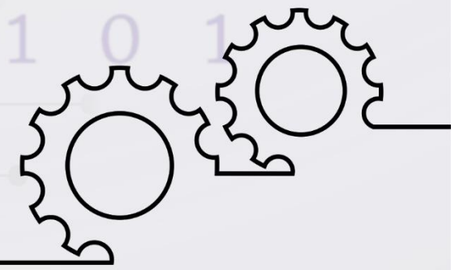
SIMATS
School of Engineering

Computer Architecture

Computer Science and Engineering



Saveetha Institute of Medical And Technical Sciences,Chennai.

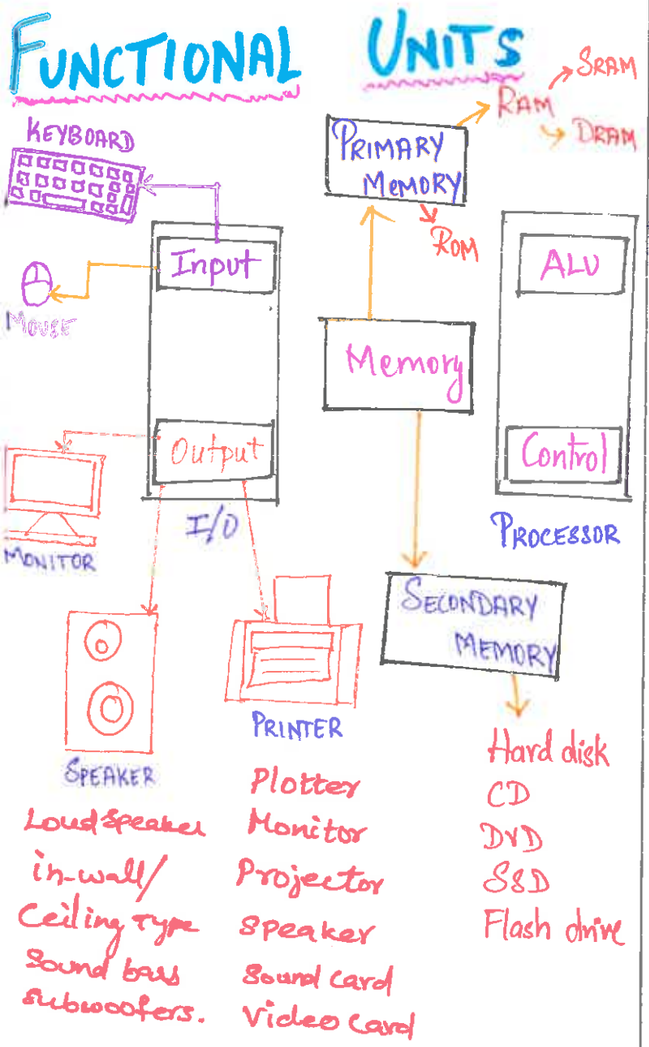
CSA12 Computer Architecture

CSA12

COMPUTER ARCHITECTURE:

Computer architecture is the Organisation of the Components which make up a Computer System and the meaning of the operations which guide its function.

FUNCTIONAL UNITS

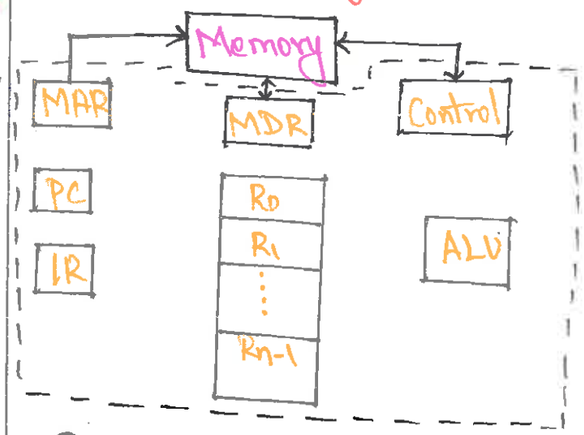


BASIC OPERATION OF COMPUTER:

I/p unit ← accepts information

information → MEMORY → ALU and Control unit → o/p → o/p unit → Pgm → Memory (stored)

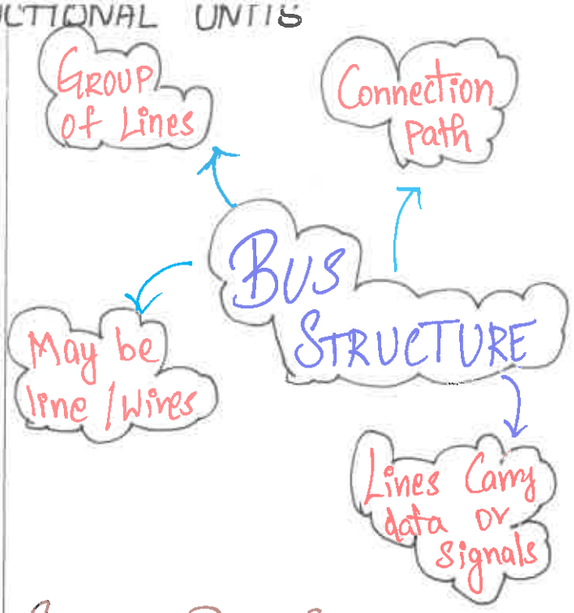
Connections between the Processor and the Memory:



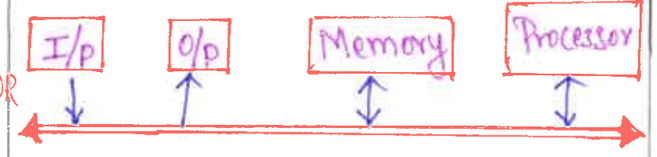
COMPUTER TYPES:

1. DIGITAL COMPUTER
2. PERSONAL COMPUTER
3. DESKTOP COMPUTER
4. NOTEBOOK COMPUTER
5. SUPERCOMPUTER
6. MAINFRAMES

FUNCTIONAL UNITS



SINGLE BUS STRUCTURE:



BUS STRUCTURES

SINGLE

* One transfer
* Low cost
* Low performance
* flexible
* Can attach peripheral.

MULTIPLE

* one/more transfer
* high cost
* Performance high

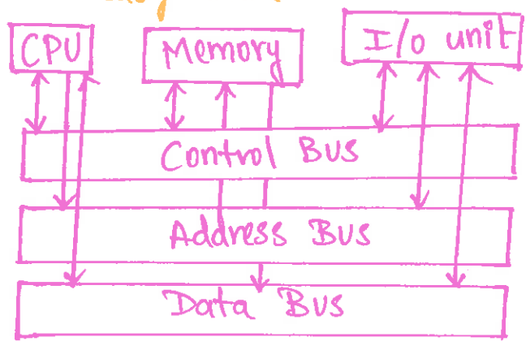
TYPES OF BUS STRUCTURE:

* Address Bus
* Data Bus
* Control Bus

Address Bus:
- Carry data
- Bidirectional
- fetch instruction from memory

Data Bus:
- Carry Data.
- Bidirectional.
- fetch inst → Memory.
- Store in memory.

Control Bus:
- Memory Read/Write
- I/o Read/Write
- Ready state.



BUFFER REGISTER:

Holds the information during data transfer.

Prevent high speed processor from being locked.

Allows the processor to switch rapidly from one device to another.

Interweaving with several devices.

DIFFERENCE BETWEEN SYNCHRONOUS AND ASYNCHRONOUS BUS

SYNCHRONOUS BUS	ASYNCHRONOUS BUS
* Works at a fixed clock rate	* Not dependent on a fixed clock rate.
* Affected by clock skew.	* Not affected by clock skew.
* Data transfer takes place in the block	* Data transfer is character-oriented.

FACTORS OF PERFORMANCE

Performance → how fast the computer executes

The best way to measure performance is using time. The following aspects are used to measure the performance of the computer

① Execution time (Response time):

The time between program starts and and the completion of the program execution.

Elapsed time:

time for execution of the program

Processor time:

period when processor active.

Processor clock: processor circuit controlled by a timing signal

Clock cycle: Regular time interval

Hertz (Hz): cycles per second

Throughput: total amount of work done in given time.

