SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 19ITT202 – COMPUTER ORGANIZATION AND ARCHITECTURE

II YEAR/ III SEMESTER

UNIT – I Basic Structure of Computers

Topic: Instruction and Instruction Sequencing

Ms.Narmada C

Assistant Professor

Department of Computer Science and Engineering

Instructions

An instruction is a set of codes that the computer processor can understand.

- The code is usually in 1s and 0s, or machine language.
- It contains instructions or tasks that control the movement of bits and bytes within the processor.

A computer must have instructions capable of performing 4 types of operations.

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Instruction Transfer

Instructions must describe the transfer of information from one location in the computer to another.

Possible locations that may be involved in such transfers are memory locations, processor registers or registers in the I/O subsystem.

The two types of instruction notation are

→ Register Transfer Notation

→ Assembly language Notation

Register Transfer Notation

The contents of the location are denoted by placing square brackets around the name of the location.

Example- $R1 \leftarrow [LOC]$

The contents of the memory location LOC are transferred into processor register R1.

$$R3 \leftarrow [R1] + [R2]$$

The contents of registers R1 and R2 is added & placed their sum into register R3.

This type of notation is known as *Register Transfer Notation* (RTN). Note that the right-hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location.

Assembly Language Notation

Assembly language notation is another type of notation to represent machine instruction and programs, which use assembly language format.

Example-

```
Move   LOC,R1
```

The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.

The second example of adding two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement

```
Add    R1,R2,R3
```

Basic Instruction Types

→ Three Address Instruction

→ Two Address Instruction

→ One Address Instruction

Example-

C = A + B

RTN representation **C ← [A] + [B]**

Three Address Instruction

Add A,B,C

Operands A and B are called the *source* operands, C is called the *destination* operand, and Add is the operation to be performed on the operands. A general instruction of this type has the format

Operation Source1,Source2,Destination

Two Address Instruction

Operation Source,Destination

An Add instruction of this type is

Add A,B

which performs the operation B ← [A] + [B]. When the sum is calculated, the result is sent to the memory and stored in location B, replacing the original contents of this location. This means that operand B is both a source and a destination.

The operation can be performed by two instruction sequence

Move B,C

Add A,C

One Address Instruction

A processor register, usually called the accumulator, may be used for this purpose.

Add A

means the following: Add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator. Let us also introduce the one-address instructions

Load A

and

Store A

The Load instruction copies the contents of memory location A into the accumulator, and the Store instruction copies the contents of the accumulator into memory location A. Using only one-address instructions, the operation C ← [A] + [B] can be performed by executing the sequence of instructions

```
Load   A
Add    B
Store  C
```

Note that the operand specified in the instruction may be a source or a destination, depending on the instruction. In the Load instruction, address A specifies the source operand, and the destination location, the accumulator, is implied. On the other hand, C denotes the destination location in the Store instruction, whereas the source, the accumulator, is implied.

Few instruction examples using registers -

Eg 1- Add Ri,Rj,Rk

Eg 2-

```
Move  A,Ri
Add   B,Ri
Move  Ri,C
```

Eg3 -

```
Move  A,Ri
Move  B,Rj
Add   Ri,Rj
Move  Rj,C
```

Instruction Execution And Straight Line Sequencing

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) 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.

