EXTENDED SPL
AN ASSEMBLY LANGUAGE FOR THE SCC 4700 COMPUTER

March 1970

by

Billy P. Buckles

PREPARED FOR:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
COMPUTATION LABORATORY
MAN-MACHINE SYSTEMS BRANCH

under contract NAS8-18405

Man-Machine Systems Section
Systems Development Operations
COMPUTER SCIENCES CORPORATION

Huntsville, Alabama

NASA CONTRACTOR
REPORT

EXTENDED SPL - AN ASSEMBLY LANGUAGE FOR THE SCC 4700 COMPUTER

Prepared under Contract NAS8-18405

by

Billy P. Buckles

COMPUTER SCIENCES CORPORATION
SYSTEMS DEVELOPMENT OPERATIONS
HUNTSVILLE, ALABAMA

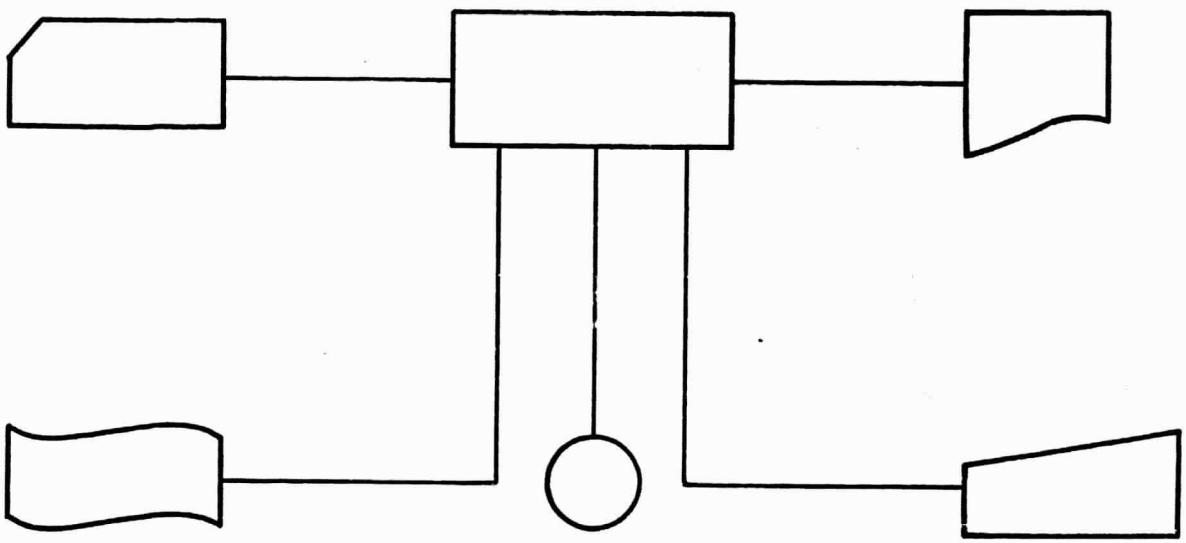
For

NASA - GEORGE C. MARSHALL SPACE FLIGHT CENTER
HUNTSVILLE, ALABAMA

May 21, 1970

EXTENDED SPL

AN ASSEMBLY LANGUAGE FOR THE SCC 4700 COMPUTER



EXTENDED SPL
AN ASSEMBLY LANGUAGE FOR THE SCC 4700 COMPUTER

by

Billy P. Buckles

ABSTRACT

Presented in this report are the syntax requirements and coding aids available for constructing source statements for ESPL. Examples and tables are used to explain the relationships between the symbolic source statements and the machine data that is generated.

EXTENDED SPL
AN ASSEMBLY LANGUAGE FOR THE SCC 4700 COMPUTER

Billy P. Buckles

SUMMARY

Extended SPL consists of instructions and directives. Instructions are assembled directly into machine executable operations while directives generate data or exercise control over the assembly process.

Convenient methods are provided to represent values inherent to the assembly process. These include decimal and hexadecimal notation, literal declaration, and a hierarchy of operators for computation in-line.

Several instructions and/or directives may be combined into a macro definition and retrieved subsequently by referencing the macro name. Specified elements of the macro definition may be changed with each call. Macros may be nested up to seven levels.

Error flags and operator messages are provided to indicate abnormal conditions detected either within the source statements or peripherals utilized by the system.

TABLE OF CONTENTS

INTRODUCTION

This manual describes Extended SPL 4700, an assembly language for the SCC 4700 computer. Its syntax design and mnemonic designations are based on SPL 4700, the assembly language provided by Scientific Control Corporation (SCC). This document is intended as a reference manual and assumes the user is familiar with digital computer programming in general and the SCC 4700 digital computer in particular.

Extended SPL 4700 was implemented under contract NAS8-18405 in support of Man-Machines Systems Branch, NASA, MSFC. It is designed to relieve the programmer from menial chores without relinquishing the tight machine control afforded by assembly language programming.

In this document, instructions are symbolic representations of machine commands which are translated by the assembler. Directives are messages which control the assembly process or create data. Comments appear only on the assembly listing and are ignored during the assembly process. Statements consist of these three classifications and are entered via punched cards at the DCT-132. Device selection is not incorporated due to the unique programming methods required for communications I/O.

Extended SPL 4700 is a three pass assembler. The address mode selection feature requires the additional pass. The source statements are read once. Programming features provided by Extended SPL 4700 include the following:

FREE FORM SOURCE FORMAT

A field may begin in any column but must be separated from the previous field by a blank. An exception is the label field which must begin in Column 1. Unless otherwise stated, blanks may not appear within a field.

MNEMONIC OPERATION CODES

Directives and machine instructions are written symbolically and are translated by Extended SPL 4700 into assembly and machine functions.

CONVENIENT DATA REPRESENTATION

Extended SPL 4700 allows data to be specified as decimal, hexadecimal, or alphanumeric. It provides both floating

and fixed point decimal constants. Binary point specification is permitted in fixed point numbers. In addition, a hierarchy of arithmetic/logical operators is provided for fixed point constant manipulation.

LITERAL REFERENCING

Certain instructions which reference a storage address may reference instead a literal. The literal table is assembled beginning in the first unused location following the user's program. Literals may be fixed point, floating point, or relocatable variables.

SELECTIVE OBJECT OUTPUT

Object code may be generated in absolute binary format or relocatable binary format. The option may be specified external or internal to the assembly process. In event of conflict the internal specification prevails.

SOURCE PROGRAM EDITING

A line edit capability is included within the assembler. Corrections to a program are assembled as entered if the original source code can be retrieved from a disk file.

