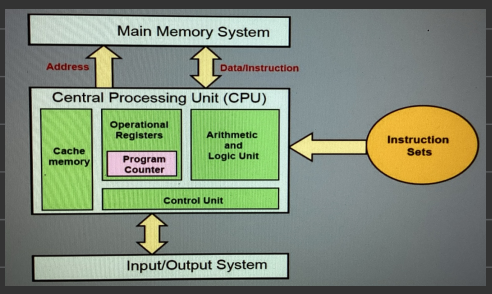
Sumedh

- Topics

1. Computer units
2. Computer instructions
3. Bus Structures
4. Memory location & address
5. Microprocessors
6. 8086
7. Instruction set n sequencing
8. Instruction format
9. Addressing mode n types

- Functional Units of a Computer

1. Input Unit
2. Output Unit
3. CPU (ALU & control unit)
4. Memory
5. Bus Stoucture



- ALU (Arithmetic logical unit)

1. Arithmetic & logical operation are carried out.
2. Operands are brought in ALU from the memory
3. After performing operations it sets flags of the resulting answer

- Output Unit

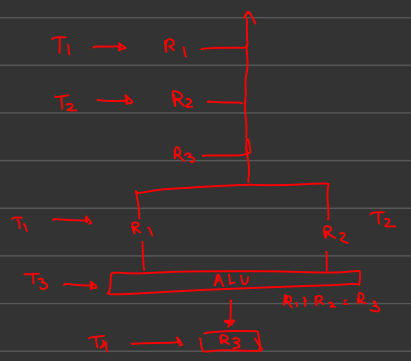
1. Interface with output device
2. Convert info into binary form in output devic

Processor
Memory
Output unit

- Control Unit

1. Accepts information from user (Input unit)
2. Stores the info (Memory)
3. Process the info (ALU)
4. Provide processed results (Output Unit)
5. Generate timing signals when a operation takes place

- It works with a reference signal called processor clock
- Clock divides operations in basic steps with one clck cycle



- CPU

1. The brain of the machine
2. Contains ALU, CU, Registors

Here, ALU - Perfoms task
CU - Control the task & give signals
Reg - Stores result

- Instructions of the computer

Add R0, LOCA

It takes a number frm a mmry location called 'LOCA'
It adds the number in reg R0
The result is stored in R0 replacing og value
The no. in loca stays same

- LOAD R2 , LOC

It tells comp to take data stored in mmry location 'LOC' and puts in reg R2

Steps

1. The comp first reads the load R2, LOC from mmry to processor
2. Operation to be performed is determined by CU
3. The processor fetches the data frm 'LOC'
4. Now the data fetched from LOC is placed in R2 replacing the prev data

- Add R4, R2, R3

Add the value in R2 & R3
Stores the sum in R4
R2 & R3 remain unchanged, R4 is overwritten

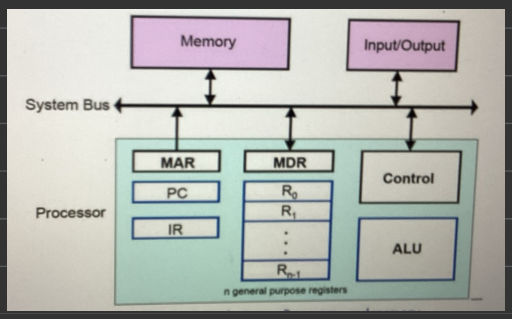
- Store R4, LOC

Saves the value frm R4 to mmry loc LOC
The old value replaced with value frm R4
The value in R4 remains unchanged

- Registors

They are fast storage spots in CPU for temporary data

- MAR (Memory Address Reg) : Holds the address of where data shld be read / written
- MDR (Memory data reg): Holds the actual data being transferred to / frm memory
- IR : Holds the instruction currently being executed
- Prog Counter : Keeps track of the next instruction to fetch frm mmry
- Gen purpose Reg : Used to hold data and results during processing



