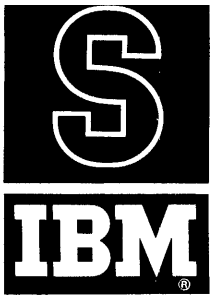


1401 DATA PROCESSING SYSTEM BULLETIN

IBM 1401 SYMBOLIC PROGRAMMING SYSTEM: PRELIMINARY SPECIFICATIONS

This bulletin is a minor revision of, but does not supersede, the original edition, form J28-0200. Principal changes are the addition of information regarding the output accompanying the object program (page 10) and the inclusion of Control and Execute commands as part of the Processor Control Operations (page 19).

In order to solve a given problem or perform a given function, the IBM 1401 Data Processing System must execute a logical sequence of instructions known as a program. To be meaningful, this program must be written in the actual language of the machine. A program written in this manner is said to be coded in machine language. This method of coding can be time-consuming and prone to clerical and logical errors. Also a hand coded machine language program, which may represent a great deal of programming effort, is not relocatable; that is, it cannot be executed from a different section of the processing unit without being manually changed. Another difficulty arises when instructions are inserted or deleted after a program has been coded. In this case, all instructions that refer to other instructions must be changed appropriately.

In order to simplify the preparation of a program for the IBM 1401 and to minimize the shortcomings mentioned above, the 1401 Symbolic Programming System was developed. Figure 1 is an illustration of a routine written for the 1401. Figure 1a shows the routine as coded in 1401 symbolic language. Figure 1b illustrates the listing of the same routine after it has been assembled into the machine language program.

This bulletin is devoted to a description of the preliminary specifications of the 1401 Symbolic Programming System. It also describes, briefly, the basic principles and concepts of symbolic programming in general. A knowledge of programming in actual 1401 machine language is presupposed. The listing, program deck and operating instructions for the symbolic programming processor will be made available at a later date.

The processor for the 1401 Symbolic Programming System is designed to operate on configurations of the 1401 Data Processing System equipped with a 1402 Card Read Punch.

TABLE OF CONTENTS

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	RES.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	RES.	d	COMMENTS
010			ORG	0500									
020	4		WMS	0001									
030	1	START	UR										
040	8		BZN	PRINT				0001				K	PRINT IF X-COL 1
050	7	PUNCH	MV	0080				0180					MV CARD TO PUNCH
060	4		UP	START									PNCH-BR TO START
070	7	PRINT	MV	0080				0280					MV CARD TO PRINT
080	4		UW	START									PRNT-BR TO START
090			END	START									
100													

1a

							LOCATIONS	ASSEMBLED MACHINE LANGUAGE INSTRUCTIONS	
1 010			ORG	0500					
1 020	4		WMS	0001			0500	, 001	
1 030	1	START	UR				0504	1	
1 040	8		BZN	PRINT	0001	K	0505	V 524 001 K	PRINT IF X-COL 1
1 050	7	PUNCH	MV	0080	0180		0513	M 080 180	MV CARD TO PUNCH
1 060	4		UP	START			0520	4 504	PNCH-BR TO START
1 070	7	PRINT	MV	0080	0280		0524	M 080 280	MV CARD TO PRINT
1 080	4		UW	START			0531	2 504	PRNT-BR TO START
1 090			END	START				B 504	

1b

Figure 1

BASIC PRINCIPLES AND CONCEPTS OF SYMBOLIC PROGRAMMING

In the early stages of data processing system development, programs were almost always written directly in the numerical and alphabetic notation (machine language) used by the system. As data processing systems developed, larger and more sophisticated machines were designed. Concurrently, programs for these machines became greater in length and complexity. As a result, programming efforts were greatly increased. Not only did it become difficult to memorize the many numerical and alphabetic codes required to write a program, but the length and complexity of the programs were conducive to increased numbers of clerical and logical errors. In addition, the problem of correcting errors in a program was intensified by the difficulty in tracing a machine language program through its many steps and finding a convenient way of including corrections.

In order to relieve the programmer of writing programs in machine language, symbolic programming systems were developed. Symbolic programming may be defined as a method wherein names, characteristics of instructions, or closely related symbols are used in writing a program. Data to be processed is referred to by name or other meaningful designation. Operation codes are written in an easily remembered mnemonic form rather than in the numerical language of the machine. For example, a payroll routine written in symbolic language to subtract the withholding tax from gross pay and store the net pay in another location might look like this:

Operation	Locations
ZA	GROSS FIRST
S	WHTAX FIRST
MV	FIRST NETPAY

The specific benefits derived from using easily recognizable symbols consisting of letters, numbers and special characters to represent corresponding elements of machine language, will depend to a large extent on the characteristics of the machine for which the system is designed. Generally, however, this feature makes it easier for persons unfamiliar with a program to follow the program's logic. It frequently eliminates the necessity of detailed flow charts and coding comments by providing, in the body of the instruction itself, the information usually found in these items. Also, coding is simplified, thereby decreasing the time required for coding and increasing coding accuracy.

Another advantageous feature of symbolic programming is the relative relationship between symbolic entries. Coding is facilitated by this feature as it permits instructions and data to be referred to before they are assigned actual machine addresses and without regard to their machine addresses. The feature permits sections of other programs or subroutines (short programs or routines common to a number of programs) to be easily incorporated in a program. It also enables each routine in a program to be written independently of the others with little or no loss of efficiency in the final program. Since instructions are not assigned storage locations by the programmer, the addition, deletion, modification or correction of instructions entail no reassignment of address. Finally, the feature makes programs and subroutines readily relocatable — i. e., they can be placed in varying machine locations as desired.