Best performance: i) Less execution time

ii) more throughput

Million denoted by mega (M)

Billion denoted by Giga (G)

500 million cycles per second = 500 mega Hertz

1250 million cycles/sec = 1.25 Giga Hertz

clock periods = 2 and 0.8 nano seconds (ns)

Formula - I

$$\text{Performance} = \frac{1}{\text{execution time}} \quad \text{— ①}$$

Reduction in execution time will increase throughput.

performance of a computer is directly related to throughput and hence it is reciprocal of execution time.

$$\text{performance}_A > \text{performance}_B \quad \text{— ②}$$

② can be written as

$$\frac{1}{\text{execution time}_A} > \frac{1}{\text{execution time}_B}$$

⇒ A is faster than B (or) B is slower than A

$$\frac{\text{performance}_A}{\text{performance}_B} = n$$

$$\Rightarrow \text{Performance}_A = n * \text{performance}_B$$

⇒ Performance of computer A is n time faster than performance of computer B

$$\frac{\text{Performance}_A}{\text{performance}_B} = \frac{\frac{1}{\text{execution time}_A}}{\frac{1}{\text{execution time}_B}} = \frac{\text{execution time}_B}{\text{execution time}_A} = n$$

Formula - II

cpu execution time / cpu time:

- It is the time spent by the cpu to compute task

$$\text{CPU time} = \text{cpu clock cycles for program} \times \text{clock cycle time}$$

$$\text{Clock cycle time} = \frac{1}{\text{Clock rate}}$$

$$\text{CPU time for a program} = \frac{\text{cpu clock cycle for a program}}{\text{clock rate}}$$

Formula - III

- In terms of instruction

cpu clock cycle for a program = No. of instruction × Average clk cycle of each instruction

Average clock cycle of each instruction = clock cycle per instruction (CPI)

$$CPI = \frac{\text{cpu clock cycle for a program}}{\text{number of instruction}}$$

Formula - IV

Basic performance equation is given by

$$T = \frac{N \times CPI}{R} = \frac{N \times CPI}{\text{Clock rate}} = N \times CPI \times \text{clock cycle time}$$

Where CPI: The average number of clock cycle each instruction takes to execute

T: cpu execution time

N: No. of instruction

R: clock rate

Example:

① If a computer$_A$ runs a program in 10sec and Computer$_B$ runs the same program in 25 sec. who is faster and how much faster?

$$\frac{\text{Performance}_A}{\text{performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

$$= \frac{25}{10} = 2.5$$

A is 2.5 times faster than B

② Computer A runs a program in 12sec with 3GHz clock. we have to design a computer B such that it can run the same program within 9sec. determine the clock rate for computer 'B' cpu design of computer B requires 1.2 times as many clock cycle of computer A.

A	B
12 seconds	9 seconds
3 GHz	?

cpu clock cycle of program in A = cpu time × clock rate

$= 12 \times 3\text{GHz}$

$= 36 \times 10^9 = 36\text{GHz}$

$$\text{cpu clock cycle of B} = \frac{1.2 \times \text{cpu clock cycle of A}}{\text{cpu time B}}$$

$$= \frac{1.2 \times 36\text{ GHz}}{9}$$

$$= \frac{1.2 \times 36 \times 10^9}{9} = 4.8\text{ GHz}$$

problem:

Consider 3 processor P_1, P_2, P_3 executing same instruction set

Processor	Clock rate	CPI
P_1	3GHz	1.5
P_2	2.5GHz	1
P_3	4GHz	2.2

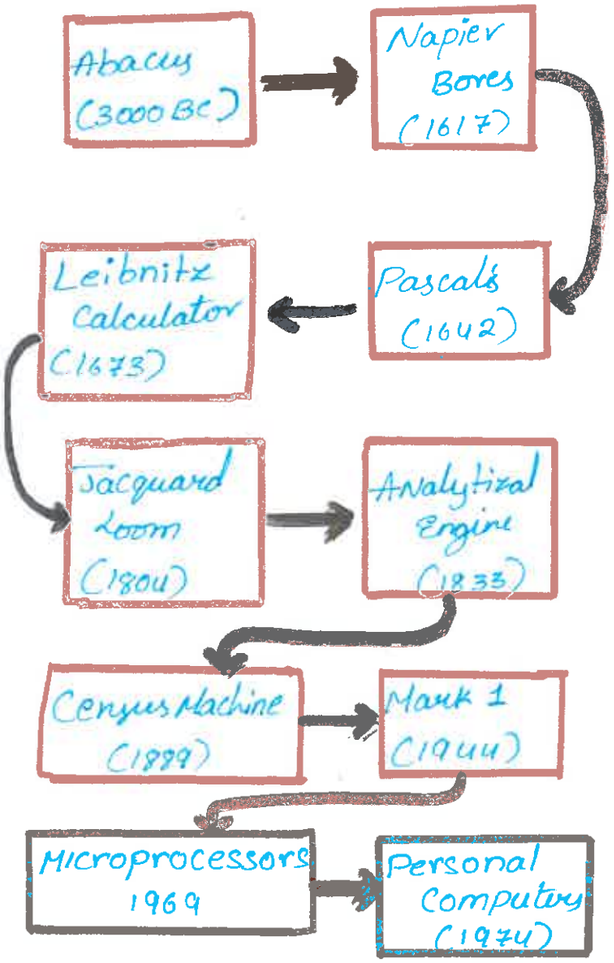
i) Which processor has the highest performance?

ii) All the processor execute a program in 10 sec. find the number of cycle required and number of instruction required for the same program

EVOLUTION OF COMPUTERS.

Generation	period	Technology used.
1st Gen.	1946-1959	Vacuum Tubes
2nd Gen.	1959-1965	Transistors
3rd Gen.	1965-1971	Integrated Cirauits
4th Gen	1971-1980	Microprocessors
5th Gen	1980-present	AI & ULSI

HISTORY OF COMPUTERS



GENERATION OF COMPUTER.

FIRST GENERATION [1940-1956]

* Vacuum Tubes – Core Techonlogy
* Machine Language – Low level
eg:- ENAC, UNIVIAC, EDVAC, EDSAC, IBM, MARK 1, MARK 2, MARK 3
* Calculation in Milli Sec.
* Large in Size * High Cost
* Storage was very less.

SECOND GENERATION [1956-1963]

* Transistors
* Early version of High level Language
* COBOL, FORTRAN
* Store instructions in Memory
* Faster * Reliable
* Calculations in Micro Seconds
* Cost was high * Cooling system.

THIRD GENERATION [1964-1971]

* Integrated circuits [ICs]
* Smaller Size than I & II GENERATION
* Input-output devices used.
* High Level Language such as BASIC, COBOL, PASCAL
* Calculations in Nano Seconds
* More Storage * Less heat
* Cooling System.

FOURTH GENERATION [1971-1980]

* Micro processor – * Smaller
* Powerfull – * Highly Reliable
* High level language Like C, C++, DBASE, FoxPRO, etc.
* Time sharing, Network based.
* Reduced heat * Less Cost
* High performance.

FIFTH GENERATION [1980-PRESENT]

* ULSI * AI software
* Parallel processing software
* Distibuted operating Systems.
* HLL includes C, C++, JAVA, .NET, PYTHON, etc.
* Eg:- Desktop, Laptop, Tablet, Mobile phone, ultra book, etc.

DATA REPRESENTATION

The Form in which data is stored, processed, and transmited.

TYPES OF DATA REPRESENTATION

Decimal – Base 10 → 0,1,2,---9.
Eg:- $(29)_{10}$

Binary – Base 2 → 0,1
Eg:- $(110)_2$

Octal – Base 8 → 0,1,2,---7 eg:- $(17)_8$

Hexadecimal – Base 16 → 0,1,2---9,A,B,C,D,E,F.
eg:- $(12B)_{16}$

ONE's Complement
→ 0-1, 1-0

Two's Complement
Add one (1) to one's complem

Bit :-
A smallest possible unit of data.

Byte
Group of eight bits is called a byte.

Nibble
Half a byte is called Nibble.

Byte Value	Bit Value
1 Byte	8 Bits
1024 Bytes	1 Kilobyte
1024 Kilobytes	1 Megabyte
1024 Megabytes	1 Gigabyte
1024 Gigabytes	1 Terabyte
1024 Terabytes	1 petabyte
1024 petabytes	1 Exabyte
1024 Exabytes	1 zettabyte
1024 zettabytes	1 yottabyte
1024 yottabytes	1 Brontobyte
1024 Brontobytes	1 Geopbytes

TEXT CODE
→ A format used to Represnt alphabets, punctuation Marks and other Symbols.

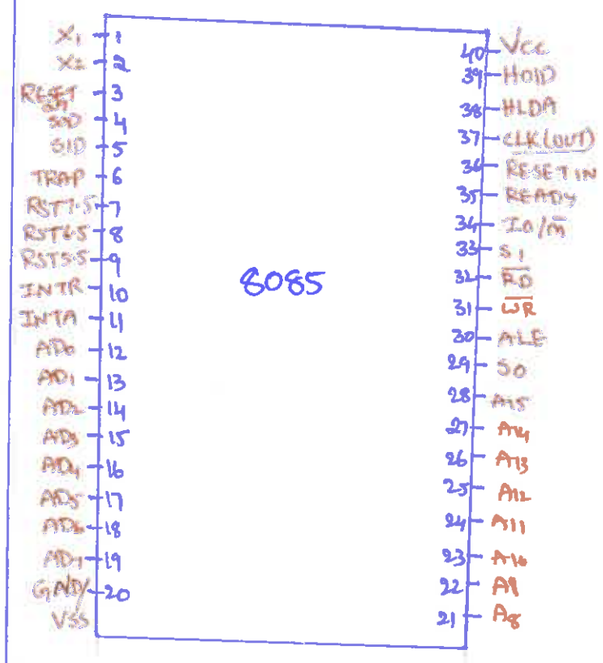
- EBCDIC – Extended Binary Coded Decimal Interchange Code.
- ASCII – American standard code for Information Interchange.
- Extended ASCII – Specifies character values from 128 to 255.
- Unicode – USES 4 to 32 bits to represent letters, numbers and symbol

MICROPROCESSOR 8085 Pin Diagram & Registers

What is microprocessor?

* Computer's Central Processing (CPU) built on single IC (Integrated circuit) is called a microprocessor.
* Microprocessor consists of ALU (Arithmetic & Logic unit), control unit and Registers.