- **Instruction fetch**

The program is loaded into mmry from an input device
The PC is set to first instruction
The PC value is sent to MAR and read cmd is send to mmry
The first intruction is read frm mmry & loadid in MDR
The instruction in MDR is moved to IR for decoding

- **Instruction Execution**

1. **Operation Check** : The ALU checks instruct^{n} in IR to figure what operation to perform

2. **Fetching Operands** : From memory : The add sent to MAR, read into MDR then sent to ALU
   From regs : If operand is gen purpose, it's used directly

3. **Perform Operation** : ALU performs operation and send result to GP or memory

4. **Storing Result** : The result is sent to MDR which has the add for storing in MAR, The pc points nexts ins to fetch new cycle

- **Bus Structures**

The CPU and memory are connected by 3 type of buses

1. data bus : transfers data b/w CPU & memory (8-bit bidir)
2. Address bus : carries the add of memry loc to be accessed (16-bit uni)
3. Control bus : Sends control signals to manage the ops of CPU (bidir)

- **Single bus structure**

A single set of wires (bus) used for comm b/w CPU & other devices

eg : This bus handles both sending data to peripherals and receiving data frm them

- Drawbacks

1. speed differences : devices connected to the bus have diff speeds eg (keyboards are slow, disks are fast)

2. efficiency issues : the single bus can't handle these speed diffs

3. Solutions : buffers - temp storage to manage data during transfers
   two-bus struc - use seperate buses for instructions

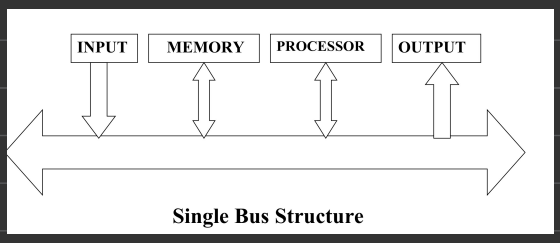
- Two-bus Structure

Uses one bus for instruction fetches and another for data fetches
It has faster operation becoz it seperates instruction & data traffic

- Multi-bus structure

uses multiple buses to further improve performana

- **Address Bus:** Sends address information from the CPU to memory or peripherals (unidirectional).
- **Data Bus:** Transfers data both ways between the CPU and peripherals (bidirectional).
- **Control Bus:** Sends control signals both ways to manage operations (bidirectional).



Single Bus Structure

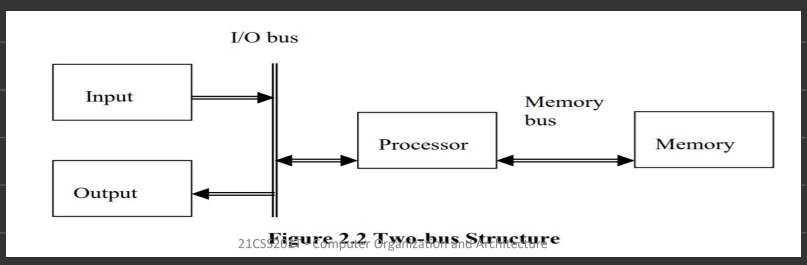
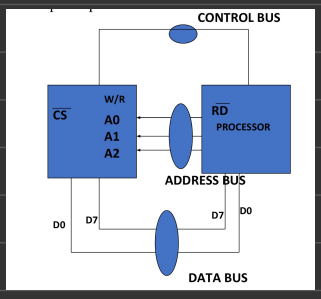
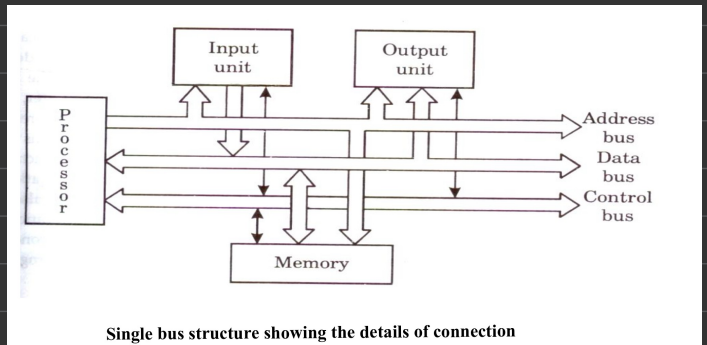


