization.

4.2 ARITHMETIC

All arithmetic operators allow as operands any numeric data types, in any combination. Conversions are implied by the use of mixed types.

a) The use of an integer sending field with a real sending field implies an internal conversion of the integer operand to floating point representation.

b) If the receiving field is real, any result stored is normalized, regardless of input types.

c) Any fractional part of a result is lost if the receiving field is integer.

d) The sign of an unsigned integer sending field is assumed to be positive.

e) The sign of a result is not stored if the receiving field is unsigned.

f) The toggles are set to indicate the sign of the result, whether the sign is stored or not. Low means negative, equal means a zero result and high means positive.

4.2.1 Add

```
ADD     N1, N2, N3
```

Add N1 to N2, result in N3.

4.2.2 Add-one

```
ADD1    N1, N2
```

Add one to N1, result in N2.

4.2.3 Increment

```
INCR    N1, N2
```

4.2.3 Increment (Continued)

Add N1 to N2, result in N2.

4.2.4 Increment-by-one

```
INC1    N1
```

Add one to N1, result in N1.

4.2.5 Subtract

```
SUBT    N1, N2, N3
```

Subtract N1 from N2, result in N3.

4.2.6 Subtract-one

```
SUB1    N1, N2
```

Subtract one from N1, result in N2.

4.2.7 Decrement

```
DECR    N1, N2
```

Subtract N1 from N2, result in N2.

4.2.8 Decrement-by-one

```
DEC1    N1
```

Subtract one from N1, result in N1.

4.2.9 Multiply (two forms)

```
MPY     N1, N2
```

4.2.9 Multiply (two forms) (Continued)

Multiply N1 times N2, result in N2.

```
MULT    N1, N2, N3
```

Multiply N1 times N2, result in N3.

4.2.10 Divide (two forms)

```
DIV     N1, N2
```

Divide N1 into N2, result in N2.

```
DIVD    N1, N2, N3
```

Divide N1 into N2, result in N3.

4.2.11 Remainder-divide

```
MOD     N1, N2, N3
```

Divide N1 into N2, remainder in N3.

4.2.12 Double (two forms)

```
DBL     N1
```

Add N1 to N1, result in N1.

```
DUBL    N1, N2
```

Add N1 to N1, result in N2.

4.2.13 Halve (two forms)

```
HLF     N1
```

Divide N1 by 2, result in N1.

```
HALF    N1, N2
```

4.2.13 Halve (two forms) (Continued)

Divide N1 by 2, result in N2.

4.2.14 Absolute-value (two forms)

```
ABS     N1
```

Set N1 to its absolute value.

```
ABSV    N1, N2
```

Move the absolute value of N1 to N2.

4.2.15 Negate (two forms)

```
NEG     N1
```

Set N1 to its additive inverse.

```
NEGT    N1, N2
```

Move the additive inverse of N1 to N2.

4.3 LOGICAL

4.3.1 And (two forms)

AND S1, S2

Logically AND the units of S1 with the units of S2, result in S2.

LAND S1, S2, S3

Logically AND the units of S1 with the units of S2, result in S3.

4.3.2 Or (two forms)

OR S1, S2

Logically OR the units of S1 with the units of S2, result in S2.

LOR S1, S2, S3

Logically OR the units of S1 with the units of S2, result in S3.

4.3.3 Exclusive-or (two forms)

XOR S1, S2

Logically exclusive-OR the units of S1 with the units of S2, result in S2.

EXOR S1, S2, S3

Logically exclusive-OR the units of S1 with the units of S2, result in S3.

4.3.4 Not (two forms)

NOT S1

Logically negate the units of S1, result in S1.

LNOT S1, S2

Logically negate the units of S1, result in S2.

4.4 RELATIONAL

4.4.1 Compare

CMPR O1, O2

Set the comparison flip-flops to indicate the relation of O1 to O2.

4.4.2 Compare-under-table

CMPTBL O1, T2, O3

Set the comparison flip-flops to indicate the relation of O1 to O3 relative to table T2.

4.4.3 Compare-with-zero

CMPZRO O1

Set the comparison flip-flops to indicate the relation of O1 to zero.

4.4.4 Class-test

CLASS CH1

Set the condition flip-flops to indicate whether the value of CH1 is numeric, alphanumeric or alphabetic.

4.4.5 Range-compare

RANGE O1, O2, O3

Set the comparison flip-flops to indicate the relation of O2 to the range specified by O1 and O3.

4.4.6 Bounds-check (two forms)

CHCK O1, O2

Cause a program fault if O1 is greater than O2.

CHECK O1, O2, O3

Set the comparison flip-flops to indicate the relation of O2 to to the range specified by O1 and O3 and cause a program fault if O2 is less than O1 or greater than O3.

4.5 OPERAND MANIPULATE

4.5.1 Clear

```
CLEAR   O1
```

Fill O1 with the appropriate padding unit.

4.5.2 Push-operand

```
PUSH    O1
```

Move O1 to the top of the Process Stack.

4.5.3 Move

```
MOVE    O1, O2
```

Set O2 to the value of O1.

4.5.4 Move-right-justified

```
MOVERJ  O1, S2
```

Right justify the value of O1 in S2.

4.5.5 Move-data

```
MOVED   O1, O2, O3
```

Move data from O1 to O2, length is (O3 - O2). No conversions.

4.5.6 Move-repeat

```
SMEAR   O1, O2
```

Repeat the contents of O1 throughout O2.

4.5.7 Edit

```
EDIT   O1, CH2, S3
```

Move O1 to S3 using the micro-instruction string in CH2.

4.5.8 Scan-string-match

```
SCANEQL S1, S2
```

Scan S2 for the first occurrence of a substring equal to S1.

4.5.9 Scan-string-no-match

```
SCANNEQ S1, S2
```

Scan S2 for the first occurrence of a substring not equal to S1.

4.5.10 Scan-set-member

```
SCANSET S1, T2
```

Scan S1 for the first occurrance of a unit whose bit is ON in table T2.

4.5.11 Search-set-of-records

```
SEARCH  O1, O2, A3
```

Search a set of records starting at O2 for a comparison to O1, according to the information at A3.

4.5.12 Translate

```
XLATE   S1, T2, S3
```

Translate S1 through table T2 into S3.

4.5.13 Shift-units-right (two forms)

SHFTUR N1, S2

Shift S2 right by N1 units, result in S2.

SHIFTUR N1, S2, S3

Shift S2 right by N1 units, result in S3.

4.5.14 Shift-units-left (two forms)

SHFTUL N1, S2

Shift S2 left by N1 units, result in S2.

SHIFTUL N1, S2, S3

Shift S2 left by N1 units, result in S3.

4.5.15 Shift-bits-right (two forms)

SHFTBR N1, HX2

Shift HX2 right by N1 bits, result in HX2.

SHIFTBR N1, HX2, HX3

Shift HX2 right by N1 bits, result in HX3.

4.5.16 Shift-bits-left (two forms)

SHFTBL N1, HX2

Shift HX2 left by N1 bits, result in HX2.

SHIFTBL N1, HX2, HX3

Shift HX2 left by N1 bits, result in HX3.

4.6 POINTER MANIPULATE

4.6.1 Construct-address

```
ADDR   O1, A2
```

Place at address A2 an operand reference to O1, with all indirection and indexing resolved.

4.8.3 Push-address

```
PUSHADR O1
```

Place on the top of the stack an operand reference to O1, with all indirection and indexing resolved.

4.8.4 Push-name (STUFF)

```
PUSHNAM O1
```

The same as PUSHADR, but also place into the process dynamic segment dictionary a descriptor referencing the segment in which O1 resides.

4.7 BIT MANIPULATE

These operators access a particular bit within a hex field. The bits of a field are considered to be numbered high-order to low-order, with the high-order bit of the high-order unit being bit zero and the following bits being 1, 2, 3, 4, .. n-1 (where n is the number of bits in the hex field). Bit indices which are negative or which reference a bit beyond the end of the hex field cause a fault.

4.7.1 Set-bit

```
SET     N1, HX2
```

Set the N1'th bit of HX2 to TRUE.

4.7.2 Reset-bit

```
RSET    N1, HX2
```

Reset the N1'th bit of HX2 to FALSE.

4.7.3 Complement-bit

```
FLIP    N1, HX2
```

Complement the N1'th bit of HX2.

4.7.4 Test-bit

```
TEST    N1, HX2
```

Set the condition flip-flops to indicate whether the N1'th bit of HX2 is TRUE or FALSE.

4.8 EXPONENT MANIPULATE

4.8.1 Set-exponent

```
SETX    N1, R2
```

Convert N1 to binary and insert the value in the exponent field of R2.

4.8.2 Increment-exponent

```
INCX    N1, R2
```

Convert N1 to binary and add the value to the exponent field of R2.

4.8.3 Add-to-exponent

```
ADDX    N1, R2, R3
```

Convert N1 to binary and add the value to the exponent field of R2, store the resulting real number in R3.

4.8.4 Convert-exponent-decimal

```
CVTX    R1, N2
```

Convert the exponent of R1 to decimal and store the result in N2.

4.9 MISCELLANEOUS

4.9.1 Initiate I/O

4.9.2 Reset-overflow-flip-flop

Reset the arithmetic overflow indicator toggle.

4.9.3 Set-condition-flip-flops

```
SETCOND N1
```

Set the comparison indicator toggles to the value indicated by N1.

4.9.4 Event

```
EVENT   N1, A2
```

If the event class bit specified by N1 is not TRUE in the Process State Record, branch to address A2.

4.9.5 Monitor

```
MONITOR O1
```

Make the information in O1 available on the "backplane" for physical monitoring.

4.9.6 Read Timer

4.9.7 Set Timer

5 COMPATIBILITY MODE

This section specifies the differences in semantics between MS-1/MS-2 and MS-3 compatibility mode.

The following MS-2 operators will not be executed in compatibility mode:

90	BRE	Reinstate
91	SRD	Scan Result Descriptor
92	RAD	Read Address
94	IIO	Initiate I/O
95	RDT	Read Timer
96	RCT	Read and Clear Timer
97	STT	Set Timer
	Load	Initialization

All Snap operations

All other operations will be executed in the same way, giving the same answers, with the following exceptions:

a. Operations on undigits which give unspecified results on MS-1, MS-2, will give unspecified results in MS-3 compatibility mode.

b. Operations involving overlapping fields leading to unspecified results on MS-1, MS-2, will give unspecified results in MS-3 compatibility mode.

c. MS-1, MS-2 data alignment requirements will be the same in MS-3 compatibility mode.

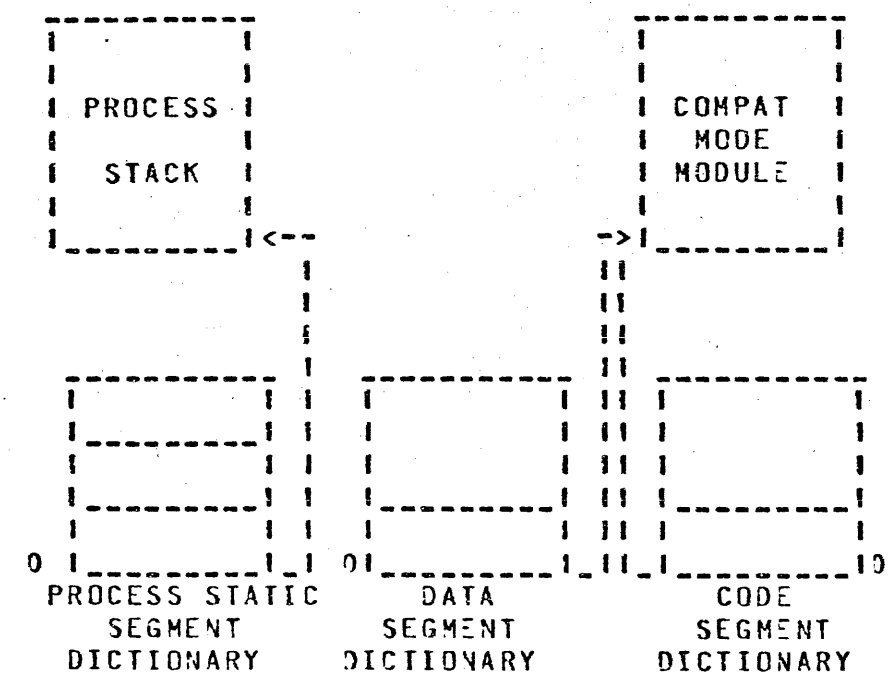
d. Normal state programs only will be supported.