CONCORDANCE LISTINGS

A concordance listing with an alphabetical listing of all labels referenced, the value, the type, and the statement numbers at which each was referenced is provided as a programmer option.

CONDITIONAL ASSEMBLY

Statements within a program may be assembled or skipped depending upon the value of assembly parameters.

ADDRESS MODE SELECTION

Certain memory reference instructions can be either one or two computer words long depending on their relative addresses. As a programmer option, the assembler will select the optimum address mode.

PROGRAM LISTINGS

A printout consisting of source statements input to the assembler and object code generated is provided as a programmer option.

ERROR CHECKING

Each source statement is evaluated thoroughly for incorrect usage of the language. Errors are indicated on the assembly listing.

MACRO CAPABILITIES

Statement sequences often used may be defined as macros and generated again in the program code stream by means of a single reference label. Macros may be nested up to seven levels.

EXECUTION TRACE CAPABILITY

Program trace statements may be inserted into the source deck and assembled or ignored on programmer option. The trace statement will create linkage to a system program that prints the trace identifier on the console device during program execution.

1. BASIC LANGUAGE FORMAT

1.1 General Description

In writing instructions for the Extended SPL 4700 assembler language, the programmer is primarily concerned with four fields: Label, Operation, Variable, and Comment. With the exception of the label field, a field may begin in any column but must be preceded by a blank.

1.1.1 Label Field

Labels consist of from one to six alphanumeric characters and must begin in column one. The first character of a label must be alphabetic. The character, $, is considered to be alphabetic but is generally reserved for system programs. The label field is optional unless otherwise stated and is used to identify a line of code.

CORRECT LABELS:

```
TABLE
SR24
I
$START
```

INCORRECT LABELS:

```
1XAR      (First character not alphabetic)
S1.T      (Non-alphanumeric character)
VARIATE   (More than six characters)
```

1.1.2 Operation Field

The operation field must follow the label, if a label is present, and consist of:

- An instruction mnemonic and possibly indirect addressing designation,
- An assembler directive,
- A mnemonic previously used in the label field of a MACRO directive (See MACROS), or,
- A mnemonic previously used in the label field of a OPD directive (See ASSEMBLER DIRECTIVES).

The operation field is terminated by one or more blanks.

1.1.3 Variable Field

The variable field begins at the first non-blank character following the operation field and is terminated by one or more blanks. Its format is controlled by the type of operation field employed and is omitted in some cases.

1.1.4 Comment Field

The comment field begins at the first non-blank character following the variable field or operation field if the variable field is omitted. It is used by the programmer for clarifying information and is ignored during the assembly process. The comment field may contain imbedded blanks.

A comment statement may be inserted into the code stream by by placing an asterisk (*) in column one.

1.2 Relative Addressing Instructions

A "Relative Addressing Instruction" may reference data directly that is in direct memory or is within relative addressing range of the instruction's location. Otherwise, it must reference the data indirectly through another computer word.

The format is:

LABEL	OPERATION	VARIABLE	COMMENT
	LDA	m, X	

Label and comment fields are optional. Indirect addressing, if desired, may be specified by placing an asterisk (*) immediately following the operation mnemonic. The variable field consists of two subfields. The address subfield designated by "m" is mandatory and consists of an expression. The index subfield is optional and must be entered as shown. The subfields are separated by a comma.

1.2.1 Automatic Addressing Mode Selection

The programmer may specify that the assembler select the optimum addressing mode. In this case the programmer codes relative addressing instructions directly referencing the data and the assembler will insert the indirect address word if needed.

For example, an instruction coded as

will be assembled as if coded

if the value "SAVE" does not meet the requirements for direct referencing.

1.2.2 Use of Literals

The "m" subfield may be replaced by a literal. This is done by entering an equal sign (=) and following it with an expression. (See Paragraph 2.3, Literals.) All literals used are assembled in a table beginning at the first location not used by the program.

1.3 Immediate Operand Instuctions

Instructions which reference data within the same computer word are "Immediate Operand Instructions".

LABEL	OPERATION	VARIABLE	COMMENT
	LDL	m	

Label and comment fields are optional. The variable field must consist of an absolute expression. If an ASCII character is desired, enter quotation marks (") in the variable field and follow with a single ASCII character.

1.4 Extended Operation Code Instructions

Instructions that do not require a variable or need an additional computer word to reference an operand are "Extended Operation Code Instructions". The format is

LABEL	OPERATION	COMMENT
	JRT	

Label and comment fields are optional. If indirect addressing is desired an asterisk (*) must immediately follow the operation mnemonic.

1.5 Shift Instructions

"Shift Instructions" are immediate operand instructions with an additional subfield, the index.

LABEL	OPERATION	VARIABLE	COMMENT
	SLL	m, X	

The "m" subfield is mandatory and must be an absolute expression. Label and comment fields are optional. The index subfield of the variable field is optional and must be entered as shown if desired. A comma separates the subfields.

1.6 Operate Instructions

The general register change and test instructions in the SCC 4700 are termed "Operate Instructions". A total of 512 operate instructions are available to the programmer. Table 2 lists the basic operation mnemonics and their function. The general form is:

LABEL	OPERATION	VARIABLE	COMMENT
	RCPY	s, d, t	

The label and comment field are optional along with the skip-test (t) subfield of the variable field.

The source (s) and destination (d) subfields of the variable field are mandatory. Permissable entries in each subfield are one of the characters A, B, E, or X, and depict one of the hardware registers.

The skip-test subfield depicts a test performed upon the contents of the destination register upon completion of the operate instruction function. A skip is performed if the test is true. Table 1 contains a list of permissable entries and their function.

SKIP-TEST SUBFIELD	FUNCTION
N	If the contents of the destination register is negative upon completion of the operation, a skip is performed.
Z	If the contents of the destination register is zero upon completion of the operation, a skip is performed.
G	If the contents of the destination register is positive and non-zero upon completion of the operation, a skip is performed.

TABLE 1

If the skip test subfield is not specified, no skip-test is performed.

The subfields are separated by commas.

1.7 Channel Commands

"Channel Commands" are not CPU instructions but data words processed by an I/O channel when the IOC instruction is processed by the CPU. A channel command is programmed following the IOC instruction. If the IOC instruction specifies indirect addressing, the word following must be a pointer to a channel command.

The correct form is:

LABEL	OPERATION	VARIABLE	COMMENT
	SIO	m, X	

Table 2 contains the operation field mnemonics, their functions and assembled values. The label and comment fields are optional as is the index subfield of the variable field.

