File Number 1401-21
Form C24-1480-0

IBM Systems Reference Library

IBM 1401 Symbolic Programming Systems: SPS-1 and SPS-2 Specifications and Operating Procedures

This manual provides programmers with the information necessary to code a 1401 program in SPS language and assemble a machine-language object-program. It is assumed that the programmer has a basic knowledge of 1401 machine language programming.

It describes symbolic programming principles and concepts and gives detailed specifications of the 1401 Symbolic Programming Systems, SPS 1 and SPS 2.

Operating instructions for processing the SPS source program are enumerated. The SPS processor program can assemble a machine language program on configurations of the 1401 Data Processing System equipped with a 1402 Card Read-Punch.

A sample program is included for the convenience of the beginning SPS programmer. Input and output forms, a block diagram of the program procedure, the symbolic program, and SPS output listings of the symbolic and machine-language programs are shown.

Preface

This manual describes the language specifications and operating procedures for the IBM 1401 Symbolic Programming Systems SPS-1 and SPS-2.

This manual is designed for programmers who know the input, output, and processing characteristics, as well as the basic functions and operations of the IBM 1401 Data Processing system.

The language is presented in a special format that:

- describes generally each type of SPS statement
- describes specifically the construction of the statement
- describes generally the processing and assembly functions
- shows an example of how each statement can be used in a program.

The operating procedures are presented in detail with descriptions of deck assemblies, error notes, etc.

A sample problem shows a simple payroll listing including input and output documents, the SPS source program, and the output from assembly.

This SRL publication, C24-1480, obsoletes the IBM 1401 Data Processing System Bulletins: IBM 1401 Symbolic Programming System: Preliminary Specifications, J24-0200, and IBM 1401 Symbolic Programming Systems: SPS-1 and SPS-2, J24-1412.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments regarding the content of this publication to IBM Product Publications, Endicott, New York.

Contents

With the increasing capability of data processing systems, programming in the actual machine language of a system has become more complex. Not only does machine-language coding require memorization of a great many numeric and alphabetic codes but also the length and intricate design of programs written in machine language make them prone to logical and clerical errors.

Also, the problem of correcting errors in an actual machine-language program is intensified because of the difficulty in tracing the steps of a machine-language program to include corrections and relocate the problem in storage.

Symbolic programming, the use of mnemonic characters to write a program, has been developed to facilitate computer programming. When mnemonic instructions are used, data may be referred to in terms which are logical to the layman, as well as to the experienced programmer. Another advantage of symbolic programming is that the checking of each program may be performed by a person other than the programmer.

In a symbolic system, a routine to add current withholding tax to total miscellaneous deductions, subtract the total from gross pay, and store the amount as net pay might look like this:

Operation Code		Operands
ZA	CURWTX	ACCUM
A	TOTMDN	ACCUM
ZA	GROSS	NETPAY
S	ACCUM	NETPAY

The first instruction in this routine sets to zero a machine storage area, which is arbitrarily labeled ACCUM. It then adds the current withholding tax (an area called CURWTX) into this area. The next instruction, total miscellaneous deductions (TOTMDN), is added to the contents of ACCUM, which is CURWTX.

Then a storage location, labeled NETPAY, is set to zero, and gross pay (GROSS) is added into it. In the last step the contents of ACCUM (CURWTX + TOTMDN) is subtracted from the contents of NETPAY (GROSS). This puts (GROSS-CURWTX-TOTMDN) in a suitably labeled location (NETPAY), to which the programmer may refer later in the program.

Once the data and working areas have been defined and labeled, it is easier to follow the logic of this short routine written in symbolic form than to comprehend the same routine written in machine language, which might look like this:

Operation Code	Operands
P	060 S93
A	045 S93
P	050 A95
S	S93 A95

A program written in symbolic programming language (the source program) requires translation into an actual machine-language program (the object program) before a computer can execute it. This translation is done by a machine-language program called a *processor.* In general, the translation is "one-for-one." That is, for each instruction written in symbolic form, one machine-language instruction is produced.

The processor, sometimes called an assembly system, generally utilizes the machine for which the symbolic program is written. It analyzes all symbolic entries and converts them to actual machine operating data and instructions, establishing specified relationship between them.

As an additional feature, assembly programs also indicate various types of errors such as coding, out-of-sequence cards, etc. Symbolic programming saves time and simplifies coding. Because actual machine addresses of data and instructions are assigned automatically by the processor, the programmer need not concern himself with this detail. He can, however, refer to these addresses symbolically.

Once there is an agreement on label terminology, subroutines (short programs or routines common to a number of programs) may be easily incorporated in any program, and a major program may be written in independent parts with no loss of efficiency in the final program. Corrections and modifications of program instructions also entail no reassignment of addresses by the programmer. Finally, the automatic assignment of addresses makes programs and subroutines readily relocatable, i.e., they can be placed in varying machine locations as desired.

Because of the advantages in symbolic programming for the IBM 1401 Data Processing System, IBM has developed the 1401 Symbolic Programming Systems (SPS-1 and SPS-2).

IBM 1401 Symbolic Programming System

The IBM 1401 basic Symbolic Programming System, SPS-1, operates on the 1400-character machine with the 1402 Card Read-Punch and the 1403 Printer, but it can assemble programs for any object machine configuration up to 4000 positions of storage. An expanded Symbolic Programming System, SPS-2, can assemble programs for any size object machine (1,400 to 16,000 positions of storage), but requires assembly on a 1401 system that has at least 4,000 storage positions and the 1402 Card Read-Punch, and the 1403 Printer. The general description and operating procedure of both systems are the same; however, any characteristics that apply only to the expanded programming system, because of its larger machine storage requirements, will be noted throughout the manual.

Advantages of IBM 1401 SPS

Some significant advantages of the IBM 1401 Symbolic Programming System are:

- Simplifies program writing and organization. For example, it is easier to write and refer to an instruction such as S WHTAX GROSS (Subtract Withholding Tax from Gross) than to look up the addresses of withholding tax and gross for an instruction like S 599 618.
- Provides continuity for group programming efforts. Upon agreement of labeling terminology, program routines can be written independently and efficiently and can be combined for assembling because addresses are automatically assigned by the processor.
- Simplifies program adjustment. If programs require partial revision, only the affected routines need to be rewritten.
- Detects coding errors. Illegal operation codes, invalid addresses, sequence errors, etc., are detected by the SPS listing routine before the program is actually assembled.
- Facilitates program testing. Explanatory comments may be listed next to program instructions.

Programming with SPS

Effective programming in IBM 1401 requires a knowledge of the methods of programming in machine language. Before the programmer begins to code his program in symbolic language, (as when writing in actual machine language) he draws a block diagram of the procedure the program must take to accomplish a desired end result. From this block diagram he must determine which constants and work areas are needed.

Constants are fixed data, such as a standard FICA LIMIT calculation. *Work areas* are locations within core storage where the data can be manipulated, such as input and output areas, accumulator fields, etc. Then he writes the instructions for the program.

Symbolic Language

The symbolic language includes a standard set of *mnemonics*. These mnemonics are standardized abbreviations for operation code descriptions, and are usually easier to remember than the machine-language codes. For example:

Description	Mnemonic	Machine Language Code
Multiply	M	@
Clear Word Mark	CW	⌑

A list of mnemonic operation codes is shown in Figure 38. Also included as part of the symbolic language are standard methods for defining areas and entering constants, comments, etc.

By using the symbolic language, the programmer can control the locations of record and work areas if he so chooses, or he can leave this job to the processor program.

Processor Program

A standard deck of cards furnished by IBM contains the processor program that produces the *object program* (actual machine-language program) from the *source program* (symbolic language program). The SPS processor, also called an assembly program, assembles the object program from information given in the source program statements. The object program is then punched in one-instruction-per-card format. The punched output deck is then used to load the assembled machine-language program into core storage prior to its execution.

Information Requirements

The information that the processor program requires to assemble the object program is divided into three major categories:

Area Definition (Declarative Operations)
Instructions (Imperative Operations)
Processor Controls (Process Control Operations)

Area Definition

These statements assign sections of storage space for work areas. The assigned areas will be used by the object program and may contain the data to be processed and/or the constants required to execute the object program. Area definition statements in most cases do not result in instructions to be executed as part of the object program. However, the processor program does produce for these statements cards containing constants and their assigned machine addresses. These constants cards are loaded with the object program each time the program is used.

Instructions

Most of the statements on the program sheet are the instructions for the data processing job to be performed. These statements are translated by the processor program into their machine-language equivalents in the object program.

Processor Controls

These statements are special signals to the processor program. They allow the programmer to adjust certain portions of the assembly process. These statements are never executed in the object program.

These three types of information are presented to the processor program in the form of SPS statements written originally on a special Symbolic Program Coding Sheet.

Coding Sheet

The IBM 1401 Symbolic Program Coding Sheet (Figure 1) is fixed-form. A special field is provided for each item of information required by the processor. Each statement is written on a separate line.

Before assembly, each line of the coding sheet is key punched into a card. These cards make up the source program deck which is the input to the processor program.

Column numbers on the coding sheet indicate the punching format for all source program cards.

To facilitate key punching, IBM makes available a special card, electroplate C55369, for use with the 1401 SPS. This card is also used to contain the object program as it is punched as output from the assembly process.

The function of each portion of the coding sheet is explained in the following paragraphs.

Page Number (Columns 1 and 2)

This two-character entry provides sequencing for coding sheets. Only numerical characters may be used. Standard collating sequence for the IBM 1401 should be followed when sequencing pages.

Line Number (Columns 3-5)

A three-character line number sequences entries on each coding sheet. The first 20 lines are prenumbered 010-200. The third position is punched zero, the lowest number in the collating sequence. The six unnumbered lines at the bottom of each sheet can be used to continue line numbering or to make insertions between entries elsewhere on the sheet. The units position of the line number indicates the sequence of inserts. Any numerical character can be used, but standard collating sequence should be used. For example, if an insert is to be made between lines 020 and 030, it could be numbered 021. Line numbers do not necessarily have to be consecutive, but the deck should be in collating sequence, for sorting purposes.

The programmer should note that insertions can affect address adjustment. An insertion might make it necessary to change the adjustment factor in the operand of one or more entries. All insertions should be placed in their proper sequence in the source program deck before assembly.

Count Field (Columns 6 and 7)

The number of characters the assembled actual machine instruction or defined area is to contain is punched into this field. The processor uses this number to allocate storage locations for data and instructions. For example, if the count field of an area definition statement contains a 6, six storage locations will be allocated for that area. The processor will also use this number to assign an address for the area. See *Address Assignment.*

Because the processor can determine the length of an instruction from the information presented in the statement, the programmer can leave the count field blank for instruction entries. The processor will develop and punch the count in the count field of the object program deck in any case. For instructions, the processor will override any punching in the count field.

Label Field (Columns 8-13)

The label is a symbol or descriptive term selected by the programmer to identify the specific area or instruction represented by the source program statement in which the label appears. The label may then be used elsewhere in the program, i.e., in an operand of another source program statement, to refer to the area or instruction which it identifies.

It is advantageous to devise a label that suggests the meaning of the area or instruction, so that the source program can be easily interpreted by anyone concerned with the program. For example:

Type of Statement	Meaning	Label
Area definition	Withholding Tax	WHTAX
Instruction	Update	UPDATE

Labels are used only with area definition and instruction statements. Remember that core storage is allocated, and an address is assigned for all instructions and most area definitions. If the statement is labeled, the assigned address is known as the "equivalent address" of the label. The processor maintains during assembly a table of labels and their equivalent addresses. When a label appears in the *operand* of a statement, its equivalent address may be found and substituted for the label in the assembled statement.

The equivalent address of the label of an instruction is made equal to the leftmost, or high-order, core-storage position of those positions allocated to the instruction. The equivalent address of the label of a defined area or constant is made equal to the rightmost, or low-order, core-storage position of those positions allocated to the area or constant.

No two labels in the source program may be identical.

The label is punched beginning in column 8 of the label field. It can be as many as six alphamerical characters in length and the first, or leftmost, character must be alphabetic. No actual machine addresses or special characters should be used for this purpose. Blanks must not appear *within* a label.

Operation (Columns 14-16)

This field contains the operation code for the statement. Area-definition and processor control statements always have mnemonic operation codes. Instruction statements can have either mnemonic or actual operation codes. If the mnemonic op code is used, it is written and punched beginning in column 14 of the operation field. Actual op codes are punched in column 16.

Operands (Columns 17-27 and 28-38)

In the (A) and (B) operand fields are placed the programmer's designations of:

1. For instruction statements: the addresses of the

data to be operated upon, or the input-output units to be operated.
2. For area definition statements: the constant being defined, the address at which the constant is to be stored, or the address or input-output unit which is to be the equivalent of the label.
3. For processor control operations: the address which is to be used with the particular control of the processor being invoked by the programmer at that point in the program.

It is evident, then, that each entry in the operand fields serves one of the following purposes:
1. It designates a core-storage address.
2. It designates an input-output unit.
3. It provides the constants being defined.

