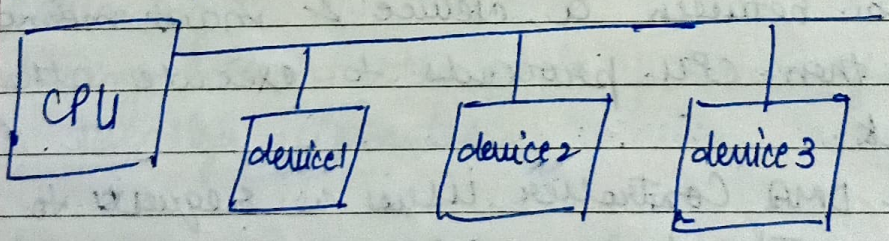
* Priority Interrupt :-

→ There are number of I/O devices attached to the computers.
→ They all are capable of generating the interrupt.
→ When the interrupt is generated from more than one device simultaneously. Priority interrupt system is used to determine which is to be serviced first.
"A priority interrupt is a system that establishes a priority over the various sources to determine which condition (interrupt) is to be serviced first when two or more requests arrive simultaneously."
→ Devices with high speed transfers are given higher priority (eg : Harddisk)
→ And slow devices are given lower priority (eg : keyboard)
→ When multiple devices interrupt CPU at the same time, the device with higher priority served first.



⇒ Establishing Priority of Simultaneous Interrupts 2 ways

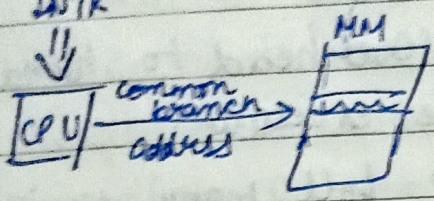
Using Software Method
→ Polling

Using Hardware Method
Serial connection
Parallel connection

Serial connection
↓
Daisy chaining
Priority Interrupt
(Serial Priority interrupt)

Parallel Priority Interrupt

- By Software (Polling)



→ Polling is a software method of establishing priority of simultaneous interrupts.
→ In this method, when processor detects an interrupt, it branches to a common ISR whose job is to poll each I/O device to determine which device caused the interrupt.
→ In this method.
- There is one common branch address for all interrupts.
- Branch address contain program code that polls the interrupt sources in sequence (i.e checks each source whether they have generated the interrupt).
- Highest priority source is test first, if its interrupt signal is ON, control branches to a ISR for this source. Otherwise the next lower priority source is test & so on.

Intrupt Service Routine.

is determined by the order in which devices are polled.

Advantage: (1) Flexible since established by software.

(2) Low cost since it need a very low hardware.

Disadvantage: (1) Very slow

(2) Time Overhead to handle many interrupts can be excessive.
[Time required to poll many devices can exceed the time ~~avail~~ available to service the I/O device.]

○ By Hardware Method:

→ Requires a priority interrupt manager which accepts all the interrupt requests to determine the highest priority request. (Hardware Priority Interrupt Unit)

→ Fast Since identification of highest priority interrupt request is identified by hardware.

→ To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly

→ Thus no polling is required because all decisions are established by the hardware priority interrupt unit.

→ Hardware priority function can be established by either serial or a parallel connect of interrupt lines.

Instruction Formats

Page No.
Date / /

Program: A sequence of instructions
Instruction: It is a command given to the computer to perform a specific operation.

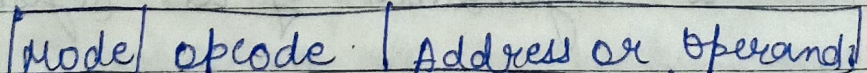
- An instruction is a group of bits (binary code) that instructs the computer to perform a specific operation.
- The purpose of an instruction is to specify both operation to be performed by CPU & the set of operands (or data)

→ Operation include add, sub, mul, shift etc. Operations are performed on data (operands).

→ Operands include the input data & the results that are produced.

What is Instruction Format?