Figure 2.2 Two-bus Structure





Single bus structure showing the details of connection

- Memory locations and addresses

  - Memory hierarchy

    It aims to provide the fastest access to data while keeping the cost low, it's done by organizing mmry in layers with the fastest and most expensive types at the top

  - Memory loc & address

- mmry is made up of many storage calls, each capable of storing 1 bit of data
- data is usually accessed in grp of bits. The grp is called 'word'
- Memory layout: The mmry can be visualized as a collection of words lined up, each with a uniqe pos
- Main mmry is a critical part of the comp where data and progms are stored
- Uniqe addresses: each storage loc in the main memory has a uniqe identifier known as address
- words & bytes: data is transferred blw CPU & memory in "words", a word can be 8, 16, 32, 64 bits in size

If word is 8 bits, its called a byte

  - Address space

- Address space: to access data in mmry, each word must have a uniqe identifier, called an address, to total number of uniqe addresses available is "add space"

  eg 32Mb needs 25 bits i.e 2^{25} bytes

  - Memory Operations

- Progrm execution - comps use progrms to process data, where both the program instructions and data stored in mmry.

basic ops

i) Load (Read / fetch): The CPU reads data frm a specific mem loc
ii) Store (Write): The CPU writes data to a specific mem loc

- **Byte address assignment**

  - little - Endian eg : For a 32-bit data (eg 46, 78, 96, 54) the least significant byte (46) is stored at lowest add

  - Big- endian : for a 32-bit data (eg 46, 78, 98, 54) , the most sig byte (54) is stored at lowerst add

- **Aligned v/s Unaligned words**

  ↓

  In a 16-bit sys , aligned words start at adds like 0, 2, 4, etc
  32 0, 4, 8, etc
  67-bit 0, 8, 16, etc

  Unaligned : Words can start at any add

- **Microprocessors**

  - **Microprocessors & Microcontrollers**

    ↓

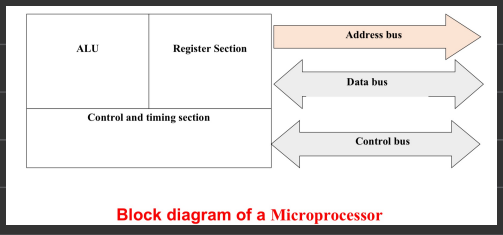
    The brain of the comp where all calc are carried out .
    It's a CPU on a single chip

    Microcontroller are more integrated chip that includes not just the CPU but also mmry and I/O controls
    eg AVR mirco con

  - **Internal Struc of a microprocessor**

  - **Microp tasks**

1. Data transfer : Moving data b/w processor & mmry or other I/O systems
2. Arith & logic ops : performing calcs & making decisions
3. progrm flow : deciding the next steps in a prog based on condition



Block diagram of a Microprocessor

- Components

1. ALU : performs Calcs
2. Registors : store temporary data
3. Control n timing unit : manages timing of ops & control signals
4. Add bus : carries adds to access mmry
5. Data bus : carries data to / from mmry
6. Control bus : Send control signals to mange ops

- Types of microprocessors

- Categories :-

1. Word Size : refers to amt of data the processor can hangle (eg 8-bit , 16, 32)

2. Instruction set : i) RISC (Reduced Instruc set comp)
   Simplied set of instruct^ns
   ii) CISC (Complex " .. ..)
   Complex set

