CONSTRUCTION OF AN LGP-2 7 PROGRAM 3

A computer program consists of a series of step-by-step instructions from the programmer to the computer. To illustrate the basic concept, the following steps would have to be specified to solve the problem below:

$$\left[\frac{(7+8)}{3}\,9\right] -6 = x.$$

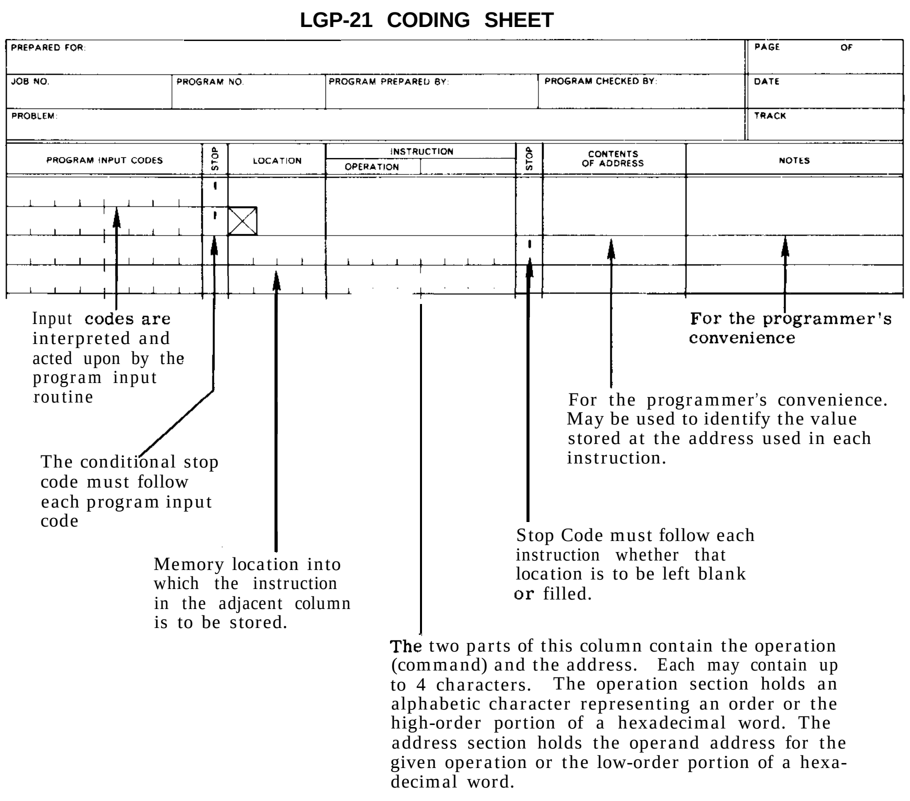
As explained in Chapter 2, "A" is the alphabetic symbol for addition, "M" for multiplication, "D" for division, and "S" for subtraction. According to the definition, each instruction must consist of a command portion which identifies the operation to be performed and an address. Therefore, assuming that the numbers 7, 8, 3, 9, and 6 are stored in memory in locations 0300, 0301, 0302, 0303, and 0304 respectively, the program would look like this:

Step	Order	Address	Notes
1	B	0300	Bring the number 7 to the Accumulator.
2	A	0301	Add 8 $\quad 7 + 8 = 15$
3	D	0302	Divide by 3 $\quad \frac{7+8}{3} = 5$
4	M	0303	Multiply by 9 $\quad (\frac{7+8}{3})\,9 = 45$
5	S	0304	Subtract 6 $\quad ((\frac{7+8}{3})\,9) -6 =$
6	H	0305	Hold the answer in 0305
7	Z	0000	stop

It is important to clearly understand the distinction between the address of a memory location and the contents of that location. An address, such as 0300, refers to a place on the disc, while contents refers to the word recorded at that place.

LGP-21 CODING SHEET

Programs are usually written on LGP-21 Coding Sheets. The sample below (Figure 3.1) shows the general format and explains in detail the purpose of the seven columns provided.



Input codes are interpreted and acted upon by the program input routine

The conditional stop code must follow each program input code

Memory location into which the instruction in the adjacent column is to be stored.

The two parts of this column contain the operation (command) and the address. Each may contain up to 4 characters. The operation section holds an alphabetic character representing an order or the high-order portion of a hexadecimal word. The address section holds the operand address for the given operation or the low-order portion of a hexa-decimal word.

Stop Code must follow each instruction whether that location is to be left blank or filled.

For the programmer's convenience. May be used to identify the value stored at the address used in each instruction.

For the programmer's convenience

FIGURE 3.1 LGP-21 Coding Sheet

The last column, "Notes", should be used to provide all the necessary explanatory information which will be helpful for subsequent reading of a program. The programmer will find it very useful to develop the habit of providing such information.

Anything written in parenthesis on the coding sheet should be read as "the contents of"; an arrow as "replace"; and the abbreviation "Acc." will be used for "Accumulator." For example, (m) is to be read "the contents of memory location m," and (m)$\longrightarrow$(Acc.) is to be read "the contents of memory location m replaces the contents of the Accumulator." This notation will be used throughout the manual.

