

IBM Systems Reference Library

IBM 1620 SPS III Programming System Reference Manual

This publication describes the specifications and operating procedures for both 1620 SPS III and 1620-1443 SPS III Programming Systems.

This publication supersedes the following two publications:

IBM 1620 SPS III (Form C26-5749-1)

IBM 1620 SPS III for IBM 1443 Printer
(Form C26-5736-0)

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Comments concerning the contents of this publication may be addressed to:
IBM, Product Publications Department, San Jose, Calif. 95114

© 1963, 1964 by International Business Machines Corporation

Contents

	<i>Page</i>		<i>Page</i>
IBM 1620 SPS III		1620 SPS III Processor Program	52
Introduction	1	Storage Layout	52
Symbolic Programming	1	Paper Tape Processor Program	54
Coding Sheet	1	Card Processor	54
Statement Writing	4	Condensed Object Deck Alterations	59
Programming the 1620 Using SPS	10	Operating Procedures	60
Declarative Statements	10	Switches	60
Imperative Statements	14	Loading the Processor	60
Processor Control Statements	22	Processing the Source Program	61
1620 Subroutines	29	Editing the Source Program	62
Subroutine Macro-Instructions	30	Error Messages	62
Floating Point Arithmetic	32	1620-1443 SPS III	
Description of 1620 Subroutines	34	Operating Procedures	64
Subroutine Error Messages	43	Index	66
Adding Subroutines	44		
Adding a Subroutine to a Card Deck	44		
Adding a Subroutine to Tape	46		
Writing a Subroutine	48		

Preface

The Symbolic Programming System permits the programmer to code in a symbolic language that is more meaningful and easy to handle than numerical machine language. SPS automatically assigns and keeps a record of storage locations and checks for coding errors. By relieving the programmer of these burdensome tasks, SPS significantly reduces the amount of programming time and effort required.

This manual is intended to serve as a reference text for 1620 SPS III and 1620-1443 SPS III. It assumes the reader is familiar with the methods of data handling and the functions of instructions in the IBM 1620 Data Processing System. For those without such knowledge, information may be found in the following publications:

IBM 1620 Central Processing Unit, Model 1 (Form A26-5706)

IBM 1620 Central Processing Unit, Model 2 (Form A26-5781)

IBM 1620 Input/Output Units (Form A26-5707)

IBM 1620 Data Processing System, Model 2 Binary Capabilities and Index Registers (Form A26-5764)

Machine Requirements

The minimum machine and special feature requirements for assembling with 1620 SPS III are as follows:

1. 1620 Data Processing System with 20,000 positions of core storage.
2. 1621 Paper Tape Unit or 1622 Card Read-Punch.
3. Indirect Addressing (standard feature of 1620-2).

In addition to the above requirements, the 1443 Printer is required when assembling with 1620-1443 SPS III.

Introduction

The SPS III Programming System may be divided into the symbolic language used in writing a program, the library containing the subroutines and linkage instructions (macro-instructions) that may be incorporated into the program, and the processor that is used to assemble the user's programs.

Symbolic Language

Symbolic language is the notation used by the programmer to write (code) the program. The program written in SPS language is called a "source program." This language provides the programmer with mnemonic operation codes, special characters, and other necessary symbols. The use of symbolic names (labels) makes a program independent of actual machine locations. Programs and routines written in SPS language can be relocated and combined as desired. Routines within a program can be written independently with no loss of efficiency in the final program. Symbolic instructions may be added or deleted without reassigning storage addresses.

Macro-instructions

The macro-instructions that are written in a source program are commands to the processor to generate the necessary linkage instructions. Linkage instructions provide the path to a subroutine and a return path to the user's program. These subroutines may be any of seventeen IBM Library subroutines like floating divide, square root, and arctangent; or special subroutines prepared by the user. The ability to process macro-instructions simplifies programming and further reduces the time required to write a program.

Processor Program

After a source program is written, it is punched into cards or into paper tape. The source program is then assembled into a finished machine language program known as the *object program*.

Assembly is accomplished by the SPS III Processor program. The function of the processor is to translate the symbolic language of the programming system into the machine language of the 1620. The translation is one for one — the processor produces one machine language instruction for each source statement (except macro-instructions).

Symbolic Programming

Symbolic programming may be defined as a method wherein names, characteristics of instructions, or closely related symbols are used in writing a program. The core of the symbolic language is the operation code. SPS permits the programmer to write (code) in a more simple, familiar language and does not require as detailed machine knowledge because, in coding the program, the programmer uses operation codes that are in easily remembered mnemonic form rather than in the numerical language of the machine. Operation codes are of three types: Declarative, Imperative, and Control.

Declarative Operation Codes

Declarative operation codes are used for assignment of core storage for input areas, output areas, and working areas. The assigned areas are utilized by the object program and may contain the data to be processed and/or the constants (numerical or alphabetic characters) required in the object program when the data is processed. Declarative statements never generate instructions in the object program, but may generate constants that are assembled as part of the object program.

Imperative Operation Codes

Imperative operation codes specify the operations or instructions that the object program is to perform. In this group are included all arithmetic, branching, and input/output statements. Most statements on the coding sheet prepared by the programmer are of this type. These statements are translated one for one and are assembled as the machine language instructions of the object program.

Control Operation Codes

Control operation codes are commands to the processor that provide the programmer with control over portions of the assembly process. Instructions of this type do not normally generate instructions in the object program.

The actual and mnemonic operation codes within these categories are presented under **PROGRAMMING THE 1620 USING SPS**.

Coding Sheet

The programmer enters all information relevant to the coding of the source program and subsequent assembly

of the object program on coding sheet, Form X26-5627 (Figure 1). Figure 2 shows a sample input card, Form J59692. The format of the input card or paper tape record follows the headings on the coding sheet. In paper tape, the first punching position of a record is said to be column 1. The card columns assigned to a single heading are referred to as a *field*. Following is an explanation of the headings in the order of their appearance on the sheet.

Heading Line

Space is provided at the top of each page for the name of the Program, Routine, and Programmer, and for the date. This information does not constitute part of the source program language and is not punched.

Page Number (Columns 1-2)

A 2-digit page number is entered to maintain the order of the program sheets. This entry (normally numerical)

becomes the first two digits of each statement that is punched from the sheet.

Line Number (Columns 3-5)

A 3-digit line number is entered on the sheet to maintain the sequence of the statements coded. The first 20 lines on each sheet are prenumbered 010, 020, 030, etc., through 200. At the bottom of the sheet, six nonnumbered lines are provided for inserts or for continuing the line numbering. The inserted statement should be numbered so that it falls sequentially between the statements immediately preceding and following it. The arrangement of the prenumbered lines, 010, 020, etc., permits up to nine statements to be inserted between any two statements. After the cards for each of the lines are punched, they should be placed in correct numerical order.

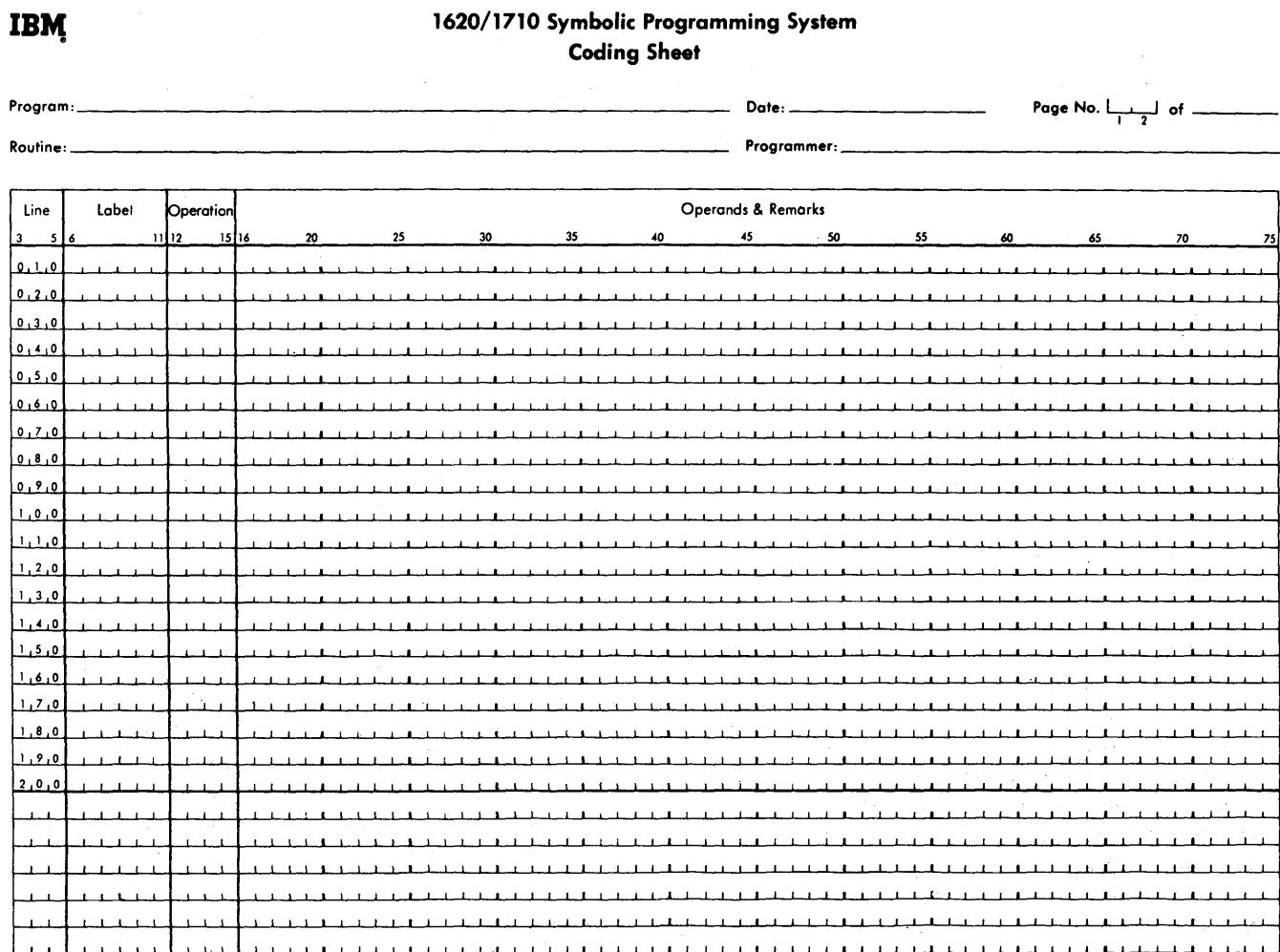


Figure 1. 1620/1710 SPS Coding Sheet

Figure 2. SPS Source Program Card

Label (Columns 6-11)

The label field represents the machine location of either data or instructions. The field may be left blank or may be filled with a symbolic address. Only the data or instructions that are referred to elsewhere in the program need have a label.

A label may consist of up to six alphameric characters beginning at the left-most position in the label field. At least one of the characters must be alphabetic or one of four permissible special characters, namely, the equal sign (=), slash symbol (/), at sign (@), and period (.).

The best labels to select are those that are mnemonically descriptive of the area or instruction to which they are assigned. Labels that have an obvious meaning not only provide easily remembered references for the original programmer but also assist others who may assume responsibility for the program.

Operation (Columns 12-15)

The four-position operation field contains the actual two-digit numerical operation code or the mnemonic representation of the operation code to be performed. If the first character is numerical, no check of the operation code occurs and the numerical parts of the two-digit internal representation of the first two characters is taken as the operation code, that is, if the programmer writes 4BNF, the resulting operation code is 42.

In either case, the first character of the operation code must start in the leftmost position, column 12, of the operation field. Listings of permissible mnemonic codes and actual operation codes are shown under PROGRAMMING THE 1620 USING SPS.

Operands and Remarks (Columns 16-75)

The operands and remarks field is used to specify the information that is to be operated upon and may contain, if desired, any additional remarks concerning the statement.

For declarative operation statements, the first operand usually defines the length. The remaining operands, if present, specify constants, an address, and remarks.

For imperative operation statements, the operands and remarks field contains, at most, four items: three of these are operands and the fourth, remarks. The first two operands may be the symbolic or actual address of data or instructions, the P and Q portions of the instruction. The third operand, which should be numerical, is called the flag indicator operand and is used to set flags in the assembled instruction. The final item consists of the remarks associated with each statement. Imperative statements need not contain all four items. Any one or more than one may be omitted.

A control operation statement normally consists of only one operand.

Statement Writing

Certain rules must be observed in writing or coding the statements that make up the source program. This section contains rules that apply to the statements and their elements, rules governing the length and types of statements, use of special characters, the flag indicator operand and immediate (Q) operand, types of addresses used as operands, and address adjustment by arithmetic, a method that relieves the programmer of considerable effort and reduces the number of symbols required for a source program.

Statements

Symbolic statements are classed according to the operation code they contain, and thus are designated Declarative, Imperative, or Control statements. In addition to the page and line number, a statement may contain a label, operation code, operands, and remarks. No statement in the source program may exceed 75 characters in length. Since page number, line number, label, and operation require 15 positions, the operands, and remarks field may not exceed 60 characters. In the case of the paper tape SPS, the end-of-line character is considered to be part of the operands and remarks field.

Use of Special Characters in Statement Writing

The comma, asterisk, end-of-line character, blank, at (@) sign, and dollar sign are special characters which possess distinct meanings in the writing of source programs. Their use as well as that of the special characters used as operators for address adjustment are explained in detail in this section.

COMMA

The comma is normally used to separate items in a statement. The term *item* refers here to parts of the operands and remarks field, such as the P and Q operands, the flag indicator operand, remarks, length, constants, etc. An imperative statement may consist of four items: the P and Q operands, the flag indicator operand, and remarks, but need not contain all four items. Any one or more than one may be omitted.

If one item is omitted and more items follow, the comma that normally follows the omitted item must be present. For example, if the flag indicator operand is omitted but remarks are present in the instruction, the format of the field will be:

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		TF	DELTAX,X									

All imperative statements that contain remarks must include three commas in the operands field, even when the operands are omitted. During assembly, the omitted P or Q operands will be replaced by zeros in the P or Q portion of the assembled instruction.

Commas indicating omission need not be present in statements in which the last item(s) is omitted. For example, in the statement in which both the flag indicator operand and remarks are omitted, no commas need be used following the second operand.

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		TF	DELTAX,X									

Examples

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		H	,,,HALT, INSTRUCTION,									
0.2.0		A	,,1.6352,,1.7865,,ADD, FACTOR, B, TO, A,									

Statement 010 which is a halt operation requires three commas (, ,) in front of the remarks, to take the place of the P, Q, and flag indicator operands. In statement 020, the first two commas set off the P and Q operands, whereas the third comma takes the place of the omitted flag indicator operand. The number of commas required for declarative statements may be one or two as explained under DECLARATIVE OPERATIONS.

ASTERISK

The asterisk has three uses: in writing comments (only), as an operand or term of an operand, and in address adjustment.

Lines of descriptive information may be inserted in the program by placing an asterisk (*) in column 6 of the label field. Comments then may be written in columns 7 through 75. Comments inserted in this way will appear in the symbolic output, but will not affect in any way the operation of the program. A comment statement does not produce an entry in the object program.

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0.1.0	*	REMARKS	CAN BE WRITTEN IN SUCH A MANNER SO									
0.2.0	*	A	S TO DESCRIBE THE PROGRAM									

Statements 010 and 020 are remarks that do not generate instructions.

The asterisk is used as the first character or *term* of an operand in an imperative statement and is interpreted by the program as the address of the high-order (leftmost) position of the address of the instruction. It may be used as any *term* of the operand to indicate the high-order (leftmost) position of the address of the instruction.

When the asterisk is used in address adjustment as an *operator*, it indicates to the processor that a multiplication must be performed in order to adjust the address.

PARENTHESES

Parentheses are used to enclose the integer that specifies which index register is to be used in modification of the operand.

Line	Label	Operation	Operands & Remarks											
			11-12	15-16	20	25	30	35	40	45	50	55	60	65
3 0.1.0	A.	CTR(A,4),NET												

END-OF-LINE CHARACTER

An end-of-line character (E) is required on source statements that are to be processed on paper tape. Use of this character allows statements to be located on the

tape immediately adjacent to each other, with no intervening blank characters. The statements are in "free" form; that is, they are not assigned a fixed number of positions.

Source statements that are to be processed in punched card form do not require an end-of-line character; the remainder of the line is left blank and this is recognized by the processor as the end of the statement.

When the end-of-line character is punched in a card for off-line conversion to paper tape, it is represented by a 12, 5, 8 punch combination.

BLANK CHARACTER

A blank character in operands of the source statements is ignored by the processor except in DAC statements (alphabetic constants), in which blanks are considered valid characters. In effect, the statement is condensed before it is processed.

Because blanks are ignored by the processor, the programmer, to achieve clarity on his coding sheets and output listing, may write his statements in modified "fixed" form as shown in the example at the bottom of the page. In this example, columns 16, 36, and 57 are arbitrary choices for the locations of the operands. The comma following or replacing the P operand may be in any column from 16 through 35; the comma following or replacing the Q operand must be in column 56, the position preceding the flag indicator operand.

Line	Label	Operation	Operands & Remarks											
			11-12	15-16	20	25	30	35	40	45	50	55	60	65
1 0.1.0	SW2	B ODDVN												
0.2.0	A	AREA							TEMP1-4					
0.3.0	*	INITIALIZATION FOR FSUBODD												F0+FNE
0.4.0	TF	XSUBN							DELTAX					
0.5.0	TFM	MULT+1							4					
0.6.0	TDM	SW2+1							9					10
0.7.0	TF	ACCM							2					
0.8.0	TF	TEMP3							DELTAX					
0.9.0	A	TEMP3							TEMP3					
1.0.0	B	ASINE-3*L												
1.1.0	ODDVN	A ACCUM							TEMP1					
1.2.0	A	XSUBN							TEMP3					
1.3.0	C	XSUBN							NINES					
1.4.0	BNH	ASINE-3*L												
1.5.0	MULT	MM ACCUM												
1.6.0	SF	B8												

Blanks are not permitted within a flag indicator operand. For paper tape input, this operand must begin immediately following the second column and must be immediately followed by a comma or end-of-line character. For card input, the flag indicator operand can be followed by a comma, record mark, or blanks in the remainder of the card. A blank or blanks in the address operand of a *declarative* statement, when set off by commas, is interpreted by the processor as a zero address.

"AT" SIGN

When the "at" sign (@) is used as part of a constant being defined by a DC, DSC, or DAC statement, a record mark (=) is created by the processor and inserted into the constant in place of the @. Specific rules for use of the @ are covered under DECLARATIVE OPERATIONS

DOLLAR SIGN

The dollar sign (\$) is used in an operand to instruct the processor that the symbolic address in an operand has a specific heading character. The \$ is written between the heading character and the symbol. For example, in an operand the heading character "5" and the symbol "SUM" appear as 5\$SUM. For additional information on the use of the \$, refer to HEAD-HEADING in the Control Operations section.

Operands

FLAG INDICATOR OPERAND

The flag indicator operand specifies the positions that are to be flagged in the assembled instruction. These positions are numbered from left to right, 0 through 11, and must be listed sequentially. For example, if positions 2, 7, and 10 are to be flagged, the flag indicator operand should be coded 2710, not 2107. All positions may be flagged, if desired. The operand then will be coded 01234567891011 and must be written in that order.

Normally, no flags are set when the flag indicator operand is omitted. However, if the flag indicator operand is omitted from all immediate instructions, except TDM, a flag is automatically set in position Q₇. If the operand is present, only the positions indicated are flagged.

The flag indicator operand can be used to insert a flag over the units position of the P and Q addresses, if the source program is written for a 1620 or 1710 that has Indirect Addressing.

IMMEDIATE (Q OPERAND)

With immediate-type instructions such as Add Immediate (AM), Subtract Immediate (SM), and with actual operation codes that begin with the digit 1, the Q operand represents the actual data to be used by the

instruction. It may be absolute or symbolic as previously defined. High-order zeros of absolute data may be eliminated.

During assembly, the processor automatically places a flag over position Q₇ of an immediate instruction unless a flag indicator operand indicates otherwise. For example, the statement

Line	Label	Operation	Operands & Remarks						
3	5 6	11 12 15 16	20	25	30	35	40	45	50
0 1 0	.	SM	TOTAL	,10023					

causes the numbers 10023 to be subtracted from the field called TOTAL because the flag that terminates the field to be subtracted is automatically placed over position Q₇. However, the statement

Line	Label	Operation	Operands & Remarks						
3	5 6	11 12 15 16	20	25	30	35	40	45	50
0 1 0	.	SM	TOTAL	,10023,10					

will cause only the number 23 to be subtracted from the field called TOTAL because the flag indicator operand directs that the field terminating flag be placed over position Q₁₀ rather than Q₇. There is one exception to this rule: a Transmit Digit Immediate instruction (TDM, code 15) does not require a flag; therefore, none is automatically set by the processor.

MASK DIGIT OPERAND

A mask digit operand is required to specify the mask digit for the Branch on Mask and Branch on Bit instructions. The D in the following examples shows the position of the mask operand in the source statement.

Line	Label	Operation	Operands & Remarks						
3	5 6	11 12 15 16	20	25	30	35	40	45	50
0 1 0	.	BMK	P, Q, D, FLAG, COMMENT						

which assembles to: 91 PPPPP DQQQQ

Line	Label	Operation	Operands & Remarks						
3	5 6	11 12 15 16	20	25	30	35	40	45	50
0 1 0	.	BKT	P, Q, D, FLAG, COMMENT						

which assembles to: 90 PPPPP DQQQQ

The mask operand may be symbolic or absolute. The P, Q, and flag operands are processed in the same man-

ner as for other instructions. If the mask operand is an absolute value, the units character replaces the Q₇ character of the Q operand. For example:

Given:

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0 1 0	A	DS	1 2 3 4 4									
0 2 0	B	DS	2 3 4 5 6									
0 3 0	MKD	DS	5									

BMK A,B,B,I

will assemble to: 91 12344 13456

If the mask operand is a symbol, the units position of the *symbolic address* will be inserted in the Q₇ position of the assembled instruction. For example:

Given:

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0 1 0	A	DS	1 2 3 4 4									
0 2 0	B	DS	2 3 4 5 6									
0 3 0	MKD	DS	5									

BMK A,B,MKD

will assemble to: 91 12344 53456

Types of Addresses Used as Operands

Operands assembled by the processor may be of three types: actual, symbolic, and asterisk. The individual applications for a particular type of address are described in the section PROGRAMMING THE 1620 USING SPS.

ACTUAL ADDRESS

An actual address consists of five digits 00000-19999 for a standard capacity machine and is, as the name implies, the actual core storage address of a piece of data or an instruction. High-order zeros of an actual address may be eliminated. For example, the statement

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0 1 0	A	3684,12251										

causes the data in storage location 12251 to be added to the data in storage location 03684.

SYMBOLIC ADDRESS

A symbolic address is the name assigned by the programmer to the location of an instruction or a piece of data. A symbolic address is valid only if it is defined (given an actual numerical value) by a declarative statement somewhere in the source program or if it is used as the label of an instruction. Symbolic addresses may contain from one to six characters (letters, digits, or special characters) with the following restrictions:

- a. At least one character must be nonnumerical.
- b. The only permissible special characters are: equal sign (=), slash symbol (/), at sign (@), and period (.).

It should be noted that blanks have no meaning within a symbol because they are eliminated during assembly.

The example shown below contains both an actual address and a symbolic address.

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0 1 0	A	TOTAL,12251										

In this example, the data in the field whose *actual* address is 12251 is added to a field whose address is the *symbolic name* TOTAL.

ASTERISK ADDRESS

When the asterisk is used as the first character of an operand in an imperative operation, it is interpreted by the processor as meaning the address of the high-order (leftmost) position of the instruction itself. For example, the statement

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0 1 0	BNF	START,*										

indicates to the processor that the Q portion of the instruction should contain the address of the instruction itself. This instruction is assembled as 44 01234 01876 where START equals 01234 and the address assigned to the instruction is 01876. Thus, when executed in the object program, this instruction examines its own leftmost position (01876) for a flag and either branches to the instruction at location 01234 or continues, on the basis of the examination, to the next instruction located at 01888.

When an asterisk (*) address is used with either declarative or control operations, it refers to the rightmost position of storage last assigned by the location assignment counter of the processor — not to the leftmost character of the instruction. For example, the statements

Line	Label	Operation	Operands & Remarks								
3	5 6	11 12 15 14	20	25	30	35	40	45	50		
0.1.0		TFM 12045, 70000									
0.2.0		DC 1, @, *									

produce the instruction

16 12045 7000+

Since record marks can be defined only in declarative operations, an imperative statement should be followed by a DC statement when a record mark is required in the instruction. The rightmost position of the instruction is the rightmost position of storage last assigned; therefore it is also the position where the + (constant) is stored.

Address Adjustment of Operands

Address adjustment is used to tell the processor to arithmetically adjust the addresses in operands. It is permitted with all types of addresses: actual, symbolic, and asterisk, and is used to refer to a location that is a given number of positions away from a specific address. Use of this feature of the language reduces the number of symbols necessary for a source program.

By writing a + (plus sign) for addition, — (minus sign) for subtraction, and * (asterisk) for multiplication, immediately after the first or subsequent term of an operand (an asterisk as a term of an operand does not represent multiplication but means the address of the instruction, as previously explained), the programmer indicates to the processor that the address is to be adjusted.

Arithmetically adjusted operands may take the form of $A \pm B \pm C \pm D$, where the terms A, B, C, and D may be numerical quantities. The number of terms in the operand is limited only by the size of the operand and remarks field. Thus the operand $A + B * C - D$ may be further adjusted by writing after the last term another term, E, e.g., $A + B * C - D + E$.

In arithmetically adjusted operands, the operation or operations of multiplication are always performed first, followed by the addition and subtraction required to calculate the adjusted address. Intermediate results that are greater than 10 digits, or a final re-

sult (adjusted address) that is over 5 digits, cannot be calculated by the processor.