The device number and the values of the block mode and interrupt arm bits, if non-zero, must be entered by the programmer using the mandatory "m" subfield in the form of an absolute expression. Optionally, the value or any portion thereof may be placed in the index register if the index subfield is specified. A comma must separate the subfields.

CHANNEL COMMANDS		
MNEMONIC	FUNCTION	ASSEMBLED VALUE
SIO	Start I/O	'0800
HIO	Halt I/O	'0000
XMT	Transmit	'1000
EOA	Execute Order In A	'1800
OUS	Output Unit Status	'2800
TWC	Terminate When Complete	'3800
SDR	Skip If Device Ready	'4800
SDA	Skip If Device Available	'5000
IIU	Input Interrupting Unit	'5800
IUS	Input Unit Status	'6000
ISB1	Input Status Byte 1	'6400
ISB2	Input Status Byte 2	'6800
ISB3	Input Status Byte 3	'6C00

Table 2

2. EXPRESSIONS

2.1 Definition of Expression

An expression is an elementary term or a series of elementary terms connected by operators. Blanks are not permitted within an expression.

2.1.1 Elementary Terms

Elementary terms consist of labels, decimal and hexadecimal integer constants, and floating point constants. An elementary term does not contain an operator.

2.1.1.1 Labels

Any label may be used as an elementary term. When the assembler encounters a label, it substitutes the value of the label's address or a value derived from the label's use with a directive. If an address value is used, the mode (i.e. byte or word) is based on the operation field of the statement in which the expression appears.

2.1.1.2 Decimal Integer Constants

A decimal integer constant may appear in an expression as a string of one or more integers of the set 0-9. A sign may precede the constant if the term occurs at the beginning of the expression or the term is separated by parentheses. In any other case the sign is interpreted as an add or subtract operator.

EXAMPLES:

```
 15
-27
(-27)
 5
```

A single precision integer, I, must be in the range -32, 768 I 32,767. Exceeding the range will cause an error indicator to be set. Double precision decimal integers must be in the range -214, 783, 648 I 214, 783, 647 . Double precision is assembled only if the DED Directive appears in the operation field of the statement.

2.1.1.3 Fixed-Point Decimal Integers

A fixed-point decimal integer consists of two parts:

(a) A decimal integer that may be signed or unsigned and consists of the character set 0-9 and "." (decimal point),

(b) A binary place part that consists of the letter B followed by an unsigned decimal integer.

The binary place part must follow the decimal integer with no intervening blanks. The integer following the letter B denotes the position of the binary point in the assembled number. If a decimal integer, i, follows then 0 i 15 for single precision integers and 0 i 31 for double precision integers. Assume that the binary place part is B5, then the binary point will be between bits 5 and 6 of the assembled integer. Fixed-point decimal integers are used only with DEC and DED directives.

EXAMPLES:

```
10B5
1.7B4
-126.5B12
-276B9
```

2.1.1.4 Hexadecimal Integers

Hexadecimal integers consist of the character set 0-9, A-F immediately preceded by an apostrophe ('). A plus or minus sign may precede the apostrophe provided it is separated from any previous operator. No overflow checking is performed when this form is used making it ideal for "Bit-Picking" functions.

EXAMPLES:

```
'9D0
-'F000
'11C
-'90
```

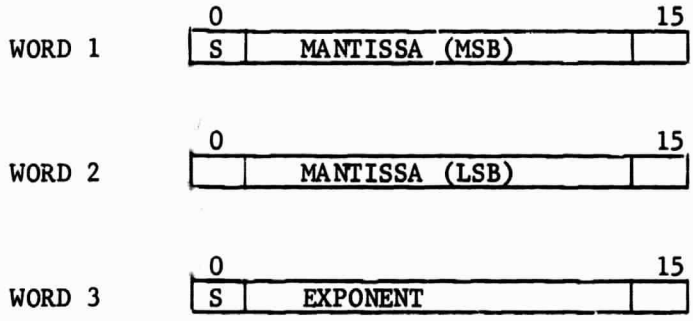
2.1.1.5 Floating Point Decimal Numbers

Floating point decimal numbers consist of two parts:

(a) A decimal part consisting of a string of characters from the set 0-9, "." (decimal point), and

(b) An optional exponent part, consisting of the letter E followed by a signed or unsigned integer, that represents a power of ten by which the number is to be multiplied.

Floating point numbers are assembled into three consecutive computer words. The first two are the mantissa with an associated sign. The third is the exponent with an associated sign.



Floating point terms are assembled only if they appear in a literal (see Paragraph 2.3, Literals), or appear in a statement having a DEC or DED directive in the operation field. A floating point decimal term may not be combined with other terms within an expression.

EXAMPLES:

```
5.0
117E-4
-6.137E16
76.174E+4
-17.4
+9700E-12
```

2.1.2 Operators

A hierarchy of seven operators are provided on three levels. Level 1 operators are assembled first and Level 3 operators are assembled last. Table 3 contains the use and combinational characteristics of the operators.

2.2 Use of Parenthetical Expressions

Any expression containing more than one elementary term may be divided into sub-expressions by means of parentheses. Parenthetical expressions may be nexted to any depth. The inner-most sub-expression is evaluated first without regard to the level of operators in the outer sub-expressions.

HIERARCHY OF OPERATORS

LEVEL	OPERATOR	DESCRIPTION
1	--	LOGICAL DIFFERENCE (EXCLUSIVE OR)
	++	LOGICAL SUM (INCLUSIVE OR)
	**	LOGICAL PRODUCT (AND)
2	*	ARITHMETIC PRODUCT
	/	ARITHMETIC QUOTIENT
	//	COVERED QUOTIENT (a//b IS EQUIVALENT TO $\frac{a+b-1}{b}$)
3	+	ARITHMETIC SUM
	-	ARITHMETIC DIFFERENCE

Table 3

2.3 Literals

Any of the basic relative addressing instructions may reference a literal in the "m" subfield of the variable field. A literal consists of an expression immediately preceded by an equal sign (=). Literal expressions may not contain a label, used as an elementary term, which corresponds to an address value greater than the current address value of the program being assembled.

A referenced literal will be assembled into a literal table starting at the first location not used by the program being processed. A value will appear in the literal table only once regardless of the number of times it occurs as a literal expression.

2.4 Construction of Expressions

Syntax errors occurring in expressions are indicated on the program listing. If a syntax error is encountered during evaluation, the generated value of the expression may not be reliable.

2.4.1 Rules Governing Expression Syntax

The following rules used in conjunction with Appendix D form a comprehensive guide for construction of expressions.