- An instruction format is a binary format which specifies a computer instruction.
- The bit of the instructions are divided into groups called fields.



Instruction format (with mode field)

The most common fields in instruction format are:

1. Opcode (Operation Code): An opcode (operation code) field specifies the operation to be performed such as add, Sub, mul, Shift, Complement etc.

2. Address: An address field specifies a memory address or processor register, where operand is stored.

3. Mode: A mode field specifies the way the operand or the effective address of the operand is determined (or located).

CPU Organization & Types Of Instructions

- Computers may have inst. of several diff. lengths containing varying number of addresses (address field may be 3, 2, 1, 0)
- The no. of address field in inst. format of a computer depends on the internal organization of its registers.

- Most computers fall into one of the three types of CPU organization.

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

Types of inst. formats / Types of address inst.

1. Three address inst formats) Single
2. Two " " " "
3. One " " " " → General
4. Zero " " " " → Short

* Classification of inst based on CPU organization.

(i) Single accumulator Organization:
- All operations are performed on an implied Accumulator (AC) register.
- With one operand implicity in the accumulator register minimizes the internal complexity of the machine & allows for short instructions.
- The inst. format uses only one address field (i.e one-address instr. format.

eg: LOAD X // AC ← M[X]
ADD X // AC ← AC + M[X]
STORE Y // M[Y] ← AC

(ii) General Register Organization:
- It uses set of general purpose register [R0, R1, R2 ...] one of the most widely accepted models for machine architecture today.
- Specifies all operands explicitly.
- The instruction format needs 2 or 3-address fields according to the operation (ie Two or three address instruction format).

eg: ADD R1, R2, R3 // R1 ← R2+R3
ADD R1, R2 // R1 ← R1+R2

ADD R1, X // R1 ← R1 + M[X]

- Each address field may specify a processor register or memory operand.

Stack Organization:

- The operands are put into the stack and the operations are carried out on the top of the stack (TOS). The operands are implicitly specified on TOS.
- It does not use an address field for the instructions like ADD, MUL etc. (i.e zero-address instruction format).
- Push and POP instructions are used to communicate with stack, which require an address field.

eg Push A] Stack ch pona
Push B
Add —— Add kruvata dona da
Pop C] C ch pata./mtlb Stack
cho kd lea.

Here ADD consist of only opcode and no address field. It has the effect of popping the top two numbers from the stack, adding them and pushing the sum onto the stack. Thus all the operands are implied to be in the stack.

Three address inst

Opcode	Destination address	Source address1	Source address2

- 3 address instruction format has three address fields.
- One address field is used for destination & two address fields for source.
- Each address field may specify a processor register or a memory operand.
- It is also called general register organization.

Eg: ADD R1, R2, R3 // R1 ← R2+R3
ADD R1, A, B // R1 ← M[A]+M[B]

- Advantage: Results in short programs when evaluating arithmetic expression.
- DisadVantage: Binary coded inst. require too many bits to specify three address, Complex

Ex: Evaluate C=A+B [using 3 address inst]
→ The contents of memory location A & B are fetched from memory.
→ Transferred into the processor where their sum is computed.
→ The result is then sent back to the memory & stored in location.

ADD C, A, B // M[C] → M[A]+M[B]

* Two address Inst

Page No.
Date / /

opcode | Destination Address | Source address |

→ Two address inst are most common in commercial Computers.
→ It has 2 address fields.
→ Each address fields may specify a processor register or a memory operand

Ex: ADD R1, R2 // R1 ← R1 + R2
MOV R1, R2 // R1 ← R2

* One Address Instruction

Opcode | operand Address (S/D) |

→ It has only one address field. The operand specified in the instruction may be source or destination, depending on instruction.
→ One address instructions use a implied ACCUMULATOR (AC) register for all data manipulation.
o Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.
→ All operations done between accumulator register & memory operand.
→ It is also called Single Accumulator organization.

Eg: ADD X // AC ← AC + M[X]

Ex ⇒ Evaluate C = A + B

Load A	// AC ← M[A]
Add B	AC ← AC + M[B]
Store C	M[C] ← AC.