8085 Pin Diagram :-



Some important Pins are:

AD0 – AD7 : multiplexed Address & Data lines.

A8 – A15 : Higher orders Address lines.

ALE : Address latch enable is an output signal. It goes high (1) when operation started.

S0, S1 : Status signal used to indicate type of operation.

$\overline{RD}$: used to read data from I/O devices or memory.
Read → Active Low

$\overline{WR}$: used to write data on I/O Devices or memory
write → Active Low

READY : used to check the status of output Devices.
Low → MP will wait till high
(0) (1)

TRAP : After Enabled, restart occurs and execution starts from address 0024H
→ Highest Priority interrupt
→ Non maskable interrupt

RST5.5, RST6.5, RST7.5 : * Maskable interrupt
* Low Priority than TRAP

$\overline{INTR}$ & $\overline{INTA}$: INTR → interrupt request signal after MP generates INTA (Interrupt Acknowledge) Signal. (Lowest Priority Interrupt)

IO/$\overline{M}$: Two modes of operation
1) I/O mode (IO/$\overline{M}$ = 1)
2) memory mode (IO/$\overline{M}$ = 0)

X1/X2 : * Crystal oscillator used for internal clock generation.
* Frequency divided by two.

CLK (out) : System clock for device Connected with the MP

$\overline{RESET\ IN}$: Reset the MP by setting the Program counter to zero.

RESET OUT : Reset all devices which are connected with MP.

HOLD : Another master requesting to use the address & data bus.

HLDA : * HOLD Acknowledge
* Indicates the CPU has received HOLD Request & it will relinquish the bus in next clock cycle.
* HLDA set to Low after HOLD Signal removed.

SOD : * Serial output data lines
* set/Reset Specified by SIM Instruction.

SID : * Serial Input data lines.
* Data is Loaded into Accumulator whenever RIM instruction executed

Vcc & Vss : Vcc → +5V
Vss → Ground

Status codes of 8085 :

S1	S0	Operations
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

Registers of 8085 :

* Used for temporary storage and manipulation of data & instruction
* Data remain in the register till sent to I/O devices or memory.

1) Accumulator (A) :

* 8 bit Register associated with ALU
* Hold one operand of ALU operation.

2) General Purpose Registers :

* Contains 6 general purpose registers (B, C, D, E, H & L)
* combination of two 8 bit register called register pair. (Holds 16 bit data).
(B-C, D-E & H-L)

3) Program Counter :-

* 16 bit special purpose register.
* Holds address of memory of next instruction to be executed.
* Track the instruction in a program when executed.
* Increment the content of next Program Counter (PC) during execution.

4) Stack Pointer (SP):

* 16 bit special purpose register used as memory Pointer.
* portion of RAM (controls the address of stack).

5) Instruction Register (IR):

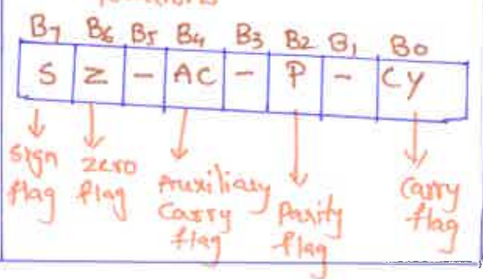
* Holds opcode of the instruction which is being decoded & executed.

6) Temporary Register : (8 bit register)

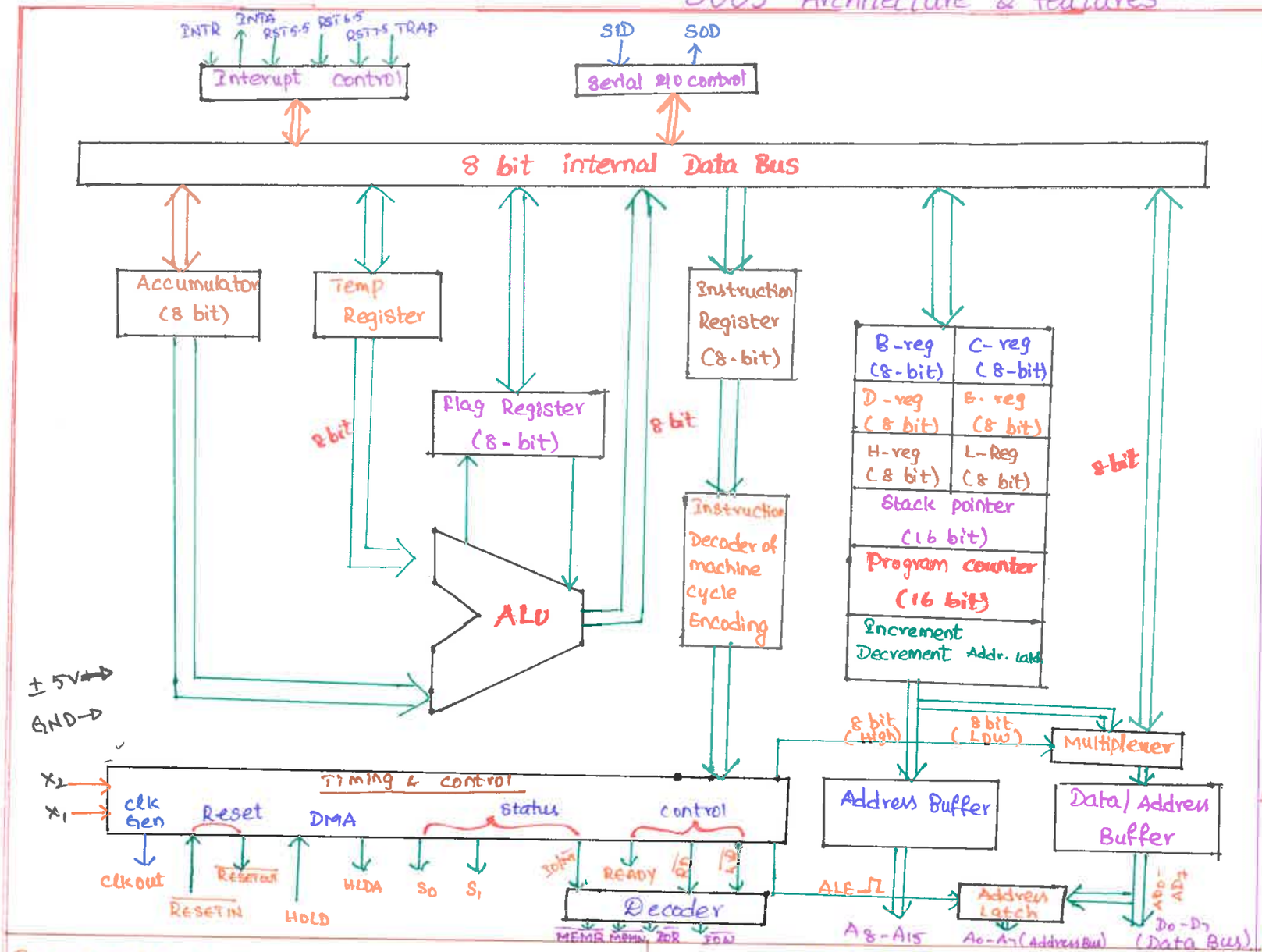
* Holds data during ALU operation.
* Not accessible to Programmer.

Flag Register :

* Five filpflops to serve as status flag.
* Set/Reset according to ALU operations



8085 Architecture & features



Architecture :-

* Interrupt control unit consists of
  a) INTR
  b) INTA
  c) RST 5.5
  d) RST 6.5
  e) RST 7.5
  f) TRAP
* Arithmetic & logic unit perform operations such as
  1) Addition
  2) subtraction
  3) Logical AND
  4) Logical OR
  5) Logical EXOR
  6) Logical NOT
  7) Increment
  8) Decrement
  9) shift, etc
* Timing & control unit consists of
  1) clock generation
     a) X_1, X_2
     b) CLK.OUT
  2) RESET
     a) $\overline{\text{RESET IN}}$
     b) RESET OUT
  3) Control
     a) Ready
     b) $\overline{RD}$
     c) $\overline{WR}$
  4) Status
     a) S_0, S_1
     b) $IO/\overline{M}$
  5) DMA
     a) HOLD
     b) HLDA

Flag Register

eg:-

```
        AC
    1 0 1 1 1 0 1 0
  + 0 1 1 0 1 0 0 1
  ---------------
  1 0 0 1 0 0 0 1
  cy
```

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	X	1	X	0	X	1
S	Z	-	Ac	-	P	-	cy

Status of different control signal for different operation:

		$IO/\overline{M}$	S_1	S_0	Control $\overline{RD}$	Control $\overline{WR}$	Control $\overline{INTA}$
Machine cycle	Opcode Fetch (OF)	0	-	-	0	-	-
	Memory Read ($\overline{MEMR}$)	0	-	-	0	-	-
	Memory write ($\overline{MEMW}$)	0	0	0	-	0	-
	I/O Read ($\overline{IOR}$)	1	-	-	0	-	-
	I/O write ($\overline{IOW}$)	1	0	0	-	0	-

Features of 8085 Hp:

* 40 pin IC package fabricated on a single LSI chip
* uses a single ± 5 vd.c
* clock speed about 3MHz, clock cycle of 320ns
* 8 bit Data bus
* 16-bit Address Bus (address upto 64 kB)
* 16 bit stack pointer
* 16 bit program Counter (pc)
* six 8 bit registers (B, C, D, E, H, L)
  Register pair (B-C, D-E, H-L)
* consists of 80 basic instruction sets & 246 opcodes

Application: Mobile phones, Microwave ovens, etc.