In using address adjustment, the programmer should be careful that insertions or deletions do not affect the adjusted address. For example, if a P operand in a branch (B) instruction refers to an address as * +48 (i.e., branch to the instruction that follows the next three sequentially higher instructions), the programmer must ensure that no new instructions are introduced within the three instructions to make the * +48 incorrect. In this example the asterisk (*) is the leftmost position of the instruction itself.

Examples

Line	Label	Operation	ADJUSTED ADDRESS	ARITHMETIC	rba
2	5 6	11 12 15 14			
0.1.0		ALPHA+40	01040 = 1000+40		
0.2.0		ALPHA-30	00970 = 1000-30		
0.3.0		ALPHA+2*L	01008 = 1000+ (2x4)		
0.4.0		ALPHA*3	03000 = 1000 X 3		
0.5.0		ALPHA*L	04000 = 1000 X 4		
0.6.0		5,0,0+2,0*3-1,1	00549 = 500+(20X3)-11		
0.7.0		1,0,0+5,2,0*3-1,1	00549 = (100X5)+(20X3)-11		
0.8.0		*+12	02012 = 2000+12		
0.9.0		*+3*2	12000 = (2000X3) X 2		

The operands shown will produce the adjusted addresses, as indicated, provided the location 1000 has been assigned to the symbolic address ALPHA, the location 4 has been assigned the symbolic address L, and the instruction location (*) is equivalent to 2000.

In some instructions such as the branch instruction, the Q address is not used, although a zero (00000) address is generated. Thus the instruction uses 12 storage positions. By using an * address in the following statements

Line	Label	Operation	Operands & Remarks								
3	5 6	11 12 15 14	20	25	30	35	40	45	50		
0.1.0		B 1,3,6,8,									
0.2.0		DORG*-3									

Line	Label	Operation	Operands & Remarks
0.3.0	NEXT	TFM 12045, 70000	

the instructions are condensed, to eliminate four positions of the unused (zero) Q address, and are stored as

49136680161204570000

whereas the statements

Line	Label	Operation	Operands & Remarks								
3	5 6	11 12 15 14	20	25	30	35	40	45	50		
0.1.0		B 1,3,6,8,									
0.2.0	NEXT	TFM 12045, 70000									

are stored as

491366800000161204570000

because the unused Q address is not eliminated. In the first example, only four positions of storage are saved; however, a considerable amount of storage can be saved in a program that contains many instructions where the Q or both the Q and P portions of instructions are unused. Because the * in the DORC statement (see CONTROL OPERATIONS) refers to the rightmost position of storage last assigned (Q_{11} of the B instruction), * -3 is the address where the next instruction starts.

By placing a minus sign in front of the first term of an operand, a flag (minus sign) can be inserted over the units position of the adjusted address. This feature of address adjustment can be used for inserting flags required for Indirect Addressing. However, an operand written as -0 (minus zero) does not insert the flag in the units position over the zero. When the minus sign is written in front of the first term in order to set a flag over the units position, other signs following the first term should be reversed so that the correct address is obtained.

Operand Modification with Index Registers

Any operand that can be indirectly addressed may be modified with an index register. The index register is specified by placing the number of the index register in parentheses, as shown in the following examples,

Line	Label	Operation	Operands & Remarks
3	516		
3	516	11 17 15 16	20 25 30
0.1.0	A	ADDR-5(3)	SUBR(6)
0.2.0	A	ADDR-5(A3)	SUBR(A6)

either of which will be assembled as

21 XXXXX XXXXX

The number in parentheses must be an integer from 0 to 7. The processor decodes the number and places the proper flags over the operands. In the second example, the A in the index register portion of the P operand is ignored by the processor; however, it may be included by the programmer as an aid in keeping track of the index register band currently selected. The processor decodes the rightmost numerical character within the parentheses as the index register number.

Programming the 1620 Using SPS

This section describes the various steps to be followed in writing a program for the 1620 using SPS. The material is divided into three categories: Declarative Statements, Imperative Statements, and Control Statements.

Declarative Statements

In programming the 1620, all records and any other data that is to be processed by the program must be assigned storage areas. Normally, all records and data to be processed consist of fields of known length and arrangement. Unless otherwise specified, areas are automatically assigned core storage locations in the order in which they appear in the source statements.

To assign addresses for instructions, constants, etc., the processor uses an address assignment counter. This counter is adjusted for each assignment made by the processor. If an address is assigned by the programmer, the counter is not adjusted.

The declarative statements provide the object program with the input/output areas, work areas, and constants it requires to accomplish its assigned task. These statements do not produce instructions that are executed in the object program. The entries, DS, DSS, DAS, and DSB assign storage for work areas in the object program. The entries, DC, DSC, DAC, DSA, and DNB usually assign storage, and also produce, in the object program, both the machine address of the area assigned and the constants that are to be stored in this area. Constants are then loaded with the object program.

Declarative statements may be entered at any point in the source program. However, these statements are normally placed by themselves, preferably at the beginning or end of the program — not within the instruction area. If not placed at the beginning or end, the programmer is required to branch around an area assigned to data so the program will not attempt to execute what is in a data area as an instruction.

The declarative mnemonic operation codes and their description are as follows:

Code	Description
DS	Define Symbol (Numerical)
DSS	Define Special Symbol (Numerical)
DAS	Define Alphabetic Symbol
DC	Define Constant (Numerical)

DSC	Define Special Constant (Numerical)
DAC	Define Alphabetic Constant
DSA	Define Symbolic Address
DSB	Define Symbolic Block
DNB	Define Numerical Blank

DS — Define Symbol (Numerical)

A DS statement may be used to define symbols used in the source program (i.e., to assign storage addresses or values to symbolic addresses or labels) and to assign storage for input, output, or working areas. A DS statement does not cause any data to be loaded with the object program.

The length of the field is defined by the first operand. This operand may be an absolute value or a symbolic name. If a symbolic name is used, the symbol must previously have been defined as an absolute value, that is, it must have appeared in the label field in a statement of the source program preceding the one in which it is used. Address adjustment may be used with this operand.

The address in core storage of the field being defined may be assigned by the programmer or the programmer may let the processor assign the address. If the processor assigns the address, the statement is terminated after the first operand. If the programmer assigns the address, a second operand, which may be symbolic, asterisk, or actual, is used to establish the address of the field. Since data fields are addressed at their right-most (low-order) digit the processor assigns this position as the address of the field. Address adjustment may be used with the second operand. If the second operand is symbolic, it also must previously have been defined. Addresses assigned by the programmer do not disrupt the sequence of addresses assigned by the processor.

A DS statement may also be used to define a symbol, without assigning any storage, i.e., to define it as an absolute value. In this case, the first operand is omitted (or written as 0) and the second operand represents the value (may not exceed five digits in length). The second operand may be an actual value or a previously defined symbol. To define storage which will not be referred to symbolically, the label of the DS statement may be omitted.

The following statements define the field length only. When remarks are added to the statement, the field length must be followed by two commas.

Line	Label	Operation	Operands & Remarks							
3	5 6	11 12 13 14	20	25	30	35	40	45	50	
0.1.0	DELTAX	DS	,7							
0.2.0	DELTAX	DS	,7, TWO, COMMAS, REQUIRED, FOR, REMARKS							

In the next example, the programmer assigns the address of the field and excludes the field length (the first operand) from the statement and replaces it with a comma. The following statements cause the processor to associate the address 12930 with the label SUM:

Line	Label	Operation	Operands & Remarks							
3	5 6	11 12 13 14	20	25	30	35	40	45	50	
0.1.0	SUM	DS	,12930							
0.2.0	SUM	DS	,12930, TWO, COMMAS, AGAIN, REQUIRED							

Again, in this example, two commas are required when remarks form part of the statement.

The following statement, which is similar to the one previously given, is assigned a value that is other than an address.

Line	Label	Operation	Operands & Remarks							
3	5 6	11 12 13 14	20	25	30	35	40	45	50	
0.1.0	FL	DS	,17, FLD LENGTH, FOR, SUBSEQUENT, STMTS							

This statement defines the symbol FL as being equivalent to the value 17. Subsequent uses of this symbol are permitted because the symbol has been defined.

It should be noted that an area defined by the processor for a DS statement is always addressed at the rightmost position. However, to use this area for input/output, the leftmost digit must be addressed. This is done by using a DSS statement in place of a DS statement or by address adjustment with a DS statement, which subtracts a number that is one less than the length of the area from the address of the area. In a previous example, where DELTAX was defined as having a field length of 7, the operand of another instruction to read numerical data into the DELTAX field should be written as DELTAX—6.

DSS — Define Special Symbol (Numerical)

The DSS statement is similar to the DS statement with one exception: when the second operand is omitted,

the processor assigns the leftmost position as the address of the field. If a second operand is assigned by the programmer, this address is assumed to be equivalent to the leftmost position of the field. A DSS statement is normally used to define a storage area for input/output. The data in such an area may be moved during execution of the object program by a Transmit Record instruction which requires that an address assigned to an area be that of the leftmost position.

DAS — Define Alphabetic Symbol

The DAS statement is similar to the DS statement with two exceptions:

1. The length specified by the first operand is automatically doubled by the processor to allow for alphabetic data. Each alphabetic character requires two storage positions.
2. The address of the field, if generated by the processor, is the leftmost position of the field plus one. The position is always odd-numbered, as it must be with any alphabetic field.

The following example illustrates a DAS statement.

Line	Label	Operation	Operands & Remarks							
3	5 6	11 12 13 14	20	25	30	35	40	45	50	
0.1.0	TITLE	DAS	30							
0.2.0	TITLE	DAS	30, REMARKS, REQUIRE, TWO, COMMAS							

This statement defines an area for input/output that can contain 30 alphabetic characters. The processor assigns 60 positions in core storage to accommodate alphabetic coding. The omission of the second operand causes the processor to assign an address. During internal transmission of a field which utilizes an input/output area that is defined with a DAS, the area must be addressed at its rightmost position. In the example, the address may be achieved through address adjustment, i.e., TITLE + 2 * 30 - 2.

DC — Define Constant (Numerical)

The DC statement may be used to enter numerical constants into the object program, and, for ease of reference, to assign names to the constants. The label field contains the name by which the constant is known. DC statements consist of three operands. The first operand indicates the length of the constant field; the second, the actual constant; the third, the storage address of the constant. The third operand is not used when the

programmer prefers to let the processor assign the storage address. The assigned address is the rightmost storage position of the constant. The leftmost storage position is the position over which the processor places a flag.

Whenever remarks form part of a dc statement, three commas must be included in the statement. The first and third operands may be symbolic or actual. They are subject to address adjustment. A symbolic address must previously have been defined to be valid.

If the first operand (length of constant) is smaller than the constant, an invalid condition results. If it is larger than the constant, zeros are inserted to the left of the constant so that the number of zeros plus the number of positions in the constant equals the length of the field (first operand).

A constant that is a positive number will be stored in the form of an unsigned integer; a negative number, in the form of a signed integer. A negative number has the minus sign written in front of the constant as part of the second operand. During assembly, a negative number produces a flag (minus sign) over the units position of the constant.

If the constants 0100000 and -0004337769 are required, they may be defined as follows:

Line	Label	Operation	Operands & Remarks
3	5 6	11 12 15 16	20 25 30 35 40 45 50
0.1.0	CONST DC	7,1,00000	
0.2.0	CONST DC	7,1,00000,,3	COMMAS FOR REMARKS
0.3.0	CONST DC	10,-4337769	
0.4.0	CONST DC	10,-4337769,,3	COMMAS FOR REMARKS

In both cases, constant 1 and constant 2, the length of field is greater than the constant, and the addresses of the constants are assigned by the processor. These constants will appear in the object program as

0100000
0004337769

A record mark may be used in a constant but must be in the units position and must be written as the character @. The following example contains statements that:

1. Store a record mark by itself as a constant.
2. Store a constant 6 and record mark.
3. Store a minus 0773 and record mark.

Line	Label	Operation	Operands & Remarks
3	5 6	11 12 15 16	20 25 30 35 40 45 50
0.1.0	RMARK DC	1,@,STORE, A RECORD MARK ONLY	
0.2.0	CONST DC	2,6@,,STORE, A SIX AND RECORD MARK	
0.3.0	CONST DC	5,-773@,,STORE, A MINUS 773 AND RM	

These constants appear in the object program as:

±
6 ±
0773 ±

A constant 7 with a flag (7) is generated by either of the following statements:

Line	Label	Operation	Operands & Remarks
3	5 6	11 12 15 16	20 25 30 35 40 45 50
0.1.0	CONST DC	1,-7,,STORE A 7 WITH A FLAG	
0.2.0	CONST DC	1,7,,STORE A 7 WITH A FLAG	

A flag is always placed over a one-digit constant (except a record mark), regardless of the sign. Therefore the programmer must use two positions to define a positive one-digit constant.

Constants may not exceed 50 characters. The following statement generates a constant containing 50 zeros.

Line	Label	Operation	Operands & Remarks
3	5 6	11 12 15 16	20 25 30 35 40 45 50
0.1.0	ZERO DC	50,0,,STORE FIFTY ZEROS	

To store a zero with a flag at location 00401, the following statement can be used:

Line	Label	Operation	Operands & Remarks
3	5 6	11 12 15 16	20 25 30 35 40 45 50
0.1.0	DC	1,-0,401,,STORE A ZERO WITH A FLAG	

Because a label is not included in this statement, the actual address (00401) must be used by any other instruction when referring to this constant.

DSC — Define Special Constant (Numerical)

The dsc statement is similar to the dc statement with two exceptions:

1. When the third operand is omitted, the address assigned by the processor to the field is that of the leftmost position of the field. If the third operand is present, the address of the constant is assumed to be the leftmost position of the field, and the constant will be stored with its leftmost digit at this address when the object program is loaded.
2. A flag is not placed in the leftmost position of the field.

DAC — Define Alphameric Constant

To define a constant consisting of alphameric data, the mnemonic **DAC** is used. The **DAC** statement is similar to the **DC** statement with three exceptions:

1. The first operand (length) is automatically doubled by the processor to allow two storage positions for each alphameric character.
2. The storage address of the constant is the address of the leftmost position plus one. This address must be an odd-numbered address to comply with the requirements for alphameric data storage. An odd-numbered address will automatically be assigned, if it is assigned by the processor. If it is specified by the programmer (as in line 020 of the following example), the processor assigns the specified address and provides that the constant is stored beginning one position to the left of the specified address. In the latter case, the processor makes no test of whether or not the address is odd-numbered or whether the address (or the position to the left) has been previously assigned.
3. High-order zeros are not automatically inserted in the constant by the processor, as in the case with a **DC** statement when the field length exceeds the number of characters. The number of characters including blank characters should not be greater or less than the specified length (first operand). When the rightmost position or positions of the constant are blank characters, they should be followed by a comma or end-of-line character. For card input, the rightmost position must be followed by a comma or a record mark.

NOTE: Only **DAC** and **DNB** instructions may be used to insert blank characters into storage.

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	13	14	20	25	30	35	40	45	50
0.1.0	NOTE1	DAC	1	7	,DECK	34,78	PUNCHED,,	,	END	MESSAGE,,			
0.2.0		DAC	6,				9,0,1	,STORE	6,	ALPHA	BLKS,,		
0.3.0	RMARK	DAC	1,	0	,	1	ALPHA	RM	FOR	OUTPUT	AREA,,		
0.4.0	CONST	DAC	1,3,	DELTAX=0.0000	,	3,	STORE	,	CNST,	RM,,			

In the example shown:

1. Statement 010 uses 34 storage positions to store the 17-position constant (deck 3478 punched).
2. Statement 020 places 6 alphameric blanks into storage locations 00900 through 00911. Also, a flag is set in location 00900.

3. Statement 030 records an alphameric record mark in storage.
4. Statement 040 places a 13-position constant, including a record mark, in storage.

A 50-character alphameric constant (maximum allowable) occupies 100 positions of storage. A flag is set over the leftmost position of the field. Addressing this constant for internal field transmission requires the address **OUTPUT +50 * 2-2**, where **OUTPUT** is the symbol (label) which represents the leftmost address plus one.

DSA — Define Symbolic Address

The **DSA** statement may be used to store a series of up to ten addresses as constants, as part of the object program. These addresses can be used for instruction modification or for setting up a table of addresses through which the programmer may index to modify a routine.

Each entry (symbolic or actual) in the operands field, with the exception of the last entry, is followed by a comma. The equivalent machine address of each entry is stored as a 5-digit constant. The constants are stored adjacent to each other with a flag over the high-order position of each. The label field of this statement must contain the symbolic name by which the table of constants may be referred to. An address at which this table is stored in core storage may not be assigned by the programmer nor may any remarks be included in the **DSA** statement. The address assigned by the processor is the address at which the rightmost digit of the first constant will be located.

NOTE: If the last operand is followed by a comma, an additional zero address (00000) is assembled in the table.

In the example that follows, symbols **ALPHA**, **ORIGIN**, and **OUTPUT** are equivalent to addresses 01000, 00600, and 15000, respectively.

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	13	14	20	25	30	35	40	45	50
0.1.0	TABLE	DSA	ALPHA,,	ORIGIN,,	1,2,3,4,,	OUTPUT-50,,							

The constants are stored as

01000006000123414950
 ↑ ↑
 (01200) (01204)

If the leftmost digit of these four constants is located at 01200, then the address equivalent to TABLE will be 01204, the location of the rightmost digit of ALPHA.

DSB — Define Symbolic Block

A DSB statement is used to define an area of storage for storing a numerical array. A DSB statement does not cause any data to be loaded with the object program. The label of this statement is converted to the address at which the first element of the array is stored (i.e., the rightmost position of the first element). The first operand indicates the size of each element, the second, the number of elements.

Either or both operands may be symbolic or actual. If symbolic, the symbol must have been previously defined. A third operand is required if the programmer wishes to assign the address. For example, to store an array of 75 elements, each element containing 15 digits, the statements used would be:

Line	Label	Operation	Operands & Remarks								
3	5 6	11 12 15 16	20	25	30	35	40	45	50		
0.1.0	ARRAY	DSB	15	75	, 1514						

In this example, the array begins at location 01500 (leftmost position of the first 15-digit element). ARRAY is equivalent to 01514 (address of the first element).

DNB — Define Numerical Blank

A DNB statement is used to define a field of numerical blanks. (The 8-4 card code denotes a numerical blank.) Up to fifty blanks may be specified in each DNB statement. In addition to a label, two operands can be assigned by the programmer. The first of these specifies the number of blank characters desired (field length) and the second, the rightmost address of the field where the blanks are stored in the object program.

If the second operand is omitted and the statement is labeled, the address assigned to the label by the processor is the rightmost storage position of the blank field. The blank field does not contain a flag in its leftmost position.

If the programmer wishes to move a blank field in core storage, he must either define a single-digit constant with a flag bit in the position in front of the leftmost position of the blank field or a record mark in the position following the rightmost position of the blank field.

If six numerical blanks are required, they may be defined as follows:

Line	Label	Operation	Operands & Remarks								
3	5 6	11 12 15 16	20	25	30	35	40	45	50		
0.1.0	BLANKS	DNB	6								

The processor assigns the storage address of the six blank positions to the label BLANKS. In the example that follows, the programmer assigns the storage address as 01625.

Line	Label	Operation	Operands & Remarks								
3	5 6	11 12 15 16	20	25	30	35	40	45	50		
0.1.0	BLANKS	DNB	6, 1625, STORE SIX NUMERICAL BLANKS								

In a DNB statement, two commas are required whenever remarks are included in the statement; the first after the length operand and the second after, or in place of, the address operand.

Summary of Declarative Statements

As stated earlier, areas being defined by the processor are assigned core storage locations in the order in which they are processed. To do this, the processor program uses a location assignment counter to keep track of the address of the last assigned storage location. Table 1 shows the amount added to the location assignment counter for each instruction and summarizes the coding and operation of each declarative operation. "Alpha Record Address" in the table refers to the leftmost position plus one of an alphameric field, whereas "Field Address" refers to the rightmost position of a field. The term "Numerical Record Address" refers to the leftmost position of a field.

Imperative Statements

This section describes the five classes of Imperative statements and gives examples of statements in symbolic form. For a detailed description of the function of each instruction, refer to the appropriate machine reference manual.

- Imperative statements are divided into five classes:
1. Arithmetic
 2. Internal Data Transmission
 3. Logic
 4. Input/Output
 5. Miscellaneous

Arithmetic Statements

Table 2 lists the Arithmetic statements, some of which pertain to special features. Since some features are "special" for the 1620 Model 1 and "standard" for the

Table 1. Summary of Declarative Operations

DECLARATIVE STATEMENT FORMAT		OPERANDS	AMOUNT ADDED TO LOCATION ASSIGNMENT COUNTER IF ADDRESS (A) IS BLANK	VALUE STORED IN SYMBOL TABLE AS EQUIVALENT TO "SYMBOL"	DATA FIELDS WHICH ARE LOADED AS A PART OF THE OBJECT PROGRAM
LABEL	OP CODE				
SYM	DS	L,A	L (length). If L is blank, 0 is added.	A address. If A is blank, the field address from the location assignment counter is stored.	None.
SYM	DSS	L,A	L (length). If L is blank, 0 is added.	A address. If A is blank, the numerical record address from the location assignment counter is stored.	None.
SYM	DAS	L,A	2 x L is added. If L is blank, 0 is added.	A address must be odd. If A is blank, the alpha record address from the location assignment counter is stored.	None.
SYM	DC	L,C,A	L is added.	A address. If A is blank, the field address from the location assignment counter is stored.	C, the (numerical) constant.
SYM	DSC	L,C,A	L is added.	A address. If A is blank the numerical record address from the location assignment counter is stored.	C, the (numerical) constant.
SYM	DAC	L,C,A	2 x L is added.	A address must be odd. If A is blank, the alpha record address from the location assignment counter is stored.	C, the (alphabetic) constant.
SYM	DSA	D,E,F,G, H,I,J,K, L,M	5 x (number of addresses) is added.	Field address of the first address on list.	A list of the actual addresses that correspond to D,E,F, etc.
SYM	DSB	L,N,A	Length of each element times the number of elements is added.	A address. If A is blank, field address of the first element is stored.	None.
SYM	DNB	L,A	L is added.	A address. If A is blank, the field address from the location assignment counter is stored.	Number of blank characters that equal L.

1620 Model 2, no indication is made in the table to differentiate between the two types.

Examples

These statements cause the following operations to be performed:

Line 010 — Add labor amount to cost amount.

020 — Same as line 010 except three commas are required for remarks.

030 — Subtract a constant 02 from the field located at STORE plus 4.

040 — Add a constant 05 to the field at storage location 00088.

050 — Move DDND (dividend) to the product area (storage location 00097).

Line	Label	Operation	Operands & Remarks
3 5 6			11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
0,1,0	A	COST, LABOR	
0,2,0	A	COST, LABOR,, ADD LABOR TO COST	
0,3,0	SM	STORE+4,2,10	
0,4,0	AM	88,05,10,HALF-ADJUST, POSITIVE AMT	
0,5,0	LD	97,DDND	
0,6,0	D	86,DVR	

Table 2. Arithmetic Instructions

NOTE: Indirect Addressing and indexing are allowable with all P address operands listed below. An * to the left of the Q operand indicates these features may be used with it.

OPERATION	OPERATION CODES		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Add	A	21	Storage address of units position of augend	*Storage address of units position of addend
Add Immediate	AM	11	Same as code 21	Q ₁₁ of instruction is units position of addend
Subtract	S	22	Storage address of units position of minuend	*Storage address of units position of subtrahend
Subtract Immediate	SM	12	Same as code 22	Q ₁₁ of instruction is units position of subtrahend
Multiply	M	23	Storage address of units position of multiplicand	*Storage address of units position of multiplier
Multiply Immediate	MM	13	Same as code 23	Q ₁₁ of instruction is units position of multiplier
Load Dividend	LD	28	Storage address in product area to which units position of field (dividend) is to be transmitted	*Storage address of units position of dividend
Load Dividend	LDM	18	Same as code 28	Q ₁₁ of instruction is units position of dividend
Divide	D	29	Storage address at which first subtraction of the divisor occurs	*Storage address of units position of divisor
Divide Immediate	DM	19	Same as code 29	Q ₁₁ of instruction is units position of divisor
Floating Add (Special Feature)	FADD	01	Storage address of units position of exponent of augend	*Storage address of units position of exponent of addend
Floating Subtract (Special Feature)	FSUB	02	Storage address of units position of exponent of minuend	*Storage address of units position of exponent of subtrahend
Floating Multiply (Special Feature)	FMUL	03	Storage address of units position of exponent of multiplicand	*Storage address of units position of exponent of multiplier
Floating Divide (Special Feature)	FDIV	09	Storage address of units position of exponent of dividend	*Storage address of units position of exponent of divisor

060 – Divide the dividend by successive subtractions of the DVR (divisor), starting in storage location 00086.

Internal Data Transmission Statements

Table 3 lists the mnemonics for Internal Data Transmission statements. Some statements pertain to instructions that require special features; however, some special instructions for the 1620 Model 1 are standard on the 1620 Model 2. No indication is made in the table to differentiate between the two types.

These statements cause the following instructions to be executed:

Line 010 – A numerical digit at the location called DIGIT is moved to the location called FIELD.

Line	Label	Operation	Operands & Remarks
3	3	TD	FIELD, DIGIT
4	4	TDM	FIELD, 3
5	5	TF	STORE, RATE1,, MOVE RATE 1 TO STORE
6	6	TFM	STORE, 3525,, MOVE 03525 TO STORE
7	7	TFM	*-1,1,41,10,CHGE, PREV, OP, CODE, TO NOP
8	8	TNS	A,B,, CONVERT FLD A TO NUMER CODING
9	9	TNF	C,D,, CONVERT FLD D TO ALPHA CODING

020 – A digit 3 is moved to the location called FIELD.

030 – RATE 1 is moved to the field called STORE.