If the example problem were written on a coding sheet, with the instructions to be stored in locations 1000 through 1006, it would appear as follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION		STOP	CONTENTS OF ADDRESS	NOTES
			OPERATION	ADDRESS			
	'						
	'	☒					
		1000	B	0300	'	7	(0300) $\longrightarrow$ (Acc.) = 7
		1001	A	0301	'	8	7 + (0301) $\longrightarrow$ (Acc.) = 15
		1002	D	0302	'	3	15 ÷ (0302) $\longrightarrow$ (Acc.) = 5
		1003	M	0303	'	☒ 9	5 x (0303) $\longrightarrow$ (Acc.) = 45
		1004	S	0304	'	6	45 - (0304) $\longrightarrow$ (Acc) = 39
		1005	H	0305	'		(Acc.) $\longrightarrow$ (0305)
		1006	Z	0000	'	STOP	

THE 4-PHASE INSTRUCTION CYCLE

At the start of an operation, the computer memory must contain the data to be processed, and the instructions which tell the computer what operations to perform on these data. Ignoring, for the moment, how this information is initially entered into the computer, it need merely be remembered here that any memory location may be used to store one instruction word or one data word. To start execution of the instructions, the programmer specifies the storage location of the first instruction to be executed. After it is found and operated on, the computer automatically takes all successive instructions from sequential memory locations (e.g. , if execution starts at Location 1400, the next instruction will be taken from 1401, then 1402, etc.). The time required for completing a specified operation depends, in part, on the location in memory of the instruction and of its operand, if one is necessary. The process by which the computer obtains and executes an instruction is called an instruction cycle. An instruction cycle begins with a memory search for the instruction word and ends with the commencement of the search for the next instruction word.

The complete cycle consists of four phases:

Phase 1 - Search for the instruction.

Phase 2 - Transfer the instruction from main memory to the Instruction Register and increment the Counter Register by 1.

Phase 3 - Search for the operand.

Phase 4 - Execute the instruction.

SECTOR REFERENCE TIMING TRACK

In order for the computer to find a specific location in memory, a Sector Reference Timing Track is used. This track contains the sector numbers 00 through 127 permanently pre-recorded at the time of manufacture. As explained in Chapter 1, there are actually 32 concentric circles on the disc which are divided into 128 sectors each. However, for programming purposes, sector addresses are numbered 00 through 63. Therefore, on the Sector Reference Timing Track numbers greater than 63 are interpreted modulo 64. For example, sector 97 on the Sector Reference Timing Track represents sector 33 for odd-numbered tracks (i.e. 97 - 64 = 33).

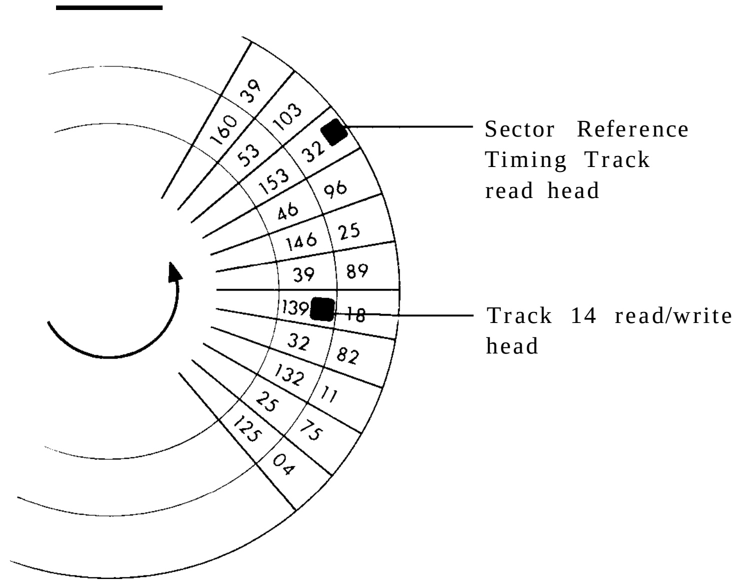


FIGURE 3.2 Sector Reference Timing Track

The Sector Reference Timing Track (Figure 3.2) has only a read-head and cannot be modified by the programmer. The numbers on this track pass under its read-head one sector before the corresponding sector in main memory does. Thus, when a specified sector address is read on the Sector Reference Timing Track, the read/write head on the appropriate track is activated, and the word can be read from or recorded in memory. For example, assume the contents of Location 1432 is to be brought to the Accumulator. Because Track 14 is even-numbered, the Sector Reference Timing Track searches for sector 32. When it is read, read-head 7, which serves Tracks 14 and 15, is activated; and as sector 32 moves under that read-head, its contents is copied into the Accumulator.

This sequence of actions may be more easily understood if two instructions are considered in terms of the instruction cycle. For example:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
		☒					
		1115	B	4458		8	8 (Acc.)
		1116	A	4452		7	8 + 7 (Acc.)

During Phase 1 the Counter Register contains the address 1115. Since 11 is an odd-numbered track, the computer searches the Sector Reference Timing Track for sector 79 (79 - 64 = 15). When it is read, the read-head 5, serving Tracks 10 and 11, is activated, and Phase 1 ends (Figure 3.3).