3. Function : can be general-purpose or specialized

- Evolution of microprocessors

1st Gen
- 4-bit
- 1971
- Intel 4004
- Perform basic Arit & log ops

2nd Gen
- 8-bit
- 1973
- Could handle 8-bit data

3rd Gen
- 16-bit
- 1978
- Intel 8086
- Capable of handling more complx tsks

4th Gen

- 32-bit
- Intel 80386

5th Gen

- 1993
- 64 bit
- Pentium processors
- Allows multiple microp to work together

- **Typical Microprocessors**

1. 68K : Motorola's processor
2. X-86 : Intel's series of p
3. IA-64 : Intel's Itanium architecture
4. MIPS : microp w/o interlocked pipeline stages
5. ARM : Adv RISC machine
6. Power PC : Used by Apple, IBM & motorola
7. Atmel AVR : microp used in various apps

- **8086 microprocessor**

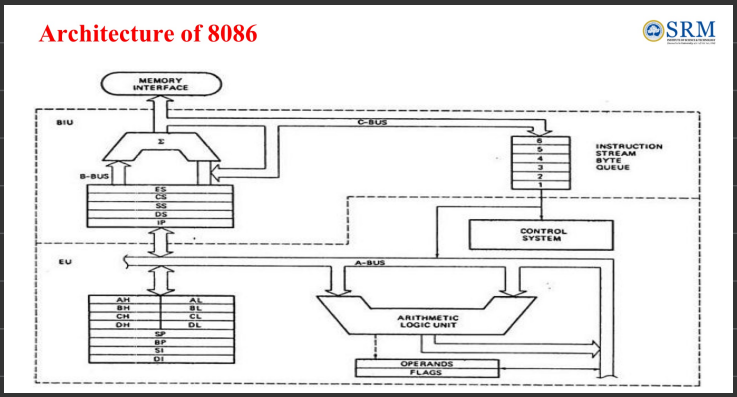
Designed by intel in 1976

16-bit pro. with :-
- 20 add lines (1mb of mmry)
- 16 data lins
- Powerful set of instruc like (+, -, x, ÷)
- Two modes
  - Max^m : for system with multiple pro
  - Min^m : for system with single pru

- **Features of 8086**

- Can store upto 6 instruc bytes
- Handles 16-bit ops & data
- pipelining : improves performance by fetching & executing instruc in stages

Fetch Stage : pre-fetches 6byte of instru
Execute " : exccuts these instru.



- Memory Segments

1. Code segment : Stores the progrm
2. data seg : Stores the data
3. extra seg : used for string ops
4. Stack seg : used for stack ops

- General purpose regs

1. AX (Accumulator)
   - 16 bit reg
   - 2 8-bit parts AH & AL
   - performs arith & log ops
   - performs 8-bit instruc

2. BX (Base register)
   - 16-bit reg
   - 2 8-bit parts BH & BL
   - stores offsets & adds
   - performs 8-bit instruc

3. CX (Counter Register)
   - 16 bit reg
   - 2 8-bit parts CH & CL
   - used for counting loops
   - performs 8-bit instruc

4 DX (Data
   - 16-bit reg
   - 2 8-bit parts DH & DL
   - used for I/O ops & multiplicatior
   - performs 8-bit instruc

· Pointers and Index registers

1. SP (Stack pointr)
   - points to top of the stack
   - 16 bit reg
   - When stack is empty, the pointr will be (FFFE)H
   - offset add relative to stack seg

2. BP (Base pointer)
   - used to access parametrs passed by stack
   - 16-bit reg
   - offset add relative to stack seg

3. SI (Source Indux)
   - used as a pointer for source data in ops
   - as a source in some string related ops
   - 16-bit reg
   - offset rel. to data seg

4. DI (Destination Indux)
   - used as a pointer for addressing of data
   - 16-bit register
   - offset rel to eactra Seg