After a symbolic program is completed, it must be converted into the format required by the machine on which it will be used. Assembly programs are designed to accomplish this conversion quickly, accurately, and as automatically as possible. Usually, an assembly program utilizes the machine for which the symbolic program is written. The typical assembly program analyzes all symbolic entries and converts them to actual machine operating instructions and data, and establishes the specified relationships between them. As an additional feature, assembly programs also indicate various types of errors.

Although symbolic programming systems were originally developed to keep pace with the evolution of larger scale computers, their flexibility and many inherent advantages made them practical for use in smaller sized machines. The 1401 Symbolic Programming System is a symbolic programming and assembly system developed by IBM to simplify the preparation of programs for the IBM 1401 Data Processing System.

GENERAL DESCRIPTION OF THE 1401 SYMBOLIC PROGRAMMING SYSTEM

In general, the 1401 Symbolic Programming System can be divided into two separate areas of discussion: the symbolic language and the processor.

The symbolic language is the information with which a programmer codes the program. This language is in the form of mnemonic operation codes provided within the framework of the system, and also includes definitions of record areas, work areas and other data supplied by the programmer. Each separate item of information is written on a coding sheet with one entry per line. Entries are normally written in the order in which they are to be executed unless some other sequence is specifically indicated by the programmer. A program written in this manner, intended for translation into machine language is called a "source" program. After the program has been written in the symbolic language it is punched into program cards with one entry (one line of the coding sheet) per card. These cards then become the input to the processor.

The processor is the 1401 program which performs the actual functions of translation and assembly. The processor takes the source program in symbolic language, translates the mnemonic codes into machine language codes, assigns core storage addresses to instructions and symbolic data references, and assembles a finished machine language program known as the "object" program.

Should the need arise to change, revise, or rearrange the object program after it has been assembled, these changes can be incorporated by manually changing only those areas of the source program affected and then reassembling the corrected program.

The remaining pages of this bulletin are devoted to a detailed description of the 1401 symbolic language and processor. The organization is as follows:

1. The following two sections are concerned with a description of the coding form on which the source program will be written and a description of the organization of the processor. Operating instructions for the processor will be made available in a subsequent publication.

2. The next section called, "Programming the IBM 1401 using the Symbolic Programming System," describes in detail the various steps to be followed in coding a source program including features and restrictions of the 1401 symbolic language. It describes the manner in which storage areas and constants are defined and contains examples of all pertinent information.

A. 1401 SYMBOLIC PROGRAMMING SYSTEM CODING SHEET

All information relevant to the coding and subsequent assembly of the object program is entered on the 1401 Symbolic Programming System Coding Sheet. This form is illustrated in Figure 2. The information required by the 1401 falls into three categories.

1. Area Definition - These entries are used to reserve storage space for working areas. The areas may contain the data to be processed or they may contain the constants (fixed factors or combinations of characters) required in processing the data.

X28-1152-1

IBM 1401 Symbolic Programming System

Coding Sheet

Program _____ Page No. [ ][ ] (1 2) of _____

Programmed by _____ Date _____ Identification [ ][ ][ ][ ] (76 80)

LINE (3 5)	COUNT (6 7)	LABEL (8 13)	OPERATION (14 16)	(A) OPERAND ADDRESS (17)	± (23)	CHAR. ADJ.	RES. (27)	(B) OPERAND ADDRESS (28)	± (34)	CHAR. ADJ.	RES. (38)	d (39)	COMMENTS (40 55)
010													
020													
030													
040													
050													
060													
070													
080													
090													
100													
110													
120													
130													
140													
150													
160													
170													
180													
190													
200													

Figure 2

2. Instructions - Most of the entries on the program sheet will be the instructions, in symbolic language, which will be translated and assembled as the object program.

3. Processor Control Operations - Processor control operations are commands to the processor which provide the programmer with control over portions of the assembly process. Instructions of this type are never executed in the object program.

The following paragraphs explain the use of each field on the program sheet. The term "field", as used in connection with the program sheet, applies collectively to the character positions under each heading.

Heading Line

A line is provided at the top of the sheet to identify and date the program. The Page Number is a two-character entry which sequences the program sheets. This number, which must be numerical, will be punched in columns 1 and 2 of each card punched from this sheet. The Identification entry will be punched in columns 76-80 of each card in the source program deck. The identification entry may be any group of alphameric characters. The areas labeled Problem, Programmer and Date are not part of the source program and will not be punched.

Line Number (Card Columns 3-5)

A three-character line number sequences entries on each program sheet. The first 20 lines on the front of the program sheet are prenumbered 010-200. Since the units position of each of these numbers is 0, up to nine insertions may be made between each prenumbered entry. The six non-numbered lines at the bottom of the page are provided for the insertions and/or for sheet extension. Cards punched from insertions should be in their proper sequence in the source program deck prior to entering the cards into the machine.

Count Field (Card Columns 6-7)

The count field must be filled in by the programmer with the number of characters the assembled instruction or defined area will contain.

Label (Card Columns 8-13)

A label is a symbol chosen by the programmer to name an area being defined or an instruction referred to elsewhere in the program. All labels are assigned addresses in storage during assembly. A reference to a label in the program is a reference to the address of the area or instruction so labeled. Consequently, a programmer need not be concerned with actual memory locations. Only those items specifically referred to elsewhere in the program need have a label. Unnecessary labels delay the assembly process. Those instructions not referenced elsewhere in the program should contain a blank label field.

A label may consist of up to six alphameric characters, left-justified in the label field. The first character of the label must be alphabetic (A through Z). This type of label is known as a "symbolic" address. It is always to the best advantage to choose labels which are descriptive of the area or instruction to which they are assigned. Labels which have an obvious meaning not only provide easily remembered references for the original programmer, but also assist others who may assume responsibility for the program.

