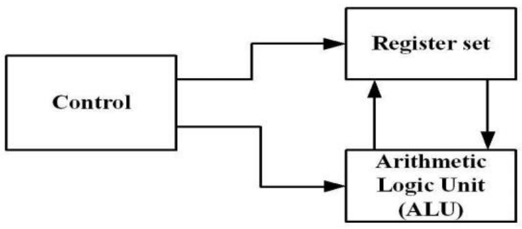
Unit 5:

Central Processing Unit

- The main part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU.
- The CPU is made up of three major parts, as shown in Fig

Central Processing Unit



Major components of CPU

1. The register set stores intermediate data used during the execution of the instructions.
2. The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions.
3. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

Most computers have one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

Single Accumulator Organization:

1. In an accumulator type organization all the operations are performed with an implied accumulator register.

2. The instruction format in this type of computer uses one address field.

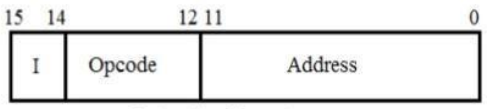
3. For example, the instruction that specifies an arithmetic addition defined by an assembly language instruction as

ADD X here X is the address of the operand.

4. The ADD instruction in this case results in the operation

$AC \leftarrow AC + M[X]$.

5. AC is the accumulator register and M[X] symbolizes the memory word located at address X



Instruction Format

General register organization:

1. The instruction format in this type of computer needs three register address fields.
2. Eg 1; the instruction for an arithmetic addition may be written in an assembly language as ADD R1, R2, R3 This denote the operation

   $R1 \leftarrow R2 + R3$.
3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
4. Eg2; Thus the instruction ADD R1, R2 would denote the operation

   $R1 \leftarrow R1 + R2$.

   Only register addresses for R1 and R2 need be specified in this instruction.
5. General register-type computers employ two or three address fields in their instruction format.
6. Each address field may specify a processor register or a memory word.
7. An instruction symbolized by ADD R1, X would specify the operation

   $R1 \leftarrow R1 + M[X]$.

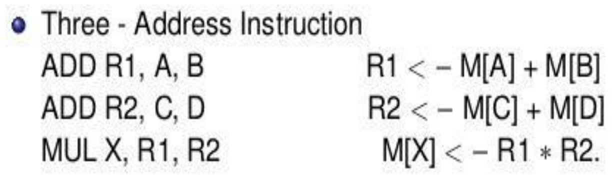
It has two address fields, one for register R1 and the other for the memory address X.

Stack organization

1. The stack-organized CPU has PUSH and POP instructions which require an address field.
2. Thus the instruction PUSH X will push the word at address X to the top of the stack.
3. The stack pointer is updated automatically.
4. Operation-type instructions do not need an address field in stack-organized computers.
5. This is because the operation is performed on the two items that are on top of the stack.
6. The instruction ADD in a stack computer consists of an operation code only with no address field.
7. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack.
8. There is no need to specify operands with an address field since all operands are implied to be in the stack.
9. Most computers fall into one of the three types of organizations.
10. Some computers combine features from more than one organizational structure.
11. To illustrate The influence of the number of addresses on computer programs, we will evaluate the arithmetic statement
    a. X= (A+B) * (C+D)
12. using zero, one, two, or three address instructions.
13. using the symbols ADD, SUB, MUL and DIV for four arithmetic operations.
14. MOV for the transfer type operations;
15. LOAD and STORE for transfer to and from memory and AC register.
16. Assuming that the operands are in memory addresses A, B, C, and D and the result must be stored in memory at address X and also the CPU has general purpose registers R1, R2, R3 and R4.

Three Address Instructions:

1. Three-address instruction formats can use each address field to specify either a processor register or a memory operand.
2. The program assembly language that evaluates X = (A+B) * (C+D) is shown below, together with comments that explain the register transfer operation of each instruction.



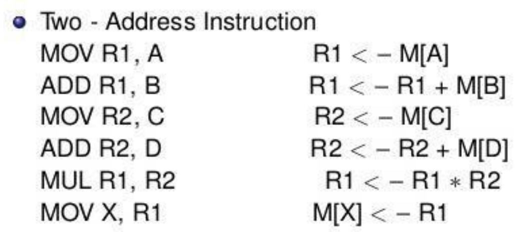
- Three - Address Instruction

```
ADD R1, A, B          R1 < − M[A] + M[B]
ADD R2, C, D          R2 < − M[C] + M[D]
MUL X, R1, R2         M[X] < − R1 ∗ R2.
```

3. The symbol M [A] denotes the operand at memory address symbolized by A.
4. The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.
5. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two Address Instructions:

1. Two-address instructions formats use each address to specify either a processor register or memory word.
2. The program to evaluate X = (A+B) * (C+D) is as follows