8085 Instruction Set

Instruction Format:

* One Byte Instructions
  Eg: RRC, CMA, MOV B, C
* Two Byte Instructions
  Eg: MVI B, 45H
* Three Byte Instructions
  Eg: CALL, JMP

Instruction set:

1) Data transfer Instruction Set:

* No flags are affected

Instruction set	Explanation	Addressing	Example
MOV r_1, r_2 1→Machine Cycle (MC)	$r_1 \leftarrow r_2$	Register	MOV A, B
MOV r, m 2→MC	r ← [HL] (memory)	Register Indirect	MOV B, M
MOV M, r 2←MC	[HL] ← r	Register Indirect	MOV M, C
MVI r, Data 3←MC	r ← data	Immediate	MVI B, 45H
LXI rp, Data 3←MC	rp ← 16 bit data rh ← 8 bit MSB rL ← 8 bit LSB	Immediate	LXI H, 2000H
LDA addr 4←MC	A ← [addr]	Direct	LDA 2000H
STA addr 4←MC	[addr] ← A	Direct	STA 3000H
LHLD addr SHLD addr 5←MC	[L] ← [addr] [H] ← [addr+1] [addr] ← L [addr+1] ← H	Direct Direct	LHLD 2500H SHLD 2600H
XCHG 1←MC	[HL] ↔ [DE]	Register	XCHG

* Similar to LDA & STA, LDAX for 16 bit notation (2→MC)

2) Arithmetic Instruction:-

Instruction set	Explanation	Flags	Addressing	Example
ADD r 1←MC	A ← A + r	ALL	Register	ADD B
ADD M 2←MC	A ← A + [H-L]	ALL	Register indirect	ADD M
ACC r 1←MC	A ← A + r + CY	ALL	Register	ACC B
ADC M 2←MC	A ← A + [HL] + CY	ALL	Register indirect	ADC M
ADI data 2←MC	A ← A + Data	ALL	Immediate	ADI 45H
ACI data 2←MC	A ← A + data + CY	ALL	Immediate	ACI 50H
DAD rp 3←MC	H-L ← HL + [rp]	CY	Register	DAD B

Similar to Addition, Subtraction can be expressed as SUB r, SUB M, SBB M SUI data, SBI data

INR r 1←MC	r ← r+1	All except CY	Register	INR B
INR M 3←MC	[HL] ← [HL]+1	All except CY	Register indirect	INR M
INX rp 1←MC	rp ← rp+1	None	Register	INX H
DAA	Decimal Adjust Acc	–	–	DAA

Similar to increment, Decrement instruct is expressed as DCR r, DCR M, DCX rp.

3) Logical Instruction:

Instruction set	Explanation	Flags	Addressing	Example
ANA r 1←MC	A ← (A) ∧ (r) (AND)	All	Register	ANA C
ANA M 2←MC	A ← (A) ∧ (M) (HL)	All	Register indirect	ANA M
ANI data 2←MC	A ← (A) ∧ (data)	All	Immediate	ANI 22H

Similar AND operation, OR and EXOR operations can expressed as

1) ORA r, ORA M, ORI data eg: (A ← (A) ∨ (r) ∨ (M)
   A ← (A) ∨ (M)
2) XRA r, XRA M, XRI data eg: A ← (A) ⊕ (r)
   A ← (A) ⊕ (M)

CMA CMC 1←MC	A ← $\bar{A}$ CY ← $\bar{CY}$	None CY	Implicit Implicit	CMA CMC
STC 1←MC	CY ← 1 Set Carry	CY	Implicit	STC
CMP r 1←MC	Compare A and r-register (A>B, A<B, A=B)	All	Register	CMP B
CMP M 2←MC CPI data 2←MC	CMP A & M CMP A & data	All All	Indirect Immediate	CMP M CPI 22H
RLC	$A_{N+1} \leftarrow A_N$ $A_0 \leftarrow A_7$ CY ← A_7	CY	Implicit	RLC
RRC	Reverse of RLC $A_N \leftarrow A_{N+1}$ $A_7 \leftarrow A_0$ $A_7 \leftarrow$ CY	CY	Implicit	RRC
RAL	$A_{N+1} \leftarrow A_N$ CY ← A_7 A_0 ← CY	CY	Implicit	RAL

* Similarly RAR with Reverse steps of RAL

BRANCHING LOOPING, COUNTING and INDEXING OPERATIONS

JUMP:

CONDITIONAL JUMP INSTRUCTIONS:-

Instruction	Explanation	Example
JC	Jump on Carry if Carry flag is '1'	JC 2000
JNC	Jump on No Carry If Carry flag is '0'	JNC 2050
JZ	Jump on Zero if Zero flag is '1'	JZ 2000
JNZ	Jump on NO Zero, If Zero flag is '0'	JNZ 2050
JPE	Jump on Parity Even ; PF is '1'	JPE 2000
JPO	Jump on Parity ODD ; PF is '0'	JPO 2050
JM	Jump on Sign 'MINUS' If SF is '1'	JM 2000
JP	Jump on Sign 'plus' If SF is '0'	JP 2050

CALL:

CONDITIONAL CALL INSTRUCTIONS:-

Instruction	Explanation	Example.
CC	Call on Carry flag CF: 1	CC 2000
CNC	Call on No Carry flag CF: 0	CNC 2050
CZ	Call on Zero flag ZF: 1	CZ 2000
CNZ	Call on No Zero flag ZF: 0	CNZ 2050
CPE	Call on Parity Even PF: 1	CPE 2000
CPO	Call on Parity ODD PE: 0	CPO 2050
CM	Call on Minus SF: 1	CM 2000
CP	Call on Plus SF: 0	CP 2050

Three Types of Branching :-

① JUMP (Conditional & unConditional)
② CALL (Conditional & unConditional)
③ RET (Conditional & unconditional).

UNCONDITIONAL JUMP :-

JMP address ; Jump to the address

Ex: JMP 2000.

UN CONDITIONAL CALL :-

CALL address ; Call to the address

UN CONDITIONAL RETURN :-

RET address ; Return to the address

Ex: CALL 2000

Ex: RET

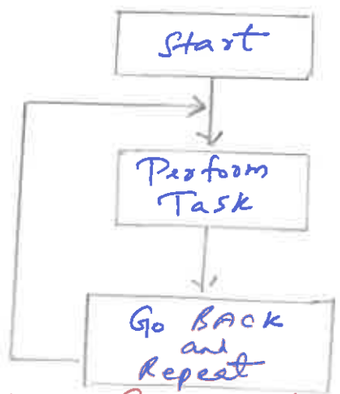
Looping in 8085 :-

→ Instructs MicroProcessor to repeat tasks.

① Continous Loops
② Conditional Loops.

Continous Loops :- Repeats a task Continously.

Conditional Loop :- Repeats a task if Some conditions are Satisfied



Flowchart for Continous loop

RETURN:-

CONDITIONAL RET INSTRUCTIONS

Instruction	Explanation	Example
RC	Return on Carry CF: '1'	RC
RNC	Return on No Carry CF: '0'	RNC
RZ	Return on Zero ZF: '1'	RZ
RNZ	Return on No zero ZF: '0'	RNZ
RPE	Return on Parity EVEN PF: '1'	RPE
RPO	Return on Parity ODD PF: 0	RPO
RM	Return on Minus Signflag: '1'	RM
RP	Return on plus signflag: '0'	RP

INPUT- OUTPUT & STACK Instructions :-

* IN Port address ; Input to accumulator from I/o Port
* OUT Port address ; output from accumulator to I/o Port
* PUSH rp ; Push Content rp to stack.
* PUSH PSW ; Push processor status word.
* POP rp ; PoP Content from Stack.
* POP PSW ; PoP Processor Status Word.

MACHINE CONTROL INSTRUCTIONS:-

* HLT ; Halt
* XTHL ; Exchange stack top with H-L.
* SPHL ; HL-pair to stack pointer.
* EI ; Enable Interrupt
* DI ; Disable Interrupt
* SIM ; Set Interrupt Mask
* RIM ; Read Interrupt Mask
* NOP ; No operation.

