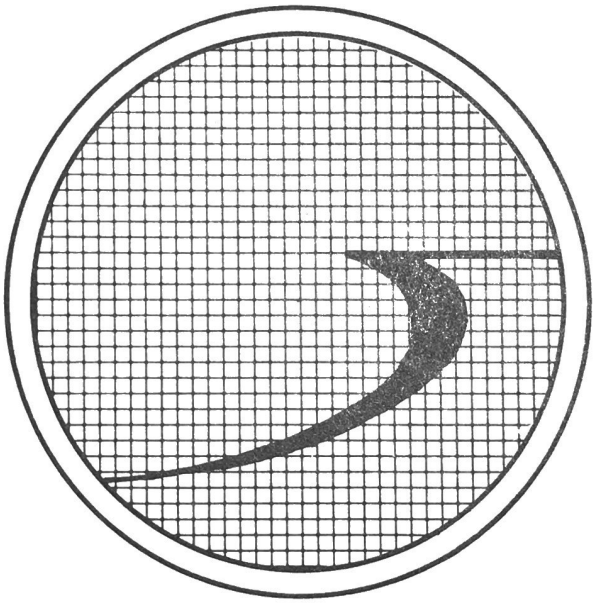
INTRODUCTION TO PROGRAMMING OF A DIGITAL COMPUTER

By James A. Pederson



REPRODUCED FROM THE

NSA TECHNICAL JOURNAL, JULY 1957

Introduction To Programming of A Digital Computer

BY JAMES A. PEDERSON

Unclassified

A discussion of the fundamentals of computer programming, including a simple application of a computer to routine statistical analysis.

I. INTRODUCTION:

The intent of this paper is to provide an introduction to the programmer's art and give an appreciation of the process of thinking that is involved in solving a problem on a computer.

A brief description of the function of the various major components of a digital computer will be given. A simple decimal computer will be hypothesized and its command code and the characteristics will be given. Next, we shall discuss the sort of analysis which takes place in reducing a problem to computer terms. A few examples will be included of the various machine techniques which have been developed by programmers to avoid the frustrating and tedious task of specifying each separate instruction. Finally, a sample program will be given which will show the application of the computer to routine statistical analysis.

II. LOGICAL DESCRIPTION OF AN AUTOMATIC DIGITAL COMPUTER.

Most automatic digital computers may be logically divided into five components. The components are the input devices, the storage unit, the arithmetic unit, the output devices, and the control unit. The interrelationship of these units with respect to information flow is shown in Fig. 1.

The input devices are the means of translating information from a form generated by human operators to a form suitable for use within the computer. The basic unit of input information is the character. A character may be thought of as any symbol or other result of striking a typewriter key. The term thus includes the letters of the alphabet, the digits 0 to 9, and the various punctuation and format control operations such as spacing, tabulation, and carriage return.

Keyboards, punched paper tape readers, punched card readers, and magnetic tape reading devices are typical methods of input. Present research is also developing devices which will read information directly

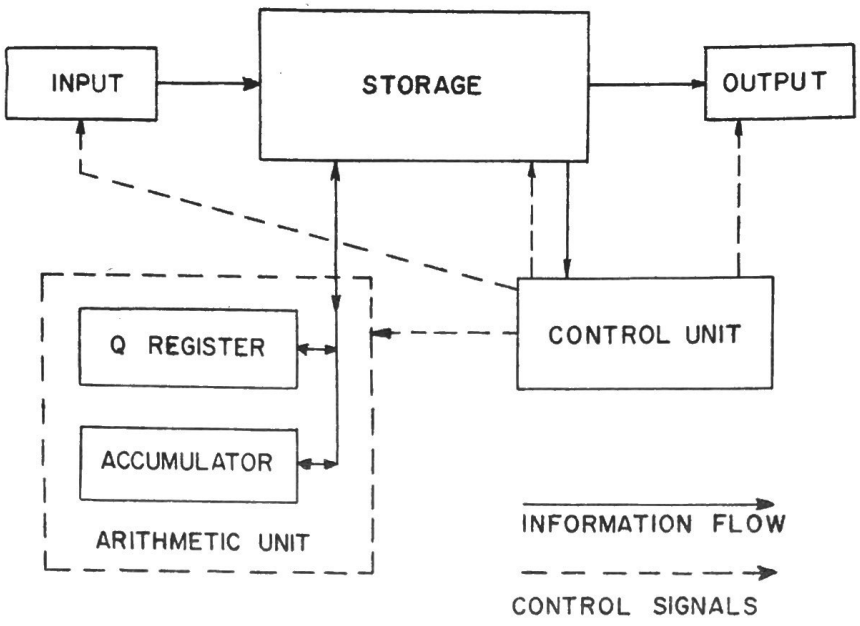


Fig. 1

from the printed page instead of depending on a human being to read the information and actuate a keyboard. At the input device, the input information is converted from a key depression or the sensing of a hole into an electrical, coded form which is used internally in the computer.