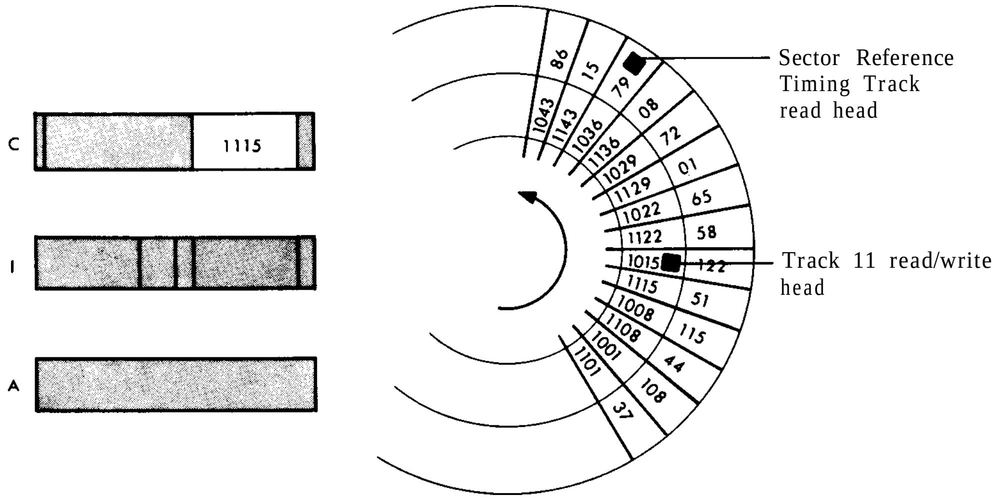


FIGURE 3.3 Instruction Cycle Phase 1

In Phase 2 the contents of Location 1115 is copied into the Instruction Register, and the Counter Register is incremented by 1, so that it now contains 1116 (Figure 3.4).

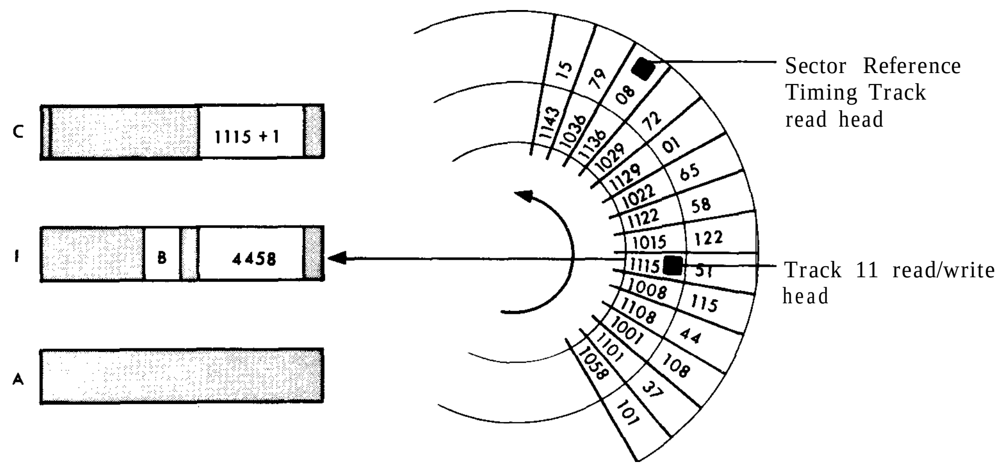


FIGURE 3.4 Instruction Cycle Phase 2

During Phase 3 the computer searches the Sector Reference Timing Track for the operand sector specified in the Instruction Register- that is, sector 58 (Figure 3.5).

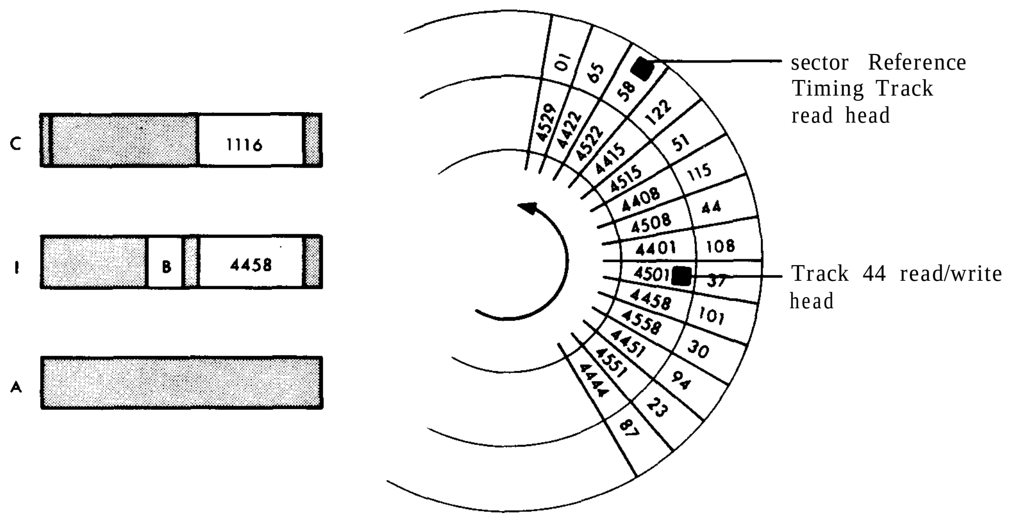


FIGURE 3.5 Instruction Cycle Phase 3

When sector 58 is read, Phase 3 ends, and the computer goes to Phase 4 (Figure 3.6) to execute the instruction B4458. Therefore, the contents of Location 4458 (the number 8) is copied into the Accumulator.