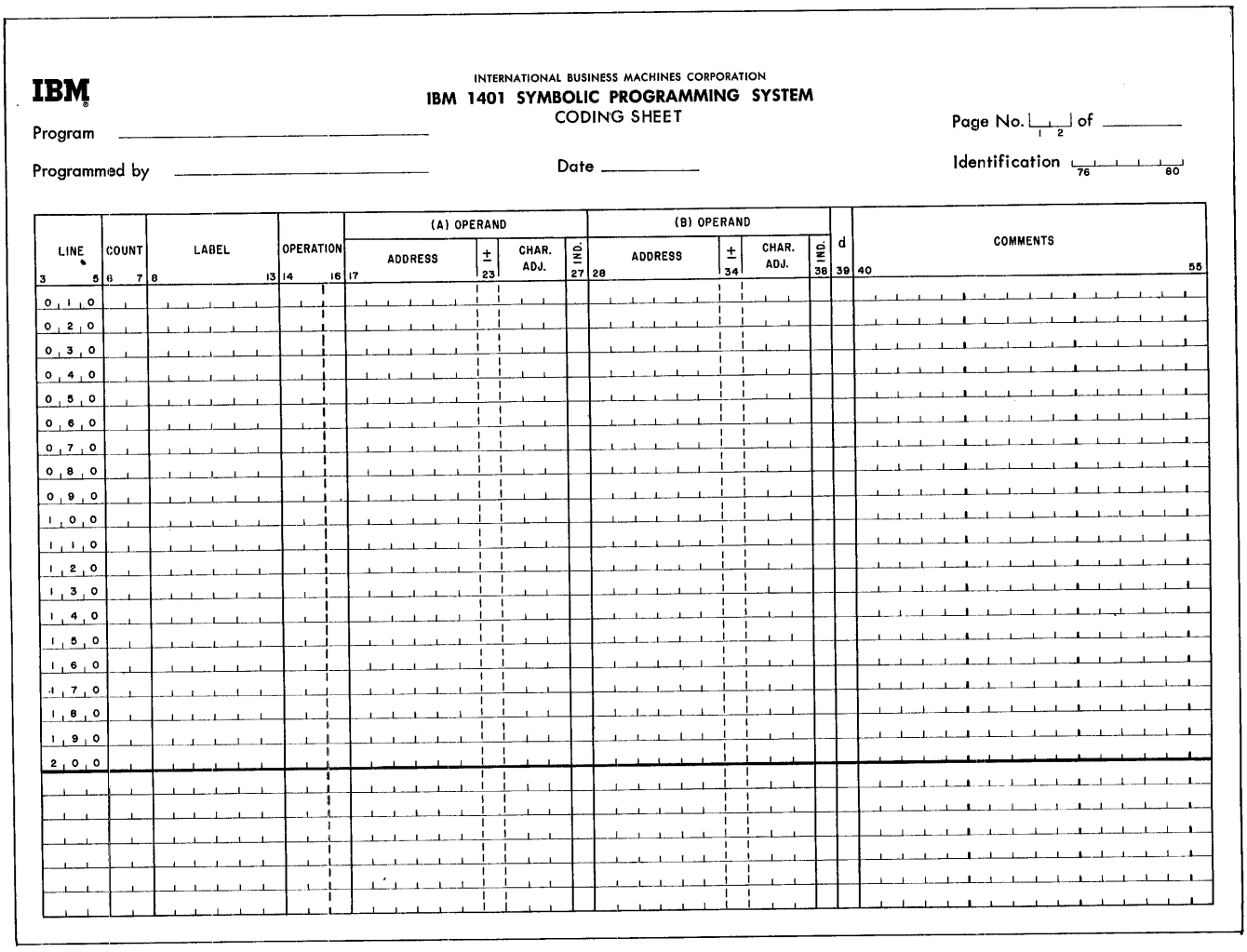
Core-Storage Address Operands

These are designated by using the Address, Character Adjustment, and Indexing portions of the operand field. There are four types of core-storage address operands: Symbolic, Actual, Asterisk, and blank.

SYMBOLIC ADDRESSES

A symbolic address can be composed of as many as six letters or digits (no special characters), but the first (high-order) character must be a letter. It refers to an instruction or area definition statement in the source program whose label is identical to it. For example, if ENTRYA is used as the label for an instruction in the source program, ENTRYA can be used as the symbolic operand of another instruction which references it such as B ENTRYA (Branch to the instruction whose label is ENTRYA).

Each symbol used must have a corresponding label because the processor substitutes the address assigned to the label wherever the symbol appears in the operand field of another source program statement. If character adjustment or indexing is associated with the


IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

Program _____ Date _____ Page No. ___ of _____

Programmed by _____ Identification ___ 76 ___ 80

LINE 3 5	COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	IND. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	IND. 38	d 39	COMMENTS 40 55
0 1 0													
0 2 0													
0 3 0													
0 4 0													
0 5 0													
0 6 0													
0 7 0													
0 8 0													
0 9 0													
1 0 0													
1 1 0													
1 2 0													
1 3 0													
1 4 0													
1 5 0													
1 6 0													
1 7 0													
1 8 0													
1 9 0													
2 0 0													

Figure 1. IBM 1401 Symbolic Program Coding Sheet

symbolic address, the processor will modify the address assigned to the label accordingly.

The programmer may write a symbolic address in a statement that precedes the labeled statement in the source program, because the processor does not assign addresses until all source program statements have been examined and the entire label table has been created.

ACTUAL ADDRESSES

In SPS-1 an actual address is a four-digit number written and punched beginning in column 17 of the A-operand field or in column 28 of the B-operand field.

In SPS-2 actual addresses can be either four or five digits, depending upon machine size, and are written and punched beginning in column 17 of the A-operand field or in column 28 of the B-operand field.

Figure 2 shows the use of one symbolic and one actual address in the source program.

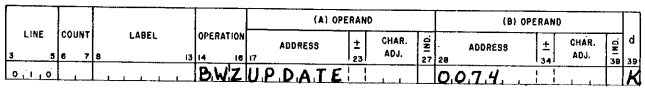


Figure 2. SPS Instruction with Symbolic and Actual Addresses

ASTERISK ADDRESSES

An asterisk address is the character "*" left-justified in the address field. Like symbolic addresses, it has a core-storage equivalent, but this equivalent address is simply the rightmost, or low-order, position in the instruction or data field defined by the statement.

Figure 3 shows an SPS statement with an asterisk address. In this example, the instruction was assigned the address 0906 during assembly. Because the instruction is seven characters in length, the low-order position of the instruction is 0912. The processor substitutes this address in the A-operand of the statement shown in Figure 3 and assembles it: M 912285. Thus, the actual machine-language instruction will appear in core storage as shown in Figure 4 after the object program has been loaded.

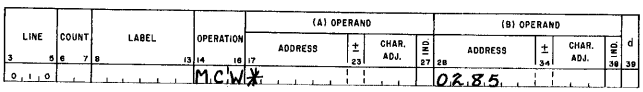


Figure 3. SPS Statement with an Asterisk Address

Character	M	9	1	2	2	8	5
Core-Storage Location	906	907	908	909	910	911	912

Figure 4. Assembled Instruction in Core Storage

BLANK ADDRESSES

Blank addresses are valid in instructions where no operand is needed. (See *Imperative Operations.*)

Input-Output Operands

The three-character addresses of tape units, disk storage units, and other input-output or storage units requiring special addresses are written left-justified in the A-operand field.

For example, the three-character address of tape unit 1 is the A-operand of the SPS statement shown in Figure 5.

Constant Operands

A constant operand is valid only in an area-definition statement which contains a constant to be loaded with the assembled object program. The constant should be written and punched exactly as it is to be loaded and stored. It must begin at column 24 and may extend to column 55.

Figure 6 shows a constant operand. The field to be defined is labeled FICALT. The constant is 480000.

Character Adjustment (Columns 23-26 and 34-37). It is possible to reduce the number of labels required in a source program by using a character-adjustment factor with an actual, asterisk or symbolic address in the operand field of an SPS source program statement.

The adjustment factor is written and punched as a plus or minus (& or –) in the ± column (23 or 34) and the number of positions of adjustment (as many as 3 digits) right-justified in the Char. Adj. field (columns 24-27 or 35-38).

During the assembly of the machine-language object program entry, this adjustment factor is added to or subtracted from the actual address assigned to the label by the processor. Thus, one label can

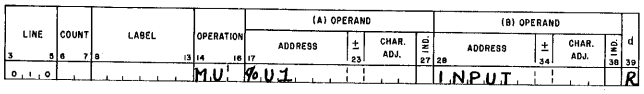


Figure 5. Addressing a Tape Unit in an SPS Statement

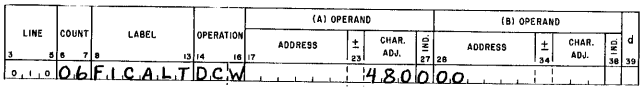


Figure 6. Constant Operand

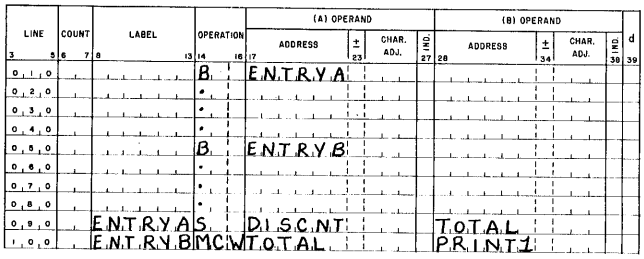


Figure 7. Two Labels Used

serve as a reference point for more than one address. For example, if ENTRYA is the label for an instruction which precedes the instruction, which could be called ENTRYB, the programmer can use ENTRYA as the reference point for ENTRYB.

If the instruction whose label is ENTRYA is 7 characters long, the symbolic entry operand ENTRYA +007 will produce an actual address equal to the address that would have been created if ENTRYB had been used as the label for the second instruction. Figure 7 shows a section of a source program in which two separate labels are used. Figure 8 shows the same section in which one label is used. Both sections produce the same result in the object program (Figure 9).

Figure 10 shows how character adjustment can be used to address a location within a labeled field.

DATE is a twelve-character constant. The character adjusted operand will cause only the first six digits of the date (i. e., DEC 28, rather than DEC 28, 1961) to be moved to 0243.

Because the number of labels used in a source program can have a significant effect on the amount of time required to assemble an SPS program, character adjustment is especially recommended in source programs which require a considerable amount of labeling. The number of labels that can be processed in one pass of SPS assembly depends upon the size of the processing 1401. Details are given in the operating section of this publication.

NOTE: The programmer must be careful when making insertions in a source program where character adjustment has been used. The insertion could necessitate changing the adjustment factor in one or more SPS statements. This same caution also applies to patching.

Indexing (Columns 27 and 38). If the advanced-programming feature is available in the machine that executes the object program (object machine), the programmer may indicate that an actual, symbolic, or asterisk address is to be indexed. He does this by writing the SPS index code in the index column of an SPS statement.

The index codes are 1, 2, and 3. A code 1 in an index column specifies that the address in the same operand field is to be indexed by the contents of index location 1 (core-storage locations 087-089) when the object program is executed. A code 2 specifies index location 2 (core-storage locations 092-094) and a code 3 specifies index location 3 (core-storage locations 097-099).

When the processor program encounters an indexed operand, it puts tag bits over the tens position of the assembled three-character machine address as follows:

Index Code	Tag Bits Tens Position	Zone Punch
1	A-bit	0
2	B-bit	11
3	A and B-bits	12

For example, the source program statement shown in Figure 11 specifies that the B-operand is to be modified by the contents of index location 1. The processor will assemble an instruction that will cause TOTAMT to be moved to a field whose address is equal to 596 plus the contents of index location at program execution time. If index location 1 contains 100 when the instruction is executed, TOTAMT will be moved to 696. Assume that the equivalent address of TOTAMT is 428. The machine-language instruction produced by the SPS processor program will be: M 428 5Z6.

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	(A) OPERAND ±	(A) OPERAND CHAR. ADJ.	(A) OPERAND IND.	(B) OPERAND ADDRESS	(B) OPERAND ±	(B) OPERAND CHAR. ADJ.	(B) OPERAND IND.	d
010			B	ENTRYA								
020			*									
030			*									
040			*									
050			B	ENTRYA	+	007						
060			*									
070			*									
080			*									
090		ENTRYA	S	DISCNT				TOTAL				
100			MCW	TOTAL				PRINT1				

Figure 8. One Label Used with Character Adjustment

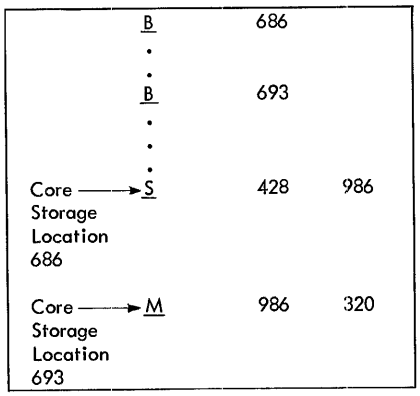


Figure 9. Assembled Instructions

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	(A) OPERAND ±	(A) OPERAND CHAR. ADJ.	(A) OPERAND IND.	(B) OPERAND ADDRESS	(B) OPERAND ±	(B) OPERAND CHAR. ADJ.	(B) OPERAND IND.	d
010			MCW	DATE	-	6		02.43				

Figure 10. Character Adjustment Used in Addressing a Location within a Field

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	(A) OPERAND ±	(A) OPERAND CHAR. ADJ.	(A) OPERAND IND.	(B) OPERAND ADDRESS	(B) OPERAND ±	(B) OPERAND CHAR. ADJ.	(B) OPERAND IND.	d
010			MCW	TOTAMT				0596			1	

Figure 11. SPS Statement Specifying Indexing

NOTE: Character adjustment and indexing entries are valid only with core-storage address operands, and then only when the address field of the operand is actual, symbolic, or asterisk. These entries are not valid with input-output unit operands or constant operands, nor are they valid when a core-storage address operand designates the address at which a defined area or constant is to be stored.

d-Character (Column 39)

Some 1401 instructions require a special modifier to the operation code called a *d-character.* It is a single alphabetic, numerical, or special character written and punched in column 39. The d-characters are always written in machine-language and are simply transferred by the processor to the d-character position of the assembled machine-language instruction.