Load & Store instruction: Used for transfers to and from memory to AC register.

* Zero address instruction

| opcode |

(zero address inst format)

→ In zero address inst stack is used. Also called stack based organization.

→ A stack based computer does not use an address field in the inst. (like add, Mul)

→ Push & Pop inst. however, need an address feild to specify the operand that communicate with stack.

→ To evaluate a arithmetic expression first it is converted to Revere Polish Notation i.e Post fix Notation.

→ The name "zero-address" is given to this type of computer of the absence of an address feild in an inst.

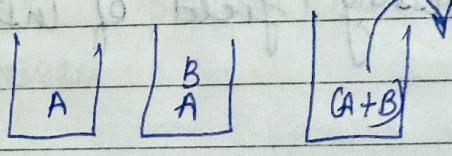
Eg: C = A + B

Push A

Push B

Add

Pop C



Addressing Modes

DOMS | Page No. | Date / /

▷ Address Field: Contains the info. needed to determine the location of the operands & the result of an operation.

▷ Addressing Mode: Specifies how to interpret the info within this address field, how to compute the actual or effective address of the data needed

Definations:

- The different ways of specifying the location of an operand in an inst are called as addressing modes.
- The addressing mode specifies a rule for interpreting or modifying the address field of the inst. before the operand is actually referenced.

Purpose of Using addressing mode:

- To give the programming versatility to the user.
  - Pointers to memory, counters for loop, indexing of data, program relocation etc
- To reduce the number of bits in addressing field of instruction.

Instruction Cycle

- Fetch the inst from memory and PC ← PC + 1
- Decode the inst.
- Execute the inst

Program Counter

- PC keeps track of the inst. in the program stored in memory.
- PC holds the address of the next inst to be executed.
- PC is incremented each time an inst is fetched from memory.

Addressing mode of inst.

- Distinct Binary Code: In some computers the addressing mode of the inst is specified with a distinct binary code, just like the operation code is specified.
- ~~binary~~
- Single Binary Code: other computer use a single binary code that designates both the operation and the mode of inst.

Effective address (EA): EA is the actual address of the operand, where it is actually stored.

Types of addressing modes

COMS | Page No.
Date / /

1. Implied / Implicit Mode
2. Immediate
3. Direct
4. Indirect
5. Register direct
6. Register Indirect
7. Auto Increment / Decrement
8. Relative addressing mode (PC)
9. Indexed (Index Register XR)
10. Base register (BR)

→ No address field is required

3,4 Address specifies memory location

5-7 Address is processor register

8-10 Address field + content of CPU register (These are called displacement addressing)

1. Operands are specified implicitly in the definition of inst.

Ex: CMA: Complement Accumulator
a operand in AC is implied in the definition of the inst.

CLA: Clear accumulator
Operand in AC is implied in the definition of the inst.

Push: Stack Push
Operand is implied to be on top of the Stack (TOS).

- All register reference inst. that use an accumulator are implied-mode inst.

Zero address inst. in a stack organized computer are implied mode instructions since the operands are implied to be on top of the stack

2 Immediate Mode
→ Operand is specified in the instruction itself.
→ An immediate mode instruction has an operand field rather than the address field.
→ Operand field contain the actual operand.
→ Operand = Address.

opcode	operand

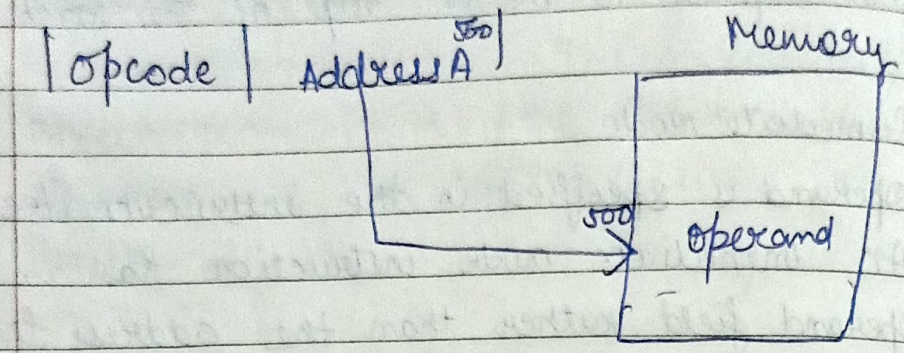
eg: load #7 // AC ← 7
Add #5 // AC ← AC+5