5. IP (Instruction Pointer)
   - 16-bit reg
   - Stores the add of nxt instruc
   - offset rel to Code seg (CS) reg

· Segment Registers

1. CS (Code Segment Reg)
   - user cannot modify the content of thuse regs
   - only the micvp compiler can do this

2. DS (Data seg Reg)
   - user can modify the content of data seg

3. SS (Stack seg Reg)
   - used to store info abt mmry seg
   - ops are push/pop

4. ES (Extra Seg Reg)
   - The control of the compiler remains the DS
   - used for copying purpose

- **Flag Status Reg**
  - 16-bit reg
  - contain 9 flag
  - 7-bits free
  - if flag val is 1, the flag is set, if 0 it resets
  - flags tell status of the processor of any op

- **Instruction set n sequencing (Assembly lang Starts ....)**
  ↓
  a cmd given to comp to perform a specific task

sequencing : The order in which instruc are carried out

- **Types of ops**

1. data transfer : moving data b/w mmry & processor
2. A & L ops : calcs done by proc : managing order of tsks
3. prog sequencing & control :
4. I/O transfer : moving data b/w proc & external devices

- **Register transfer Notation [RTN]**

describes how data moves b/w regs & mmry

LOC, PLACE, MEM : Memory locs
R_1, R_2 : processor registers
DATA_IN, DATA_OUT : reg I/O data
[] : Show's what's inside
RHS : The source of data to be moved
LHS : Where the data gets

eg $R_2 \leftarrow [LOCN]$ Moves data frm LOCN to R_2
$R_4 \leftarrow [R_3] + [R_4]$ result of $R_3 + R_2$ stored in R_4

- **Data Transfer Instruction**

These cmds move data b/w regs, mem & I/O devices

MOV : Copy data frm one place to another
LDA : load data into accu
STA : Store data frm accu to mmry
PUSH : Save data frm a reg to stack
POP : pop off stack to reg pair

- **Data Manipulation Instruc**

  - i) Arithmetic inst
  - ii) logical & bit manipulation
  - iii) Shift Instruc

i) Arithmetic Instruc

INC : Increment data byl
DEC : decrement "
ADD : add two val
ADC : add with carry bit
MUL : multiply 2 val

ii) logical & bit manipulation

AND : perform bitwise AND op
OR : "
NOT : Inverse each bit
XOR : exclusiv OR over every bit

iii) Shift Instruc

SHR : Shifts bit to right & put 0 in MSB
ROL : rotate bit to left, i.e MSB to LSB & to CF
RCL : rota " ", i.e MSB to CF & CF to LS

- Instruction formats

Types : i) Operation field : specifies op to be performed
ii) Add field : Contains location of operand
iii) Mode field : specifies how to find the operand

Types based on number / length of add / instruc

1. Zero add instruc
2. One
3. Two
4. Three

- Zero add instruc

Used in stack-based computers where add not mentioned operands are stored in push down stack

Operations :-

MOVE D, Ri : Move data frm mem loc D to reg Ri
ADD E, Ri : Add data frm mem loc E to reg Ri
MOVE Ri, F : Move result frm Ri to mem loc F

ALU ops

LOAD D, Ri : Load data D to Ri
LOAD E, Rj : E to Rj
ADD Ri, Rj : Add
MOVE Rj, F : Move result frm Rj to F

- One add instruc

Uses implied accumulator reg for data manipulatn
one operand in accu & other in reg / mem loc

- Two add instruc

Common in commercial comps
Allows two adds in one instruc

- Three add

Allows 3 adds for more complex ops.