040 – A constant 3525 is moved to the location called STORE.

050 – A constant 41 is moved to O₀ and O₁ positions of the preceding instruction in the object program.

060 – Field A is moved to field B and converted from alphabetic coding (2 digits per character) to numerical coding (1 digit per character).

070 – Field D is moved to field C and converted from numerical coding to alphabetic coding.

Logic Statements

Table 4 lists the mnemonics for Logic statements. Note that the Branch Indicator (BI) and Branch No Indica-

Table 3. Internal Data Transmission Instructions

NOTE: Indirect Addressing and indexing are allowable with all P address operands listed below. An * to the left of the Q address operand indicates these features may be used with it.

OPERATION	OPERATION CODES		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Transmit Digit	TD	25	Storage address to which single digit is transmitted	*Storage address of single digit to be transmitted
Transmit Digit Immediate	TDM	15	Same as code 25	Q ₁₁ of instruction is the single digit to be transmitted
Transmit Field	TF	26	Storage address to which units position of field is transmitted	*Storage address of units position of field to be transmitted
Transmit Field Immediate	TFM	16	Same as code 26	Q ₁₁ of instruction is the units position of the field to be transmitted
Transmit Record	TR	31	Storage address to which high-order position of the record is transmitted	*Storage address of high-order position of the record to be transmitted
Transmit Record No Record Mark	TRNM	30	Same as code 31	*Same as code 31
Transfer Numerical Strip	TNS	72	Storage address of rightmost position of alphabetic field to be transmitted	*Storage address of the units position of the numerical field
Transfer Numerical Fill	TNF	73	Storage address of rightmost position of alphabetic field	*Storage address of the units position of the numerical field to be transmitted
Floating Shift Right (Special Feature)	FSR	08	Storage address to which units (rightmost) digit of mantissa is transmitted	*Storage address (rightmost) digit of mantissa to be transmitted
Floating Shift Left (Special Feature)	FSL	05	Storage address to which high-order digit of the mantissa is transmitted	*Storage address of low-order digit of mantissa to be transmitted
Transmit Floating	TFL	06	Storage address to which units position of exponent is transmitted	*Storage address of units position of exponent of field to be transmitted
Move Address (Special Feature)	MA	70	Storage address of units position of 5-digit field to which data is transmitted	*Storage address of units position of 5-digit field to be transmitted
OR to Field (Special Feature)	ORF	92	Storage address of leftmost position of first field for OR logic input	*Storage address of leftmost position of second field for OR logic input
AND to Field (Special Feature)	ANDF	93	Storage address of leftmost position of first field for AND logic	*Storage address of leftmost position of second field for AND logic
Exclusive OR to Field (Special Feature)	EORF	95	Storage address of leftmost position of first field for Exclusive OR logic	*Storage address of leftmost position of second field for Exclusive OR logic
Complement Octal Field (Special Feature)	CPLF	94	Storage address of leftmost position of field to which data is transmitted	*Storage address of leftmost position of field to be complemented
Octal to Decimal Conversion (Special Feature)	OTD	96	Storage address of the units position of the power-of-eight table	*Storage address of leftmost position of field to be converted
Decimal to Octal Conversion (Special Feature)	DTO	97	Storage address of the units position of the highest power-of-eight required	*Storage address of leftmost position of field to be converted

Table 4. Logic (Branch and Compare) Instructions

NOTE: Both the BI (Branch Indicator) and BNI (Branch No Indicator) instructions require one of the switch or indicator codes listed in Table 21 as a Q address. The code indicates the switch or indicator to be interrogated for status. To relieve the programmer of having to code a Q address, unique mnemonics are included in SPS language for both BI- and BNI-type instructions. For a unique mnemonic, the processor generates the actual machine language code 46 (Branch Indicator) or 47 (Branch No Indicator) and the Q address that represents the switch or indicator.

Indirect Addressing and indexing are allowable with all P address operands listed below except Branch Back. An * to the left of the Q address operand indicates these features may be used with it.

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Compare	C	24	Storage address of units position of the field to which another field is compared	*Storage address of units position of the field to be compared with the field at the P address
Compare Immediate	CM	14	Same as code 24	Q_{11} of instruction is units position of the field to be compared with the field at the P address
Branch	B	49	Storage address of the leftmost digit of the next instruction to be executed	Not used
Branch No Flag	BNF	44	Storage address of the leftmost digit of next instruction to be executed if branch occurs	*Storage address to be interrogated for presence of a flag bit
Branch No Record Mark	BNR	45	Same as code 44	*Storage address to be interrogated for presence of a record mark character
Branch No Group Mark	BNG	55	Same as code 44	*Storage address to be interrogated for presence of a group mark character
Branch On Digit	BD	43	Same as code 44	*Storage address to be interrogated for a digit other than zero
Branch Indicator	BI	46	Storage address of leftmost position of next instruction to be executed if indicator tested is on	Q_8 and Q_9 of instruction specify the program switch or indicator to be interrogated (see Table 5)
Unique Branch Indicator Mnemonics:				
Branch High	BH	46	Same as BI	None required
Branch Positive	BP	46	Same as BI	None required
Branch Equal	BE	46	Same as BI	None required
Branch Zero	BZ	46	Same as BI	None required
Branch Overflow	BV	46	Same as BI	None required
Branch Any Data Check	BA	46	Same as BI	None required
Branch Not Low	BNL	46	Same as BI	None required
Branch Not Negative	BNN	46	Same as BI	None required
Branch Band A Selected	BBAS	46	Same as BI	None required
Branch Band B Selected	BBBS	46	Same as BI	None required
Branch Neither Band Selected	BNBS	46	Same as BI	None required
Branch Console Switch 1 On	BC1	46	Same as BI	None required
Branch Console Switch 2 On	BC2	46	Same as BI	None required

Table 4. Logic (Branch and Compare) Instructions (cont'd)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Branch Console Switch 3 On	BC3	46	Same as BI	None required
Branch Console Switch 4 On	BC4	46	Same as BI	None required
Branch Last Card	BLC	46	Same as BI	None required
Branch Exponent Check (Special Feature)	BXV	46	Same as BI	None required
Branch on Channel 9	BCH9	46	Same as BI	None required
Branch on Channel Overflow	BCOV	46	Same as BI	None required
Branch No Indicator	BNI	47	Storage address of leftmost position of next instruction to be executed if indicator tested is off	Q_8 and Q_9 of instruction specify program switch or indicator to be interrogated (see Table 5)
Unique Branch No Indicator Mnemonics:				
Branch Band A Not Selected	BANS	47	Same as BNI	None required
Branch Band B Not Selected	BBNS	47	Same as BNI	None required
Branch Either Band Selected	BEBS	47	Same as BNI	None required
Branch Not High	BNH	47	Same as BNI	None required
Branch Not Positive	BNP	47	Same as BNI	None required
Branch Not Equal	BNE	47	Same as BNI	None required
Branch Not Zero	BNZ	47	Same as BNI	None required
Branch No Overflow	BNV	47	Same as BNI	None required
Branch Not Any Data Check	BNA	47	Same as BNI	None required
Branch Low	BL	47	Same as BNI	None required
Branch Negative	BN	47	Same as BNI	None required
Branch Console Switch 1 Off	BNC1	47	Same as BNI	None required
Branch Console Switch 2 Off	BNC2	47	Same as BNI	None required
Branch Console Switch 3 Off	BNC3	47	Same as BNI	None required
Branch Console Switch 4 Off	BNC4	47	Same as BNI	None required
Branch Not Last Card	BNLC	47	Same as BNI	None required

Table 4. Logic (Branch and Compare) Instructions (cont'd)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Branch Not Exponent Check (Special Feature)	BNXV	47	Same as BNI	None required
Branch and Transmit	BT	27	P address minus one is the storage address to which the units position of the Q field is transmitted. P address is leftmost digit of the next instruction to be executed	*Storage address of units position of the field to be transmitted
Branch and Transmit Immediate	BTM	17	Same as code 27	Q_{11} of instruction is units position of field to be transmitted
Branch and Transmit Address	BTA	20	Same as code 27	*Storage address of units position of the field to be transmitted
Branch and Transmit Address Immediate	BTAM	10	Same as code 17	Q_{11} of instruction is units position of field to be transmitted
Branch Back	BB	42	Not used	Not used
Branch and Transmit Floating	BTFL	07	P address minus one is the storage address to which the units position of the exponent portion of the Q field is transmitted. P is the storage address of the leftmost digit of the next instruction to be executed	*Storage address of units position of exponent of field to be transmitted
Branch and Select	BS	60	Storage address of the leftmost position of the next instruction	Q_{11} specifies condition to be selected
Unique Branch and Select Mnemonics:				
Branch and Select Indirect Addressing	BSIA	60	Same as BS	None required
Branch and Select No I/A	BSNI	60	Same as BS	None required
Branch and Select Band A (Special Feature)	BSBA	60	Same as BS	None required
Branch and Select Band B (Special Feature)	BSBB	60	Same as BS	None required
Branch and Select No Index Register (Special Feature)	BSNX	60	Same as BS	None required
Branch and Modify Index Register (Special Feature)	BX	61	Same as BS	**Storage address of units position of field to be added to selected index register
Branch and Modify Index Register Immediate (Special Feature)	BXM	62	Same as BS	**Five digits of Q field are added to selected index register
Branch Conditionally, Modify Index Register (Special Feature)	BCX	63	Same as BS if (after modification) IX sign has not changed or result is not zero	Same as BX

Table 4. Logic (Branch and Compare) Instructions (cont'd)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Branch Conditionally, Modify Index Register Immediate (Special Feature)	BCXM	64	Same as BCX	Same as BX
Branch and Load Index Register (Special Feature)	BLX	65	Same as BS	**Storage address of units position of 5-digit field to be loaded to selected index register
Branch and Load Index Register Immediate (Special Feature)	BLXM	66	Same as BS	**Five digits of Q field are loaded to selected index register
Branch and Store Index Register (Special Feature)	BSX	67	Same as BS	**Storage address of units position of field where selected index register data is to be stored
Branch on Bit (Special Feature)	BBT	90	Storage address of the leftmost position of next instruction if comparison is successful	*Q ₈₋₁₁ specifies storage address of units position of field to be compared with bits of the Q ₇ digit
Branch on Mask (Special Feature)	BMK	91	Same as code 90	*Q ₈₋₁₁ specifies storage address of units position of field to be compared with Q ₇ digit

** Specific index register is selected by flags over the Q₈₋₁₀ positions of the instruction.

tor (BNI) instructions require one of the switch or indicator codes listed in Table 5 to be a Q address. The code indicates the switch or indicator to be interrogated for status. To relieve the programmer of having to code the Q address, unique mnemonics are provided for most of the possible combinations of operation codes and indicators.

labeled START.

060 – Branch unconditionally to an instruction whose address is saved in IR-2 or PR-1.

Table 5. Switch and Indicator Codes used as Actual Q Addresses in BI and BNI Instructions

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0	C	B, A, 1, 1, COMPARE FIELD A WITH FIELD B											
0.2.0	B	START, 1, 1, 1, BRANCH TO LABEL START											
0.3.0	BJ	START, 1, 00, 1, IF SWI. ON, BR. TO START											
0.4.0	BC1	START, 1, SAME AS LINE 030											
0.5.0	BNC1	START, +3#1, 2, 1, 1											
0.6.0	BB												

These statements cause the following operations to be performed in the object program, as follows:

Line 010 – Compare field A with field B.

020 – Branch to an instruction labeled START.

030 – If Program switch 1 is on, branch to the instruction labeled START.

040 – Same as line 030 with the exception that the unique mnemonic operation code used does not require a Q address.

050 – If Program switch 1 is not on, branch to the third instruction following the one

Q ADDRESS					SWITCH OR INDICATOR
Q ₇	Q ₈	Q ₉	Q ₁₀	Q ₁₁	
X	0	1	Y	Y	Program Switch 1
X	0	2	Y	Y	Program Switch 2
X	0	3	Y	Y	Program Switch 3
X	0	4	Y	Y	Program Switch 4
X	0	6	Y	Y	Read Check Indicator*
X	0	7	Y	Y	Write Check Indicator*
X	0	9	Y	Y	Last Card Indicator (special feature)
X	1	1	Y	Y	High/Positive Indicator
X	1	2	Y	Y	Equal/Zero Indicator
X	1	3	Y	Y	High/Positive or Equal/Zero Indicator
X	1	4	Y	Y	Overflow Check Indicator
X	1	5	Y	Y	Exponent Check Indicator
X	1	6	Y	Y	MBR-Even Check Indicator*
X	1	7	Y	Y	MBR-Odd Check Indicator*
X	1	9	Y	Y	Any Data Check
X	2	5	Y	Y	Printer Check*
X	3	3	Y	Y	Channel 9
X	3	4	Y	Y	Channel 12
X	3	5	Y	Y	Printer Busy

X indicates any digit value or blank is permissible

Y indicates any digit value is permissible

* indicates the Any Data Check indicator (19) also is on when this indicator is on.

Input and Output Statements

Table 6 lists the mnemonics for Input/Output statements. If a two-character mnemonic (for example, RA) is used, the unit code must be included as part of the Q operand. The unit codes are shown in Table 7.

Examples

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		WA	OUT	PUT	,100								
0.2.0		WAT	Y	OUT	PUT	,11	SAME	AS	LINE	0.10			
0.3.0		K	,10	1.1	,11	SAME	AS	LINE	0.40				
0.4.0		SPTY											

These statements cause the following operations to be performed in the object program.

- Line 010 – Type out alphabetic data from a storage location called OUTPUT.
- 020 – Same as line 010, however, a unique mnemonic is used.
- 030 – Single space on the typewriter.
- 040 – Same as line 030; however, a unique mnemonic is used.

Miscellaneous Statements

The statements that are listed in Table 8 are those that do not fit in any group already described.

Examples

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		CF	OUT	PUT	-5								
0.2.0		MF	3	3	5	2	,1	6	9	4			
0.3.0		H	,0	,0	3	HALT	1						
0.4.0		NOP											

These statements cause four different operations to be performed in the object program, as follows:

- Line 010 – Clear a flag at the storage location, OUTPUT minus 5.
- 020 – Move a flag from storage location 01694 to storage location 03352.
- 030 – Cause the program to halt.
- 040 – Perform no operation but proceed to the next sequential instruction.

Processor Control Statements

Control statements are orders to the processor (SPS III assembly program) that give the programmer control over portions of the assembly process. The SPS language includes the following control statements.

DORG	Define Origin
DEND	Define End
SEND	Special End
HEAD	Heading
TCD	Transfer Control and Load
TRA	Transfer to Return Address

With the exception of the TRA and DORG, none of the above statements may be labeled.

DORG — Define ORIGIN

The DORG statement instructs the processor to override its automatic assignment of storage and to begin the assignment of succeeding entries at the particular location specified by the programmer. In this way the programmer is able to control assignment of storage to instructions, constants, and data. If a Define Origin statement is not the first entry in a source program, the processor begins the assignment of storage at location 00402.

A Define Origin statement is coded as follows:

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		DORG	7	8	2	0							

This statement directs the processor to reset its location assignment counter to the particular address specified in the operand (actual or symbolic), and this causes the assignment of succeeding entries to begin at this address. When an actual address is entered by the programmer, care must be taken to avoid inadvertent overlapping with areas assigned by the processor.

If the operand is left blank, assignment of storage starts with address 00000. Since the arithmetic tables are stored in locations 00100 through 00401, constants and instructions cannot occupy these storage locations.

If a symbolic address is entered, it must appear as a label earlier in the program sequence. An * address refers to the current contents of the location assignment counter. A Define Origin statement can take any of the following individual forms:

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		DORG	X	Y	Z								
0.2.0		ORIGIM	DORG	X	Y	Z	+5.0						
0.3.0		ORIGIM	DORG	*+5.0	LOCATION	ASSIGN	COUNTER	+5.0					

Table 6. Input and Output Instructions

NOTE: Indirect Addressing and indexing are allowable with all P address operands, where a P operand is required. None of the Q operands shown may be used with Indirect Addressing or Index Registers.

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Read Numerically	RN	36	Storage address at which leftmost (first) numerical character is stored	Q_8 and Q_9 of instruction specify input unit
Unique Read Numerically Mnemonics:				
Read Numerically Typewriter	RNTY	36	Same as RN	None required
Read Numerically Paper Tape	RNPT	36	Same as RN	None required
Read Numerically Card	RNCD	36	Same as RN	None required
Write Numerically	WN	38	Storage address from which leftmost (first) numerical character is written	Q_8 and Q_9 of instruction specify output unit
Unique Write Numerically Mnemonics:				
Write Numerically Typewriter	WNTY	38	Same as WN	None required
Write Numerically Paper Tape	WNPT	38	Same as WN	None required
Write Numerically Card	WNCD	38	Same as WN	None required
Print Numerically	PRN	38	Same as WN	None required
Print Numerically and Suppress Spacing	PRNS	38	Same as WN	None required
Dump Numerically	DN	35	Same as WN	Same as WN
Unique Dump Numerically Mnemonics:				
Dump Numerically Typewriter	DNTY	35	Same as WN	None required
Dump Numerically Paper Tape	DNPT	35	Same as WN	None required
Dump Numerically Card	DNCD	35	Same as WN	None required
Printer Dump	PRD	35	Same as WN	None required
Printer Dump and Suppress Spacing	PRDS	35	Same as WN	None required
Read Alphamerically	RA	37	Storage address at which numerical digit of leftmost (first) character is stored. (Zone digit of first character is at P minus one.)	Q_8 and Q_9 of instruction specify input unit

Table 6. Input and Output Instructions (cont'd)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Unique Read AlphamERICALLY Mnemonics:				
Read Alpha-merically Typewriter	RATY	37	Same as RA	None required
Read Alpha-merically Paper Tape	RAPT	37	Same as RA	None required
Read Alpha-merically Card	RACD	37	Same as RA	None required
Read Binary Paper Tape (Special Feature)	RBPT	37	Same as RA	None required
Write Alpha-merically	WA	39	Storage address of numerical digit of left-most (first) character to be written. (Zone digit of first character is at P minus one.)	Q ₈ and Q ₉ of instruction specify output unit
Unique Write AlphamERICALLY Mnemonics:				
Write Alpha-merically Typewriter	WATY	39	Same as WA	None required
Write Alpha-merically Paper Tape	WAPT	39	Same as WA	None required
Write Alpha-merically Card	WACD	39	Same as WA	None required
Write Binary Paper Tape (Special Feature)	WBPT	39	Same as WA	None required
Print Alpha-merically	PRA	39	Same as WA	None required
Print Alpha-merically and Suppress Spacing	PRAS	39	Same as WA	None required
Control	K	34	Not used	Q ₈ and Q ₉ specify input/output unit. Q ₁₁ specifies control functions
Unique Control Mnemonics				
Backspace Typewriter	BKTY	34	Not used	None required
Tabulate Typewriter	TBTY	34	Not used	None required
Index Type-writer	IXTY	34	Not used	None required
Return Carriage Typewriter	RCTY	34	Not used	None required
Space Type-writer	SPTY	34	Not used	None required
Skip Immediate	SKIP	34	Not used	See Table 14
Skip After Printing	SKAP	34	Not used	See Table 14

Table 6. Input and Output Instructions (cont'd)

OPERATION	OPERATION CODE		OPERANDS	
	MNEMONIC	ACTUAL	P ADDRESS	Q ADDRESS
Space Immediate	SPIM	34	Not used	See Table 15
Space After Printing	SPAP	34	Not used	See Table 15
Seek		34	Storage address of disk control field	X07X1
Read Disk/WLRC		36	Same as Seek	X07X0
Write Disk/WLRC		38	Same as Seek	X07X0
Check Disk/WLRC		36	Same as Seek	X07X1
Read Disk Track/WLRC		36	Same as Seek	X07X4
Write Disk Track/WLRC		38	Same as Seek	X07X4
Check Disk Track/WLRC		36	Same as Seek	X07X5
Read Disk		36	Same as Seek	X07X2
Write Disk		38	Same as Seek	X07X2
Check Disk		36	Same as Seek	X07X3
Read Disk Track		36	Same as Seek	X07X6
Write Disk Track		38	Same as Seek	X07X6
Check Disk Track		36	Same as Seek	X07X7

If xyz (label) is previously defined as 01002, the first entry directs the processor to begin the assignment of succeeding entries at location 01002. The second entry directs the processor to begin the assignment of succeeding entries at the location that has been assigned

to the symbol xyz plus 50. The symbol ORIGIN can be used at any point in the program to refer to that address. The third entry directs the processor to begin the assignment of succeeding entries at the address specified by the current contents of the location assignment counter plus 50. A comma must follow the operand when remarks are included in a DORG statement.

Table 7. Input and Output Unit Codes Used as Actual Q Addresses in RN, WN; DN, RA, and WA Instructions

Q ADDRESS					DEVICE
Q ₇	Q ₈	Q ₉	Q ₁₀	Q ₁₁	
X	0	1	Y	Y	Typewriter
X	0	2	Y	Y	Paper Tape Punch
X	0	3	Y	Y	Paper Tape Reader
X	0	4	Y	Y	Card Punch
X	0	5	Y	Y	Card Reader
X	0	9	Y	Y	Printer

X indicates any digit value or blank is permissible
 Y indicates any digit value is permissible

DEND — Define END

The DEND statement is the last statement entered in the source program, it instructs the processor that all statements of the source program have been processed. A DEND statement may also be used to cause execution of the object program to begin immediately after it has been loaded. To do this, the DEND statement requires the presence of an operand representing the starting address of the program. The operand may be actual or symbolic. The 1620 will halt at the completion of loading of the object program, and execution then will begin at the address corresponding to the starting address, upon depression of the Start key. If the operand specifying the starting address is omitted, the program will halt and the operator will have to start the program manually.

Table 8. Miscellaneous Instructions

OPERATION	OPERATION CODE			OPERANDS								Q ADDRESS
	MNEMONIC	ACTUAL	P ADDRESS									
Set Flag	SF	32	Storage address at which flag X bit is placed									Not used
Clear Flag	CF	33	Storage address from which X flag bit is cleared									Not used
Move Flag	MF	71	Storage address to which flag X bit is moved									Storage address of flag bit to be moved X
Halt	H	48	Not used									Not used
No Operation	NOP	41	Not used									Not used

The following statements illustrate both types of entries.

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		DEND											
0.2.0													
0.2.0		DENDSTART											

The program is halted after *loading* by either statement. In the second case, execution begins at the address corresponding to START, upon depression of the Start key.

When a DEND statement includes comments but no operand, the operand must be replaced by a comma.

SEND — Special END

A SEND statement is provided to halt the tape processor on both passes of the source program. If a SEND statement is encountered by the card processor, the card processor will not be halted. This statement does not produce any output in the object program.

The SEND statement is used:

1. To halt the processor after one tape of a source program is processed and to allow the remainder of the source program from another tape to be threaded and then processed.
2. To halt the processor so that program switch settings may be changed and processing resumed.

The SEND statement takes the following form and requires no operands.

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		SEND											

When this statement in the source program is encountered by the processor, the program halts and the

message "LOAD NEXT TAPE" is typed. The operator threads the next tape or changes switch settings or both, and then depresses the Start key.

If the first part of the source program is entered from the typewriter, Program Switch 1 will be off and the statements will be entered one at a time. When the SEND statement is entered, the processor halts. The operator may then turn on Program Switch 1, thread the source program tape, and continue processing the source program. SEND is especially useful where a long source program is punched in tape and certain addresses associated with labels being defined by that tape must be changed. For example, the following statements, part of source program A, may require that addresses 01000 and 02000 be changed to 01250 and 02500, respectively.

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		CONTRLDS	,0	1000									
0.2.0		NDIGITDS	,0	2000									

To change these addresses, the operator will enter the following statements at the typewriter

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0.1.0		CONTRLDS	,0	1250									
0.2.0		NDIGITDS	,0	2500									
0.3.0		SEND											

and follow them by source program A. The first time labels CTRL and NDIGIT are encountered by the processor, they are assigned addresses; the second time they are encountered, no addresses are assigned but an error message is typed. In this case, the operator can ignore the error message.

In some cases, the programmer may choose to end a source program with a **SEND** statement rather than a **DEND** statement. This situation allows the programmer to enter additional statements, either additional declarative statements or corrections to imperative statements. However the last additional statement entered must be a **DEND** statement.

HEAD — HEADING

It is frequently convenient, and sometimes necessary, to write a source program piecemeal, and to assemble these pieces into the total program. Parts of the program may be written by different programmers, or by the same programmer at different times with considerable time lapses between.

Suppose, in such a situation, that a program block, say B_1 has been written; that another program block, B_2 , is in the course of being written; and that B_1 and B_2 eventually are to be joined to compose a single program. Certain symbols may already have been used in writing block B_1 , and certain symbols, varying from the symbols used in B_1 , may be used in writing block B_2 . To avoid duplication of symbols in each block, the programmer writing block B_2 must be concerned with the symbols used in B_1 .

Symbols used in block B_2 can duplicate those in B_1 , provided they are less than six characters in length and have been prefaced by a **HEAD** statement. The programmer can completely ignore the symbols in B_1 by prefacing B_2 with the following control statements:

Line	Label	Operation	Operands & Remarks					
2 3 5 6		11 12 13 14	20	25	30	35	40	45 50
0 1 9		HEAD X						

where the single character X may be any one of the characters A to Z, 0 to 9, or blank.

The control instruction, **HEAD X** generates no instructions or data in the object program. When the processor encounters a **HEAD** statement, it treats the symbols in the label or operands fields of the following statements, provided the symbols are less than six characters in length, as though they were headed by the character X. The processor continues to do this until it encounters another **HEAD** statement.

Thus the symbols used in block B_1 which contain less than six characters cannot possibly conflict with the symbols used in block B_2 . Six-character symbols are not affected, that is, a six-character label, **COMMON**, following the control instruction **HEAD 9** is not treated as **9COMMON**, for it would be a seven-character symbol, and only a maximum of six characters can be handled by the symbol table.