Operation (Card Columns 14-16)

The three-digit operation field will contain the mnemonic representation of the operation to be performed. These mnemonics will be left-justified. In the case of instructions, actual machine operation codes may be used but must be right-justified. A list of the mnemonic codes is found on page 22.

Operands (Card Columns 17-38)

The contents of the operands are the addresses or designations of the data to be operated upon. The two operand fields, (A) (Card Columns 17-27) and (B) (Card Columns 28-38) are each subdivided as follows:

a) six positions for the address
b) one position to denote character adjustment
c) three positions to indicate the amount of character adjustment
d) one position is reserved

Several types of addresses may be used in the operands (i. e., actual, symbolic, etc.). The types of addresses permitted in the operands and a complete description of character adjustment, including examples, are given where pertinent in subsequent portions of this publication.

"d" Modifier (Card Colum 39)

Certain types of instructions (i. e., conditional program transfers, test instructions, etc.) occasionally require the use of a digit known as a "d" modifier. In such cases, column 39 of the card must contain the actual character required.

Comments (Card Columns 40-55)

The comments field is provided where additional information concerning the program may be included if desired. The comments field may also contain constants if special constant areas are being defined. It is always to the best advantage to make complete use of the comments field. Since comments are ordinarily simply transcribed to an output program listing, they can become valuable aids in tracing a program. Also English language comments provide easy reference for all persons concerned with the program.

B. ORGANIZATION OF THE PROCESSOR

There are three main parts to the 1401 processor. They are:

1. Listing Routine
2. Pass 1 of Assembly
3. Pass 2 of Assembly

Listing Routine

The listing routine is a short editing and listing routine which may be used to print the source program before assembly. Its function is to check for possible errors such as invalid addresses, data inadvertently omitted, etc. While it is not absolutely mandatory to list the source program prior to assembly, it is wise to make use of this routine in order to check for coding accuracy and consistency. The listing routine may also be used to list the object program after assembly has been completed. This latter listing will contain one-instruction-per-line and each line will show an operation as written in source language, and as it appears after translation into 1401 machine language (see page 24). A listing of this kind is very helpful when debugging the program.

Itemized below are some of the functions of the listing routine. Entries which contain errors will be so indicated on the listing.

a) The identification field will be checked for consistency.

b) A sequence check of page and line numbers will be made.

c) The count field will be checked to see if instruction cards have the correct count and/or any blank operands.

d) The label field will be checked for illegal characters.

e) The operation field will be checked for an illegal mnemonic operation code.

f) The sign column of the operands will be checked for invalid characters.

g) The character adjustment field will be checked for numerical data only.

h) The number of storage positions required for the program will be totaled, and the highest address used by the program will be listed.

Pass 1 of Assembly

The assembly function of the processor is, basically, a two-pass operation. It may be necessary, however, for each pass to be repeated one or more times. The number of times each pass is repeated will depend upon the number of labels used in the source program. For example, if the source program contains 45 or fewer labels, the assembly process will be complete at the end of the two passes. Should the source program contain 65 labels, only the first 45 labels will be processed the first time passes 1 and 2 are executed.

The remaining 20 labels must then be processed by repeating passes 1 and 2. It is estimated that 45 labels will be adequate for a minimum of 150 instructions.

The input to the first pass is the source program punched with one entry (one line of the coding sheet) per card. The following functions will be performed during pass 1.

a) Operation codes are changed from mnemonic to actual machine notation.

b) Each instruction is assigned an address in core storage.

c) A table of symbolic labels is prepared and each label is assigned an address. By this process, storage is reserved for instructions, work areas, and constants.

d) Partially processed cards are punched out during this pass. These cards are the input for the second pass.

Pass 2 of Assembly

The following functions will be performed during pass 2.

a) Operands are processed. Symbolic operands are looked up in the symbol table for their equivalent locations. Character adjustment is performed, if required, to complete the operand.

b) Numerical addresses which have been used to represent those machine addresses in which an alphabetic or special character is required, will be assigned the proper machine address; e.g., address 1213 will be changed to S13.

c) The object program is punched out. Each card of the object program will contain the source program entry and the corresponding assembled instruction. A listing to aid in debugging may be obtained from these cards through the use of the listing routine. The first two cards punched with the object program will contain a self-loading clear storage routine. This routine will clear core storage of all characters and word marks; its use, prior to loading, is optional. The third card punched with the object program will contain a self-loading load routine to load the assembled object program into core storage. The load routine is loaded into and executed from positions 181-199. Use of these positions by the 1401 program must therefore be avoided.

PROGRAMMING THE IBM 1401 USING THE SYMBOLIC PROGRAMMING SYSTEM

This section describes in detail the various steps to be followed in writing a program for the IBM 1401 using the Symbolic Programming System. The material contained in this section has been divided into the three categories of information required to write a symbolic program: Area Definition, Instructions, and Processor Control Operations. In an effort to make this material more easily understandable, a simple payroll program is used as a theme from which many examples are extracted to illustrate pertinent parts of the text. The listing for this program is contained on page 24.

A. AREA DEFINITION — Storing Constants and Defining Work Areas

In the course of performing its given function, a program for the 1401 will generally require the use of "work areas," and/or "constants". A work area is a portion of core storage assigned as an area into which a record or part of a record will be transferred for processing. Areas such as these might be used for the accumulation of totals, or to assemble records. A constant is a fixed quantity or item of information which will remain the same throughout the course of the program or a phase of the program. For example, the FICA limit or a date may be considered constants.