Comments (Columns 40-55)

This field is reserved for programmer's notes or comments about a particular entry. A source program that contains a complete set of comments can be more easily understood and traced by all persons concerned with a given program. The comments have no effect on the object program as it is assembled or executed. Columns 56-75 of source program cards must be left blank, or incorrect processing will occur.

COMMENTS CARD

To provide the programmer with the ability to insert more extensive descriptive information in the program listing than is possible by using the comments field on a program entry card, a comments card may be included in the source program deck.

Comments cards will not be assembled nor will they affect the assembling procedure. When encountered by the processor, they will be reproduced unaltered in the SPS output deck, and will be bypassed when the object program is being loaded.

The Programmer:

1. Indicates with an asterisk in the first position of the label field (column 8) that the card is a comments card.
2. May write the comment beginning at any position (columns 9-55). Comments extending beyond position 55 may cause an error during processing.

The Processor: Reproduces (unaltered) the comment in proper sequence in the program listing.

Example: In the sample program, the entry in Figure 12 is a comments card entry.

Identification (Columns 76-80)

This field may contain any 1401 characters which the programmer selects to identify the program.

Address Assignment

To assign addresses to instruction and area-definition entries, the 1401 SPS processor uses a "storage assignment counter." This counter stands at 333 at the beginning of assembly (333 is the first available storage location beyond the standard read punch and print areas). However, the programmer can force the processor to begin assigning addresses elsewhere. See *Origin.*

During the first pass of assembly, the processor allocates storage for constants, work areas, and instructions. The amount of storage needed for each such entry is determined by the number in the count field. This number is simply added to the storage-address assignment-counter to develop addresses.

If the statement being processed has a label, the processor transfers it and its equivalent address to the label table.

During the second pass the processor converts these addresses to three-character machine addresses and stores them in the object program statements where corresponding symbols appeared in the operand fields of the source program statements. If character adjustment and indexing are specified, addresses are modified before they are stored.

Addresses for asterisk operands in instructions are determined by the address assigned to the low-order position of the instruction. This information is part of the loading data and is thus available when instructions are assembled for the object program deck.

NOTE: Actual addresses in area-definition statements do not affect the storage assignment counter and the count field in these statements is ignored. However, labels and the actual addresses associated with them are stored in the label table. The programmer must be careful that the locations specified in these statements are not the same locations that will be allocated by the processor to other statements in the source program. Otherwise, the processor will re-allocate them, and part of the object program will be destroyed at program load time.

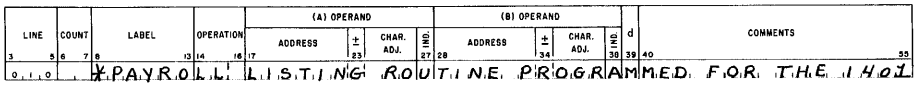


Figure 12. Typical Entry on an SPS Comments Card

Declarative Operations

(Area-Definition Statements)

The IBM 1401 SPS provides four different declarative operations for reserving work areas and constants.

Mnemonic Op Code	Purpose
DCW	Define Constant with Word Mark
DC	Define Constant (no Word Mark)
DS	Define Symbol
DSA	Define Symbol Address

DCW — Define Constant With Word Mark

General Description: A DCW statement causes a constant to be loaded into a core-storage area and a word mark to be set in the high-order position of this area at program load time.

The Programmer:

1. writes DCW in the operation field.
2. writes in the count field the number of core-storage positions needed to store the constant or work area.
3. writes a symbol in the label field if he wishes to refer later to the address of the field where the constant is stored.
4. writes the address of the area in which the constant is to be stored. If the programmer wishes to let the processor assign the address, he simply writes an asterisk (*) in column 17. Otherwise, he writes an actual address beginning in column 17. In any case the address will refer to the low-order (units) position of the defined area.
5. writes the constant beginning in column 24 of the coding sheet. The constant may extend to the end of the comments field (column 55). Thus, the maximum size of a constant is 32 core-storage positions.
6. may write a comment in columns 40-55 if these positions are outside the range of the constant itself as specified in the count field.

NOTE: In SPS-1, comments in DC and DCW statements are never listed. DCW and DC operands may not have character adjustment or indexing.

NUMERICAL CONSTANTS

A plus or minus sign can precede a numerical constant. A plus sign causes AB-bits to be placed over the units position of the constant; a minus sign causes a B-bit to be put there. The plus or minus sign is written and punched in column 23. If no plus or minus sign appears in column 23, the constant is stored at program load time as an unsigned field.

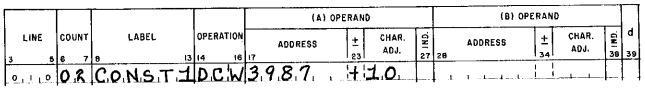


Figure 13. Numerical Constant (Signed)

Example: Figure 13 shows the numerical constant +10 defined in a DCW statement. The programmer has selected core-storage positions 3986 and 3987 as the location for the constant. It will be stored 1?.

Figure 14 shows an unsigned numerical constant (designed for use as a work area). In this example seven zeros are loaded to initialize the work area to 0000000. The asterisk indicates that the processor is to assign the address of the constant.

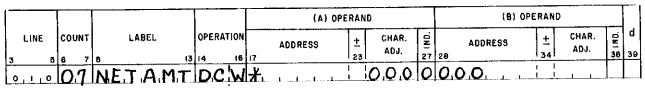


Figure 14. Work Area Defined by a DCW Statement

ALPHAMERICAL CONSTANTS

An alphamerical constant can consist of any valid 1401 characters. Alphamerical constants are always unsigned.

Example: Figure 15 shows the alphamerical constant DATE defined by a DCW statement. The programmer has selected 0499 as the address of the constant. Because DATE is now equivalent to address 0499, the constant can be referred to as either DATE or 0499. The constant will be loaded as DEC 28, 1961.

Figure 16 shows the alphamerical constant EDTWD1 defined in a DCW statement. The constant $bb,bb0.bb will appear in core storage as a field whose units position is determined by the processor.

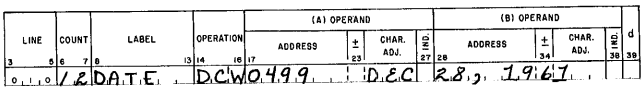


Figure 15. Alphamerical Constant

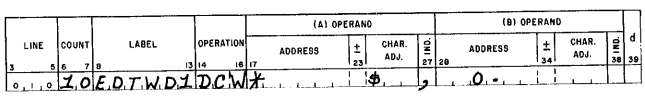


Figure 16. Edit Control Word Defined by a DCW Statement

BLANK CONSTANTS

A blank constant appears as blanks in the constant field (columns 24 through the length of the area as specified by the count field).

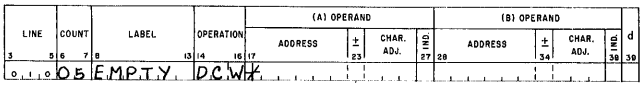


Figure 17. Blank Constant

Example: Figure 17 shows a blank constant bbbbb coded in a DCW statement and labeled EMPTY. The processor assigns the address.

The Processor:

1. allocates a field in core storage to store the constant. The number in the count field determines the number of positions allocated.
2. adds the number in the count field to the number that was standing in the storage assignment counter if there is an * in column 17. The result becomes the address of the constant. This address is made equivalent to the label, and the two are stored in the label table.

   If the programmer has specified the address, this address is equated to the label before it is stored in the label table. The count field is examined and used to create the loading data for the DCW card, but the storage assignment counter is undisturbed.
3. substitutes the equivalent addresses of labels in the operands of symbolic program statements which have corresponding symbolic addresses during assembly of the object program.
4. produces a card, as part of the object program, containing the data defined, with a sign if required, and instructions to load the constant into core storage with a word mark in the high-order position. This card is loaded with the object program, and the constant is stored exactly as the DCW source program statement defined it. For blank constants, the area is cleared of all existing word marks except the high-order word mark for the DCW constant.

DC — Define Constant (No Word Mark)

General Description: This statement is the same as a DCW statement except that the processor does not set a word mark in the high-order position of the constant.

NOTE: The storage area to which the constant is to be moved should be cleared of word marks.

DS — Define Symbol

General Description: DS statements cause the processor to assign equivalent addresses to labels or to assign storage for work areas. DS statements differ from DCW and DC statements in that no data is loaded into the defined area at program load time. A DS-defined area is unaffected during the loading of the object program. Data, word marks, instructions, previously put in the area, remain unaltered. Thus, if DS statements are used only to define areas, using a clear storage routine before loading the program is recommended.

Some DS statements affect the storage assignment counter. These can be used to bypass areas needed for independent routines (instructions or data not included in the source program being assembled) or for storing constants for which the programmer has selected the actual storage locations.

The Programmer:

1. writes DS in the operation field.
2. writes a symbol in the label field if he wishes to refer symbolically to the low-order position of the area.
3. writes the address of the area.

If the processor is to assign the address, he writes an asterisk in column 17 and writes a number in the count field to indicate the size of the area.

If the programmer wants to equate the label to an actual address or I/O unit operand, he writes the address or I/O operand beginning in column 17. In this case no storage will be allocated by the processor, so the count field is left blank.

NOTE: DS statements cannot be character-adjusted or indexed.

The Processor:

1. adds the number in the count field to the storage assignment counter and equates the resulting address to the label if one appears in the DS statement if there is an asterisk in column 17. If an asterisk or I/O unit operand appears in the DS statement, the processor equates the label to the operand.
2. substitutes the equivalent addresses of labels in the operands of symbolic program statements which have corresponding symbolic addresses during the assembly of the object program.
3. leaves the defined area unaltered during object program loading.

Examples: Figure 18 shows a DS statement with an asterisk address. This 8-position area which can be referred to as INPUTA will be assigned an actual address which will be the low-order core-storage position occupied by the area when the program has been loaded.

Figure 19 shows a DS statement used to equate a label to an actual core-storage address. (This type of DS statement is used for assembly only, and does not require any storage space in the object machine. Thus,

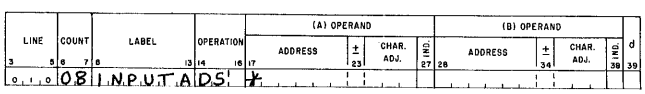


Figure 18. Defining an Area with a DS Statement

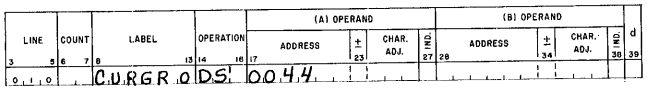


Figure 19. DS Statement Equating a Symbolic Address to an Actual Address

it is very useful in creating symbols for addresses that must be referenced often in the source program.) An input card contains current gross in card columns 40-44. When the card is read into 1401 core-storage, current gross will be in positions 0040-0044. The programmer wants to refer to current gross as CURGRO so that he does not have to remember the actual address (0044). The statement shown in Figure 19 allows him to use CURGRO as a symbolic address for 0044.

Figure 20 shows how a DS statement can label a tape unit. The programmer can now use TAPE1 instead of %U1 in the A-operand of a magnetic tape instruction.

Figure 21 shows how a DS statement can bypass 20 positions of core storage. The constants of the storage assignment counter will be increased by 20 when the processor encounters this statement. The processor will not re-allocate these 20 storage positions. However, an actual address in another area-definition statement or an instruction patch card can cause data to be loaded into them. For example, if the counter contained 0936 before this statement was processed, it would contain 0956 afterward. The DCW statement shown in Figure 21 would cause CONST2 (1875) to be loaded into storage positions 0953-0956 at object program load time.

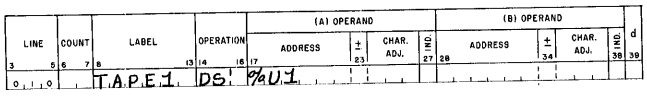


Figure 20. Assigning a Label to a Tape Unit

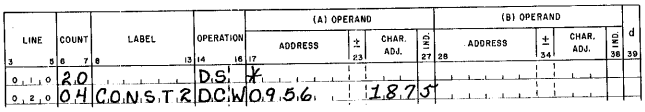


Figure 21. Advancing the Storage Assignment Counter and Using the Bypassed Area

DSA — Define Symbol Address

General Description: A DSA statement causes the three-character machine address which the processor has assigned to a label to be stored as a constant when the object program is loaded. This three-character address is called an *address constant.* Address constants are used to modify addresses during the *execution* of an object program.

The Programmer:

1. writes DSA in the operation field.
2. may leave the count field blank because an address constant is automatically assigned three core-storage positions by the processor.
3. may write a symbol in the label field if he wishes to refer later to the address of the address constant.
4. writes the address of the area in which the address constant is to be stored. If the programmer wishes to let the processor assign the address, he writes an asterisk (*) in column 17. In any case, the address will refer to the units position of the three-character address constant.
5. writes the symbol whose equivalent address is to be the address constant beginning in column 28 of the (B) operand field. This operand may have character adjustment and indexing.

The Processor:

1. allocates a three-position field in core storage in which will be stored the address constant at program-load time.
2. adds three to the number that was standing in the core storage assignment counter if there is an * in column 17. The result becomes the address of the address constant. This address is made equivalent to the label, and the two are stored in the label table. If the programmer has specified the address, this address is equated to the label before it is stored in the label table. The storage assignment counter is not affected when an actual address is used.
3. looks up in the label table, the equivalent address of the symbol in the B-operand and uses this address as the address constant.
4. produces a DSA card containing the address constant and instructions to load this constant into core storage with a word mark in the high-order position of the three-character field.