5.1 COMPATIBILITY MODE PROCESS STRUCTURE

The highest level module only of a process may run in compatibility mode. That is, no compatibility mode module may be called by another module. Initiation to compatibility mode is dependent on information in the process state record (compatibility mode bit). BCT's are interpreted as a module call to a given MCP routine. Information is stored on the call which allows return to compatibility mode upon exit. One parameter is passed which is the address of the BCT instruction.

5.1 COMPATIBILITY MODE PROCESS STRUCTURE (Continued)

While running in compatibility mode, data references are considered to be relative to segment zero of the data environment, and code is fetched from segment zero of the code environment. It is the responsibility of the MCP to insure that these two segments are identical. All references by the hardware to program reserved memory (edit masks, index registers, indirect field lengths, etc) are relative to segment zero of the data environment. Figure 5.1-1 below indicates the compatibility mode process structure



5.2 DATA REPRESENTATION

The following modes of data representation, 4-bit mode and 8-bit mode, are defined for use.

5.2.1 FOUR-BIT MODE

In four-bit mode, data is interpreted in units of 4 bits. Where a sign is expected, it is interpreted as a separate and lleading 4-bit unit.

The internal code in 4-bit mode as interpreted by the arithmetic units is as follows:

Digit Code	Sign Code
0000 - 0	+
0001 - 1	+
0010 - 2	+
0011 - 3	+
0100 - 4	+
0101 - 5	+
0110 - 6	+
0111 - 7	+
1000 - 8	+
1001 - 9	+
1010 - Undefined*	+
1011 - Undefined*	+
1100 - Undefined*	+
1101 - Undefined*	-
1110 - Undefined*	+
1111 - Undefined*	+

*Undefined - Engineering does not guarantee the consistency of results. Use of undigits may not give normal arithmetic results.

When signed 4-bit format is specified in the receiving field for any operation, the sign-digit is set as follows:

ASCII Mode	EBCDIC Mode
+ 1011	+1100
- 1101	-1101

A plus sign compares as "high" relative to a minus sign when it is interpreterd as a sign digit.

5.2.2 EIGHT-BIT MODE

In eight-bit mode, data is interpreted in units of 8 bits unless otherwise specified.

The internal representation of data can be extended binary code decimal interchange code (EBCDIC) or an eight bit extension of ASCII.

Conversion between 4-bit representration and 8-bit representation is accomplished automatically during the execution of instructions.

For code sensitive instructions involving numeric data, the most significant four bits of the receiving field are automatically set to the code indicating the numeric subset of the selected 8-bit code. These four bits in the case of EBCDIC are 1111 and in the case of ASCII are 0101.

Eight-bit data is considered unsigned except in the case of the move alphanumeric, move numeric, and edit instructions. Additional details are given in the description of the three instructions.

All alphanumeric (8-bit) data must start at even addresses. If the source field is an odd address, an address error will occur.

Alphanumeric comparisons are binary and thus the (low to high) collating sequence for EBCDIC is symbols-alphas-digits and for ASCII it is symbols-digits-alphas.

5.3 INSTRUCTION REPRESENTATION

"Reserved" or "not specified" bits, digits, or characters must be false, and they are reserved for expansion as assigned for use solely by Burroughs's Engineering.

Invalid bits, digits or characters are recognized by the hardware as such and are reported as "invalid" in the Result Descriptor".

All fields are addressed most significant digit first unless specifically noted otherwise.

5.3 INSTRUCTION REPRESENTATION (Continued)

The data fields addressed by the final addresses and their related controllers in the descriptor are referred to as the A-field, the B-field, and the C-field.

All instructions must start at even addresses.

Code sensitivity in instructions is based on EBCDIC or ASCII (8-bit extension) as determined by a mode flip-flop.

All data for fixed-length arithmetic operands must be Mod 4.

5.3.1 INSTRUCTION FORMAT

The processor instructions may vary in lenght from 4 to 30 digits with the format as shown below. The format is also shown when extended addressing is used. An instruction may use a mixture of Extended and Non-extended addressing.

Non-Extended Format	Extended Format
OP VV	OP VV
OP VVVV	OP VVVV
OP AAAA	OP AAAA
OP AAAAAA	OP AAAAAAAA
OP AFBF AAAAAA BBBBBB	OP AFBF AAAAAAAA BBBBBBBB
OP AFBF AAAAAA BBBBBB CCCCCC	OP AFBF AAAAAAAA BBBBBBBB CCCCCCCC

where:
OP = Operator Code
V = Variant Digits
AFBF = A and B field variant digits
A,B,C = Address of respective data fields

The instruction format is digit numbered, beginning with "D1" for the first digit of the OP code and continuing thru "D30".

5.3.2 OPERATOR CODE

Two digits (8-bits) identified as D1 and D2 are used for operator coding. All unassigned Operator Codes are reserved for expansion by Engineering. The occurrence of invalid codes or operator codes requiring non-present options is detected and sets an interrupt.

The Operator Code implies the unextended instruction length and additional syllables of the instruction are fetched automatically.

5.3.3 FIELD LENGTH - FORMAT A

Digits identified as D3, D4, D5, and D6 when specifying field length, are limited to decimal digits. Maximum field length is 100 digits or characters unless otherwise specified.

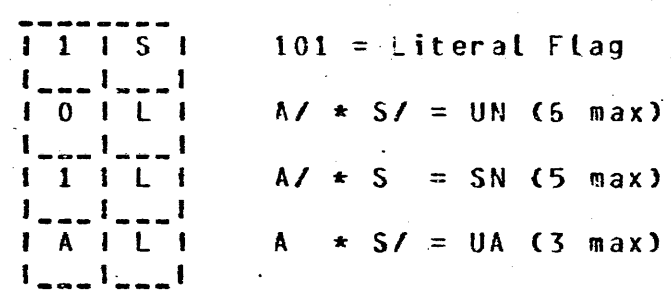
5.3.3.1 Indirect Field Length - Format A

Indirect Field Length for the A field is specified by a "1" bit in the two high order bits of D3. Indirect Field Length for the B field is specified by a "1" bit in the two high order bits of D5. If field length is determined by the concatenation of D3, D4, D5, and D6, the same rules apply, that is zero, one, or two segments may be indirect. The Indirect Field Length operation can be carried to any depth except infinite. Infinite results in a Timeout Processor Result Descriptor.

The base relative address of the Indirect Field Length for the A field is specified by the two low order bits of D3 and the three high order bits of D4. D3 can take on values 0 to 3 and D4 can take on even values 0, 2, 4, 6, or 8. Thus 20 indirect addresses ranging from 00 to 38 are available. The address of the Indirect Field Length for the B field is similarly obtained from D5 and D6. The low order bit of D4 and that of D6 are ignored. Indirect Field Length can not be used in address branch, exit, halt-branch, nor for the mask in bit test, bit set, bit reset, nor for fixed- length arithmetics, and halt-breakpoint instructions.

5.3.3.2 Literals

AF (Only)



Another use of the AF variant is to indicate that the A field of the instruction does not contain an address, but instead contains literal data to be used directly by the instruction. This option is indicated when the high order bits of digit D3 are "101". The six digits identified as D7 thru D12 of the instruction are the operand itself and not the address index, address controller, or the address of the operand. The literal is left justified.

The length of the literal is specified by the value of the three low order bits of digit D4. Length is limited to 1 to 6 for unsigned 4-bit format, 1 to 5 for signed 4-bit format, 1 to 3 for 8-bit format and 1 to 2 for Floating Point instructions. The LSB of D3 and MSB of D4 specify these formats as shown below.

LSB of D3	MSB of D4	Meaning
0	0	Unsigned 4-bit Format
0	1	Signed 4-bit Format
1	0	8-bit Format
1	1	Reserved

Literal capability can be used to obtain the B-field length while the A address is used as a literal if the length value is 7 for unsigned 4-bit format.

Indirect field length is not to be used with the literal capability for the A-field.

The literal capability can be used in the following instructions.

5.3.3.2 Literals (Continued)

	OP	Mnemonic	
1.	01-06	INC, ADD, SUB, MPY, DEC, DIV	All variable-field length arithmetic instructions.
2.	80-83	FAD, FSU, FDV, FPM	All floating point instructions Mantissa is limited to 2 digits.
3.	10	MVA	Move alphanumeric
4.	11	MVN	Move numeric
5.	14	MVR	Move repeat
6.	09	MVL	Move links
7.	45	CPA	Compare alphanumeric
8.	46	CPN	Compare numeric
9.	42	AND	And
10.	43	ORR	Or
11.	44	NOT	Not
12.	16	SDE	Scan to delimiter - equal
13.	17	SDU	Scan to delimiter - unequal
14.	18	SZE	Scan to delimiter - zone equal
15.	19	SZU	Scan to delimiter - zone unequal
16.	39	SEA	Search
17.	37	SLL	Search linked list
18.	38	SLD	Search link delink

Use of the Literal capability in the following instructions is illegal:

	OP	Mnemonic	
1.	15	TRN	Translate
2.	49	EDT	Edit
3.	40	BZT	Bit zero test
4.	41	BOT	Bit one test
5.	31	NTR	Enter
6.	12	MVW	Move word
7.	13	MVC	Move and clear word
8.	50	IAD	Fixed length arithmetic
	51	IAS	"
	52	ISU	"
	53	ISS	"
	54	IMU	"
	55	IMS	"
	57	IMI	"
	58	ILD	"
	59	IST	"
	70	RAA	"
	71	RAS	"

5.3.3.2 Literals (Continued)

	72	RSU	"
	73	RSS	"
	74	RMU	"
	75	RMS	"
	76	RDV	"
	77	RDS	"
	78	RLD	"
		79	RST
9.	34	BST	Bit set
10.	33	BRT	Bit reset
11.	08	MVD	Move data

5.3.4 ADDRESSES

Normally, addressing is non-extended. If Extended addressing is desired, the extended function adapter must be installed in the processor.

Digits identified as address digits are limited to the decimal digits 0..9 except as noted in section 5.3.3.2, 5.3.4.1, and 5.3.4.2. Undigits in address digits may or may not cause invalid errors to be detected

5.3.4.1 Non-Extended Format

The Non-Extended address format is shown in figure 5.3.4-1 where direct addressing capability is from 000,000 through 299,998, or if indexing is used the maximum address is 999,998.

5.3.4.1 Non-Extended Format (Continued)

5.3.4.2 Extended Format

The format of extended addressing and its comparison to the non-extended form is shown in figure 5.3.4-1.

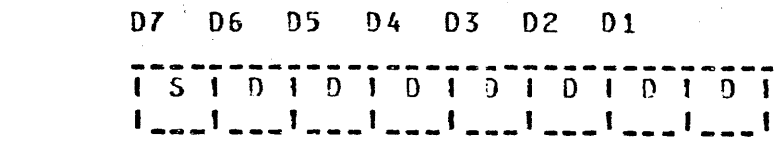
If the 2nd most significant digit of an address syllable contained in an instruction is a binary 12 (1100), the next six digits contain the address. This address has direct addressing capability from 000,000 thru 999,998. If indexing is used the addressing capability extends from 000,000 thru 9,999,998. Note that Address Extension applies only to addresses contained in instructions and includes indirect addresses.

5.3.4.3 Address Indexing

The two high order bits of the first digit of the address field signify indexing as follows:

00 - No Indexing
01 - Index Register I (Base Relative Address 8)
10 - Index Register II (Base Relative Address 16)
11 - Index Register III (Base Relative Address 24)

The following Index Register format is defined:



S = SIGN
D = Decimal number 0 thru 9

The address of the Index Register points to the sign digit. The Index Register contents D1 thru D7 are added to or subtracted from any address depending on the sign digit. However, in any operation involving an Index Register as an operand, the Index Register must have an appropriate address controller and length.

