INSTRUCTIONS OF COMPUTER

A computer must have instructions capable of performing four 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

REGISTER TRANSFER NOTATION:-

Transfer of information from one location in the computer to another. Possible locations that may be involved in such transfers are memory locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem. Most of the time, we identify a location by a symbolic name standing for its hardware binary address.

Example, names for the addresses of memory locations may be LOC, PLACE, A, VAR2; processor registers names may be R0, R5; and I/O register names may be DATAIN, OUTSTATUS, and so on. The contents of a location are denoted by placing square brackets around the name of the location. Thus, the expression

$$R1 \leftarrow [LOC]$$

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

As another example, consider the operation that adds the contents of registers R1 and R2, and then places their sum into register R3. This action is indicated as

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

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 places, overwriting the old contents of that location.

ASSEMBLY LANGUAGE NOTATION:-

Another type of notation to represent machine instructions and programs. For this, we use an assembly language format. For example, an instruction that causes the transfer described above, from memory location LOC to processor register R1, is specified by the statement

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 INSTRUCTIONS:-

The operation of adding two numbers is a fundamental capability in any computer. The statement

$$C = A + B$$

In a high-level language program is a command to the computer to add the current values of the two variables called A and B, and to assign the sum to a third variable, C. When the program containing this statement is compiled, the three variables, A, B, and C, are assigned to distinct locations in the memory. We will use the variable names to refer to the corresponding memory location addresses. The contents of these locations represent the values of the three variables. Hence, the above high-level language statement requires the action.

$$C \leftarrow [A] + [B]$$

To carry out this action, the contents of memory locations A and B are fetched from the memory and transferred into the processor where their sum is computed. This result is then sent back to the memory and stored in location C.

Let us first assume that this action is to be accomplished by a single machine instruction. Furthermore, assume that this instruction contains the memory addresses of the three operands – A, B, and C. This three-address instruction can be represented symbolically as

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, Source 2, Destination

If k bits are needed for specify the memory address of each operand, the encoded form of the above instruction must contain 3k bits for addressing purposes in addition to the bits needed to denote the Add operation.

An alternative approach is to use a sequence of simpler instructions to perform the same task, with each instruction having only one or two operands. Suppose that two-address instructions of the form

Operation   Source, Destination

Are available. An Add instruction of this type is

Add A, B

Which performs the operation $B \leftarrow [A] + [B]$.

A single two-address instruction cannot be used to solve our original problem, which is to add the contents of locations A and B, without destroying either of them, and to place the sum in location C. The problem can be solved by using another two-address instruction that copies the contents of one memory location into another. Such an instruction is

Move B, C

Which performs the operations $C \leftarrow [B]$, leaving the contents of location B unchanged.

Using only one-address instructions, the operation $C \leftarrow [A] + [B]$ can be performed by executing the sequence of instructions

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

Some early computers were designed around a single accumulator structure. Most modern computers have a number of general-purpose processor registers – typically

8 to 32, and even considerably more in some cases. Access to data in these registers is much faster than to data stored in memory locations because the registers are inside the processor.

Let R_i represent a general-purpose register. The instructions

```
Load    A, Ri
Store   Ri, A  and
Add     A, Ri
```

Are generalizations of the Load, Store, and Add instructions for the single-accumulator case, in which register Ri performs the function of the accumulator.

When a processor has several general-purpose registers, many instructions involve only operands that are in the register. In fact, in many modern processors, computations can be performed directly only on data held in processor registers. Instructions such as

```
Add    Ri, Rj
       Or
Add    Ri, Rj, Rk
```

In both of these instructions, the source operands are the contents of registers Ri and Rj. In the first instruction, Rj also serves as the destination register, whereas in the second instruction, a third register, Rk, is used as the destination.

It is often necessary to transfer data between different locations. This is achieved with the instruction

```
Move    Source, Destination
```

When data are moved to or from a processor register, the Move instruction can be used rather than the Load or Store instructions because the order of the source and destination operands determines which operation is intended. Thus,

```
Move   A, Ri
```

Is the same as

```
Load   A, Ri
```

And

```
Move   Ri, A
```

Is the same as

```
Store  Ri, A
```

In processors where arithmetic operations are allowed only on operands that are processor registers, the C = A + B task can be performed by the instruction sequence

```
Move   A, Ri
Move   B, Rj
```

```
Add     Ri, Rj
Move    Rj, C
```

In processors where one operand may be in the memory but the other must be in register, an instruction sequence for the required task would be

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

The speed with which a given task is carried out depends on the time it takes to transfer instructions from memory into the processor and to access the operands referenced by these instructions. Transfers that involve the memory are much slower than transfers within the processor.

We have discussed three-, two-, and one-address instructions. It is also possible to use instructions in which the locations of all operands are defined implicitly. Such instructions are found in machines that store operands in a structure called a pushdown stack. In this case, the instructions are called zero-address instructions.