Principles of Programming

Section 3: Coding Fundamentals

IBM Personal Study Program

Section 3: Coding Fundamentals

In order to process data with a computer, it is necessary to provide the machine with a *program* of *instructions*. A computer instruction is a command to the machine, expressed in a coded combination of numbers and letters, to carry out some simple operation. Once the basic data processing task has been completely defined, the job of *coding* is to put together a suitable set of these elementary instructions to do the defined task. When the set of instructions, which is called a *program* or *routine*, is loaded into the internal storage of the computer, the instructions can be executed by the machine and the desired data processing thereby carried out.

In this section we shall learn what a few of the simpler instructions are and how they operate. We must begin, however, by investigating the characteristics of the internal storage of the computer.

3.1 Computer Storage and Its Addressing

The *storage* of a computer (also sometimes called the *memory*) is the part of the machine where instructions must be placed before they can be executed, and also where the data being processed by the instructions must be placed. By this definition we refer to *internal* storage; such things as magnetic tapes and magnetic disks are *external* storage. Instructions can be executed only from internal storage, and the data currently being processed must be put into internal storage before any processing can be done on it. When data in external storage is to be processed, it must first be read into internal storage by the execution of instructions.

We saw that in working with cards it was necessary to deal with groups of columns, called *fields*. We saw that the interpretation of a group of columns constituting a field was completely under the control of the user of the equipment. Similarly, in working with the internal storage of a computer we must work with groups of characters, which are called *words*. A computer word may be defined as any collection of characters that is treated as a unit. For instance, when the sales cards of Section 1.3 and 1.4 are read into computer storage, such things as the product number and the unit price are words. An instruction is also considered to be a word.

In many computers, the number of characters in a word is fixed by the design and construction of the machine. A typical size is ten char-

acters. Such machines are said to have *fixed word length*. Other computers, including the IBM 1401 and several others, permit words to be of any length from one character up to (in principle) the size of the storage. Such machines are said to have *variable word length*.

In any computer, whether of fixed or variable word length, it is necessary to be able to identify every location in storage where a word can be stored. For this purpose an *address* is assigned to every word location in a fixed word length machine, and to every character location in the variable word length case. The addresses start at zero and run up to one less than the number of storage locations.

Note carefully that an address identifies a word *location*, not a word. For example, the address 593 in the 1401 refers to a *place* where a character may be stored; it does not by itself tell us what is stored there. A location that contains the character A at one time may be used a moment later to store the digit 7. We must always make a most careful distinction between the address of a location, and the word or character currently stored at the location identified by that address.

The internal storage of the 1401 is made up of magnetic cores, as pointed out previously. The smallest model of the 1401 is able to store 1,400 characters of instructions or data; larger versions are available which store 2,000, 4,000, 8,000, 12,000 or 16,000 characters. In this book we assume a machine that can store 4,000 characters. Each of the 4,000 positions is able to store any one of the 48 digits, letters or special symbols; it is also possible to store 16 other symbols that have various meanings within the computer. Thus, each of the 4,000 character positions is able to hold any one of 64 different characters.

Each character position is composed of eight magnetic cores, each core holding one bit. Six cores are required for six bits of this character itself, as discussed in Section 2.5. One core holds the parity bit, and the eighth is used for the word mark bit. This last has the function of defining the length of words within storage. Any character position in which the word mark bit is *one* is thereby identified as being the *high-order* (leftmost) position of a word. As we shall see, word mark bits can be *set* (made one) or *cleared* (made zero) by the execution of appropriate instructions.

When a data word is referenced by the computer, it is always by the address of its low-order (rightmost) character position. The machine is built so that addresses increase as the character positions are taken from left to right, which means that the low-order character of a word has the largest address. To summarize: When a character position is addressed for data, the computer takes the character in that position and all higher-order (but lower address) characters as comprising a word, until it reaches a character with the word mark bit on. If the character position that is addressed has its word mark bit on, the word will consist of just that one character.

We shall see in Section 3.2, in connection with instructions, that all

storage addresses are written as three 1401 characters. The first thousand addresses are written simply as numbers between zero and 999. Addresses of 1000 and over are handled in a special manner to fit into three characters, by the following scheme. The numerical parts of the three characters are always the hundreds, tens, and units digits of the address. The zone bits of the high-order (hundreds) character are regarded as the thousands digit, according to the following pattern:

If the zone bits are B	A	Then the thousands digit is
0	0	0
0	1	1
1	0	2
1	1	3

Thus, the binary coded form of the address 1234 would be:

01 0010 00 0011 00 0100

1 2 03 04

(Word mark and parity bits not shown.) The address 3789 would be coded:

11 0111 00 1000 00 1001

3 7 08 09

Naturally, we do not want to have to show the binary coded form of such addresses; instead we write them as though the high-order character were the character represented by the combination of zone and numerical bits. Looking at the table on page 4, we see that the address 1234 would be written S34, and 3789 would be written G89. The complete pattern of three-character addresses is shown in Figure 1, for addresses up to 3999. Larger addresses are handled by using the zone bits of the units digit in a similar system.