The information is then retained in the storage unit for further processing by the rest of the computer.

The storage unit provides a means of retaining information within the computer for varying lengths of time. This information includes input information, instructions to be interpreted by the control unit, data used in the program, intermediate results of the computation, and final results ready for output.

The unit of information in the storage unit is a *word*,—a number, an instruction, or a number of characters. The size of the word is fixed by the design of any one computer[1]. In present computers, the word sizes range from 4 to 9 characters which may represent from 7 to 15 decimal digits.

The storage unit is divided into discrete storage locations. Each storage location (occasionally abbreviated hereafter to SL) will retain one word of information. Each storage location is uniquely identified

[1] With the exception perhaps of the IBM 702 and 705, which are called variable word-length computers. My own contention however is that the word in these machines is one character in length, and the variable feature is that the instructions may be designated to operate on a variable number of these one-character words.

by an *address* which makes it possible to locate any desired word of information.[2]

The arithmetic unit is capable of performing the basic mathematical operations of addition, subtraction, multiplication, and division as well as some specialized logical operations. It receives information from, and returns the results of arithmetic operations to, the storage unit. The operations to be performed by it are directed by the control unit as specified by the instructions in the program.

The output devices accept electrical signals representing information from the storage unit of the computer and convert them to a form which is suitable for utilization, either directly or indirectly, by the ultimate human user of the computer.

The most direct output is in the form of printed matter generated by an electric typewriter or by a line printer. Indirect output may be in the form of punched cards, punched paper tape, or magnetic tape. These three forms of data may in turn be used to actuate electric typewriters, or line printers separate from the computer, to produce the printed output; or they may be used at a later time as input to the same or different computers.

The control unit examines instructions from sequential locations in the storage unit, one at a time, and causes the input, output, and arithmetic units to perform the operations specified.

The fact that the control unit can examine only one[3] instruction at a time is the source of much of the difficulty of programming. Since only one operation can be performed at a time, each problem must be broken down into a series of discrete steps which must be performed in sequence.

---

[2] It may help to think of the storage unit of the computer as a very special type of filing cabinet. Each drawer of the cabinet corresponds to a storage location and will hold one card (or word) of information. Each drawer is labeled with its address. To locate information from this "storage unit" the address of the information is given. Then by going to the drawer with this address label, the information may be read from the card contained in the drawer.

[3] There are several forms of instruction codes in popular use. The major ones are *one-address, two-address, three-address, one-plus-one-address and three-plus-one-address.* The first three get successive instructions from sequential storage locations except on the occurrence of a "jump" instruction which starts a new sequence. The location of the next instruction to be performed is specified by a number contained in a special register (called a *program address counter*) which increases by one each time an instruction is executed. A "jump" instruction causes an entirely new number to replace the previous one in the program address counter. The *one-plus-one* and *three-plus-one* address codes have an additional address which specifies the location of the next instruction. Thus, in this case, instructions are not necessarily found in sequential storage locations.

III. A SIMPLE DECIMAL COMPUTER.

We will hypothesize a simple decimal computer for use in the rest of this paper. I have chosen this attack, instead of using an existing computer, to avoid getting snarled in the complexities of binary arithmetic and machine idiosyncrasies, and thus losing sight of basic programming principles.

Since all computers must have a name, we will call this one the SIDAC from the first letters of Simple Integral Decimal Automatic Computer. This is not stretching the point as much as the Atomic Energy computer called ORACLE from the name, Oak Ridge Automatic Computing and Logical Engine.[4]

Input to SIDAC is from a punched paper tape. To simplify our programming we shall assume that only decimal information may be put into the machine, although alphabetic information could be used as input without too much trouble. The punched paper tape is previously prepared by an operator using a keyboard and following a set format specified by the programmer.

Storage for SIDAC is capable of retaining 1,000 words of information. The storage locations are addressed from 000 to 999. Each word in the storage unit is a ten-decimal-digit quantity plus a sign indication. Numbers will be retained in the storage unit in the form of the absolute value of the number, plus a sign-indication for use by the arithmetic unit. Instructions will appear as ten digit numbers with a positive sign. The ten-digit word may also be considered to be just a collection of ten symbols in the form of decimal digits with no numeric significance.