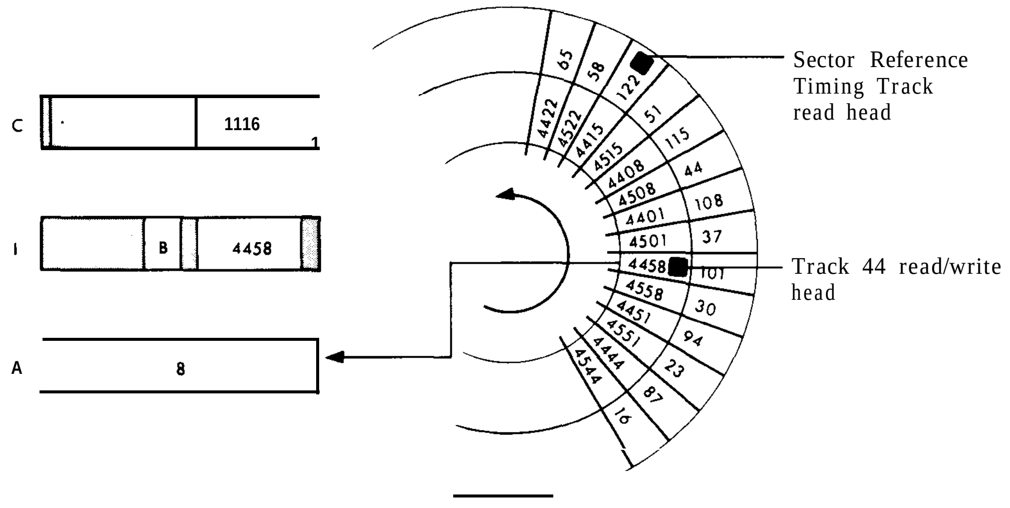


FIGURE 3.6 Instruction Cycle Phase 4

Then the cycle begins again:

Phase	Activity
1	Counter Register contains 1116, therefore search for sector 80 on the Sector Reference Timing Track. When sector 80 is found, activate read-head 5 for Track 11.
2	Copy contents of Location 1116 (A4452) into the Instruction Register. Increment the Counter Register by 1 to 1117.
3	Search for sector 52. When it is read, activate read-head 22 for Track 44.
4	Execute the instruction; that is, add the contents of 4452 (the number 7) to the contents of the Accumulator (8) and leave the result (15) in the Accumulator.

The minimum time required for a complete 4-phase cycle is 18 word-times. A "word-time" is the time required for one word to pass under the read/write head. Since the disc revolves at approximately 1180 rpm, a word-time takes approximately .40 millisecond for the LGP-21. The maximum time for the 4-phase cycle is 146 word-times (one disc revolution plus 18 word-times). Program execution time can be minimized by selecting operand addresses according to a special method which is called "optimizing." This process is explained in Chapter 8. However, optimization is not considered in most of the examples given in this manual.

TRANSFER INSTRUCTIONS

When the computer has to take the next instruction from some location other than the next one in sequence-that is, execute a branch-two types of instructions may be used: an unconditional or a conditional transfer instruction.

The Unconditional Transfer instruction, Urn, tells the computer to branch unconditionally to location m to obtain the next instruction, instead of going to the next location in sequence. After this transfer, the sequential mode is resumed, starting at location m, until another transfer instruction is encountered. If the U instruction is regarded in terms of the 4-phase cycle, it can be explained as follows:

Instruction	Explanation
U	m $\longrightarrow$ (Counter)

Instructions are executed in Phase 4. If m replaces the contents of the Counter in Phase 4, the computer will go to m during Phase 1 of the next 4-phase cycle to obtain the next instruction. The contents of the Counter is replaced by the address portion of the U instruction which is in the Instruction Register during Phase 4. (Note: It is not the contents of m which replaces the contents of the Counter.)

The Conditional Transfer instruction, Tm, tells the computer to branch to location m to obtain the next instruction only if the Accumulator contains a negative word; otherwise, to go to the next location in sequence for the next instruction. If the transfer takes place, the sequential mode is resumed, starting at m, until another transfer instruction is encountered. If the T instruction is thought of in terms of the 4-phase cycle, the instruction can be explained as follows:

Instruction	Explanation
T	If the Accumulator contains a negative word, m $\longrightarrow$(Counter); if the Accumulator contains a positive word (Counter) remains unchanged. In either case (Acc.) remains unchanged.

Phase 3 for U and T instructions is a dummy phase, as a memory search for an operand is unnecessary in conjunction with these two instructions.

Consider a problem using these transfer instructions. The problem requires one of two calculations to be made-the choice depending upon the sign of a certain number. B, C, D, and E are given, and the problem is stated as follows:

If B is positive, calculate $\left[\frac{B}{C}\right]D$ = answer

If B is negative, calculate $\frac{B+E}{C}$ D = answer

Data Storage

Location	Data
0300	B
0301	C
0302	D
0304	E
0400	Answer

The coding for this problem follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		1040	B	0300	'	B → (Acc.)	
		1041	T	1046	'	Test B for	positive or negative
		1042	D	0301	'	(Acc.) ÷ C →	(Acc.)
		1043	M	0302	' ☒	(Acc.) x D →	(Acc.)
		1044	H	0400	'	(Acc.) → 0	400
		1045	Z	0000	'	HALT	
		1046	A	0304	'	(Acc.) + E →	(Acc.)
		1047	U	1042	' ☒	Branch back	to complete calculations
					'		

The T1046 instruction in Location 1041directs the computer to 1042 for the next instruction if B is a positive number. If B is a negative number, the computer branches to Location 1046 to obtain the next instruction. Starting at 1047 it is necessary to execute the same instructions which are in Locations 1042 through 1045; to avoid repeating these instructions, a U1042 instruction in Location 1047 is used to transfer back to them.

