DATE / /
PAGE NO.

UNIT - 2

- Memory location and Addresses.

Unit : 0/1 - Bit

4 bits - Nibble

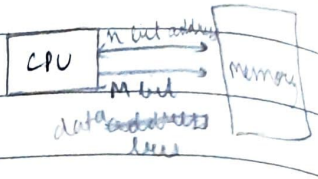
8 bits - Byte

16 bit or more - Word

2^{10} - 1 K

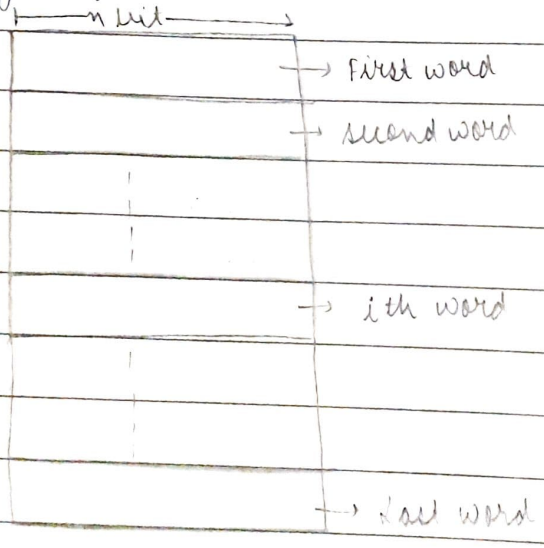
2^{20} - 1 M

2^{30} - 1 G

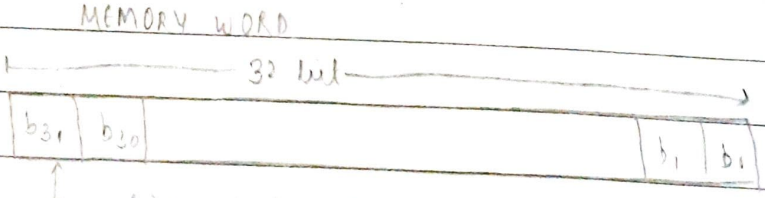


2^n Memory location can be address. The range of the addresses is from 0 to $2^n - 1$.

e.g. if 16 bit bus is there then $2^{16} = 2^{10} \times 2^6 = 64K$ location can be accessed and will be in the range of 0 to $(2^{16} - 1)$.



Eg-



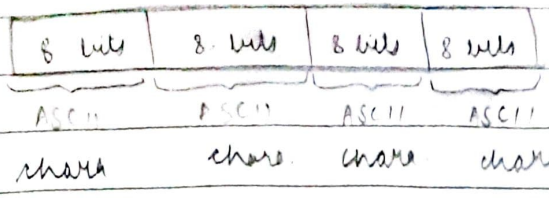
Sign bit b_{31} - 0 for +ve nos

1 for -ve nos

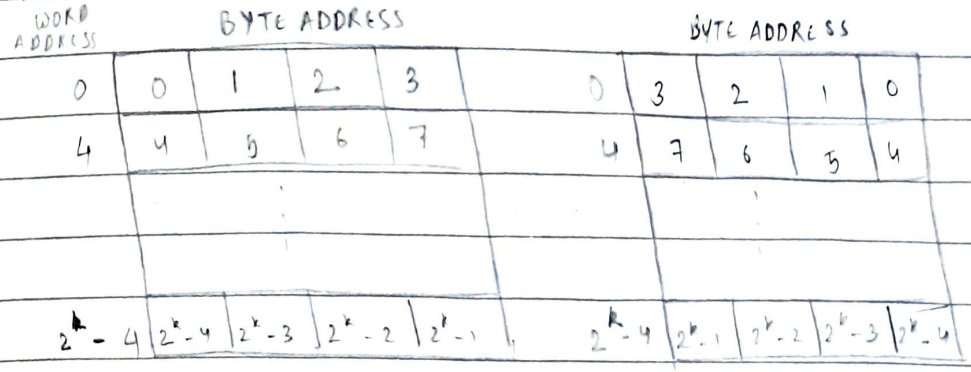
(1) Signed Integer

Date / /
Page No.