Arithmetic is performed in the arithmetic unit by the use of two *registers* or specialized one-word storage units. These registers are the *accumulator* (referred to as A) and the *quotient register* (referred to as Q). The accumulator will receive information (usually a number) and retain it, add new information to it or subtract new information from it. In conjunction with the Q register, it is also used for multiplication and division. The quotient register is used for multiplication and division and also in some of the logical instructions mentioned above. For convenience, we will say that a number consists of a sign digit, five digits of integral information, a decimal point (implied) and five digits of fractional information. This means that the largest

---

[4] The frequent occurrence of the letters *AC* in computer names as in BINAC, UNIVAC, ILLIAC, SEAC, SWAC, MIDAC, FLAC and others has led one wag to remark that Dr. Howard Aiken, one of the pioneers in the computer field, should design a new machine to be called the AIKEN *B*inary *A*utomatic *C*omputer, so that the name could be abbreviated to the AIKEN BAC. Another wag, on hearing of the Illinois Automatic Computer (Illiac), suggested the foundation of the *S*ociety for *A*utomatic *C*omputation *R*esearch *O*n the Illinois Automatic Computer.

value which can be represented in SIDAC without fancy programming is plus 99999.99999, which is sufficient for most practice problems.

The *Output* from SIDAC is on a directly connected electric typewriter. The output instructions in the instruction list specify its method of operation. We could include more output devices for completeness, but a typewriter will suffice for the sample problem.

The *Control* of SIDAC interprets a one-address type of instruction, which contains two pieces of information:

1. The address of the storage location which contains the operand for the instruction.

2. The operation to be performed. This is done by specifying one of the arithmetic registers or input-output devices and how it is to be used with the operand from storage.

The first instruction of a program will always be placed in storage location 000. The control unit will examine the instruction in that location, perform it and find the next instruction in the next sequential location (in this case 001). This process continues until a "jump" instruction, as described in footnote 3, takes effect.

Below is a list of the instructions which SIDAC will perform. This list gives the numeric code of the instruction, the name of the instruction, an abbreviation to make the function of the instruction easier to remember, and a definition of the instruction. The actual form of the instruction will be given in a discussion of word structure.

The following terminology will be used in the instruction list:

s The 3-digit address of a storage location.

(s) The number or information contained in storage location s.

(A) The number or information contained in the accumulator.

(Q) The number or information contained in the quotient register.

Name	Numeric Form	Mnemonic Form	Description
Load A.	01 s	LDA s	Replace (A) with (s)
Add	02 s	ADD s	Form in A the sum (A)+(s)
Subtract	03 s	SUB s	Form in A the difference (A)−(s)
Store A	04 s	STA s	Replace (s) with (a)
Load Q	11 s	LDQ s	Replace (Q) with (s)
Multiply	12 s	MUL s	Form in A the product (Q)×(s)[5]

Name	Numeric Form	Mnemonic Form	Description
Divide	13 s	DIV s	Form in Q the quotient (A) ÷ (s)[5] (The remainder will be found in A.)
Store Q	14 s	STQ s	Replace (s) with (Q)
Input Character	21	INC	Replace the rightmost digit of (Q) with the next character from the input device.
Input Word	22	INW	Replace (Q) with the next word (10 digits) read from the input device.
Output Character	23	OTC	Cause the typewriter to print the rightmost digit of (Q).
Output Word	24	OTW	Cause the typewriter to print (Q)
Output Format	25 s	OTF s	Cause the typewriter to perform the format function specified by the values of s. The functions available and values of s to specify them are:
Unconditional Jump	31 s	UJP s	Obtain the next instruction from Storage Location s.
Negative Jump	32 s	NJP s	If (A) is negative, obtain the next instruction from SL s.
Zero Jump	33 s	ZJP s	If (A) is zero, obtain the next instruction from SL s.
Stop	40	STP	Stop computation.

Function	Value of s
space	1
carriage return	2
tabulate	3
period (.)	4
comma (,)	5

[5] The relationship of Q and A will seem a little less mysterious if one thinks of multiplication as continued addition going on in A up to the number of times specified in Q, and division as continued subtraction in A, the number of times being recorded in Q.

The instructions all operate in the same manner by taking information from a source, or two sources, performing an operation and placing the result in a destination. The reader should note that the instructions never destroy a number in the source, but only in the destination. Thus if (A) is x and (s) is y, the effect of LDA s is that (A) becomes y and (s) remains y.