5.3.4.3 Address Indexing (Continued)

The value of J7 is ignored unless Extended Memory Mode is used (EMMF controlled by the SMF instruction).

5.3.4.4 Address Controller

The two low order bits of the first digit of the address field provide information that refers to the particular address or to the data stored at the address to which it refers.

The two bits carry the following significance:

00 - Unsigned 4-bit Format
01 - Signed 4-bit Format
10 - Unsigned 8-bit Format
11 - Indirect Address

Any of the four combinations are valid except as specifically prohibited in some instructions.

If address extension, section 5.3.4.2 is not used, then the branch address in the Address Branch, Halt Branch, Enter, and Exit instructions have address controller bits of the address field which carry the following significance:

00 = 0 - Most Significant Digit of Address
01 = 1 - Most Significant Digit of Address
10 = 2 - Most Significant Digit of Address
11 = 2 - Indirect Address

An indirect address must be even and is checked after indexing, if any. An odd indirect address is considered to be a non-synced address contained in an instruction and causes an interrupt.

The values of the first digit for both the low and high-order bits are shown below.

5.3.4.4 Address Controller (Continued)

UN	SN	UA	IA	
0	1	2	3	No Indexing
4	5	6	7	IX1
8	9	A	B	IX2
C	D	E	F	IX3

5.3.5 OVERLAPPING FIELDS

Overlapped fields are identical, or have "total overlap" if the address controllers are identical and lengths are the same. Two fields partially overlap when one or more, but not all information units of the two fields have the same memory locations.

No partial overlap is permitted of any two or all three fields, A, B, and C, except for the following cases which occur in move instructions:

1. Replication throughout the destination field of the first information unit (digit, charater, word) of the origin field:

Move Numeric (11 MVN) B Address = A Address + 1,
AC = UN, BC = UN.
Move Alphanumeric (10 MVA) B Address = A Address + 2,
AC = UA, BC = UA.

Move Word (12 MVW) B Address = A Address + 4

2. Shift of the entire destination field one information unit left.

Move Numeric (11 MVN) B Address = A Address - 1,
AC = UN, BC = UN.

Move Alphanumeric (10 MVA) B Address = Address - 2,
AC = UN, BC = UN.

5.3.5 OVERLAPPING FIELDS (Continued)

Move Word (12 MVW) B Address = A Address - 4.

Move & Clear Word (13 MVC) B Address = A Address - 4.

3. Right justification of first word of the origin field in the destination field, with leading zeros.

Move & Clear Word (13 MVC) B Address = A Address + 4.

Total (identical) overlap is permitted only where specified in command descriptions.

5.5 LOGICAL UNITS

The effect an instruction has on the following Logical Units is described in the individual instructions, Section 5.6 of this specification.

5.5.1 OVERFLOW FLIP-FLOP

The Overflow Flip-Flop indicates that the result field length of and arithmetic, move alphanumeric, or move numeric operation is not sufficient to store the result.

This Overflow Flip-Flop is not cleared at the beginning of an arithmetic operation, but is preserved. Therefore, it indicates overflow that has occurred any time before or during a series of arithmetic operations or other interjected non-arithmetic operations.

The Overflow Flip-Flop is cleared by the Conditional Branch on Overflow, the Edit and the Search instructions.

The Overflow Flip-Flop is affected by the following instructions:

	OP	Mnemonic	Name
1.	30	BCT	Communicate
2.			Interrupt
3.	01-04 06	INC, ADD DEC, SUB, DIV	Arithmetic Instructions Except Multiply (Variable Length)
4.	50-55	IAD, IAS	Fixed-Length Arithmetic Instructions

5.5.1 OVERFLOW FLIP-FLOP (Continued)

		ISU, ISS	
	57-59	IMU, IMS,	
		IMI, ILD	
		IST	
5.	80-83	FAD, FSU	Floating Point Instructions
		FPM, FDV	
6.	39	SEA	Search
7.	11	MVN	Move Numeric
8.	10	MVA	Move Alphanumeric
9.	31	NTR	Enter
10.	32	EXT	Exit
11.	28	OFL	Address Branch on Overflow
12.	49	EDT	Edit

5.5.2 COMPARISON FLIP-FLOPS

The states of the Comparison Flip-Flops are:

00	Cleared
01	Greater or High
10	Less or Low
11	Zero or Equal

The comparison Flip-Flops are affected by the following:

	OP	Mnemonic	Name
1.	30	BCT	Communicate
2.	01-06	INC, ADD,	Arithmetic Instructions,
		DEC SUB,	Variable length
		MPY, DIV	
3.	80-83	FAD, FSU,	Floating Point Instructions
		FPM, FDV	
4.	50-55	IAD, IAS,	Fixed-Length Arithmetics
		ISU, ISS	
	57-59	IMU, IMS,	
		IMI, ILD,	
		IST	
5.	11	MVN	Move Numeric
6.	10	MVA	Move Alphanumeric
7.	31	NTR	Enter
8.	32	EXT	Exit
9.	46	CPN	Compare Numeric
10.	45	CPA	Compare Alphanumeric
11.	91	SRD	Scan Result Descriptor
12.	49	EDT	Edit

5.5.2 COMPARISON FLIP-FLOPS (Continued)

13.	42	AND	And
14.	43	OR	Or
15.	44	NOT	Not
16.	40,42	BZT, BOT	Bit Test Instructions
17.	16	SDE	Scan to Delimiter - Equal
18.	17	SDU	Scan to Delimiter - Unequal
19.	18	SZE	Scan to Delimiter - zone Equal
20.	19	SZU	Scan to Delimiter - zone Unequal
21.	37-38	SLL,SLD	Search Instructions, Search Link list, Search Link Delink
22.	33	BRT	Bit Reset
23.	34	BST	Bit Set

Neither conditional nor unconditional branching alters the Comparison Flip-Flops.

Any instruction which destroys the Comparison Flip-Flops settings also clears them to the 00 state.

5.5.6 INDEX REGISTERS

There are three Index Registers occupying a reserved area of memory. The reserved area is eight digits including a sign digit for each Index Register, and is base relative. Thus all programs have three Index Registers available:

Index Register 1 address is Base plus 8.
Index Register 2 address is Base plus 16.
Index Register 3 address is Base plus 24.

The Index Registers are affected by the following:

1. All Instructions That Can Address Memory
2. Enter (IX3)
3. Exit (IX3)
4. Search (IX1)
5. Search Link List (IX1)
6. Search Link Delink (IX1, IX2)

5.5.7 MODE FLIP-FLOPS

Several mode/enable flip-flops are present to enable various features.

5.5.7.1 EBCDIC-ASCII Mode Flip-Flop

This mode flip-flop (EA) selects the data format to which some instructions are sensitive. The one-state indicates ASCII Mode and the zero-state indicates EBCDIC Mode.

The following code sensitive instructions are affected by the EA Mode Flip-Flop.

1. Fixed Length Arithmetics (Ref. Table 6-1)
2. Variable Length Arithmetics (Ref. Table 6-1)
3. Move Alphanumeric (MVA - OP 10)
4. Move Numeric (MVA -OP 11)
5. Move Repeat (MVR - OP 14)
6. Edit (EDT - OP 49)
7. Translate (TRN - OP 15)
8. Scan to Delimiter - Equal (SDE - OP 16)
9. Scan to Delimiter - Unequal (SDU - OP 17)

The EA Mode Flip-Flop is affected by:

1. Set Mode Flip-Flop Instruction (SMF - OP 47)
2. Interrupt
3. Enter (NTR - OP 31)
4. Exit (EXT - OP 32)

5.5.7.2 User Program Flip-Flop

This mode flip-flop indicates that a special program is in operation.

5.5.7.3 BCT Mode Flip-Flop

This flip-flop is affected by the BRE (OP 90) instruction only if EMMF is on (SMF - OP 47).

6.0 COMPATABILITY MODE INSTRUCTIONS

A summary of the operation codes is contained in Table 6-1. A detailed description of the instructions begins with Section 6.1. Table 6-1 Operation Code Summary

6.0 COMPATABILITY MODE INSTRUCTIONS (Continued)

	OP. Code	Mnemonic	Name	No. of Addresses
	Arithmetic, Variable Field Length			
	01	INC	Add-Two Address	2
	02	ADD	Add-Three Address	3
	03	DEC	Subtract-Two Address	2
	04	SUB	Subtract-Three Address	3
	05	MPY	Multiply	3
	06	DIV	Divide	3
**	80	FAD	Floating Add	3
**	81	FSU	Floating Subtract	3
**	82	FPM	Floating Multiply	3
**	83	FDV	Floating Divide	3
	Arithmetic, Fixed Length Instructions:			
	Integer -			
*	50	IAD	Integer Add	1
*	51	IAS	Integer Add and Store	1
*	52	ISU	Integer Subtract	1
*	53	ISS	Integer Subtract and Store	1
*	54	IMU	Integer Multiply	1
*	55	IMS	Integer Multiply and Store	1
*	57	IMI	Memory Increment - Decrement by 1	1
*	58	ILD	Integer Load	1
*	59	IST	Integer Store	1

*Optional (Part of Extended Function Adapter)

**Floating Point Adapter

	OP Code	Mnemonic	Name	No. of Addresses
	Real -			
*	70	RAA	Real Add	1
*	71	RAS	Real Add and Store	1
*	72	RSU	Real Subtract	1
*	73	RSS	Real Subtract and Store	1

6.0 COMPATABILITY MODE INSTRUCTIONS (Continued)

*	74	RMU	Real Multiply	1
*	75	RMS	Real Multiply and Store	1
*	76	RDV	Real Divide	1
*	77	RDS	Real Divide and Store	1
*	78	RLD	Real Load	1
*	79	RST	Real Store	1
*	84	ACC	Accumulator Manipulate	0
	Address Branching:			
	20	NOP	No Operation	1
	21	LSS	Less Than	1
	22	EQL	Equal	1
	23	LSS	Less Than or Equal	1
	24	GTR	Greater Than	1
	25	NEQ	Not Equal	1
	26	GEQ	Greater Than or Equal	1
	27	BUN	Unconditional	1
	28	OFL	Overflow	1
	29	HBR	Halt, Branch	1
	Control Branching			
	30	BCT	Communicate	0
	31	NTR	Enter	1
	32	EXT	Exit	1

* Optional (Part of Extended Function Adapter)

	OP Code	Mnemonic	Name	No. of Addresses
	Data Movement:			
*	08	MVD	Move Data	3
	09	MVL	Move Links	3
	10	MVA	Move Alphanumeric	2
	11	MVN	Move Numeric	2
	12	MVW	Move Word	2
	13	MVC	Move and Clear Word	2
	14	MVR	Move Repeat	2
	Logical:			
	15	TRN	Translate	3
	16	SDE	Scan to Delimiter - Equal	2
	17	SDU	Scan to Delimiter - Unequal	2

6.0 COMPATABILITY MODE INSTRUCTIONS (Continued)

	28	SZE	Scan to Delimiter - Zone - Equal	2
	19	SZU	Scan to Delimiter Zone - Unequal	2
*	33	BRT	Bit Rest	1
*	34	BST	Bit Set	1
*	37	SLL	Search Link	2
*	38	SLD	Search Link	2
	39	SEA	Search	3
	40	BZT	Bit Zero Test	1
	41	BOT	Bit One Test	1
	42	AND	And	3
	43	ORR	Or	3
	44	NOT	Not	3
	45	CPA	Compare Alphanumeric	2
	46	CPN	Compare Numeric	2
	47	SMF	Set Mode Flip-Flop	0
	48	HBK	Halt, Breakpoint	0
	49	EDT	Edit	3

* Optional (Part of Extended Function Adapter)

6.1 FIXED LENGTH ARITHMETIC INSTRUCTIONS

These functions are invalid unless the Extended Function Adapter hardware is installed.

The Fixed Length Arithmetic commands use a 20-digit accumulator which holds the instruction result within the processor as an operand for the next operation. The accumulator can be loaded or stored. Every instruction has an implied reference to the accumulator.