8086 MICROPROCESSOR

FEATURES

→ Design : Intel - 1976 → Updated Version of 8085 Microprocessor

→ 16 bit, N-Channel High Speed Metal Oxide Semiconductor

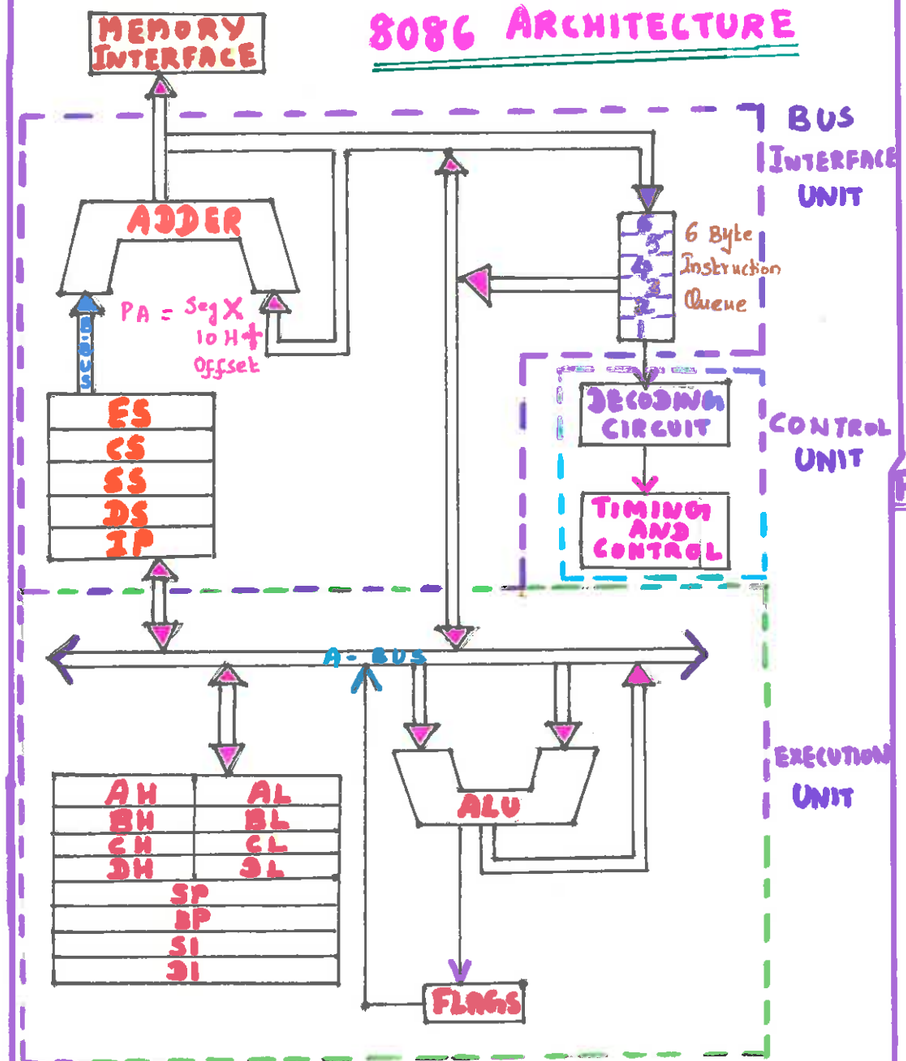
→ Built on Single semi conductor chip and packed in 40 pin IC pack (Dual Inline Package)

→ Uses 20 Address lines & 16 Data lines

→ Addresses upto 2^{20} = 1 Mbyte Memory

→ Size of I/O : 2^{16} = 64 kB

8086 ARCHITECTURE



→ 8086 Microprocessor consists of two units

1. Bus Interface Unit
   - ↳ Contains Segment Registers, Instruction Pointer and 6 Byte Instruction Queue
2. Execution Unit
   - ↳ Contains Programmable Registers, Index Registers, Pointer Register and Flag Registers. & ALU

REGISTERS

GENERAL PURPOSE REGISTERS

SEGMENT REGISTERS

POINTER & INDEX REGISTERS

AX - Accumulator
Bx - Base
Cx - Counter
Dx - Data

CS - Code Segment
SS - Stack Segment
Ds - Data Segment
ES - Extra Segment

SP - Stack Pointer
BP - Base Pointer
SI - Source Index
DI - Destination Index
IP - Instruction Pointer

Two Operating modes : Maximum mode, Minimum mode

FLAGS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	A_c	X	P	X	C_y

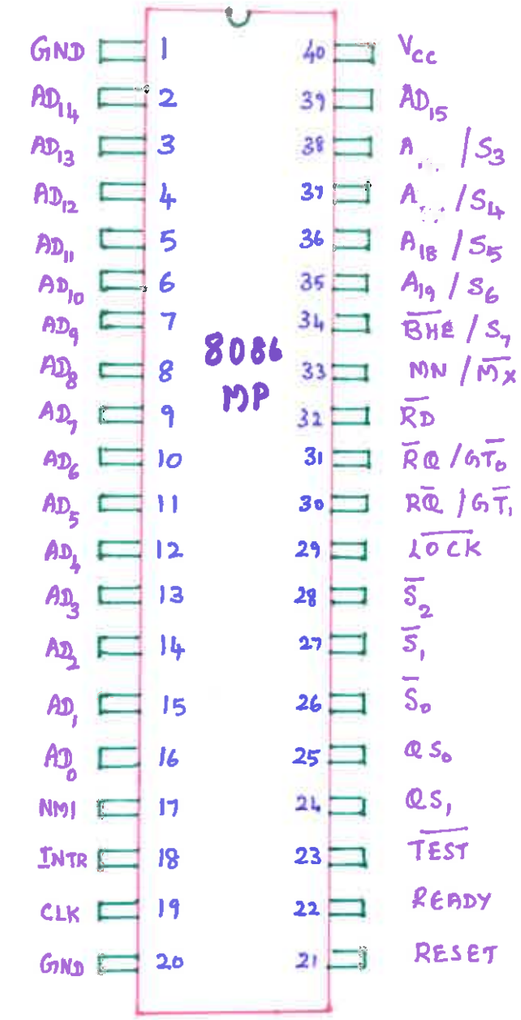
Conditional Flags

* Carry Flag (C_y)
* Auxiliary Flag (A_c)
* Parity Flag (P)
* Zero Flag (Z)
* Sign Flag (S)
* Overflow Flag (O)

Control Flags

* Trap Flag (T)
* Interrupt Flag (I)
* Direction Flag (D)
  - Clear Direction Flag
  - Set Direction Flag

PIN CONFIGURATION



* V_{cc} -> Power Supply => +5 V
* $AD_0 - AD_{15}$ -> A 16 Bit Address Data Bus
* $A_{16} - A_{19}$ -> Higher Order Address lines & Multiplexed with status signals
* $\overline{BHE}$ / S_7 -> Bus High Enable / Status
* $\overline{RD}$ -> Read , GND - Ground
* RESET -> System Reset
* CLK (i/p) -> Clock 5, 8, 10 MHz
* INTR -> Interrupt Request
* NMI -> Non Maskable Interrupt Request

8086 INSTRUCTION SET

INSTRUCTION SETS OF 8086 MICROPROCESSOR

TYPES OF INSTRUCTION SETS

* DATA TRANSFER
* ARITHMETIC
* LOGICAL
* STRING MANIPULATIONS
* CONTROL TRANSFERS
* PROCESSOR CONTROL

* DATA TRANSFER

MOV = MOVE
Register/Memory to/from Register
Immediate to Register/Memory
Immediate to Register
Memory to Accumulator
Accumulator to Memory
Register/Memory to Segment Register
Segment Register to Register/Memory
PUSH = Push:
Register/Memory
Register
Segment Register
POP = POP:
Register/Memory:
Register
Segment Register
XCHG = Exchange
Register/Memory with Register
Register with Accumulator
IN = Input from:
Fixed port
Variable port
XLAT = Translate Byte to AL
LEA = Load EA to Register
LDS = Load pointer to DS
LAHF = Load AH with Flags
LES = Load pointer to ES
SAHF = Store AH into Flags
PUSHF = push Flags
POPF = POP flags

* ARITHMETIC

ADD = Add:
Reg/Memory with Register to Either
Immediate to Register/Memory
ADC = Add with carry:
Reg/Memory with Register to Either
Immediate to Register/Memory
Immediate to Accumulator
INC = Increment:
Register/Memory
Register
AAA = ASCII Adjust for Addition
DAA = Decimal Adjust for Addition
SUB = Subtract
Reg/Memory and Register to Either
Immediate from Register/Memory
Immediate from Accumulator
SBB = Subtract with Borrow
Reg/Memory and Register to Either
Immediate from Register/Memory
Immediate from Accumulator
DEC = Decrement:
Register/Memory
Register
NEG = Change sign
CMP = compare:
Register/Memory and Register
Immediate with Register/Memory
Immediate with Accumulator
AAS = ASCII Adjust for Subtract
DAS = Decimal Adjust for Subtract
MUL = Multiply (unsigned)
IMUL = Integer Multiply (signed)
AAM = ASCII Adjust Multiply
DIV = Divide (Unsigned)
IDIV = Integer Divide (signed)
AAD = ASCII Adjust for Divide
CBW = convert Byte to word
CWD = convert word to Doubleword

* LOGICAL

NOT = Invert
SHL/SAL = Shift Logical/Arithmetic Left
SHR = Shift Logical Right
SAR = Shift Arithmetic Right
ROL = Rotate Left
ROR = Rotate Right
RCL = Rotate through carry Flag. Left
RCR = Rotate through carry Right
AND = And:
Reg/Memory and Register to Either
Immediate to Register/Memory
Immediate to Accumulator
TEST = And Function to Flags, No Result:
Register/Memory and Register
Immediate Data and Register/Memory
Immediate Data and Accumulator
OR = OR:
Reg/Memory and Register to Either
Immediate to Register/Memory
Immediate to Accumulator
XOR = Exclusive OR:
Reg/Memory and Register to Either
Immediate to Register/Memory
Immediate to Accumulator

* STRING MANIPULATIONS

REP = Repeat
MOVS = Move Byte/word
CMPS = compare Byte/word
SCAS = scan Byte/word
LODS = Load Byte/wd to AL/AX
STOS = Stor Byte/wd from AL/AX