Examples: Figure 22 shows how a DSA statement can develop an address constant. In another part of his source program, the programmer has used the symbol WHTAX. During assembly, the processor will assign a three-character machine address to WHTAX, but the programmer does not know what that address will be. Yet for an address modification operation he needs this machine address (stored as a constant)

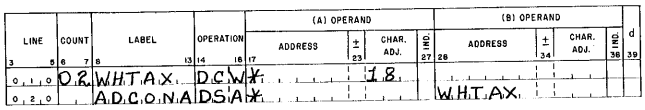


Figure 22. Area Definition Entries

so that he can write an instruction that will move it into the A- or B-operand of another instruction at program *execution* time. The programmer wishes to refer to the address of the address constant as a ADCONA.

For the example shown in Figure 22 assume that the storage assignment counter is standing at 584 when the DCW statement is encountered.

The processor will assign 586 as the address of WHTAX and 589 as the address of ADCONA. These two constants will appear in core storage as shown in Figure 23, after the object program has been loaded.

Figure 24 shows another section of the source program that uses ADCONA.

Assume that the equivalent address of INSTR is 829 and WORKAR has an equivalent address of 763. The processor will assemble these two instructions. M 589 832 A 000 763. When the first instruction is executed in the object program, the second instruction will be modified to A 586 763. When the second instruction is executed, WHTAX (18) will be moved to WORKAR.

Figure 25 shows a DSA statement with an actual address in the B-operand. In this case the programmer knows the actual address he wants to store as an address constant, but he does not want to bother to translate the actual address to a three-character machine address. The DSA statement causes the machine address of 12332 (33B) to be stored as an address constant in core-storage locations 14088-14090. The programmer can refer to the address constant as 14090 or as ADDRS1.

	WXTAX		ADCONA		
Character	1	8	5	8	6
Core-Storage Location	585	586	587	588	589

Figure 23. Constants in Core Storage

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d
010			MCW	ADCONA				INSTR	+	3		
020		INSTR	A	000				WORKAR				

Figure 24. Instructions Using an Address Constant

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d
010		ADDRS1	DSA	14090				12332				

Figure 25. DSA Statement with Actual B-address

Imperative Operations (Instructions)

General Description: SPS imperative operations are direct commands to the object machine to act upon data, constants, auxiliary devices, or other instructions. Thus, they are the symbolic statements for the instructions to be executed in the object program. Most of the statements in the source program will be imperative instructions. The programmer must be careful to write instructions that use only the features and devices that are included in the machine that will execute his program.

The Programmer:

1. writes the operation code for the instruction in the operation field. Mnemonic op codes are written left-justified in the operation field. Actual op codes are written in column 16. (Also, see *Coding Sheet*).
2. leaves the count field blank if he so desires.
3. writes a symbol in the label field if the instruction is an entry point for a branch instruction elsewhere in the program or, if he wishes to make other reference to it. This label is assigned an address equal to the core-storage location occupied by the operation code of the associated instruction at program load time. Thus, the programmer can use this label as a symbolic address in another SPS instruction.
4. writes in the (A) and (B) operand fields, core-storage addresses or I/O Unit Addresses.

Core Storage Addresses. These are symbolic, actual, asterisk, or blank addresses representing the A/I or B addresses of actual machine-language instructions. Symbolic, actual, or asterisk operands may have character adjustment and indexing.

NOTE: Blank operands are valid:

a. in an instruction that does not require an operand (such as a read instruction).
b. in instructions in which useful A- or B-addresses are supplied by the chaining method such as MCW FIELDA FIELDB MCW MCW.

If an instruction is to have addresses stored by other instructions, the operand or operands affected must not be left blank. Zeros are recommended as shown in the DSA example.

Input/Output Unit-Addresses. An I/O unit operand is valid only in the A-operand field of an SPS statement. It is the three-character address of an auxiliary device such as tape unit (%Ux).

5. writes the d-character in column 39 if one is needed for the instruction. The d-character is always written in actual machine language.

NOTE: Blank is not a significant d-character except in the Branch-If-Character-Equal instruction. A Branch-If-Bit-Equal (BBE) instruction with a blank d-character will be assembled as a seven-character instruction.

The Processor:

1. substitutes the actual machine-language operation code in place of the mnemonic operation code and transfers it to the operation-code position of the assembled machine-language instruction. Actual op codes are simply transferred as they are written in an SPS instruction statement.
2. counts the number of characters that will appear in the assembled instruction and adds this number to the number that was standing in the storage-assignment counter.
3. allocates a field in core storage that will be occupied by the assembled machine-language instruction.
4. stores the label (if one appears in the label field) and its equivalent address in the label table. Remember that an equivalent address assigned to a label in an instruction statement is the address assigned to the position occupied by the operation code when the instruction is loaded in the object machine.
5. looks up in the label table the equivalent addresses of symbols used in the operand fields of an instruction and inserts them in the actual machine-language instructions.

   Converts actual addresses to three-character machine addresses and transfers them to the machine-language instruction.

   Replaces asterisk addresses with the address occupied by the low-order position of the instruction in object-core storage, and transfers them to the machine-language instruction.

   Transfers an input-output unit operand to the A-address portion of the machine-language instruction.
6. produces a card, as part of the object program, which contains the machine-language instruction, and the information necessary to load it with a word mark in the op-code position.

NOTE: All instruction operations: Arithmetic, Data Control, Logic Control, and System Control are explained in the *IBM 1401 Data Processing System Reference Manual,* Form A24-1403. Operations requiring special features (noted by an asterisk in Figure 38) are also described in this manual. The programmer should thoroughly review the operation code functions before attempting to program in SPS.

Special Mnemonic Operation Codes

Three special mnemonic instruction operation codes are included for use with SPS-1 and SPS-2: Modify Address (MA), Load Unit (LU), and Move Unit (MU).

MA — Modify Address

The IBM 1401 MA operation code facilitates address arithmetic for systems equipped with more than 4,000 positions of core storage. It causes the data defined by the A- and B-operands to be added together and the result to be stored in the B-field at program execution time. Thus, a new address is developed in the B-field.

If the MA statement has an (A) operand only, the three-character machine address is added to itself and the result is stored in the A-field at program execution time.

The SPS-1 processor will accept the MA mnemonic operation code and assemble it as an A (Add) command, even though the modify-address feature is not available on 1400, 2000, and 4000 systems.

The SPS-2 processor will assemble the MA operation code as a modify-address command if the object machine has more than 4,000 storage positions. If there are 4,000 core-storage positions or fewer, it will assemble an A (Add) as in SPS-1.

Example: Figure 26 shows an SPS-MA statement coded to double the address assigned to ADCONA. After MA instruction is executed in the object program, the field whose address is ADCONA will contain +50, the machine address equivalent of 1050 (0525 + 0525).

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d
010		ADCONA	DSA	X				0525				
020			MA	ADCONA								

Figure 26. MA Statement with A-address Only

LU — Load Unit

The LU mnemonic is convenient to use for instructions that address magnetic tape units, RAMAC®, and other input-output devices. The processor produces a LOAD (L) instruction that will transfer data and word marks from the unit to the field whose address appears in the (B) operand. Figure 27 shows an LU statement that will produce an instruction that will read a tape record (with word marks) into core storage at program execution time.

MU — Move Unit

The MU mnemonic has the same function as LU, except that word marks are not transferred by the MOVE (M) operation it produces.

Figure 28 shows an MU statement that will produce an instruction to read information into storage from an IBM 1412 Magnetic Character Reader.

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d
010			LU	%UR				1020				W

Figure 27. LU Statement

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d
010			MU	%S1				INPUT				R

Figure 28. MU Statement

Processor Control Operations

The IBM 1401 Symbolic Programming System provides four processor control operations. These following commands, which are never executed in the object program, control the assembly process:

Operation Code	Purpose
CTL	Control
ORG	Origin
EX	Execute
END	End

CTL — Control

General Description: The control card is placed at the beginning of the source deck, so that the SPS processor is able to distinguish the storage sizes of the processing machine (machine which *assembles* the object program), and the object machine (the one that *executes* the assembled object program).

In SPS-1, the CTL card also signifies the availability of the punch-release feature to the processing machine. The punch-release feature is not used in SPS-2.

The Programmer:

1. writes the mnemonic code (CTL) in the operation field.
2. indicates in column 17 the size of the processor machine. This will determine the maximum number of labels that can be processed per iteration. See *Labels.*

SPS-1 Codes

Column 17 Code	Storage Position
1	1400
2	2000
3	4000

If a number other than one of these code digits is specified, if the card column is blank, or if the CTL card is omitted from the source program deck, the processor assumes a 1400-character machine.

SPS-2 Codes

Column Code	Storage Position
3	4,000
4	8,000
5	12,000
6	16,000

If a number other than one of these code digits is specified, if the card column is blank, or if the CTL card is omitted from the source program deck, the processor assumes a 4000-character machine.

3. indicates in column 18 the size of the object machine. This will indicate to the processor how much storage space will have to be cleared at load time for the assembled program.

For both processors, SPS-1 and -2 the machine codes are the same as previously listed. If column 18 is blank, the processor assumes the object machine size is the same as the processor machine. Also, in both SPS-1 and -2, the processor assumes the object machine to have 1400-character storage if there is an illegal code punched in column 18.

4. if he is using SPS-1, the programmer indicates in column 19 whether the punch release feature is available to the processor. This feature is used by pass one of the processor.

Column 19 Code	Meaning
1	Punch Release Available
blank	Punch Release Not Available

If any other digit is punched, the processor assumes no punch release feature.

The Processor interprets the machine size and feature codes and processes the source program accordingly.

ORG — Origin

General Description: An ORG statement causes the processor's storage assignment counter to assign addresses beginning at a particular location specified by the programmer. If it is entered as the first card of the source program, an ORG card can cause the initial assignment of addresses to be at a location other than 333. An ORG statement may be included at any desired point in the source program. This will cause the counter to be reset and cause all future entries to be assigned addresses beginning at the particular location designated by the programmer. Character adjustment and indexing are not valid in an ORG statement.

The Programmer:

1. writes ORG in the operation field.
2. writes the actual machine address at which assignment is to begin left-justified in the (A) operand.
3. inserts the card in the desired place in the program.

The Processor:

1. assigns addresses to instructions, constants, and work areas, beginning at the address specified in the (A) operand.
2. causes the storage assignment counter to assign subsequent addresses beginning at the address written in the (A) operand if an ORG statement is encountered at any point in the source program.

The first symbolic program entry following the ORG statement in Figure 29 will be assigned storage with location 900 as a reference point. For example, if the

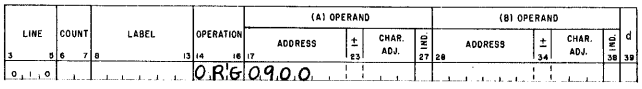


Figure 29. ORG Statement

first entry is an instruction, the op-code position of that instruction will be 900; if the first entry is a seven-character DCW, it will be assigned address 906, etc.

EX — Execute

General Description: During the loading of the assembled machine-language program, the programmer may wish to discontinue the loading process temporarily to execute a portion of the program just loaded. This can be accomplished through the use of an EX statement placed in the source program.

Using an execute command, the programmer can divide his program into several program sections if his total program exceeds the limit of available storage capacity.

The Programmer:

1. writes EX mnemonic operation code in the operation field.
2. writes a symbolic or actual address left-justified in the (A) operand. This indicates to the processor which instruction is to be executed after the loading process has been stopped. A blank or asterisk operand should not be used.
3. to continue the loading process after the desired portion of the program has been executed, the programmer must provide as the last instruction of the portion executed an instruction to read a card and branch to location 0056. This location contains an instruction to load the rest of the program.

   If the read area will be altered by the execution of the portion of the program, the programmer must provide, as the last instruction of the portion executed, instructions to clear the read area, and set word marks in locations 0024, 0056, 0063, 0067, as well as the read and branch operation as previously explained.

The Processor assembles a branch instruction. This instruction is not part of the object program, but it causes the loading operation to halt at the appropriate time. The branch instruction is then executed.

Example: It is sometimes desirable to execute the initial, or housekeeping, steps that are not necessary to continuous running or restarting of the program so that they may be then destroyed and new instruction or constants loaded over them. This routine, called an *overlay*, is executed as directed by an EX statement.

In the routine shown in Figure 30 the housekeeping instructions will be executed, the read area will be re-initialized, and the rest of the program will be loaded.

NOTE: The NOP instruction insures that a word mark follows the R0056 instruction.

The condensing routine output is compatible with this format. The routine, upon encountering an EX card, punches the cards necessary to re-initialize the read area for the condensed routine.

END — End

General Description: An END statement is a signal to the processor that the last card in the source program has been processed. If the programmer specifies in the (A) operand the actual or symbolic address at which the object program is to begin execution, an END statement will produce an instruction that will start program execution immediately after loading. If the (A) operand is blank, the 1401 will halt when the last instruction has been loaded.

The Programmer:

1. writes END in the operation field.
2. may write a symbolic blank, or actual machine address (left-justified) in the (A) operand. An asterisk operand is not permissible.

The Processor clears the read area (positions 001-080) of core storage and assembles an instruction that branches to the address specified in the (A) operand after loading is completed.