6.1.1 Instruction Formats

Two instruction formats are used:

One Address Instruction: This format is used whenever memory references are required.

No Address: This format is used for those operations which reference the accumulator only. This is a 4-digit instruction.

6.1.1.1 One Address Instructions

Format B is used--see Sections 3.1 and 3.4.1 except for the following redefinitions of the address controller:

Address Controller for OP's 50-59 except 57:

00 No Indirect Address
01 Reserved
10 Reserved
11 Indirect Address

Address Controller for OP 57:

00 Increment the Memory Value
01 Decrement the Memory Value
10 Reserved
11 Indirect Address

Address Controller for OP's 70-79:

00 Single Precision
01 Double Precision
10 Reserved
11 Indirect Address

In all cases, the final address, assembled after indexing and indirect addressing, must be Mod 4. Non-Mod-4 addresses will be treated as address errors during the fetch cycle.

No literal capability is available with these instructions.

6.1.1.2 4-Digit Instruction

The Accumulator Manipulate command (ACM) uses a 4-digit Instruction consisting of D1 and D2 as OP code with D3 and D4 as variants. The following variants are specified:

D3 (BITS) 8421	D4	Meaning
0000	0	Normalize Accumulator
0001	0	Convert Read to Integer
0010	0	Set Sign of Mantissa to +

6.1.1.2 4-Digit Instruction (Continued)

0011	0	Set Sign of Mantissa to -
0100	0	Complement Sign of Mantissa
0101	0	Clear Accumulator to -99+0
0110	0-9	Increment Algebraically the Exponent by D4
0111	0	0-9

All other variants are reserved. (See * in Section 5.2.1).

6.1.2 Data Formats

Two types of data are used: Integer (Fixed Point) and real (Floating Point).

Integer Format is a signed numeric field with a fixed length of 8 digits--a sign digit followed by the most significant digit of a 7-digit numeric field which is right justified.

Real Format consists of a sign-exponent field followed by a mantissa field. The sign-exponent field has a length of 4 digits where:

D1	is the exponent sign
D2, D3	gives the exponent magnitude, D2 is MSD
D4	is the mantissa sign

The mantissa is a numeric field of two possible lengths: Single precision (8 digits) or double precision (16 digits). The mantissa is assumed to be a fraction with the decimal point on the left.

The only real zero specified has an exponent of -99 and a mantissa of all zeros.

The signs of results are sensitive to the ASCII Mode Flip-Flop as in the format of Section 5.2.1 (4-bit mode). The internal code interpreted by the fixed-length arithmetics is as noted in Section 5.2.1 (4-bit mode).

6.1.3 OP's 50,51,52,53,54,55,57,58,59 (Integer Instructions)

OP 58 - ILD (Load)

6.1.3 OP's 50,51,52,53,54,55,57,58,59 (Integer Instructions) (Continued)

Load the value at the A-Address, which is of the form sign plus seven digits:

```
S  X  X  X  X  X  X  X
   1  2  3  4  5  6  7
```

into the 20-digit accumulator as follows:

```
+  0  8  S  X  X  X  X  X  X  X  0  0  0  0  0  0  0  0
            1  2  3  4  5  6  7  8
```

unless an exception as noted in Section 6.1.8 occurs.

OP 51, 53, 55, 59 - IAS, ISS, IMS, IST (Integer - Add & Store, Sub & Store, Multiply & Store, and Store).

Depending upon the instruction, perform the Add, Subtract, or Multiply as described in OP's 50, 52, and 54 and then store the results in exactly the same manner as OP 59.

OP 59 - IST (Integer Store)

Store the accumulator, which is of the form:

```
S W W S Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
1 1 2 2 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1
                        0 1 2 3 4 5 6
```

into memory at the A-Address as follows:

```
S Y Y Y Y Y Y Y
2 2 3 4 5 6 7 8
```

unless an exception, as noted in Section 6.1.8 occurs.

OP 50, 52, - IAD, ISU (Add/Sub)

Add/Subtract the value at the A-Addrerss to/from the contents of the accumulator and leave the result in the accumulator. The initial values of memory and accumulator are assumed to be of the form shown in "Load", above; the result generated is of like form.

6.1.3 OP's 50,51,52,53,54,55,57,58,59 (Integer Instructions) (Continued)

OP 57 - IMI (Memory Increment - Decrement by 1)

Increment or decrement the value of the A-Address by the value one (1) and store the result at the A-Address. The result in the accumulator is the same as the result in A.

OP 54 - IML (Multiply)

Multiply the value at the A-Address by the value in the accumulator; the result is in the accumulator. The initial values used from memory and the accumulator are assumed to be of the form shown in "Load" above; the result generated is of like form.

6.1.4 OP's 70 - 79 (Real Instructions)

OP 78- RLD (Load)

Load the value at the A-Address into the accumulator. If the operation is to be single-precision, the least significant 8 digits of the mantissa will be set to zero. The result exponent will be the memory expontent, unless an exception described in Section 6.1.8 occurs.

OP's 71, 73, 75, 77, 79 - RAS, RSS, RMS, RDS, RST (Real-Add & Store, Subtract & Store, Multiply & Store, Divide & Store, and Store)

Depending upon the instruction, perform the Add, Subtract, Multiply, or Divide as described in OP's 70, 72, 74 and 76 and then store the results in exactly the same manner as OP 79.

A description of OP 79 - RST (Real Store) follows:

Store the accumulator value at the A-Address. The accumulator value is unchanged. If the operation is to be single precision, the least significant 8 digits of the accumulator are ignored.

OP 70, 72, - RAA, RSU (Add/Sub)

6.1.4 OP's 70 - 79 (Real Instructions) (Continued)

Add/Subtract the value at the A-Address to/from the accumulator leaving the result in the accumulator. The memory value, the initial value in the accumulator, and the final value in the accumulator will have the same precision.

If the operation is to be single precision, the initial least significant 8 digits of the accumulator are cleared to zero before the operation begins.

Before alignment, both operands are expanded by one additional digit by adding a trailing zero. Alignment is performed on the extended values with non-aligning digits discarded (after alignment exponents are equal). The sum or difference is computed on the extended values. The extended result is left justified. After justification, the extended digit is discarded.

Neither operand is required to be normalized (left justified).

OP 74 - RMU (Multiply)

Multiply the value at the A-Address by the value in the accumulator; the result is left in the accumulator. If the operation is single precision, the initial least significant 8 digits of the accumulator are cleared to zero before the operation begins.

If both operands are normalized, the result will be normalized. Unnormalized operands may or may not give unnormalized results.

In Multiply, double precision, the operands are extended as described in Add/Sub above; the extended product is computed and normalized by one digit if necessary. After normalization the extended digit is discarded.

OP 76 - RVD (Divide)

Divide the contents of the accumulator by the value at the A-Address. If the operation is single precision, the least significant 8 digits of the accumulator are cleared to zero before the operation begins.

6.1.4 OP's 70 - 79 (Real Instructions) (Continued)

Unnormalized operands are treated as zero in a Divide operation.

The operands are extended as in Add/Sub, above. The extended quotient is computed, and the extended result is normalized with the extended digit discarded after normalization.

For exceptions, including division by zero, see Section 6.1.6.

6.1.5 OP 84 - ACC (Accumulator Manipulate Instruction)

All variants of this instruction reference the entire accumulator with no regard to data type (real or integer) or precision (single or double).

Normalize Accumulator

If all 16 digits of the accumulator mantissa are not zero, shift the entire mantissa left and decrement the exponent by 1 for each digit shift, until the leading digit is non-zero. If all 16 digits of the accumulator are zero, the exponent is set to -99+ (see Section 6.1.6).

Convert Real to Integer

The accumulator mantissa is assumed to be normalized.

If the accumulator exponent is not greater than +07, shift the entire mantissa right and increment the exponent for each digit shift until the exponent is +08 or until the most significant 8 digits of the mantissa are zero. If all 16 digits of the accumulator are now equal to zero, the exponent is set to -99+ (see Section 5.6.1.6). Then clear the least significant 8 digits to zero.

Other Variants

All variants other than those for "normalize accumulator" and "convert read to integer" affect only the 4-digit sign-exponent portion of the accumulator: these variants are described in Section 6.1.1.2, 4-Digit Instruction, and Section 6.1.6.

6.1.6 Setting Proper Sign, Exponent, and Comparisons

For all instructions considered in Section 6.1.3, 6.1.4, and 6.1.5 after successful completion of the operation and before storing the result in memory, the exponent sign and mantissa sign are set to reflect the ASCII mode flip-flop. Furthermore, if all 16 digits of the accumulator mantissa are zero, the 4 digits of the sign- exponent are unconditionally set to -99+. Also, the comparison flip-flops are set to incidate:

High: Mantissa is positive and all 16 digits of the mantissa are not zero

Low: Mantissa is negative and all 16 digits of the mantissa are not zero.

Equal: All 16 digits of the mantissa are zero.

If the operation was not successful, the comparison settings will be:

High = overflow, Low = Underflow, Equal = Division by zero.

6.1.7 Implied Store Operations

An Implied Store instruction is performed exactly as if the corresponding No-Store instruction were followed by the execution of the Store instruction.

6.1.8 Error Traps

Cause of Traps

1) The resultant normalized mantissa is non-zero and the exponent is larger than +99, or the result is an integer value of more than 7 digits (overflow).

2) The resultant normalized mantissa is non-zero and the exponent is algebraically smaller than -99 (underflow).

3) The most significant mantissa digit of the divisor is zero (divide by zero).

6.1.8 Error Traps (Continued)

The above conditions can occur during execution of the following operations:

1) Integer Operators 50 through 57

2) Real Operators 70 through 77

3) The following variants of Operator = 84

   a) Normalize (Underflow)

   b) Real to Integer Conversion (Overflow)

   c) Increment Exponent

   d) Decrement Exponent

Data Handling On Error

Store

The store function does not occur if there is an error in an "Implied Store" operation.

Integer Instructions

Overflow causes the correct but oversized result to be left in the accumulator in a form of double precision in which the least significant digit of the result is located in the LSD position of a real double precision number.

Real Instructions

Division by zero does not change the contents of the accumulator. For all other real instructions, the correct results will be in the accumulator but the magnitude of the exponent will be Mod 100. (The ON state of the overflow flip-flop indicates that the exponent is Modulo 100).

Accumulator Manipulate Instructions

Conversion from real to integer does not take place if it would cause overflow. The other variants are treated as in the real instructions.

6.1.8 Error Traps (Continued)

Trap

Trap Enable

The Trap is enabled if the two digit key stored at base-relative addresses 64 and 65 has all 8 bits on (15, 15 or hexidecimal F, F).

Trap Address

The Trap Address is a 6-digit base-relative address obtained from base-relative locations 66 through 71. The Trap Address is a full 6 digits having no index controller and no address controller. The Trap Address must specify an even address.

Trap Operation

After handling the error the following occurs:

1) Set overflow flip-flop.

2) Set comparison flip-flops as in Section 6.1.6.

3) If the trap has not been enabled, execute the next program instruction.

4) If the trap has been enabled, store the 6-digit base-relative address of the current instruction followed by the 6-digit base-relative address of the next program instruction. The 12-digit field thus constructed will be stored at the location specified by the trap address. After storing the 12-digit field, the instruction located immediately following the 12-digit field will be executed (at the trap address +12).

6.2 ARITHMETIC INSTRUCTIONS, VARIABLE FIELD LENGTH

When the address controller of any adder input specifies 8-bit format, only the less significant 4 bits of each character are effective at the adder input--the other 4 bits are ignored. When 8-bit format is specified by the

6.2 ARITHMETIC INSTRUCTIONS, VARIABLE FIELD LENGTH (Continued)

address controller of the result of any arithmetic operation, the more significant 4 bits of each character are automatically set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code.

The count given by the concatenation of D3, D4 and by the concatenation of D5, D6 specifies the number of digits/characters (information units) in the A and B fields, respectively, but dos not include the sign when the address controller specifies signed data. A count of 00 specifies field length of 100. Full generality of address controller usage applies to all address fields.

In all cases, whether the result field is signed or unsigned, upon successful completion of an operation the comparison of flip-flops are set to indicate whether the result is greater than, equal to, or less than zero. The absolute value of the result is stored when the result field is unsigned.