WORD STRUCTURE

The basic ten-character word of the computer may be interpreted as a number, an instruction, or data. The structure of the interpretation of the word is as follows.

Number	±	X X X X X . X X X X X
Instruction	+	X X 0 0 0 0 0 Y Y Y
Data		X X X X X X X X X X

The leftmost digit, labeled ±, is the sign digit. It is 0 for a positive number and 1 for a negative number. In the instruction the two digits XX are the numeric form of the operation code. The next five digits are zeros and usually are not written on the program form, while the three Y digits are used to specify the address of the storage location containing the operand used in that particular instruction.

IV. PROGRAMMING METHODS.

The procedure used in attacking a computer problem can best be shown by a simple example of evaluating an equation. The equation to be evaluated is:

$$Y = \frac{(A+B)\,C}{D} - E$$

First the programmer has to break the above function into elemental steps which can be performed successively. One possible sequence of steps is:

1. Add A and B to get $A+B$; save this result in storage.
2. Multiply the result of Step 1 by C to get $(A+B)C$.[6]
3. Divide the result of Step 2 by D to get $(A+B)C/D$; save this result in storage.
4. Subtract E from the result of Step 3 to get the value desired.[6]
5. Store the result and stop.

This sequence corresponds to writing the equation as:

$$Y = \qquad (A+B)C/D \qquad -E$$

Having decided on the sequence of steps to be followed in the program, the programmer then allocates the storage spaces necessary for the

[6] As will presently appear, this result can be used where it stands, and so need not be saved in storage.

solution of the problem. Thus we decide that the values A, B, C, D, and E will be stored in storage locations 100 to 104 respectively, and the final result in storage location 105. In the process it will be necessary to retain the value $A+B$ temporarily before we can perform the multiplication in the arithmetic registers, and similarly $(A+B)C/D$ for subtraction. We therefore reserve 106 to be used for temporary storage. The program will be written starting at storage location 000 and occupying as many storage locations as necessary to complete the problem. The programming is then as follows.[7]

PROGRAM 1

SL	MNEM Code	OP Code		Address	Comment
0 0 0	L D A	0 1		1 0 0	A→Accumulator.
0 0 1	A D D	0 2		1 0 1	A+B.
0 0 2	S T A	0 4		1 0 6	Save A+B.
0 0 3	L D Q	1 1		1 0 2	C→Q register.
0 0 4	M U L	1 2		1 0 6	(A+B)C→Acc.
0 0 5	D I V	1 3		1 0 3	(A+B)C/D→Q register.
0 0 6	S T Q	1 4		1 0 6	Save (A+B)C/D.
0 0 7	L D A	0 1		1 0 6	
0 0 8	S U B	0 3		1 0 4	[(A+B)C/D]−E.
0 0 9	S T A	0 4		1 0 5	Place function in SL 105.
0 1 0	S T P	4 0		0 0 0	

SL	Contents	Comment
1 0 0		Contains A.
1 0 1		Contains B.
1 0 2		Contains C.
1 0 3		Contains D.
1 0 4		Contains E.
1 0 5		Will contain function.
1 0 6		Will contain A+B or $\frac{(A+B)C}{D}$. for temporary storage.

[7] At this point many people ask the question, "How does the information get into the storage units of the computer?" For our purpose it is best to say that if information is written on a programming form it will eventually get into the storage unit

A slight rearrangement of the sequence of steps taken in the program will reduce the number of instructions from ten to six. This sequence, corresponding to writing the equation as

$$Y = [(A+B)/D]C-E,$$

is as follows:

1. Add A and B to get $A+B$
2. Divide the result of Step 1 by D to get $(A+B)/D$
3. Multiply the result of Step 2 by C to get $(A+B)C/D$
4. Subtract E from the result of Step 3 to get the desired value.
5. Store the result and stop.

The source of the saving is that the results of the operations are in the correct arithmetic register for the next operation, so that it is not necessary to resort to the storage unit of the computer. Thus the sum from Step 1 is left in the accumulator. Step 2 performs a division of the number in the accumulator to give a quotient in the Q register. Step 3 multiplies the contents of the Q register by the value C to get the product in the accumulator while Step 4 subtracts E from the results in the accumulator. The program for the above sequence is as follows.