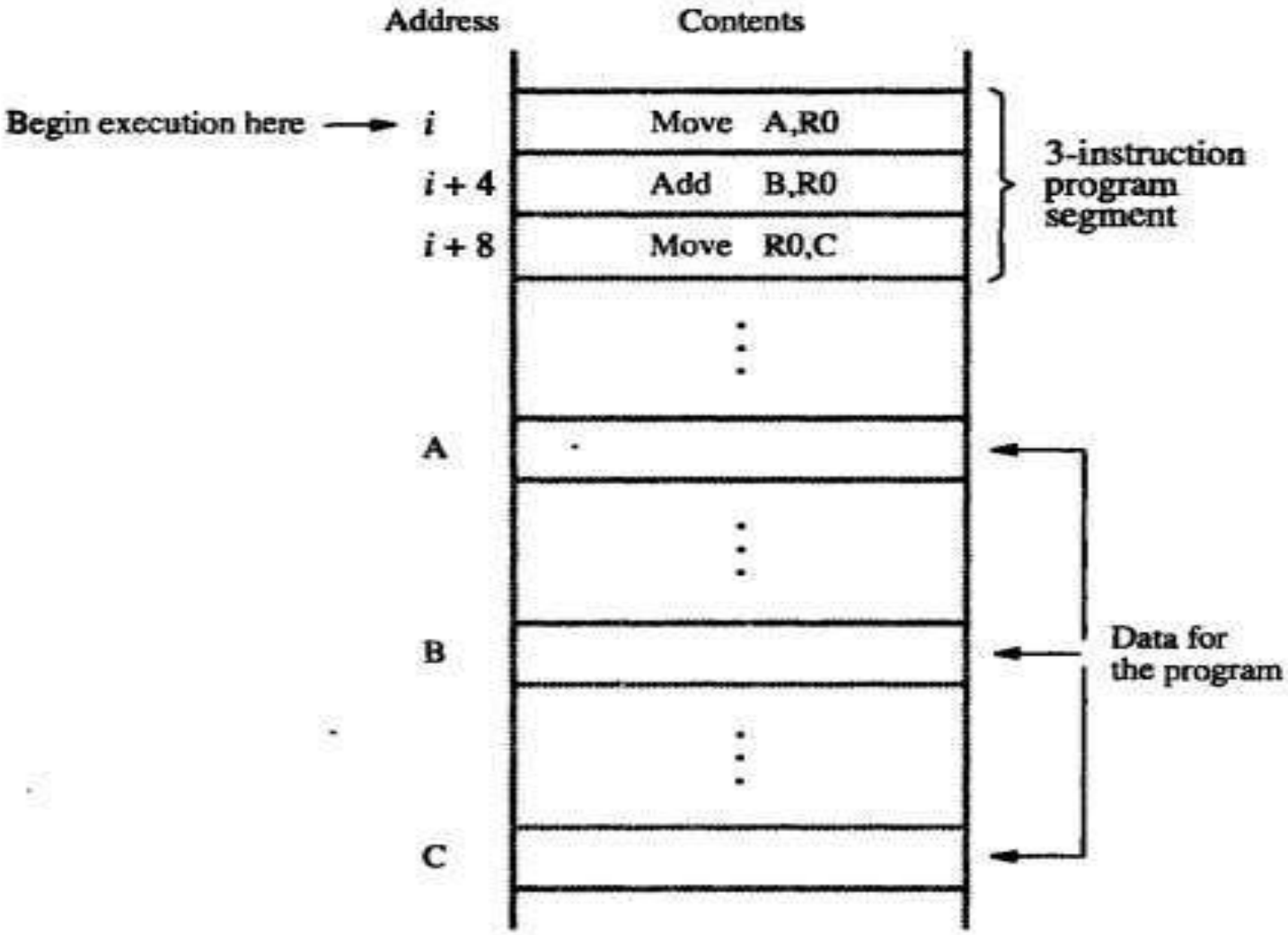

Figure 2.8 A program for C ← [A] + [B].

two-phases of instruction execution

Instruction Fetch

The instruction is fetched from the memory location whose address is in the PC. The fetched instruction is placed in the instruction register(IR) in the processor.

Instruction Execute

At the start of the second phase, called instruction execute, the instruction in IR is examined to determine which operation is to be performed. The specific operation is then performed by the processor. And result is stored in destination location.