INSTRUCTION MODIFICATION AND LOOPING

As already explained, the instruction to be executed is transferred to the Instruction Register during Phase 2 and executed during Phase 4. It should be noted that the computer can only interpret a word as an instruction word when it is in the Instruction Register. An instruction word in any other place is interpreted as a data word. This makes it possible to manipulate instruction words as if they were data words. For example, using the appropriate sequence of instructions, one can bring an instruction word into the Accumulator, modify it in some way (possibly by adding some constant to it), and hold the modified instruction back in its original location. The computer is unaware that it is actually processing an instruction word. The modified instruction word will not be interpreted as an instruction until it is transferred to the Instruction Register during Phase 2 of some subsequent 4-phase cycle; and this will not occur until the address of this instruction is in the Counter Register during Phase 1 of the subsequent 4-phase cycle. This LGP-21 feature—internally stored program operation which permits modification of instructions—can be a very useful programming aid.

Consider this problem:

128 numbers are stored in Locations 0300 through 0463 (Tracks 03 and 04). Compute their sum and store the result in Location 0500.

This problem could be solved by bringing the first of above numbers into the Accumulator with a Bring instruction, then adding the other 127 numbers by using 127 Add instructions, and finally storing the result as specified. This would be a tedious way to code the problem, though it would be a possible approach. However, the program can be reduced to a few instructions by using the instruction modification feature. The coding would be as follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0008	C	0500	'		0→(Acc.)
		0009	C	0500	'	Sum	0→(Acc.)
		0010	B	0500	'	Bring sum to Acc.	
		0011	A	0300	'	☒ Add the next number	
		0012	H	0500	'	Hold the sum in 0500.	

The first instruction will be stored in Location 0008. This is possible since the computer will start execution of the program at any location specified by the programmer.

The first CO500 instruction places whatever is in the Accumulator into 0500 and creates a zero in the Accumulator. The second CO500 instruction (which could just as well be H0500) sets the sum in 0500 to zero. The next 3 instructions bring the sum (zero at this time), add to it the first number (which is in 0300), then hold this answer back in 0500 as the new sum. The next sequence of instructions must effect (1) the address modification of the A0300 instruction in Location 0011, (2) a branch back to Location 0010 to repeat the sequence, and (3) a means of terminating the repetition. This process is called "looping". Thus, the instruction in Location 0011 can be changed to A0301 for the next time it is executed; then changed to A0302, etc. There must be control over the number of times that the instruction is modified and the loop repeated; then an exit from the loop can be made after the 128 numbers have been summed.

Continuing with the coding, the instructions in Locations 0013 through 0015 accomplish the modification of the A0300 instruction in Location 0011:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0008	C	0500	'	Zero→(Acc)	
		0009	C	0500	'	Store zero in 0500 [the sum]	
		0010	B	0500	'	Bring the sum to the Acc.	
		0011	A	0300	'	☒ Add the next number	
		0012	H	0500	'	Hold the sum in 0500	
		0013	B	0011	'	Bring the instruction to be modified to	
					'	the Acc.	
		0014	A	0019	'	☒ Add Z 0001 to the instruction	
		0015	H	0011	'	Hold modified instruction 0011	
		·			'		
		·			'		
		0019	Z	0001	'	☒ Constant used in address modification	
					'		

The BOO11 instruction in Location 0013 brings into the Accumulator, from Location 0011, the instruction to be modified. Now arithmetic operations can be performed on this instruction word as if it were a data word. In the Accumulator is the instruction word A0300, to which another word must be added, so that the instruction word A0301 will be obtained as the result. The A0019 instruction in Location 0014 accomplishes this by adding the contentsof Location 0019 to the contents of the Accumulator and leaving the sum in the Accumulator. This addition takes place:

A 0300 - Initial contents of Accumulator
+ Z 0001 - Plus contents of Location 0019

A 0301 - Final contents of Accumulator

(A "Z" in the command portion of an instruction is treated as a zero by the computer.)

Thus, when the computer is ready to execute the instruction in Location 0015, the Accumulator contains the instruction word A0301. The HO011 instruction in 0015 places the contents of the Accumulator in Location 0011. Therefore, the A0300 instruction in Location 0011 has been replaced by the instruction A0301.

When the sum of the 128 numbers has been accumulated in Location 0500, the program must exit from the loop. The instructions in Locations 0016 and 0017 enable the program to determine whether the loop is to be repeated or terminated:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
		☒					
		0008	C	0500		Zero the A umulator	
		0009	C	0500		Store zero i 0500 (the sum)	
		0010	B	0500		Bring the su to the Acc.	
		0011	A	[0300]*		☒ Add the nex number	
		0012	H	0500		Hold the sum in 0500	
		0013	B	0011		Bring the instruction to be modified to	
						the Acc.	
		0014	A	0019		☒ Add Z 0001 to the instruction	
		0015	H	0011		Hold modified instruction in 0003	
		0016	S	0020		Subtract A 0500 from the instruction	
		0017	T	0010		Return to beginning of loop if (Acc.)	
						☒ negative	
		0018	Z	0000		HALT	
		0019	Z	0001		Constant use in oddress modification	
		0020	A	0500		Constant use to test for end of loop	
						☒	

Before the SO020 instruction in 0016 is executed, the Accumulator contains the A instruction which has just been held in 0011 by the instruction in 0015. What is the address portion of the A instruction now in the Accumulator? It depends on how many times the loop, extending from 0010 through 0017, has been executed. If it has been executed once, the A instruction reads A 0301; if twice, A 0302 and so on. If the loop has been executed 128 times, the instruction reads A0500. The following example shows that the subtraction will yield a negative result whenever the A instruction has an address portion less than **0500:**

A XXXX
-A 0500

Result: Some negative number for any XXXX <0500

* It is good practice to enclose an address with brackets to indicate that it will be modified during the execution of the program. The brackets have no other significance, but make it easier for a programmer to follow the program.