The sign of a zero result is set positive. An unsigned operand is considered to be positive.

When the length of the result field is not large enough to contain the result of an arithmetic operation without loss of significant leading digits, overflow occurs. Overflow does not occur if leading zero digits or zero characters are lost.

In three-address add or subtract, total overlap of the A and B fields is specified; overlap of the A or B fields by a C field is not allowed. In 2-address add or subtract the two fields may overlap identically.

6.2.1 OP 01 - INC (Add-Two Address)

Algebraically add an addended in location A to an augend in location B and store the sum in location B unless overflow occurs, in which event the overflow flip-flop is set and the B field retains the value it contained before the operation.

6.2.1 OP 01 - INC (Add-Two Address) (Continued)

Addition proceeds from high order to low order. If the addend and augend are of unequal length, as specified in the instruction, the shorter of the two is assumed to be filled with leading zeros until it matches the length of the longer.

If the length of the sum which would have been produced is longer than the specified length of the result B field, an overflow condition occurs. The overflow flip- flop is set and the comparison flip-flop settings are unchanged.

6.2.2 OP 03 - DEC (Subtract-Two Address)

Algebraically subtract a subtrahend in location A from a minuend in location B and store the difference in location B unless overflow occurs, in which event the overflow flip-flop is set and the B field retains the value it contained before the operation.

Subtraction proceeds from high order to low order. If the subtrahend and minuend are of unequal length, as specified in the instruction, the shorter of the two is assumed to be filled with leading zeros until it matches the length of the longer.

If the length of the difference which would have been produced is longer than the specified length of the result B field, an overflow condition occurs. The overflow flip-flop is set and the comparison flip-flop settings are unchanged.

6.2.3 OP 02 - ADD (Add-Three Address)

Algebraically add an addend in location A to an augend in location B, and store the sum in location C unless overflow occurs, in which event the overflow flip-flop is set and the C field retains the value it contained before the operation.

Addition proceeds from high order to low order. If the addend and augend are of unequal length, as specified in the instruction, the shorter of the two is assumed to be

6.2.3 OP 02 - ADD (Add-Three Address) (Continued)

filled with leading zeros until it matches the length of the longer. The C field length is the greater of the A and B field lengths.

If the length of the sum which would have been produced is longer than the specified length of the result field, an overflow condition occurs. The overflow flip-flop is set and the comparison flip-flop settings are unchanged.

6.2.4 OP 04 - SUB (Subtract-Three Address)

Algebraically subtract a subtrahend in location A from a minuend in location B and store the difference in location C unless overflow occurs, in which event the overflow flip-flop is set and the C field retains the value it contained before the operation.

Subtraction proceeds from high order to low order. If the subtrahend and minuend are of unequal length, as specified in the instruction, the shorter of the two is assued to be filled with leading zeros until it matches the length of the longer. The C field length is the greater of the A and B field lengths.

If the length of the difference which would have been produced is longer than the length of the result field an overflow condition occurs. The overflow flip-flop is set and the comparison flip-flop settings are? unchanged.

6.2.5 OP 05 - MPY (Multiply)

Algebraically multiply a multiplicand in location B by a multiplier in location A and store the product in location C.

The C field length is the sum of the A and B field lengths.

Overflow cannot occur. The overflow flip-flop is not affected by this instruction.

In multiplication, totally overlapped operands may be used. However, use of a product field which partially or totally overlaps either the multiplier or the multiplicand field is not allowed.

6.2.6 OP 06 - DIV (Divide)

Algebraically divide a dividend in location B by a divisor in location A and store the quotient in location C and the remainder in location B, unless an overflow condition occurs, in which event the B field and the C field are not altered. A normal and correct divide has no effect on the overflow flip- flop.

If the absolute value of the divisor is not greater than the absolute value of the equivalent number of leading digits of the dividend, the overflow flip-flop is set and the operation is terminated.

If the length of the B field is not greater than the length of the A field, the overflow flip-flop is set and the operation is terminated. The C field length is the difference between the B and A lengths.

The sign of the quotent is positive if the sign of the divisor and dividend are the same or the quotient is zero, otherwise negative. The sign of the remainder after division is that of the original dividend. If the dividend is unsigned, the remainder is unsigned.

No overlap of fields is permitted in division.

Note: See Section 6.20 for Floating Point, Variable-Length field operations.

6.3 BRANCH INSTRUCTIONS

Branching can take place unconditionally or conditionally depending upon the state of one or more of the following processor flio-flops:

Comparison Flip-Flops
Overflow Flip-Flops

Whenever branching is made on the basis of the overflow flip-flop, the flip-flop is automatically reset following the branching.

6.3 BRANCH INSTRUCTIONS (Continued)

Neither conditional nor unconditional branching alter the comparison flip-flops.

The following conditions of branch are available in addition to a "no operator" branch, in which a branch is made to the next instruction and in addition to a "halt and branch" instruction described in Section 6.31.1.

OP 20 - NOP (No Operator)
OP 21 - LSS (Less Than)
OP 22 - EQL (Equal)
OP 23 - LEQ (Less Than or Equal)
OP 24 - GTR (Grater Than)
OP 25 - NEQ (Not Equal)
OP 26 - GEQ (Greater Than or Equal)
OP 27 - BUN (Unconditional)
OP 28 - OFL (Overflow)

All conditional branch instructions based on the comparison flip-flops are effectively a NOP when the comparison flip-flops are cleared to 00.

6.3.1 Address Branch

If the branch is unconditional or if the condition specified for the branch is true, select the Address A as the address specifying the location of the next instruction address in normal sequence.

The address controller bits specify the most significant digit of the address or indirect address as follows: 00 = 0, 01 = 1, 10 = 2, and 11 = indirect address. This permits branching to any address up to and including 299,998 without indexing or address extension. If address extension is used, the address portion of the MSD must be zero or specify indirect address.

6.4 OP 30 - BCT (COMMUNICATE)

See Section 5.1.

6.6 OP 31 - NTR (ENTER)

Store the following in the sequence listed, starting at the base relative location specified in base relative location 00040-00045.

1. The six digit base relative address of the next instruction (that is, the address pointing past the NNNN character following this instruction). Note: For proper operation of the Exit instruction, this address must be less than 300,000.

2. The eight digit contents of IX3.

3. One character indicating the EBCDIC/ASCII mode, overflow and comparison flip-flop states. The four high order bits are set to zero. Bit position four indicates the EBCDIC/ASCII mode flip-flop state, bit position three indicates the overflow flip- flop state, and bit positions two and one indicate the comparison flip-flops states.

4. The NNNN characters following this instruction.

Set the contents of IX3 to the initial address specified in base relative location 00040-00045. The two high order digits of IX3 are cleared to zero.

The EBCDIC/ASCII mode flip-flop remains unchanged, and the comparison and overflow flip-flops are cleared.

Store in base relative location 00040-00045, the base relative address pointing past the new location of the NNNN characters of data which were moved.

Branch unconditionally to the location specified by the A address.

The number of characters moved is given by the count specified in D3, D4, D5 and D6 of this instruction. The maximum number of characters moved is 9,999. The count 0000 moves no data.

6.6 OP 31 - NTR (ENTER) (Continued)

The address controller bits specify the most significant digit of the address or indirect address as follows: 00 = 0, 01 = 1, 10 =2 and 11 = indirect address. This permits branching to any address up to and including 299,998, without indexing or address extension. If address extension is used, the address portion of the MSD must be zero or specify indirect address.

If the A address is indexed by IX3 the initial contents of IX3 is used.

6.7 OP 32 - EXT (EXIT)

Restore the setting of the EBCDIC/ASCII mode, overflow, and comparison flip-flops from the character location specified by IX3 plus 14 if the least significant bit position of the most significant digit of the character contains a zero bit. If it contains a one bit, leave the respective flip-flops unchanged.

Transfer the least significant six digits contained in IX3 to base relative location 00040-00045.

Transfer the eight digits at the location specified by IX3 plus 6 to IX3.

Branch unconditionally to the instruction specified by the A address. The address controller bits specify the most significant digit of the address or indirect address as follows: 00 = 0, 01 = 1, 10 = 2 and 11 = indirect address. This permits branching up to and including 299,998 without indexing or address extension. If address extension is used, the address portion of the MSD must be zero or specify indirect address.

Instruction format B is used.

Note: The next instruction address is obtained during the fetch cycle and thus if the A address is indexed by IX3 the initial contents of IX3 are used.

6.7 OP 32 - EXT (EXIT) (Continued)

Note: The normal return address can be obtained as follows:

Address Controller = 11
Address Index = 11

6.8 OP 08 - MVD (MOVE DATA 3-ADDRESS)

This instruction is invalid unless the Extended Function Adapter hardware is installed.

Move words from the origin location specified by the A address to the destination location specificed by the B address until terminated by the B address equal to the C address.

A word is interpreted as 16 bits (2 characters) of information.

All Addresses must be divisible by four without a remainder.

No information is moved into the location specified by the C address if D4 = 0.

No information is moved into the location specified by the B address if D4= 1.

All address controllers must specify unsigned 4-bit format or indirect address.

Digits D3, D5 and D6 are reserved and must be zero.

D4 = 0 Move then increment the A and B addresses.

D4 = 1 Decrement A and B addresses then move (move backward).

The condition toggles (ASCII, Overflow, Comparison) are unchanged.

6.9 OP 10 - MVA (MOVE ALPHANUMERIC)

Move characters or digits from the origin location specified by the A address to the destination location specified by the B address. The count given by the concatenation of D3, D4 and by the concatenation of D5, D6 specifies the length of the A and B fields, respectively, but does not include the sign of the signed fields. A count of 00 specifies a field length of 100. Full generality of address controller usage applies to both addresses.

If the destination field length is shorter than the origin field length, the origin field is truncated from the right and the overflow flip-flop is set. The move is completed.

If the destination field length is longer than the origin field length, the destination field is filled in with trailing "blank" characters for 8-bit format or trailing "zero" digits for signed or unsigned 4-bit format.

When both address controllers specify 8-bit format, each character is moved.

When both address controllers specify unsigned 4-bit format, each digit is moved.

When both address controllers specify signed 4-bit format, each digit is moved and the receiving field sign digit is set to indicate the appropriate algebraic sign for the selected (EBCDIC or ACSII) code.

When the address controllers of the A and B address specify 8-bit and signed 4-bit format, respectively, only the low order 4 bits of each character are moved, but the most significant four bits portion of the A field is inspected for sign and the receiving field sign digit is set to indicate the appropraite algebraic sign for the selected (EBCDIC or ASCII) code.

When the address controllers of the A and B addresses specify 8-bit and unsigned 4-bit format, respectively, only the low order 4 bits of each character are moved, the other 4 bits are ignored.

6.9 OP 10 - MVA (MOVE ALPHANUMERIC) (Continued)

When the address controllers of the A and B address specify signed 4-bit and 8-bit format, respectively, the high order four bits of each character in the destination field are set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code except that the high order 4 bits of the most significant character are set to indicate the appropriate algebraic sign for the selected (EBCDIC- ASCII) code.

When the address controllers of the A and B addresses specify unsigned 4-bit and 8-bit format, respectively, the high order 4 bits of each character in the destination field are set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code.

When the address controllers of the A and B addresses signed 4-bit and unsigned 4-bit format, respectively, each digit is moved and the sign is not moved.

When the address controllers of the A and B addresses specify unsigned 4-bit and signed 4-bit format, respectively, each digit is moved and the sign of the receiving field is set to the selected (EBCDIC or ASCII) positive sign code.

Upon completion of the instruction of the comparison flip-flops are set to indicate whether the value of the portion of the origin field moved was greater than, equal to, or less than zero. For an 8-bit origin field, only the low order digit of each character moved is inspected for the equal to zero condition. The sign of the origin field is always inspected when the origin field is signed, regardless of the format of the receiving field. When the origin field specifies 8-bit format, the origin field is inspected for a sign in the most signficant 4 bits of the most significant character when and only when the receiving field specifies signed four bit format. In all cases the origin field is assumed positive.