Review Questions

1. *What is the difference between internal and external storage?*
2. *Explain the following statement: instructions can be* stored *in external storage, but they cannot be* executed *while in external storage.*
3. *What is the three-character form of the address 1643?— of 2700?*
4. *The eight character positions 678-685 contain the characters 9 3 8 6 5 2 7 4, where underlining a character means that that character position has a word mark. If we address character position 684 for data, what word will result?*

CODED ADDRESSES IN STORAGE

ACTUAL ADDRESSES		3-CHARACTER ADDRESSES
000 to 999	No zone bits	000 to 999
1000 to 1099	A-bit, using 0-zone	=00 to =99
1100 to 1199		/00 to /99
1200 to 1299		S00 to S99
1300 to 1399		T00 to T99
1400 to 1499		U00 to U99
1500 to 1599		V00 to V99
1600 to 1699		W00 to W99
1700 to 1799		X00 to X99
1800 to 1899		Y00 to Y99
1900 to 1999		Z00 to Z99
2000 to 2099	B-bit, using 11-zone	I 00 to I 99
2100 to 2199		J00 to J99
2200 to 2299		K00 to K99
2300 to 2399		L00 to L99
2400 to 2499		M00 to M99
2500 to 2599		N00 to N99
2600 to 2699		*O00 to O99
2700 to 2799		P00 to P99
2800 to 2899		Q00 to Q99
2900 to 2999		R00 to R99
3000 to 3099	A-B-bit, using 12-zone	?00 to ?99
3100 to 3199		A00 to A99
3200 to 3299		B00 to B99
3300 to 3399		C00 to C99
3400 to 3499		D00 to D99
3500 to 3599		E00 to E99
3600 to 3699		F00 to F99
3700 to 3799		G00 to G99
3800 to 3899		H00 to H99
3900 to 3999		I00 to I99

* Letter O followed by Zero Zero

Figure 1. Core storage address codes.

3.2 Instructions

A computer instruction is an order to carry out some elementary operation. Some instructions call for information to be read into internal storage from an *input* device such as a card reader, or to be written out to an *output* device such as a line printer. Other instructions perform *arithmetic*. A third class *moves* and *rearranges* data within the computer. A final group is used to make various kinds of *decisions* based on data or results.

All instructions have an *operation code*, which tells the machine what operation to perform. In most computers, each instruction also has a fixed number of *address parts*, which in most cases specify where in storage to obtain data or place results. There may also be other parts having special purposes in a particular machine.

In the 1401, every instruction has a one-character operation code—which in a few cases is the entire instruction. Most instructions also have one or two address parts, which are three characters, and some have a one-character d-modifier which has a variety of functions depending on what the instruction does. An instruction in the 1401 may thus be from one to eight characters in length, making it a *variable instruction length* computer.

The general form of a 1401 instruction is:

Operation Code	A-address	B-address	d-character
X	XXX	XXX	X

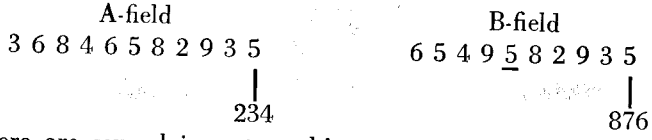
Any parts that are not used on a particular instruction are simply omitted. Some instructions, for instance, consist of only an operation code and one address, or an operation code and a d-character. As with data words, an instruction word is required to have a word mark in its high-order position, which is always the operation code.

To see how these parts fit together, let us consider a typical instruction used to move a word from one part of storage to another. Such an instruction might be: M 234 876. In this instruction, M is the operation code. It means Move Characters to A or B Word Mark. The 234 is called the *A-address* and 876 the *B-address*. (These addresses could in general be any two addresses in storage.) The instruction means to move the word starting at the A address, to the word starting at the B address, with the length of the word moved being defined by the first word mark to appear in *either* place.

Suppose for example that the following characters are in storage:

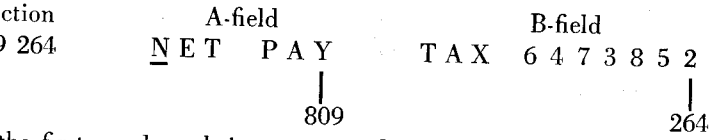
A-field		B-field	
3 6 8 4 6 5 8 2 9 3 5		6 5 4 9 6 2 7 9 5 7	
	234		876

In this example, the instruction means to move the word starting at 234 to the word starting at 876. The length of the word moved will be established by encountering a word mark in either field in storage; the only word mark in this case is in character position 871. The word moved will thus be 582935. After the instruction has been executed, the storage fields will be:

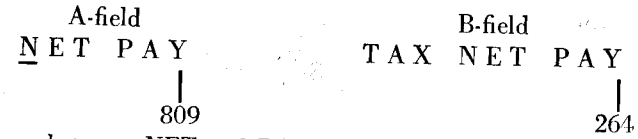


There are several important things to notice about this example. First, the storage positions *from which* the word was moved were not affected. To be technical about it, the word is not really "moved" but "copied and moved." Second, the previous contents of the storage positions *to which* the word is moved are destroyed. As a completely general principle, any time anything is placed in storage locations the previous contents of the locations are erased. It is the programmer's responsibility to be sure that the previous contents are no longer needed. Third, the Move instruction does not change word marks. In fact, word marks are not affected by most instructions; when word marks are to be set or cleared, special instructions are used. Thus, when word marks are set to define fields (words) in storage, the definitions stay in effect until deliberately altered.