The A instruction is in the Accumulator and in Location 0011 before the SO020 instruction is executed. Whenever this A instruction has an address portion less than 0500, the result of the SO020 instruction in 0016 will be a negative word in the Accumulator. The TOO10 instruction in 0017 will then branch to the beginning of the loop at Location 0010. At the start of the 128th execution of the loop, the A instruction in 0010 will be A0463. Therefore, the instructions from 0010 through 0012 will add in the last number and hold the sum in 0500. The instructions from 0013 through 0015 will modify the instruction in 0011 to read A0500 and leave this instruction in the Accumulator. The 50020 instruction in 0016 will subtract A0500 from this instruction word. For the first time the result will be positive (zero). Therefore, rather than branching back to the beginning of the loop, the TOO10 instruction will allow the computer to exit to the instruction in location 0018, a halt. At this time 0500 will contain the sum of the 128 numbers stored in 0300 through 0463.

A question on elementary arithmetic might have occurred to the reader. If the above program is to work correctly, the following answer must result from the modification of the instruction in 0011:

```
 A 0363  - Initial contents of Accumulator
+Z 0001  - Plus contents of Location 0013
 A 0400  - Final contents of Accumulator
```

If the addition were done according to the rules of decimal arithmetic, the answer would be A 0364. However, there is no address 0364, and the computer gives A 0400 as the answer. This is due to the following rule: when the sector portion exceeds 63, as in 0364, subtract 64 from the sector and add 01 to the track to arrive at the "right" answer.

THE Y INSTRUCTION

The Y instruction stores the address portion of the contents of the Accumulator in location m, replacing the address portion of the word in location m. The remaining bit positions of location m are unchanged.

Instruction	Explanation
Y	Address portion of (Acc.)-address portion of (m); (Acc.) and all but the address portion of (m) remain unchanged.

This allows storage of the address portion of the word in the Accumulator in memory without changing the command portion of the word already there. The most common use of the Y instruction is in address modification. Consider the following problem: Add the contents of 0300 to the contents of 0400 and store the sum in 0500; add the contents of 0301 to the contents of 0401 and store the sum in 0501; and so forth, until all the values in Track 03 have been added to the values in the corresponding sectors in Track 04 and stored in the corresponding sectors in Track 05. The coding for this follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		1100	B	[0300]	'	Add contents of corresponding sectors	
		1101	A	[0400]	'	in Tracks 03 and 04 and hold sum in	
		1102	H	[0500]	'	corresponding sector in Track 05.	
		1103	B	1100	' ☒	Bring instruction from 0000 into Acc.	
		1104	A	1112	'	Add Z 0001 to the instruction in Acc.	
		1105	Y	1100	'	Store modified address into instruction	
					'	in 0000.	
		1106	A	1113	' ☒	Add Z 0100 to the instruction in Acc.	
		1107	Y	1101	'	Store modified address into instruction in	
					'	0001.	
		1108	A	1113	'	Add Z 0100 to the instruction in Acc.	
		1109	Y	1102	' ☒	Store modified address into instruction in	
					'	0002.	
		1110	S	1114	'	Subtract B 0600 from instruction in Acc.	
		1111	T	1100	'	Return to beginning of loop if (Acc.) negative.	
		1112	Z	0001	' ☒	HALT — also used as constant in address	
					'	modification.	
		1113	Z	0100	'	Constant used in address modification.	
		1114	B	0600	'	Constant used to test for end of loop.	
. . .		.			☒		

The instructions in 1100 through 1102 perform the addition described in the "Notes" column. The B1100 instruction in Location 1103 places the contents of Location 1100, B0300, in the Accumulator. Then, the A1112 instruction in Location 1104 adds the contents of Location 1112, 20001, to the contents of the Accumulator, B0300, resulting in B0301, as follows:

```
 B 0300 - Initial contents of Accumulator
+Z 0001 - Plus contents of 1112
-------
 B 0301 - Final contents of Accumulator
```

The YllOO instruction in Location 1105 now replaces the address portion of the instruction word in Location 1100, B0300, by the address portion of the instruction word in the Accumulator, B0301. The result is to change the instruction in 1100 from B0300 to B0301. The YllOO instruction does not alter the word in the Accumulator. Therefore, B0301 remains in the Accumulator.

The A1113 instruction in Location 1106 adds the contents of 1113 to the contents of the Accumulator, as follows:

```
 B 0301 - Initial contents of Accumulator
+Z 0100 - Plus contents of 1113
-------
 B 0401 - Final contents of Accumulator
```

The Y1101 instruction in Location 1107 now replaces the address portion of the word in 1101, A0400, by the address portion of the word in the Accumulator, B0401. The result of this is to change the instruction in 1101 from A0400 to A0401. Notice the command portion, A, of the word in Location 1101 did not change even though it is different from the command portion, B, of the word in the Accumulator.

The A1113 instruction in Location 1108 adds the contents of 1113 to the contents of the Accumulator, as follows:

```
  B 0401 - Initial contents of Accumulator
 +Z 0100 - Plus contents of 1113
 --------
  B 0501 - Final contents of Accumulator
```

The Y1102 instruction in Location 1109 then changes the instruction in 1102 from HO500 to H0501.