A symbol is said to be "unheaded" if, and only if, its representation uses exactly six characters. The symbol **COMMON**, for example, is unheaded. The symbol **ALPHA** whose length is less than six characters is considered to be headed, whether under a **HEAD** control instruction or not. If **ALPHA** is under control of **HEAD X**, then **ALPHA** is said to be "headed by X." If **ALPHA** is not under control of any **HEAD** instruction, then **ALPHA** is said to be "headed by blank."

A symbol, **ALPHA**, headed by the character X, is not identical with the symbol **XALPHA**. The leading character is essentially on a different level from the characters which make up the symbol. However, **ALPHA** headed by a blank should be regarded as identical to the symbol **ALPHA** used without a heading statement.

If a **HEAD** statement with a nonblank character does not occur in the entire source program, all considerations of heading can be ignored.

A **HEAD** statement with a blank character must be used if the programmer desires to modify the heading process in the example. Note that the statement **HEAD** and the statement **HEAD 0** are quite different. For example, if block B_1 and B_2 are to be joined in one program, and B_2 must be nested somewhere in the middle of B_1 , as follows:

<u>Operation</u>	<u>Operands</u>
:	: } first part of block B_1
:	: }
HEAD	X
:	: } block B_2
:	: }
HEAD	
:	: } second part of block B_1
:	: }

the entire program might have been prefaced by a **HEAD** statement with a blank character operand. As implied previously, however, such a **HEAD** instruction is superfluous, since the symbols in the first part of block B_1 are automatically headed by blank, being under the control of no **HEAD** instruction at all.

Often it is inconvenient to refer to a symbol that is defined in another headed region because of the requirement that the symbol be six characters in length. To facilitate cross referencing between headed blocks, the following convention can be used:

Suppose that a symbol, say **SUM**, under **HEAD 1**, has been defined by some instruction. Suppose further that this symbol is to be referred to in an instruction under

the control of the instruction HEAD 2. Then the desired reference can be made by writing 1\$SUM as it appears in the following instructions.

Line	Label	Operation	Operands & Remarks								
3 5 6		11 12 15 16	20	25	30	35	40	45	50		
0.1.0	A.	TOTAL, 1\$SUM									

In general, if the two-characters "C\$", where C is any allowable heading character, is placed in front of the headed reference symbol SUM, then the result is SUM headed by C. To specify SUM headed by blank, one simply writes \$SUM, with no character preceding the \$ character.

If the processor finds an operand containing a six-character symbol plus a head character, such as 9COMMON, the processor will produce an error message indicating that the symbolic address contains more than six characters (see **ERROR MESSAGES, ER 5**).

If a label is used in a **HEAD** statement, it is ignored.

TCD — Transfer Control and Load

The TCD statement may be used to cause the processor to produce an unconditional branch instruction. When this instruction is encountered during the loading of the object (machine language) program, it causes the loader to stop the normal loading process and to branch to the location (ADDR) specified in the operand.

Line	Label	Operation	Operands & Remarks								
3 5 6		11 12 15 16	20	25	30	35	40	45	50		
0.1.0		T.C.D ADDR									

ADDR may be actual or symbolic.

This statement allows programs which are too large to fit into core storage to be loaded and executed piece-meal, by terminating each piece with a TRA statement. In effect, a TCD instruction can be used in conjunction with a DORG statement to execute portions of the program that have already been loaded into storage and to overlap these with other instructions.

Whenever the processor encounters a TCD statement, it causes the arithmetic tables, an unconditional branch instruction, and the loader program, to be punched into the object program tape or cards in that order. Therefore, when the object program is being loaded, the arithmetic tables will be loaded into storage before the branch occurs. Because the arithmetic tables are loaded into a portion of storage previously occupied by the loader program, the loader program will be destroyed.

However, it will be restored (again loaded into storage) when a TRA statement is encountered in the object program. That statement will be at the end of the portion of the object program that has been executed.

During assembly, the TCD instruction does not affect the location assignment counter or alter the symbol table.

TRA — Transfer to Return Address

The TRA statement causes the normal loading sequence of an object program to be resumed once it has been broken by a TCD statement. When a TRA statement is encountered by the processor, a read-a-record (card or tape) instruction and an unconditional branch instruction to the loader program are produced in the object program. This processor control operation increments the location assignment counter by 24. The last statement of that part of a source program that is executed, when loading is interrupted by a TCD statement, must be a TRA statement. When the TRA instruction equivalences are encountered in the object program, the program loader is reloaded and the normal loading process continues. The TRA statement, which takes the following form, uses no operands.

Line	Label	Operation	Operands & Remarks								
3 5 6		11 12 15 16	20	25	30	35	40	45	50		
0.1.0		TRA									

The following example illustrates the use of the TCD and TRA mnemonics.

Line	Label	Operation	Operands & Remarks								
3 5 6		11 12 15 16	20	25	30	35	40	45	50		
0.1.0	START	(first instruction)									
0.2.0			
0.3.0			
0.4.0			
0.5.0			
0.6.0			
0.7.0		TRA									
0.8.0		T.C.D START									
0.9.0		DORG START									
0.9.0		(Remaining Instructions)									
1.0.0			
1.1.0			
1.2.0			

The TCD statement causes a branch to the location assigned to the symbol START, followed by the execution of instructions from START through the TRA statement. The TRA statement causes a branch to the load program, which resumes loading of the remainder of the object program beginning with the location labeled START.

The use of a macro-instruction preceding a TCD statement is not allowed.

1620 Subroutines

A program or routine is a set of coded instructions that are arranged in a logical sequence; it is used to direct the 1620, or any IBM data processing system, to perform a desired operation or series of operations. Generally, programs contain one or more short sequences of instructions that are parts or subsets of the entire program and that are used to solve a particular part of a problem. These parts of the program or routine are called *subroutines*.

Usually, a subroutine performs a specific function, is common to a number of programs, and may be executed several times during the course of the program of which it is a part (main program). For example: a subroutine that extracts the square root of a number may be required several times during the execution of a pipe stress analysis program. The same subroutine may be used to extract a square root in a bridge and truss design program.

An efficient programming procedure is obviously one in which all necessary subroutines are coded only once, are retained on file, and are incorporated into a program whenever the operation performed by the subroutine is required. IBM Programming Systems has developed a group of subroutines that are more frequently required because of their general applicability. Seventeen subroutines are available; they fall into three general categories: arithmetic, data transmission, and functional.

Arithmetic subroutines

- Floating Point Add
- Floating Point Subtract
- Floating Point Multiply
- Floating Point Divide
- Fixed Point Divide

Data transmission subroutines

- Floating Shift Right
- Floating Shift Left
- Transmit Floating
- Branch and Transmit Floating

Functional subroutines

- Floating Point Square Root
- Floating Point Sine
- Floating Point Cosine
- Floating Point Arctangent
- Floating Point Exponential (natural)
- Floating Point Exponential (base 10, common)
- Floating Point Logarithm (natural)
- Floating Point Logarithm (base 10, common)

The methods used by the functional floating point subroutines to evaluate the functions of arguments are shown in Table 9.

Table 9. Subroutine Method for Evaluating Arguments

SUBROUTINE	METHOD	
	FIXED LENGTH	VARIABLE LENGTH
Square Root	Odd integer	Odd integer
Sine and Cosine	Based on Hastings' approximation*	Series approximation
Arctangent	Truncated series	Series approximation for arctangent
Exponential (natural and base 10)	Hastings' approximation 10^B . 10^B is converted to e^B	Series approximations of 10^B and convert to e^B
Logarithm (natural and base 10)	Truncated series for $\ln B$. $\ln B$ is converted to $\log 10^B$	Series approximation of $\ln B$ and convert to $\log B$

* Hastings, Cecil Jr., Approximations for Digital Computers, Princeton University Press, Princeton, New Jersey. The Rand Corporation, 1955.

The combined subroutines are written in machine language and are provided in card or paper tape form for floating point numbers with either a fixed-length or variable-length mantissa. The term "variable length" or "fixed-length," as applied to subroutines in this manual, refers to the number of digits (*L*) in the mantissa, not to the length of the subroutine itself.

The five types of subroutine card decks or paper tapes are:

1. *Fixed* length subroutines for machines *not* equipped with the automatic divide feature.
2. *Fixed* length subroutines for machines equipped with the automatic divide feature.
3. *Variable* length subroutines for machines *not* equipped with the automatic divide feature.
4. *Variable* length subroutines for machines equipped with the automatic divide feature.
5. *Variable* length subroutines for machines equipped with automatic *floating point* feature (automatic divide feature, prerequisite).

Although type 2 and type 4 subroutines are designed to work with the automatic divide feature, the "fixed point divide" subroutine is included as part of the subroutine package. Type 5 variable length subroutines that are designed to work with the automatic floating point feature include a complete set of subroutines as part of the package.

A PICK subroutine is included in the object program with any of the 17 subroutines previously mentioned. This subroutine performs the function of getting the data specified for a subroutine and storing the result produced by that subroutine.

The processor selects the subroutines used by the source program that are to be included in the object deck or tape. When the object deck or tape is loaded, the subroutines are loaded to the first even-numbered address following the object program. Although this address is assigned by the processor, care must be exercised by the programmer to provide a storage area, between the position assigned by the processor and position 19999 (standard capacity machine), that is large enough to accommodate the subroutines called for. To find the amount of storage required for the subroutines, the programmer may total the storage requirements of the subroutines used.

The fixed point divide subroutine (DIV) is used by some floating point functional subroutines. For this reason it will automatically be incorporated into a program which uses these subroutines when the machine used to run the program is not equipped with automatic divide.

In addition to the subroutines provided, the user may include up to twelve subroutines of his own. The method used to incorporate these routines into a program is explained under ADDING SUBROUTINES

Subroutine Macro-Instructions

All linkages for the 1620 subroutines are generated automatically through the use of certain macro-instructions. The programmer places the macro-instruction that is related to a particular subroutine in the source program at the point at which the subroutine is desired. This causes the SPS processor, during assembly, to generate linkage to the desired subroutine. In addition, the processor arranges for the subroutine to be added to the object program.

The data and addresses required by the subroutine and supplied in the macro-instruction are incorporated into the linkage instructions where they are made available for use. In this way the subroutine obtains the information it requires to perform its given task and also to compute a return address to the main program. Control is returned to the main program at the completion of the subroutine by transferring to the return address. The macro-instruction statement related to each subroutine is as follows:

Arithmetic Subroutines Macro-instructions

Line	Label	Operation	Operands & Remarks
2 5 6		11 12 13 14 20 25	
0.1.0		F A , B	(Floating Add)
0.2.0		F S , B	(Floating Subtract)
0.3.0		F M , B	(Floating Multiply)
0.4.0		F D , A , B	(Floating Divide)
0.5.0		D I V , A , B , A 1 , B 1	(Divide)

Data Transmission Subroutines Macro-instructions

Line	Label	Operation	Operands & Remarks
2 5 6		11 12 13 14 20	
0.1.0		F S R S A , B	(Floating Shift Right)
0.2.0		F S L S A , B	(Floating Shift Left)
0.3.0		T F L S A , B	(Transmit Floating)
0.4.0		B T F S A , B	(Branch and Transmit Floating)

Functional Subroutines Macro-instructions

Line	Label	Operation	Operands & Remarks
2 5 6		11 12 13 14 20	
0.1.0		F S Q R A , B	(Floating Square Root)
0.2.0		F S I N A , B	(Floating Sine)
0.3.0		F C O S A , B	(Floating Cosine)
0.4.0		F A T M A , B	(Floating Arctangent)
0.5.0		F E X A , B	(Floating Exponential, Natural)
0.6.0		F E X T A , B	(Floating Exponential, Base 10)
0.7.0		F L N A , B	(Floating Logarithm, Natural)
0.8.0		F L O G A , B	(Floating Logarithm, Base 10)

In the arithmetic statements, the B operands represent the addresses of quantities to be added, subtracted, etc., to quantities at addresses specified by the A operands. For the fixed point divide routine, two additional operands, A1 and B1, are required. These operands, as well as the A and B operands, are explained in greater detail under each macro-instruction as it is described. In the data transmission statements, the B operand generally represents the address of the field to be transmitted, whereas the A operand represents the address to which the field is to be transmitted. The function of the A and B operands differs slightly for functional subroutine macro-instructions. In this case, the B operand represents the address of the argument to be evaluated and the A operand represents the address where the result is to be placed in storage.

Indirect addressing can be used with the operands of all macro-instructions on machines with or without the automatic floating point feature. To indicate an indirect address, an operand should be preceded by a minus sign. An indirect address in the form \bar{xxxx} is generated

by the processor. A flag is automatically placed in the leftmost and rightmost positions of the address. If indirect addressing is attempted on a machine that does not have the indirect addressing feature, the flagged operand is *not* interpreted as an indirect address.

For each macro-instruction statement in a source program, two machine language linkage instructions, and a 5-digit address for each operand, are generated by the processor in the object program. These linkage instructions replace the macro-instruction, which never appears in the object program. A label written with a macro-instruction references the leftmost position of the first linkage instruction generated. If the programmer wishes to use this label in address adjustment, he must remember that the instruction location following a macro-instruction is not `LABEL + 12`.

When using a macro-instruction, the programmer must code the exact number of operands required for that macro-instruction. Every macro-instruction must have at least two operands. Remarks and flag operands are not permitted in macro-instructions. Omitted operands require the insertion of commas as in Imperative statements.

All operands in macro-instructions may be symbolic or actual; all are subject to address adjustment. If an * is used as an operand, its address is that of the left-most position of the first linkage instruction.

The linkage instructions generated for a macro-instruction by the processor are *equivalent* to the following series of symbolic instructions:

Line	Label	Operation	Operands & Remarks									
3	5 6	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		TFM	PICK + 11, * + 23									
0.2.0		B	SUBR									
0.3.0		DORG	* - 4									
0.4.0		DSA	A, B, C, D									

where `PICK` is the address of the first instruction in the Pick subroutine; `SUBR` is the secondary linkage for the desired subroutine (the subroutine specified by the macro-instruction); and `A, B, C, D...` are the series of 5-digit addresses or constants that are equivalent to the operands specified by the macro-instruction. This linkage allows the macro-instruction to contain any number of operands, an ability that is significant for "adding subroutines."

During execution of the object program, the secondary linkage instructions set up the address of the first instruction in the desired subroutine as a part of one of the instructions in the Pick subroutine. Secondary linkage instructions are generated by the processor for each subroutine used by the source program. They are equivalent to the following symbolic instructions.

For *arithmetic* subroutines (not including `DIV`):

`SUBR TFM PICK + 402, ADDR`
`B PICK`

For *data transmission* subroutines (not including `FSRS, FSLS`):

`SUBR TFM PICK + 402, ADDR`
`B PICK + 104`

For *functional* subroutines:

`SUBR TFM PICK + 402, ADDR`
`B PICK + 104`

For `DIV, FSRS, and FSLS` subroutines:

`SUBR TF ADDR + 11, PICK + 11`
`B ADDR`

`PICK` is the address of the first instruction of the Pick subroutine, and `ADDR` is the actual address where the first instruction of the desired subroutine is located. In the secondary linkage, `PICK + 402` is the address equivalent for the variable length subroutines only. That address should be replaced by `PICK + 414` for fixed length subroutines. For variable length subroutines using the floating point feature, the secondary linkage instructions generated are equivalent to the following symbolic instructions:

For *arithmetic* subroutines:

`SUBR TF ADDR + 11, PICK + 11`
`B ADDR`

For *data transmission* subroutines (not including `FSRS, FSLS`):

`SUBR TF PICK + 174, ADDR`
`B PICK + 24`

For *functional* subroutines:

`SUBR TF PICK + 174, ADDR`
`B PICK + 24`

The subroutine card deck or paper tape will contain the subroutines in the order shown. All except the first three are floating point subroutines.

1. Subroutine Processor
2. Pick
3. Divide (fixed point)
4. Subtract – Add
5. Multiply
6. Divide
7. Square Root
8. Cosine – Sine
9. Arctangent
10. Exponential (natural and base 10)
11. Logarithm (natural and base 10)
12. Shift Right
13. Shift Left
14. Transmit Floating
15. Branch and Transmit Floating

Many subroutines have been paired (i.e., add and subtract, sine and cosine, natural and base 10 exponential, natural and base 10 logarithm), into single subroutines to conserve storage by sharing program steps which are common to both. The individual subroutines within each pair are distinguished from each other solely by the point at which they are entered. The correct entry point is obtained through use of the macro-instruction pertaining to the particular subroutine desired.

Because the arctangent subroutine and both logarithm subroutines (natural and base 10) require the fixed point divide routine, the macro-instructions **FATN**, **FLN**, and **FLOG** cause the fixed point divide subroutine to be added to the object program output (provided the machine is not equipped with automatic divide). For the fixed length mantissa subroutines, any of the previously listed subroutines 3-8, 10, 12-15, may be called for and added to the object output without calling any other subroutine; i.e., floating add-subtract may be called without calling floating multiply. For the variable length mantissa subroutines, any of the four arithmetic macro-instructions (**FA**, **FS**, **FM**, **FD**) will cause all three arithmetic subroutines (4. floating add-subtract, 5. floating multiply, and 6. floating divide) to be called for and added to the object program output. However, the other subroutines may be called for independently of each other.

In the object program output, each subroutine except the Pick subroutine is preceded by its secondary linkage. However, when the object program is loaded, the secondary linkages for all of the subroutines are stored in storage positions that immediately follow the object program, starting with the first even-numbered address. **PICK** and other subroutines are loaded into sequentially higher addresses (in the order previously listed).

The subroutine processor is loaded into an area of storage which is occupied by the SPS processor, thereby destroying a portion of the SPS processor. To restore the SPS processor to its original status, the part destroyed must be reloaded after selection of the subroutines is completed. Restoration of the SPS processor is accomplished automatically with the data which follows the last subroutine of the subroutine deck or tape. This data includes the loader, arithmetic tables, and that part of the SPS processor previously destroyed. The subroutine processor is never a part of the object program output or final object program.

Floating Point Arithmetic

Scientific and engineering computations frequently involve lengthy and complex calculations necessitating the manipulation of numbers that may vary widely in

magnitude. To obtain a meaningful answer, problems of this type usually require retention of as many significant digits as possible during calculation, and correct positioning of the decimal point at all times. When the computer is used for such problems, several factors must be considered, of which the most important is the location of the decimal point.

In general, a computer does not recognize the presence of a decimal point in any quantity during calculation. A product of 41454 results whether the factors are 9.37×44.2 ; $93.7 \times .442$; or 937×4.42 ; etc. The programmer must be cognizant of the location of the decimal point during and after the calculation and arrange the program accordingly. In adding, the decimal points of all numbers must be lined up to obtain the correct sum. The programmer facilitates this arrangement by shifting the quantities as they are added. In the manipulation of numbers that vary greatly in magnitude, it is conceivable that the resulting quantity could exceed allowable working limits.

Processing numbers expressed in ordinary form, e.g., 427.93456, 0.0009762, 5382, -623.147, 3.1415927, etc., can be accomplished on a computer only with extensive analysis to determine the size and range of intermediate and final results. The percentage of time required for this analysis and subsequent number scaling is frequently much larger than the percentage of time required to perform the actual calculation. Moreover, number scaling requires complete and accurate information regarding the bounds on the magnitude of all numbers that come into the computation (input, intermediate, output). Since prediction of the size of all numbers in a given calculation is not always possible, analysis and number scaling are sometimes impractical.

To alleviate this programming problem, a system must be employed which provides information regarding the magnitude of all numbers in the calculation along with the quantities in the calculation. Thus, if all numbers are represented in some standard, predetermined format that instructs the computer in an orderly and simple fashion as to the location of the decimal point, and if this representation is acceptable to the routine that performs the calculation, then quantities that range from minute fractions having many decimal places to large whole numbers having many integer places can be handled. The arithmetic system most commonly used, in which all numbers are expressed in a format that has these characteristics, is called "floating point arithmetic."

The notation used in floating point arithmetic is basically an adaptation of the scientific notation that is widely used today. In scientific work very large or very

small numbers are expressed as a number, between one and ten, times a power of ten. Thus,

427.93456 is written as 4.2793456×10^2 ,

and 0.0009762 is written as 9.762×10^{-4}

In the 1620 floating point arithmetic system, the range of the fractional part of the number is modified to extend between .10 000 and .99 999, that is, the decimal point of all numbers is placed to the left of the high-order (leftmost) nonzero digit. Hence, all quantities may be thought of as a decimal fraction times a power of ten. For example,

427.93456 becomes $.42793456 \times 10^3$

and 0.0009762 becomes $.97620000 \times 10^{-3}$

where the fraction is called the *mantissa*, and the power of ten, indicating the number of places the decimal point was shifted, is called the *exponent*. The use of floating point numbers during processing, besides offering advantages inherent in scientific notation, eliminates the need for analyzing operations in order to determine the positioning of the decimal point in intermediate and final results, since the decimal point is always immediately to the left of the high-order, nonzero digit in the mantissa.

In 1620 floating point operations, a floating point number is a field consisting of a variable length or fixed length mantissa and a 2-digit exponent. The exponent is in the two low-order positions of the field, and the mantissa is in the remaining high-order positions, as shown:

$\bar{M} M\bar{E}\bar{E}$

For the subroutines, the variable length mantissa may have a minimum of two digits and a maximum of 45 digits. Two operand fields that are added together must have mantissas of the same length. A flag over the high-order digit marks the extremity of the field. A fixed length mantissa must have eight digits.

The exponent is established on the premise that the mantissa is less than 1.0 and equal to or greater than 0.1. It always consists of two digits ranging between -99 and +99. A flag over the high-order (tens) digit defines the exponent.

The high-order digit of the mantissa and the high-order digit of the exponent *must* contain flag bits to operate properly with floating point subroutines.

The mantissa and the exponent, if negative, must have an algebraic sign, represented by a flag, over the units position of the respective fields; if they are positive, they are not flagged. A floating point number with a negative mantissa and a negative exponent is represented as follows:

$\bar{M} M\bar{E}\bar{E}$

Sign control of the results of all computations is maintained according to the standard rules of arithmetic operations.

In all floating point numbers the decimal point is assumed to be at the left of the high-order digit, which must be a nonzero digit. Such a number is referred to as *normalized*. When a number has one or more high-order zeros, it is considered to be *unnormalized*. An unnormalized number resulting from a floating point subroutine computation is normalized automatically, but unnormalized terms are not recognized as such when entered as data. Therefore, it is necessary for all data to be entered in normalized form. Although unnormalized numbers will be processed, correct results cannot be assured. For example, the number $\bar{0}6823494\bar{0}5$ should be entered as $\bar{6}823494\bar{0}4$, assuming the fixed point number is 6823.494, and an 8-digit mantissa is required.

The floating examples demonstrate the conversion of numbers in ordinary form to 1620 floating point notation for an 8-digit mantissa.

1620 FLOATING		
NUMBER	NORMALIZED	POINT
123.45678	$.12345678 \times 10^3$	$\bar{1}2345678\bar{0}3$
.00765438	$.76543800 \times 10^{-2}$	$\bar{7}6543800\bar{0}2$
-.12348693	$-.12348693 \times 10^0$	$\bar{1}234869300$
-.00000070	$-.70000000 \times 10^{-6}$	$\bar{7}000000006$
.00000000	$.00000000 \times 10^{-99}$	$\bar{0}000000099$

NOTE: A zero mantissa is associated with a $\bar{9}\bar{9}$ exponent.

The result of a floating point operation is normalized automatically. For example, the result .00123456 when normalized becomes $\bar{1}23456NN\bar{0}2$, where N is an inserted digit and $\bar{0}2$ is the exponent. The value of the N digit (0 through 9) is determined by the programmer, who in most cases will choose to use zero. The storage location of the N digit is the first odd-numbered location following the last secondary linkage of the assembled program. *The programmer must always store the N digit.* He may do this by using the following statements, where the constant (N digit) is zero.

Line	Label	Operation	Operands & Remarks
3	5 4		
		11 12 13 14 15 16 20 25 30 35 40 45 50	
0.1.6		DAC 1,0	
0.2.9		DEND	

In normalizing, certain low-order digits in a mantissa may lose significance. To recognize these digits, the floating point arithmetic can be performed twice, using a different N digit for each run, e.g., zero for the first run and nine for the second run. The significance of these digits can be readily distinguished by comparing the

two results. For example, if the programmer compares the following:

	Mantissa	Exponent
Result, 1st run	.12345000	04
Result, 2nd run	.12345099	04

he will see that the two low-order positions of the mantissa have lost significance because they are significantly different.

When intermediate floating point results enter into additional floating point calculations, inserted digits may become a part of the result of the additional calculation.

In the case of lengthy computations using floating point results, precision gradually decreases because of truncation. The magnitude of the truncation error depends on the individual computation process and cannot be predicted without a knowledge of the process in question. However, the truncation error in such cases is usually no greater than the degree of error present in a rounded amount. Results in floating point subroutine are not rounded. The maximum truncation error for a fixed length mantissa will not exceed 10^{-8} or for a variable length mantissa, 10^{-L} , except under certain conditions described in the explanation of floating point functional subroutines.

Exponent Overflow and Underflow

In the 1620 floating point subroutines, numbers with a magnitude equal to or greater than 10^{99} create a condition called *exponent overflow*; those with a magnitude of less than 10^{-99} create a condition called *exponent underflow*. If either of these conditions is generated as a result of an arithmetic operation, the programmer has two options.

Overflow

1. To halt the program or
2. To cause $\bar{9} \dots \bar{9}99$ to be placed in the result field and to continue executing the subroutine.

Underflow

1. To halt the program or
2. To cause $\bar{0} \dots \bar{0}99$ to be placed in the result field and to continue executing the subroutine.

The options function independently of each other. Therefore, it is possible to halt on an overflow and place zeros in the result field on an underflow, or to halt on an underflow, and place nines in the result field on an overflow.