- A literal must be preceded by an equal sign (=).
- A plus (+) or minus (-) preceding an expression is allowed.
- Floating point expressions may contain only the single elementary term.
- Expressions must be single precision unless a directive in the operation field specifies otherwise.
- Parenthetical expressions may be nested to any level with the inner-most subexpression evaluated first.
- An operator may not consecutively follow another operator and an elementary term may not consecutively follow another elementary term with the following exceptions:

(a) a minus may follow the covered quotient operator; and

(b) a minus or plus may precede an elementary term if an open parenthesis separates it from the preceding operator.

In both cases the plus or minus is interpreted as a sign indicator and not as an operator.

- Labels representing address values may be used subject to the following conditions:

  (a) Functions may not be performed on address values that cause them to lose their addressing properties; and

  (b) Expressions which render an address value are converted to the mode (word or byte) dictated by the operation field.

- Imbedded blanks are illegal.

Expression syntax errors are indicated on the program listing.

2.4.2 Examples of Expressions

LEGAL:

```
4*7+'D++'C
COST+7
'18*(-4*(16+2//7))
```

If TABLE is an address value in byte mode then

```
TABLE//-1
```

Converts to an address value in word mode. No value other than -1 may be substituted without causing the expression to lose its basic addressing properties.

If TABLE is an address value in word mode then

TABLE//1

Converts to byte mode. Obviously, no value other than 1 may be substituted for the second elementary term.

ILLEGAL:

```
4*7(4--5)
    consecutive elementary terms
14+(7*(16-'A)
    missing parenthesis
6E4+1.7
    combining floating point elementary terms
(108+(7**(PRICE+6)))
    if PRICE is an address value its adressing
    properties are lost
15//17++'1D
    double precision
```

3. MACROS

3.1 The MACRO Directive

The MACRO directive denotes the beginning of a MACRO skeleton, a body of code that can be reproduced in the code stream by means of the MACRO call statement.

The general form is:

LABEL	OPERATION	COMMENT
ANY	MACRO	

The comment field is optional.

3.2 The ENDM Directive

The ENDM directive denotes the end of a MACRO skeleton.

The general form is:

LABEL	OPERATION	COMMENT
	ENDM	

Label and comment fields are optional. If a label is used it will assume the address value of the next computer word assembled.

3.3 The MACRO Skeleton

Statements occurring between the MACRO directive and the ENDM directive are called skeletal statements.

3.3.1 Skeletal Statements

The general form of a skeletal statement is:

LABEL	OPERATION	VARIABLE	COMMENT
Valid Label or dummy argument	Valid Operation or dummy argument	An expression; may contain dummy argument(s)	

The contents and format of the variable field are dictated by the type of operation field. If indirect addressing is required and the operation field is a dummy argument, it may be followed by an asterisk (*). Indexing may be specified in the variable field either through the argument list or by coding the index subfield into the skeletal statement.

3.3.2 Dummy Arguments

Dummy arguments appear in the MACRO skeleton and are substituted with the desired character stream from the argument list when the MACRO is called. Dummy arguments are of the form "#i" where "i" is a decimal integer of the set 0 through 19.

The dummy argument, #0, may appear only in the label field and will cause a label appearing in the MACRO call statement to be substituted for it. If a label appears in a MACRO call statement and no #0 dummy argument appears in the MACRO skeleton the label will automatically be attached to the first statement in the code stream that results from expansion of the MACRO skeleton.

The dummy arguments #1 through #19 are replaced by the corresponding character streams appearing in the argument list of the MACRO call statement.

3.4 The MACRO Call Statement

The general form of the MACRO call statement is:

LABEL	OPERATION	VARIABLE	COMMENT
	A MACRO NAME	ARG1;ARG2;...	

Label and comment fields are optional. The MACRO name appearing in the operation field must be previously defined in the label field of a MACRO directive. The variable field contains the argument list if dummy arguments are used. An argument may not contain an imbedded semi-colon (;), and the variable field may not contain imbedded blanks. The number of arguments must be equal to the greatest integer used as a dummy argument in the MACRO skeleton. Arguments are separated by semi-colons.

3.5 MACRO Nesting

MACROs may be nested to a level of seven deep by placing a MACRO call of a previously defined MACRO in the MACRO skeleton. Nesting may be accomplished to the same level by using MACRO names in the argument list of MACRO call statements, causing the skeletal statement to generate another MACRO call.

3.6 MACRO Examples

The example below shows how a MACRO is designated and assembled when called.

The MACRO skeleton:

```
3                       SAVE    MACRO
4                       #0      STA     #1
5                               STB     #1+1
6                               STX
7                               PAR     #1+2
8                               ENDM
```

The MACRO call and expansion:

```
87                              SAVE    SY1X23
     0042  7431
     0043  3431
     0044  0638
     0045  0075
```

The example below shows one method by which MACROs may be nested.

```
143                     CAT     MACRO
144                     #0      LDA     #1
145                             #2      #3
146                             #4*     #1,X
147                             JMP     D
148                             ENDM
149                     RAT     MACRO
150                             CAT     FOLLIE;ADD*;#1;STA
151                             STA     #1
152                             ENDM
153                             RAT     B,X

     019E  C5ED
     019F  9FD1
     01A0  7FEB
     01A1  45D5
     01A2  77CE
```

4. ASSEMBLER DIRECTIVES

4.1 Functions of Directives

The symbolic assembler directives within extended SPL control or direct the assembly processor just as operation codes control or direct the central processor unit. These directives are represented by mnemonics entered in the operation field of a statement. They are used to equate expressions, to adjust the location counter, and to afford the programmer special control over the generation of data.

4.2 Directive Description

The general format is:

LABEL	OPERATION	VARIABLE	COMMENT
	Directive		

All directives do not necessarily contain all four fields. A detailed description of the 23 directives in Extended SPL follow.

4.2.1 BBSS, Byte Block Starting Storage

The BBSS directive is used to reserve a block of core for data storage or working area. The variable field must contain an absolute expression, the value of which determines the number of bytes reserved. If a label is present it is assigned the byte address value of the first byte in the block. The BBSS directive in no way affects the contents of the core reserved.