The S1114 instruction in 1110 subtracts the contents of 1114 from the contents of the Accumulator. This results in a negative word, as shown below, since B0600 is mathematically larger than B0501.

```
  B 0501 - Initial Contents of Accumulator
- B 0600 - Subtract the contents of 1114
 --------
  Negative Word - Final contents of Accumulator
```

The T1100 instruction in Location 1111 will, therefore, transfer to the beginning of the loop to add the next pair of numbers from Tracks 03 and 04 and store the result in Track 05.

At the beginning of the final pass through the loop, the instructions in 1100 through 1102 read as follows:

Location	Instruction
1100	B 0363
1101	A 0463
1102	H 0563

The final sum, therefore, is stored in 0563, The instructions in 1103 through 1109 then modify the above instructions to read as follows:

Location	Instruction
1100	B 0400
1101	A 0500
1102	H 0600

The S1114 instruction in Location 1110, for the first time, results in a positive word (zero) as follows:

```
    B 0363 - Initial contents of Acc. as a result of the B 1100 instruction in
             Location 1103.
  + z 0001 -
    ------
    B 0400 - Contents of Acc. as a result of the A ill2 instruction in Loca-
             tion 1104.
  + z 0100
    ------
    B 0500 - Contents of Acc. as a result of the A 1113 instruction in Loca-
             tion 1106.
  + z 0100
    ------
    B 0600  - Contents of Acc. as a result of the A 1113 instruction in Loca-
             tion 1108.
  - B 0600
    ------
    ZERO    - Contents of Acc. as a result of the S 1114 instruction in Loca-
             tion 1110.
```

The TllOO instruction in Location 1111, therefore, rather than branching back to 1100 and through the loop again, allows the computer to continue to Location 1112, where it halts.

Notice the ZOO01 instruction in 1112 is used both as a halt, when the loop terminates, and as a constant by the A1112 instruction in Location 1104. This is convenient if the program must be in a limited memory area in the computer. Generally, however, this dual function is not used.

INITIALIZATION

Taking another simple program, compute the product of consecutive pairs of numbers on Track 03 and store these 32 products in Locations 0400 through 0431 as follows: (0300) x (0301) $\longrightarrow$ (0400); (0302) x (0303) $\longrightarrow$ (0401); etc. , through (0362) x (0363) $\longrightarrow$ (0431).

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000	B	[0300]	'	Compute the product of a pair of	
		0001	M	[0301]	'	numbers from Track 03 and store	
		0002	H	[0400]	'	on Track 04.	
		0003	B	0000	'	☒ Bring instruction from 0000 into Acc.	
		0004	A	0014	'	Add Z 0002	
		0005	Y	0000	'	Store modified address into instr. in 0000	
		0006	A	0015	'	Add Z 0001	
		0007	Y	0001	'	☒ Store modified address into instr. in 0001	
		0008	B	0002	'	Bring instruction from 0002 into Acc.	
		0009	A	0015	'	Add Z 0001	
		0010	Y	0002	'	Store modified address into instr. in 0002	
		0011	S	0016	'	☒ Subtract H 0432	
		0012	T	0000	'	Test for end of loop	
		0013	U	1400	'	Transfer to output program	
		0014	Z	0002	'	Constant used in address modification	
		0015	Z	0001	'	☒ Constant used in address modification	
		0016	H	0432	'	Constant used to test for end of loop.	
					'		
					'		

Assume the program has been executed and the products in 0400 through 0431 have been printed out, or otherwise disposed of, so they are no longer needed. With the program still in memory, a new set of data could be stored in Track 03, and the program restarted at Location 0000. Would it make the same calculations on the data and store the answers in Locations 0400 through 0431? In other words, after replacing the old data with new, could the program be restarted and do exactly the same thing the second time? The answer is no, because the instructions in 0000 through 0002 were modified during execution of the program so that, when the program halts, these instructions read:

Location	Instruction
0000	B0400
0001	MO401
0002	HO432

The program is set to process data on Track 04, not Track 03, and to store the products starting at 0432 instead of 0400. These modified instructions must be reset or "initialized" before the program is executed a second time. One method for initializing these instructions is to enter a new program in memory with the same instructions as in the original program. The more efficient and preferred method is to write the program to be "self-initializing" as follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
		☒					
		0000	B	[0300]			Compute the product of a pair of
		0001	M	[0301]			numbers from Track 03 and
		0002	H	[0400]			store on Track 04.
		0003	B	0000	☒		Bring instruction from 0000 into Acc.
		0004	A	0014			Add Z 0002.
		0005	Y	0000			Store modified address into instr. in 0000.
		0006	A	0015			Add Z 0001.
		0007	Y	0001	☒		Store modified address into instr. in 0001.
		0008	B	0002			Bring instr. from 0002 into Acc.
		0009	A	0015			Add Z 0001.
		0010	Y	0002			Store modified address into instr. in 0002.
		0011	S	0016	☒		Subtract H 0432.
		0012	T	0000			Test for end of loop.
		0013	U	1400			Transfer to output program.
		0014	Z	0002			Constant used in address modification.
		0015	Z	0001	☒		Constant used in address modification.
		0016	H	0432			Constant used to test for end of loop.
		0017	B	0024			Bring Z 0300 into Accumulator.
		0018	Y	0000			Initialize instr. in 0000 to read B 0300.
		0019	A	0015	☒		Add Z 0001 resulting in Z 0301.
		0020	Y	0001			Initialize instr. in 0001 to read M 0301.
		0021	B	0025			Bring Z 0400 into Acc.
		0022	Y	0002			Initialize instr. in 0002 to read H 0400.
		0023	U	0000	☒		Branch to beginning of loop.
		0024		Z 0300			Constant used in initializing
		0025		Z 0400			Constant used in initializing

The instructions in Locations 0017 through 0025 are initializing instructions which make the program self-initializing when additional data are to be processed by it. After the program is in memory, it can be executed as many times as desired by starting execution at Location 0017, not 0000. All programs should be self-initializing. The execution of the program then starts at the beginning of the initializing instructions.