3 ~~Addr~~ Direct Mode:
→ The operand resides in memory & its address is given directly by the address field of inst.
→ Address field of inst contains the effective address EA (or actual address) of operand
→ EA = A (address field)
eg. ADD A // AC ← AC + M[A]
LOAD B // AC ← M[B]
→ It is also called absolute addressing mode.

Advantage:
→ Simple, → Single memory reference to access data, → No additional calculations to work out effective address.

Disadvantage:

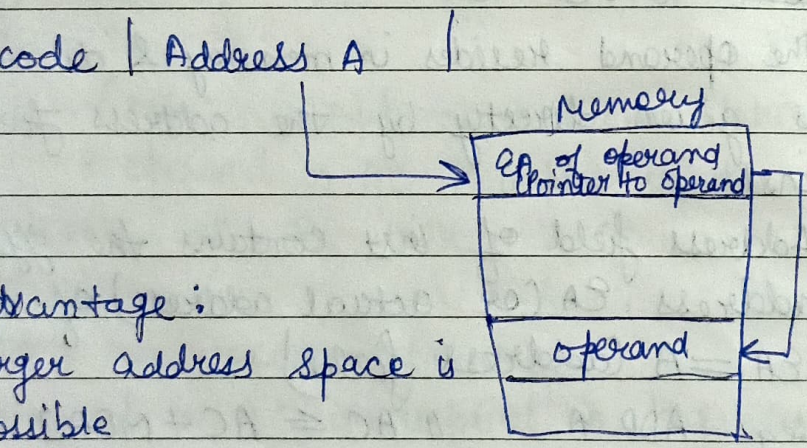
→ Address space is limited to the size of the address field.



4. Indirect Mode.

→ Address field of inst. specifies the address where the effective address of operand is stored in memory.

→ EA = (A) or EA = @A Look in A, find address (A) and look there for operand.



* Advantage:
  - Larger address space is possible
    2^n where n = word length.

* Disadvantage:
  - Multiple (two) memory reference to find the operand.
  - Hence slower.

5. Register Mode :

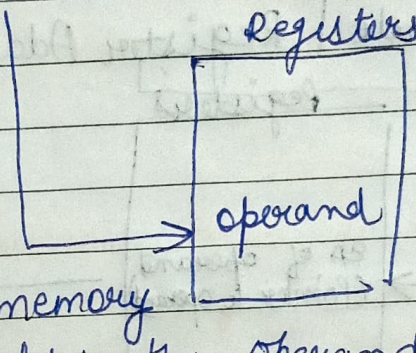
→ Operand is stored in the register. (Register resides within the CPU)

→ Address field of the inst refers to a CPU register that contains the operand

⇒ EA = R_1

→ A k-bit field can specify any one of 2^k register.

opcode | Register address R_0 |



Eg: MOV R_1, R_2

// $R_1 \leftarrow R_2$

No reference to memory is required to fetch the operand.

* Advantages :
- Due to limited no. of registers, very small address field.
- Shorter inst.
- No memory reference (access)
- Very fast execution.