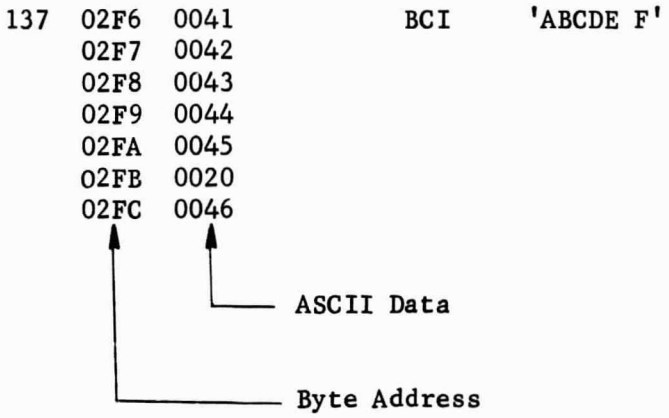
```
136   ┌→031A    FOLLIE BBS    3
      │
      └──┤starting byte address
```

4.2.2 BCI, Binary Coded Information

The BCI directive is used to insert ASCII data into the object code. The variable field is of the form:

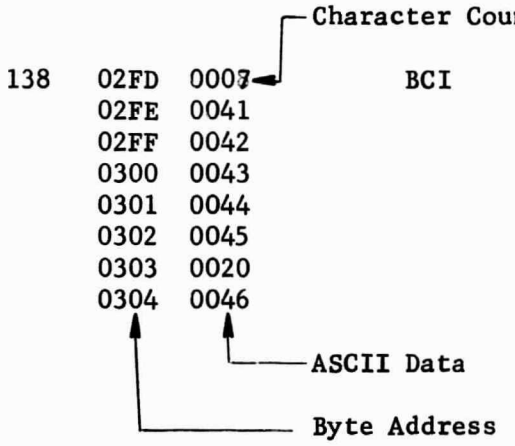
dstringd

where "d" is the alphanumeric string delimeter and "string" is any set of ASCII characters that do not contain "d". As each character is decoded it is inserted into the next available byte. If a label is present it is assigned the byte address value of the first ASCII character. See Appendix C for a partial list of ASCII codes as used by the BCI directive.



4.2.3 BCIC, Binary Coded Information and Count

The BCIC directive, like the BCI directive is used to insert ASCII data into the object code. It is decoded in the same manner as the BCI directive except the first byte generated is the count of ASCII characters occurring between the two string delimeters in the variable field.



4.2.4 BEQU, Byte Equivalence

The BEQU directive equates the symbol in the mandatory label field with the value of the expression in the variable field. Generally the expression field contains an address value. The programmer should be aware that if the symbol in the label field is used in subsequent expressions and is an address value, it will assume the mode dictated by the operation field of the statement in which the expression occurs. The

variable field may not contain a floating point number nor a symbol representing an address value greater than the current location counter. The BEQU directive does not affect the value of the location counter.

```
133              0813   F        BEQU      E
                  |
                  |___ Byte address of "E"
```

4.2.5 BPAR, Byte Parameter

The BPAR directive is used to insert byte data into the object code. The variable field contains one or more absolute expressions separated by semi-colons. As each expression is evaluated it is inserted into the next available byte of the program being processed. If a label is used it is assigned the byte address of the first byte generated.

```
122  02CE  000A          BPAR   'A;'B;'C;'D;'E;'F;0;-3
     02CF  000B
     02D0  000C
     02D1  000D
     02D2  000E
     02D3  000F
     02D4  0000
     02D5  00FD
      ^     ^
      |     |___ Generated Byte Data
      |
      |_____ Byte Address
```

4.2.6 BPTR, Byte Pointer

The BPTR directive is used to insert a data word, generally a byte address, into the object code. The variable field contains one or more expressions separated by semi-colons. The programmer should be aware that symbols representing address values that appear in the variable field are converted by the assembler to byte mode during the expression evaluation process. If a label is used, it is assigned the address value of the first word generated.

```
121  02C8  01C8          BPTR     ORG+'100;*;-3
     02CA  02CA
     02CC  FFFD
      ^     ^
      |     |___ Generated Data
      |
      |_____ Byte Address
```

4.2.7 BSS, Block Starting Storage

The BSS directive is used to reserve a block of core for data storage or working area. The variable field must contain an absolute expression, the value of which determines the number of words reserved. If a label is present, it is assigned the address value of the first word in the block. The BSS directive in no way affects the contents of the core reserved.

```
232  0131  0132            PAR    SYRA20
233  0132        SYRA20    BSS    5
234  0137  0408  SYRB20    RCPY   X,A

     Address of First Word
```

4.2.8 CALL, Call

The CALL directives generates a calling sequence of two computer words in the object code and necessary linkage information for the relocatable loader. The variable field must contain a symbol, indexed if desired, that is used in conjunction with a name directive in anotherprogram. A label, if present, assumes the address value of the first word generated. The CALL directive may be used only in relocatable programs.

```
143  0085  AC01            CALL   $SY23
143  0086  0000
144  0087  00CD            PAR    SY1B24+1

           Address value of $SY23
           Assigned by relocatable loader
```

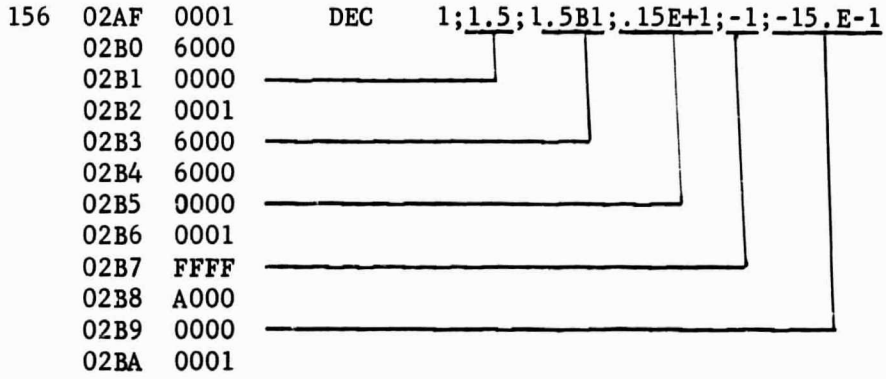
4.2.9 DEC, Decimal

The DEC directive is used to insert binary data into the object code the variable field must contain one or more signed or unsigned decimal numbers separated by semi-colons. A variable field entry may be of the form:

DECIMAL INTEGER
FIXED POINT DECIMAL INTEGER
FLOATING POINT

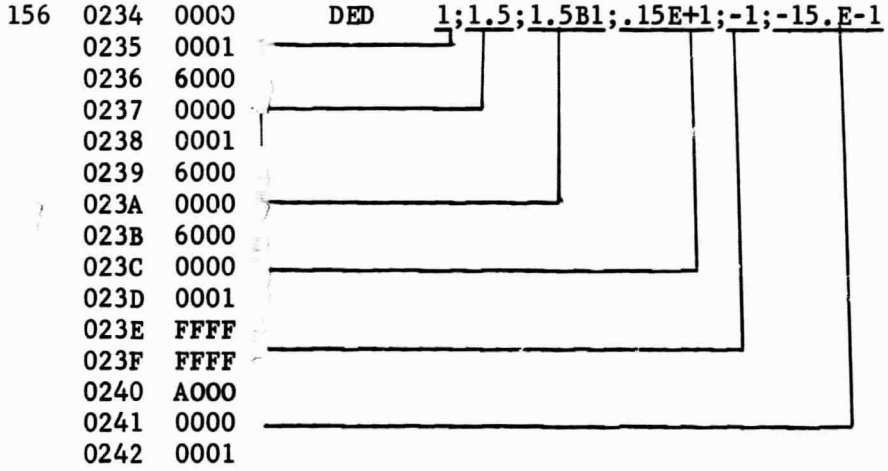
Floating point entries are assembled according to the form in Paragraph 2.1.1.5. Decimal integers and fixed point decimal integers must be single precision and are assembled into a single computer word. An error indication is given in case of overflow.