The use of symbolic programming enables the programmer to refer to work areas and constants by their descriptive name without regard to their physical location within core storage. For example, the sample program used for reference utilizes two work areas and three constant factors. Of the two work areas, one is used to total deductions and the other is used to store the net amount of an employees pay for use later in the program. These two work areas are appropriately referred to as TOTALS and NETAMT. The three constants used consist of two fixed editing formats and a date. These are referred to symbolically as EDTWD1 (Edit Word 1), EDTWD2, and DATE. Thus, if the net amount is to be edited in the first format, the instructions would be written:

```
L    EDTWD1    0260 — (Load edit word 1 into location 0260)
E    NETAMT    0260 — (Edit net amount)
```

Note how, in this example, the programmer refers to the work area and constant by name and need not concern himself with the actual addresses of these areas. Note also that the programmer chose to use the actual address for the output area into which net amount was transferred. The reason for the latter is that the addresses of the fixed input and output areas afford such a direct correlation to either card columns or print positions that document layouts provide the proper actual address.

To reserve core storage space for work areas and to store constants requires the use of one of the four following types of cards.

	Operation Code	Purpose
a)	DCW	Define Constant With Word Mark
b)	DC	Define Constant (No Word Mark)
c)	DS	Define Symbol
d)	DSA	Define Symbol Address

DCW — Define Constant With Word Mark

Of the four types of cards mentioned above, the DCW will probably be the most commonly used. This is because it will load into the area designated, the exact information indicated on the input card and set a word mark at the leftmost (or high-order) position of the constant or work area. For this card, the mnemonic DCW must appear in the operation field of the card. The length of the entry being defined must be entered in columns 6 and 7. Core storage space will be allocated equal to the number of positions specified for each field.

The label (or name) by which the area or constant being defined will be known must be entered in the label field. The label may consist of from one to six alphabetic or numerical characters, the first of which must be alphabetic. The address at which this entry will be stored in core storage may be specified by the programmer, or the programmer may wish to let the processor assign the address. The address, if assigned by the programmer, must appear as a four-digit address left-justified in the (A) operand. If the processor is to assign the address, an asterisk must appear in the first column of the (A) operand. The constant must begin at column 24 and may extend to the end of the comments field (col. 55). Thus, a maximum of 32 digits is allowed for the constant.

Several of the DCW entries which appear in the sample program are explained below.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17	23		27	28	34		38
10	EDTWD1	DCW	*		$	b	0.			

stores a 10 digit constant ($ b b , b b 0 . b b) in core storage. The name of the constant is EDTWD1. Although the programmer will not know the location assigned to this constant (the asterisk indicates that the processor is to assign the address), he may refer to this constant throughout the source program as EDTWD1.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17	23		27	28	34		38
7	NETAMT	DCW	*		0000	0	000			

reserves an area in core storage, seven positions in length, for the accumulation of a net amount. The seven zeros, though not required, are included to initialize the field to 0000000. Again the asterisk indicates that the processor is to assign the address.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	RES. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	RES. 38
12	DATE	DCW	0499		JAN		27,1960			

will store the date, JAN 27, 1960, in core storage. The constant may be referred to as DATE or 0499 since the latter is the address assigned by the programmer.

In each of the above examples, a word mark is set at the high-order position of the field in core storage.

DC — Define Constant (No Word Mark)

A DC entry is identical to a DCW entry with one exception. That is, a word mark will not be set at the high order position of the instruction or working area being defined.

The DCW and DC cards used for constants will, during assembly, produce cards containing the constant and the machine address at which the constant will be stored. These cards will be loaded with the object program.

DS — Define Symbol

A DS entry may be used to define symbols (i. e., symbolic addresses or labels) used in the program and to reserve storage. The following examples illustrate its use.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	RES. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	RES. 38
10	CUSTNO	DS	*							

will reserve a 10-position field in core storage whose label or name is CUSTNO. The actual address assigned to CUSTNO will be determined by the processor. This 10-position area will be unaffected during the loading of the object program. Thus, data, word marks, etc., previously in this area will remain unaltered.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	RES. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	RES. 38
	CUSTNO	DS	0900							

will assign to the field CUSTNO the address 0900. When the address is specified, the length of field need not be indicated.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17	23		27	28	34		38
2 0		D S	*							

is an entry which will reserve 20 positions of core storage. A notation of this type must have an asterisk in the (A) operand. An illustration of the use of this type of notation will be given in a later section (see page 20).

DSA — Define Symbol Address

In order to store a constant which is equivalent to an actual core storage address assigned to a label, a DSA card may be used.

A DSA entry must contain a 3, right-justified, in the count field. The label field may contain the name of the location at which the constant will be stored or it may be left blank. The (A) operand must contain an asterisk or a machine address. The (B) operand must contain the label whose address is desired as the constant. The following example illustrates the use of a DSA:

Suppose a field, whose label (i. e. , name or symbol) is WHTAX, is assigned the machine address of 1219 by the processor. The programmer decides to store this address as a constant in a field he designates as FIELDA. During coding, however, the programmer will, in all likelihood, be unaware of the address to be assigned to WHTAX. Thus, the following entry, in which the constant is referred to as WHTAX, is used.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17	23		27	28	34		38
3	F I E L D A	D S A	*				W H T A X			

The constant S19 will be stored in the location labeled FIELDA. (S19 is the actual machine address equivalent to 1219, the address assigned to WHTAX by the processor.)

In the following illustration, FIELDA will be assigned address 0900.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17	23		27	28	34		38
3	F I E L D A	D S A	0 9 0 0				W H T A X			

The result is that machine location 0900, which may be referred to elsewhere in the source program as FIELDA, will contain S19, the address of WHTAX.

If the constant is not referred to elsewhere in the program, the label field may be left blank.

A DSA card will produce a card containing the constant and the machine address at which the constant will be stored. These cards will be loaded with the object program.

B. 1401 SYMBOLIC PROGRAMMING INSTRUCTIONS

The preceding section discussed four types of entries which provided the object program with the work areas and constants it requires to accomplish its assigned task. These four types of entries will never produce instructions which are executed in the object program. This section discusses the operations (instructions), written in symbolic language, which will be translated by the processor into 1401 machine language.