The detection of an overflow or underflow condition causes the subroutine being executed to examine core storage position 00401 to determine the course of action. Options available to the programmer must be represented in position 00401 by one of the following characters.

		Halt	UNDERFLOW Store Zeros in Result Field
O	V	Halt	0
E	R		
F	L	Store All Nines in Result Field	1
O	W		

To store the code determining the option in 00401, an unlabeled Define Constant (DC) statement may be used, as shown in the following example:

Line	Label	Operation	Operands & Remarks
3 5 6		DC	2. -0.401 HALT ON OFLOW OR UNDERFLOW
0 1 0			

Positive codes (0 or 1) need not be preceded by a plus sign.

Overflow and/or underflow conditions can only arise in six of the floating point subroutines presented in this manual; namely, the four arithmetic subroutines and the two exponential functional subroutines.

If the subroutine halts on an overflow or underflow condition, the operator can continue processing by depressing the Start key on the console. In the case of an overflow, execution of the subroutine begins after $\bar{9} \dots \bar{9}99$ is placed in the result field; in the case of an underflow, after $\bar{0} \dots \bar{0}99$ is placed in the result field.

Description of 1620 Subroutines

In this section, the various subroutines are described and examples are given to show how the associated macro-instructions are written. Table 10 shows the storage requirements for each subroutine.

During execution of the *arithmetic* subroutines, the overflow, high/positive, and equal/zero indicators are used. The overflow indicator is always reset at the beginning of each arithmetic subroutine. If it is desired to determine its status prior to the execution of an arithmetic subroutine, the indicator must be tested and its condition stored before the linkage instructions are executed. The high/positive and equal/zero indicators are set according to the mantissa of the result. Whenever a zero mantissa results ($\bar{0} \dots \bar{0}99$), the equal/zero indicator is turned on.

Table 10. Subroutine Storage Requirements and Identification Data

SUBROUTINE	NUMBER OF STORAGE POSITIONS REQUIRED				
	FIXED LENGTH		VARIABLE LENGTH		
	WITHOUT AUTOMATIC DIVIDE	WITH AUTOMATIC DIVIDE	WITHOUT AUTOMATIC DIVIDE	WITH AUTOMATIC DIVIDE	WITH AUTOMATIC FLOATING POINT
PICK	872	872	1136	1136	896
DIV	1047	187	1035	199	199
FA	543	543	1207	1163	639
FS					
FM	239	239	659	659	659
FD	523	335			
FSQR	579	579	1098	1054	1054
FCOS	867	867			
FSIN		1487	1379	1379	
FATN	1077				989
FEXT	820	776	1258	1118	1118
FEX					
FLOG	886	842	1209	1145	1145
FLN					
FSRS	279	279	279	279	96
FSLS	372	372	372	372	96
TFLS	31	31	31	31	31
BTFS	79	79	79	79	43
Identification Number (Col. 77 of card deck)	2	3	4	5	6

At the conclusion of a functional subroutine, the status of the high/positive, equal/zero, and overflow indicators does not necessarily reflect the result of the operation, because the indicators are disturbed during the execution of a functional subroutine. Therefore, their status at the conclusion of a functional subroutine should not be assumed to be the same as it was prior to the execution of the subroutine.

Pick

This subroutine is common to all fixed length and variable length mantissa subroutines. The pick subroutine, during execution of the object program:

1. Sets up A and B operands (more, if designated) to be operated upon, calculates the return address to the mainline program, and branches to the subroutine.
2. Stores the calculated result in the proper storage area and branches back to the mainline program.
3. Initiates typing of error messages and branches to the subroutine if the error condition allows processing of the subroutine to be resumed.
4. Provides constants and working storage for the other subroutines.

The average execution time for the pick subroutine can be determined by the formula:

$$\text{Average time (in } \mu\text{s}) = 100L + 8320$$

where L equals the length of the mantissa and the numbers are expressed in microseconds. Therefore, an 8-digit mantissa (same as fixed length mantissa) requires 9120 μ s.

$$100 \times 8 = 800$$

$$\underline{8320}$$

$$9120 (\mu\text{s})$$

or approximately 9 milliseconds (ms). If indirect addressing is used, the average time is increased according to the number of levels of indirect addressing used.

NOTE: For the variable length subroutines used with the automatic floating point feature,

$$\text{Average time (in } \mu\text{s}) = 100L + 4500$$

The above timings apply to object programs being executed on the 1620 Model 1. The 1620 Model 2 timings for the Pick subroutine are:

Fixed length and variable length

$$\text{Average time (in } \mu\text{s}) = 318L + 2470$$

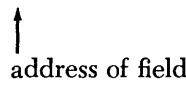
Floating Add

Macro-instruction

Line	Label	Operation	20	25	30	35	40	45	50	Operands & Remarks
3 5 6		11 12 15 16								
9 1 0		F A	A, B							

The A and B addresses refer to the units position of the exponents of the fields:

MMMMMMMMEE



where Ms represent digits of the mantissa and Es represent digits of the exponent.

Operation

Field B is added to field A. The floating point sum replaces field A; field B remains unchanged.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 9 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 5L^2 + 482L + 6854$$

where L = length of mantissa

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 4,100 \mu\text{s}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 70L + 3420$$

Floating Subtract

Macro-instruction

Line	Label	Operation	20	25	30	35	40	45	50	Operands & Remarks
3 5 6		11 12 15 16								
9 1 0		F S	A, B							

The A and B addresses refer to the units position of the exponents of the fields:

Operation

Field B is subtracted from field A. The floating point difference replaces field A; field B remains unchanged.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 10.5 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 5L^2 + 482L + 7474$$

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 4,200 \mu\text{s}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 70L + 3500$$

Floating Multiply

Macro-instruction

Line	Label	Operation	20	25	30	35	40	45	50	Operands & Remarks
3 5 6		11 12 15 16								
9 1 0		F M	A, B							

The A and B addresses refer to the units position of the exponents of the fields:

Operation

Field A is multiplied by field B. The floating point product replaces field A; field B remains unchanged.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 18 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 168L^2 + 240L + 7400$$

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 5,300 \mu\text{s}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 36.6L^2 + 48L + 3420$$

Floating Divide

Macro-instruction

Line	Label	Operation	20	25	30	35	40	45	50	Operands & Remarks
3 5 6		FD A, B								

Operation

Field A is divided by field B. The floating point quotient replaces field A; field B remains unchanged.

Average Execution Time (1620 Model 1)

Fixed length

With automatic divide

Average time = 55 ms

Without automatic divide

Average time = 70 ms

Variable length

With automatic divide

Average time (in μ s) = $520L^2 + 1500L + 7890$

Without automatic divide

Average time (in μ s) = $1.9(520L^2 + 1500L + 7890)$

Average Execution Time (1620 Model 2)

Fixed length

Average time = 10,900 μ s

Variable length

Average time (in μ s) = $98.5L^2 + 200L + 3490$

Fixed Point Divide

Macro-instruction

Line	Label	Operation	20	25	30	35	40	45	50	Operands & Remarks
3 5 6		D/V A, B, A1, B1								

In addition to the A and B operands, representing the addresses of the dividend and divisor, the divide macro-instruction requires two additional operands; one specifies the number of zeros to be inserted to the right of the dividend (A1 operand) and the other, the shift factor needed by the subroutine (B1 operand). Specifically, the

A operand is the core storage address of the dividend.

B operand is the core storage address of the divisor.
A1 operand is 00099 minus the number of zeros desired to the right of the units position of the dividend.

B1 operand is 00100 minus the length of the quotient. The quotient must be at least two digits in length.

NOTE: The quotient address after the division is executed will be equal to 00099 minus the length of the divisor.

Prior to the divide operation, the divide subroutine always resets to zeros (clears) the positions 00080 through 00099, the product area where the 20-digit quotient and remainder are developed. For the variable length mantissa subroutines, where L (length of mantissa) is greater than 10, the number of positions which are reset to zeros is equal to $99 - 2L$. When the quotient plus the remainder exceeds the number of positions cleared to zeros, positions lower than the last position cleared must be reset to zeros by *programming*. One additional position should also be cleared to allow for a possible overdraw. For example, if 25 positions are required for the quotient and remainder in a fixed length mantissa subroutine, 00074-00079 will have to be reset to zeros before the divide macro-instruction is given.

The fixed point divide macro-instruction may be used with any of the subroutine packages. Whenever it is used, the fixed point divide subroutine will be incorporated into the user's program. For the subroutine packages that are designed to work with automatic divide, the fixed point divide subroutine uses automatic divide in performing its operation. For the subroutine packages that are designed to work without the automatic divide feature, the fixed point divide subroutine performs its operation as instructed by the routine without the aid of the automatic divide feature. Coding of the macro-instructions is the same for all of the subroutine packages.

Operation

The product area (00080-00099) is automatically reset to zeros. The dividend (A address) is transmitted to the product area (A1 address), beginning at the low-order dividend digit and terminating at the flag bit marking the high-order position of the dividend field. The A1 address is 00099 minus the number of zero positions desired to the right of the dividend.

The algebraic sign of the dividend is automatically placed in location 00099, regardless of where the right-most dividend digit is placed by the A1 address. A flag bit automatically marks the high-order digit of the dividend.

The divisor (B address) is successively subtracted from the dividend. The B1 address of the divide macro-instruction positions the divisor for the first subtraction from the high-order position(s) of the dividend, as in manual division. The B1 address is determined by subtracting the number of digits in the quotient from 100. For the subroutines using program divide, the value of B1 must be between 0 and 99. For subroutines using automatic divide, the value of B1 is not restricted.

The quotient and remainder replace the dividend in the product area. The address of the quotient is 00099 minus the length of the divisor. The algebraic sign of the quotient (determined by the signs of the dividend and divisor) is automatically placed in the low-order position of the quotient. The address of the remainder is 00099 and a flag bit is automatically placed in that position. The remainder has the sign of the dividend and the same number of digits as the divisor.

The high/positive indicator is on if the quotient is positive and not zero; the equal/zero indicator is on if the quotient is zero. Neither indicator is on if the quotient is negative.

The quotient must be at least two digits in length. One position is required for the sign and one for the field mark (flag bit).

Examples

1. The macro-instruction

DIV A, B, 99, 96

will perform the division for $\overline{0273} \overline{)3972}$ and store the result $\overline{0014}$ in storage locations 00092 through 00095.

2. The macro-instruction

DIV A, B, 96, 93

will perform the division for $\overline{0273} \overline{)3972.000}$ and store the result $\overline{0014.549}$ in storage locations 00089 through 00095.

NOTE: In both examples (1 and 2), A represents the address of the dividend 3972 and B represents the address of the divisor 0273.

Incorrect Positioning of Divisor

The following error conditions are caused by an incorrect B1 address.

An incorrectly positioned divisor can cause more than nine successful subtractions and an incorrect quotient. The divide operation is terminated, the Overflow indicator and Overflow Arithmetic Check light are turned on, but processing will not stop unless the Overflow Check switch is set to STOP.

The high-order digit of the dividend is assumed by the 1620 to be one position to the left of the high-order digit of the divisor. The high-order digits of the dividend are lost if the divisor is positioned too far to the right. Processing continues with no indication of an incorrect quotient.

If the B address is less than 10000, i.e., between 00100 and 09999, the divide operation will terminate when a subtraction occurs at 0XX99. This, in effect, restricts the size of the dividend to 10,020 digits if only 20,000 positions of core storage are installed.

Average Execution Time (1620 Model 1)

Fixed length and variable length with automatic divide

$$\text{Average time (in ms)} = 980 + .040 \text{ LDVD} + (.520 \text{ LDVR} + .740) (100 - \text{B1})$$

where LDVD is length of the dividend field,
LDVR is length of the divisor field, and
B1 is value specified in the macro-instruction.

NOTE: Multiply 3.2 times the result to find the average execution time for the fixed length and variable length subroutines without automatic divide.

Average Execution Time (1620 Model 2)

Fixed length and variable length without automatic divide

$$\text{Average time (in ms)} = 8.265 + .015 \text{ LDVD} + (.465 + .027 \text{ LDVR}) (100 - \text{B1})$$

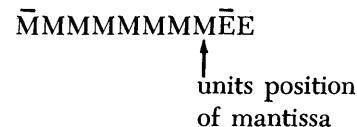
where LDVD is length of the dividend field,
LDVR is length of the divisor field,
and B1 is the value specified in the macro-instruction.

Floating Shift Right

Macro-instruction

Line	Label	Operation	Operands & Remarks									
3	5	4	11	12	13	14	20	25	30	35	40	45
0	1	0	F	S	R	S	A	.	B			

The effect of this macro-instruction is to shrink the mantissa by shifting it to the right and truncating the low-order digits. The A address is normally the units position of the mantissa.



The B address specifies the digit of the mantissa which will become the low-order digit of the mantissa.

Operation

The field at the B address (the portion of the mantissa to be retained) is shifted right to the location specified

by the A address. The exponent is not moved or altered. For example, the macro-instruction

FSRS 00097,00093

causes the mantissa

$\overline{3}0590011325\overline{7}01$
↑ ↑
Storage Storage
Address Address
00093 00097

to be shifted, producing the following result

$0000\overline{3}059001\overline{1}01$
↑ ↑
Storage Storage
Address Address
00093 00097

Vacated high-order positions are set to zeros. An existing flag at the A address is retained for algebraic sign; the field flag bit is transmitted with the high-order digit of the B field.

Average Execution Time (1620 Model 1)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s)} = 4960 + 960L - 880(A - B)$$

Average Execution Time (1620 Model 2)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s)} = 2270 + 45(A - B)$$

Floating Shift Left

Macro-instruction

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0	1	0											

FSLS A, B

The effect of this macro-instruction is to expand the mantissa by shifting it to the left and filling the vacated positions with zeros. It is important to note that the B address is the low-order position of the field moved, and the A address is the high-order position of the resulting field.

Operation

The field at the B address, which is the *low order* digit of the mantissa, is shifted left so that the *high order* digit is moved to the location specified by the A address.

The exponent is not moved or altered. For example, the macro-instruction:

FSLS 00090, 00097

causes the mantissa

$00\overline{1}1325\overline{7}01$
↑ ↑
Storage Storage
Address Address
00090 00097

to be shifted, producing the following result

$\overline{1}1325700\overline{0}1$
↑ ↑
Storage Storage
Address Address
00090 00097

An existing flag bit at the B address is retained for algebraic sign; the field flag bit is transmitted with the high-order digit of the B field.

Average Execution Time (1620 Model 1)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s)} = 6460 + 1520(B - A) - 360L$$

Average Execution Time (1620 Model 2)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s)} = 3830 + 350(B - A) - 7.5(B - A)^2$$

Transmit Floating

Macro-instruction

Line	Label	Operation	Operands & Remarks										
3	5	6	11	12	15	16	20	25	30	35	40	45	50
0	1	0											

TFLS A, B

The B address refers to the low-order digit of the floating point field exponent, whereas the A address refers to the low-order position to which the field is transmitted.

Operation

The field at the B address is transmitted to the location specified by the A address. The B field remains unchanged in storage. Flag bits in the three low-order positions of the B field are also transmitted; starting with the fourth low-order position, only one additional flag bit is transmitted, and it stops transmission. For the

variable length subroutine, L must be ≤ 49 , where L equals the number of mantissa digits in the field to be transmitted. For the fixed length subroutine, L must be ≤ 19 .

Average Execution Time (1620 Model 1)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s}) = 400 + 40L$$

Average Execution Time (1620 Model 2)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s}) = 530 + 15L$$

Branch and Transmit Floating

Macro-instruction

Line	Label	Operation	Operands & Remarks									
3	5 4	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		BTFS A, B										

The B address is normally the low-order position of the floating point field exponent, whereas the A address is the leftmost position of the next instruction to be executed.

Operation

The address of the next instruction is saved at a storage location equivalent to BTFS + 78 and the field at the B address is transmitted to the A address minus one. The normal exit of a routine which is entered by a BTFS is a Branch Back (BB) instruction. The instruction at the A address is the next one executed. The B field remains unchanged in core storage. Any flag bits in the three low-order positions of the B field are transmitted; starting with the fourth low-order position, only one additional flag bit is transmitted, and it stops transmission.

Average Execution Time (1620 Model 1)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s}) = 2280 + 40L$$

Average Execution Time (1620 Model 2)

Fixed length and variable length

$$\text{Average time (in } \mu\text{s}) = 645 + 15L$$

Floating Square Root

Macro-instruction

Line	Label	Operation	Operands & Remarks									
3	5 4	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		FSQR A, B										

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The square root of argument B is extracted and the result, in floating point form, is stored at A. The argument, which must be in floating point form, is unchanged by the operation.

The floating point square root subroutine accepts all numbers within the floating point range that are greater than or equal to zero. If the argument is less than zero, the subroutine executes a programmed halt. The operator has two options:

1. Using the information found in the halt instruction, he may branch back to the main routine, or
2. Continue the execution of the subroutine and compute the square root of B.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 120 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 620L^2 + 9776L + 5328$$

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 29 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 100L^2 + 2000L + 5500$$

Floating Sine

Macro-instruction

Line	Label	Operation	Operands & Remarks									
3	5 4	11 12 15 16	20	25	30	35	40	45	50			
0.1.0		FSINA, B										

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The sine of argument B is computed and the result, in floating point form, is stored at A. The argument must be in radians and in floating point form. The computation does not disturb the original value of the argument.

The floating point sine subroutine accepts all numbers of floating range up to and including exponent 08 (fixed length mantissa) or L (variable length mantissa). The operator may branch back to the mainline program as explained under SUBROUTINE ERROR MESSAGES.

For arguments with exponents less than 03, the magnitude of the maximum truncation error in the mantissa of the result does not exceed 10^{-L} . Accuracy in the man-

tissa of the result decreases as the size of the argument (exponent of $\bar{0}3$ or greater) increases. The operator has the option of branching back to the main program or proceeding with the computation. Any result so computed will contain an error that varies directly with the magnitude of the exponent.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 150 \text{ ms}$$

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s)} = 168L^3 + 3792L^2 + 13340L + 4708$$

Without automatic divide

$$\text{Average time (in } \mu\text{s)} = 1.9 (168L^3 + 3792L^2 + 13340L + 4708)$$

NOTE: For all Floating Sine and Floating Cosine subroutines, arguments greater than 2π are reduced by subtractions of 2π until within range. Therefore, the time required to perform these subtractions should be added to the average time required for an argument less than 2π .

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 33.3 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s)} = 5L^3 + 320L^2 + 3100L + 4900$$

Floating Cosine

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 3 4	11 12 13 14	FATNA, B	20 25 30 35 40 45 50
9 1 0			

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The cosine of argument B is computed and the result, in floating point form, is stored at A. The argument must be in radians and in floating point form. The computation does not disturb the original value of the argument.

The allowable range of the argument, maximum accuracy, etc., for the cosine subroutine are the same as those previously described for the sine subroutine.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 155 \text{ ms}$$

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s)} = 168L^3 + 3792L^2 + 13420L + 5228$$

Without automatic divide

$$\text{Average time (in } \mu\text{s)} = 1.9 (168L^3 + 3792L^2 + 13420L + 5228)$$

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 33.3 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s)} = 5L^3 + 296L^2 + 2950L + 5400$$

Floating Arctangent

Macro-instruction

Line	Label	Operation	Operands & Remarks
3 5 6	11 12 13 14	FATNA, B	20 25 30 35 40 45 50
9 1 0			

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The floating point value of the arctangent of B is computed and the result stored at A. The argument must be in floating point form; the result in radians will be in floating point form.

The arctangent subroutine accepts any number within the floating point range.

During the evaluation of the arctangent of B, use will be made of the divide subroutine.

The maximum truncation error in the mantissa of the result is $\pm 10^{-L}$, except for results having an exponent less than or equal to $\bar{0}2$ ($E \leq \bar{0}2$). The maximum error for these results is ± 1 in the $(L + 1)$ th decimal place. L = 08 for the fixed length mantissa.

Average Execution Time (1620 Model 1)

Fixed length

$$\text{Average time} = 260 \text{ ms}$$

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s)} = 168L^3 + 2996L^2 + 7792L + 7260$$

Without automatic divide

$$\text{Average time (in } \mu\text{s)} = 1.9 (168L^3 + 2996L^2 + 7792L + 7260)$$

Average Execution Time (1620 Model 2)

Fixed length

$$\text{Average time} = 31.7 \text{ ms}$$

Variable length

$$\text{Average time (in } \mu\text{s}) = 35L^3 + 570L^2 + 400L + 7500$$

Floating Exponential (Natural)

Macro-instruction

Line	Label	Operation	Operands & Remarks							
3	5 6	11 12 15 16	20	25	30	35	40	45	50	
0.1.0		FEX, A, B								

Operation

The A and B addresses refer to the units position of the exponents of the fields. The value of e^B , where B is in floating point form, is computed and the result, also in floating point form, is stored at A.

An input value that exceeds

227.955924206n.....n (227955924206n.....n03)

causes an exponent overflow and one which is less than

-227.955924206n.....n
(227955924206n.....n03)

causes an exponent underflow. An exponent overflow or underflow causes the subroutine to examine core storage position 401 to determine the course of action.

For negative arguments, the subroutine uses the absolute value of the argument to evaluate the function, and then finds the reciprocal value.

For positive and negative arguments, the maximum truncation error in the mantissa of the result is $\pm 10^{-1}$.

Average Execution Time (1620 Model 1)

Fixed length

Average time = 160 ms

NOTE: Add 70 to the average time if B is negative.

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s}) = 168L^3 + 35824L^2 + 15890L + 26418$$

Without automatic divide

$$\text{Average time (in } \mu\text{s}) = 1.9(168L^3 + 35824L^2 + 15890L + 26418)$$

NOTE: For a negative argument, add the result of $520L^2 + 1880L + 1480$ to the average time.

Average Execution Time (1620 Model 2)

Fixed length

Average time = 38 ms

NOTE: Add 11.4 ms if B is negative.

Variable length

$$\text{Average time (in } \mu\text{s}) = 21L^3 + 240L^2 + 6000L - 1300$$

NOTE: Add time for VL Divide if B is negative.

Floating Exponential (Base 10)

Macro-instruction

Line	Label	Operation	Operands & Remarks							
3	5 6	11 12 15 16	20	25	30	35	40	45	50	
0.1.0		FEX, A, B								

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The value of 10^B , where B is in floating point form, is computed and the result, also in floating point form, is stored at A.

An input value which exceeds 98.9n.....n (989n.....n02) causes an exponent overflow and one which is less than -98.9n.....n (989n.....n02) causes an exponent underflow. An exponent overflow or underflow causes the subroutine to examine core storage position 401 to determine the next course of action.

This subroutine handles negative arguments in the manner they are handled by the natural exponential subroutine. Maximum accuracy is the same.

Average Execution Time (1620 Model 1)

Fixed length

Average time = 145 ms

NOTE: Add 70 ms to the average time if B is negative

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s}) = 168L^3 + 3656L^2 + 15414L + 24538$$

Without automatic divide

$$\text{Average time (in } \mu\text{s}) = 1.9(168L^3 + 3656L^2 + 15414L + 24538)$$

NOTE: For a negative argument, add the result of $520L^2 + 1889L + 1480$ to the average time.

Average Execution Time (1620 Model 2)

Fixed length

Average time = 39.8 ms

NOTE: Add 11.4 ms if B is negative.

Variable length

$$\text{Average time (in } \mu\text{s}) = 23L^3 + 240L^2 + 6050L - 1300$$

NOTE: Add time for VL Divide if B is negative.

Floating Logarithm (Natural)

Macro-instruction

Line	Label	Operation	Operands & Remarks							
3 5 6		FLN	A	, B						

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The floating point value of the $\ln B$ is computed and stored at A. Input arguments must be in floating point form.

This subroutine accepts all arguments greater than zero within the floating point range. An input argument equal to or less than zero results in a programmed halt. A branch back to the main program can be effected as described under SUBROUTINE ERROR MESSAGES.

Average Execution Time (1620 Model 1)

Fixed length

Average time = 290 ms

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s) } = 168L^3 + 3440L^2 + 10530L + 12180$$

Without automatic divide

$$\text{Average time (in } \mu\text{s) } = 1.9(168L^3 + 3440L^2 + 10530L + 12180)$$

Average Execution Time (1620 Model 2)

Fixed length

Average time = 51.7 ms

Variable length

$$\text{Average time (in } \mu\text{s) } = 36.5L^3 + 590L^2 + 1500L + 8600$$

Floating Logarithm (Base 10)

Macro-instruction

Line	Label	Operation	Operands & Remarks							
3 5 6		FLDG	A	, B						

The A and B addresses refer to the units position of the exponents of the fields.

Operation

The floating point value of the $\log_{10}B$ is computed and stored at A. Input arguments must be in floating point form.

This subroutine accepts all arguments greater than zero within the floating point range. An input argument equal to or less than zero results in a programmed halt. A branch back to the main program can be effected as described under SUBROUTINE ERROR MESSAGES.

Average Execution Time (1620 Model 1)

Fixed length

Average time = 305 ms

Variable length

With automatic divide

$$\text{Average time (in } \mu\text{s) } = 168L^3 + 3608L^2 + 11610L + 15108$$

Without automatic divide

$$\text{Average time (in } \mu\text{s) } = 1.9(168L^3 + 3608L^2 + 11610L + 15108)$$

Average Execution Time (1620 Model 2)

Fixed length

Average time = 56.6 ms

Variable length

$$\text{Average time} = 33.5L^3 + 680L^2 + 2100L + 5900$$

Subroutine Error Messages

In 1620 subroutines the presence of special conditions causes an error message. This message is typed out in the following form:

XXXXX00XX
R S

where R is a return address to the main program and S is a code that identifies the special condition.

With the exception of exponent overflow or underflow, where the course of action depends upon the digit at location 00401, a subroutine always halts immediately after typing the error message. The error message code indicates the reason for the halt. The operator may insert a branch instruction at storage location 00000. The branch instruction will contain the return address to the main program. In cases such as floating square root, execution of the subroutine may be made to continue, after a halt, by depressing the start key.