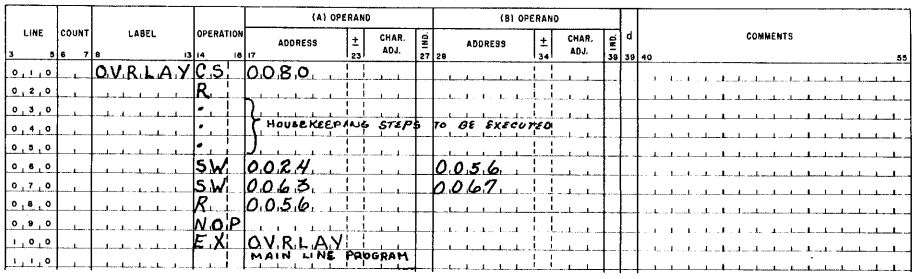


Figure 30. Using an EX Statement in Overlay Programming

The IBM 1401 Symbolic Programming System is composed of five separate programs:

- Pre-Process Listing Routine
- Processor – Pass One } Processor Assembly Program
- Processor – Pass Two }
- Post-Process Listing Routine
- Condensing Routine

The processor program listings and a table of address displays for 1401 halts is found in the IBM 1401 Program Library publication: *Symbolic Programming Systems, SPS-1 and SPS-2* (File Number 2.0.003).

Pre-Process Listing Routine

The SPS Pre-Processor Listing Routine makes it possible for the programmer to detect many coding or keypunching errors in the program deck before assembly.

The Operator:
1. puts sense switch A on.
2. places decks to be loaded behind the routine in the read hopper.
3. resets the 1401 and loads the program.

The Routine:
1. restores the printer carriage. All input cards to the program are selected to stacker 1.
2. prints the card image and messages into eleven fields on the printed page in the following format:

Page Number (Card Columns 1-2). Zeros are suppressed in this listing.

Line Number (Columns 3-5). Line number is printed as is.

Count (Columns 6-7). Zeros are suppressed in listing. The routine determines the count for instructions and DSA cards.

Label (Columns 8-13). The routine prints only the labels to be used by the processor. Thus, a label on an ORG card would not be listed.

Operation (Columns 14-16). This symbolic operation code is reprinted.

(A) Operand (Columns 17-27); *(B) Operand* (Columns 28-38). These are reprinted from the card except that there is a space between character adjustment and the indexing indicator. Only the digit position of the index appears.

d-Character (Column 39). This actual machine-language modifier is reprinted as is.

Location. Each time an ORG card is sensed, and at the end of the object program, the highest storage address used is printed in the location column. It is not printed for the first origin card unless it has been preceded by fields whose addresses are assigned by the processor.

SPS-1 Error Notes

In the pre-process listing, the processor may indicate in code, one of five errors under this field:

Err 1. Page-Line Sequence. Page or line number out of sequence.

Err 2. Count. Indicates illegal count for DC and DCW cards. If the count is greater than 32, only columns 23-55 are printed.

Err 3. Illegal Op Code. Indicates illegal mnemonic operation code. A CTL card in any other position than the first in the source deck will be processed as an instruction card and thus give this error.

Err 4. Illegal Operand. Indicates illegal operand. An instruction card containing a non-blank address with the high-order position blank, will cause this error. For DCW, DC, DSA, and DS statements, this error is indicated when the (A) operand is blank, symbolic, or % (except for DS).

Err 5. Column 56 Not Blank. Indicates column 56 is not blank. NOTE: *Information in columns 56-74 can cause improper processing when assembling.*

SPS-2 Error Notes

Because of increased storage available to the routine, additional error-checking features are included. The complete error legend is:

Err 1. Page-Line Sequence. Indicates that a page or line number is out of sequence.

Err 2. Count. Indicates the count for a DC or DCW is greater than 32 or less than 1, *or* that the programmer has not indicated the count for a DC, DCW, or DS statement.

Err 3. Label. Indicates that the first character in a label is blank, numeric, or a special character. Also recognizes a DS card without a label whose (A) operand is not an asterisk (*). This type of card is meaningless to the processor and is noted as a potential error.

Err 4. Illegal Op Code. Indicates illegal mnemonic operation code or a blank operation code field. A CTL card in any other position than the first in the source deck will be processed as an instruction card and thus give an illegal op code error. (These cards will be created as DCW's if there is a count.)

Err 5. Illegal (A) Operand indicates:

1. An instruction with a (B) operand but not (A) operand.
2. A blank or symbolic (A) operand for a DCW, DC, DS, or DSA statement.
3. A non-numeric address for an ORG statement.
4. An asterisk address for an EX or END statement.
5. In a general operand error:
   a. The indexing is not 1, 2, or 3.
   b. Character adjustment is non-numeric or left-justified.
   c. Character adjustment sign is not + or —.
   d. The first character of the operand is blank, but the balance of address is not.
   e. The operand begins with % but is greater than three characters. It does not appear to be an I/O device.
   f. The first character is numeric but the remaining are not.
   g. The numerical operand is fewer than four characters.
   h. The numerical operand is greater than the object-machine size specified.
   i. The first character of the address is the asterisk (*) but the balance is not blank.

Err 6. Illegal (B) Operand indicates the sign position (column 23) is not +, —, or blank in a DCW or DC statement or a general operand error. See item 5 under *Err 5).*

Err 7. Columns 56-74 Not Blank. The information in Columns 56-74 can cause an improper assembly.

Comments (Columns 40-55). The comments in a source program card are entered in this field on the printed page.

The preceding format is adhered to in the pre-process listing with the following exceptions:

1. Comments cards. The routine lists the comments (columns 8-55) centered in the middle of the page.
2. Constants. Constants are right-justified beyond the d-character column. The count determines the length of a constant. In this way, low-order blanks appear more distinctly. The sign (column 23) appears in the high-order position.

Additional Comments — SPS-1

In addition to the preceding listing, the pre-process routine will:

1. print at the end of the source program, the number of significant labels in the entire object program.
2. cause the highest storage address assigned by the processor (exclusive of the actual address assigned by the programmer) to be printed.

Additional Comments — SPS-2

The additional listing features for the SPS-2 pre-process routine are:

1. If the first card in the source deck is not a control card, NO CONTROL CARD is printed on the first line after the heading.
2. If the last card is not an END card, NO END CARD is printed below the listing.
3. At the end of the source program listing, the total number of cards and the highest storage address that the processor will assign (exclusive of actual addresses assigned by programmer) are printed.

End Card Stop. When the processor encounters an END card, the highest storage address and number of labels (and the card count, in the case of SPS-2) are printed. If there are more cards in the reader, the carriage is restored and the routine restarts at the beginning of the program. If sense switch A is on and it is the last card, there is a programmed halt. Pressing the start button restarts the program.

Processor Assembly Program

Because of the serial nature of the 1401, the processor must be a two-pass system (Figure 31). In general, pass one assigns all imperative and declarative statements

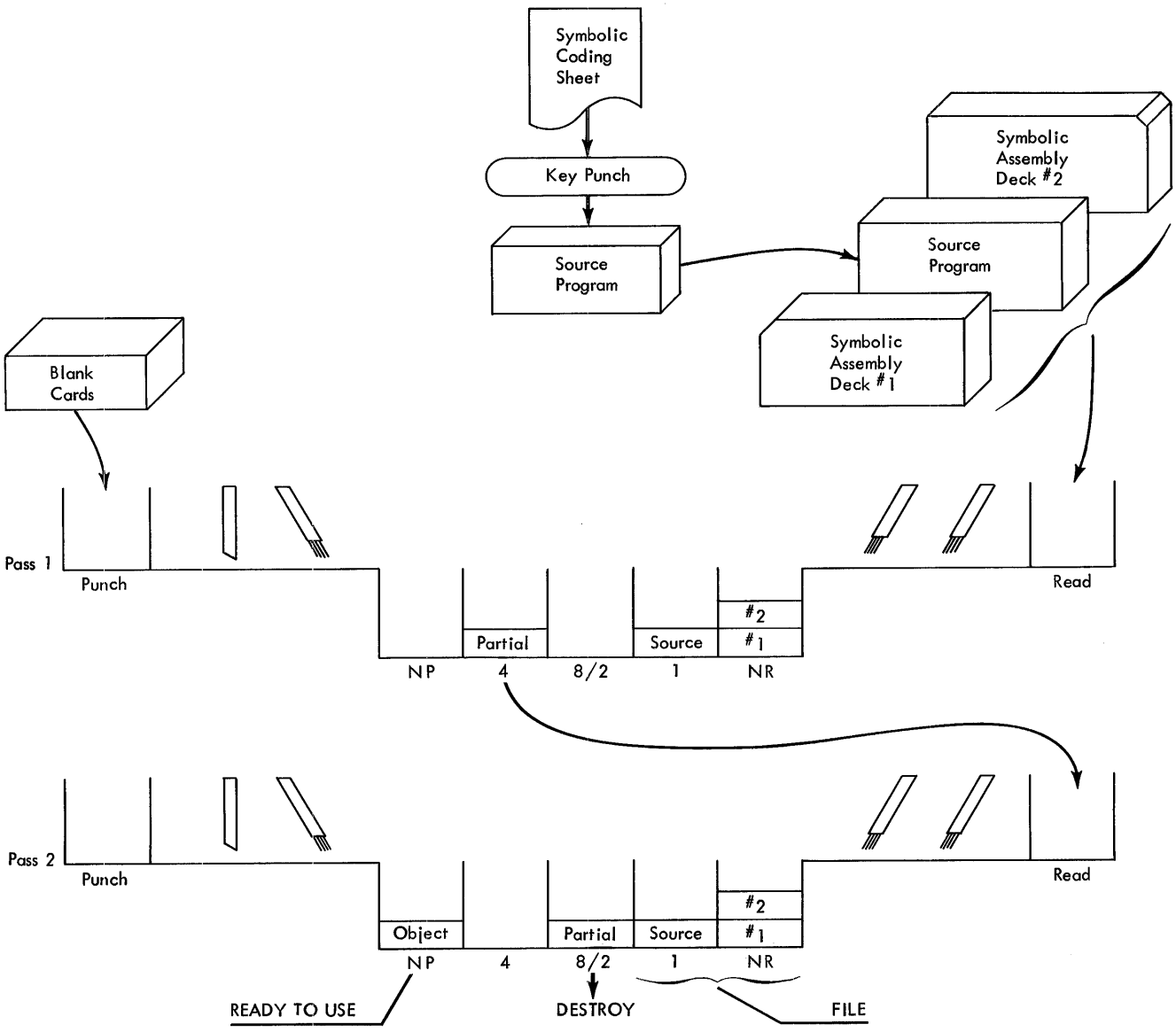


Figure 31. IBM 1401 SPS Assembly Procedure

an equivalent machine address and puts the label, if any, and its equivalent address into a table in storage. Pass two, upon encountering a symbolic operand, searches the table for the label and translates it into the machine-language equivalence.

Although one execution of passes one and two may be sufficient to assemble certain programs, it is likely others may require another iteration (complete execution of passes one and two) to complete the assembly. The number of iterations required to assemble a complete program depends upon the number of labels used in the source program and the storage capacity of the 1401 on which the program is being assembled. The number of labels that can be processed in one iteration is as follows:

SPS-1

Processor Machine Size	Number of Labels
1400	40
2000	100
4000	300

SPS-2

4000	260
8000	660
12000	1060
16000	1460

If the number of labels in a source program exceeds the quantity allowed, it is necessary to *reiterate*. Thus,

the output from the initial run of the processor becomes input to a second run. This is continued until all labels are processed.

Processing — Pass One

To prepare the source deck for processing, the operator must:

1. Be sure that the first card in the source deck is a CTL card, and that the last card is an END card (Figure 32).
2. Put the source deck between the processor program sections labeled PASS ONE and PASS TWO and put them in the read hopper.
3. Reset the computer and press the load button. The processor cards for pass one will be loaded into core storage, and program execution will begin automatically. During processor program execution, the SPS pass one deck will fall into the normal read stacker, the source deck into stacker 1, and the punched output deck into stacker 4.

During program execution the processor:

1. Changes the mnemonic operation codes to actual machine-language codes.
2. Assigns an address in core storage to each instruction and field designation.
3. Prepares a table of symbolic label and assigns an equivalent address to each label.
4. Allocates storage for instructions, work areas, and constants.
5. Punches out cards and drops them into stacker 4. These cards contain information that will be used by the processor during the second pass. The number of positions occupied by each instruction is automatically calculated and punched into the count field (columns 6-7) of these cards.
6. Loads automatically the processor program cards for pass two. This loading occurs if the END card of the source deck has been processed.

NOTE: Because pass one generates the symbol table, it is not possible to stop assembling at the end of pass one and continue with pass two at a later time. However, it is possible to postpone the completion of an assembly after any complete iteration.

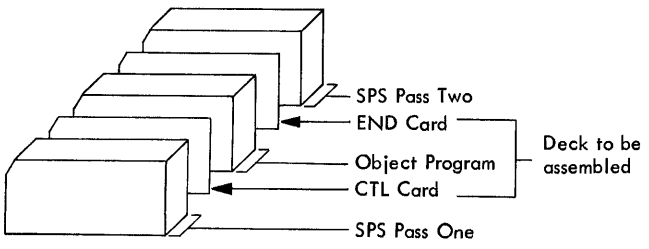


Figure 32. SPS Assembly Card Order

PROGRAMMED HALTS — PASS ONE

SPS-1 Illegal Mnemonic Operation Code. The processor causes the 1401 to halt, if the card just read (the last card in stacker 1) has an illegal mnemonic operation code.

If the punch release option is used and there is an illegal mnemonic, an output card is punched anyway. It is, however, selected to stacker 8/2 and should be discarded by the operator.

To restart the processor, the operator should:

1. press START to bypass the card, *or* repunch the card. Put it into the hopper as the first input card and replace the rest of the source program deck. Press START, *or* if it is an instruction that is not an eight-character branch-on-blank instruction, restart at the address noted in the IBM 1401 Program Library SPS listing. In the assembled deck, the actual operation code will be left blank. This error will be noted in the post-process listing.