All instructions which can be performed by the IBM 1401 are valid input to the processor. Instructions are written on the program sheet, one instruction per line, in the exact sequence in which they are to be executed. The various data to be included within each field of the coding sheet is described on pages 5 to 8. The following material illustrates the various symbolic language instructions.

Labels

An instruction may be labeled, (i. e., assigned a symbolic name) by placing a symbol in columns 8 to 13 of the coding sheet. In general, an instruction which has been labeled will be an instruction referred to elsewhere in the program. Thus, it will be possible for another instruction to refer to the labeled instruction by its symbolic name.

The following illustration has been extracted from the sample program. This program contains three routines: START, UPDATE, and ERROR.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	RES. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	RES. 38	d 39
1	START	UR									
8		BZN	UPDATE				0074				K
7		CLS	START				0299				
7	UPDATE	WMS	0009				0016				
5		B	ERROR								/
7		CLS	START				0180				
1	ERROR	H									

Each input card to the above program contains a code, in column 74, which determines which routine will be executed. Note the instruction which tests the code and branches, if necessary, to the UPDATE routine. Note also in the UPDATE routine the instruction which branches to the error routine is merely B ERROR. At the conclusion of the START and UPDATE routines, the program returns to START.

Operation

The operation field may contain the mnemonic representation of the 1401 instruction or the actual machine operation code. The mnemonics must be left-justified and the actuals right-justified.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	RES. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	RES. 38
7		WMS	0021				0039			
7		,	0021				0039			

Both instructions shown above will cause the object program to set word marks in locations 0021 and 0039.

Operands

The entries in the operand fields are the addresses or designations of the data to be operated upon. Several types of addresses may be used in the operands.

Symbolic

A symbolic operand must correspond to a symbol used elsewhere in the source program. The operands may refer to another instruction, the name of a constant or of a work area. A symbolic address may contain from one to six letters or digits (no special characters) the first of which must be a letter. The use of symbolic addresses has been illustrated in previous examples.

Actual

An actual address must be a four-digit number, left-justified in the address portion of the appropriate operand. A three digit number must be preceded by a high order zero, e.g., 359 must be written 0359. Addresses which contain alphabetic or special characters are also written as four-digit numbers. Thus /67 will be written 1167 and S50 is written as 1250. The processor will convert these addresses to their proper machine notations.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND ADDRESS 17	± 23	CHAR. ADJ.	RES. 27	(B) OPERAND ADDRESS 28	± 34	CHAR. ADJ.	RES. 38
7		ZA	0065				1247			

The preceding instruction will be assembled as $\overset{+}{0}$ 065 S47. This feature allows the programmer to refer to locations as they appear logically, and places the burden of converting numbers to special characters on the processor.

Blank

Operand fields for instructions which require no operands should be left blank. These operands will appear as blank in the output deck and listing, but, because of the variable length feature of 1401 instructions, will not occupy unnecessary space in core storage.

Asterisk

The use of asterisks in the Area Definition type entry has already been illustrated. In this case it is merely an indication to the processor that the entry is to be assigned an address. An asterisk, used as an operand of an instruction, is an indication that this operand designates the rightmost or low-order address of the instruction in which it appears. For example, suppose the following instruction were assigned the address of 0706 during the assembly process.

COUNT 6 7	LABEL 8 13	OPERATION 14 16	(A) OPERAND 17				(B) OPERAND 28			
			ADDRESS	± 23	CHAR. ADJ.	RES. 27	ADDRESS	± 34	CHAR. ADJ.	RES. 38
7		A	*				0285			

Since the instruction is seven characters in length the rightmost or low-order position of the instruction will be 0712. Thus, the assembled instruction will be A 712 285. The asterisk must be left-justified in the operand and may be character adjusted only if it appears in an instruction entry. Asterisks in Area Definition entries may not be character adjusted.

Character Adjustment

Any core storage position within a designated field, area or instruction may be addressed by the use of character adjusted operands. With the exception of the restriction mentioned above, all operands may be character adjusted. The character adjustment factor is always written after the operand to which it applies. The number and direction of character adjustment is indicated by a plus or minus sign followed by the factor, which may be up to three digits in length.

Character adjustment could conceivably reduce the number of labels required by the source program by giving the programmer the ability to refer to a location which is a given number of locations away from a symbolic, actual, or * address. The following examples illustrate its use.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17 23			27	28 34			38
7		MV	0285				*	+	7	

The above instruction will, when assembled, modify the instruction which immediately follows it. If the instruction shown is assigned address 0576 then the asterisk is equivalent to 582, and * + 7 = 589, which is the right-most position of the succeeding instruction assuming the succeeding instruction is seven characters in length. The illustrated instruction will be assembled as M 285 589.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17 23			27	28 34			38
4		B	START	+	21					

If START has been assigned address 0900, then the operand START +21 will be assembled as 921.

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17 23			27	28 34			38
7		MV	DATE	-	6		0243			
				.						

DATE is a twelve character constant. The above entry will cause only the first six digits of the date (i. e., JAN 27 rather than JAN 27, 1960) to be moved to 0243.

Care must be exercised by the programmer when using character adjusted operands. Insertions or deletions could affect the adjusted operand in such a way that * + 14 should be changed to * + 21 or * + 7.

d Modifier

The d modifier must always contain the actual machine modifier the instruction requires.

Comments

This field is reserved for comments which may be helpful in checking the program. Note that on page 24, the comments associated with each instruction provide the programmer with a complete description of the whole program. The information in this field will have no effect on the object program.

C. PROCESSOR CONTROL OPERATIONS

For the 1401 Symbolic Programming System, four types of commands are provided which control the assembly process, but are never executed in the object program. They are Control, Origin, Execute, and End. In addition to a description of these four operation codes, this section also includes a discussion of a Comments card.