Table 11 lists the error codes for special conditions and the appropriate action to be taken.

Table 11. Subroutine Error Codes

ERROR CODE	DESCRIPTION OF ERROR	OPERATOR'S ACTION WHEN SUBROUTINE HALTS
01	FA or FS, Exponent Overflow	May continue execution of subroutine by depressing start key
02	FA or FS, Exponent Underflow	Same as code 01
03	FM, Exponent Overflow	Same as code 01
04	FM, Exponent Underflow	Same as code 01
05	FD, Exponent Overflow	Same as code 01
06	FD, Exponent Underflow	Same as code 01
07	FD, Attempt to divide using a number with a zero mantissa divisor	May not continue execution of subroutine but may branch back to main program using return address
08	FSQR, Attempt to find the square root of a negative number	Pressing the start key causes the subroutine to extract the square root of the absolute value of the argument
09	FSIN or FCOS, Input argument has an exponent value greater than $\bar{0}8$ (fixed length mantissa) or L (variable-length mantissa)	Same as code 07
10	FSIN or FCOS, For a fixed-length mantissa, the input argument has an exponent (X) such that $\bar{0}3 \leq X \leq \bar{0}8$. For a variable-length mantissa, the input argument has an exponent (X) such that $\bar{0}3 \leq X \leq L$ where L = length of a mantissa.	Same as code 01
11	FEX or FEXT, Exponent Overflow	Same as code 01
12	FEX or FEXT, Exponent Underflow	Same as code 01
13	FLN or FLOG, Input argument has a zero mantissa	Same as code 07
14	FLN or FLOG, Input argument is negative	Pressing the start key causes the subroutine to continue execution, using the absolute value of the argument

Adding Subroutines

Up to 12 subroutines may be added to any of the card or paper tape subroutine packages.

In brief, the steps required to add a subroutine are:

1. Write the subroutine and assemble it in condensed form.
2. Add the new subroutine to the card or tape package.
3. Add the macro-instruction mnemonic to the SPS processor.

The specific details of adding a subroutine or its macro-instruction are covered in ADDING A SUBROUTINE TO A CARD DECK OR ADDING A SUBROUTINE TO TAPE.

Adding a Subroutine to a Card Deck

After a subroutine is written, it must be assembled in condensed form. Then, discard the first two loader cards and the last seven table cards of the object deck.

(Discard the last seven cards if assembly was made for a Model 1; discard the last 6 cards if assembly was made for a Model 2.) These cards are replaced by a subroutine header card and a subroutine trailer card. The formats for the subroutine header and trailer cards are described below. The new subroutine, together with its header and trailer cards, is inserted into the subroutine deck between the trailer card of the last subroutine presently in the deck and the "end of deck" card (identified by a record mark (#) in column 76).

Header Card Format

Columns 1-4 Length of subroutine - 4 digits in the form $\bar{xx}xx$.

5-13 Subroutine numbers (each in two digits \bar{xx}) followed by a record mark; a subroutine may have up to four entry points, with each entry represented by a unique mnemonic.

- 14-23 Number of storage positions (three digits \bar{xxx}), between the second (third and fourth) entrance(s) and the regular entrance of the subroutine. These 3-digit fields must be terminated by a record mark (for example, $\bar{1}20\bar{1}86\#$). If the subroutine has only one entrance, a record mark should be punched in column 14.
- 24 0 (zero) for subroutine deck without automatic divide; 1 for subroutine deck with automatic divide; \neq for variable length subroutine decks with divide or with automatic floating point; $\bar{0}$ for variable length subroutines without automatic divide.
- 25-43 The secondary linkage in machine language. The linkage contains two instructions, the second of which is always a branch (operation code 49). The operation code in the first instruction and the three addresses can be specified by the programmer. The breakdown of these columns is as follows:
- 25-26 Two-digit numerical operation code for the first instruction. Modification to the P and Q addresses is indicated by a flag or no flag on the first and second digit, respectively. A flag implies that the address is relative to the subroutine itself while no flag means it is relative to the Pick subroutine.
- 27-31 P address of the first instruction, expressed as an increment to PICK or the subroutine. For example, PICK + 23 will be 00023 and SUBR1 + 59 will be 00059.
- 32-36 Q address of the first instruction, expressed as an increment to PICK or the subroutine. For example,
- PICK + 23 will be 00023 and SUBR1 + 59 will be 00059.
- 37-38 Operation code 49; a flag or no flag on digit 4 indicates modification to the P address with respect to the subroutine or PICK.
- 39-43 P address expressed as an increment to PICK or the subroutine.
- 44-75 Blanks.
- 76 0 (zero).
- 77-80 Sequence number.

Trailer Card Format

Column 76 1
 77-78 Subroutine number.
 79-80 Sequence number.

Sample Problem Illustrating Header Card, Subroutine, and Trailer Card

In this example the subroutine is to be inserted in the variable length subroutine deck without automatic divide. The Floating Branch and Transmit subroutine (BTFS) is used as the new subroutine; thus it is assumed that the subroutine deck contains no BTFS subroutine. This example shows the header card coding, the modifier constants, the O_0 and O_1 flag indicators associated with the new subroutine, and the trailer card coding. For each field of the header and trailer cards, the data contained in the field as well as a description of the data is given.

HEADER CARD

Columns 1-4 : $\bar{0}079$	Program requires 79 storage positions.
5-13: $\bar{1}7\neq$	Subroutine identifying number.
14-23: \neq	Subroutine has only one entry point.
24: $\bar{0}$	Variable length subroutine without automatic divide.
25-43: $\bar{1}6$ 00402	00000 49 00104 These instructions correspond to the secondary linkage: TFM PICK + 402, ADDR B PICK + 104

The P operands of the TFM and B will be modified by adding the address of PICK when it is found. The Q operand of the TFM will be modified by adding the starting address of the BTFS subroutine to it.

44-75: Blanks
76: 0
77-80: 4000

The 4 is the identifying number of the subroutine set. The 000 is the card sequence number.

SUBROUTINE

DORG 5000

Standard DORG statement for all subroutines.

DC 14, 05000005050500, 314

This statement provides the modifier digits for the seven instructions which follow.

BTFS1 TF * + 66, STORE + 6, 0

A flag over O₀ indicates to the subroutine processor that the P operand must be modified with respect to the relocated addresses of the BTFS subroutine. With respect to PICK, the Q operand is modified by the second digit of the pseudo constant.

TF * + 30, * + 54, 01

The P and Q operands need only be modified with respect to the BTFS subroutine. Therefore, the only modification required is flagging O₀ and O₁.

SM * + 18, 3, 010

* + 18 is modified with respect to its own subroutine. The Q operand needs no modification since O₃ is wanted. The flag on Q₁₀ is needed in the computation.

TF , BETA - 2

With respect to PICK, the Q operand is modified by the eighth digit of the pseudo constant.

TF * + 30, STORE +30, 0

O₀ flagged to modify * + 30 with respect to BTFS subroutine. Q operand was previously modified. With respect to PICK, the Q operand is modified by the tenth digit of the pseudo constant.

BT , BETA

With respect to PICK, the Q operand is modified by the twelfth digit of the pseudo constant.

B

No modification.

DEND

No modification.

TRAILER CARD
Column 76 1

Library Change Card

For each macro-instruction added, a library change card must be prepared for insertion in the processor deck. These cards must be inserted immediately in front of the last nine cards of the processor deck.

The programmer must assign a unique mnemonic operation code to each macro-instruction. New macro-instruction mnemonics must be four alphabetic characters in length.

The format of the library change card is as follows:

Columns 1-8	Mnemonic operation code in 2-digit form (column 1 must be flagged, columns 2 to 8 must not be flagged).
9-10	subroutine number.
11	7.
12	record mark (0, 2, 8 punches).
13-62	blanks.
63-64	01 (column 63 must be flagged).
65-69	address of leftmost location where data from columns 1-11 is to be stored. Column 65 must be flagged.
70-74	address plus 1 of rightmost location where data from columns 1-11 is to be stored. Column 70 must be flagged.
75	blank.
76	-(flag only).

The address in columns 65-69 for subroutine 18 can be found by referring to the program listing (symbol XDEND+12). This number must be increased by 11 for each subsequent subroutine. The address in columns 70-74 is 11 greater than the address placed in columns 65-69 for subroutine 18. This address must also be increased by 11 for each subsequent subroutine being added.

Adding a Subroutine to Tape

A tape modifier program is included in the SPS III Programming System. Its purpose is to allow the addition and/or deletion of subroutines and subroutine macro-instructions to or from the subroutine tape.

Procedure

A brief summary of the procedures to be followed when using the tape modifier program is:

1. Write the subroutine in SPS language with origin at 05000 (DORG 5000).
2. Assemble the subroutine.
3. Prepare the subroutine header data so that it may be entered at the keyboard when called for by the tape modifier program.
4. Load the tape modifier program.
5. Produce a new SPS III processor as directed by the typed messages.
6. Produce a new subroutine tape as directed by the typed messages.

Program Switches

The following switch information is typed out after the tape modifier program has been loaded.

SSW1 – ON TO ADD SUBROUTINES
SSW2 – ON TO DELETE SUBROUTINES
SSW3 – ON TO BYPASS SPS III PROCESSOR
MODIFICATION
SSW4 – ON TO CORRECT ENTRY ERROR SET
SWITCHES THEN DEPRESS START

Program Switches 1, 2, or 3, or all three can be turned on at this time.

Adding and Deleting Subroutine

Macro-Instruction Mnemonics

After the switches are set and the Start key is depressed, either of the following messages or both are typed, depending on the switch settings:

ENTER NEW MACROS, ONE AT A TIME
FOLLOW LAST ENTRY WITH A RECORD MARK

ENTER MACROS TO BE DELETED, ONE AT A TIME
FOLLOW LAST ENTRY WITH A RECORD MARK

NOTE: New macro-instruction mnemonics must be four alphabetic characters in length followed by the new subroutine number. Deleted mnemonics are entered as they actually appear in the present op-code table, e.g., FA for Floating Add, etc. Deleted mnemonics must also be followed by the subroutine number they represent. After each entry of an added or deleted mnemonic, a Release and Start must be executed prior to entering the next mnemonic. The last entry must be followed by a record mark, e.g., FA03#.

If Switches 1 and 2 are on, the second typeout occurs after all new macro-instruction mnemonics are entered.

If Switch 3 is on, the processor modification is bypassed and the program calls for the subroutine processor tape as described below. However, if Switch 3 is off

after entering the new or deleted macro-instruction mnemonics, the message

LOAD SPS III TAPE AND DEPRESS START

is typed. This loading results in the creation of a new SPS III tape which reflects the addition and/or deletion of the mnemonics that were typed in.

If the op-code table cannot accommodate all the new mnemonics (a maximum of 12 have already been entered), the following message is typed, followed by a list of all mnemonics that could not be entered:

OP-CODE TABLE FULL
THE FOLLOWING MACROS HAVE NOT BEEN ENTERED

If the op-code table can handle all new mnemonics, the program calls for the subroutine processor tape by the following typeout:

LOAD SUBROUTINE TAPE AND DEPRESS START

Adding and Deleting Subroutines

When the subroutine tape is loaded, a new tape is punched. If a subroutine(s) is being added, punching stops when the beginning of the "end of tape" record is reached. The program then calls for the header data pertaining to the new user-written subroutine(s). If no subroutines are being added, the modification is complete when punching stops. The header information is entered when the following message is typed.

ENTER HEADER INFORMATION FOR NEW
MACRO XXXX

where XXXX is the name of the new macro. The header data called for consists of the following:

LENGTH = Four digits in the form XXXX.

SUBROUTINE NUMBER(s) = Two digits in the form
XX#. If more than one number applies, the form
XXXX# is used. Up to four numbers (two digits each) can be used.

ENTRY POINT DIFFERENCES = Three-digit field for each additional entry point. If there is only one entry point, only a record mark is entered. For each additional entry point, the number entered is the difference between the first entry and the second or third, etc.; the form is XXXXXX#.

SECONDARY LINKAGE FORMAT = Secondary linkage in machine language.

NOTE: If a subroutine is written that has multiple mnemonics and numbers, such as FSIN and FCOS, the header data should be entered when any one of the relevant mnemonics appears in a call for header data.

After each bit of data is entered, a Release and Start allows entry of more data until all necessary information is entered. The program then calls for the object tape associated with the header data just entered. The typeout is

LOAD OBJECT TAPE FOR MACRO XXXX
AND DEPRESS START

Here, XXXX is the mnemonic of the new subroutine being added. Run out the subroutine tape from the reader using the Nonprocess Runout key. Then load the object tape and depress the Start key.

After the new subroutine has been punched into the subroutine tape along with the appropriate header and trailer data, the program calls for more header data if more subroutines are to be added. If no more are to be added, an "end of subroutines" record is punched, the message

RELOAD LIBRARY SUBROUTINE TAPE

is typed, and the machine halts. Reload the subroutine tape (from the beginning) and depress the Start key.

After the remainder of the library subroutine tape is processed, the message

MODIFICATION COMPLETE

is typed, and the machine halts.

At any time during the operation, operator entry errors can be corrected by turning on Program switch 4, depressing the R-S key, turning off Switch 4, and re-entering the data.

Writing a Subroutine

When writing a subroutine, the programmer should be aware of certain information concerning PICK, namely, the functions of PICK, PICK address equivalents, linkage, common work areas in PICK, and the means of signifying instructions that are relative to PICK.

Functions of PICK

PICK is common to all subroutines in the subroutine package, except DIV, FSRS and FSLS. Therefore, it is to the advantage of the subroutine writer to make use of PICK. The listing of the appropriate Pick subroutine (furnished with the library package) should be studied. Briefly, PICK performs the following operations. It:

1. Moves the A operand into ALPHA (exponent and mantissa).

2. Moves the B operand into BETA (exponent and mantissa).
3. Calculates the return address to the mainline program.
4. Resets location 00401 (subroutine error digit).
5. Stores the computed result (which is in ALPHA) back into the address of the A operand.
6. Contains constants and storage areas that are common to other subroutines in the package.

Subroutines Linkages

The following linkage is generated in the mainline program for all subroutine macro-instructions.

TFM	PICK + 11, * + 23
B	SUBR
DORG	* - 4
DSA	A, B

The branch is to the secondary linkage for the specific subroutine used. The secondary linkage is generated by the subroutine processor at the time the subroutines are being relocated and punched into the object deck. For example, the secondary linkage for a variable length subroutine is:

1. SUBR TFM PICK + 402, ADDR } two operands
B PICK
2. SUBR TFM PICK + 402, ADDR } one operand
B PICK + 104

ADDR is the address of the first instruction of the subroutine in question.

This linkage moves the starting address of the subroutine into PICK to allow a later branch to the subroutine (B ADDR). Next, the secondary linkage branches to PICK. PICK moves the data contained in the A operand into ALPHA and the data in the B operand into BETA. It also computes the return address to the mainline program and branches to the subroutine. After the subroutine is executed, the program branches back to the Pick subroutine.

Figure 3 shows the sequence of events that occur when the macro-instruction equivalence is encountered during execution of the object program.

NOTE: When the A operand in the diagram is used in the computation, the secondary linkage branches to PICK; when the A operand is not used in the computation (as in the functional subroutines), the secondary linkage branches to PICK + 104 (variable length) and the B operand alone is placed in BETA.

Note that if the macro-instruction contains more than two operands, arrangements must be made in the programmer's subroutine to store the B, or B and C operands in some location other than **BETA**. This is to prevent the B, or B and C operands from being destroyed, for **PICK** automatically stores any operand, after the first operand, in **BETA**.

The return address to the mainline program, calculated by the **Pick** subroutine is correct only if all operands associated with the subroutine have been processed.

The computed result is always assumed to be stored in **ALPHA**. In addition, the result at **ALPHA** is stored by the **Pick** subroutine at the address specified by the **A** operand, prior to the return to the mainline program.

There are various working areas for constants in the **Pick** subroutine that may be used (shared) by the added subroutines. The programmer may refer to the subroutine program listing (provided with the library package) to make effective use of the **Pick** subroutine.

Bypassing PICK

The subroutine writer may, if he desires, bypass **PICK** completely by setting up the secondary linkage in columns 25-43 of the subroutine header card (see **HEADER CARD FORMAT**).

Since the first linkage puts the address of the **A** operand in **PICK + 11**, the secondary linkage can move it from there to any place in the subroutine itself (and branch to the subroutine). This is what is done in the **DIV**, **FSLS**, and **FSRS** subroutines, where the secondary linkage is of the form

```
TF    ADDR + 11, PICK + 11,0
      B    ADDR,,0
```

When the programmer bypasses **PICK**, he is respon-

sible for performing all necessary operations normally performed by **PICK**.

PICK Address Equivalents

Listed in Table 12 are certain **PICK** address equivalents for fixed length and variable length subroutine decks, as well as for the subroutine deck which uses automatic floating point.

Instructions Relative to PICK

When writing a subroutine, the programmer must set a flag over position O_0 and/or O_1 of instructions where the **P** and/or **Q** operands are relative to the origin (location 05000), e.g., an instruction located at 05300, such as

```
TF * + 23, * - 1
```

in machine language should be $\overline{26}0532305299$. Therefore it should be written as

```
TF * + 23, * - 1, 01
```

If the **P** operand alone is relative, then only O_0 should be flagged, as

```
AM * + 18, 5, 07
```

The **Q**, digit is flagged because the instruction is of the "immediate" type.

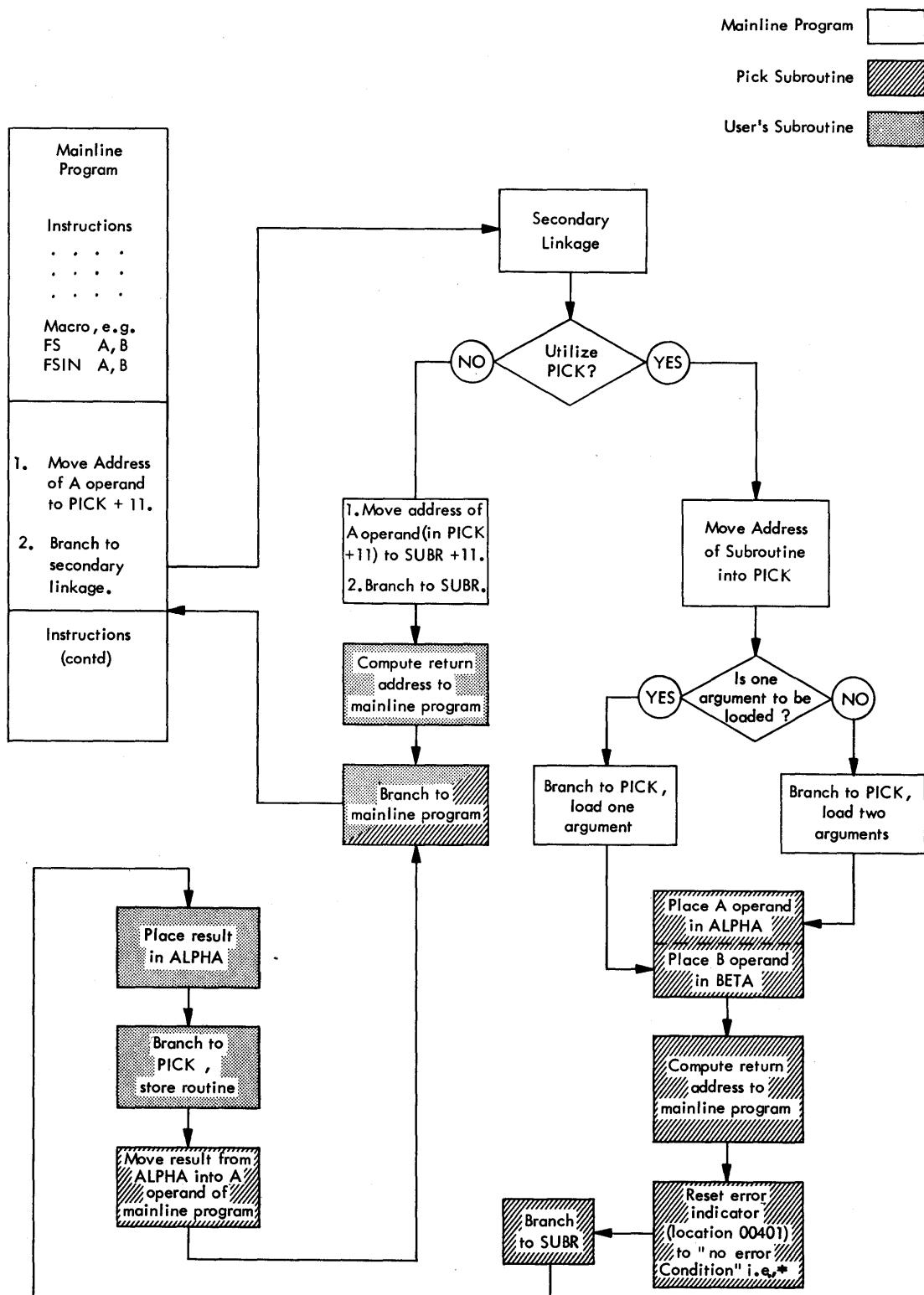
Operand Modification

Whenever **PICK** is used, the programmer must use instructions in his subroutine which make reference to the **Pick** subroutine. The subroutine relocator program must adjust the operands of these instructions to make

Table 12. Pick Address Equivalents

ADDRESS EQUIVALENTS			DESCRIPTION
FIXED LENGTH	VARIABLE LENGTH	AUTOMATIC FLOATING POINT	
PICK	PICK	PICK	Entry for subroutines that <u>use</u> A operand data in the computation.
PICK + 104	PICK + 104	PICK + 24	Entry for subroutines that <u>do not use</u> A operand data in the computation.
PICK + 140	PICK + 140	PICK + 60	Re-entry to pick up additional operand (other than the A and B operand).
PICK + 414	PICK + 402	PICK + 174	Address of subroutine (P address of instruction that branches to the subroutine).
PICK + 416	PICK + 404	PICK + 176	Re-entry from subroutines to store result of computation.
PICK + 482	PICK + 434	PICK + 194	Return address of mainline program (P address of instruction that branches to mainline program).
PICK + 711	PICK + 657	PICK + 417	Alpha (A operand data itself),
PICK + 743	PICK + 802	PICK + 562	Beta (B operand data itself).

Figure 3. Basic Flow Chart of User's Subroutine – PICK Relationship



them correspond to the actual addresses of PICK in the object program. This is done by using a pseudo constant (DC statement). The constant does not become a part of the object program; its only function is to indicate to the subroutine relocator program that the instructions that follow are to be modified.

One DC statement can modify up to 25 instructions. Each instruction, whether or not it is to be modified, requires two digits in the pseudo constant, one for the P operand and one for the Q operand. The statement itself consists of three operands: the first specifies the length of the constant which may not be greater than 50 nor less than 2; the second, the actual constant; the third, the storage address of the constant. This address must be specified as an absolute value in the following form: 00320 for a 20-digit constant, 00342 for a 42-digit constant, etc. The P and Q operand modifier constants follow.

P and Q Operand Modifiers

	<u>Modification</u>
0	No Modification
1	Add L
2	Subtract L
3	Add 2L
4	Subtract 2L
5	Modify with respect to PICK, no L modification
6	Modify with respect to PICK, add L

7	Modify with respect to PICK, subtract L
8	Modify with respect to PICK, add 2L
9	Modify with respect to PICK, subtract 2L

The following example shows how a variable length mantissa subroutine may be modified, by use of modifier constants, to use three operands in its computation. Secondary linkage 1 (two operands) is used in this example.

The A operand data is stored in ALPHA (PICK + 657) and the B operand data is in BETA (PICK + 802). Therefore:

SUBR	DC	6, 275050, 306	
	TR	GAMMA — 1, 801, 0	Transmit beta into gamma
	TFM	402, * + 20, 17	Set up return address to added subroutine
	B	140	Go to Pick subroutine to obtain next operand
		DORG * — 4	

NOTE: Intervening DORG statements and constants between instructions are never modified in this manner.

Data from the last operand processed will be in BETA (PICK + 802). The maximum number of operands allowed in secondary linkage 1 is two; however, the A and B operands may be the same.

1620 SPS III Processor Program

The SPS III processor, which is available in card or paper tape form, is a two-pass program. The user's source program, written in symbolic language, is the input for both passes. The functions of Pass 1 and Pass 2 are listed below:

Pass 1

1. Checks for valid mnemonic operation codes. Invalid operations are considered NOP and are processed as such if Program Switch 2 is off.
2. Processes symbolic labels and prepares a table of the symbolic labels and their assigned addresses for use in the second pass.
3. Assigns storage positions to instructions, work areas, and constants.
4. Performs checking necessary to produce error messages.

Pass 2

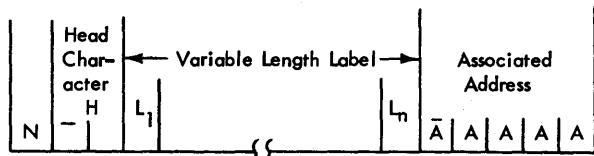
1. Processes operation codes. Converts mnemonic program operation codes to their corresponding 1620 machine language codes.
2. Processes operands according to the type of operation code. Looks up assigned addresses and symbolic operands in the symbolic table prepared during Pass 1. Performs address adjustment, if necessary, to complete the operands. Sets flags in the assembled instruction, as specified by the flag indicator operand.
3. Types error messages for those statements that cannot be assembled properly.
4. Prepares the assembled output and lists the symbol table, if desired.

Storage Layout

The storage layout of the SPS processor is shown in Figure 4. The operation code table contains all valid mnemonic operation codes and their equivalent machine language codes. Any alterations to the processor will change the addresses shown in this figure.

Symbol Table

A variable length label entry is used to store as many labels as possible in the area reserved for the symbol table. Each label when stored takes the following form:



where $N = \text{number of characters in label plus head character (2-6)}$

$H = \text{head character (two-position alphabetic coding)}$

$L \dots L_n = \text{five characters of label (two-position alphabetic coding)}$

$AAAAA = \text{assigned address (five numerical positions)}$

NOTE: The rightmost position of L_n contains a flag for any true 6-character label.