Identical overlap of the A and B fields is permitted. See Section 3.9 for permissible partial overlap.

6.10 OP 11 - MVN (MOVE NUMERIC)

Move characters or digits from the origin location specified by the A address to the destination location specified by the B address.

The count given by the concatenation of D3, D4 and by the concatenation of D5, D6 specifies the length of the A and B fields, respectively, but does not include the sign of signed fields.

A count of 00 specifies a field length of 100. Full generality of address controller usage applies to both addresses.

If the destination field length is shorter than the origin field length, the origin field is truncated from the left, if and only if, the truncation affects characters with a low digit of zero or zero digits. If characters with a non-zero low order digit or non-zero digits are affected, the overflow flip-flop is set, the comparison flip-flops are unchanged and all fields are unchanged.

If the destination field length is longer than the origin field length the destination field is filled in with leading "zero" characters or digits.

When both address controllers specify signed 4-bit sign digit is set to indicate the appropriate algebraic sign for the selected (EBCDIC or ASCII) code.

When the address controllers of the A and B addresses specify 8-bit and signed format, respectively, only the low order 4 bits of each character are moved, but the most significant four bits of the A-field are inspected for sign and the receiving field sign digit is set to indicate the appropriate algebraic sign for the selected (EBCDIC or ASCII) code.

When the address controllers of the A and B addresses specify 8-bit and unsigned 4-bit format, respectively, only the low order 4-bits of each character are moved, the other 4 bits are ignored.

6.10 OP 11 - MVN (MOVE NUMERIC) (Continued)

When the address controllers of the A and B addresses specify signed 4-bit and 8-bit format, respectively, the high order 4-bits of each character in the destination field are set to the code (1111 or 01010) indicating the numeric subset of the selected (EBCDIC or ASCII 8-bit code except that the high order 4-bits of the most significant character are set to indicate the appropriate algebraic sign for the selected (EBCDIC or ASCII) code.

When the address controllers of the A and B addresses specify unsigned 4-bit and 3-bit format, respectively, the high order 4-bits of each character in the destination field are set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code.

When the address controllers of the A and B addresses specify signed 4-bit and unsigned 4-bit format, respectively, each digit is moved and the sign is not moved.

When the address controllers of the A and B addresses specify unsigned 4-bit and signed 4-bit format, respectively, each digit is moved and the sign of the receiving field is set to the selected (EBCIDC or ASCII) positive sign code.

When both address controllers specify 8-bit format, only the low order 4 bits of each character are moved. The high order bits of each character in the destination field are set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCIDC or ASCII) 8-bit code.

When both address controllers specify unsigned 4-bit format, each digit is moved.

Upon completion of the instruction the comparison flip-flops are set to indicate whether the portion of the origin field moved was greater than, equal to, or less than zero. For the case of 8-bit origin field, only the low order digit of each character moved is inspected for the equal to zero condition. The sign of the origin field is always inspected when the origin field is signed, regardless of the format of the receiving field. When the

6.10 OP 11 - MVN (MOVE NUMERIC) (Continued)

origin field specifies 8-bit format, the origin field is inspected for a sign in the most significant 4-bits of the most significant character when and only when the receiving field specifies signed four-bit format. In all other cases the origin field is assumed positive.

See Section 3.9 for permissible partial overlap. A and B fields can overlap indentically.

6.11 OP 12 - MVW (MOVE WORD)

Move Words from the origin location specified by the A address to the destination location specified by the B address.

A word is interpreted as 16 bits (2 characters) of information.

The origin and destination addresses must be divisible by four without a remainder--that is, they are synchronized to word boundaries.

The number of words moved is specified by the increment NNNN contained in the four digits D3, D4, D5 and D6 of this instruction. The maximum number of words moved is 10,000 and is determined by using a word count of 0000. Both address controllers must specify unsigned 4-bit format or indirect address. If indirect address is specified, the final address must specify unsigned 4-bit format.

The condition toggles are unchanged.

See Section 3.9 for permissible partial overlap. A and B fields can overlap identically.

6.12 OP 13 - MVC (MOVE AND CLEAR WORD)

Move Words from the origin location specified by the A address to the destination location specified by the B address. The origin location is cleared to zero before the word is written into the destination location.

6.12 OP 13 - MVC (MOVE AND CLEAR WORD) (Continued)

A word is interpreted as 16 bits (2 characters) of information.

The origin and destination addresses must be divisible by four without a remainder--that is, they are synchronized to word boundaries.

The number of words moved is specified by the increment NNNN contained in the four digits D3, D4, D5 and D6 of this instruction. The maximum number of words moved is 10,000 and is determined by using a word count of 0000. Both address controllers must specify unsigned 4-bit format or indirect address. If indirect address is specified, the final address must specify unsigned 4-bit format.

Toggles are unchanged.

See Section 3.9 for permissible partial overlap. A and B fields can overlap identically.

6.13 OP 14 - MVR (MOVE REPEAT)

Move characters or digits from the origin location specified by the A address to the destination location specified by the B address.

The count given by the concatenation of D3, D4 specifies the length of the A field. The count given by the concatenation of D5, D6 specifies the number of repetitions of the move. A count of 00 in the respective positions specifies the maximum field length of 100 and the maximum number of repetitions of 100. The length of the B field is given by the product of the A field length and the number of repetitions.

Both address controllers can specify unsigned 4-bit format, 8-bit format or indirect address.

When both address controllers specify 8-bit format, each character is moved.

6.13 OP 14 - MVR (MOVE REPEAT) (Continued)

When both address controllers specify unsigned 4-bit format, each digit is moved.

When the address controllers of the A and B address specify 8-bit and unsigned 4-bit format, respectively, only the low order 4 bits of each character is moved, the other 4 bits are ignored.

When the address controllers of the A and B address specify unsigned 4-bit and 8-bit format, respectively, the high order 4 bits of each character in the destination field are set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code.

The condition toggles are unchanged.

See Section 3.9 for permissible partial overlap. A and B fields can overlap identically. Move Links.

6.14 OP 45 - CPA (COMPARE ALPHANUMERIC)

Compare, according to the (Binary) collation sequence, the A field with the B field and set the comparison flip-flops to indicate whether the A field is less than (low), equal to (equal), or greater than (high) the B field.

The overflow flip-flop is not affected by this instruction.

The count given by the concatenation of D3, D4, and by the concatenation D5, D6 specifies the length of the A and B field, respectively. Both address controllers must specify 8-bit format or indirect address. If indirect address is specified the final address must specify 8-bit format. Maximum field length is 100 and is determined by a field count of 00.

All eight bits of a character are used in the comparison. If the two fields are of unequal length, the number of characters of the shorter field is made equal to the longer field by assuming trailing "blank" characters.

6.14 OP 45 - CPA (COMPARE ALPHANUMERIC) (Continued)

Note: EBCDIC blank is the same as ASCII blank 0100 0000.

The instruction is terminated immediately with the appropriate comparison condition set, if an unequal condition is detected.

Both fields retain their original values in memory after execution of this instruction.

A and B fields may overlap identically. No partial overlap is allowed.

6.15 OP 46 - CPN (COMPARE NUMERIC)

Compare algebraically the A field with the B field and set the comparison flip-flops to indicate whether the A field is less than (low), equal to (equal), or greater than (high) the B field. The comparison is algebraic with numeric format.

The overflow flip-flop is not affected by this instruction.

The count given by the concatenation of D3, D4 and by the concatenation D5, D6 specify the lengths of the A and B fields, respectively, but does not include the sign of the signed fields. Full generality of address controller usage applies to both addresses. Maximum field length is 100 and is determined by a field count of 00.

When the address controller specifies 8-bit format in either or both addresses, only the less significant 4 bits of each character are compared, the other 4 bits are ignored. The sign is assumed positive. Note that 8-bit format comparisons are not algebraic.

If the two fields are of unequal length, the number of characters/digits of the shorter field is made equal to the longer field by assuming leading zeros.

The instruction is terminated immediately with the appropriate comparison condition set, if an unequal condition is detected.

6.15 OP 46 - CPN (COMPARE NUMERIC) (Continued)

Unsigned fields are assumed positive (8-bit or unsigned 4-bit format).

The comparison of minus zero with plus zero indicates an equal condition.

Both fields retain their original value in memory after execution of this instruction.

A and B fields may overlap identically. No partial overlap is allowed.

6.16 OP 15 - TRN (TRANSLATE)

Translate (perform suostitution of) characters of digits from the location given by the A address using the substitution table located at the B address and store the result in the location given by the C address.

The number of characters or digits translated is given by the number NNNN specified in the four digits positions D3, D4, D5 and D6 of this instruction. The maximum number of characters or digits translated is 10,000 and is determined by a count of 0000.

The B address must be specified as BB000, that is, the translation table address must contain zeros in the three least significant 4-bit digit positions.

When the address controller of the A field specifies 8-bit format, the character to be translated is interpreted as a nine bit character by assuming a zero appended on the right. When the address controller of the A field specifies sign or unsigned 4-bit format, 4-bits (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code are assumed to be appended on the left and a zero assumed to be appended on the right. The resultant 9-bit character is split into groups of three and inserted into the three low-order digit positions of the B address. The most significant bit of each of the three digits positions is set to zero.

6.16 OP 15 - TRN (TRANSLATE) (Continued)

The 8-bit character at the resultant address in the translation table is transferred into the destination string beginning at the C address.

Standard address controller usage applies to the A address. The address controller for B address must specify 8-bit format or indirect address. The address controller for C address must specify 8-bit format or indirect address. If indirect address is speified, the final address must specify 8-bit format.

Identical A and C fields are permitted. No other overlap is allowed.

The digit position in the A field containing the sign is ignored when the address controller specifies signed 4-bit format.

6.17 OP 09 - MVL (MOVE LINKS)

Move the contents of the location specified by the A address to a register. Move the contents of the location specified by the B address to the location specified by the A address. Move the contents of the location specified by the C address to the location specified by the B address. Store the contents of the register (original contents of the location specified by the A address) into the location specified by the C address.

Results: If none of the A, B, and C fields overlap, this instruction moves the starting A-field data to the C-field, the starting B-field data to the A-field, and the starting C-field data to the B-field.

The length specified by the concatenation of D3 and D4 is the length of all fields. A count of 00 represents the maximum of 100.

Standard address controller usage applies to all addresses. The final address controllers for all three addresses must be identical. The C-address control is used. If the C-address controller specifies 3-bit format, an odd A or B address causes an address error and results in unspecified erroneous results. See Section 4.1.

6.17 OP 09 - MVL (MOVE LINKS) (Continued)

Only two uses of identical fields are specified: (1) field B identical to field C, and (2) field A identical to field B. No partial overlap is allowed.

6.18 SCAN INSTRUCTIONS

The count given by the concatenation of D3, D4 and by the concatenation of D4 and D6 specifies the length of the A and B fields respectively. Maximum field length is 100 and is determined by a field count of 00.

When an address controller specifies unsigned 4-bit format, a high order digit (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code is assumed to be appended on the left of each digit being compared.

Neither partial nor total overlap is allowed in fields used in Scan instructions.

6.18.1 OP 16 - SDE (Scan to Delimiter - Equal)

The first character or digit from the location given by the B address is compared with a sequence of characters or digits starting in the location given by the A address.

If the first character from the B field is not equal to any of the characters from the A field, the next character from the B field is accessed and compared with the sequence of characters from the A field. The process continues until the B field is exhausted or until a character from the B field is found that is equal to any of the characters from the A field.

If the first character from the B field is equal to any of the characters from the A field, the comparison condition low is set and the number 00 is stored in base relative location 38-39. If the B field is exhausted without detection of a character from the B field that is equal to any of the characters from the A field, the comparison condition high is set and the number of characters in the B field minus one is stored in base relative location 38-39.

6.18.1 OP 16 - SDE (Scan to Delimiter - Equal) (Continued)

If a character from the B field other than the first is detected that is equal to any of the characters from the A field, the comparison condition equal is set and the number of characters of the B field compared prior to the detection of the equal character is stored in base relative location 38-39.

6.18.2 OP 17 - SDU (Scan to Delimiter - Unequal)

The first character or digit from the location given by the B address is compared with a sequence of characters or digits starting in the location given by the A address.