For another example, suppose that the instruction and storage contents are as follows:



Here the first word mark is encountered in the A-field. The result of this instruction is:



The space between NET and PAY is no accident. In showing the example this way it is assumed that a blank space between NET and PAY is desired. In order to obtain it, a character position must be set aside for the purpose. The character "blank" is thus a character with the same status as any other.

The essential information about the instruction Move Characters to A or B Word Mark is summarized below. *In order to make this summary a source of all the reference information about the instruction, it is necessary to list some things that will not be explained until later.*

Move Characters to A or B Word Mark

FORMAT	Mnemonic	Op Code	A-address	B-address
	MCW	M	xxx	xxx

FUNCTION The word in the A-field is moved to the B-field. The data in the A-field is not changed; the previous data in the B-field is lost.

WORD MARKS The first word mark encountered in either field stops the operation. If the first word mark is in the A-field, the character at that position is moved; if the first word mark is in the B-field, that position receives a character from the A-field. Word marks are not disturbed in either field. If the fields are the same length, only one of them need have a word mark.

TIMING $T = 0.0115\ (L_I + 1 + 2\ L_w)$ ms.

We see that to move words within storage (and in fact to do almost any data manipulation) it is necessary to have word marks set. This naturally means that some way must be provided for setting and clearing word marks within a program of instructions. This facility is provided by two instructions called Set Word Mark and Clear Word Mark. These instructions may have either one or two addresses, allowing us to deal with either one or two word marks at a time. The operation code "," is recognized by the computer as meaning Set Word Mark, so that the instruction

, 200258

would mean to set the word mark bits of character positions 200 and 258. (Setting the word mark bit means making it a one and clearing means making it a zero. It is convenient to use phrases like "the first character with a word mark" instead of the more precise "the first character in which the word mark bit is a one.")

Set Word Mark

FORMAT	Mnemonic	Op Code	A-address	B-address
	SW	,	xxx	xxx
	or SW	,	xxx	

FUNCTION The word mark is set in both locations specified, or in the one location if only one address is written. The character(s) at the location(s) are unchanged.

TIMING $T = 0.0115\ (L_I + 3)$ ms.

The operation code for the Clear Word Mark instruction is ¤ which is called a *lozenge*. Like Set Word Mark, this instruction may have one or two addresses. Its effect is to set to zero the word mark bit in the character position or positions addressed.

Clear Word Mark

FORMAT	Mnemonic	Op Code	A-address	B-address
	CW	¤	xxx	xxx
	or CW	¤	xxx	
FUNCTION	The word mark is cleared in both locations specified, or in the one location if only one address is written. The character(s) at the location(s) are unchanged.			
TIMING	$T = 0.0115\ (L_I + 3)$ ms.			

For an example of the use of these instructions, suppose storage positions 600-608 contain the following characters:

A H 8 4 K 7 L 5 6
608

Executing the pair of instructions

¤ 608 604 ,602

would leave storage looking like

A H 8 4 K 7 L 5 6

Notice that setting and clearing word marks does not have any effect on the character stored in a position.

The reading of a card is called for by executing the Read a Card instruction, the operation code of which is 1. This instruction, which need not have any address, causes a card to be read and the information placed in storage in positions 1-80, which is called the *read storage area*. The character in column 1 is placed in position 1, the character in column 2 is placed in position 2, etc., which makes it quite easy to work with the card information when it has been read into storage. There is no way to read the card information into any other positions than 1-80; as we shall see later, when an address is used on a Read a Card instruction, it does not refer to data. Reading a card destroys any previous contents of positions 1-80, except that word marks are not affected.

Read a Card

FORMAT	Mnemonic	Op Code
	R	1
FUNCTION	A card feeds and the 80 columns of information are read into storage locations 001 to 080.	

WORD MARKS	Not disturbed.
TIMING	$T = 0.0115\ (L_I + 1)$ ms + I/O

The punching of a card is called for by the Punch a Card instruction, which has the operation code 4. This instruction, which also need have no address, causes whatever is in the punch storage area, positions 101-180, to be punched into a card. Punching a card does not affect the contents of the punch storage area.

Punch a Card

FORMAT	Mnemonic	Op Code
	P	4
FUNCTION	The data in storage locations 101 through 180 is punched into an IBM card.	
WORD MARKS	Not disturbed.	
TIMING	$T = .0115\ (L_I + 1)$ ms + I/O	

The printing of a line of information on the printer is called for by the Write a Line instruction, which has the operation code 2. The line printed consists of the 100 characters in the print area, positions 201 300.

Write a Line

FORMAT	Mnemonic	Op Code
	W	2
FUNCTION	The data in storage locations 201-300 is transferred to the printer. The printer takes one automatic space after printing a line.	
WORD MARKS	Not affected.	
TIMING	$T = .0115\ (L_I + 1)$ ms + I/O	

The IBM 1403 Printer can optionally be equipped with 132 printing positions, in which case the print area consists of positions 201-332.