A label entry will always contain N, H, and A data and at least one L character. Therefore, the minimum size label (one character) requires 10 storage positions. Each additional L character will use two additional storage positions up to eight positions. The maximum size label (5 or 6 characters) requires 18 storage positions.

A six-character label is stored without a head character and the leftmost character of that label occupies the head character (H) storage position. For the six-character label, a flag is placed in the rightmost position of L_n so that the processor may distinguish between a 5-character label with a head character and a true 6-character label without the head character. When a label which is not preceded by a HEAD statement is placed in storage, the head character (H) is assigned as blank by the processor.

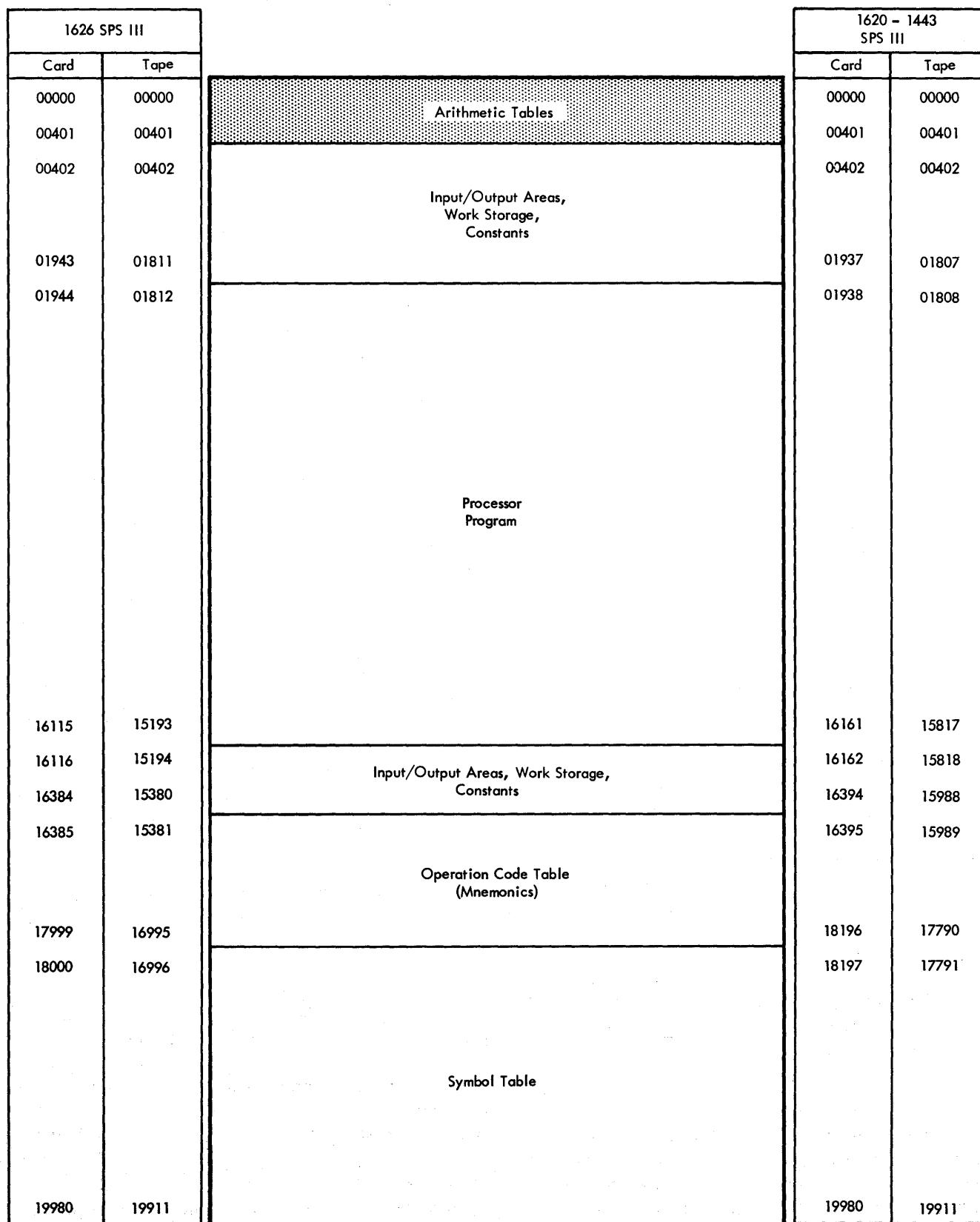


Figure 4. Storage Layout

CAPACITY

The maximum number of symbols or labels cannot be specified due to their variable length. However, the following formula may be applied to find the allowable number of symbols (within ± 1) for any given source program.

$$K = \left[\sum_{\substack{e=1 \\ e=5}} L_e (8 + 2e) \right] + 18L_6$$

e = number of characters in label

L_e = number of labels of length "e"

L_6 = number of six-character labels

K = the storage capacity of the symbol table. K can have a maximum value of:

	<u>1620 SPS III</u>	<u>1620-1443 SPS III</u>
Card	1980	1783
Paper Tape	2715	2120

for 1620 Systems with 20,000 positions of core storage. K should be increased by 20,000 or 40,000 when the assembling system is equipped with 40,000 or 60,000 positions of storage, respectively.

Paper Tape Processor Program

The paper tape processor program accepts input for the first pass in either paper tape form or directly from the typewriter, depending upon the setting of the program switch. If the typewriter is used to enter the source statements, the processor produces a source program tape to be used as input for the second pass.

When subroutines are used in the source program, the subroutine program paper tape must follow the source program as input for the second pass.

Output from the second pass may include an object program tape and/or a typewriter listing. Error messages indicating errors in the source program are typed out during Pass 1 or Pass 2. The programmer has the option of correcting these errors either as indicated or after assembly is finished.

Make-up of Output Tape

The output tape produced by Pass 2 contains the assembled machine language instructions, constants, and other data that are part of the object program. Loading instructions appear at the beginning of the object tape followed by the object program, selected subroutines, and multiplication and addition tables (condensed

form) in that order. A complete self-contained program tape is produced, ready to be entered in the 1620.

Card Processor

Input may be from cards or the typewriter. If the typewriter is used, a source program card deck is punched as output for Pass 1. This card deck becomes the input for Pass 2. Error messages are typed out for both passes. Affected statements may be corrected when the message is noted or at the completion of Pass 2 after all messages have been recorded.

The input for Pass 2 is the source program deck followed by the subroutines, provided they are used. The typewriter output from this pass may consist of the object program with error messages, or error messages only, as determined by the program switch setting.

An object program card deck is produced in condensed form or uncondensed form, depending upon the setting of the program switches. The condensed card contains up to five machine language instructions, thus requiring fewer cards than the uncondensed version, which has multiple cards for each statement. Immediately after an uncondensed object deck is obtained from Pass 2, the programmer can get a condensed deck by processing the source cards a third time as described under OPERATING PROCEDURES.

Both the condensed and uncondensed card decks are complete with loader and arithmetic tables. The uncondensed deck contains both symbolic and absolute information, but only absolute data is loaded.

Make-up of Output Deck

The object program is preceded by two loader cards and followed by seven cards that perform the following:

1. Interrupt the loading sequence of the object program.
2. Load the arithmetic tables.
3. Branch to the start of the object program or branch to a halt.

Uncondensed Object Deck Format

The individual card format of each type of statement is given. The number of cards per statement may range from one to several, depending on the type of operation (imperative, declarative, control, macro-instruction, comments) or type of individual statement. For the SEND or HEAD statements, no output cards are produced.

Imperative Operation Cards

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.

Card 2. Columns 1-5 page and line number.
 6-10 high-order leftmost address where assembled instruction is to be stored.
 11-22 assembled instruction.
 23 \neq
 63-64 11.
 65-69 leftmost address where assembled instruction is to be stored.
 70-74 rightmost address plus one, where assembled instruction to be stored.
 76 9.
 77-80 card number.

Cards 3-9. Arithmetic tables.
 Cards 10-11. Loader program.

Declarative Operation Cards

DSS, DSS

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. Columns 1-5 page and line number.
 6-10 rightmost address of field.
 13-17 field length.
 76 9.
 77-80 card number.

NOTE: For the DSS operation, columns 6-10 of card 2 contain the leftmost address of the field.

DAS

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. Columns 1-5 page and line number.
 6-10 leftmost address plus one, of field.
 13-17 field length (number of alphanumeric characters).
 76 9.
 77-80 card number.

DSB

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. Columns 1-5 page and line number.
 6-10 rightmost address of first element of array.
 13-17 element length.
 76 9.
 77-80 card number.

DSA

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. NOTE: A card of this type is punched for each operand.
Columns 1-5 page and line number.
 6-10 rightmost address where field is to be stored.
 13-17 field length (constant 00005).
 18-22 the 5-digit field (address itself) being stored.
 23 \neq
 63-64 18.

Control Operation Cards

DORG, DEND

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. Columns 1-5 page and line number.
 6-10 address specified.
 76 9.
 77-80 card number.

TRA

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. Columns 1-5 page and line number.
 6-10 leftmost address where instruction is to be stored.
 11-22 assembled instruction (first instruction).
 23 \neq
 63-64 11.
 65-69 leftmost address where instruction is to be stored.
 70-74 rightmost address plus one, where instruction is to be stored.
 76 9.
 77-80 card number.

Card 3. Same as card 2 (11-22 is second instruction).

TCD

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.
Card 2. Columns 1-5 page and line number.
 6-10 address specified.
 76 9.
 77-80 card number.

65-69 leftmost address where field
 is to be stored.
 70-74 rightmost address plus one,
 where field is to be stored.
 76 9.
 77-80 card number.

DC, DSC

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.

Card 2. Columns 1-5 page and line number.
 6-10 rightmost address where constant is to be stored.
 13-17 field length of the constant.
 76 9.
 77-80 card number.

Card 3. Columns 1-5 page and line number.
 6-n the constant itself starts in column 6 and is terminated by a record mark (=) in the first column following the constant.
 63-64 06.
 65-69 leftmost address where constant is to be stored.
 70-74 rightmost address plus one, where constant is to be stored.
 76 0.
 77-80 card number.

NOTE: For the DSC statement, columns 6-10 of card 2 contain the leftmost address where the constant is to be stored.

DNB

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.

Card 2. Columns 1-5 page and line number.
 6-10 rightmost address where constant (blank) is to be stored.
 13-17 field length.
 76 9.
 77-80 card number.

Card 3. Columns 1-5 page and line number.
 7-n the numerical blanks (coded 4, 8) start in column 7 and are terminated by a record mark (=) in the first column following the constant.
 63-64 07.
 65-69 leftmost address where constant (blanks) is to be stored.

70-74 rightmost address plus one, where constant is to be stored.
 76 0
 77-80 card number.

DAC

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.

Card 2. Columns 1-5 page and line number.
 6-10 leftmost address plus one, where constant is to be stored.
 13-17 field length (number of alphanumeric characters).
 76 9.
 77-80 card number.

Card 3. NOTE: A constant that contains over 25 characters causes two cards to be punched in this format. Up to 25 characters may be punched on each card.
 Columns 1-5 page and line number.
 6-n the constant itself starts in column 6 and is terminated by a record mark (=) in the first column following the constant.
 63-64 06.
 65-69 leftmost address where constant is to be stored.
 70-74 rightmost address plus one where constant is to be stored.
 76 0.
 77-80 card number.

Macro-instruction Cards

Card 1. Same as source statement card with the exception of a 0 in column 76 and card number in columns 77-80.

Card 2. Columns 1-5 page and line number.
 6-10 leftmost address where first linkage instruction is to be stored.
 11-22 assembled instruction.
 23 =
 63-64 11.
 65-69 leftmost address where instruction is to be stored.
 70-74 rightmost address plus one, where instruction is to be stored.
 76 9.
 77-80 card number.

Card 3. Same as card 2 (second linkage instruction).

Card 4. NOTE: A card of this type is punched for each operand.

Columns 1-5 page and line number.

6-10 leftmost address where field is to be stored.

13-17 field length (constant 00005).

18-22 the address itself (5-position field) to be stored.

23 \neq

63-64 18.

65-69 leftmost address where field is to be stored.

70-74 rightmost address plus one, where field is to be stored.

76 9.

77-80 card number.

Comments Cards

Card 1. Same as source statement with the exception of a 0 in column 76 and card number in columns 77-80.

Card 2. A digit 9 in column 76 and card number in columns 77-80.

Listing the Uncondensed Object Deck

Figures 5 and 6 show control panel wiring diagrams designed for listing an uncondensed object program on the IBM 407 and on the IBM 407-E8.

Condensed Object Deck Format

Condensed cards are punched as described under the particular card type.

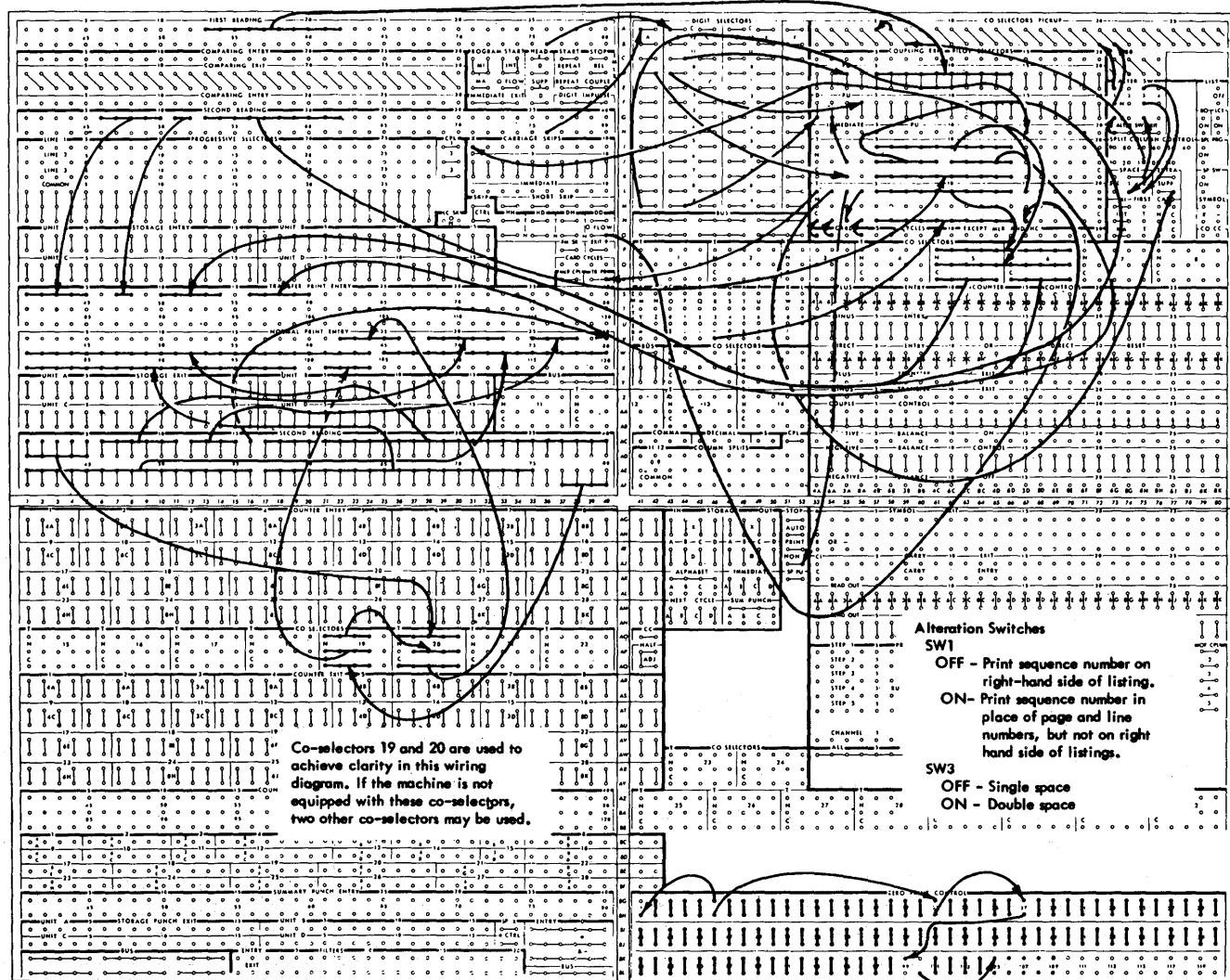


Figure 5. 407 Control Panel Wiring Diagram for Listing Uncondensed Output

Cards Containing Instructions

Columns 1-12

13-24

25-36

37-48

49-60 five instructions.

61 ± (record mark).

62 0.

63-64 01

65-69 left

70-74 rightmost address plus one,
where instructions are to be loaded.

where instructions are to be loaded.

76 (flag only).

77-80 card number.

Cards Containing Constants

Columns 1-61 constants may be from 1 to 60 characters followed immediately by a record mark (\mp).

62 1.

63-64 01.

65-69 leftmost address where constants are to be loaded.

70-74 rightmost address plus one,
where constants are to be
loaded.

76 (flag only).

77-80 card number

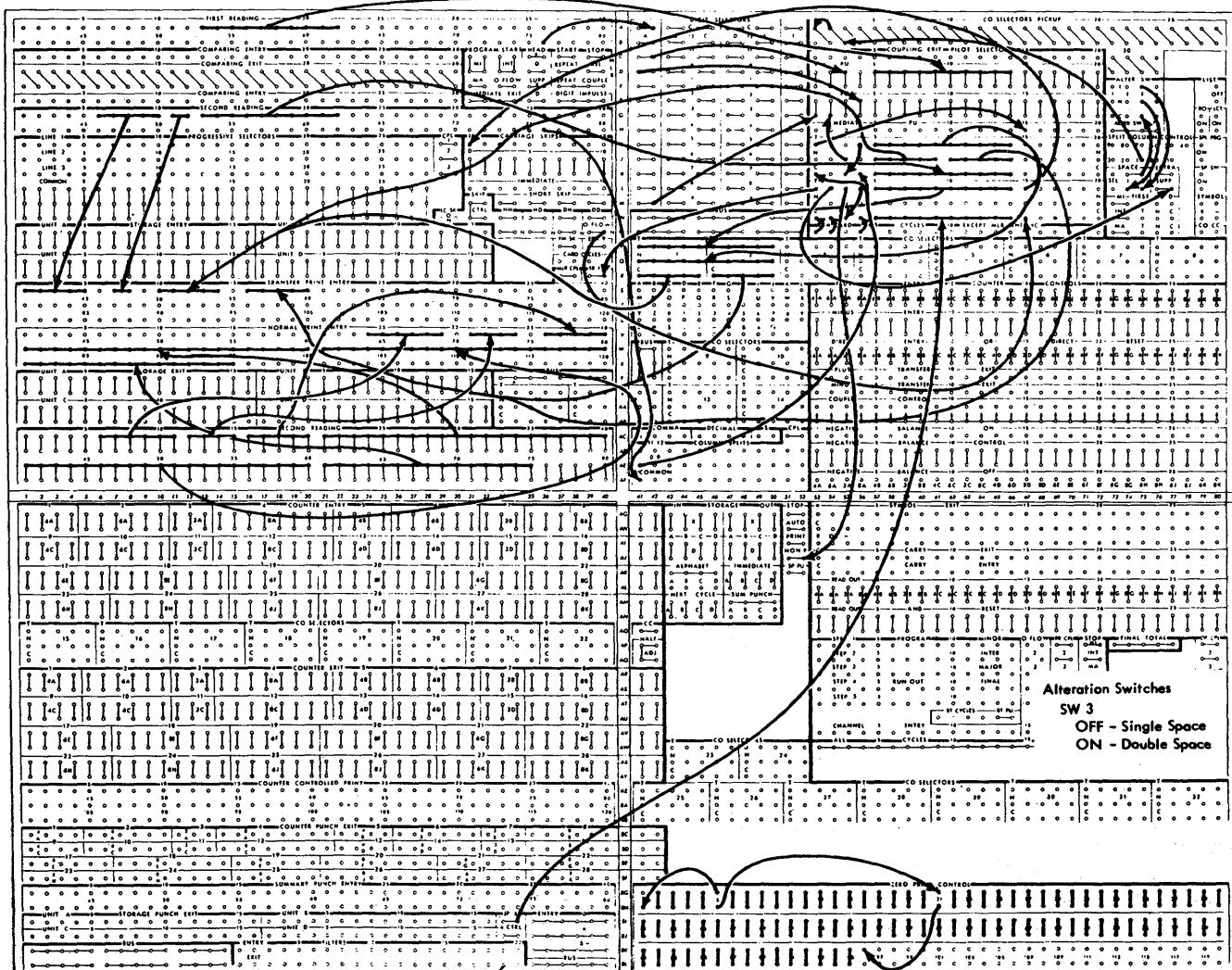


Figure 6. 407-E8 Control Panel Wiring Diagram for Listing Uncondensed Output

Condensed Object Deck Alterations

While testing an object program, it is often necessary to change or patch some of the original instructions. To do this, the SPS source deck and the condensed deck must be updated each time an instruction is changed. If the SPS output is an uncondensed deck that is later condensed, it is necessary to update three card decks (the source, uncondensed, and condensed decks). The procedure described here provides an orderly, rapid, and accurate means of correcting the source program.

Patch Card and Coding Sheet

When an instruction requires correction, the corrected machine language instruction as well as the SPS coding should be recorded on a patch coding sheet. The coding sheet shown in Figure 7 or a similar coding sheet can be used for this purpose. Each entry on the coding sheet is punched into a prepunched patch card. The *prepunched* data is arranged on the patch card as follows:

Columns 13 \neq (0, 2, 8 punches)
 62-65 001
 70 -(flag only)

75 blank
76 0
77 9

Use of Patch Cards

Patch cards are placed in front of the last seven cards of the condensed object deck but no cards are removed or changed. The condensed object deck is loaded in the same manner as it was before the patch cards were added.

Corrections to the Source Program

After the object program is tested, the source program deck can be corrected by selecting patch cards with a code 9 in column 77 from the object deck. Card columns 14 through 61 in the selected cards should be reproduced into columns 1 through 48 of blank cards. The reproduced cards should then be inserted in page and line number sequence into the source program deck, and the cards that are being replaced should be discarded.

Patch cards may then be returned to the condensed object deck or the source deck may be reassembled to produce a new object deck.

Figure 7. Patch Card Coding Sheet

Operating Procedures

This section describes the procedures to be followed when using the 1620 SPS III Processor.

Switches

Both the Parity switch and the I/O switch should be placed in the STOP position, and the Overflow switch in the PROGRAM position. Program switches for controlling the processor should be set as outlined in Table 13. Note the different functions of the Program switches during the "Loading Processor - Pass 1 - Pass 2 - Post Assembly" phases.

Program Switch 1

The processor is set up to punch the add table (required for Model 1s only) in the object program under control of Program Switch 1. Switch 1 is interrogated when the processor is being loaded, and, if the switch is off, the add table will be punched in the object program. If Program Switch 1 is on during loading of the processor, the add table will not be included in the object program. This option enables the 1620 Model 2 user to load programs (even with TRA-TCD operations) without loading over the area of the index registers. During loading, the message

1620 SPS III, MODEL 1

or

1620 SPS III, MODEL 2

is typed depending upon the setting of Program switch 1.

Loading the Processor

(a) Card system

1. Depress the console Reset key.
2. Set Program switch 1 to the desired setting.
3. Place the processor deck in the read hopper.
4. Depress the Load key.

(b) Paper Tape System

1. Thread the processor tape on the paper tape reader.
2. Depress the Reset and Insert keys.
3. Set Program switch 1 to the desired setting.
4. Enter 360000000300 from the typewriter.
5. Depress the R-S key.

Table 13. 1620 SPS III Program Switch Settings

	SWITCH NO.	POSITION	
		ON	OFF
LOADING PROCESSOR	1	No add table will be punched in object program.	Add table will be punched in object program.
	2, 3, 4	Not used.	
PASS 1	1	Input is from the card reader or paper tape reader.	Input is from typewriter.
	2	The machine stops after an error message has been typed so that corrected statement can be entered at the typewriter.	Processing continues after error message is typed, but error is adjusted as described under ERROR CORRECTIONS.
	3	Not used.	
	4	Turn on to correct typing error made while entering a statement, and depress R-S key, then off, and re-enter the entire statement at the typewriter. Leave off when assembling program.	
PASS 2	1	The source statement and assembled instruction is typed out.	No listing is typed
	2	Same as Pass 1	Same as Pass 1
	3	a. For card processor, when the object program is to be in condensed form. b. Must be on for editing.	For the card processor, when the object program is to be in uncondensed form.
	4	a. No object program will be punched. b. Must be on for editing.	The object program is punched.
POST ASSEMBLY	4	Symbol table is not listed.	Symbol table is listed.

NOTES:

1. If listing and/or uncondensed output has just been obtained, set switch 3 ON to obtain condensed deck (re-enter source input).
2. If edit just completed, set switch 4 OFF to obtain condensed deck (re-enter source input).
3. To re-enter Pass 1, turn switch 3 OFF (set switches 1, 2, and 4 also, and load source input).

Upon completion of loading, 48 will appear in the Operation Register.

While the processor is being loaded, the size of the assembling machine is determined by programming.

The size of the object machine is set to the same size as that of the assembling machine. When the processor is used on a system with more than 20,000 positions of core storage, the additional storage allows a larger area for source program labels and their assigned addresses.

Processing the Source Program

The processor is ready to start the assembly process as soon as the processor has been loaded.