If the first character from the B field is equal to any of the characters from the A field, the next character from the B field is accessed and compared with the sequence of characters from the A field. The process continues until the B field is exhausted or until a character from the B field is found that is not equal to any of the characters from the A field.

If the first character from the B field is not equal to any of the characters from the A field, the comparison condition low is set and the number 00 is stored in base relative location 38-39.

If the B field is exhausted without detection of a character from the B field that is not equal to any of the characters from the A field, the comparison condition high is set and the number of characters in the B field minus one is stored in base relative location 38-39.

If a character from the B field other than the first is detected that is not equal to any of the characters from the A field, the comparison condition equal is set and the number of characters of the B field compared prior to the detection of the not equal character is stored in base relative location 38-39.

6.18.3 OP - SZE (Scan to Delimiter - Zone Equal)

This instruction is the same as Scan to Delimiter - Equal with the following exceptions:

1. Address controllers may specify indirect address or 8-bit format only.

2. The low order digit of each character is ignored in the comparison.

6.18.4 OP 19 - SZU (Scan to Delimiter - Zone Unequal)

This instruction is the same as Scan to Delimiter - Unequal with the following exceptions:

1. Address controllers specify indirect address or 8-bit format only.

2. The low order digit of each character is ignored in the comparison.

6.19 OP 49 - EDT (EDIT - DATA MOVEMENT)

Move data from the A field (source) under control of the micro-operators given in the B field to the C field (destination).

The three fields--A, B and C-- are completely separate, except that the A and C fields can be identical if the B field is completely separate.

Digits D3 and D4 are reserved and must be zero. The count given by the concatenation of D5 and D6 specifies the number of micro-operators in the B field. A count of 00 specifies a field length of 100. Full generality of address controller usage applies to the source (A) field. The address controller of the micro-operator (B) field and the destination (C) field must specify 8-bit format or indirect address. If indirect address is specified, the final address must specify 8-bit format. The source field is considered positive for unsigned 4-bit format. For 8-bit format, the most significant digit of the most significant character is considered to be a sign. For

6.19 OP 49 - EDT (EDIT - DATA MOVEMENT) (Continued)

signed 4-bit format, the sign is considered to be the leading digit of the field in the usual manner. After the operation, the comparison flip-flops indicate whether the source field is positive, negative or zero, if at least one character from the source field has been moved. If no character is moved, the comparison flip-flops are set to zero condition. The low order digit of a character is inspected for the zero condition.

The A and B field are not altered following completion of this instruction.

The overflow flip-flop is cleared to zero at the completion of this instruction.

The instruction is terminated by the exhaustion of the B field.

The B field consists of a string of two-digit micro-operators. Each micro-operator has the form MA, where M and A are digits. M refers to the operation code and A, in most cases to the 0th, 1st, 2nd, ...7th character of a table in a base relative location.

"T" denotes a toggle which is zero initially, and which is set to one when zero suppression ends.

"Q" denotes a toggle which is zero initially, and which is set to one if check protection is desired.

"S" denotes a toggle which is set to zero if the A field sign is negative and to one otherwise. Unsigned 4-bit format fields are considered positive.

The following three micro-operators contain a repeat count in the A (2nd digit) portion of the micro- operator. Zero represents no repeat.

Move Digit,	M = 0, A = 0...9
Move Character,	M = 1, A = 0...9
Move Suppess,	M = 2, A = 0...9

6.19.1 Move Digit

Set T = 1 and move a digit/character from the (A) source field to the (C) destination field with the more significant 4 bits of the digit/character set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code.

6.19.2 Move Character

Set T = 1 and move a digit/character from the (A) source field to the (C) destination field, the more significant 4 bits in the case of a digit is set to the code (1111 or 0101) indicating the numeric subset of the selected (EBCDIC or ASCII) 8-bit code. A character is moved unchanged.

6.19.3 Move Suppress

If T = 1 or if T = 0 and the source digit or the low order digit of the source character is not zero, the operation move digit is performed. If T = 0 and the source digit or the low order digit of the source character is zero, and Q = 0, move a blank (0100 0000) to the destination; if Q = 1, move table entry two (normally an asterisk) from the table to the destination. The following micro-operator contains a control meaning in the A (2nd digit) portion of the micro-operator.

Set T = 0	M = 9, A = 0
Set T = 1	M = 9, A = 1
Complement Q	M = 9, A = 2
Skip source digit or character	M = 9, A = 3

The following six micro-operators contain in the A (2nd digit) portion of the micro operator, either a pointer referring to a particular character in a table located in fixed base relative location in memory (A = 0...7 refers to the 0th, 1st...7th character), or a code denoting a reference to a table entry or a non-table entry and an action to be performed.

6.19.3 Move Suppress (Continued)

Insert Unconditionally	M = 3
Insert on Plus (=1)	M = 4
Insert on Minus (S = 0)	M = 5
Insert Suppress	M = 6
Insert Float	M = 7
End Float	M = 8

The following actions are performed for the Code A = 1000, 1001, 1010, and 1011 in the above six micro-operations only if during the execution of the M portion of the micro-operator reference is made to the A portion.

For A = 1000, if S = 1, use table entry 0 and if S = 0, use table entry 1.

For A = 1001, if S = 1, use blank (0100 0000) in lieu of a table entry and if S = 0, use table entry 1.

For A = 1010, if S = 0, use blank (0100 0000) in lieu of a table entry and if S = 1, use table entry 0.

For A = 1011, the next 8-bit character in the mask is not a micro-operator but is to be used as an 8-bit character in the same manner as an entry in the table. In all cases, the next micro-operator is obtained following this 8-bit character.

6.19.4 Insert Unconditionally

Move the character from the "A" location in the table to the destination field.

6.19.5 Insert on Plus

If S = 1, move the character from the "A" location in the table to the destination field. If S = 0 and Q = 0, move blank (0100 0000) to the destination. If S = 0 and Q = 1, move table entry two (normally an asterisk) from the table to the destination.

6.19.6 Insert on Minus

If S = 0, move the character from the "A" location in the table to the destination field. If S = 1 and Q = 0, move blank (0100 0000) to the destination. If S = 1, and Q = 1, move table entry two (normally an asterisk) from the table to the destination.

6.19.7 Insert Suppress

If T = 1, move the character from the "A" location in the table to the destination field. If T = 0 and Q = 0, move a blank (0100 0000) to the destination. If T = 0 and Q = 1, move table entry two (normally an asterisk) from the table to the destination.

6.19.8 Insert Float

If T = 1, the operation move digit is performed. If T = 0, and the source digit or the low order digit of the source character is not zero, move the character from the "A" location in the table to the destination field and then perform the operation move digit. If T = 0 and the source digit or the low order digit of the source character is zero then if Q = 0 move a blank (0100 0000) to the destination field. If Q = 1 move table entry two (normally an asterisk) from the table to the destination.

6.19.9 End Float Mode

If T = 0 move the character from the "A" location in the table to the destination field. If T = 1, go to the next micro-operator.

6.20 FLOATING POINT ARITHMETIC, VARIABLE LENGTH FIELD

The representation of a Floating Point number is:

S/X Exp S/M Mantissa
where:

6.20 FLOATING POINT ARITHMETIC, VARIABLE LENGTH FIELD (Continued)

S/M is the sign of the Mantissa (1 digit)
S/X is the sign of the Exponent (1 digit)
Exp is the Exponent (2 digit)
Mantissa (variable length, decimal at right)

Exponents and Mantissa are integers.

The decimal counts given by D3D4 and D5D6 (each 1 to 99 and 00 = 100) specify the numbers of digits of the mantissa in the A and B field, respectively, but do not include the signs or exponent digits. Note: Although field length refers only to mantissa digits, signs and exponent digits are always present. Sign convention is the same as in fixed point arithmetic. The address controllers must specify signed 4-bit format and/or indirect address. If indirect address is specified, the final address controller must specify signed 4-bit format.

Zero has a mantissa of all zeros and an exponent of -99. A zero can be used as an operand in any FP operation (see FP Divide for special cases).

If the most significant digit of a non-zero mantissa is zero, that F.P. number can be used as an operand in addition and subtraction. However, if it is used as an operand in multiplication or division it is treated as a zero regardless of the size of its exponent.

Results of all floating point operations are normalized (exception - remainder in divide).

The comparison flip-flops settings indicate whether the result is greater than, equal to, or less than zero. A zero result sets comparison conditions to equal. (Exception - flagged underflow).

"Overflow" occurs when a result exponent exceeds +99. Overflow sets the overflow flip-flop and the comparsion high condition. No fields in memory are altered. Exponent overflow can occur with any of the four F.P. operations.

Definition of underflow depends on the operation. The underflow flag is overflow and low comparison flip-flops on.

6.20.1 Alignment of Operands for Addition and Subtraction

When exponents of the two operands used in floating point addition or subtraction are algebraically equal, or become so during the automatic procedure A and/or B, described below, the fixed point operation is performed on the mantissas of the equal exponents.

If field lengths of the operands are equal, procedure "B" is used. Otherwise, the operand with the smaller field length is adjusted.

Procedure A: If the exponent to be adjusted is algebraically larger than the other, count down that exponent and count up its mantissa field length (effectively shifting left and adding trailing zeros). Continue adjustment until either (1) the adjusted exponent becomes equal to that of the other operand, or (2) the field lengths become equal but exponents are not equal. If (2) occurs, adjust the other operand by procedure B.

Procedure B: If the exponent to be adjusted is algebraically smaller than the other, count up that exponent and count down its mantissa field length (effectively shifting right and truncating the mantissa LSD each time the exponent is increased by one). Continue adjustment until either (1) the adjusted exponent becomes equal to that of the other operand, or (2) the shorter field is exhausted. If (2) occurs, the result is the floating point operand with the larger exponent.

6.20.2 OP 80 - FAD (Floating Point Add)

These instructions are invalid unless the Floating Point Adapter hardware is installed.

After proper alignment of operands, algebraically add an addend located in A to an augend located in B and store the result in C. Operands need not be in normalized form. (Result is normalized.)

The result mantissa length (in digits) is the same as the larger of the A and B field lengths.

6.20.2 OP 80 - FAD (Floating Point Add) (Continued)

If the result exponent is greater than +99 (possible only if mantissa overflow increments a +99 result exponent) overflow occurs.

If mantissa overflow occurs the result exponent is incremented by one, and a leading one is supplied as the first result digit in the mantissa. The least significant digit of the unadjusted result mantissa is dropped.

If a normalized or unnormalized result exponent is less than -99, the result is set to zero. The underflow flag is not set.

In Floating Point Add no fields are to overlap. (Exceptions: (1) A and B fields are identical in sign, exponent, and mantissa, (2) B and C fields are identical and A field value is -99 + 0.)

6.20.3 OP 81 - FSU (Floating Point Subtract)

After proper alignment of operands, algebraically subtract a subtrahend located in A from a minuend located in B and store the result in C. Operands need not be in normalized form.

The result mantissa length in digits is the same as the length of the larger of the A and B field lengths.

If mantissa overflow occurs, the result exponent is incremented by one. A leading one is supplied as the first result digit in the mantissa, and the least significant digit of the unadjusted result mantissa is dropped.

If the result exponent is greater than +99 (possible only if mantissa overflow increments a +99 result exponent) overflow occurs.

If the result exponent is less than -99, before or after normalization, the result is set to zero. The underflow flag is not set.

6.20.3 OP 81 - FSU (Floating Point Subtract) (Continued)

In Floating Point Subtract, no fields are to overlap. Exception: A and B fields can be identical.

6.20.4 OP 82 - FMP (Floating Point Multiply)

Algebraically multiply a multiplicand located in B by a multiplier located in A and store the result in C. Operands are assumed to be in normalized form. A non-normalized operand is treated as zero.

The result exponent is the sum of the operand exponents, decremented by one if required for normalization. The length of the result mantissa is the sum of the operand lengths. The result mantissa is the normalized product of the operand mantissas.

Mantissa overflow cannot occur. Exponent overflow can occur.