PROGRAM 2

SL	MNEM Code	OP Code		Address	Comment
0 0 0	L D A	0 1		1 0 0	A→Acc.
0 0 1	A D D	0 2		1 0 1	A+B→Acc.
0 0 2	D I V	1 3		1 0 3	(A+B)/D→Q.
0 0 3	M U L	1 2		1 0 2	[(A+B)/D]C→Acc.
0 0 4	S U B	0 3		1 0 4	[(A+B)/D]C−E→Acc.
0 0 5	S T A	0 4		1 0 5	Store function.
0 0 6	S T P	4 0		0 0 0	Stop.

of the computer. Some machines like the ERA 1103 have a special load mode of operation by which program tapes are read into the machine. In this mode the computer is changed to a storage loading device by the flip of a switch, and has no computing ability. Other machines like the IBM 704, and ALWAC are loaded by having a special loading program in the machine which will cause other programs to be read in and thus loaded into the machine. Needless to say, if the loading program is lost from the storage unit of the computer there is hell to pay, and a lot of work involved to reload the loading program.

SL	Contents	Comment
1 0 0		Contains A.
1 0 1		Contains B.
1 0 2		Contains C.
1 0 3		Contains D.
1 0 4		Contains E.
1 0 5		Will contain function.

MODIFICATION OF INSTRUCTIONS.

The main advantage of a computer is in performing repetitious operations. In this case, the programmer can use his imagination to save himself the unnecessary effort of writing out and specifying a large number of similar instructions.

As an example, let us assume that at a certain point in a program it is necessary to transfer 100 numbers from storage locations 500 to 599 to storage locations 900 to 999. To move a number it is necessary to load the accumulator with it and then store it, as it appears in the accumulator, at the desired location.

The following program shows the so called *brute force* method of performing the transfer operation.

PROGRAM 3

SL	MNEM Code	OP Code		Address	Comment
0 0 0	L D A	0 1		5 0 0	Move 1st number.
0 0 1	S T A	0 4		9 0 0	
0 0 2	L D A	0 1		5 0 1	Move 2nd number.
0 0 3	S T A	0 4		9 0 1	
0 0 4	L D A	0 1		5 0 2	Move 3rd number.
0 0 5	S T A	0 4		9 0 2	
.	.	.		.	
.	.	.		.	
.	.	.		.	
1 9 6	L D A	0 1		5 9 8	Move 99th number.
1 9 7	S T A	0 4		9 9 8	
1 9 8	L D A	0 1		5 9 9	Move 100th number.
1 9 9	S T A	0 4		9 9 9	
2 0 0	S T P	4 0		0 0 0	Stop.

There are several things to note about this program. First, every other instruction is similar except that the address of the instruction is being increased by one. Next, it takes two hundred instructions and thus two hundred storage locations for the program to perform the transfer. However, the total transfer operation will only take two hundred instruction-execution times.

In an attempt to improve this situation with respect to the number of instructions required, the programmer will note the pattern of repetition followed in the straightforward manner of programming and also note that the instruction to load *A* from storage location 500 appears in the computer storage as 0100000500, and that by adding the constant 0000000001 to that number he will generate the number 0100000501 which can be interpreted by the machine control as an instruction to load *A* from storage location 501. Similarly, the instruction to store the contents of the accumulator at storage location 901 reads 0400000901, and this in turn can be generated from the instruction in storage location 000 by performing the addition:

$$\begin{array}{r} 0100000501 \\ 0300000400 \\ \hline 0400000901 \end{array}$$

The above mentioned techniques are shown in Program 4.

PROGRAM 4

SL	MNEM Code	OP Code		Address	Comment
0 0 0	L D A	0 1		(5 0 0)[8]	Transfer one number.
0 0 1	S T A	0 4		(9 0 0)[8]	
0 0 2	L D A	0 1		0 0 0	Increase address of instruction in SL 00000 by one.
0 0 3	A D D	0 2		0 5 0	
0 0 4	S T A	0 4		0 0 0	
0 0 5	A D D	0 2		0 5 1	Create new instruction for SL 00001.
0 0 6	S T A	0 4		0 0 1	
0 0 7	U J P	3 1		0 0 0	Repeat sequence.
0 0 8					

[8] Parentheses are used to remind the programmer that the value within them is being modified by the program as it is executed.

SL	Contents	Comment
0 5 0	0 0 0 0 0 0 0 0 0 1	
0 5 1	0 3 0 0 0 0 0 4 0 0	

The only difficulty with the program above is that it is too efficient. It will never stop and will transfer all of the numbers in the storage unit four hundred locations ahead of their original location. This problem is met by checking to see the form of the storing instruction as it appears after moving the last desired number to the series starting at 900.