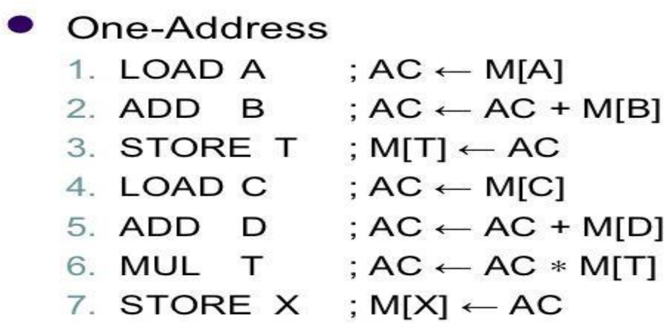
- Two - Address Instruction

```
MOV R1, A          R1 < − M[A]
ADD R1, B          R1 < − R1 + M[B]
MOV R2, C          R2 < − M[C]
ADD R2, D          R2 < − R2 + M[D]
MUL R1, R2         R1 < − R1 ∗ R2
MOV X, R1          M[X] < − R1
```

3. The MOV instruction moves or transfers the operands to and from memory and processor registers.
4. The first symbol listed in an instruction is assumed be both a source and the destination where the result of the operation transferred.

One Address Instructions:

1. One-address instructions use an implied accumulator (AC) register for all data manipulation.
2. AC contains the result of all operations.
3. The program to evaluate X=(A+B) * (C+D) is



- One-Address

```
1. LOAD  A     ; AC ← M[A]
2. ADD   B     ; AC ← AC + M[B]
3. STORE T     ; M[T] ← AC
4. LOAD  C     ; AC ← M[C]
5. ADD   D     ; AC ← AC + M[D]
6. MUL   T     ; AC ← AC ∗ M[T]
7. STORE X     ; M[X] ← AC
```

operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

Zero Address Instructions

1. A stack-organized computer does not use an address field for the instructions ADD and MUL.
2. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
3. The following program shows how X = (A+B) * (C+D) will be written for a stack-organized computer. (TOS stands for top of stack).

- Evaluate *X = (A +B) * (C +D)*

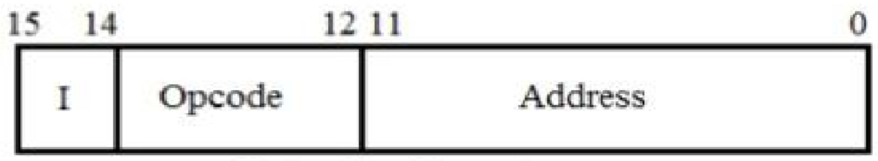
```
• PUSH    A    TOS ← A
• PUSH    B    TOS ← B
• ADD          TOS ← (A+B)
• PUSH    C    TOS ← C
• PUSH    D    TOS ← D
• ADD          TOS ← (C+D)
• MUL          TOS ← (C+D)*(A+B)
• POP     X    M[X] ← TOS
```

Push A
Push B
ADD
Push C
Push D
ADD
Mult
Store

Instruction formats

- An instruction is a group of bits that instructs the computer to do some operation. These bits are arranged in some instruction code format.
- Control unit in the CPU will interpret each instruction code and provide the necessary control functions needed to process the instruction.
- A computer will usually have a variety of instruction code formats.
- The format of an instruction is represented in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be perform
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.



Instruction Format

1. Computers may have instructions of several different lengths containing varying number of addresses.
2. The number of address fields in the instruct format of a computer depends on the internal organization of its registers.

Addressing modes

Operands are chosen during program execution depending on the addressing mode of the instruction. Computers use addressing mode techniques to provide facilities such as pointers to memory, counters for loop control, indexing of data, and program relocation. To reduce the number of bits in the addressing field of the instruction

Types of addressing modes

- Implied Mode
- Immediate Mode
- Register Mode
- Register Indirect Mode
- Autoincrement or Autodecrement Mode
- Direct Address Mode
- Indirect Address Mode
- Relative Address Mode
- Indexed Addressing Mode
- Base Register Addressing Mode

Most addressing modes modify the address field of the instruction; there are two modes that need no address field at all. These are ***implied and immediate modes***

Implied Mode:

- In this mode the operands are specified in the definition of the instruction.
- For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- All register reference instructions that use an accumulator are implied mode instructions.
- Zero address in a stack organization computer is implied mode instructions.

Immediate Mode:

- In this mode the operand is specified in the instruction itself.
- In other words an immediate-mode instruction has an operand rather than an address field.
- Immediate-mode instructions are useful for initializing registers to a constant value.

- Register Mode:
- When the address specifies a processor register, the instruction is said to be in the register mode.
- In this mode the operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction.

Register Indirect Mode:

- In this mode the instruction specifies a register in CPU whose contents give the address of the operand in memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.
- The advantage of a register indirect mode instruction is that the address field of the instruction uses few bits to select a register than would have been required to specify a memory address directly.

Auto-increment or Auto-Decrement Mode:

- This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.

- Direct Address Mode:
- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- In a branch-type instruction the address field specifies the actual branch address.

Indirect Address Mode:

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU.
- The effective address in these modes is obtained from the following computation:
- Effective address =address part of instruction + content of CPU register
- The CPU register used in the computation may be the program counter, an index register, or a base register.
- We have a different addressing mode which is used for a different application.

Relative Address Mode:

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

Indexed Addressing Mode:

- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- An index register is a special CPU register that contains an index value.

Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.