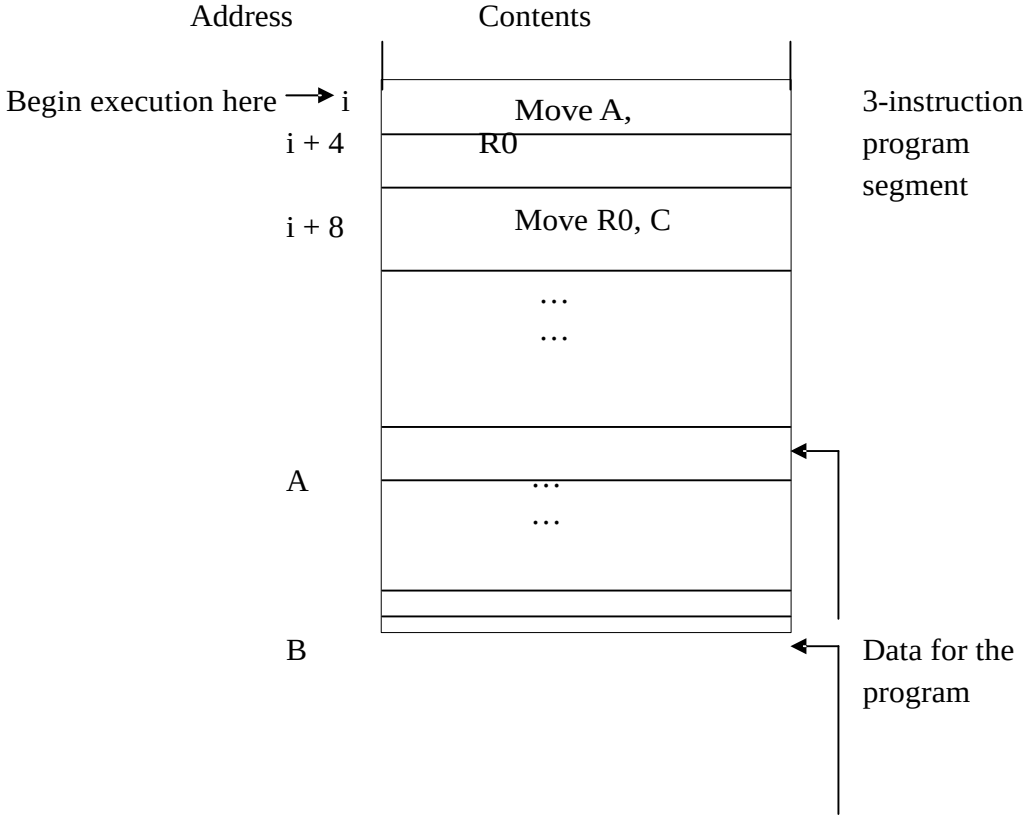
INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

In the preceding discussion of instruction formats, we used to task C ← [A] + [B]. fig 2.8 shows a possible program segment for this task as it appears in the memory of a computer. We have assumed that the computer allows one memory operand per instruction and has a number of processor registers. The three instructions of the program are in successive word locations, starting at location i. since each instruction is 4 bytes long, the second and third instructions start at addresses i + 4 and i + 8.



Let us consider how this program is executed. The processor contains a register called the program counter (PC), which holds the address of the instruction to be executed next. To begin executing a program, the address of its first instruction (I in our example) must be placed into the PC. Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called straight-line sequencing. During the execution of each instruction, the PC is incremented by 4 to point to the next instruction. Thus, after the Move instruction at location i + 8 is executed, the PC contains the value i + 12, which is the address of the first instruction of the next program segment.

Executing a given instruction is a two-phase procedure. In the first phase, called instruction fetch, the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in the instruction register (IR) in the processor. The instruction in IR is examined to determine which operation is to be performed. The specified operation is then performed by the processor. This often involves fetching operands from the memory or from processor registers, performing an arithmetic or logic operation, and storing the result in the destination location.

BRANCHING:-

Consider the task of adding a list of n numbers. Instead of using a long list of add instructions, it is possible to place a single add instruction in a program loop, as shown in fig b. The loop is a straight-line sequence of instructions executed as many times as needed. It starts at location LOOP and ends at the instruction Branch > 0. During each pass through this loop, the address of the next list entry is determined, and that entry is fetched and added to

Move NUM1, R0	i	
Add NUM2, R0	i+4	
Add NUM3, R0		i+8
… …		
Add NUMn, R0		
Move R0, SUM		
	i+4n-4	
…. ….		i+4n
….		