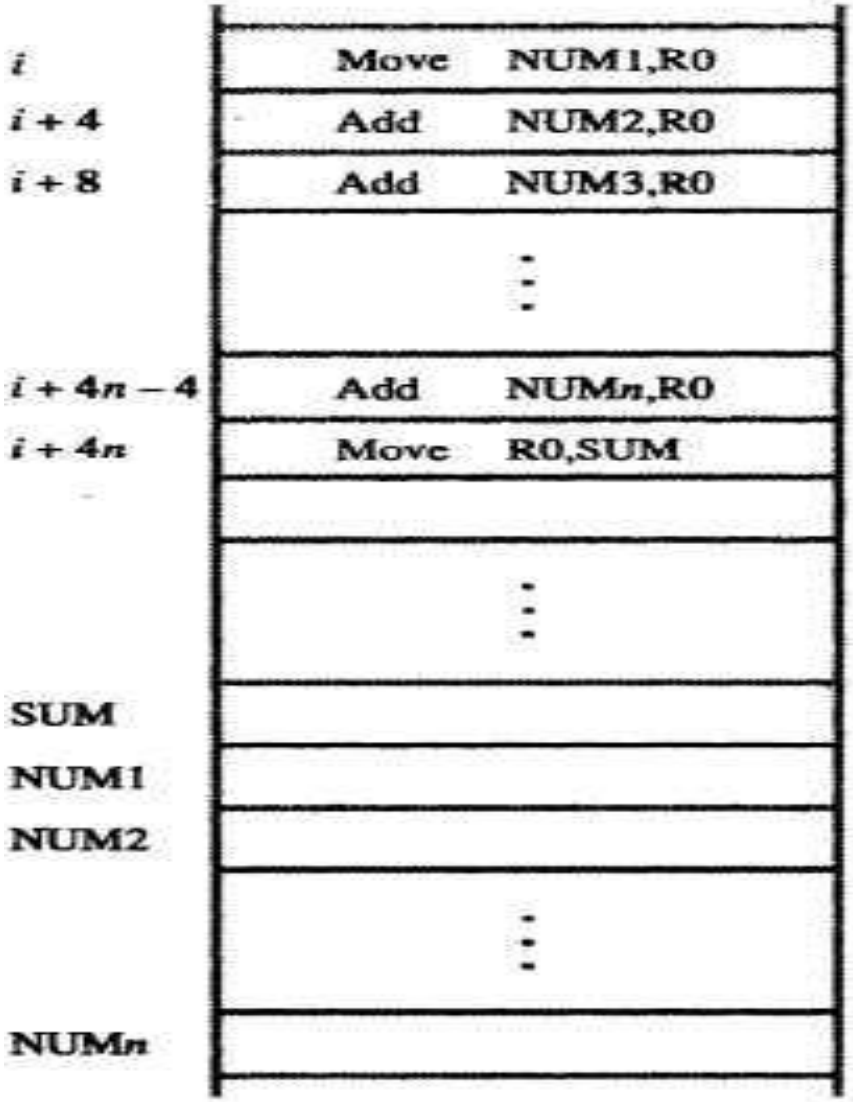

Figure 2.9 A straight-line program for adding *n* numbers.

Branching

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 Figure 2.10. 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 R0.

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 times 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. (A similar type of operation is performed by an Increment instruction, which adds 1 to its operand.) Execution of the loop is repeated as long as the result of the decrement operation is greater than zero.

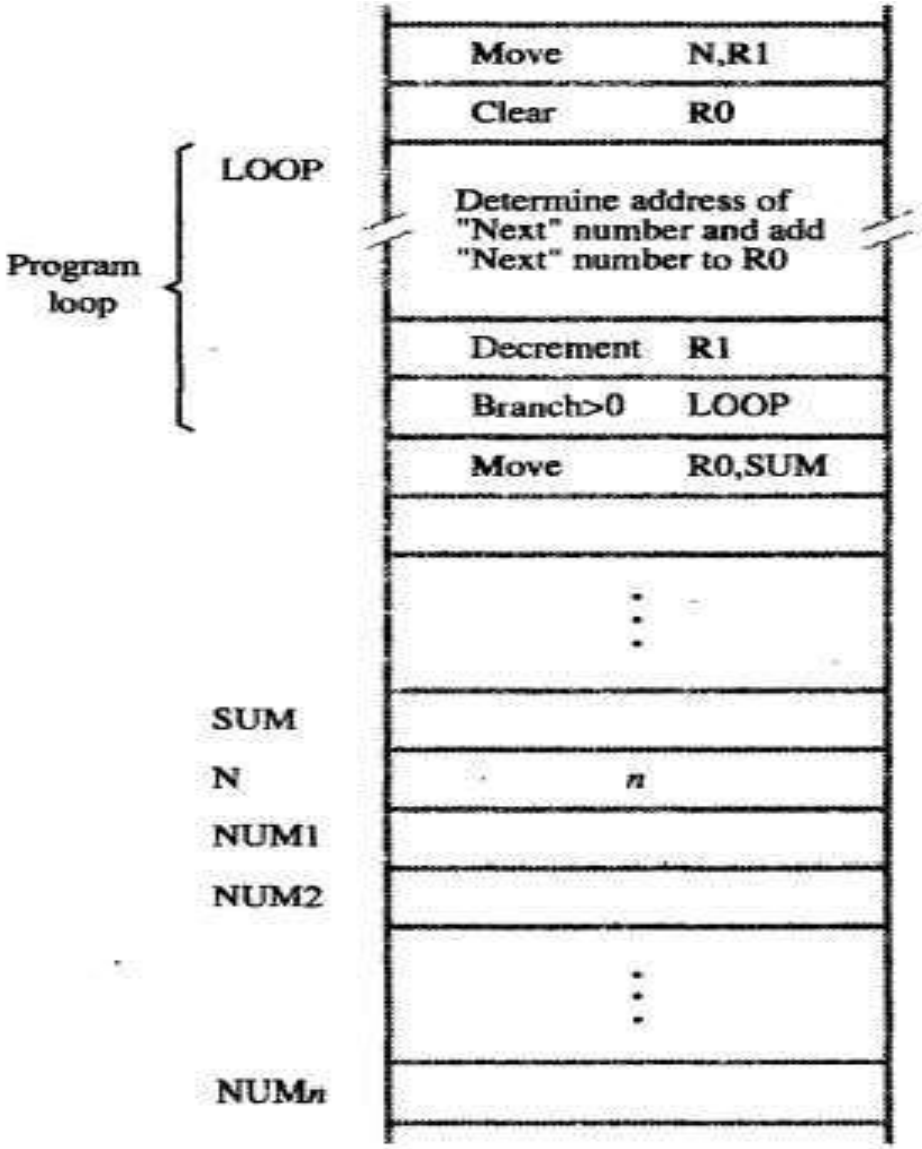

Figure 2.10 Using a loop to add *n* numbers.

We now introduce *branch* instructions. 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.

In the program in Figure 2.10, the instruction

Branch>0 LOOP

(branch if greater than 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 than 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 *n*th pass through the loop, the Decrement instruction produces a value of zero, and, hence, branching does not occur. Instead, the Move instruction is fetched and executed. It moves the final result from R0 into memory location SUM.

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 to 1 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


Thank You!