It is often necessary to clear an area of storage. For instance, suppose that certain data and results are to be moved into the print area and printed. The words moved into the area will ordinarily not occupy every position, and we naturally want to erase the contents of the unused positions before printing, to eliminate the unwanted characters. Furthermore, it is often necessary to clear word marks in an entire area of storage; once again the print storage area is a good example. The Clear Storage instruction makes it possible to clear as many as 100 positions with one instruction, putting the character *blank* in all, and

clearing all word marks. The operation code is / which is technically called a *virgule* but is more commonly referred to as a *slash* or *slant*.

Since one of the functions of this instruction is to clear word marks, it obviously cannot depend on the detection of a word mark to stop its action. Instead, it is built to clear all positions from the one addressed down to and including the nearest hundreds position. That is, if the instruction / 799 is executed, positions 799, 798, 797, . . . , 701, 700 are set to blank and word marks cleared. If the instruction / 801 is executed, positions 801 and 800 would be cleared. The instruction / 400 would clear position 400 only.

Clear Storage

FORMAT	Mnemonic	Op Code	A-address
	CS	/	xxx
FUNCTION	Clearing starts at the A-address and continues leftward through the nearest hundreds position. The area is set to blanks, and word marks are cleared.		
WORD MARKS	Word marks are not required to stop the operation.		
TIMING	$T = .0115\ (L_I + 1 + L_x)$ ms.		

To illustrate the use of the instructions described so far, suppose that we are required to read a card and print some of the information on it in a readable format. The card format for the sales card of Section 1.3 was:

Columns	1-4	Product number
	5-8	Units sold
	9-11	Salesman
	12-13	District

Suppose that we are required to print this same information in the following positions:

Print positions	1-4	Product number
	10-13	Units sold
	19-21	Salesman
	27-28	District

This spaces the numbers out, so that they can more easily be read.

As we start this operation, we do not know what is in the read and print areas in storage—and even if we did know it probably would be unwanted information and the word marks would likely be in the wrong places. In the course of carrying out a complete program there are ordinarily several different types of cards to be read and lines to be printed, so that word marks must be set properly for each type before trying to use the information from cards or trying to move information to the print area.

For these reasons we must begin the program by clearing the read and print areas, which can be done with three Clear Storage instructions (we assume that the printer has the additional print positions):

```
/ 080
/ 299
/ 332
```

As soon as a card has been read, it will be necessary to move the four words from the read area to the print area, which will require word marks to define the length of the fields. As far as we are concerned in this particular example, it would not matter whether the word marks were set before or after reading the card. However, the normal situation would be to read and print many cards, all having the same format, in which case we would repeat part of the program each time a card is read and the line printed. When this is done, it is pointless to set the word marks for every card; reading a card does not erase them. Therefore, it is desirable to set the word marks before reading the card.

We recall that the Move Characters to A or B Word Mark instruction is stopped by a word mark in either the A- or the B-field, so that it is not necessary to set word marks in both the read and print areas. In this example it really doesn't matter much which area has them; we shall set the word marks in the read area. Remembering that the word mark of a data word must be in the *high-order* position, we need to set word marks in positions 1, 5, 9 and 12. This can be done with the instructions:

```
, 001 005
, 009 012
```

Now the card can be read, which requires only the operation code 1. With the data from the card in the read area, we can move the words to the print area, which requires the following four instructions:

```
M 004 204
M 008 213
M 011 221
M 013 228
```

Recall that a Move instruction addresses the *low-order* position (but largest address) and moves characters until it encounters a word mark in the *high-order* position of either field, in this case the A-field.

The data in the print area can now be printed, which requires only the operation code 2. The program is thus as shown in Figure 2.

IBM 1401 PROGRAM CHART — PRINTED IN U.S.A.

Program:

Programmer: Date:

Step No.	Inst. Address	Instruction O P	A/I d			B d			d	Remarks	Effective No. of Characters Inst.	Data	Total
		/	0	8	0					CLEAR STORAGE POSITIONS 000-080			
		/	2	9	9					" " " 200-299			
		/	3	3	2					" " " 300-332			
		,	0	0	1	0	0	5		SET WORD MARKS POSITIONS 001, 5,			
		,	0	0	9	0	1	2		9, 12			
		1								READ A CARD INTO 001-080			
		M	0	0	4	2	0	4		MOVE POSITIONS 1-4 TO 201-204			
		M	0	0	8	2	1	3		" " 5-8 TO 210-213			
		M	0	1	1	2	2	1		" " 9-11 TO 219-221			
		M	0	1	3	2	2	8		" " 12-13 TO 227-228			
		2								WRITE A LINE			

Figure 2. Program segment to clear storage, set word marks, read a card, and print some of the information from the card.

Review Questions

1. *What does the operation code of an instruction do?*
2. *In the example on page 5, suppose there had been a word mark in position 870. Would the word moved have been the same or different?*
3. *Suppose there had been a word mark in position 234. What would have been moved?*
4. *Can word marks be set or cleared with a Move instruction?*
5. *Suppose that storage contains the following characters:*

A-field								B-field								
2	5	8	D	P	7	F	G	5	H	K	L	M	8	9	5	3	V
								604								709

What will be the contents of the storage positions if we execute the three instructions

```
, 709
M 604 709
¤ 709
```

Starting with the original contents again, what would result from:

```
M 602 704
```

6. *Can you suggest why the computer was designed so that the Clear Storage instruction clears to blanks rather than zeros?*

3.3 Storage of Instructions

We have so far spoken of instructions in terms of what they cause the machine to do, and have not said anything about how the machine deals with the instructions themselves.