fig a A straight-line program for adding n numbers

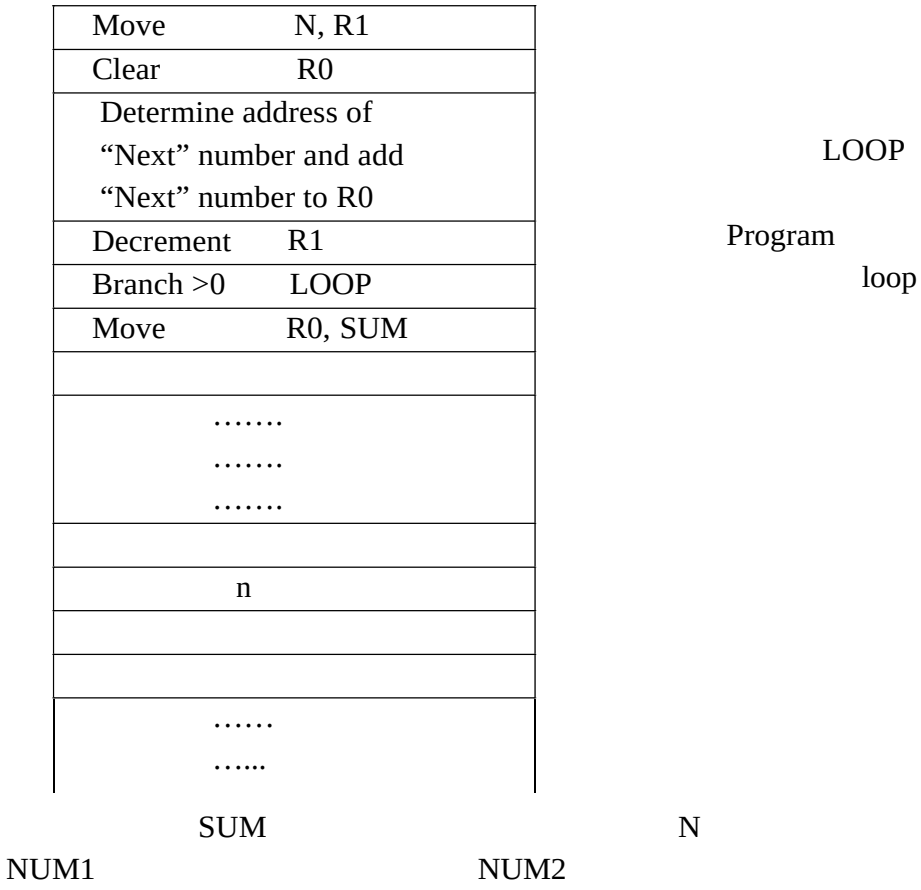


Fig b Using a loop to add n numbers

Assume that the number of entries in the list, n, is stored in memory location N, as shown. Register R1 is used as a counter to determine the number of time the loop is executed. Hence, the contents of location N are loaded into register R1 at the beginning of the program. Then, within the body of the loop, the instruction.

Decrement R1

Reduces the contents of R1 by 1 each time through the loop.

This type of instruction loads a new value into the program counter. As a result, the processor fetches and executes the instruction at this new address, called the branch target, instead of the instruction at the location that follows the branch instruction in sequential address order. A conditional branch instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in

the normal way, and the next instruction in sequential address order is fetched and executed.

Branch > 0 LOOP

(branch if greater that 0) is a conditional branch instruction that causes a branch to location LOOP if the result of the immediately preceding instruction, which is the decremented value in register R1, is greater that zero. This means that the loop is repeated, as long as there are entries in the list that are yet to be added to R0. at the end of the nth pass through the loop, the Decrement instruction produces a value of zero, and hence, branching does not occur.

CONDITION CODES:-

The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions. This is accomplished by recording the required information in individual bits, often called condition code flags. These flags are usually grouped together in a special processor register called the condition code register or status register. Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

Four commonly used flags are

N(negative)	Set to 1 if the result is negative; otherwise, cleared to 0
Z(zero)	Set to 1 if the result is 0; otherwise, cleared to 0
V(overflow)	Set ot1 if arithmetic overflow occurs; otherwise, cleared to 0
C(carry)	Set to 1 if a carry-out results from the operation; otherwise, cleared to 0

The instruction Branch > 0, discussed in the previous section, is an example of a branch instruction that tests one or more of the condition flags. It causes a branch if the value tested is neither negative nor equal to zero. That is, the branch is taken if neither N nor Z is 1. The conditions are given as logic expressions involving the condition code flags.

In some computers, the condition code flags are affected automatically by instructions that perform arithmetic or logic operations. However, this is not always the case. A number of computers have two versions of an Add instruction.

GENERATING MEMORY ADDRESSES:-

Let us return to fig b. The purpose of the instruction block at LOOP is to add a different number from the list during each pass through the loop. Hence, the Add instruction in the block must refer to a different address during each pass. How are the

addresses to be specified ? The memory operand address cannot be given directly in a single Add instruction in the loop. Otherwise, it would need to be modified on each pass through the loop.

The instruction set of a computer typically provides a number of such methods, called addressing modes. While the details differ from one computer to another, the underlying concepts are the same.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iv-computer-organization-10cs46-notes.pdf