SUBROUTINE CONCEPT

The solution to a problem often requires that the same operation be performed more than once. This can be graphically shown in the form of a "Flow Chart" (Figure 3.7):

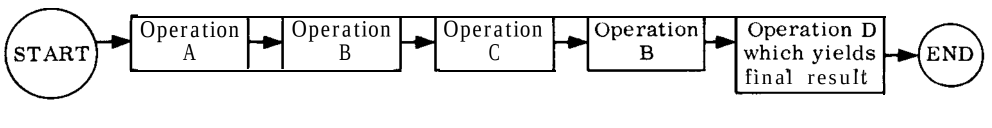


FIGURE 3.7 Typical Flow Chart

Note that the example above constitutes an extreme simplification of a programming flow-chart. In actuality, each operation would be plotted out in every detail, so that Operation A alone might represent a series of steps which could cover an entire page or more.

Assuming now that Operation B is long and involved and the program coded as flow-charted above, it would further be necessary to show the same long sequence of instructions for Operation B twice. Obviously, it would be preferable if the instructions for this operation could be written just once and used again wherever required in the program.

If this is done, the program could transfer (with a U instruction) at the end of Operation A or C to the beginning of the sequence of instructions which performs Operation B. The question now arises, how does one exit from (or branch out of) Operation B to the appropriate place in the program- the beginning of either Operation C or Operation D? The exit instruction from Operation B is a U instruction with a variable address portion and will be set prior to transfer to Operation B. At the end of Operation A and before the transfer to Operation B, the address portion of the U instruction must be set to exit from Operation B to Operation C; at the end of Operation C and before entering Operation B, the address portion of the U instruction must be set to exit from Operation B to Operation D.

This introduces the R instruction:

Instruction	Explanation
R	(Counter) + 1 -address portion of (m); that part of (m) other than the address portion is unchanged.

In other words, the contents of the Counter Register plus 1 replace the address portion of memory location m. At the time an instruction is executed, the Counter contains the location of the next instruction to be executed. Adding 1 to the contents of the Counter when the R instruction is executed gives the location of the R instruction plus 2. Therefore, the R instruction causes its own location plus 2 to replace the address portion of (m). The rest of the word in location m is unchanged.

The skeleton coding for the flow-charted problem could look like this:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000		ANY	'		
		⋮		⋮	'	Operation A	
		0010		ANY	'		
		0011		0040	'	☒ Set exit from Operation B to return to 0013.	
		0012	U	0032	'	Enter Operation B	
		0013		ANY	'		
		⋮		⋮	'	Operation C	
		0020		ANY	'	☒	
		0021	R	0040	'	Set exit from Operation B to return to 0023	
		0022	U	0032	'	Enter Operation B	
		0023		ANY	'		
		⋮		⋮	'	☒ Operation D	
		0030		ANY	'		
		0031	Z	0000	'	End – Halt	
		0032		ANY	'	Entrance Point	Operation B
		⋮		⋮	'	☒	
I I		0039		ANY	'		
I		0040	U	[ ]	'	Exit Point	

Operation A extends from Locations 0000 through 0010. The R0040 instruction in 0011 sets the address portion of the U instruction in 0040 to 0013 (location of R0040 instruction plus 2). The UO032 instruction in 0012 branches to Operation B. A branch to Operation C will occur at the end of Operation B because the U instruction in 0040 now reads UO013.

Operation C extends from Locations 0013 through 0020. The R0040 instruction in 0021 sets the address portion of the U instruction in 0040 to 0023 (location of R0040 instruction plus 2). The UO032 instruction in 0022 transfers to Operation B again. This time, at the end of Operation B there will be a transfer to Operation D, because the U instruction in 0040 now reads UO023.

Operation D extends from Locations 0023 through 0030, and a Halt is at 0031.

Operation B, in Locations 0032 through 0040, is termed a "subroutine," and the instructions in Locations 0000 through 0031 constitute a "source program." The source program may "call" (use) the subroutine any number of times. In the example, the subroutine is only called twice. The entry point to the sample subroutine is Location 0032 and the exit point is 0040. Actually, the entry point does not have to be the first instruction in the written subroutine as in the example, nor does the exit point have to be the last instruction. Subroutines are programs which are used many times. Thus, like all programs, they may start and end anywhere in the written program.

The R-U sequence which is used to call the subroutine is termed a "calling sequence." In the example, the calling sequence consists merely of these two instructions. Some subroutines may require more elaborate calling sequences.

For example, some subroutines may require, before being entered, that certain information to placed in the Accumulator. Also, it is possible to "nest" subroutines to any desired depth; i.e., one subroutine could call another subroutine, which in turn could call still another, and so on. When standard subroutines from the Commercial Computer Division library are acquired, they are accompanied by a program description which details the function of the program, how to load it, what the exact calling sequence must be, and any other information necessary for its operation.