The first and most important thing to realize is that the program of instructions must be prepared *before* the processing is done, and that the program must be *in storage* before it is executed. We write the program, punch it on cards in a suitable format, load the instructions into storage, and then the instructions control the machine without any further action on our part.

This means, among other things, that when we write the program we must anticipate everything that the machine will have to do. We must know, for instance, the maximum sizes of the fields that the computer will process, but we cannot know the actual numbers that will be dealt with. The instructions must be set up to handle *any* data of the general type that it is designed to handle. If something comes up that we did not anticipate, the program will still do what the instructions say to do, even though the results may be meaningless. The fundamental consideration is that by the time the instructions are executed by the machine, *we* are no longer in the picture.

Another consequence of the storage of instructions is that they must be capable of being stored in the same storage that is used for data, and they must be set up so that the machine can determine such things as where one instruction ends and another begins. And, since it is frequently necessary to repeat the execution of groups of instructions or to skip around in the program, we must have some way to identify an instruction by where it is located in storage.

This brings us to a discussion of how instructions are stored within the computer, which is one of the most important topics in the entire study of programming. The crucial concept is that instructions are brought to the control unit for execution from internal storage, where they are stored in the same way data is stored. We may therefore talk about where instructions are stored in much the same way as we talk about where data is stored.

In the IBM 1401, instructions are executed from consecutively higher-numbered storage locations, unless special action is taken to break the consecutive sequence. The operation code of every instruction *must* have a word mark. Every character of each instruction is stored in a character position; an instruction is identified by the address of the operation code. Note that the operation code is the *high-order* character of the instruction, so that the addressing of instruction words is opposite to that of data words. Furthermore, instruction words are picked up from storage from left to right, whereas data words are picked up from right to left.

For an application of these ideas, consider the program that was developed in the previous section. This could be stored in any location that does not conflict with the storage of data; it should be obvious that since the program is stored just as data is, the program storage must not overlap the data storage. A storage location can store either one character of an instruction or one character of data, but not both at the same time, obviously. In this example, the only locations used for data are the read and print areas; the program could in principle be placed anywhere else. We will avoid the punch area, however, on general principles: in most cases it will be needed for storing information to be punched on cards, and we prefer not to get in the habit of putting instructions where they could get in the way in some problems.

Let us make the arbitrary choice of storage location 700 for the first character of the first instruction of this illustrative program. Then the complete program in storage, viewed the same way we view data, would be:

```
/080/299/332,001005,0090121M004204M008213M011221M0132282
700 704 708 712     719 726 727     734     741     748     755
```

The underlining here represents word marks, as with data. The characters with word marks are of course the operation codes. Simply by counting character positions from the first location of the program, we can determine the address of each character of the program. The most important location for each instruction is the one that contains the operation code, since it is by the address of the operation code that we refer to an instruction.

It would obviously be inconvenient to show instructions strung out along a line this way, which was done to emphasize the similarity of storage of data and instructions. The normal way to write instructions is on a 1401 Program Chart, as shown in Figure 2. On this form, the step number is used at the discretion of the programmer, for his convenience. It may be used to identify the instruction when it is punched on a card; it does not enter the computer or have anything to do with the computer's operation. The instruction address is the address of the storage location where the operation code is stored. OP stands for the operation code. A/I is the address of the A data field, or, as we shall see a little later, the address of the next instruction. B is the address of the B data field—if there is one, of course. d is the d-character, which we shall also consider later. The Remarks space may be used to explain what the instruction does, for ease of understanding by other programmers, or as a reminder to the original programmer as to what the program does. (It is surprising how unfamiliar one's own work can seem after six months.) The Effective Number of Characters space is used to determine how much computer time will be used by the instruction;

we shall not be greatly concerned with this problem.

It is important to be clear on how much of this gets into the computer: only the instruction itself. The instruction address is not part of the instruction; it merely tells *us* where the instruction is located in storage (or, rather, *will* be located, after the program is put into storage). The other parts—the step number, the remarks, and the effective number of characters—are strictly for the convenience of the programmer.

As we have seen, an instruction for the 1401 can vary in length, whereas in most computers the length is fixed. It is necessary to fill in only as many boxes on the form as are used on each instruction.

Review Questions

1. *Does the fact that instructions are stored in a manner very similar to the way data is stored, suggest that it might be possible to do arithmetic on instructions?*
2. *For both data and instructions, the word mark is placed in the high-order character. Are data and instructions both addressed in the same way also?*
3. *Can you tell, without knowing anything about the program organization, whether a given character belongs with data or instructions?*
4. *What is the use of the instruction address column on the coding sheet?*

3.4 Arithmetic and Control Registers

The computer carries out its work of interpreting instructions and processing data by use of several *registers*, a register being an electronic device that can hold one or more characters. Some registers are involved in the transmission of information between internal storage and other parts of the machine. Some are used to hold the parts of an instruction while it is being executed. Others are used to hold the data or results of arithmetic operations.

