Chapter 6
Central Processing Unit

→ The part of the computer that performs the bulk of data processing operation is called central processing unit (CPU) which consists of ALU, control unit and register array.

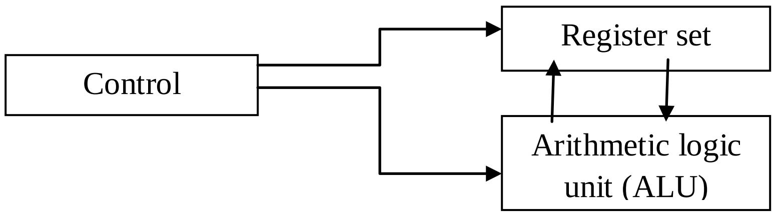


Fig: Major components of CPU

→ CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.

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

6.1 CPU Organizations

There are three types of CPU organization based on the instruction format

1. Single accumulator organization
   - ✓ In this type of organization all the operations are performed with an implied accumulator register.
   - ✓ Basic computer is the good example of single accumulator organization.
   - ✓ The instruction of this type of organization has an address field
     E.g. ADD X $\Rightarrow$ AC←AC+M[X] where X is the address of the operand

2. General register organization
   - ✓ When a large number of processor registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfer, but also while performing various microoperations. Hence, it is necessary to provide a common unit that can perform all the arithmetic, logic and shift microoperations in the processor.
   - ✓ In this type of organization the instruction has two or three address field.
     E.g. ADD R1, X $\Rightarrow$ R1←R1+M[X]
     ADD R1, R2 $\Rightarrow$ R1←R1+R2
     ADD R1, R2, R3 $\Rightarrow$ R1←R2+R3

3. Stack organization
   - ✓ Last-in, first-out (LIFO) mechanism.
   - ✓ A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
   - ✓ In this type of organization of CPU, all the operations are performed with stack.
   - ✓ The PUSH and POP instruction only need address field. The operation-type instructions do not need address field.
     E.g. PUSH X $\Rightarrow$ TOS←M[X]
     POP X $\Rightarrow$ M[X] ←TOS
     ADD

- ✓ This ADD instruction in the stack organization performs addition of two top of the stack element and stores the result in the top of the stack. First pops two operands from the top of the stack; adds them and stores the result in the top of the stack.

6.2 Instruction Formats

Mode	Op-code	Address

Mode bit: It specifies the way the operand or the effective address is determined.
Op-code field: It specifies the operation to be performed.
Address field: It designates a memory address or a processor register.
→ The length of instruction varies with the variation of number of address in an address field.

There are four types of instruction on the basis of address field they used.

1. Three-Address Instruction
- ✓ Computer with three address instruction can use each address field to specify either processor register or memory operand.
- ✓ Advantage –it minimize the size of program
- ✓ Disadvantage –binary coded instruction requires too many bits to specify three address fields
- ✓ E.g. ADD R1, A, B / R1←M[A]+M[B]

Program to evaluate the following arithmetic statement
X = (A+B) * (C+D) using three address fields 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
```

2. Two-Address Instruction
- ✓ Computer with two address instruction can use each address field to specify either processer register or memory operand
- ✓ Advantage –it minimize the size of instruction
- ✓ Disadvantage –the size of program is relatively larger

Program to evaluate the following arithmetic statement
X = (A+B)*(C+D) using two address field 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. One-Address Instruction
- ✓ Execution of one address field instruction use an implied accumulator register for All data manipulation
- ✓ Advantage –relatively small instruction size
- ✓ Disadvantage –relatively large program size

Program to evaluate the following arithmetic statement
X = (A+B)*(C+D) using one address field instruction