1b) Four characters



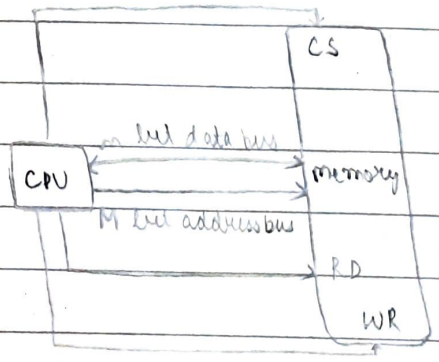
- BIG ENDIAN AND LITTLE-ENDIAN ASSIGNMENT.



1a) Big-endian assignment 1b) Little-endian assignment

BYTE AND WORD ADDRESSING

- MEMORY OPERATION
- Read Operation
- Write Operation



- INSTRUCTION AND INSTRUCTION SEQUENCING

A computer must have instructions capable of performing 4 types of operation:

1. Data transfers b/w the memory and the processor register
2. Arithmetic and logic operations on data
3. Program sequencing and control
4. I/O transfers

Notations: ⇒ Register Transfer Notation
e.g. LOAD R2, LOC
e.g. R4 ← [R2] + [R3]
⇒ Assembly Language Notation
e.g. LOAD R2, LOC;
ADD R4, R2

- CISC AND RISC

- Complex Instruction set computer (CISC)
A complex instruction set computer (CISC), is a computer where single instruction can execute several low-level operation (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instruction, as its name suggest "COMPLEX INSTRUCTION SET"
- Eg- of CISC instruction set architectures are PDP-11, VAX, Motorola 68K, and your desktop PCs on intel's x86 architecture based too.

- Reduced Instruction Set Computer (RISC)
- A RISC is a computer which only use simply instruc that can be divide into multiple instructions which perform low-level operation within single clock cycle, as its name suggest "REDUCED INSTRUCTION SE
- Ex of RISC families include DEC Alpha, AMD 29K, ARC Atmel AVR, Blackfin, Intel i860 and i960, MIPS, Motorola 88000, PA-RISC, Power (including PowerPC), SuperH, SPARC and ARM too

STRAIGHT LINE SEQUENCING

- A program of $C \leftarrow [A] + [B]$

Address	contents	
→ i	Load R2, A	4-instruction program segment
i+4	Load R3, B	
i+8	Add R4, R2, R3	
i+12	store R4, C	
A		Data for the program
B		
C		

Begin execution here

- A program of adding n nos

i	Load R2, NUM1
i+4	Load R3, NUM2
i+8	Add R2, R2, R3
i+12	Load R3, NUM3
i+16	Add R2, R2, R3
	⋮
	⋮
i+8n-12	Load R3, NUMn
i+8n-8	Add R2, R2, R3
i+8n-4	store R2, Sum.
	⋮
	⋮
SUM	
NUM1	
NUM2	
	⋮
	⋮
NUMn	

- TYPES OF INSTRUCTION OF THE BASE OF ADDRESS USED
  - 3 Address Instruction
  - 2 Address Instruction
  - 1 Address Instruction
  - 0 Address Instruction

- Three Address Instructions (Opcode, destination and source)

- Program to evaluate $X = (A+B)*(C+D)$
- 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)
- Result in short programs
- Instruction becomes long (many bits)

- Two - Address Instructions.

- Program to evaluate $X = (A+B)*(C+D)$
- 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)

- One - Address Instructions

Use an implied AC register for all data manipulation

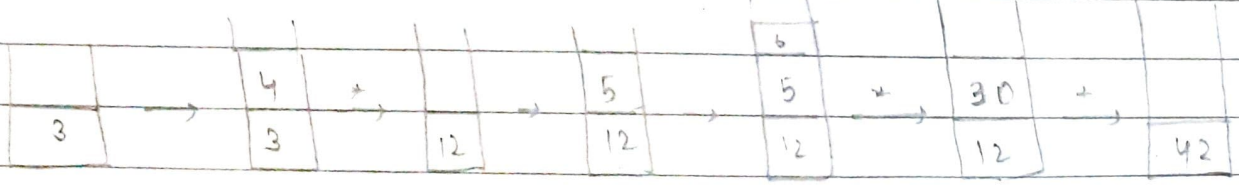
- Program to evaluate $X = (A+B)*(C+D)$
- LOAD A (AC M[A])
  ADD B (AC AC + M[B])
  STORE T (M[T] AC)
  LOAD C (AC M[C])
  ADD D (AC AC + M[D])
  MUL T (AC AC * M[T])
  STORE X (M[X] AC)

Date / /
Page No.