There are six registers in the part of the 1401 that interprets instructions and operates on information in the internal storage of the machine, as shown in Figure 3. (There are a number of other registers involved in transferring information between internal storage and input or output devices, but we shall not be concerned with them.)

The most heavily used register of these is the *B-register*, which holds one character. Every character leaving core storage enters the B-register and is then directed elsewhere, depending on what is being done at the moment. If the character is the first character of an instruction,

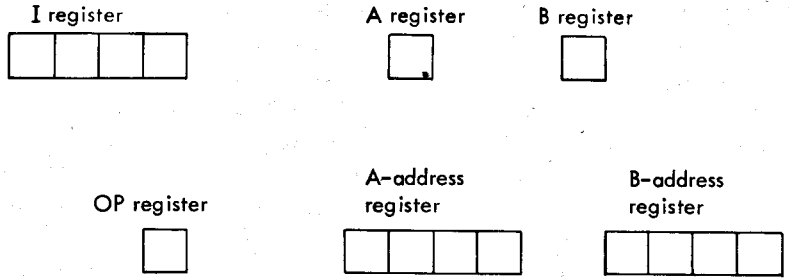


Figure 3. 1401 Processing Unit registers.

which is the operation code, it is sent to the *OP-register*, where the machine inspects the character and uses it to determine what is to be done by this instruction.

If the character from storage is part of the A/I address, it is sent to the proper position of the *A-address register*, which is a three-character register that will later determine (in most cases) the address of the next data character to be obtained from storage. If the character entering the B-register is a part of the B-address, it is sent to the proper position of the *B-address register*, also three characters, where it will later determine (in most cases) the next location to which to send a character in storage.

The A-address and B-address registers are actually three-character registers, corresponding to the three characters in a 1401 address, but for convenience they are displayed on the console of the machine in four-character form. For this reason they are shown as four characters in the diagram of Figure 3.

The d-character is not stored in a separate register.

We see that the A- and B-address registers are used primarily to keep track of the addresses of data characters. The *I-address register* performs the same function for instructions. This clearly is necessary; since instructions are stored as data is, the machine must have some way to keep track of where the next instruction character is to come from.

The operation of the registers may be explained more fully in terms of an example. Suppose that the instruction to be executed is:

M 4 1 0 7 8 9
|
350

In order to execute the instruction, the I-address register must contain 350. The 350 would normally be there as the result of the execution of the previous instruction—that is, the last character of the previous instruction was located in position 349, and we said that the register always contains the address of the *next* instruction character to be obtained from storage.

The machine operates in two *phases:* an *instruction phase* and an *execute phase*, or *I-phase* and *E-phase*. The I-phase is used to obtain and interpret the instruction, and the E-phase is used to carry out the instruction. When the I-phase begins, the machine uses the contents of the I-register to determine from where in storage to obtain the first character of the instruction, which is always the operation code. When this character is obtained from core storage, it moves through the B-register and into the OP-register. The machine "looks at" the operation code in the OP-register, with suitable electronic circuitry, and determines what the function of this instruction will be. This also tells the machine something about the function of the remaining characters of the instruction as they are obtained.

As soon as the first character of the instruction has been obtained, the contents of the I-register are increased by one, giving the address of the next character of the instruction. This is then obtained; it goes through the B-register to the A-address register. The I-register is again increased by one, the next character obtained and placed in the A-address register, etc. In our example of a Move instruction with two addresses, this process would continue until both addresses had been obtained and placed in the A- and B-address registers. At this point the I-register would contain 357, the address of the next instruction character from storage. This character would also be obtained from storage and placed in the B-register, but the machine would at this point detect a word mark, since this character would be the operation code of the next instruction, whatever it is. The word mark would signal the machine that this instruction is complete, and would thus end the I-phase.

No data has been moved yet! This much simply gets the instruction from storage and prepares for the *execution* of the instruction, which may now begin. The starting addresses of the two fields are in place in the A-address register and the B-address register, and circuits in the control section of the machine have been set up to carry out the Move function as a result of interpreting the M in the OP-register as meaning "move."

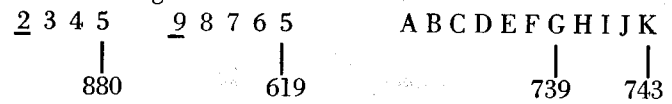
The first step of the E-phase is to obtain the first character of the A-field, the address of which is given by the contents of the A-address register. This character is brought from storage, placed in the B-register, checked to see whether it has a word mark, moved to the A-register, and placed back in storage at the location specified by the contents of the B-address register. As the character is stored, the machine is able to check whether the position at which it is being stored has a word mark. This completes the movement of one character. The contents of the A-address register and the B-address register are both decreased by one, to prepare for dealing with the next character. If a word mark was detected in either storage position, the instruction

execution is completed and we go back into the I-cycle to obtain and interpret the next instruction; if not, the next character is moved. This process of getting one character, moving it to another location, and checking both places for word marks to determine when the movement is finished, is repeated until a word mark is finally detected.