The argument goes as follows. The instruction in storage location 001 originally is of the form 0400000900 for storing the first number. It is modified to read 0400000901 after storing the first number and before storing the second. Carrying out this logic, the instruction in storage location 001 will read 0400000999 when the last number of the list has been moved to its final resting place. When this instruction is modified and before the next execution, it will have the value 0400001000. The machine can check for this value, and stop the program when it is found. This is done by subtracting the given value from the unknown value and seeing if the result is zero. If it is zero, the zero jump instruction will cause the program to jump to a different storage location and stop the operation of the routine.

Program 5 illustrates this technique.

PROGRAM 5

SL	MNEM Code	OP Code		Address	Comment
0 0 0	L D A	0 1		(5 0 0)	Transfer one number.
0 0 1	S T A	0 4		(9 0 0)	
0 0 2	L D A	0 1		0 0 0	Add one to address of instruction in SL 00000.
0 0 3	A D D	0 2		0 5 0	
0 0 4	S T A	0 4		0 0 0	
0 0 5	A D D	0 2		0 5 1	Generate new instruction in SL 00001.
0 0 6	S T A	0 4		0 0 1	
0 0 7	S U B	0 3		0 5 2	
0 0 8	Z J P	3 3		0 1 0	Check for end of routine.
0 0 9	U J P	3 1		0 0 0	
0 1 0	S T P	4 0		0 0 0	Stop.

SL	Contents	Comment
0 5 0	0 0 0 0 0 0 0 0 0 1	
0 5 1	0 3 0 0 0 0 0 4 0 0	
0 5 2	0 4 0 0 0 0 1 0 0 0	Limit.

EFFICIENCY TIME AND SPACE

The programmer's objective is to develop the most efficient program. Program 5 is aesthetically satisfying in that it requires 14 storage locations and is efficient in terms of the space required for it. It is inefficient, however, in the matter of time, since it requires the execution of 10 instructions for each number on the list and thus requires a total of 1,000 instruction times for completion. Program 3, which does the same job, is inefficient space-wise, requiring 201 storage locations, but is efficient time-wise in requiring only 201 instruction times for completion.

The programmer is constantly faced with the problem of selecting the best method of performing any given task. The fairly limited size of the storage system requires the use of programs which are compact, to conserve storage location for use in other portions of the problem. The cost of time on the computer calls for the fastest possible program, which consumes more storage locations than a more compact and thus slower program. The two conflicting requirements can only be resolved by experience.

FLOW CHARTING

While it is possible to write a program without resorting to any other aids, the intricate detail of the decisions in a large program makes it necessary for the programmer to develop some means of visualizing the operation and layout of his program as he writes it. The writing of a large program is a task of several weeks to several months duration. Often during that time the programmer will forget what he has written in an earlier portion of the program. By meticulously reading the program, it is possible to follow the scheme of operations, but it is more convenient to use a flow chart.

A flow chart consists of a number of boxes containing statements of the operation to be performed at the time the program reaches that box. The flow of the program is indicated by the interconnection of the boxes by lines with arrows. Most boxes will have one output line and possibly several input lines. The jump instructions, such as the zero

jump and the negative jump will be indicated by a box containing one input line and two output lines.

One problem in the use of flow charts is in developing notation. Many computer installations have developed a standard meaning to be assigned to certain shapes of boxes, but the notation used within the boxes is still pretty much of an individual programmer's problem. The flow chart presents a convenient shorthand for the programmer to keep his own program in mind.

COUNTING AND TESTING

Assume that it is necessary to perform some sequence of instructions in the computer ten times. The stupid method of doing this would be to write out the full set each time.[9] A more elegant method is to use the same set of instructions the ten times required. The re-use of instructions may be controlled by the use of a counting sequence of instructions to tie the tail of the program to the beginning.

Assuming that a counter starts at zero, the position of the counter affects the choice of the top limit to be used. Three possibilities for the limit are shown in Fig. 2.

---

[9] At times the "stupid" method is the best method, provided storage space is available. If the same recurrent group of operations is used many times during the execution of a program, the most efficient (time-wise) method of doing the operations would be to write out each instruction as in Program 3.

The programmer would much prefer to write the same problem as in Program 5. The operation of modifying instructions and checking for the end of the repetition operation is known as "housekeeping" or "red tape". A group of instructions which are repeated, with modification, is known as a "loop" of instructions.