SPS-2 Illegal Mnemonic Operation Code. The processor causes a halt if the card just read has an illegal mnemonic operation code. The operator should follow the same instructions as for SPS-1.

PROCESSOR STOPS WITH READER EMPTY

In pass one of SPS-1 and SPS-2, if all cards have been processed, and there is no programmed halt, the program may stop on a read operation code (the op register displays a *1*). This can occur in the following cases:

1. The program has not yet sensed an END card.
2. Pass one is completed and the program is trying to load pass two.

Processing — Pass Two

At the end of pass one, pass two is automatically loaded. It is possible, however, to load pass two by pressing the LOAD button.

During the loading of pass two, if *no* additional iterations are required, three cards will fall into stackers. The programmer should:

1. discard the card in stacker 8/2. This is a duplicate of the END card punched by pass one.
2. clear the storage routine of the assembled deck with the two cards in the normal punch stacker. Used at loading time for the assembled program, these cards will clear the amount of storage of the object machine which was specified on the CTL card.

After the loading of pass two, the operator:

1. places the output from pass one (stacker 4) in the read hopper when the loading of pass two has stopped.
2. presses the start button to resume processing.

During pass two the processor:

1. processes the operands by substituting actual machine locations for symbolic operands and performing character-adjustment operations.
2. changes numerical machine address assigned by the programmer into proper machine address when an alphabetic or special character is required by the 1401. For example, the core-storage address 1213 would be changed to S13 during this pass.
3. punches object program in a self-loading one-instruction-per-card format. Each card of the object program will contain the source program statement, the corresponding assembled instruction, or data, and the information required to load the assembled instruction.

   These object-program cards are selected into the normal punch stacker. The first two cards (see *Notes)* contain a self-loading clear-storage routine which clears core storage of all existing characters and word marks. The third card in the normal stacker, punched with the object program, will contain instructions to initialize the read area with word marks for the self-loading instruction cards. This is a bootstrap card (see *Processor Output).*
4. Causes the input deck (output of pass one) to be selected to stacker 8/2. These are to be discarded by the programmer.

ADDITIONAL ITERATIONS

If additional iterations are required, the machine will attempt to read a card (the first card of pass one).

If the 1401 attempts to read a card after pass two of the assembly program, an additional iteration is needed. If this is the case, the operator:

1. places the output from pass two (from normal punch stacker) between passes one and two.
2. places the combined deck into the read hopper.
3. presses the START button to repeat the operating procedure.

NOTE: For all additional iterations, all input cards are selected to stacker 8/2. These cards are to be discarded after assembly.

FIRST CARD STOP — PASS TWO

If pass one has been completed and the next card does not appear to be the first card of pass two, the SPS-2 processor will halt. To load pass two, the operator:

1. runs out the cards.
2. makes the first card of pass 2 the first input card.
3. resets the machine.
4. presses the LOAD button.

END OF JOB

After the last card has been processed by pass two, the machine will come to a halt if this is the last iteration. Pressing the START button at this point will have no effect.

PROCESSING STOPS WITH READER EMPTY — PASS TWO

In pass two of SPS-1 and SPS-2, if all cards have been processed and there is no programmed halt, the program may stop on a read operation code (a *1* in the op register). This occurs when:

1. the program has not yet sensed an END card.
2. the program has determined that reiteration is necessary and is trying to load pass one.

Non-Programmed Halts — Passes One and Two

The processor may come to an unexpected halt if the following errors are encountered:

1. END card misplaced (SPS-1 only). Because the processor automatically loads pass two after processing an END card, an END card placed in the middle of an object program will cause the card following it to be treated as the transfer card of pass two. This means that the page-line number is treated as an instruction and, if the tens digit of the page-line number is a valid input-output command, it will be executed prior to the halt. There is no convenient way to continue the assembly at this point because of the end-of-program procedure.
2. Illegal indexing indicator. If an indexing column (column 27 or 38) is punched with a special character, pass two will stop. The units position of the storage address displayed will show the special character with zoning removed.
3. Illegal DCW or DC count (SPS-1). If the processor encounters a DCW or DC card with a count greater than 57 (a legal DCW or DC card may only define a maximum field of 32 positions) and it is necessary to sign the constant (column 23 contains + or —) the sign will be placed somewhere with storage locations 081-122. However, locations 081-099 contain constants used by the processor. Consequently, it is possible for these constants to be changed by an illegal DCW or DC count in the object program. This error will be caught in the pre-process listing routine.

   For both passes, locations 081-099 should always contain:

STORAGE LOCATION	CONTENTS
081-089	01N001056
090	0 or 1
091-095	TLB10
096	T for 1.4k object machine Z for 2k object machine I for 4k object machine
097-099	The highest storage address of the processor (e.g. I99 for a 4k machine)

NOTE: In SPS-2, an illegal DC or DCW count will not interfere with locations 081-099 because the processor does not store any constants in these locations.

Miscellaneous Operating Information

1. Bypassing a card in pass one other than a comments card will cause an improper assembly. In pass two, if a card is bypassed, it will have no effect on the assembly of other cards.
2. It is not possible to reassemble an assembled deck. However, an assembled deck may be reproduced leaving columns 56 through 75 blank. The reproduced deck may then be reassembled.

Processor Output

The processor output, i.e., the assembled deck, consists of two clear-storage cards, a bootstrap card, and the assembled program.

CLEAR-STORAGE CARDS

The first two cards in the normal punch stacker are the clear-storage routine for the assembled deck. At loading time for the assembled deck, these cards will clear the amount of storage specified by the CTL card which designates the object machine size. The arithmetic overflow latch is set by the clear-storage card.

BOOTSTRAP CARD

This card, the third in the normal punch stacker, initializes the read area with four word marks necessary to make the assembled deck self-loading. The locations of these word marks are 024, 056, 063, and 067.

Assembled Program

The assembled program cards are duplicates of the input cards with the assembled information in columns 56-75.

Columns 56-75. In assembled instruction and constants cards, columns 56-62 contain the instruction necessary to bring the data into storage. An EX card assembles an unconditional branch in this field (columns 56-62) while an END card assembles a clear-the-read-area-and-branch instruction. All other cards generate a bypass instruction

N 001 056 (NOP 0001 0056).

Columns 63-66. All assembled program cards contain a 1056, the instruction R0056 in columns 63-66. This causes the 1401 to read a card and branch to execute the instruction in column 56 and the next card.

Columns 67-74. This field is blank except for instruction and DS cards. In construction cards, columns 67-74 contain the assembled instruction. In DS cards, columns 67-70 contain the four or five-character address of the symbol. This is saved by the processor for listing purposes.

Column 75. This column contains a card code used by the post-process listing and condensing routines to determine the card type.

CODE	CARD TYPE
blank	Instruction
blank	Comments
F	DCW
5	DC
G	DSA
D	DS
Q	EX
J	END
R	CTL
2	ORG

OUTPUT FORMAT

According to card type the assembled program output format is:

	COLUMN	CONTENTS
Instruction Card	56	L
	57-59	Low-order card position of the instruction (equals 66 plus instruction length).
	60-62	Low-order storage address
	63-66	1056
	67-74	the assembled instruction
	75	Blank
DCW, DC Card	56	L for DCW; M for DC
	57-59	Low-order card position (equals 23 plus constant length)
	60-62	Low-order storage address
	63-66	1056
	67-74	Blank
	75	F for DCW; 5 for DC
DSA Card	24-26	Assembled DSA
	56-59	L026
	60-62	Low-order storage address
	63-66	1056
	67-74	Blank
	75	G
DS Card	56-62	N001056
	63-66	1056
	67-71	4- or 5-character address of symbol
	72	Blank
	75	D
EX Card	56	B
	57-59	Assembled (A) Operand
	60-62	blank
	63-66	1056
	67-74	blank
	75	Q
EX Card (no [A] operand)	56-59	.063
	60-62	blank
	63-66	1056
	67-74	blank
	75	Q

END	56	/
	57-59	Assembled (A) Operand (if [A] is blank, 000 is assembled)
	60-62	080
	63-66	1056
	67-74	blank
	75	J
CTL, ORG, COMMENTS Cards	56-62	N001056
	63-66	1056
	67-74	blank
	75	R for CTL, 2 for ORG, blank for COMMENTS.

UNDEFINED SYMBOLS

Undefined symbols have ### (# is a 3-8 punch) substituted on the card for an actual address. For example, assume the symbolic instruction B LABEL1 appears in a source program. However, LABEL1 does not appear in the label field of any statement in the same source program. Therefore, B LABEL1 will be assembled B ###.

Patching

Correction or revision of an assembled deck is accomplished through a procedure known as *patching*. This makes it possible for the programmer to change the object program without having to reassemble the entire source program.

To prepare a patch card, the programmer:

1. inserts the count in columns 6-7. This is necessary for the condensing and post-process listing routine.
2. uses the following format according to card type:

FOR CONSTANTS:

a. punches the constant beginning in column 24.
b. punches in column 56 an L for DCW cards or an M for DC cards.
c. punches in columns 57-59 a zero followed by the number of the card column that contains the low-order position of the constant. This number is equal to the length of the constant plus 023.
d. punches in columns 60-62 the address of the core-storage location that will contain the units position of the constant.
e. places 1056 in columns 63-66.
f. punches in column 75, a 5 for a DC card or an F for a DCW card.

FOR INSTRUCTIONS:

a. punches the assembled instruction in actual machine language in columns 67-74.
b. places an L in column 56.
c. punches in columns 57-59 a zero followed by the number of the card column that contains the low-order position of the instruction. This number is equal to the length of the instruction plus 066.
d. punches in columns 60-62 the address of the core-storage location that will contain the units position of the instruction.
e. punches 1056 in columns 63-66.

FOR END CARDS:

a. punches a "slash" (/) in column 56, and 080 in columns 60-62.
b. punches the machine address of the first instruction to be executed after loading the object program in columns 57-59.
c. punches a J in column 75.

FOR EX CARDS:

a. punches a B in column 56.
b. punches the machine address to which the load program should branch in columns 57-59.
c. punches a Q in column 75.

The programmer places the patch cards in the assembled program before the assembled END (or EX) card.

Example: In a lengthy program, a programmer may wish to insert a statement or subroutine in the assembled program without having to reassemble. This may be accomplished by a patching procedure in which a branch instruction is substituted for another one in the main routine to cause a branch to the subroutine or instruction. The last statement in the subroutine must then cause a branch back to the main routine.

Suppose the programmer decides to insert an instruction after the branch on unequal compare operation which will move MANNO1 to FIELDA before the SW operation. The original assembled program post-process listing looks like this:

C MANNO1	0108	1126	C008108	
B ERROR		/1133	BS84/	This card removed from assembled deck.
SW 0109	0116	1138	,109116	

The subroutine would look like this in SPS and assembled instructions:

Storage Address	SPS Instructions			Assembled Instructions
1133	B	2084	/	B-84
2084	B	ERROR		/ BS84/
2089	MCW	MANNO1	FIELDA	M008208
2096	B	1138		B/38
2100	NOP (The NOP instruction guarantees that a word-mark is set after the unconditional branch.)			N

Each instruction is punched in *assembled form* in a patch-card format. The initial branch operation is substituted for the branch operation in the assembled program and the rest of the subroutine is placed before the assembled END, or EX card.

Post-Process Listing Routine

The Post-Process Listing Routine is used to list the object program after the assembly is complete. The programmer uses this listing to trace machine stops and assembly errors. The listing gives the assembled instruction as well as the original one.

The Operator:

1. puts the sense switch A on.
2. places into the read hopper any number of assembled decks to be listed.
3. resets the 1401 and presses the LOAD button.

The Routine:

1. automatically restores the print carriage. All input cards are selected to stacker 1.
2. prints the card image and error messages in twelve fields on the printed page in the following format:

Page Number (Card Columns 1-2). Zeros are suppressed in this listing.

Error indicator (□). A lozenge printed between the page and the line number indicates that at least one of the following errors appears on the card:

a. An undefined symbol in the (A) or (B) operand.
b. A blank operation code in the assembled instruction.
c. Data to be loaded has been assigned an address in the read area.
d. A constant has been assigned a count greater than 32.
e. The count column in a DC, DCW, or instruction card is blank or zero. This will cause a halt in the condensing routine. NOTE: A blank instruction count cannot occur as a result of assembling. However, it may have been inadvertently left out of patch cards.

Line Number (Columns 3-5). Line number is printed as is.

Count (Columns 6-7). Zeros are suppressed in this listing.

Label (Columns 8-13). The routine prints only the labels used by the processor.

Operation (Columns 14-16). The symbolic operation code is reprinted.

(A) Operand (Columns 17-27). This is reproduced from the card image except that there is a space between character adjustment and indexing. Only the digit position of the index appears.

(B) Operand (Columns 28-38). Reprinted same as (A) Operand.

d-Character (Column 39). This machine-language modifier is reprinted as is.

Location (Columns 60-62). This is the four-character address of the data. It refers to the high-order position for instructions and the low-order position for constants.

Instruction (Columns 67-74). The assembled instruction is printed in this field.

Comments (Columns 40-55). Comments in the source program card are entered in this field.

The preceding format is adhered to in the Post-Process Listing with the following exceptions:

1. Comments cards. The routine lists the comments (columns 8-55) centered in the middle of the page.
2. Constants. Constants are right-justified beyond the d-character column. The length of a constant is determined by its card count, and the sign (column 23) appears over the units position of the constant.