BRANCHING AND LOOPING OF 8086

INSTRUCTION SETS OF 8086 MICROPROCESSOR

* CONTROL TRANSFER

CALL = call :
Direct within Segment
Indirect within segment
Direct Intersegment
Indirect Intersegment
JMP = unconditional Jump :
Direct within Segment
Direct within Segment -short
Indirect within segment
Direct Intersegment
Indirect Intersegment
RET = Return from CALL :
within Segment
within Seg Adding Immediate to Sp
Intersegment
Intersegment Adding Immediate to Sp
JE/JZ = Jump on Equal / Zero
JL/JNGE = Jump on Less / Not Greater or Equal
JLE/JNG = Jump on Less or equal / Not Greater
JB/JNAE = Jump on Below / Not Above or Equal
JBE/JNA = Jump on Below or equal / Not above
JP/JPE = Jump on parity / Parity Even
JO = Jump on Overflow
JS = Jump on Sign
JNE/JNZ = Jump on Not Equal / Not Zero.
JNL/JGE = Jump on Not Less / Greater or Equal
JNLE/JG = Jump on Not less or equal / Greater
JNB/JAE = Jump on Not Below / Above or Equal
JNBE/JA = Jump on Not Below or equal / Above
JNP/JPO = Jump on Not par / par odd
JNO = Jump on Not overflow
JNS = Jump on Not sign
LOOP = LOOP CX Times
LOOPZ/LOOPE = LOOP while Zero / Equal
LOOPNZ/LOOPNE = LOOP while Not Zero / equal
JCXZ = Jump on CX Zero
INT = Interrupt
Type Specified
Type 3
INTO = Interrupt on overflow
IRET = Interrupt Return
Eg: INT 02H → Type 02 Interrupt

* PROCESSOR CONTROL

CLC = clear carry
CMC = complement carry
STC = set carry
CLD = clear Direction
STD = Set Direction
CLI = clear Interrupt
STI = set Interrupt
HLT = Halt
WAIT = wait
ESC = Escape (to external Device)
LOCK = Bus Lock prefix

Operation :
* STC → To set carry flag to 1
* CLC → clear / reset carry flag to '0'.
* CMC → put complement at the stat of CF
* STD & CLD
* Set Direction flag to '1'
→ Clear / resent Direction flag to '0'.
* Directing the CPU Execution Sequence in an order
* Controlling CPU

* CONDITIONAL BRANCH INSTRUCTION

1. JZ/JE → Zero
2. JNZ/JNE → NOT Zero / E
3. JS → Sign
4. JNS → NOT Sign
5. JO → over-flow
6. JNO → NOT over-flow
7. JP/JPE → parity / parity even
8. JNP → NOT parity
9. JB/JNAE/JC → Below, Not Above (or) Equal
10. JNB/JAE/JNC → Not Above, Above (or) equal
11. JBE/JNA → ~~Above~~ Below / equal
12. JNBE/JA → Above
13. JL/JNGE → Less, Not greater (or) equal
14. JNL/JGE → Not less, Greater (or) equal
15. JLE/JNC → Less / equal
16. JNLE/JE → Not less / equal

UNCONTITIONAL CALL JUMP, RETURNE :

Eg: JMP 2050H
RET
CALL 2000H

INSTRUCTION SET:

* Set of Codes that Computer processor Can Understand.
* Code is 1s and 0s, or Machine Language.
* Contains instruction or tasks that Control the movement of bits and bytes within the processor.

Examples of Some Instruction Sets -

- **ADD** - Add two numbers together.
- **JUMP** - Jump to designated RAM address.
- **LOAD** - Load information from RAM to the CPU.

TYPES OF INSTRUCTION SET

1. Reduced Instruction Set Computer (RISC)

the Concept of RISC Involves an attempt to reduce execution time by Simplifying the Instruction set of Computers

- Relatively Few Instructions.
- Relatively few addressing modes.
- Memory access limited to load and Store instructions.
- All operations done within the register of the CPU.
- Single-Cycle Instruction execution
- Fixed length, easily decoded Instruction Format.
- Hardwired rather than micro programmed Control.

2. Complex Instruction Set Computer (CISC)

CISC is a computer where a Single Instruction can perform numerous low-level operations like a load From memory and a Store From memory, etc.

- A large number of Instruction typically From 100 to 250 instructions.
- Some Instruction that performs Specialized tasks and are used infrequently.
- A large Variety of addressing modes - typically from 5 to 20 different modes.
- Instruction that manipulate operands in memory.

INSTRUCTION FORMATS

groups of bits used to perform a particular operation on the data.

Elements of Instructions

- Opcode
- Operand
- address → Source, destination
- Next Instruction

General Instruction format

opcode	operand address	operand address 2
4 bit	6 bit	6 bit

Instruction types

- According to type of operation
- No. of Addresses

INSTRUCTION TYPES:

⇒ Data processing - ALU Instruction

⇒ Data Storage - Memory Instruction

⇒ Data Movement - Data transfer Instruction

⇒ Control - Test and Branch Instruction

According to No. of Address

* 3 address Instruction

3 operands Used

Eg: ADD C, A, B [C ← A+B]

* 2 address Instruction

2 operands Used

Eg: ADD A, B [A ← A+B]

* One address Instruction

Only One operand Used

Eg: ADD B [AC] ← B + [AC]

AC = Accumulator

* ZERO Address Instruction

Eg: CMA [AC] ← [$\overline{AC}$]

Memory Reference Instruction

MODE	OPCODE	MEMORY ADDRESS
15 14	12 11	0

OPCODE = 000 To 110

Register Reference Instruction

MODE	OPCODE	REGISTER
15 14	12 11	0

OPCODE = 111 AND Addressing MODE I = 0

Input & Output -

MODE	OPCODE	

OPCODE = I = 1

INSTRUCTION FORMAT

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

ZERO ADDRESS FORMAT

```
PUSH A    TOP = A
PUSH A    TOP = B
ADD       TOP = A+B
PUSH C    TOP = C
PUSH D    TOP = D
ADD       TOP = C+D
MUL       TOP = (C*D) * (A+B)
POP X     M[X] = TOP
```

ONE ADDRESS FORMAT

```
LOAD A    AC = M[A]
ADD B     AC = AC + M[A]
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
```

TWO ADDRESS FORMAT

```
MOV R1, A     R1 = M[A]
ADD R1, B     R1 = R1 + M[B]
MOV R2, C     R2 = C
ADD R2, D     R2 = R2 + D
MUL R1, R2    R1 = R1 * R2
MOV X, R1     M[X] = R1
```

THREE ADDRESS FORMAT

```
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
```

ADDRESSING MODES

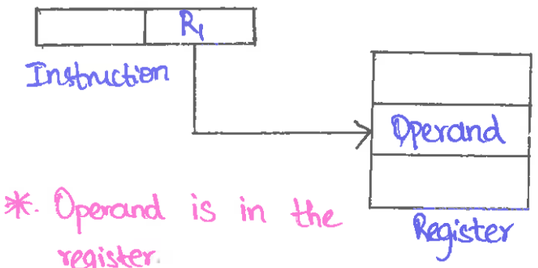
- The way in which the operand of an instruction is specified.

Effective address (EA).

- The memory address obtained from the computation dictated by the given addressing mode.

Types of Addressing Modes:

i) Register addressing mode.



*. Operand is in the register.

*. name of register is specified in the instruction.

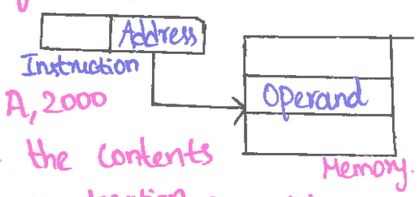
Eg: MOV R1, R2

- copies constants of register R_2 to R_1

EA - NO, Use: Fast Access.

ii) Absolute or Direct Addressing mode.

*. Operand address is given explicity as part of the instruction.



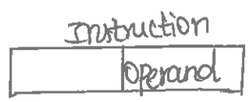
Eg:- MOV A, 2000

↳ Copies the contents in memory location 2000 into register.

iii) Immediate Addressing Mode:

→ Operand is explicitly given in the instruction.

→ # symbol indicates, value is immediate operand.

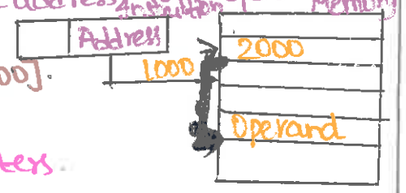


Eg: MOV A, #20

↳ Copies operand 20 into register A.

iv) Indirect addressing mode:

→ Instruct Contains address of memory which refers - address of operand.



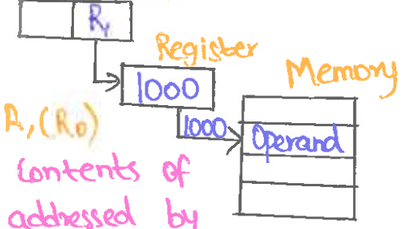
Eg: MOV A, [1000].