For the purposes of things we sometimes want to do next, it is important to realize the status of the three address registers when the instruction is finished. The I-address register contains the address of the first character (the operation code) of the next instruction in storage. We have seen that this character must have a word mark in order for the control circuits to be able to recognize the end of the current instruction. In most cases, the next instruction will be the next one in sequence in storage, but we shall see several important exceptions. The A-address register contains the address of the next character after the last one transferred. Since the data characters are picked up from storage from right to left, this address will be one less than the address of the last character obtained. Stated another way, it is the address of the next higher-order character after the last one moved. This will often be the low-order character of another data word—a fact which can sometimes be useful. The B-address register, similarly, contains the address of the next character position after the last one into which a character was moved. This will also often be the low-order character of another word.

It is frequently possible to take advantage of the contents of the registers after the completion of an instruction, using a technique called *chaining*. Any time the B-address register already contains the desired address, it is permissible to omit the B-address of the next instruction; if *both* address registers already contain the desired addresses, both addresses may be omitted. This saves storage space, obviously, and also saves the time that would have been spent in obtaining the longer instruction from storage. This is a unique feature of the IBM 1401 system.

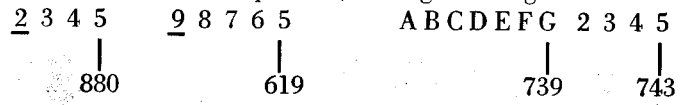
For an example of how chaining can be used, suppose storage contains the following:



Suppose that it is desired to move the field in 877-880 to 740-743, and to move the field in 615-619 to 735-739. Note that the two fields are to be placed in consecutive locations. If we first execute the instruction:

M 880 743

we will move one field as specified, leaving in storage:



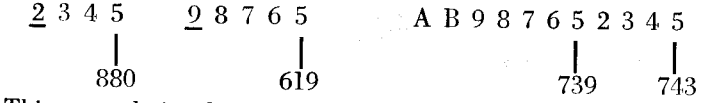
Now what are the contents of the A- and B-address registers? The A-address register contains 876, the address of the character to the left of the last one picked up from storage. This fact is of no value to us, since that is not where the next field is to come from. The contents of the B-address register *are* useful, however: 739 is just the address we would have to write on the next Move instruction. We may therefore omit it, and write:

M 619

The control section will recognize the word mark on the next instruction (whatever it is) as terminating this instruction without having picked up a new B-address. Therefore the contents of the B-address register will not be disturbed, and the previous contents will be used. The effect is the same as if the instruction had been:

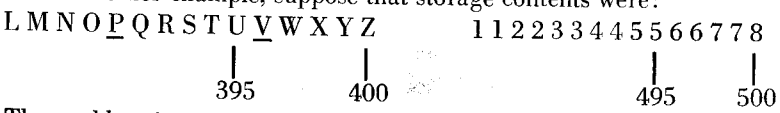
M 619 739

Either way, the storage contents will now be:



This example involves a technique called *partial chaining*, which is permissible with the Move and Load instruction only, since other instructions are treated differently by the control section.

For another example, suppose that storage contents were:



The problem is to move 390-395 to 490-495, and 396-400 to 496-500. Since the fields are consecutive in both places, we should be willing to do it all in one instruction—but this cannot be done because the first word mark will stop the movement. However, if we make the first instruction:

M 400 500

then the A- and B-address registers will both be properly set up to move the second field, since they will contain 395 and 495 respectively. The second instruction can consist of just the operation code: M.

Chaining thus saves six instruction characters and a certain amount of computer time. Properly used, it can be quite valuable. (But don't try to do things with chaining that can't be done! It is not possible to use chaining unless the following field is immediately to the left of the previous one. Furthermore, it is not possible to omit the A-address and write a B-address; the machine will always put the first address it finds into the A-address register, and has no way of "knowing" that you meant it to go into the B-address register. Therefore, if the A-address register is properly set up, but the B-address register is not, chaining is not applicable.)

Review Questions

1. *What is a register?*
2. *Which of the registers is involved in every transfer of information out of storage?*
3. *What is the difference between an I-phase and an E-phase?*
4. *What are the contents of the A-address register and the B-address register after any movement of data?*
5. *How does the control section "know" that it has reached the end of an instruction? (Bear in mind that an instruction can be from one to eight characters in length.)*
6. *What is chaining?*

3.5 Addition and Subtraction

The basic idea of addition is that the number in the A-field is added to the number in the B-field and the sum replaces the B-field. The B-field must have a word mark, because it is this word mark that stops the instruction execution. The A-field is required to have a word mark only if it is shorter than the B-field; in this case the A-field is added only until its word mark is reached, but all carries in the B-field are completed. We may illustrate the addition operation with following storage contents:

The instruction

A 608 473

would give the resulting B-field:

5 7 4 0 2 7 4

The word mark in 606 signals the end of the A-field, but all carries in the B-field are completed.

If the instruction had been

A 608 470

with the original storage contents, the result would be:

5 7 6 4 9 4 9

The word mark in 469 stops the operation, without a word mark having been detected in the A-field.

Add

FORMAT	Mnemonic	Op Code	A-address	B-address
	A	A	xxx	xxx

FUNCTION The data in the A-field is added algebraically to the data in the B-field. The result is stored in the B-field.

WORD MARKS The B-field must have a defining word mark, because it is this word mark that stops the operation.

The A-field must have a word mark only if it is shorter than the B-field. In this case, the transmission of data from A to B stops when the A-field word mark is sensed. Carries within the B-field are completed.