Each page has a heading line which identifies the respective columns. The identification (columns 76-80) appears at the extreme right of this line.

Unassembled-Card Listing. If a card does not appear to the listing routine as an assembled card, the word UNASSEMBLED CARD is printed out with a complete reproduction of the card data. This error is most likely to occur when the operator attempts to list an unassembled deck, fails to reiterate a partially assembled deck, or lists the output from pass one.

Clear Storage, Bootstrap Card Listing. At the beginning of the listing, the CLEAR STORAGE and BOOTSTRAP cards are reproduced. If any one card is missing, or if the cards are not in proper order, one of the following messages appears: FIRST CLEAR STORAGE CARD MISSING, SECOND CLEAR STORAGE CARD MISSING, NO BOOTSTRAP CARD. It is not necessary, however, that the CLEAR STORAGE cards be present when listing.

SPS-2 Listing Feature. In addition to the preceding listing, the SPS-2 processor prints, at the end of the program listing, the total number of cards, exclusive of clear storage and bootstrap cards, and the total number of errors detected by the routine.

End of Program Procedure. When the routine senses an END card and it is the last card, the program halts. Pressing the start button restarts the program. If there are more cards in the reader after an END card is processed, the carriage is restored and control transfers to the beginning of the program.

Condensing Routine

The SPS Condensing Routine is used to convert the one-instruction-per-card assembled program deck into a "condensed" deck which contains multiple instructions per card. While not considered part of the processor, the condensing program is supplied with the program to enable the user to reduce the number of cards in any program deck.

Each condensed card will contain up to 38 characters of information from the object program or seven word-mark locations, i.e., fields or instructions, whichever is reached first. The condensed card contains sufficient information to render it self-loading.

The condensing routine causes a self-loading card which initializes the read area with appropriate word marks and begins the loading process to be punched. If the two cards of the clear storage routine are entered into the condensing routine, they will be punched out before the self-loading load card.

As each card of the assembled object program is read, the assembled data is transferred to the condensed card-output field. The address of the first field placed on each card determines the starting location for the card.

The Operator:

1. places any number of assembled decks to be condensed behind the routine.
2. presses the LOAD button.

The Routine:

1. causes input cards to be selected to stacker 1. It causes output cards to be selected to the normal punch stacker.
2. causes the assembled deck to be punched out in a self-loading condensed deck. Each card has approximately six data fields in the following format:

CARD COLUMNS	MEANING
1-38	This field contains the data to be loaded into storage (maximum of 38 characters).
39-55; 63-70	Contains up to 6 four-character set-word-mark instructions. If there are fewer than six sw instructions, *1056* (R0056) appears in the next available set word mark location. This causes the loader to read a card and branch to the load instruction of the next card.
55	Contains a *1* denoting a *read-a-card* instruction. This instruction is present only if six set-word-mark instructions are present.
56-62	Contains the instruction necessary to load the data (columns 1-38) into storage.
71-74	B039 (Branch to 0039)
75	Blank
76-80	Identification

DC CARDS

In the condensed deck, the data is *loaded* into storage. This differs from the one-per-card loader in which a DC card is *moved* into storage. Consequently, if a DC is the first item in the data to be loaded by the condensed deck loader, a word mark is first set in the high-order position of the DC, but is then removed by a clear-word-mark instruction. Because data is *loaded* into storage, all existing word marks are cleared in the storage locations before entry.

EX CARDS

The programmer may insert data cards behind an EX card. To accommodate the condensed card loader, the condensing routine converts the instructions, which were written by the programmer to return control to the load routine after the program section has been executed. If data is not inserted, these cards will not affect the loading procedure.

END CARD

To facilitate patching of the condensed deck, the END card instruction is the only information present on the last card.

PATCHING

Patching a condensed deck requires that the data to be loaded in storage is placed in columns 1-38. The LOAD instruction is placed in columns 56-62, and 1056 is placed in columns 63-66.

PROGRAMMED HALTS

There are three programmed halts in each SPS Condensing Routine:

SPS-1

Illegal Card. The routine causes a halt if, in an instruction or constant card, any position in columns 57-59 is blank or if column 57 is not zero. This error cannot occur as the result of assembling but might appear on patch cards.

Blank Count Column. If the count column for instruction or constants is blank, the routine halts. This generally occurs because of a count left out in patch cards.

End of Job. Each time an END card is processed, the machine halts. After an END card is punched, the routine causes it to be selected to the normal stacker. A blank dummy punch-instruction card falls into stacker 8/2.

SPS-2

Non-Assembled Card. The routine halts if the input does not appear to be an assembled card.

Illegal DC, DCW Count. If the count field on a DC or DCW card contains a number greater than 32 or less than 1, the routine halts.

End of Job. This is the same halt as explained under SPS-1.

Restart After Error Halt. Pressing the start button after an error halt will cause the machine to bypass the error card and read the next card or, in the case of an END card halt, to restart the program.

To load and execute the assembled program, the operator

1. places the assembled deck (clear storage cards, bootstrap card, and assembled program) into the hopper.
2. puts data cards behind the assembled program.
3. presses the LOAD button to begin the loading procedure.

Figure 33 shows a block diagram for a payroll routine. A current earnings card and year-to-date card (Figure 34) are read by the 1401 and complete an employee's check and earnings statement (Figure 35).

Information from the year-to-date card is updated by the information in the current earnings cards and a new year-to-date card to be punched.

The first card read should be the current earnings card. If it is the year-to-date card (X74), there is a branch to the UPDATE routine, and the unequal comparison in man numbers results in a machine stop. In the first half of the program, the current information is edited and moved to the print area and also to work areas in the punch area. The first line of the

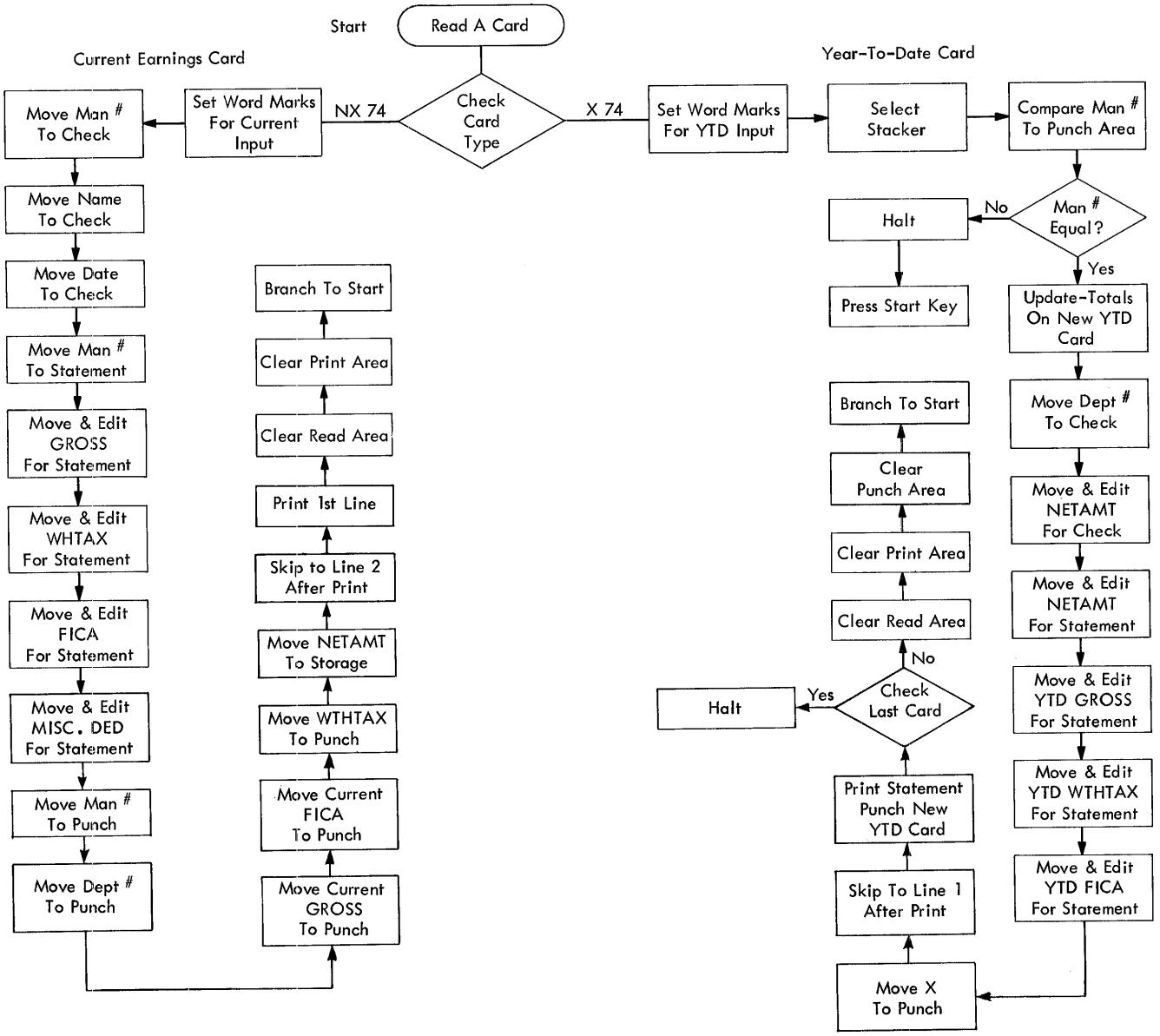


Figure 33. Block Diagram for Sample Problem

check and earnings statement is printed. The program checks for end of form.

The second half of the program, a routine labeled UPDATE, causes the year-to-date information to be added to the current information, which is in the punch area and causes the updated year-to-date card to be punched. The new year-to-date information is also edited and moved to the print area, and the second line of the check and earnings statement is printed. Then a skip to the next form is executed.

Figure 36 is an IBM 1401 SPS source program written for the payroll listing routine. Figure 37 is the SPS output program listing.

Mnemonic Operation Codes

Figure 38 shows the IBM 1401 SPS Mnemonic Operation codes.

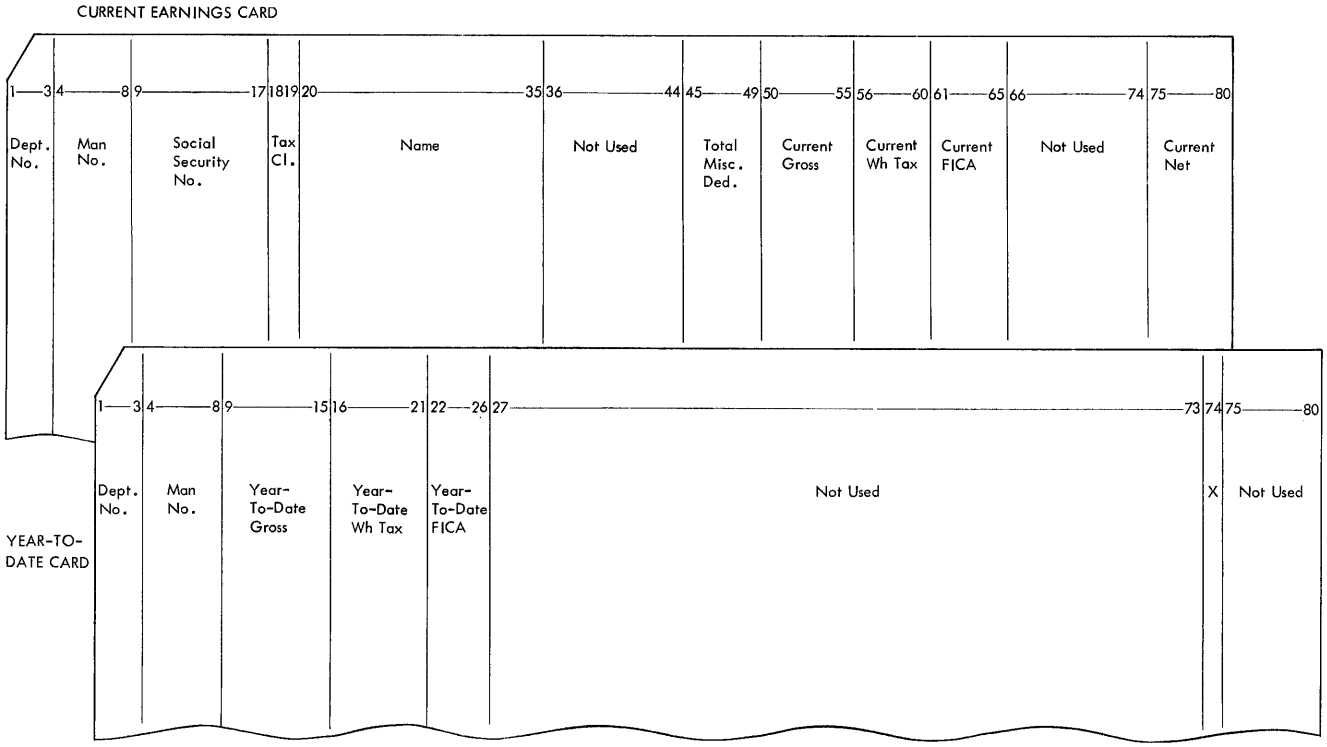


Figure 34. Current Earnings and Year-to-Date Cards

General Manufacturing Corporation

To the Order of: 23140 G J KANE

Date: JAN 27, 1961

724

Pay Exactly: $ 130.26

NEW VALLEY NATIONAL BANK

James Mercer Treasurer

EARNING STATEMENT

Man No.	Gross	WH. Tax	FICA	Misc. Ded.
23140	$ 175.00	$ 31.90	$ 5.66	$ 7.18

Net Pay	YTD Gross	YTD WH. Tax	YTD FICA
$ 130.26	$ 875.00	$ 159.50	$ 28.30

Figure 35. Employee Check and Earnings Statement

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL LISTING

Page No. 01 of 04

Programmed by

Date 7/18/62

Identification PAYRL

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			CTL	33									
020		*PAYROLL	LIST	ING ROUTINE				PROGRAMMED FOR THE 1401					
030			ORG	0900									
040		START	R										READ A CARD
050			B	UPDATE				0074				-	CHECK CARD TYPE
060			SW	0004				0009					MAN + SS NUMBERS
070			SW	0020				0045					NAME + MISC DED
080			SW	0050				0056					GROSS + WHTAX
090			SW	0061				0075					FICA + NETAMT
100			SW	0101									DEPT # IN PUNCH
110			MCW	0008				0206					MOVE MAN # TO CK
120			MCW	0035				0224					MOVE NAME TO CK
130			MCW	DATE				0241					MOVE DATE TO CK
140			MCW	0008				0255					MV MAN # TO STMN
150			LCA	EDTWD2				0266					
160			MCE	0055				0266					MV + EDIT GROSS
170			LCA	EDTWD2				0277					
180			MCE	0060				0277					MV + EDIT WHTAX
190			LCA	EDTWD2				0288					
200			MCE	0065				0288					MV + EDIT FICA