CTL — Control

A control card may be entered as the first card of the source program; it has two functions. One, it tells the processor the core storage size of the 1401 on which the source program is being assembled, and two, it provides the user with an option when utilizing the Listing Routine prior to assembly (see page 9). That is, by the use of this control card the programmer may specify whether the Listing Routine should simply perform the functions itemized on page 9 (list and note discrepancies) or if, in addition, the routine is to compute the count of each instruction (thus ignoring the count supplied by the programmer) and also reproduce the source deck. This option cannot be used to compute the count of area definition entries; this must still be supplied by the programmer. An exception to this is a DSA entry, whose count is always assumed to be 3.

The mnemonic CTL must appear in the operation field. Core storage is specified in the first position of the (A) operand field (col. 17) by the digits 1, 2, or 3.

1 - 1400 core storage positions
2 - 2000 core storage positions
3 - 4000 core storage positions

If column 17 is blank, or if the CTL card is omitted, a 1400 character machine is assumed.

The Listing Routine option is indicated in the second position of the (A) operand (col. 18) by the digits 1 or 2.

1 - List and check discrepancies as described on page 9.

2 - Compute instruction length and reporduce source deck, in addition to listing and noting discrepancies.

If column 18 is blank, or if the CTL card is omitted, the routine will assume the function of 1 above.

ORG — Origin

During program assembly, the 1401 processor assigns core storage addresses to instructions, constants and work areas, as they are encountered. If not otherwise specified, addresses will be automatically assigned beginning with address 333. At times, however, it may be desirable or necessary to assign addresses beginning at some other location.

The assignment of addresses is controlled by a "counter" which is incremented, with each card, by the number of positions the assembled output will contain. For example, when address assignment begins, the "counter", unless otherwise indicated, is initialized

to 333. The first entry of the program being assembled will be assigned this address. If the first entry is an instruction seven characters in length, a "7" will be added to 333 to produce the address of the next instruction. The number of characters in each subsequent entry is in turn added to the quantity in the "counter" to produce the address of the following entry.

To begin the assignment of addresses at a location other than 333, an ORG card may be used. An ORG card may also be included at any desired point of 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 specified by the programmer.

In the sample program, an ORG card was entered as the first card of the source program deck to begin address assignment at location 0900. The first instruction, assigned address 0900, contains one character; hence, a "1" is added to 0900 to produce the address of the second instruction. The second instruction is eight characters in length causing the "counter" to be incremented to 0909, the address assigned to the third instruction. Each instruction is thus assigned an address. As each constant being defined or work area being reserved is encountered, they are also assigned addresses determined by the "counter". An area definition entry which contains an actual address in the (A) operand, rather than an asterisk, will be assigned the address specified. The "counter" will not be affected. Note that in assigning locations to reserved areas or constants, the rightmost position of the field is addressed.

The "counter" may be advanced by any quantity at any time in the program through a special use of the DS card. For example, if the "counter" contains the quantity 0937, the entry

COUNT	LABEL	OPERATION	(A) OPERAND				(B) OPERAND			
			ADDRESS	±	CHAR. ADJ.	RES.	ADDRESS	±	CHAR. ADJ.	RES.
6 7	8 13	14 16	17 23			27	28 34			38
2 0		D S	*							

will cause the counter to be incremented by 20.

An origin card must contain "ORG" in the operation field of the entry and the location at which address assignment is to begin, in the (A) operand.

EX — Execute

It may sometimes be desirable, during the loading of the object program, to discontinue the loading process temporarily and execute the portion of the program just loaded. This can be accomplished through the use of an execute card.

The Execute (EX) operation will cause the processor to assemble a branch instruction. This instruction, though not part of the object program, will be used by the loading routine at the appropriate time to cause the normal loading process to halt and the branch instruction to be executed. The branch will be to the address specified in the (A) operand.

Any valid operand may be used for this address; however, blank and asterisk operands are ineffective. The mnemonic EX must appear in the operation field.

To continue the loading process after the desired portion of the program has been executed, the programmer must provide, as the last instruction of that portion executed, a branch to location 0195 (B 195). This is required by the loading routine.

END

The END card must always be the last card of the source program. This card is a command to the processor to indicate the end-of-file condition of the source deck. An END card can also be used to begin the execution of the object program immediately after loading. This is accomplished by including the address at which the object program is to begin execution, in the (A) operand. Any valid address (i. e., actual, symbolic, etc.) is permissible. The entry "END" must appear in the operation field. If the (A) operand is blank, the 1401 will halt when the last instruction has been loaded.

Comments Card

Comment cards may be included in the source program. These cards will not be assembled nor will they affect the assembling procedure. These cards, when encountered by the processor, will be reproduced unaltered, and will be bypassed when the object program is being loaded. A comments card provides the programmer with the ability to insert lines of descriptive information in the program listing. A comments card is indicated by an asterisk in the first position of the label field (col. 8). The comments may begin at any position (9-80).