If a label is used it is assigned the address value of the first word generated.



4.2.10 DED, Double Decimal

Like the DEC directive, the DED directive is used to insert binary data into the object code. The variable field is subject to the same form except decimal integers and fixed point decimal integers are assembled into two computer words and must exceed 32 bits before an overflow indication is given. If a label is used it is assigned the address value of the first word generated.



4.2.11 END, End of Assembly

The END directive causes a halt of the assembly process and must be the last physical statement in the program. If an expression is entered into the variable field, its value designates the starting address at which execution is begun. If several programs are being assembled and are to be linked before execution, only one may contain a starting address.

```
156                                        END

6  Errors
0  Bytes Unassigned
```

4.2.12 ENDF, End of Conditional Assembly

The ENDF directive designates the end of a portion of code that may be assembled or omitted depending on certain internal values. It is used in conjunction with the IFT directive and requires no variable field. For further details see Paragraph 4.2.16, the IFT directive.

4.2.13 ENDM, End of MACRO Skeleton

See Section 3, MACROS.

4.2.14 EQU, Equate

The EQU directive is used to equate the symbol appearing in the mandatory label field with the value of the expression in the variable field. The variable field may not contain a floating point value nor a symbol representing an address value greater than the current location counter. The EQU directive does not affect the contents of the location counter.

```
412          0005  A        EQU      5
                   |
                   +-------- Value Equated with "A".
```

4.2.15 EVEN, Even Location Counter

The EVEN directive is used to assure the next value assigned to the location counter by the assembler is even. If the value of the location counter is already even it remains unchanged. If the location counter is odd it is incremented by one. A label, if used, is assigned an address value equal to the location counter prior to the possible incrementing.

```
131          018A             EVEN
             |
             +-------- Next Address Assigned
```

4.2.16 IFT, Conditional Assembly

The IFT directive and the ENDF directive are used to bracket a code sequence that is to be assembled or ignored depending on certain conditions. The expresion in the variable field of the IFT directive is evaluated. If the value is false (zero) the statements that occur up to the next ENDF directive are ignored by the assembler. If the value is true (non-zero) the

statements up to the next ENDF directive are assembled in the normal manner. A label attached to the IFT directive is assigned an address value equal to the current value of the location counter.

```
123                              IFT     0
124                              LLL     1
125                              ENDF
126                              IFT     1
127             0180    02E2     LLL     2
128                              ENDF
```

4.2.17 MACRO, MACRO Definition

See Section 3, MACROS.

4.2.18 NAME, External Name

The NAME directive specifies symbols that may be referenced by external programs. The variable field contains one or more symbols, separated by semi-colons, that occur in the label fields of other statements in the same program. The NAME directive used in conjunction with CALL directives and XPTR directives in other programs will generate object code to enable the relocatable loader to create linkage. It may be used only in programs specified as relocatable. The location counter is not affected.

```
118                              NAME    ORG
```

4.2.19 OPD, Operation Definition

The OPD directive is used to define mnemonics that may be subsequently used in the operation field. The mandatory label field contains a symbol that is equated to the absolute expression in the variable field. The symbol may then be used as an operation conforming to the statement form of the extended operation code instruction as described in Paragraph 1.4. The location counter is not affected by the OPD directive.

```
137             0200    FAD     OPD     '0200
138                             RDEF
139         018E 0200           FAD
```

4.2.20 ORG, Origin

The ORG directive is used to change the contents of the location counter to the value of the expression in the variable field. If a symbol representing an address value appears in the variable field expression, it must have been previously defined. If the expression is absolute then the mode of the program being assembled becomes absolute, regardless of the external specification. Initially, all programs are assumed to begin at zero and the mode relocatable if not otherwise specified externally.

```
119   0064        ORG     ORG     *
120   0164                ORG     ORG+'100
```

4.2.21 PAR, Paragraph

The PAR directive is used to insert single precision data words into the object code. The variable field contains one or more single precision expressions, either absolute or relocatable, separated by semi-colons. If a label is present it is assigned the address value of the first word generated.

```
124   016B   640A  B     LDL      'A
125   016C   0020  C     PAR      '20;-6;A;B;D
      016D   FFFA
      016E   0005
      016F   016B
      0170   0171
126   0171   FFFF  D     PAR       A-6;*+2;C
      0172   0174
      0173   016C
```

Generated Data

Address

4.2.22 RDEF, Redefine Table Scan

Mnemonics appearing in the operation field may be assembler defined operation codes and are entered in the operation table; or they may be user defined, via MACRO and OPD directives, and are entered in the user symbol table. If a user defined mnemonic is identical to an assembler mnemonic the assembler normally chooses the one from the operation table. The RDEF switches priority to the user symbol table. Succeeding RDEF directives inverts the priority sequence from that being used at the time.

For further details, see Paragraph 4.2.19, The OPD Directive.

4.2.23 XPTR, External Pointer

The XPTR directive used in conjunction with the NAME directive in another program generates object code to enable the relocatable loader to create linkage. The variable field contains one or more subfields separated by semi-colons. Each subfield contains a symbol which occurs in a NAME directive of another program. A subfield may be indexed by following it with ",X". The assembler reserves one word of storage for each subfield. At load time the address of the externally defined symbol will be inserted into the reserved word by the relocatable

loader. A symbol in the label field is assigned the address value of the first word generated. The XPTR directive may only be used in relocatable programs.

```
130  0178  8000      XPTR    SIN,X;COS
     0179  0000
```

4.3 The Program Trace Statement

A "Program Trace Statement" will be assembled only if the trace option is present among the external parameters. (See Section 5, Operating Procedures.) If the trace option is not specified the program trace statement is ignored during the assembly process.