```
LOAD A              / AC←M[A]
ADD B               / AC←AC +M[B]
```

```
STORE T          / M[T]←AC
LOAD C           / AC←M[C]
ADD D            / AC←AC+M[D]
MUL T            / AC←AC*M[T]
STORE X          / M[X]←AC
```

4. Zero-Address Instruction

- ✓ This type of instruction is used in stack organization computer. There is no address field in this type of instruction except PUSH and POP.
- ✓ Advantage –small instruction size
- ✓ Disadvantages –large the program size

Program to evaluate the following arithmetic statement

X = (A+B)*(C+D) using zero address field instruction

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

6.3 Addressing Modes

- → The method of calculating or finding the effective address of the operand in the instruction is called addressing mode.
- → Effective address means the memory address where the required operand is located.
- → The addressing mode specifies the rule interpreting or modifying the address field of the instruction before the operand is actually referenced.

The various addressing modes are:

- **i.** Implied Mode
- **ii.** Immediate Mode
- **iii.** Register Mode
- **iv.** Register Indirect Mode
- **v.** Auto increment or Auto decrement Mode
- **vi.** Direct Address Mode
- **vii.** Indirect Address Mode
- **viii.** Relative Address Mode
- **ix.** Indexed Addressing Mode
- **x.** Base Register Addressing Mode

Implied Mode

- → In this type of addressing mode, operands specified implicitly in the definition of instruction.
- → All the register reference instructions that use an accumulator and zero-address instruction in a stack organized computer are implied mode instruction.

  E.g. CMA (complement accumulator)

Immediate Mode

- → In this addressing mode, the operand is specified in the instruction itself i.e. there is no any address field to represent the operand
- → Immediate mode instructions are useful for initializing register to a constant value.

E.g. LD #NBR / AC←NBR

Register Mode

→ In this type of addressing mode, the operands are in the register which is within the CPU.

E.g. LD R1 / AC←R1

Register Indirect Mode

→ In this addressing mode, the content of register present in the instruction specifies the effective address of operand.

→ The advantage of this addressing mode is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

E.g. LD (R1) / AC←M[R1]

Auto Increment or Auto Decrement Mode

→ In auto increment mode, the content of CPU register is incremented by 1, which gives the effective address of the operand in memory.

E.g. LD (R1)+ / AC←M[R1], R1←R1 + 1

→ In auto decrement mode, the content of CPU register is decremented by 1, which gives the effective address of the operand in memory.

E.g. LD (R1)- / AC←M[R1 - 1]

Direct Address Mode

→ In this addressing mode, the address field of an instruction gives the effective address of operand.

E.g. LD ADR / AC←M[ADR]

Indirect Address Mode

→ In this addressing mode, the address field of the instruction gives the address of effective address.

E.g. LD @ADR / AC←M[M[ADR]]

Relative Address Mode

→ In this addressing mode, the content of program counter is added to the address part of the instruction which gives the effective address of the operand.

→ Assume that the PC is 500 and the address part of instruction is 50. The instruction at location 500 is read from memory during fetch phase and PC is then incremented by 1. Hence, PC is 501, then effective address is 501+50=551.

E.g. LD $ADR / AC←M[PC + ADR]

Indexed Addressing Mode

→ In this addressing mode, the content of index register is added to the address field of the instruction which gives the effective address of operand.

→ This type of addressing mode is useful to access the data array, where the address field of an instruction gives the start address of data array and content of index register gives how far the operand from the start address is.

E.g. LD ADR(X) / AC←M[ADR + XR]

Base Register Addressing Mode

→ In this addressing mode, the content of the base register is added to the address part of the instruction which gives the effective address of the operand.

E.g. LD ADR(B) / AC←M[ADR + BR]

Numerical Example

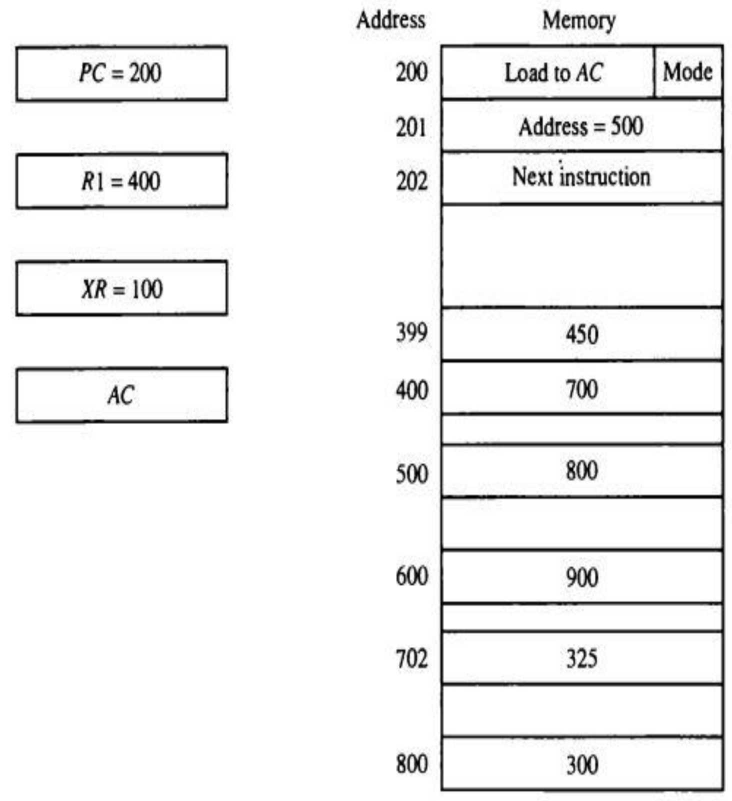
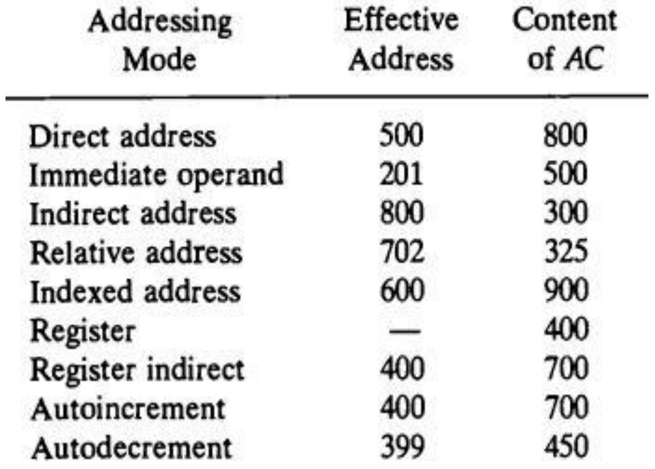


Fig: Numerical example for addressing modes



Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Fig: Content of AC after each addressing modes

6.4 <u>RISC and CISC characteristics</u>

RISC (reduced instruction set computer) characteristics

→ Relatively few instructions
→ Relatively few addressing modes
→ Memory access limited to load and store instructions
→ All operations done within the registers of the CPU
→ Fixed-length, easily decoded instruction format
→ Single-cycle instruction execution
→ The control unit is hardwired rather than micro programmed
→ Relatively large number of registers in the processor unit
→ Efficient instruction pipeline

CISC (complex instruction set computer) characteristics

→ A large number of instructions - typically from 100 to 250 instructions
→ Some instructions that perform specialized tasks and are used infrequently
→ A large variety of addressing modes – typically from 5 to 20 different modes
→ Variable-length instruction format
→ Uses memory to load and store instruction and operand as well
→ Instructions that manipulate operands in memory