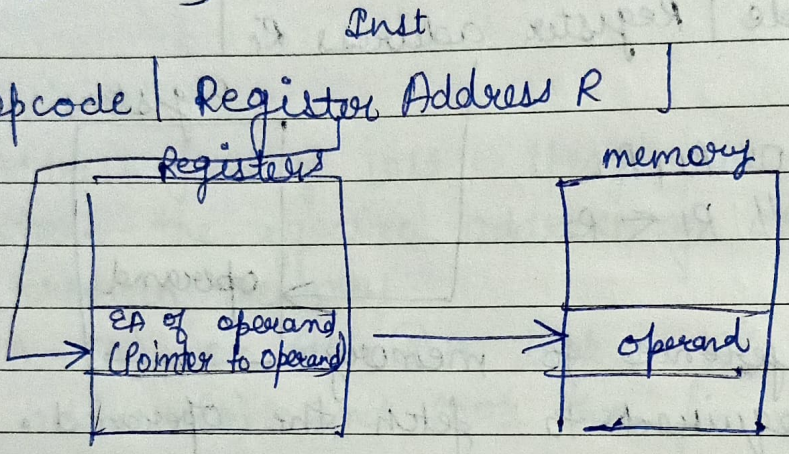
* Disadvantages :
- Very limited address space.
- Using multiple registers helps performance but it complicates the inst.

6. Register Indirect Mode :

→ The address field of inst specifies the register that contains the effective address of operand. The operand is in memory.

→ The register contains the effective address of operand rather than the operand itself.

⇒ EA = (Ri)



Ex: ADD (R1), R2 // M[R1] ← M[R1] + R2

* Advantage:
- Address field of inst uses fewer bits to select a memory address.
- Large address space : 2^n
- One fewer memory access than indirect addressing.

* Disadvantages:
- Extra memory reference

7. Auto increment / decrement mode

→ Similar to register indirect mode (effective address of operand is the content of register specified in the inst) ~~ex~~ except that

→ In Auto increment: The register is incremented after its value is used to access memory.

• EA = (Ri) +

→ In auto decrement: The register is decrement before its value is used to access memory.

• EA = -(Ri)

→ It is used when address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.

Ex: Auto increment

ADD R1, (R2)+ // R1 ← R1 + M[R2], R2 ← R2+1

Ex: Auto decrement

ADD R1, -(R2) // R2 ← R2 - 1, R1 ← R1 + M[R2]

8. Displacement Addressing

→ Few addressing modes require that the address field of inst be added to the content of a specific register in the CPU.

Effective address = Address part of inst + Content of CPU register (PC/XR/R)

→ EA = A + (R)

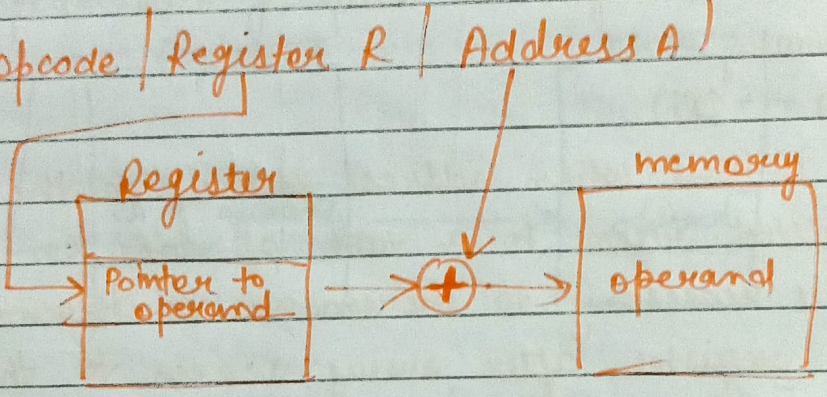
→ Address field hold two values
→ A = base value
→ R = Register that holds displacement / offset

or Vice Versa.