To avoid the need to write each instruction in a recurrent group, many programmers design a routine which will take a loop of instructions and derive the "stretched out" program which does the same job as the loop. The stretching process is done only once when the program is placed on the machine, and thus the programmer is happy since he wrote a simple loop while the running time is short since a stretched loop is much faster than the original loop. The increase in speed of a stretched loop over the original loop comes from the elimination of "red tape" instructions at all intermediate steps. Program 3 could be derived from Program 5 by this process.

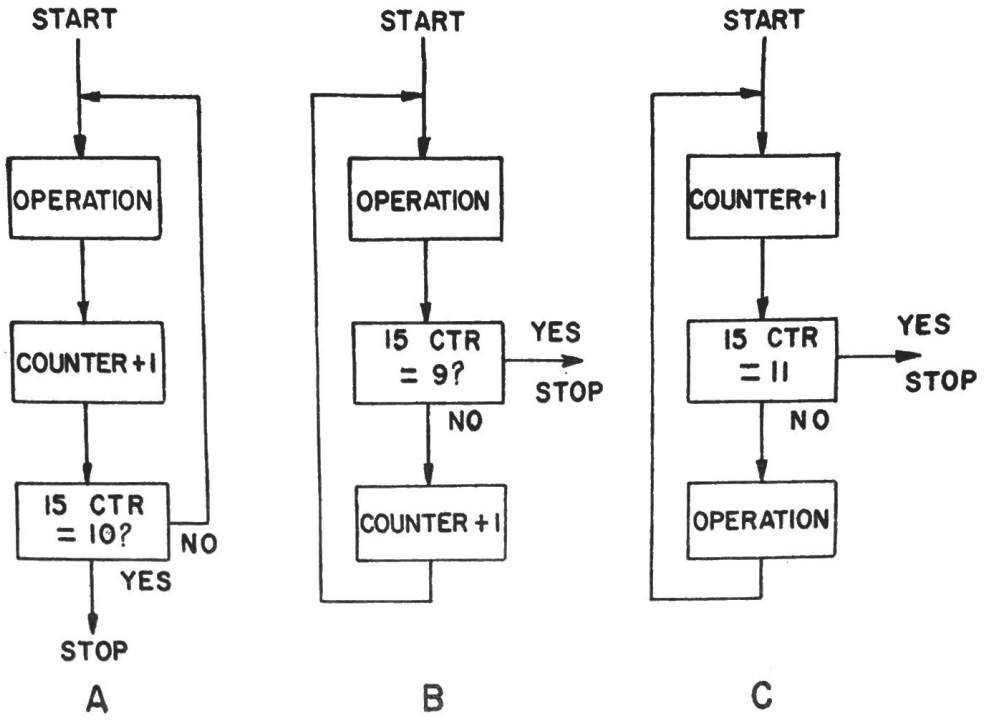


Fig. 2

A program to perform the counting operations shown in the first diagram of Fig. 2 is as follows:

PROGRAM 6

SL	MNEM Code	OP Code		Address	Comment
0 0 0	U J P	3 1		0 1 0	
0 1 0					Routine to be repeated.
0 5 0					
0 5 1	L D A	0 1		0 6 0	Add one to count.
0 5 2	A D D	0 2		0 6 1	
0 5 3	S T A	0 4		0 6 0	
0 5 4	S U B	0 3		0 6 2	Subtract limit.
0 5 5	Z J P	3 3		0 5 7	If zero—stop.
0 5 6	U J P	3 1		0 1 0	If not—do over.
0 5 7	S T P	4 0		0 0 0	

SL	Contents	Comment
0 6 0	(0 0 0 0 0 0 0 0 0 0)	Counter location.
0 6 1	0 0 0 0 0 0 0 0 0 1	Tally for adding.
0 6 2	0 0 0 0 0 0 0 0 1 0	Limit.

Storage location 060 is assumed to contain zero to start with, and will be called our "counter". A "one" contained in storage location 061 is added to storage location 60. This operation is performed by the instructions in 51, 52, and 53. After the instruction in 53 has restored the new count to the counter location, the value of the counter (still contained in the accumulator) is checked against the limit by subtracting the limit from the accumulator. If the two numbers are equal, the result will be zero, and the zero jump instructions will take effect.

V. A SAMPLE PROGRAM: DIGRAPHIC FREQUENCY DISTRIBUTION PLUS STATISTICAL TEST.

The following program is designed to perform a digraphic frequency distribution on 10,000 characters of input data available on the input tape. After the distribution is completed, the statistic

$$\phi = \sum_{i=1}^{N} f_i (f_i - 1)$$

is computed and printed out.