Pass 1

- (a) Card Input:
 - 1. Place program source deck in card reader hopper.
 - 2. Depress Reader Start key.
 - 3. Set Program switches (see Table 13).
 - 4. Depress the console Start key.
- (b) Paper Tape Input:
 - 1. Thread the input tape (source program).
 - 2. Set Program switches (see Table 13).
 - 3. Depress the console Start key.
- (c) Typewriter Input:
 - 1. Thread blank tape on tape punch or place blank cards in card punch and depress the Punch Start key (see NOTE below).
 - 2. Set Program switches (see Table 13).
 - 3. Type the source statement and follow it with a record mark.
 - 4. Depress the R-S key (repeat steps 3 and 4 until all statements have been processed).

At the completion of Pass 1, the message "END OF PASS I" is typed out and the program halts. The processor is ready to begin Pass 2.

NOTE: Source statements entered on the typewriter during Pass 1 are outputted on paper tape if the tape processor is being used or outputted to cards if the card processor is being used. This output then becomes the input for Pass 2.

Pass 2

- (a) Card Input:
 - 1. Place program source deck in the card reader hopper.
 - 2. Place blank cards in the punch hopper.
 - 3. Depress the Reader Start key and Punch Start key.
 - 4. Set Program switches (see Table 13).
 - 5. Depress console Start key.
- (b) Paper Tape input:
 - 1. Thread the input tape (source program) on the paper tape reader.
 - 2. Thread blank tape on the paper tape punch.

- 3. Set Program switches (see Table 13).
- 4. Depress console Start key.

When Pass 2 is completed, the message

LOAD SUBROUTINES

will be typed out if subroutines are required by the source programs (see **LOADING SUBROUTINES**). If subroutines are not required, the message

END OF PASS II

is typed out. At the end of this message the symbol table is typed out if Program switch 4 is off.

Typewriter Output

With Program switch 1 on during Pass 2, the typewriter types each statement starting at the left margin. After the last character of each statement is typed, the typewriter carriage returns and tabulates into position for typing the storage address and assembled instruction. Statements are typed in the format in which they are entered; however, a space is inserted before and after the operation field.

To set up the typewriter, the operator must:

- 1. Set margins to the extreme right and left positions.
- 2. Set a single tab stop at least 21 spaces to the left of the right margin.

Symbol Table Typeout Format

When the symbol table is typed, the labels and their assigned addresses are typed five to a line. The format of each address and label is as follows:

Assigned
Address Label

or
xxxxx *xxxxxx

where the leftmost position of the label contains the head character. Any true 6-character label is indicated by an asterisk in the leftmost position next to the 6-character label.

The symbol table typeout may be suppressed by turning Program switch 4 to the ON position when the message

END OF PASS II

is being typed, or during the typeout of the symbol table.

Condensed Object Deck

A condensed object deck can be obtained during Pass 2 of the card processor by having Program switch 3 in the ON position and switch 4 in the OFF position. If undecoded output is obtained during Pass 2, condensed output can also be obtained (without reloading the processor) by:

1. Setting Program switch 3 to the ON position.
2. Placing the program source deck in the card reader hopper.
3. Depressing the Reader Start and Punch Start keys.
4. Depressing the console Start key.

Loading Subroutines

Subroutines must be loaded after the message

LOAD SUBROUTINES

is typed out. Only those subroutines used by the object program are punched out as part of the object program. To load subroutines from Paper Tape input:

1. Thread the subroutine tape.
2. Depress the console Start key.

Card Input:

1. Place the subroutine card deck in the card reader hopper.
2. Depress the Reader Start key.
3. Depress the console Start key.

Errors that occur while the subroutine deck (or tape) is being processed cannot be corrected by manual intervention because any information inserted in locations 00000 through 00099 will result in erroneous address modification.

If the subroutine being processed is a variable mantissa length subroutine, the message

ENTER MANTISSA LENGTH

is typed and the program halts. The operator enters the two-digit number from the typewriter. The range of the number to be entered is from 02 to 45. If no number is entered, the mantissa length is automatically set to 08. The R-S key must be depressed to continue processing. Table 10 shows the identification numbers and storage requirements for the subroutines provided with SPS III and SPS III for Printer. The identification number is punched in column 77 of the subroutine card decks and is punched in the leader of the subroutine tapes. Subroutine secondary linkages are typed out (printed with 1630-1443 SPS III) as the subroutines are added to the object deck.

After the subroutines are processed, the message

END OF PASS II

is typed out.

Editing the Source Program

When a large program is to be assembled, the operator may choose to perform an edit operation prior to actual assembly of the object program. During the edit operation, error messages are typed out for Passes 1 and 2, no listing is typed, and no object program is punched. For this reason, edit data may be obtained in less time than is required for normal assembly. The error listings from the operation enable the programmer to correct the program prior to assembling the object program.

The operating procedures are the same as those for passes 1 and 2 described for a normal assembly except that Program switches 3 and 4 (see Table 13) must be on during Pass 2.

Error Messages

The error message codes that might be typed out on the typewriter during Pass 1 and/or Pass 2 of an assembly are listed in numerical sequence.

Code	Message	Description of Error
ER1		A record mark is in the label or operation code field.
ER2		For address adjustment, a product greater than ten digits has resulted from a multiplication.
ER3		An invalid operation code has been used.
ER4		A dollar sign, which is being used as a HEAD indicator, is incorrectly positioned in an operand.
ER5		<ol style="list-style-type: none">1. A symbolic address contains more than six characters.2. An actual address contains more than five digits.3. An undefined symbolic address is used in an operand.4. A HEAD (\$) character is improperly specified.5. More than six characters precede the number in the index register field.6. Improper position of left or right index register parenthesis.7. Index register > 7 specified.8. Non-numeric index register specified.
ER6		A DSA statement has more than ten operands.
ER7		A DS _B statement has the second operand missing.
ER8		<ol style="list-style-type: none">1. A DC, DSC, DAC, or DNB has a specified length greater than 50.2. A DC, DSC, or DAC statement has no constant specified.

- 3. A DC, or DSC has a constant which has a greater number of digits than its specified length.
 - 4. A DAC statement has a specified length not equal to the number of characters in the constant itself.
- ER9 The symbol table is full.
- ER10 A duplicate label is defined (defined more than once).
- ER11 An assembled address is greater than five digits.
- ER12 An invalid special character is used as a head character in a HEAD statement.
- ER13 A HEAD statement operand contains more than one character.
- ER14 An invalid special character is used in a label. The eight invalid special characters are: "blank) + \$ * -, (". An all-numerical label is also invalid.

Error messages are in the following form:

<u>LABEL</u>	adjustment count	error code
XXXXXX	+ XXXX	ERn

Where LABEL refers to the last defined label and the "adjustment count" refers to the number of statements between the label and the statement in error. If the first statement of a source program contains the label START, and the second statement has an error "ER1," the following message will be typed:

START + 0001 ER1

If the second statement has the label XYZ, the message still would appear as START + 0001, not as xyz + 0000.

The messages will appear in the form just shown during Pass 1 or Pass 2, if Program switch 1 is off. If Program switch 1 is on during the second pass, only the error code "ERn" will be typed opposite the statement in error, at the right-hand side of the page.

Error Correction

Each erroneous statement can be corrected individually after the error message is typed and the machine is halted or all statements containing errors can be corrected after the object program is assembled. If there are few errors, the first procedure may be advisable; where there are many errors, it is advisable to correct the source deck at the end of the run and reassemble the program.

Some errors in source statements entered from cards or tape on Pass 1 are detected again during Pass 2. Therefore, they must be corrected twice. If errors in source statements entered from the typewriter on Pass 1 are corrected, they do not have to be corrected during Pass 2, since the output of Pass 1 becomes the input to Pass 2 and contains the corrected statement.

Program Switch 2 On

With Program Switch 2 on, the processor stops after typing the error message and the carriage returns. The operator enters the corrected statement and depresses the R-S key.

Program Switch 2 Off

With Program Switch 2 off, the processor does not stop for an error; however, errors can be corrected after assembly. Errors affect the assembly process as indicated in the following list.

<u>Error Code</u>	<u>Assembly Process</u>
ER1, ER3	A NOP instruction (410000000000) is assembled. The label is treated as a blank.
ER2, ER4, ER11	The operand is assembled as 00000 (zero) address.
ER5	1-6. The operand is assembled as an absolute 00000. 7, 8. Operand is assembled, but no index register flags are set.
ER6	The first ten operands are assembled; those over ten are ignored.
ER7	The statement is assembled in the same manner as a DS statement with a length of 50.
ER8	If the operation code is: DC – it is assembled in the same manner as a DS statement with a length of 50. DSC – it is assembled in the same manner as a DSS statement with a length of 50. DAC – it is assembled in the same manner as a DAS statement with a length of 50. DNB – it is assembled as a DNB with a length of 50.
ER9, ER10, ER14	The label is treated as blank.
ER12	The head character is replaced by a blank character.
ER13	The first non-blank character specified in the operand is used as the head character.

1620-1443 SPS III

1620-1443 SPS III is a printer-oriented version of the 1620 SPS III previously described in this manual. IBM 1620-1443 SPS III contains mnemonics for printer operation codes; during assembly, the source statements and assembled instructions can be listed on the printer. The printer instructions and unique mnemonics are given in the following examples. The programmer need not be concerned with the actual Q-address modifiers when coding with the SPS language.

Examples

Label	Operation	Operands & Remarks									
4	PRN, DATA, 3, 3, PRINT NUMERICALLY.	20	21	22	23	24	25	26	27	28	29
	PRNSDATA, 3, 3, PRINT NUMERICALLY AND SUPPRESS SPACING.										
	SKIP, 2, 3, SKIP IMMEDIATE TO CARRIAGE CHANNEL 2.										
	SKAP, 5, 3, SKIP AFTER PRINTING TO CARRIAGE CHANNEL 5.										
	SPIM, 3, 3, MOVE CARRIAGE 3 SPACES, IMMEDIATE.										
	SPAP, 3, 3, MOVE CARRIAGE 3 SPACES AFTER PRINTING.										

The operand DATA represents the storage address of data to be printed, the operand control codes (2, 5, 3, 3) are taken from Tables 14 and 15 for carriage skipping and spacing, respectively.

Operating Procedures

Only the operating procedures and messages that are different from those described for 1620 SPS III are given in this section.

During loading of the 1620-1443 SPS III processor program, the message

1620 SPS III FOR PRINTER, MODEL 1

or

1620 SPS III FOR PRINTER, MODEL 2

is typed out at the console typewriter, depending on the setting of Program Switch 1. Switch 1 should be set to the ON position if the object program is to be executed on a 1620 Model 2; Switch 1 should be set to

Table 14. Carriage Skip Operations (Q Address) Control Codes

CONTROL CODES	ACTUAL Q ₁₀ Q ₁₁ MODIFIERS	
	IMMEDIATE	AFTER PRINTING (DELAY)
Skip to Channel 1	71	41
2	72	42
3	73	43
4	74	44
5	75	45
6	76	46
7	77	47
8	78	48
9	79	49
10	70	40
11	33	03
12	34	04

the OFF position if the object of the program is to be executed on a 1620 Model 1.

With Program Switch 1 on during Pass 2, the statements and assembled instructions are printed on the 1443 Printer. The assembled instruction is printed starting at the left margin and the source statement is printed to the right of the assembled instruction. Statements are printed in the same format as they are entered except that spaces are inserted between the mnemonic and P operand, and between the P and Q operands to aid in the readability of the listings.

Output Change

Only a condensed object deck (or tape) can be obtained from 1620-1443 SPS III. The list deck or "one-instruction-per-card" output is not produced since the listing can be obtained during the processing of the source program.

Symbol Table Listing

At the conclusion of Pass 2, the symbol table can be listed on the printer. The labels and their assigned ad-

Table 15. Carriage Space Operations (Q Address) Control Codes

CONTROL CODES	ACTUAL Q ₁₀ Q ₁₁ MODIFIERS	
	IMMEDIATE	AFTER PRINTING (DELAY)
Number of Spaces 1	51	21
2	52	62
3	53	63

resses are printed five to a line in the same format as that described for SPS III.

Error Messages

All error messages are still typed out on the typewriter for the convenience of the operator. The only description for error messages that changes is that for ER2. The description is expanded to include "printer control operation code was improperly specified." With Program switch 2 off, the Q operand is assembled as 00900.

Table 16 shows the Program Switch settings for 1620-1443 SPS III.

Printer Operation

To set up the printer, the operator must:

1. Prepare a carriage control tape with punches in channel 1 where printing is to start and channel 12 where printing is to stop on the page. Place the tape on the carriage tape drive.
2. Put forms in the printer and adjust forms so that the print bar is set where printing is to begin on the page.
3. Disengage the carriage clutch and depress the Carriage Restore key to position the carriage tape at channel 1.
4. Engage the carriage clutch and depress the Printer Start key.

Table 16. 1620-1443 SPS III Program Switch Settings

	SWITCH NO.	POSITION	
		ON	OFF
LOADING PROCESSOR	1	No add table will be punched in object program.	Add table will be punched in object program.
	2, 3, 4	Not used.	
PASS 1	1	Input is from the card reader or paper tape reader.	Input is from typewriter.
	2	The machine stops after an error message has been typed, so that corrected statement can be entered at the typewriter.	Processing continues after error message is typed, but error is adjusted as described under ERROR CORRECTIONS.
	3	Not used.	
	4	Turn on to correct typing error made while entering a statement, and depress R-S key,	then off, and re-enter the entire statement at the typewriter. Leave off when assembling program.
PASS 2	1	The source statement and assembled instruction is printed.	No listing is printed.
	2	Same as Pass 1	Same as Pass 1
	3	Not used.	
	4	a. No object program will be punched. b. Must be on for editing.	The object program is punched.
POST ASSEMBLY	4	Symbol table is not listed.	Symbol table is listed.

NOTES:

1. After Pass 2 is completed, a 48 appears in the operation register.
Depression of the Start key sends control to Pass 1 again.

Index

<i>Page</i>	<i>Page</i>
Add (A) instruction	16
Add Immediate (AI) instruction	16
Adding macro-instructions to processor	46, 47
Adding subroutines	44, 46
Address Adjustment	8
Addresses	
actual	7
asterisk	7
symbolic	7
Alpha	48
AND to Field (ANDF) instruction	17
Any Data Check code	19
Arithmetic instructions	16
Arithmetic subroutines	30
Asterisk (special character)	4
At(@)sign (special character)	6
Backspace Typewriter (BKY) instruction	24
Beta	48
Blank	
special character	5
defining (DNB)	14
Branch and Load Index Register	
Immediate (BLXM) instruction	21
Branch and Load Index Register	
(BLX) instruction	21
Branch and Store Index Register	
(BSX) instruction	21
Branch and Transmit Address	
Immediate (BTAM) instruction	20
Branch and Transmit Address	
(BTA) instruction	20
Branch and Transmit (BI) instruction	20
Branch and Transmit Floating	
instruction (BTFL)	20
subroutine (BTFS)	40
Branch and Transmit Immediate	
(BTM) instruction	20
Branch and Select Band A (BSBA) instruction	20
Branch and Select Band B (BSBB) instruction	20
Branch Conditionally, Modify Index	
Register (BCX) instruction	20
Branch and Modify Index Register (BX) instruction	20
Branch and Modify Index Register	
Immediate (BXM) instruction	20
Branch and Select (BS) instructions	20
Branch and Select Indirect Addressing (BSIA) instruction	20
Branch and Select No I/A (BSNI) instruction	20
Branch and Select No Index Register (BSNX) instruction	20
Branch Any Data Check (BA) instruction	18
Branch Back (BB) instruction	20
Branch Band A Not Selected (BANS) instruction	19
Branch Band A Selected (BBAS) instruction	18
Branch Band B Not Selected (BBNS) instruction	19
Branch Band B Selected (BBBS) instruction	18
Branch Conditionally, Modify Index Register	
Immediate (BCXM) instruction	21
Branch Console Switch instructions	
(BC1, BC2)	18
(BC3, BC4)	19
Branch Either Band Selected (BEBS) instruction	19
Branch Exponent Check (BXV) instruction	19
Branch Equal (BE) instruction	18
Branch High (BH) instruction	18
Branch Indicator (BI) instructions	
indicator codes summary	21
switch codes summary	21
Branch Instructions	18
Branch Last Card (BLC) instruction	19
Branch Low (BL) instruction	19
Branch Negative (BN) instruction	19
Branch Neither Band Selected (BNBS) instruction	18
Branch No Flag (BNF) instruction	18
Branch No Group Mark (BNG)	18
Branch No Indicator (BNI) instruction	
indicator codes summary	21
switch codes summary	21
Branch No Overflow (BNO) instruction	19
Branch No Record Mark (BNR) instruction	18
Branch Not Any Data Check (BNA) instruction	19
Branch Not Equal (BNE) instruction	19
Branch Not Exponent Check (BNXV) instruction	20
Branch Not High (BNH) instruction	19
Branch Not Last Card (BNLC) instruction	19
Branch Not Low (BNL) instruction	18
Branch Not Negative (BNN) instruction	18
Branch Not Positive (BNP) instruction	19
Branch Not Zero (BNZ) instruction	19
Branch on Bit (BBI) instruction	21
Branch on Channel 9 (BCH9) instruction	19
Branch on Channel Overflow (BCOV) instruction	19
Branch on Digit (BD) instruction	18
Branch on Mask (BMK) instruction	21
Branch Overflow (BV) instruction	18
Branch Positive (BP) instruction	18
Branch Zero (BZ) instruction	18
Card Input	61
Carriage Control Codes, printer	65
Clear Flag (CF) instruction	26
Coding Sheet	1, 2
Comma (special character)	4
Compare (C) instruction	18
Compare Immediate (CM) instruction	18
Complement Octal Field (CPLF) instruction	17
Condensed object deck	
format	57
alterations	59
how to obtain	62
Control Codes	
input/output instructions	25
carriage, printer	64
Control (K) instruction	24
Control Statements	
DEND	25

	<i>Page</i>		<i>Page</i>
DORG	22	Floating Exponential (Natural) subroutine	42
HEAD	27	Floating Logarithm (Base 10) subroutine	43
SEND	26	Floating Logarithm (Natural) subroutine	42
TRA	28	Floating Multiply instruction (FMUL)	16
TCD	28	subroutine (FM)	36
Core storage layout	53	Floating Point Arithmetic	32
Data transmission subroutines	30	Fixed Point Divide subroutine	37
Declarative operations		Floating Shift Left instruction (FSL)	17
card format	55	subroutine (FSLS)	39
codes	10	Floating Shift Right instruction (FSR)	17
functions	10	subroutine (FSRS)	38
Decimal to Octal Conversion (DTO) instruction	17	Floating Sine subroutine	40
Define Alphameric Constant (DAC) statement	13	Floating Square Root subroutine	40
output card format	56	Floating Subtract instruction (FSUB)	16
Define Alphameric Symbol (DAS) statement	11	subroutine (FS)	36
output card format	55	Format, output condensed	57
Define Constant (DC) statement	11	patch card	59
output card format	56	uncondensed	54
Define END (DEND) statement	25	Functional subroutines	30
Define ORIGIN (DORG) statement	22	Halt (H) instruction	26
Define Numerical Blank (DNB) statement	14	Header card format, subroutine	44
output card format	56	Heading (HEAD) statement	27
Define Special Constant (DSC) statement	12	Heading line, coding sheet	2
output card format	56	Immediate-type instructions	6
Define Special Symbol (DSS) statement	11	Imperative statements	14
output card format	55	Index Registers	9
Define Symbol (DS) statement	10	Index Typewriter (IXTY) instruction	34
output card format	55	Indicator codes (branch instructions)	21
Define Symbolic Address (DSA) statement	13	Indirect addressing	30
output card format	55	Input and Output Unit Codes Table 7	25
Define Symbolic Block (DSB) statement	14	Input and Output Statements	22-25
output card format	55	Internal Data Transmission statements	16, 17
Divide instruction (D)	16	Label, coding sheet	3
Divide Immediate (DM) instruction	16	Library change card	46
Divisor, positioning of	38	Line number, coding sheet	2
Dollar sign (special character)	6	Linkages, subroutine	48
Dump Numerically (DN) instruction	23	Load Dividend Immediate (LDM) instruction	16
Dump Numerically Card (DNCD) instruction	23	Load Dividend (LD) instruction	16
Dump Numerically Paper Tape (DNPT) instruction	23	Loading the processor	60
Dump Numerically Typewriter (DNTY) instruction	23	Location assignment counter	10, 15
Duplicate Symbols (labels)	27	Logic statements	17-21
Editing source programs	62	Machine requirements	iv
End-of-line character	5	Macro-instructions, subroutine	30
Equal sign (special character)	3	Mantissa defined	33
Error Correction	63	entering length	62
Error Messages		MAR Check Indicator code	21
processor	62	Mask Digit operand	6
subroutine	43, 44	Miscellaneous statements	22, 26
Evaluation of arguments (subroutines)	29	Messages processor	62
Exclusive OR to Field (EORF) instruction	17	subroutine	43, 44
Exponent, defined	33	Move Address (MA) instruction	17
Field	2	Move Flag (MF) instruction	26
Flag indicator operand	6		
Floating Add			
instruction (FADD)	16		
subroutine (FA)	36		
Floating Arctangent subroutine	41		
Floating Cosine subroutine	41		
Floating Divide			
instruction (FDIV)	16		
subroutine (FD)	37		
Floating Exponential (Base 10) subroutine	42		

	<i>Page</i>
Multiply Immediate (MM) instruction	16
Multiply (M) instruction	16
N digit	33
No Operation (NOP) instruction	26
Normalized, defined	33
Object deck	
condensed	57
uncondensed	54
Object program	1
Octal to Decimal Conversion (OTD) instruction	17
Operands and Remarks, coding sheet	3
Operands	
Flag	6
Immediate (Q operand)	6
Mask Digit	6
Modification	9, 49
Operating procedures	
1620 SPS III	60
1620-1443 SPS III	64
Operation code field, coding sheet	3
OR to Field (ORF) instruction	17
Output Unit Codes	25
Overflow, exponent	34
Page number, coding sheet	2
Parenthesis (special character)	5
Paper tape input	61
Patching source programs	59
Period (special character)	3
Pick, subroutine	31, 35
bypassing	49
functions	48
address equivalents	49
Print Alphamerically (PRA) instruction	24
Print Alphamerically and Suppress Spacing (PRAS) instruction	24
Print Numerically and Suppress Spacing (PRNS) instruction	23
Print Numerically (PRN) instruction	23
Printer Dump and Suppress Spacing (PRDS) instructions	23
Printer Dump (PRD) instruction	23
Printer Operation	65
Processor Control statements	
DEND	25
DORG	22
HEAD	27
SEND	26
TRA	28
TCD	28
Processor program	
1620 SPS III	52
1620-1443 SPS III	64
Product area	37
Program switch settings	
1620 SPS III	60
1620-1443 SPS III	65
Q address, disk instructions	25
Quotient address	37
Read Alphamerically Card (RACD) instruction	24
Read Alphamerically Paper Tape (RAPT) instruction	24
Read Alphamerically (RA) instruction	23
Read Alphamerically Typewriter (RATY) instruction	24
Read Binary Paper Tape (RBPT) instruction	24
Read Numerically Card (RNCD) instruction	23
Read Numerically Paper Tape (RNPT) instruction	23
Read Numerically (RN) instruction	23
Read Numerically Typewriter (RNTY) instruction	23
Record Mark	12
Return Carriage Typewriter (RCTY) instruction	24
Secondary linkage format	48
Set Flag (SF) instruction	26
Skip After Printing (SKAP) instruction	24
Skip Immediate (SKIP) instruction	24
Slash (special symbol)	3
Source program	
assembling	61
card	3
defined	5
editing	62
patching (altering) of	59
Space After Printing (SPAP) instruction	25
Space Immediate (SPIM) instruction	25
Space Typewriter (SPTY) instruction	34
Special characters	
Asterisk	4
At @ sign	6
Blank	5
Comma	4
Dollar sign	6
Equal sign	3
End-of-line character	5
parenthesis	5
period	3
slash	3
Special END (SEND) statement	26
Statement writing	4
Statements,	
Control	22
Declarative	10, 15
Imperative	14
Arithmetic	14, 16
Internal Data Transmission	16, 17
Input and Output	22-25
Logic	17-21
Miscellaneous	22, 26
Storage layout, processor	52, 53
Subroutines	
Adding	
to card	44
to paper tape	46
Arithmetic	30
Data Transmission	30
Error Messages	43, 44
Functional	30
Header card format	44
Library change card	46
Linkages	48
Macro-instructions	30
Sample	45
Storage requirements	35
Trailer card format	45
Writing	48
Subtract Immediate (SM) instruction	16
Subtract (S) instruction	16

	<i>Page</i>		<i>Page</i>
Symbol table	52	Transmit Record (TR) instruction	17
capacity	54	Transmit Record No Record Mark	
listing	64	(TRNM) instruction	17
Tabulate Typewriter (TBTY) instruction	34	Typewriter input	61
Tape Modifier program	46	Uncondensed output	54
Trailer card format, subroutine	45	Underflow, exponent	34
Transfer Control and Load (TCD)			
instruction	28	Variable lenth, defined	29
card format	55	Variable length mantissa subroutine	29
Transfer Numerical Fill (TNF) instruction	17	Wiring diagrams for 407 listing of object programs	57, 58
Transfer Numerical Strip (TNS) instruction	17	Write Alphamerically Card (WACD) instruction	24
Transfer to Return Address (TRA)		Write Alphamerically Paper Tape (WAPT) instruction	24
instruction	28	Write Alphamerically Typewriter (WATY) instruction	24
card format	55	Write Alphamerically (WA) instruction	24
Transmit Digit (TD) instruction	17	Write Binary Paper Tape (WBPT) instruction	24
Transmit Digit Immediate (TDM) instruction	17	Write Numerically Card (WNCD) instruction	23
Transmit Field (TF) instruction	17	Write Numerically Paper Tape (WNPT) instruction	23
Transmit Field Immediate (TFM) instruction	17	Write Numerically Typewriter (WNTY) instruction	23
Transmit Floating		Write Numerically (WN) instruction	23
instruction (TFL)	17	Writing subroutines	48
subroutine (TFLS)	39		

IBM

**International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601**