(A+B)* (C+D) Using 0, 1, 2, 3 address Instructions

1. Zero address

```
PUSH A   : Push A on the stack
PUSH B   :   "
ADD      : add A+B , result on top of stack
PUSH C   :      "
PUSH D   :      "
ADD      :   "
MUL      : Multiply results
POP X    : pop result to mem log X
```

2. One address

```
LOAD A   : load A in accumulatr
ADD B    : add B to    "
STORE T  : Store res in mem loc T
LOAD C   :    "
ADD D    :    "
MUL T    : Multiply accum. with T
STORE X  : Store in X
```

3. Two address

```
MOV R1, A   : Move A to R1
ADD R1, B   : Add B to R1
MOV R2, C   :      "
ADD R2, D   :      "
MUL R1, R2  : Muliply R2 to R1
MOV X, R1   : Move res of R1 to X
```

4. Three Address

```
ADD R1, A, B   : R1 = A+B
ADD R2, A, B   : R2 = C+D
MUL X, R1, R2  : X = R1 * R2
```

- **Addressing Modes**

Diff methods to specify where the data is located in an instruction

They provide flexibility
Reduce the number of bits

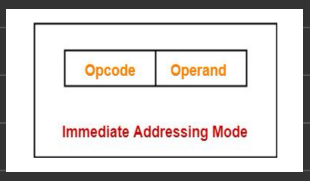
Types

1. Immediate addressing
2. direct
3. Indirect
4. Register
5. Reg indirect
6. Relative
7. Indexed
8. Auto Increment
9. Auto decrement

1. Immediate addressing

Description : The operand is directly given in instruction

Eg : ADD 5
MOV AL 25H

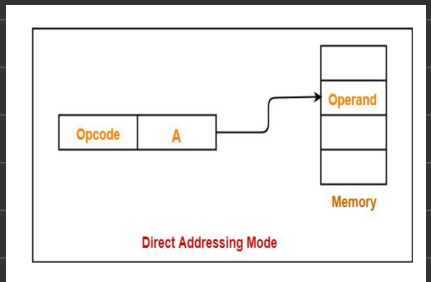


Features : No need to fetch data from memory
Fast but has a limited range

2. Direct

Des : The add fld contains add of data
Effective add (EA) = Address fld (A)

Eg : ADD A
└ address, not a operand

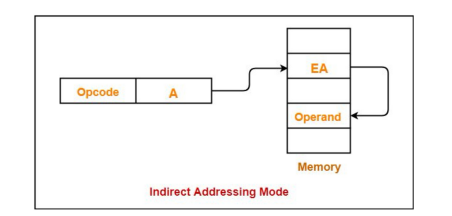


Features : Single mem to acces data
Imitd add spac
No extra cale

memory loc → add → operand

3. Indirect

Des : The add fld points to mmry that contains add of operand
Eff add = [A]



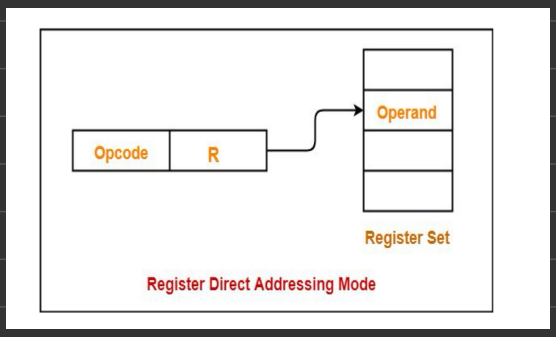
Eg : ADD (A)

Features : Requires 2 mmr loc to fetch operand

4. Register direct

Des : Operand stored in CPU reg

Eg : ADD R
AC ← AC + [R]



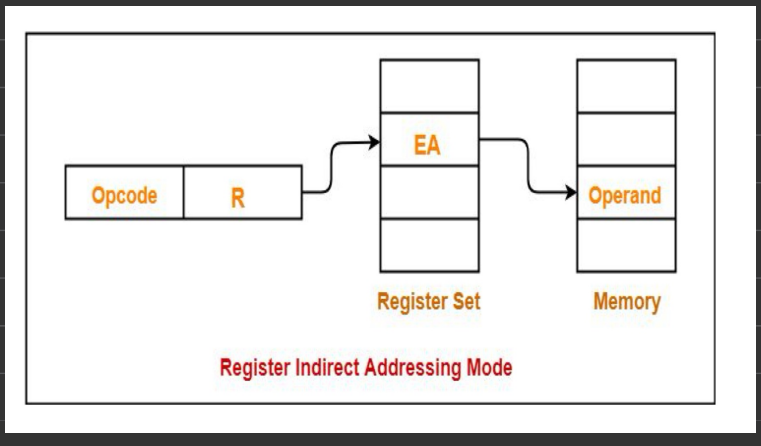
Features : No memry acces needed
fast execution
Short instrue
limitd add space