→ R = Base value
→ A = Register that holds displacement/offset

* Advantage: Flexibility

* Disadvantage: Complexity



3 versions of displacement addressing:

1. Relative addressing (EA = PC + A)
2. Indexed " (EA = A + XR)
3. Base Register " (EA = Ri + A)

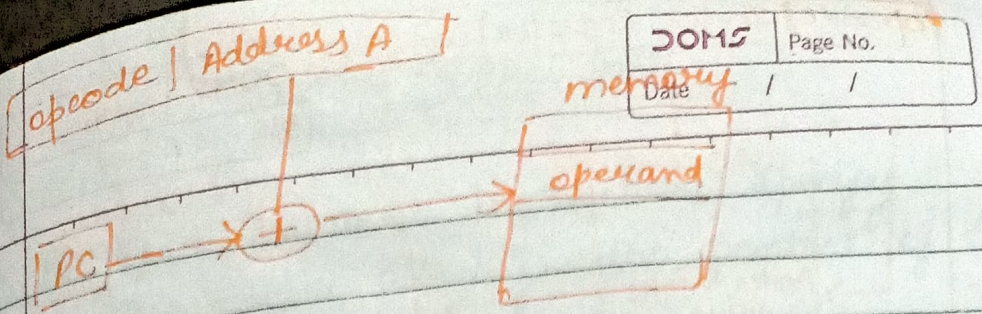
8. Relative Addressing Mode

→ Register = Program Counter (PC)
→ The content of PC is added to the address part of the inst to obtain the effective address of operand.

Effective adress = content of PC + address part of inst.

EA = (PC) + A)

Ex: Load $A or Load AC(PC) // AC ← M[PC+A]



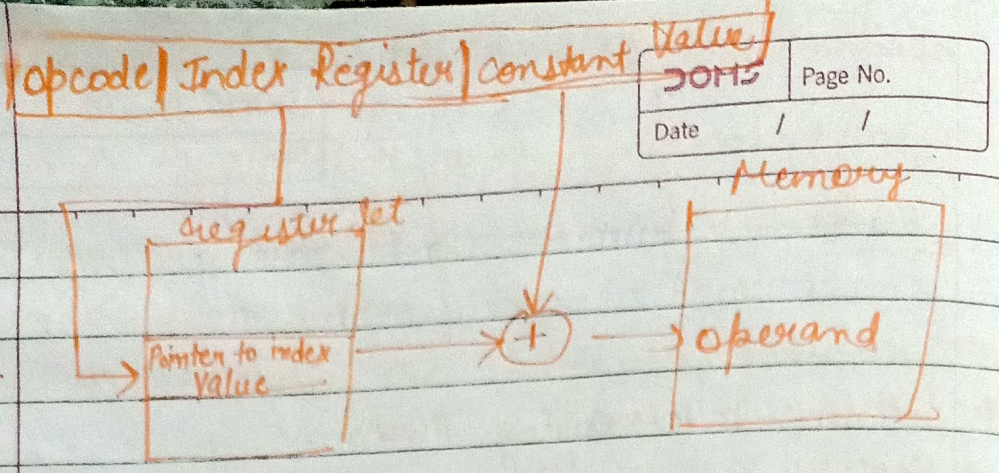
9. Indexed addressing Mode
→ Register = Index Register
→ The content of index register is added to the address part of the inst to obtain the effective address of operand

Effective address = Content of XR + Address part of inst.
EA = A + (XR) or EA = A + (Ri)
→ A contains the Constant value in inst.
→ XR/Ri is the name of register involved holds index value.

→ Index register used here is either a special CPU register (XR) provided for this purpose or commonly, it may be anyone of a set of general purpose registers (Ri) in the processor; that contain an index value.

Ex: ADD R1, 20(R2) // R1 ← R1 + M[20+R2]
ADD R1, 100(XR) // R1 ← R1 + M[100+XR]

- Index register holds a index no. that is relative to address part of the inst.
- Good for accessing arrays



10 Base Register Addressing Mode

→ Similar to indexed addressing except that the register is now called a base register.

→ Register = Base register

→ The content of base register is added to the address part of the inst. to obtain the effective address of operand.

Effective address = content of BR + Address part of inst.

EA = (Ri) + A

Ri is base register that hold base address

A holds a displacement relative to this base address.

→ Same diagram for Constant Value di fga displacement a ju.

Ex: Load A(R1) // R1 ← M[R1+A]

Add R1, 20(R2) // R1 ← R1 + M[R2+20]

- Useful in Computers to facilitate the relocation of programs in memory.