[] - Memory

use: for pointers.

v) Register Indirect Addressing Mode:

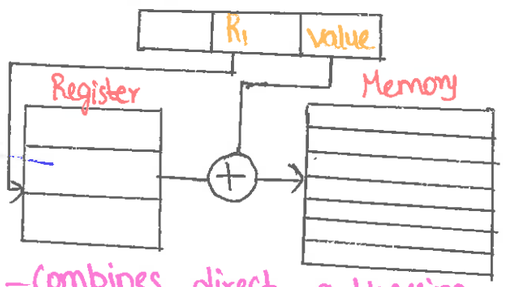
- Register Contains address of Operand



Eg: MOV A, (R_0)

↳ Copies contents of memory addressed by Contents of register R_0 into A.

vi) Displacement addressing Mode:



- Combines direct addressing and register indirect addressing.

EA = Base value + (R)

↓

Contents of referenced register.

Eg: MOV R, 20 [R_1].

20 + [1000].

EA ⇒ 1020

3 Variations;

↳ Relative addressing

↳ Base register

↳ Index addressing

vii) Relative Addressing Mode:

- referenced register is program Counter (Pc) → also known as Pc-relative addressing.

EA = PC + Address part of instruction.

Eg:- JNZ BACK

→ Causes program execution to go to the branch target location (BACK) - if condition is satisfied.

viii) Base register addressing:

- The reffered register contains main memory address.
- Address field contains - displacement

EA = A + (R) R - Base address

A - Displacement

Eg:- MOV B, [R+8] R → 1000 ⇒ 1000 + 8 = 1008.

ix) Index Addressing Mode:

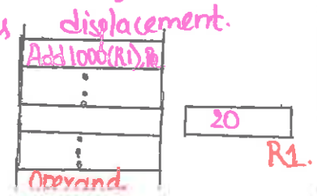
EA = memory address + (R).

Eg: MOV R_1, [R_1 + RI]

↳ Reference register RI gives the displacement.

↓

Main memory adress given by R_1.



x) Auto Increment Addressing Mode:

- After accessing the operand, the contents of this register are auto incremented.

Eg: MOV R0, (R_2)+:

↳ copies R_0 into memory location whose address is specified in R_2.

↳ R_2 is incremented then.

xi) Autodecrement Addressing Mode:

- Contents of a register specified in the Inst are decremented and then used as EA.

Eg: MOV R_1, -(R_0).

xii) Stack Addressing Mode:

→ SP

↳ Contains the address of Top of stack (TOS)

↳ operand stored.

Eg: PUSH R.

↳ decrement sp and copies Content of R on TOS pointed by stack pointer.

ASSEMBLY LANGUAGE.

A type of low-level programming language that is intended to Communicate directly with a Computer's hardware.

Integer Arithmetic – Addition & Subtraction

Relation between Addition & Subtraction:-

$(\pm A) - (+B) = (\pm A) + (-B)$

$(\pm A) - (-B) = (\pm A) + (+B)$

3 major representation of Signed integers:-

* Sign and magnitude
* one's complement (1's)
* Two's complement (2's)

Binary Signed integer representation:-

$b_3 b_2 b_1 b_0$	Sign + magnitude	1's complement	2's complement
0000	+0	+0	+0
0001	+1	+1	+1
0010	+2	+2	+2
0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Addition in 2's complement :-

(4) 0100 +(3) 0011 (7) 0111

(-4) 1100 +(3) 0011 (-1) 1111

Subtraction in 2's complement:-

Step 1:- Take 2's complement of B

Step 2:- Result => A + Step 1 Answer

Step 3:- if carry generated, result => +ve (Ignore carry)

Step 4:- If no carry, result => -ve

Ex:-

$(28)_{10} - (15)_{10}$

$(28)_{10}$ 011100

2's complement of $(15)_{10}$ 110001

1 | 001101 $(13)_{10}$

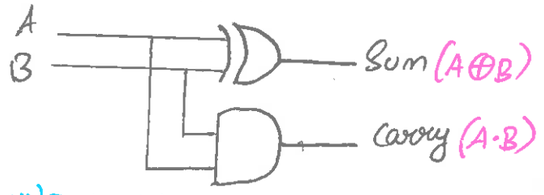
ignore

Overflow :- (+) + (+) = -ve condition (-) + (-) = +ve

Adders:

(i) Half adder :-

Inputs		Outputs	
A	B	carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



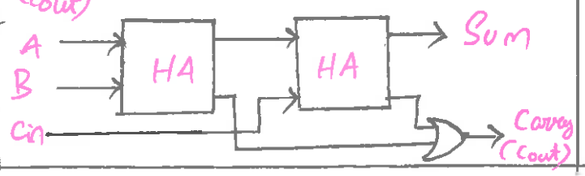
(ii) Full adder

Inputs			Outputs	
A	B	C_{in}	carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$Sum = A \oplus B \oplus C_{in}$

$Carry = AB + BC + CA$

(C_{out})



Carry Look Ahead Adder (CLA):

* CLA improves Speed by reducing the amount of time required to determine carry bits.

CLA concept :-

* create two Signals :
  1.) P (carry Propagator)
  2.) G (carry generator)
* carry propagator (P) is propagated to the next level
* carry generator (G_i) is used to generate the output carry regardless of input carry.

$P_i = A_i \oplus B_i$ — (1)

$G_i = A_i \cdot B_i$ — (2)

$Sum = P_i \oplus C_i$ — (3)

$C_{i+1} = G_i + (P_i \cdot C_i)$ — (4)

* For designing 4-bit CLA, we need i = 0, 1, 2, 3

Substitute i = 0, 1, 2, 3 in

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

$C_3 = G_2 + P_2 \cdot C_2$

$= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$

$C_4 = G_3 + P_3 C_3$

$= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

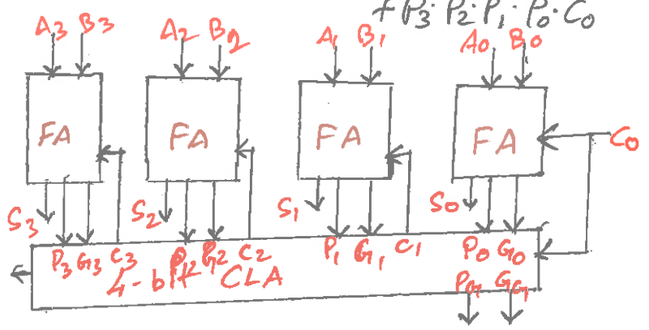
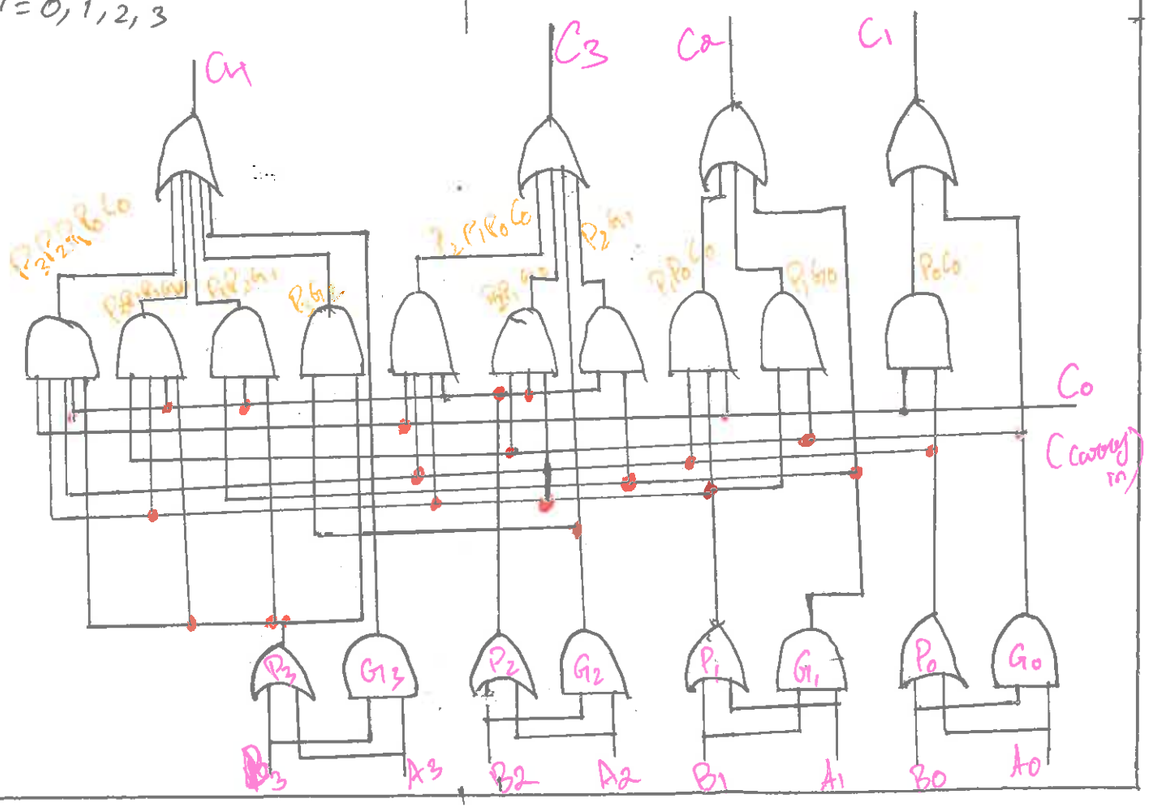


Figure:- 4 bit CLA using Full Adder (FA)

Similarly

$C_{i+1} = G_i + P_i \cdot G_{i-1} + P_i P_{i-1} G_{i-2} + \dots$

$\dots + P_i P_{i-1} \dots P_1 G_0 + P_i P_{i-1} \dots P_0 C_0$



multiplication of unsigned numbers:-

* Unsigned multiplication can be viewed as addition of shifted versions of the multiplicand.
* Product of 2 n bit numbers is at most a 2 n-bit number.