- Zero - Address Instructions.
- Program to 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)

- REVERSE POLISH NOTATION
- Arithmetic Expressions : A+B
- A+B Infix notation
- +AB Prefix or Polish notation
- AB+ Postfix or reverse polish notation
- The reverse Polish notation is very suitable for stack manipulation
- e.g. - A*B + C*D = AB* CD* +

Evaluation of Arithmetic expressions.
- Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation
- (3*4) + (5*6) : 34* 56* + = 42



- Given 0, 1, 2 and 3 address programs for given eqn

1 - $A*B+C*D+E*F \rightarrow AB*\ CD*\ EF*++$

2 - $A*B+A*(B*D+C*E) \rightarrow AB*\ ABD**\ CE*+*+$

3 - $[A*[B+C*(B+E)]]/[F*(G+H)] \rightarrow ABCDE+*+*FGH+*/$

4. $(A+B*C)/D-E*F+G*H) \rightarrow ABC*+DEFGH-*+*/$

5 $A+B*[C*D+E*(F+G)]$

6. $(A+B-C)/(D*(5*F-G))$

7 $(A-B+C*(D*E-F))/(G+H*K)$

- ADDRESSING MODES

TYPES OF ADDRESSING MODES.

- Implicit
- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement
  - Relative
  - Base Register
  - Indexing
- Stack