5. Reg Indirect

Des : The add field points to CPU reg that contains add of operand

Eg : ADD R
AC ← AC + [[R]]

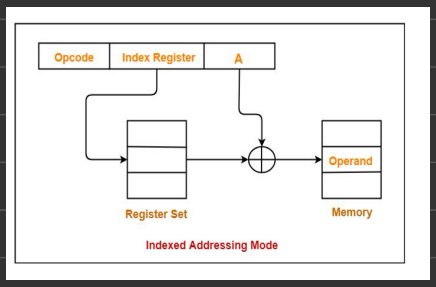
Features : Only 1 mmry loc needed to fetch



6. Indexed

Des : The effective add of operand obtained by adding contents to indx reg to add part of instruc

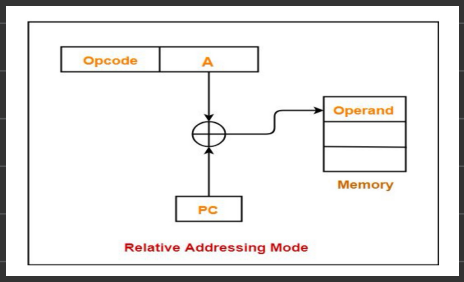
Effective add = Content of indx reg + add part of instruc



7. Relative

Des: A type of displacement addressing where EA = ...

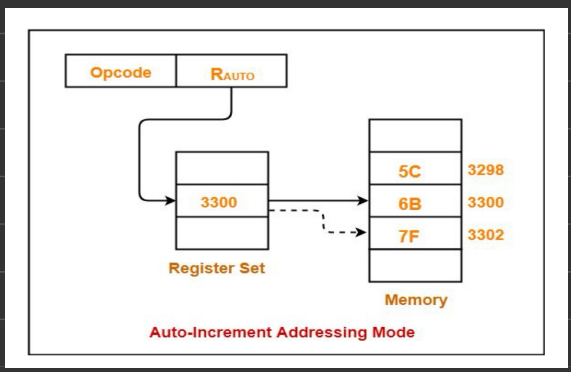
Effective add = Content of progm counter + Add part of instruc



8. Auto Increment

Des : Special case of Reg Indirect mode
EA = Content of reg

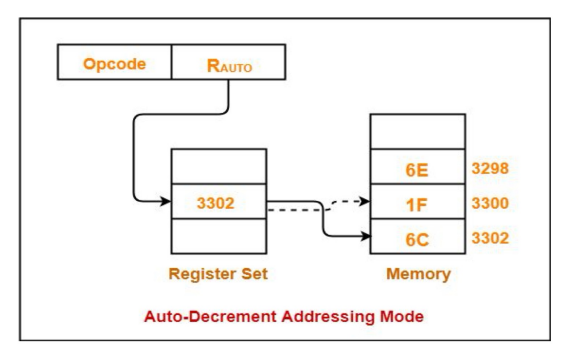
Features : After access, the content of reg increments auto by size 'd'
'd' depends on size of operand
Only 1 mmry loc needed



9. Auto decrement

Des : Special case of Reg Indirect mode
EA = Content of Reg - Step size

Features : first, the Content of reg decrement by size 'd'
'd' depends on size of operand
Then operand is read
Only one mmry needed



Reg In → Point to X
Auto Inc X+1
X-1