The general form is:

LABEL	OPERATION	COMMENT
.	Any Name	

The label field must contain a period(.) in column 1. The operation field commences with the first non-blank character after column 1. The operation field contains a mnemonic conforming to the specifications in Paragraph 1.1.1. The assembler will generate linkage to a system program that, at execution time, prints the contents of the operation field on the console device each time the trace code sequence is encountered. Use of the trace option is not limited to relocatable programs.

```
116  056A  0756   .   POINT1
     056B  504F
     056C  494E
     056D  5431
```

5. OPERATING PROCEDURES

5.1 Executing Processor Call

Due to incomplete configuration, details of loading and executing Extended SPL have not been concluded. This section of the document will be replaced when final configuration is reached and again when integration of extended SPL under an operating system occurs. In the meantime, a usable copy with loading procedures will be maintained for programmer use.

5.2 Assembly Options

Upon beginning processor execution a prompting message is typed on the IBM Selectric:

"ESPL"

The operator then types at the selectric a string of upper case alphabetic characters, representing external control parameters, followed by a carriage return. These parameters include:

A - Absolute Base Mode
R - Relocatable Base Mode
L - List Symbolic File
O - Generate Object File
N - Do Not Generate Object File
X - Produce Concordance Listing
U - Update Disk Symbolic File
I - Do Not Map Relative Addressed Instructions
P - Produce Object File On Paper Tape
T - Assemble Trace Statements

If a lower case or non-alphabetic character is entered the prompting message is displayed again and the operator may begin again. A slash (/) typed on-line will nullify all preceding parameters and the operator may follow with a new set.

If neither "A" nor "R" is entered, relocatable base mode is assumed. If neither "O" nor "N" is entered, an object file is created.

5.3 Assembler I/O

The operator may specify binary object output on the paper tape punch by entering the "P" option during initialization. If the "P" option is omitted and an object program is desired, the disk is assumed to be the output device.

Due to the unique characteristics of communications line I/O, source input device selection and symbolic output device selection

is not provided. The source input device is assumed to be a remote station card reader and the symbolic output device is assumed to be a line printer at the same remote station.

5.3.1 Source Input

Upon completion of parameter selection a prompting message is typed at the IBM Selectric:

"INITIALIZE COMM LINE".

After the operator ascertains the line is ready, the cards may be placed in the card reader hopper and the card reader started. No further intervention is required.

5.3.2 Source File Updating

The procedures for creating and deleting source files on the disk as well as identifying them for processor manipulation has not been defined and cannot be included within this document. The basic procedure for updating them will remain stable and the purpose of this paragraph is to describe the control cards necessary.

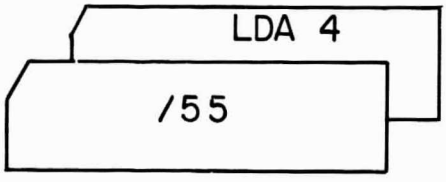
If updating is desired the "U" option must be specified in the input parameter string.

5.3.2.1 The "Addition" Card

If additional source statements are to be added they may be punched on cards in the normal manner and preceded in the source deck by an "addition" card in the form

/N

where slash (/) appears in column 1 and N is a decimal integer specifying the sequence number of the disk source file after which the following source statement(s) are to be added. N follows the slash with no intervening or imbedded blanks. For example:



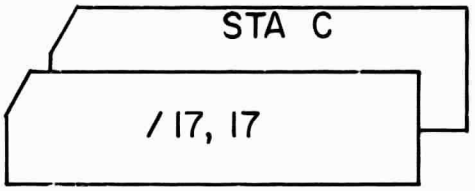
will cause a LDA instruction to be inserted after image number 55 in the source file.

5.3.2.2 The "Deletion" Card

The "Deletion" card will delete source statements from the disk source file. It may or may not be followed by source statement(s) to replace the deleted statement(s). The form is:

/N1,N2

where N1 and N2 are decimal integers representing sequence numbers to be inclusively deleted. N2 must be greater than or equal to N1 and follow the slash (/) with no imbedded or intervening blanks. They are separated by a comma. For example:



will delete image number 17 from the source file and replace it with a STA instruction.

5.3.2.3 The Update Deck

Deletion and addition cards may be combined in any manner with source statement cards. The only restriction is that the sequence number(s) on an update control card must be greater than the sequence number(s) on the preceding control card and less than the one(s) on the following control card.

6. LISTINGS

6.1 The Concordance Listing

If the "X" option is specified among the external parameters a concordance listing will be produced at the symbolic output device preceding the program listing. The concordance listing contains the following information:

- An alphabetic ordered list of all symbols appearing in the label field and not attached to OPD or MACRO directives.
- The value of the symbol, either an address value or an equated value.
- The type of value, either absolute (A) or relocatable (R).
- A numerically ordered list of statement numbers that reference each symbol.

6.2 The Program Listing

The program listing will be generated if the "L" option is included among the external parameters. The symbolic and assembled data included are:

COLUMNS	
2-6	Statement Sequence Number
9-16	Error Flags (See Appendix A)
19-22	Address Value of Assembled Image
25-28	Binary Value of Assembled Image
31-110	Symbolic Image

The listing is followed by messages to aid the user in error analysis.

XXXX ERRORS

contains the number of assembly errors encountered in the source statements.

XXXX BYTES UNASSIGNED

contains the number of bytes which occurred internal to the assembled program to which no value was assigned due to EVEN directives and combined use of byte generating directives with word generating statements.

NAME	VALUE:TYPE	REFERENCES									
DUMMY	0117:R	198	199	200	201	202	203	204	205	206	210
START	0000:R	2									
SYA20	003F:R	33									
SYA23	0058:R	128									
SYA24	0083:R	164									
SYB20	0019:R	62									
SYB23	0071:R	108	110								
SYB24	00A3:R	150									
SYC20	002D:R	58									
SYC23	0069:R	114									
SYC24	0091:R	157	163								
SYD23	0061:R	130									
SYD24	0096:R	155									
SYF24	00B1:R	158									
SYG24	009B:R	161									
SYH24	009D:R	170	173								
SYRA20	0132:R	212	229	232							
SYRB20	0137:R	218									
SYRC20	012F:R	223	239								
SYR20	011D:R	70									
SY1L23	0076:R	97	113								
SY1B23	0078:R	101									
SY1L24	00CB:R	152									
SY1B24	00CC:R	141	144								
SY1C23	007A:R	109	124								
SY1N23	0077:R	99	129								
SY1N24	0103:R	147	149								
SY1C24	0100:R	145	153	159	162						
SY1E24	00A8:R	169									

CONCORDANCE LISTING

UNREF

will be printed and followed by a list of mnemonics that appeared in the label field of a statement but was not referenced in the variable field of another instruction.