- IMPLICIT

INCR increment DCR

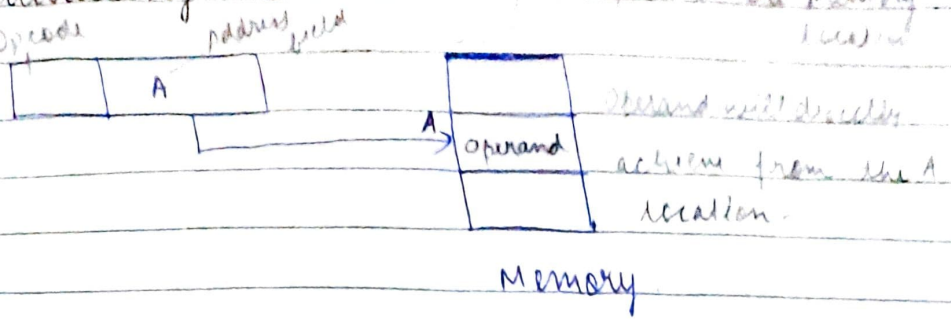
COMP complement

- IMMEDIATE the data will be directly transferred to the register

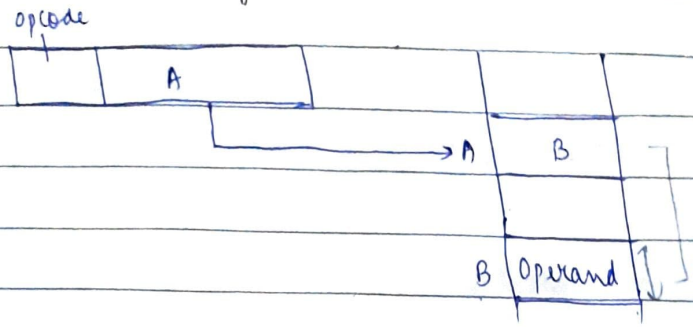
MOV A, #32H

ADD A, #22H

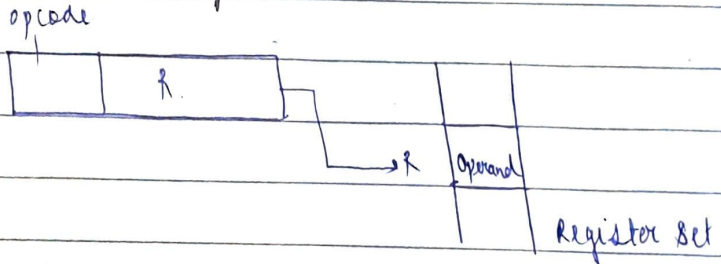
- Direct Addressing Mode



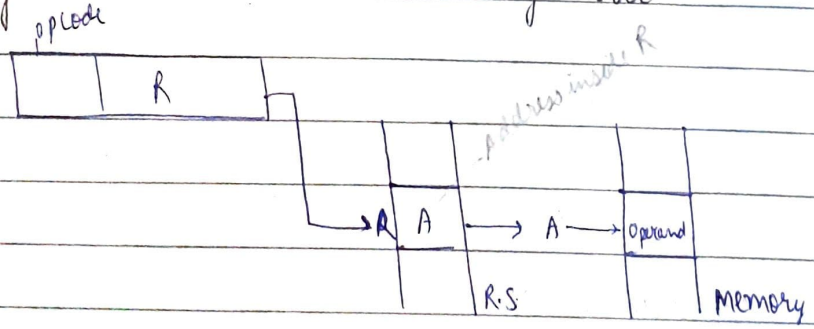
- Indirect Addressing Mode



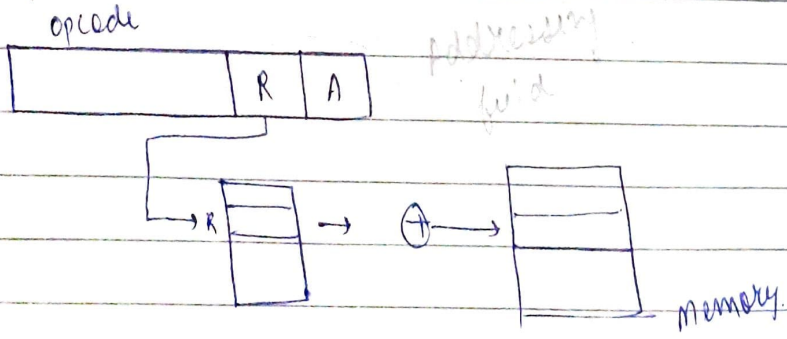
- Register Addressing Mode



- Register indirect Addressing mode.



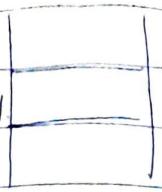
- Displacement



- Relative

MOV A, [22][4]

Base Relative address 24



- Base Register

MOV A, [ ] [ ]

R → Index Register

- Indexing

A → starting base address

R → index

Opcode	Mode	Operand

Mode - The way the operands are chosen during program execution is dependent on the addressing mode of the instruction

An Addressing mode specifies how to calculate the effective memory address of an operand by using information held in Register and for constants contained with an instruction.

The Addressing mode specifies a rule for interpreting or modifying address field of the instruction before the operand is actually referred

1. Implied Addressing Mode -

In this mode the operands are specified implicitly in the definition of the instruction

eg - complement Accumulator.

Here the operand in the accumulator register

- All register reference instruction that use an accumulator are implied mode instruction
- Zero address instruction in a stack organised computer are implied mode instruction since the operands are a implied to be on the top of the stack

2) Immediate address mode -

This mode has the operand field rather than an address field

LOAD R1, #50

Advantage - ① Fastest Execution
② No memory reference to fetch data

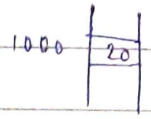
Disadvantage - ① less flexible
② small Range

3) Direct Addressing mode -

In this mode operands resides in memory & its address is given directly by the address field of instruction

Eg - LOAD R1, [1000]

R1 ← M[1000]



R1 [20]

Adv → ① Single Memory Reference to access data
② No additional calculation to calculate effective address

4) Indirect Addressing Mode

In this mode, the address field of the instruction given the address where the effective address is stored (address of the operand)

- Control fetches the instruction from memory and

uses its address part to access memory again to read the effective address

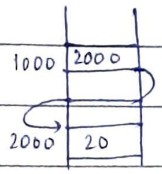
Indirect A.M

Memory Indirect A.M

eg Load R1, [1000]

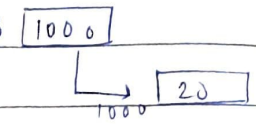
Load R1, @(1000)

( ) or @ - Indirect addressing mode



Register Indirect A.M

eg- Load R1, (B)



(5) Register Addressing Mode or Register Direct mode

- In this mode Operands are in the registers, that reside within the CPU

E.A = R

eg ADD A, B A ← A + B

Adv - ① No memory access; hence very fast execution

② very small address field needed. This makes the instruction shorter which will make faster instruction fetch

Disadv - ① Limited no. of registers

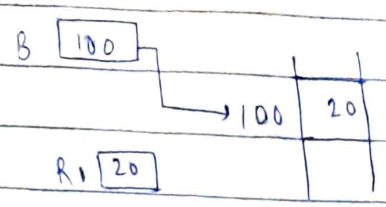
② Requires good assembly programming

(6) Register indirect addressing mode.