The program is as follows:

PROGRAM 7

SL	MNEM Code	OP Code		Address	Comment
0 0 0	I N C	2 1		0 0 0	Read 1st digit.
0 0 1	M U L	1 2		0 4 0	Shift 1st digit 1 place left in Acc.
0 0 2	I N C	2 1		0 0 0	Read 2nd digit.
0 0 3	S T Q	1 4		0 3 9	Assemble two digits in Acc.
0 0 4	A D D	0 2		0 3 9	
0 0 5	A D D	0 2		0 4 1	Form counting instruction.
0 0 6	S T A	0 4		0 0 9	
0 0 7	A D D	0 2		0 4 2	Form second counting instruction.
0 0 8	S T A	0 4		0 1 1	
0 0 9	L D A	0 1		(1 x x)	Pick up count.
0 1 0	A D D	0 2		0 4 3	Add one.
0 1 1	S T A	0 4		(1 x x)	Store new count.
0 1 2	L D A	0 1		0 4 4	Check for end of input data.
0 1 3	A D D	0 2		0 4 5	
0 1 4	S T A	0 4		0 4 4	
0 1 5	S U B	0 3		0 4 6	
0 1 6	Z J P	3 3		0 2 0	Done—go to statistical.
0 1 7	U J P	3 1		0 0 0	Not done—do over.

SL	Contents	Comment
0 3 9		Temporary storage.
0 4 0	0 0 0 1 0 0 0 0 0 0	Ten.
0 4 1	0 1 0 0 0 0 0 1 0 0	Dummy instruction for forming count.
0 4 2	0 3 0 0 0 0 0 0 0 0	
0 4 3	0 0 0 0 1 0 0 0 0 0	One.
0 4 4		Count.
0 4 5	0 0 0 0 0 0 0 0 0 1	Tally.
0 4 6	0 0 0 0 0 0 5 0 0 0	Limit.
0 4 7		Will contain $\Sigma f_i(f_i-1)$.
0 4 8	0 0 0 0 0 0 0 0 0 1	
0 4 9	0 1 0 0 0 0 0 0 0 0	
0 5 0	0 9 0 0 0 0 0 0 0 0	
0 5 1	0 3 0 0 0 0 0 2 0 0	

SL	MNEM Code	OP Code		Address	Comment
0 2 0	L D Q	1 1		(1 0 0)	Get $f_i^2 - f_i = f_i(f_i - 1)$.
0 2 1	M U L	1 2		(1 0 0)	
0 2 2	S U B	0 3		(1 0 0)	
0 2 3	A D D	0 2		0 4 7	Accumulate $\Sigma f_i(f_i - 1)$.
0 2 4	S T A	0 4		0 4 7	
0 2 5	L D A	0 1		0 2 0	Modify instruction in SL 00020.
0 2 6	A D D	0 2		0 4 8	
0 2 7	S T A	0 4		0 2 0	
0 2 8	A D D	0 2		0 4 9	Make instruction in SL 00021.
0 2 9	S T A	0 4		0 2 1	
0 3 0	S U B	0 3		0 5 0	Make instruction in SL 00022.
0 3 1	S T A	0 4		0 2 2	
0 3 2	S U B	0 3		0 5 1	Done.
0 3 3	Z J P	3 3		0 3 5	
0 3 4	U J P	3 1		0 2 0	No—Do again.
0 3 5	L D Q	1 1		0 4 7	Print $\Sigma f_i (f_i - 1)$.
0 3 6	0 T W	2 4		0 0 0	
0 3 7	S T P	4 0		0 0 0	Stop.

The instruction in SL 000–004 is "Read two digits from the input tape and form a two digit number, which will be between 00 to 99."

The counts of the frequency of occurrence of each of the possible two digit numbers will be made in storage locations 100 to 199. The instructions to perform the count are formed by adding the two digit value to a "dummy" instruction of the form 0100000100, thus forming the pick-up instruction.

After two digits have been read in and counted, a check is made from end-of-input-data by counting for reading-in 5,000 pairs of digits.

When all data have been read in, the statistic ϕ is computed by calculating the value $f^2 - f = f(f-1)$ for each of the counts, and the values are accumulated in SL 00047.

The assumptions in the program are that storage locations 100-199 originally contain zero. It should be noted that if it were necessary to use this program over again, storage locations 100 to 199 should be reset to zero, the instructions in SL 20, 21 and 22 should be reset to their original form, and SL 44 and 47 should be reset to zero.

All of the above resetting could be performed by reading the program in again, or by performing the resetting operation as a separate subroutine of the program.