If the result exponent is -99 or less before normalization, the underflow flag is set if both operands are non-zero; otherwise, a zero result is stored and comparison flip-flop is set to equal.

In Floating Point Multiply identical overlap of the A and B fields is the only permissible overlap.

6.20.5 OP 83 - FDV (Floating Point Divide)

Algebraically divide a dividend located in B by a divisor located in A; store the quotient in C and the remainder in B. Operands are assumed to be in normalized form. An operand with a mantissa MSD of zero is used as zero. The quotient is normalized. The remainder has the same field length, sign, and exponent as the original B field and is not normalized.

The C field length is the difference between the B and A field lengths. The preliminary result exponent is the difference between the dividend exponent and the divisor exponent.

6.20.5 OP 83 - FDV (Floating Point Divide) (Continued)

In Floating Point Division, when the mantissas meet conditions (1) and (2) below, regular fixed division is performed.

(1) The dividend mantissa field length (B) is greater than the divisor mantissa field length (A).

(2) The absolute value of the divisor mantissa is greater than the absolute value of the equivalent number of leading digits of the dividend mantissa.

If condition (1) above is not true, overflow occurs, the operation is terminated, and no field contents are altered.

If condition (2) is not true but (1) is true, the preliminary result exponent is incremented by one (as if a leading zero were annexed to the dividend mantissa), and the division proceeds if the preliminary exponent was less than +99.

If the result exponent falls below -99, the underflow flag is set if both operands are non-zero; otherwise a zero result is stored and the comparison flio-flop is set equal. Division of zero by non-zero quantities yields a zero result. Division of zero or of non-zero quantities by zero causes overflow.

Use of either partially or totally overlapped fields in FP Divide is not permitted.

6.21 BIT TEST, BIT SET, BIT RESET INSTRUCTIONS

These instructions are invalid unless the Extended Function Adapter hardware is installed.

Characters/digits from the A field are accessed and manipulated with the appropriate portion of the mask in digit positions D5 and D6 as indicated below.

If the address controller specifies unsigned 4-bit format and the number of digits accessed is even, the entire 8-bit mask is applied to successive groups of two digits of the A field. If the number of digits is odd the operation is the

6.21 BIT TEST, BIT SET, BIT RESET INSTRUCTIONS (Continued)

same until the last digit is accessed. The more significant 4-bit portion of the mask is applied to this digit.

The number of characters/digits treated is specified by the digits D3 and D4 of the instruction. 00 represents the maximum of 100.

The address controller must specify unsigned 4-bit format, 8-bit format or indirect address.

No fields in memory are altered for bit test.

6.21.1 OP 40 - BZT (Bit Zero Test)

The comparison condition equal is set if a zero bit is encountered in any data character/digit position which corresponds to a one bit in the mask. The comparison condition high is set otherwise.

6.21.2 OP 41 - BOT (Bit One Test)

The comparison condition equal is set if a one bit is encountered in any data character/digit position which corresponds to a one bit in the mask. The comparison condition high is set otherwise.

6.21.3 OP 33 - BRT (Bit Reset)

This instruction is invalid unless the Extended Function Adapter hardware is installed.

The bits in the operand at the location specified by the A address are reset if the corresponding bits are one bits in the operand (mask) given by D5 D6. The comparison condition high is set if the least significant bit of the result field is one. Otherwise it is set equal.

6.21.4 OP 34 - BST (Bit Set)

This instruction is invalid unless the Extended Function Adapter hardware is installed.

Set the bits in the character/digit specified by the A address if the corresponding bits in the mask, D5 D6, are ones.

The comparison condition high is set if the least significant bit of the result field is one. Otherwise it is set equal.

6.22 OP 42 - AND (AND)

The logical product (And) of the operand at the location specified by the A address with the operand at the location specified by the B address is stored at the location specified by the C address.

The counts given by the concatenation of D3D4 and by D5D6 specify the lengths of the A and B fields, respectively. Maximum field length is 100 and is determined by a field count of 00. The length of the C field is given by the larger of the two.

The address controllers must specify unsigned 4-bit format, 8-bit format, or indirect address. If indirect address is specified, the final address must specify unsigned 4-bit format or 8-bit format. The final address controllers for all three addresses must be identical.

If the two fields are of unequal length, the number of characters/digits of the shorter is made equal to that of the longer by assuming trailing characters/ digits whose bits are all zeros.

The comparison condition high is set if the least significant bit of the result field is one. Otherwise it is set equal.

Any two of the three fields may be identical. No other overlap is specified.

6.23 OP 43 - ORR (OR)

The logical sum (or) of the operand at the location specified by the A address with the operand at the location specified by the B address is stored at the location specified by the C address.

The counts given by the concatenation of D3, D4 and by the concatenation of D5, D6 specify the lengths of the A and B fields, respectively. Maximum field length is 100 and is determined by a field count of 00. The length of the C field is given by the larger of the two.

The address controllers must specify unsigned 4-bit format, 8-bit format or indirect address. If indirect address is specified the final address must specify unsigned 4-bit format or 8-bit format. The final address controllers for all three addresses must be identical.

If the two fields are of unequal length, the number of characters/digits of the shorter is made equal to that of the longer by assuming trailing characters/ digits whose bits are all zeros.

The comparison condition high is set if the least significant bit of the result field is one. Otherwise it is set equal.

Any of the three fields may be identical. No other overlap is specified.

6.24 OP 44 - NOT (NOT)

The modulo two sum (Exclusive Or) of the operand at the location specified by the A address with the operand at the location specified by the B address is stored at the location specified by the C address.

The count given by the concatenation of D3, D4 and by the concatenation of D5, D6 specifies the length of the A and B field, respectively. Maximum field length is 100 and is determined by a field count of 00. The length of the C field is given by the larger of the two.

6.24 OP 44 - NOT (NOT) (Continued)

The address controllers must specify unsigned 4-bit format, 8-bit format, or indirect address. If indirect address is specified the final address must specify unsigned 4-bit format or 8-bit format. The final address controllers for all three addresses must be identical.

If the two fields are of unequal length, the number of characters/digits of the shorter is made equal to that of the longer by assuming trailing characters/digits whose bits are all ones.

The comparison condition high is set if the least signifiant bit of the result field is one. Otherwise it is set equal.

Any two of the three fields may be identical. No other overlapping is specified.

6.25 HALT INSTRUCTIONS

Halt instructions are dependent upon an execution digit located in the Process State Record (PSR) which determines the course of action as follows:

Digit = 0 All halts are executed.
= 1 Normal state halts are ignored.
= 2 Control state halts are ignored.
= 3 All halts are ignored.
= 4 All halts are considered invalid instructions.
= 5 Normal state halts are ignored and control state halts are considered invalid instructions.
= 6 Control state halts are ingored and normal state halts are considered invalid instructions.
= 7 All halts are ignored.

6.25.1 OP 29 - HBR (HALT, BRANCH)

Select the address A as the address specifying the location of the next instruction whether the halt is executed or ignored.

6.25.1 OP 29 - HBR (HALT, BRANCH) (Continued)

The address controller bits specify the most significant digit of the address or indirect address as follows: 00 = 0, 01 = 1, 10 = 2 and 11 = indirect address. This permits branching to any address up to and including 299,998 without indexing or address extension. If address extension is used, the address portion of the MSD must be zero or specify indirect address.

6.25.2 OP 48 - HBK (Halt, Breakpoint)

An eight bit character from base relative location 46-47 is accessed and tested with an eight bit mask character in digit positions D5 and D6 of this instruction. If the eight bit character from base relative location 46-47 has one bit which corresponds to a one bit in the mask the execution digit in the PSR is accessed. This digit then determines the course of action.

If no correspondence is obtained, the halt is ignored, regardless of the execution digit. That is, the next instruction is selected in normal sequence.

Digit positions D3 and D4 are not reserved and can obtain any combination of bits including the bit combination normally specifying literals. However, indirect field length is operative and the combination? of bits specifying indirect field length will insert the value obtained from the address specifed. These bit have no significance in the execution of the instruction.

6.26 SEARCH COMMANDS

6.26.1 OP 39 - SEA (Search)

Compare the A field with the B, B+NN, B+2(NN), B+3(NN), ...fields in the manner orescribed by the variants in the C address controller until the incremented B field address is equal to or greater than the C address, or unless terminated otherwise.

6.26.1 OP 39 - SEA (Search) (Continued)

The comparison as determined by the A address controller is as follows:

00 = 4-Bit Binary
01 = Signed 4-Bit Algebraic
10 = 8-Bit Binary

The increment NN as determined by D5-D6, and the B address controller is as follows:

D5-D6	B Controller	Increment
NN	10	NN Characters
NN	00	NN Digits
NN	01	NN +1 Digit

Indexing and indirect address are allowed on all three addresses.

The overflow flip-flop is cleared to zero after the completion of the instruction.

The length of both the A field and the B field is given by the decimal count of D3D4 (00=100), but does not include the sign of the signed fields. Length is independent of and can be larger than the increment.

If the C address controller bits are 00, then the A field is compared with the B field and each subsequent B field for an equal condition. If and when an equal condition is detected, the comparison flip-flops are set equal and the address of the pertinent B field is stored in IX1. The Search is then complete. If an equal condition is not detected, the comparison flip-flops are set high and IX1 is unchanged.

If the C address controller bits are 01, then the A field is compared with the B field and each subsequent B field for a low condition. If and when a B field is detected that is lower than the A field, the comparison flip-flops are set equal and the address of the pertinent B field is set to IX1, and the Search is complete. If a low condition is not detected, the comparison flip-flops are set high and IX1 is unchanged.

6.26.1 OP 39 - SEA (Search) (Continued)

If the C address controller bits are 10, then the comparison is made for a lowest CONDITION. When and if a B field is detected that is lower than the A field, the pertinent B field is then used in the comparison with the subsequent B fields in lieu of the A field and the process continues until the B fields are exhausted. The address of the lowest field detected is stored in IX1. If no low is found, the A ADDRESS - or the address of the A ADDRESS field (if A ADDRESS is a literal) is loaded into IX1.

In some cases, after comparison of the last possible full B field in the interval between the B and C addresses, a final B field remains which is shorter than specified. (Example: A field greater than NN). If a low or lowest condition is detected in this short field before the C address is reached, it is treated as a standard low and its address is stored in IX1. If no low is found, the C address is encountered and the Search terminates with IX1 unchanged.

The comparison flip-flops are set equal if a B field address is stored. The comparison flip-flops are set high if the A field address is stored.

6.26.2 OP 37 - SLL (Search Link List)

This instruction is invalid unless the Extended Function Adapter hardware is installed.

Compare the key in A field with B+BF in the manner prescribed by the variants in the B address controller. If the condition is met, store the address of the B field in IX1 and set comparison. If the comparison is not met, find the next list entry at the address specified by B, until list entry address is zero; at that time the comparison is set high. The address specified at B must be Mod 2.

"A" controller indicates data type for the -A- and -B- fields.

00 Four Bit Mode
01 Reserved
10 8-Bit Mode

6.26.2 OP 37 - SLL (Search Link List) (Continued)

11 Indirect Address

"B" controller indicates comparison:

00 Equal: If A field is equal to B field, set comparison equal.

01 Any bit equal: If any one-bit of the A field is equal to the corresponding bit of the B field, set comparison equal.

10 Less Than: If the A field is algebraically less than the B field, set comparison low; if the A field is equal to the B field, set comparison equal.

11 No-Bit Equal: Logical sums of corresponding bits of the A and B fields are compared. The logical sum is formed for each pair (all B field bits are examined). If the logical sums are zero (bit pairs 0-0, 0-1, or 1-0) for all pairs, set comparison equal.

Indexing and address extension are allowed on A and B, but are not permitted on links within the list.

Indirect addressing is allowed on "A" only.

AF is the length of key to be searched.
BF is the offset from B to the field to be searched.
A is the address of the key to be searched.
B is the address of the first list entry.

After proper alignment of operands, algebraically subtract a subtrahend located in A from a minuend located in B and store the result in C. Operands need not be in normalized form.

The result mantissa length in digits is the same as the length of the larger of the A and B field lengths.

If mantissa overflow occurs, the result exponent is previous link.