In this mode, the instruction specifies as register whose content give the address of the operand in the memory

Here the instruction uses fewer bits to select a register than would have been required to specify a memory address directly

eg - Load R1, (B)



Adv - ① Less no of bits are required to specify the register

② One fewer, Memory access the indirect addressing

⑦ Auto increment / Auto decrement addressing mode -

```
MOV   R1, #22
INC   R1
DEC   R1
```

similar to the register indirect mode except that register is incremented or decremented after or before its value is used to access memory.

Address of the operand is in a ~~Addressing~~ register whose value is incremented / decremented after or before fetching the operands from the address

⑧ Relative Addressing mode

In this mode the ~~cot~~ content of the program counter (PC) is added to the address part of the instruction in order to obtain the E.A.

E.A = PC + Address part of instruction

When address is added to the content of the PC, the result product E.A whose position in memory is relative to one address of the next instruction

This mode is often used with branch type instruction where the branch address is in the area surrounding

instruction word itself
eg - JUMP

(9) Index ~~register~~ Addressing mode -

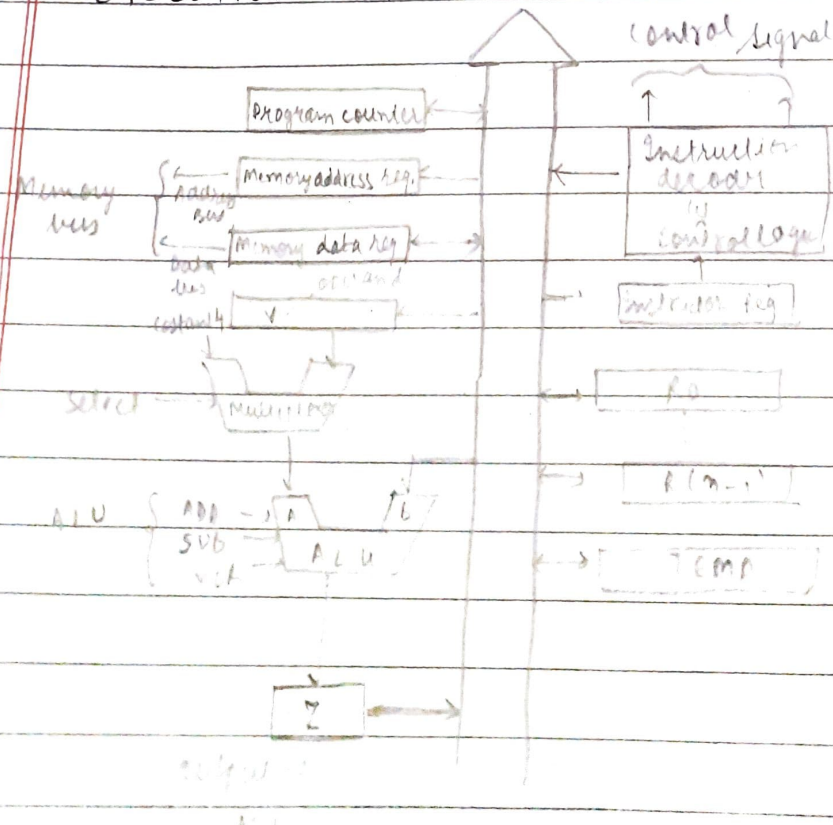
=) In this mode, the content of an index register is added to the address part of the instruction to obtain the effective address.

* Index register store an index value
→ Address ~~fi~~ field of instruction defines starting address (base address) of data array in memory
- Each operand in array is stored in memory relative to the base address

Load R, 1000 [0x]
↑ ↑ displacement
Base address

An index register hold an index no. that is relative to the address part of the instruction in index addressing mode.

- EXECUTION OF A COMPLETE INSTRUCTION



1) Fetch the Instruction from memory to IR
2) Fetch the 1st operand from memory
3) Perform addition operation
4) Load the result into R1

ADD (R3), R1 $R_3 - 1000$, $R_1 - 36$

- Control Sequence

1. PCout, MARin, Read, Select 4, Add, Zin
2. Zin, PCin, WMFC ← (Wait for Memory fn to Complete)
3. MDRout, IRin
4. R3out, MARin, Read
5. R1out, Y, YMFC
6. MDRout, Select 4, Add, Zin
7. Zout, R1in, END

Address	Content
101	Add (R3), R1

- HARDWIRED V/S MICROPROGRAMMED CPU ⇒

(Design of Control Unit).