FUNCTIONAL LIST OF 1401 SYMBOLIC PROGRAMMING MNEMONIC OPERATION CODES

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
Data Control	A	Add	A
	S	Subtract	S
	*M	Multiply	@
	*D	Divide	%
	ZA	Zero and Add	Ô (Prints as &)
	ZS	Zero and Subtract	Õ (Prints as -)
	MV	Move	M
	MVS	Move and Zero Suppress	Z
	MVD	Move Digit	D
	MVZ	Move Zone	Y
	E	Edit	E
	L	Load	L
	WMS	Word Mark Set	,
	WMC	Word Mark Clear	¤
	CLS	Clear Storage	/
Logic Control	B	Branch	B
	BZN	Branch on Zone Test	V
	C	Compare	C
	NOP	No Operation	N
	H	Halt	.
System Control	UR	Unit Record Read	1
	UW	Unit Record Write	2
	UWR	Unit Record Write Read	3
	UP	Unit Record Punch	4
	URP	Unit Record Read Punch	5
	UWP	Unit Record Write Punch	6
	UC	Unit Record Combination	7
	*URR	Unit Record Read Release	8
	*UPR	Unit Record Punch Release	9
	SEL	Select Stacker #	K
	FC	Forms Control	F
	*TC	Tape Control	U

PROCESSOR CONTROL OPERATIONS			
	Mnemonic Operation Code	Description	
	CTL	Control	
	ORG	Origin	
	END	End	
	EX	Execute	

* Pertains to an optional feature.

SAMPLE PROGRAM

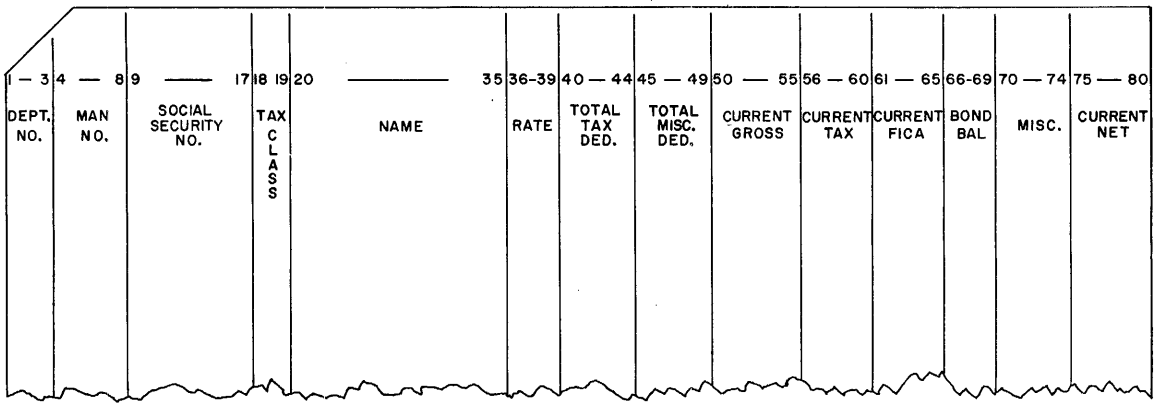
Description

In this program, two input cards are read by the 1401 to complete an employees check and earnings statement. The first card read is a current earnings card. Data from this card is computed and the results printed on the first line of the check statement. The second card is a year-to-date earnings card. The information from this card is updated and then printed as the second line of the check and statement. An updated year-to-date card is also punched out.

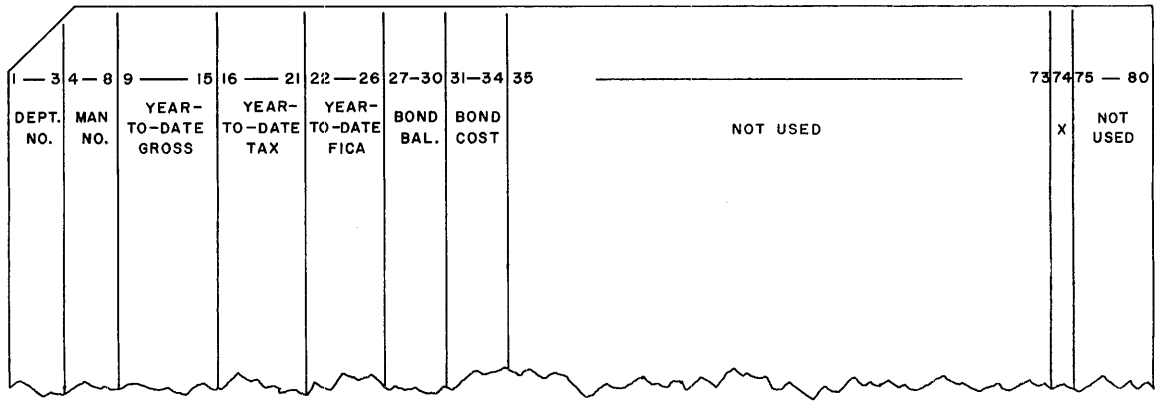
The record layouts for the input and output areas follow:

Input

1. Current Earnings Card



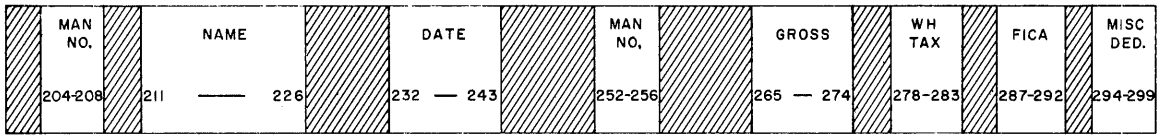
2. Year-to-Date Card



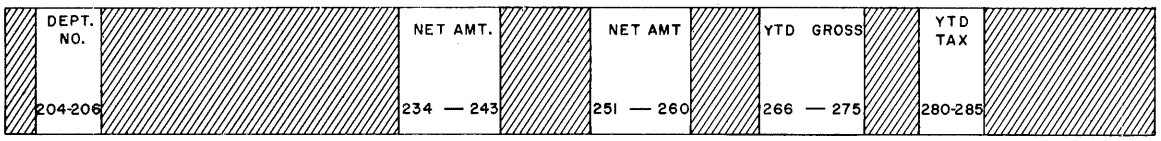
The updated year-to-date card will be punched in the same format.