Figure 36. SPS Source Program (Part 1 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL LISTING

Page No. 02 of 04

Programmed by

Date 7/18/62

Identification PAYRL

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			LCA	EDTWD2				0299					
020			MCE	0049				0299					MV + EDT MISC DNS
030			MCW	0008				0108					MOVE MAN# TO PNC
040			MCW	0003				0103					MV DEPT# TO PNCH
050			MCW	0055				0115					MV GROSS TO PNCH
060			MCW	0065				0126					MV FICA TO PNCH
070			MCW	0060				0121					MV WHTAX TO PNCH
080			MCW	0080				NETAMT					SAVE NET AMOUNT
090			CC									B	SKIP 2 AFTER PRT
100			W										PRINT 1ST LINE
110		CLEAR	CS	0080									CLEAR READ AREA
120			CS	START				0299					CLR PRT + BRANCH
130		UPDATE	SW	0004				0009					MAN# + YTDGRS
140			SW	0016				0022					YTDWHTX + YTDFCA
150			SS									1	SELECT STACKER1
160			C	MANNO1				0008					COMPARE MAN#
170			B	ERROR								/	BRANCH UNEQUAL
180			SW	0109				0116					WORD MARKS IN
190			SW	0122									PUNCH AREA
200			A	0015				0115					UPDATE YTDGROSS

Figure 36. SPS Source Program (Part 2 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL LISTING

Page No. 03 of 04

Programmed by

Date

Identification PAYRL

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			A	0021				0121					UPDATE YTD WHTAX
020			A	0026				0126					UPDATE YTD FICA
030			MCW	0103				0206					MV DEPT# TO CK
040			LCA	EDTWD1				0241					EDIT NET PAY
050			MCE	NETAMT				0241					FOR CHECK
060			LCA	EDTWD1				0265					EDIT NET PAY
070			MCE	NETAMT				0265					FOR STATEMENT
080			LCA	EDTWD1				0277					EDIT YTD GROSS
090			MCE	0115				0277					FOR STATEMENT
100			LCA	EDTWD2				0288					EDIT YTD WHTAX
110			MCE	0121				0288					FOR STATEMENT
120			LCA	EDTWD2				0299					EDIT YTD FICA
130			MCE	0126				0299					FOR STATEMENT
140			MZ	0074				0174					MOVE ZONE TO PCH
150			CC									A	SKIP 1 AFTER PRT
160			WP										PRINT + PUNCH
170			CS	0080									CLEAR READ AREA
180			CS	0299									CLEAR PRINT AREA
190			CS	START				0180					CL PNCH + BRANCH
200		LASTCD	H	0001									LAST CARD HALT
161			B	LASTCD								A	CHECK LAST CARD

Figure 36. SPS Source Program (Part 3 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152-
PRINTED IN U.S.A.

Program PAYROLL LISTING

Page No. 04 of 04

Programmed by

Date

Identification PAYRL

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010		ERROR	H	ERROR									
020		MANNO1	DS	0108									
030	10	EDTWD1	DCW	*		$bb,		bb0-bb					
040	09	EDTWD2	DCW	*		$b,		bb0-bb					
050	07	NETAMT	DCW	*		0000		000					
060	12	DATE	DCW	0499		JAN		27,		1961			
070			END	START									
080													
090													
100													
110													
120													
130													
140													
150													
160													
170													
180													
190													
200													

Figure 36. SPS Source Program (Part 4 of 4)

```
CLEAR STORAGE 1     ,008015,022026,030034,041,045,053,0570731026
CLEAR STORAGE 2     L072116,110106,105117B101/I99,027A074028■027B0010270B026/0991,001/00111710
BOOTSTRAP CARD      ,008015,022029,056063/056029                              ,0240671056


                                                                                                PAGE  1

PG LIN  CT  LABEL   OP     A OPERAND        B OPERAND        D       LOC   INSTRUCTION COMMENTS

 1 010          CTL    33
 1 020          *PAYROLL LISTING ROUTINE PROGRAMMED FOR THE 1401
 1 030          ORG    0900
 1 040   1  START  R                                               0900  1              READ A CARD
 1 050   8         B      UPDATE           0074             -     0901  B  +81 074  -  CHECK CARD TYPE
 1 060   7         SW     0004             0009                   0909  ,  004 009     MAN & SS NUMBERS
 1 070   7         SW     0020             0045                   0916  ,  020 045     NAME & MISC DED
 1 080   7         SW     0050             0056                   0923  ,  050 056     GROSS & WHTAX
 1 090   7         SW     0061             0075                   0930  ,  061 075     FICA & NETAMT
 1 100   4         SW     0101                                    0937  ,  101         DEPT # IN PUNCH
 1 110   7         MCW    0008             0206                   0941  M  008 206     MOVE MAN # TO CK
 1 120   7         MCW    0035             0224                   0948  M  035 224     MOVE NAME TO CK
 1 130   7         MCW    DATE             0241                   0955  M  499 241     MOVE DATE TO CK
 1 140   7         MCW    0008             0255                   0962  M  008 255     MV MAN # TO STMN
 1 150   7         LCA    EDTWD2           0266                   0969  L  S74 266
 1 160   7         MCE    0055             0266                   0976  E  055 266     MV & EDIT GROSS
 1 170   7         LCA    EDTWD2           0277                   0983  L  S74 277
 1 180   7         MCE    0060             0277                   0990  E  060 277     MV & EDIT WHTAX
 1 190   7         LCA    EDTWD2           0288                   0997  L  S74 288
 1 200   7         MCE    0065             0288                   1004  E  065 288     MV & EDIT FICA
 2 010   7         LCA    EDTWD2           0299                   1011  L  S74 299
 2 020   7         MCE    0049             0299                   1018  E  049 299     MV & EDT MISCDNS
 2 030   7         MCW    0008             0108                   1025  M  008 108     MOVE MAN# TO PNC
 2 040   7         MCW    0003             0103                   1032  M  003 103     MV DEPT# TO PNCH
 2 050   7         MCW    0055             0115                   1039  M  055 115     MV GROSS TO PNCH
 2 060   7         MCW    0065             0126                   1046  M  065 126     MV FICA TO PNCH
 2 070   7         MCW    0060             0121                   1053  M  060 121     MV WHTAX TO PNCH
 2 080   7         MCW    0080             NETAMT                 1060  M  080 S81     SAVE NET AMOUNT
 2 090   2         CC                                         B   1067  F  B           SKIP 2 AFTER PRT
 2 100   1         W                                              1069  2             PRINT 1ST LINE
 2 110   4  CLEAR  CS     0080                                    1070  /  080         CLEAR READ AREA
 2 120   7         CS     START            0299                   1074  /  900 299     CLR PRT & BRANCH
 2 130   7  UPDATE SW     0004             0009                   1081  ,  004 009     MAN# & YTDGRS
 2 140   7         SW     0016             0022                   1088  ,  016 022     YTDWHTX & YTDFCA
 2 150   2         SS                                         1   1095  K  1           SELECT STACKER 1
 2 160   7         C      MANNO1           0008                   1097  C  108 008     COMPARE MAN#
 2 170   5         B      ERRCR                               /   1104  B  S52 /       BRANCH UNEQUAL
 2 180   7         SW     0109             0116                   1109  ,  109 116     WORD MARKS IN
 2 190   4         SW     0122                                    1116  ,  122         PUNCH AREA
 2 200   7         A      0015             0115                   1120  A  015 115     UPDATE YTDGROSS
 3 010   7         A      0021             0121                   1127  A  021 121     UPDATE YTD WHTAX
 3 020   7         A      0026             0126                   1134  A  026 126     UPDATE YTD FICA
 3 030   7         MCW    0103             0206                   1141  M  103 206     MV DEPT# TO CK
 3 040   7         LCA    EDTWD1           0241                   1148  L  S65 241     EDIT NET PAY
 3 050   7         MCE    NETAMT           0241                   1155  E  S81 241     FOR CHECK
 3 060   7         LCA    EDTWD1           0265                   1162  L  S65 265     EDIT NET PAY
 3 070   7         MCE    NETAMT           0265                   1169  E  S81 265     FOR STATEMENT
 3 080   7         LCA    EDTWD1           0277                   1176  L  S65 277     EDIT YTD GROSS
 3 090   7         MCE    0115             0277                   1183  E  115 277     FOR STATEMENT
```

Figure 37. SPS Output Program Listing

```
                                                                                                                    PAGE  2

PG LIN  CT  LABEL    CP    A OPERAND       B OPERAND          D                LOC   INSTRUCTION COMMENTS

 3 100   7          LCA   EDTWD2          0288                                1190  L S74 288    EDIT YTD WHTAX
 3 110   7          MCE   C121            0288                                1197  E 121 288    FOR STATEMENT
 3 120   7          LCA   EDTWD2          0299                                1204  L S74 299    EDIT YTD FICA
 3 130   7          MCE   C126            0299                                1211  E 126 299    FOR STATEMENT
 3 140   7          MZ    C074            0174                                1218  Y 074 174    MOVE ZONE TO PCH
 3 150   2          CC                                                  A       1225  F A          SKIP 1 AFTER PRT
 3 160   1          WP                                                          1227  6            PRINT & PUNCH
 3 161   5          B     LASTCD                                        A       1228  B S48 A      CHECK LAST CARD
 3 170   4          CS    0080                                                1233  / 080        CLEAR READ AREA
 3 180   4          CS    C299                                                1237  / 299        CLEAR PRINT AREA
 3 190   7          CS    START           0180                                1241  / 900 180    CL PNCH & BRANCH
 3 200   4  LASTCD   H     0001                                                1248  . 001        LAST CARD HALT
 4 C10   4  ERROR    H     ERROR                                               1252  . S52
 4 020      MANNC1   DS    C108                                                0108
 4 030  10  EDTWD1   DCW   *                                  $ ,  0.         1265
 4 040   9  EDTWD2   DCW   *                                  $ ,  0.         1274
 4 050   7  NETAMT   DCW   *                                      0000000     1281
 4 060  12  DATE     DCW   0499                           JAN 27, 1961        0499
 4 070                END   START                                             /  900 080

          68 CARDS
```

Figure 37. SPS Output Program Listing (Continued)

AREA DEFINITION			
	Mnemonic Operation Code	Description	
	DCW	Define Constant With Word Mark	
	DC	Define Constant (No Word Mark)	
	DS	Define Symbol	
	DSA	Define Symbol Address	
INSTRUCTIONS			
Type	Mnemonic Operation Code	Description	Machine Language Equivalent
Arithmetic	A	Add	A
	S	Subtract	S
	*M	Multiply	@
	*D	Divide	%
	ZA	Zero and Add	? (Prints as &)
	ZS	Zero and Subtract	! (Prints as -)
Data Control	MCW	Move Characters to A or B Word Mark	M
	*MCM	Move Characters to Record or Group Mark	P
	MCS	Move Characters and Suppress Zeros	Z
	MN	Move Numeric	D
	MZ	Move Zone	Y
	MCE	Move Characters and Edit	E
	LCA	Load Characters to A Word Mark	L
	SW	Set Word Mark	,
	CW	Clear Word Mark	¤
	CS	Clear Storage	/
	*MIZ	Move and Insert Zeros (for reading 7070 Compressed Tape)	X
	MA	Modify Address	#
	*SAR	Store A Address Register	Q
	*SBR	Store B Address Register	H
Logic Control	B	Branch	B
	BWZ	Branch if Word Mark and/or Zone	V
	C	Compare	C
	NOP	No Operation	N
	H	Halt	
	*BBE	Branch if Bit Equal	W
System Control	R	Read a Card	1
	W	Write a Line	2
	WR	Write and Read	3
	P	Punch a Card	4
	RP	Read and Punch	5
	WP	Write and Punch	6
	WRP	Write, Read and Punch	7
	*SRF	Start Read Feed	8
	*SPF	Start Punch Feed	9
	SS	Select Stacker	K
	CC	Control Carriage	F
	CU	Control Unit	U
	MU	Move Unit	M
	LU	Load Unit	L
PROCESSOR CONTROL OPERATIONS			
	Mnemonic Operation Code	Description	
	CTL	Control	
	ORG	Origin	
	END	End	
	EX	Execute	

*Pertains to an optional feature.

Figure 38. SPS Mnemonic Operation Codes

Index

IBM 1401 Printed In U.S.A. C24-1480-0

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]