If the A-field is longer than the B-field, the high-order positions of the A-field that exceed the limits imposed by the B-field word mark are not processed.

If the A- and B-fields have like signs, the result has the sign of the B-field. If the signs are different, the result has the sign of the one that is larger.

If the fields to be added contain zone bits in other than the high-order position of the B-field and the sign positions of both fields, only the digits are used in a true-add operation. B-field zone bits are removed except for the units and high-order positions in a true-add operation. If the A- and B-fields have unlike signs, a complement add takes place, and zone bits are removed from all but the units position of the B-field.

If an overflow occurs during a true-add operation, a special overflow indicator is set, and the overflow indications are stored over the high-order digit of the B-field:

Condition	Result
First overflow	A-bit
Second overflow	B-bit
Third overflow	A- and B-bits
Fourth overflow	No A- or B-bits

For subsequent overflows repeat conditions 1-4.

The Branch If Indicator On (B xxx Z) instruction tests and turns off the overflow indicator and branches to a special instruction or group of instructions if this condition occurs. There is only one overflow indicator in the system. It is turned off by a Branch If Indicator On instruction.

TIMING

1. If the operation does not require a recomplement cycle:
   $T = .0115\ (L_I + 3 + L_A + L_B)$ ms.
2. If a recomplement cycle is taken:
   $T = .0115\ (L_I + 3 + L_A + 4L_B)$ ms.

Subtraction, as might be expected, is very similar to addition. The A word is subtracted from the B word; the difference replaces the B word in storage. The word mark requirements are the same as with addition: if the fields are of the same length, the A may have a word mark but need not; if the A-field is shorter, both must have word marks. (And in any case the A-field cannot be longer, because the B-field word mark stops the operation.)

Subtraction is algebraic, as is addition. The sign of the result depends on the signs of the two fields and on which of them is larger, as shown below.

Subtract

FORMAT

Mnemonic	Op Code	A-address	B-address
S	S	xxx	xxx

FUNCTION

The A-field is subtracted algebraically from the B-field. The result is stored in the B-field.

A-field sign	+	+	−	−
B-field sign	+	−	+	−
Sign of result	+ if B-field value greater − if A-field value greater	−	+	+ if A-field value greater − if B-field value greater

WORD MARKS

A word mark is required to define the B-field. An A-field requires a word mark only if it is shorter than the B-field. In this case, the A-field word mark stops transmission of data from A to B.

TIMING

1. Subtract—no recomplement:
   $T = .0115\ (L_I + 3 + L_A + L_B)$ ms.
2. Subtract—recomplement cycle necessary:
   $T = .0115\ (L_I + 3 + L_A + 4L_B)$ ms.

Review Questions

1. *On addition and subtraction, when must the A-field have a word mark?*
2. *If the A-field is shorter than the B-field, why can the execution of an Add instruction not stop when the end of the A-field is reached?*
3. *What is the sign of the result when a large negative number is subtracted from a small negative number?*

Exercises

*1. Given the following storage contents:

Show the result of executing:

M 799 200
M 796 196

2. Given the following storage contents:

Show the result of executing A 339 881.

3. Given the storage contents:

Show the result of executing A 339 881.

*4. Given the storage contents:

Show the result of executing S 497 508.

5. Given the storage contents:

Show the result of executing S 500 600.

*6. Write a program segment to read a card and then punch another card with the same information.

7. Write a program segment to read a card and print part of the information in it, as follows:

Card Columns	Print in print positions
2-7	2-7
8-11	15-18
43	22
44-70	30-56

*8. Write a program segment to read a card and print a line as follows:

Print positions	Print
1-8	Contents of card columns 1-8
20-22	Contents of card columns 10-12
30-35	Sum of contents of card columns 15-18, 19-22, and 23-26.

9. Shown below are a report form and a card form. Write a program segment to read such a card and print the required information as shown on the report.

Notice that leading zeros have been deleted in printing: where the card might have 09285, the report shows 9285. This *zero suppression* is obtained by using the *Move Characters and Suppress Zeros* instruction, which has the operation code Z. The A-field must have a word mark; word marks in the B-field have no effect and in fact are erased.

ACCOUNTS RECEIVABLE REGISTER — **HADUN MFG. CO.**

CUSTOMER NUMBER	CUSTOMER NAME	LOCATION		INVOICE NUMBER	INVOICE DATE		INVOICE AMOUNT
		STATE	CITY				
9285	COVENTRY OIL	16	67	11509	11	8	59295
9285	COVENTRY OIL	16	67	12292	12	23	95097
21217	CRESCENT BELT CO	36	471	11511	11	8	33563
21217	CRESCENT BELT CO	36	471	12263	12	17	12175
29031	DENNIS MFG CO	6	63	11615	11	14	44012
29131	DENNIS MFG CO	6	63	11676	11	23	72222
29134	DEPOSIT DIST CO	25	39	9689	9	11	64540
29134	DEPOSIT DIST CO	25	39	11605	11	11	27169
29134	DEPOSIT DIST CO	25	39	12234	12	14	55933

Print Positions 7-11 | 16-37 | 44-45 | 49-51 | 56-60 | 64-65 | 68-69 | 74-80