Output

1. Line 1 of Check and Earnings Statement



2. Line 2 of Check and Earnings Statement



Listing of Sample Program

PAGE & LINE NO.	COUNT FIELD	LABEL FIELD	OPERATION FIELD	(A) OPERAND FIELD	(B) OPERAND FIELD	d	ADDRESSES OF ASSEMBLED INPUT	ASSEMBLED INSTRUCTIONS	COMMENTS
1 010			PAYROLL	LISTING ROUTINE	PROGRAMMED FOR THE 1401				
1 020									
1 030			ORG	0900					
1 040	1	START	UR				0900	1	
1 050	8		BZN	UPDATE	0074	K	0901	V +91 074 K	CHECK CARD TYPE
1 060	7		WMS	0020	0040		0909	, 020 040	NAME & DEDUCTION
1 070	7		WMS	0045	0061		0916	, 045 061	MISC DED & FICA
1 080	7		WMS	0066	0070		0923	, 066 070	BOND & MISC INFO
1 090	7		WMS	0075	0101		0930	, 075 101	NET AMT & DEPT#
1 100	7		WMS	0004	0050		0937	, 004 050	MAN# & GROSS AMT
1 110	4		WMS	0056			0944	, 056	W/TAX
1 120	7		MVS	0008	0208		0948	Z 008 208	MOVE MAN# TO CK
1 130	7		MV	0035	0226		0955	M 035 226	MOVE NAME TO CK
1 140	7		MV	DATE	0243		0962	M 499 243	MOVE DATE TO CK
1 150	7		MVS	0008	0256		0969	Z 008 256	MV MAN# TO STMNT
1 160	7		L	EDTWD1	0274		0976	L S38 274	EDIT
1 170	7		E	0055	0274		0983	E 055 274	GROSS
1 180	7		L	EDTWD2	0283		0990	L S44 283	EDIT
1 190	7		F	0060	0283		0997	E 060 283	W/TAX
1 200	7		L	EDTWD2	0292		1004	L S44 292	EDIT
1 210	7		F	0065	0292		1011	E 065 292	FICA
1 220	7		ZA	0049	TOTALS		1018	& 049 S56	TOTAL ALL
1 230	7		A	0044	TOTALS		1025	A 044 S56	DEDUCTIONS
1 240	7		L	EDTWD2	0299		1032	L S44 299	EDIT TOTAL
1 250	7		E	TOTALS	0299		1039	E S56 299	DEDUCTIONS
2 010	7		MV	0008	0108		1046	M 008 108	MOVE DEPT# &
2 020	1		MV				1053	M	MAN# TO PUNCH
2 030	7		MV	0069	0130		1054	M 069 130	MOVE ADJACENT
2 040	1		MV				1061	M	FIELDS
2 050	1		MV				1062	M	TO PUNCH
2 060	7		MV	0055	0115		1063	M 055 115	MV GROSS TO PUNC
2 070	7		MV	0080	NETAMT		1070	M 080 S51	SAVE NET AMOUNT
2 080	2		FC			B	1077	F B	SKIP 2 AFTER PRT
2 090	1		UW				1079	2	PRINT 1ST LINE
2 100	4		CLS	0080			1080	/ 080	CLEAR READ AREA
2 110	7		CLS	START	0299		1084	/ 900 299	CLR PRT & BRANCH
2 120	7	UPDATE	WMS	0009	0016		1091	, 009 016	DEFINE
2 130	7		WMS	0022	0109		1098	, 022 109	FIELDS
2 140	7		WMS	0116	0122		1105	, 116 122	
2 150	7		WMS	0104	0127		1112	, 104 127	
2 160	7		C	0008	0108		1119	C 008 108	COMPARE MAN#
2 170	5		B	ERROR		/	1126	B S28 /	
2 180	7		A	0026	0126		1131	A 026 126	UPDATE TOTALS ON
2 190	1		A				1138	A	NEW YEAR/DATE
2 200	1		A				1139	A	CARD
2 210	7		MV	0103	0206		1140	M 103 206	MOVE DEPT#
2 220	7		L	EDTWD1	0243		1147	L S38 243	EDIT NET PAY
2 230	7		E	NETAMT	0243		1154	E S51 243	FOR CHECK
2 240	7		L	EDTWD1	0260		1161	L S38 260	EDIT NET PAY
2 250	7		E	NETAMT	0260		1168	E S51 260	FOR STATEMENT
3 010	7		L	EDTWD1	0275		1175	L S38 275	EDIT Y/D GROSS
3 020	7		E	0115	0275		1182	E 115 275	FOR STATEMENT
3 030	7		L	EDTWD2	0285		1189	L S44 285	EDIT Y/D W/TAX
3 040	7		E	0121	0285		1196	E 121 285	FOR STATEMENT
3 050	7		MV	0034	0134		1203	M 034 134	MOVE BOND INFO
3 060	2		FC			A	1210	F A	SKIP 1 AFTER PRT
3 070	1		UWP				1212	6	PRINT & PUNCH
3 080	4		CLS	0080			1213	/ 080	CLEAR READ AREA
3 090	4		CLS	0299			1217	/ 299	CLEAR PRINT AREA
3 100	7		CLS	START	0180		1221	/ 900 180	CLR PCH & BRANCH
3 110	1	ERROR	H				1228	.	STOP ON ERROR
3 120	10	EDTWD1	DCW	*			1238	$, 0.	
3 130	6	EDTWD2	DCW	*			1244	0.	
3 140	7	NETAMT	DCW	*			1251	0000000	
3 150	5	TOTALS	DCW	*			1256	00000	
3 160	12	DATE	DCW	0499			0499	JAN 27, 1960	
3 170			END	START				B 900	

CONSTANTS AND WORKING AREAS

IBM
International Business Machines Corporation
Data Processing Division, 112 East Post Road, White Plains, N.Y.
Printed in U.S.A. J28-0200-1 1/60