MUL DEF

will be printed and followed by a list of mnemonics that appeared in the label field of more than one statement.

APPENDIX A

ERROR CODES

A - Not allowed in absolute assembly. A CALL, XPTR, or NAME directive appears in an absolute program.

B - A trace statement specified an invalid identifier.

C - Constant overflow. An expression was specified that caused the generated machine word to overflow.

D - A MACRO skeletal statement contained a "#" not followed by a decimal digit.

E - Expression. The expression in the variable field contains one or more syntax errors.

G - Invalid origin. The variable field of an ORG directive contains one or more syntax errors.

I - Index incorrect. The index subfield of a memory reference instruction contains something other than "X".

J - Image overflow. The combination of arguments specified during a MACRO expansion created an image greater than 80 characters.

K - Label. The label field is blank and the operation field contains an EQU, BEQU, OPD or MACRO directive.

L - A presumed symbol begins with a character other than $ or alphabetic character.

M - Multiple defined symbol. The symbol in the label field appears also in the label field of other statement(s).

N - Name error. A symbol contains more than six characters.

O - Operation field. The operation field of a statement contains an illegal mnemonic.

P - No END directive. An END directive was not encountered in the input file. One was supplied but the file is marked in error.

Q - Illegal equivalence. A symbol appeared in the variable field of an equivalence statement which had not previously been defined.

R - Range error. Automatic address mode selection was not specified among the external parameters and a relative addressing instruction referenced an address not in direct memory and more than 256 locations away.

S - MACRO calls are nested to a level greater than seven.

T - Terminate statement error. A statement was terminated with a character other than a space. Possibly an illegal subfield was added.

U - Undefined. A statement contained a symbol in the variable field which did not occur in the label field of any statement.

V - Variable field. The variable field was omitted from a statement requiring it.

W - Forward reference. An equivalence statement or literal expression contained a symbol that had not been defined previously.

X - A MACRO skeleton contained a MACRO directive.

Y - Undefined MACRO argument. A MACRO skeletal statement references an argument that was not specified in the MACRO call statement.

Z - Operate instruction specified incorrectly. Either the source of destination subfields contain a character other than A, B, X, or E, or the test-skip sub-fields, if designated, contains a character other than G, N, or Z.

APPENDIX B

ERROR MESSAGES

If the assembler aborts processing due to internally detected conditions, one of the following messages will be printed at both the on-line console device and the symbolic output device.

$E 1

The update option was specified among the external parameters and the disk source file specified does not exist.

$E 2

An I/O failure occurred on the input device.

$E 3

An illegal sequence identifier was entered on an update control card. To continue the assembly might result in destroying the original source file.

$E 4

Core storage was exceeded due to a macro definition.

$E 5

Core storage was exceeded due to length of user's symbol/literal table.

APPENDIX C

ASCII TABLE

The table presented in this appendix represents only a partial listing of the ASCII codes generated by BCI and BCIC directives.

0-30	I-49	! - 21
1-31	J-4A	# - 23
2-32	K-4B	$ - 24
3-33	L-4C	% - 25
4-34	M-4D	& - 26
5-35	N-4E	(- 28
6-36	O-4F	) - 29
7-37	P-50	* - 2A
8-38	Q-51	+ - 2B
9-39	R-52	/ - 2F
A-41	S-53	< - 3C
B-42	T-54	= - 3D
C-43	U-55	> - 3E
D-44	V-56	? - 3F
E-45	W-57	@ - 40
F-46	X-58	\ - 5C
G-47	Y-59	
H-48	Z-5A	

(Blank)	- 21
" (Quotation)	- 22
' (Apostrophe)	- 27
, (Comma)	- 2C
- (Minus)	- 2D
. (Period)	- 2E
: (Colon)	- 3A
; (Semi-Colon)	- 3B

APPENDIX D

RULES FOR RELOCATION OF BINARY TERMS

LEVEL	1ST TERM	OP	2ND TERM	RESULT	DIRECTION	NOTES
1	ABSOLUTE	--	ABSOLUTE	ABSOLUTE	R TO L	1
	ABSOLUTE	++	ABSOLUTE	ABSOLUTE	R TO L	1
	ABSOLUTE	**	ABSOLUTE	ABSOLUTE	R TO L	1
2	ABSOLUTE	*	ABSOLUTE	ABSOLUTE	L TO R	1
	ABSOLUTE	/	ABSOLUTE	ABSOLUTE	L TO R	1
	ABSOLUTE	//	ABSOLUTE	ABSOLUTE	L TO R	1
	RELOCATABLE	//	1	RELOCATABLE	L TO R	1,2
	RELOCATABLE	//	-1	RELOCATABLE	L TO R	1,3
3	ABSOLUTE	+	ABSOLUTE	ABSOLUTE	L TO R	1
	RELOCATABLE	+	ABSOLUTE	RELOCATABLE	L TO R	1,4
	ABSOLUTE	-	ABSOLUTE	ABSOLUTE	L TO R	1
	RELOCATABLE	-	ABSOLUTE	RELOCATABLE	L TO R	1,4
	RELOCATABLE	-	RELOCATABLE	ABSOLUTE	L TO R	1

1. In "Direction" Column L is "Left" and R is "Right".

2. The 1st term must be in word mode and the result is in byte mode.

3. The 1st term must be in byte mode and the result is in word mode.

4. The 1st and 2nd terms may be transposed with no effect upon the result.

APPENDIX E

OPERATE INSTRUCTIONS

INSTRUCTION	FUNCTION
RCPY	The source register is copied into the destination register.
RINC	The source register is incremented by one. The result replaces the contents of the destination register.
RADD	The contents of the source register is added to the contents of the destination register. The result replaces the contents of the destination register.
RXOR	An exclusive OR is performed between the source and destination registers. The result replaces the contents of the destination registers.
RCMP	The one's complement of the contents of the source register is placed in the destination register.
RNEG	The two's complement of the contents of the source register is placed in the destination register.
RDEC	The contents of the source register are decremented by one. The result places the contents of the destination register.
RSUB	The contents of the destination register are subtracted from the contents of the source register. The result replaces the contents of the destination register.

21 May 1970

APPROVAL

EXTENDED SPL
AN ASSEMBLY LANGUAGE FOR THE
SCC 4700 COMPUTER

The information in this document has been reviewed for content and adherence to MSFC regulations. This document in its entirety has been determined to be unclassified.

R. N. Seitz

Dr. R. N. Seitz
Chief, Man-Machine Systems Branch
NASA-George C. Marshall Space Flight Center