Exc:-

```
    1 1 0 1   (13 - multiplicand) (M)
    1 0 1 1   (11 - multiplier (Q))
    -------
      1 1 0 1
    1 1 0 1
  0 0 0 0
1 1 0 1
-----------------
1 0 0 0 1 1 1 1   (143 - Product)
```

Signed multiplication :-

* Take 2's complement if number is negative
* padding of 5 one's if MSB of first multiplication is one, otherwise zero's should be padded.

Exc :- (-13) x (11)

```
      1 0 0 1 1   (-13)
      0 1 0 1 1   (+11)
    ---------
  1 1 1 1 1 1 0 1 1
  1 1 1 1 1 0 0 1 1
  0 0 0 0 0 0 0 0 0
  1 1 1 1 0 0 1 1
  0 0 0 0 0 0
-----------------------
1 1 0 1 1 1 0 0 0 1   (-143)
```

Product

Booth Algorithm :-

* In general, Booth Algorithm has 3 Schemes

i) '0' to '1' ⇒ add "-1" time Shifted 'M'
ii) '1' to '0' ⇒ add "+1" time Shifted 'M'
iii) '0' to '0' OR '1' to '1' ⇒ add 0

Exc :-
(+13) x (-6)

```
    0 1 1 0 1   (+13)
  x 1 1 0 1 0   (-6)
```

⇓

```
    0 1 1 0 1   (M)
  +1 -1 +1 -1 0
  ---------------
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 1 1
0 0 0 0 1 1 0 1
1 1 1 0 0 1 1
0 0 0 0 0 0
-----------------------
1 1 1 0 1 1 0 0 1 0   (-78)
```

→ Recoded Value

Product

Booth multipliers table:-

multiplier		Version of multiplicand Selected by bit i
i	i-1	
0	0	0 x M
0	1	+1 x M
1	0	-1 x M
1	1	0 x M

Worst multiplier :-
1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0

ordinary multiplier :-
1 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0

Worst case multiplier :-
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Division :-

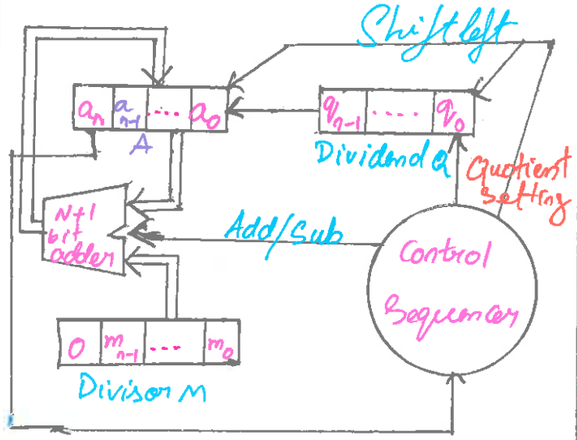
Decimal :-

```
      21
13 | 274
     26
     ---
      14
      13
      --
       1
```

Binary :-

```
              10101
1101 | 10001010010
       1101
       ------
        10000
         1101
         ----
          1110
          1101
          ----
             1
```

Circuit Arrangement for binary division :-



(i) Restoring Division :-

Step 1 :- Shift A and Q left one binary position
Step 2 :- Subtract M from A, & place result back in A
Step 3 :- If the sign of A is 1, set q_0 to 0 and M back to A (restore A); otherwise set q_0 to 1
Step 4 : Repeat above steps n times.

Exc :-

8/3

```
      10
11 | 1000
     11
     --
      10
```

8 :- 1000
3 :- 0011

```
Initially   0 0000   1000
              0011
Shift       0 0001   0 000□
Subtract    1 1101
Set q0      1 1110   0 0000   } First cycle.
Restore       11
            0 0001   0 0000
Shift       0 0010   0 000□
Subtract    1 1101
Set q0      1 1111   0 0000   } Second cycle
Restore       11
            0 0010   0 0000
Shift       0 0100   0 000□
Subtract    1 1101
Set q0      0 0001   0 0001   } Third cycle
Shift       0 0010   0 001□
Subtract    1 1101
Set q0      0 0111   0 0010   } Fourth cycle.
Restore       11
            0 0010   0 0010
            Remainder  Quotient
```

(ii) Non Restoring Division :-

Step 1 :- (Repeat N times)
→ If sign of A is '0', Shift A & Q left one bit position & subtract M from A; otherwise Shift A & Q left and Add M to A.
(Sign of A is 0 ⇒ Set q_0 to 1; otherwise '0'
Step 2 :- If Sign of A is '1', add M to A

Ex :-

```
Initially   0 0000   1000
              0011
Shift       0 0001   0 000□
Subtract    1 1101
Set q0      1 1110   0 0000   } 1st cycle
Shift       1 1100   0 00□□
Add         0 0011
Set q0      1 1111   0 0000   } 2nd cycle
Shift       1 1110   0 00□□
Add         0 0011
Set q0      0 0001   0 0001   } 3rd cycle
Shift       0 0010   0 01□□
Subtract    1 1101
Set q0      1 1111   0 0110   } 4th cycle.
            0 0011   Quotient
Remainder → 0 0010
```

Floating point Representation

* The binary point and the numbers are called floating points numbers.
* To accommodate very large integers and very small fractions.

→ The floating point Representation has 3 fields:-

* SIGN
* SIGNIFICANT DIGITS
* EXPONENT

Example:-

Let us Consider the number 111101.100110 to be Represented in floating point format

* First binary point is shifted to right of the MSB bit
* Next the number is multiplied by the Correct scaling factor to get same value.

Normalized form:-

111101.100110

$\downarrow$

1.11101100110×2^5 ∴ 2^5 → exponent, → Scaling factor

1.11101100110 → significant digits

Note: Base of scaling factors is fixed i.e, 2.

* The string of significant digits are known as mantissa.

In the above example,

SIGN = 0

MANTISSA = 11101100110

EXPONENT = 5

Note:- Bias value is added to the true exponent. This solves the problem of negative exponent

IEEE standard for floating point numbers :-

The standards for Representing floating point numbers in 32 bits & 64 bits have been developed by "IEEE"

Two types :-

* 32 bit standard Representation Single-precision Representation
* 64 bit standard Representation Double precision Representation

SINGLE precision Representation

1) occupies single 32 bit word

3 fields

SIGN → 1 bit

EXPONENT → 8-bits

MANTISSA → 23-bits

$E^1 = E + Bias$

Where E → scaling factor

Bias = 127.

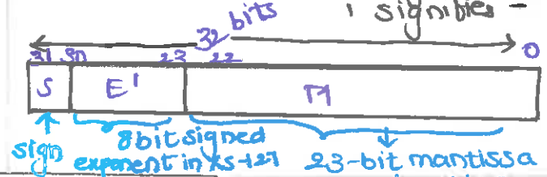
2) $E^1 = E + 127$ This representation is called excess -127 format

3) E^1 Ranges from 0 to 254.

4) Thus range of E^1 for normal values in single precision is "$0 < E^1 < 254$" i.e. $-126 \le E \le 127$

Value $= \pm 1.M \times 2^{E^1 - 127}$

∵ 0 signifies +
1 signifies −



DOUBLE Precision Representation

1) 64 bit standard representation
2) occupies two 32-bit words

3 fields

SIGN → 1 bit

EXPONENT → 11-bit

MANTISSA → 52 bit

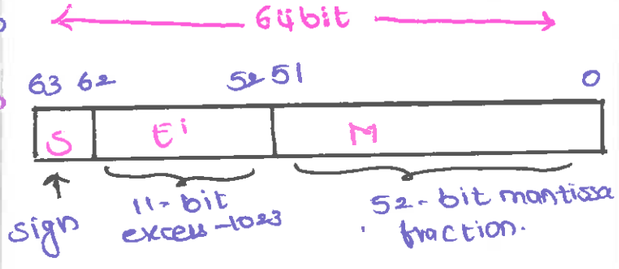
3) $E^1 = E + 1023$ here bias = 1023

Hence it is called excess-1023 format

4) The range of E^1 → to 2047.

i.e. $0 < E^1 < 2047$.

The actual exponent E is $-1022 \le E \le 1023$.



Two ways To Store Data:

Little Indian

Starting Address (A) → low order Bytes

Next Address (A+1) → high order Bytes

Big Indian

Starting Address (A) → high order Bytes

next Address (A+1) → low order Bytes

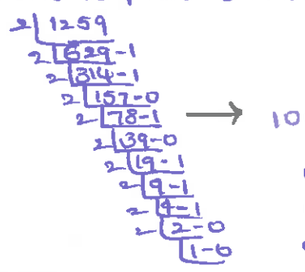
Example:1

Represent 1259.125_{10} in single precision and double precision formats

Step:1 Convert decimal number in binary format.

Integer Part : 1259_{10}

Fractional part: 0.125_{10}



→ 10011101011_2

$0.125 \times 2 = 0.25 \rightarrow 0$

$0.25 \times 2 = 0.50 \rightarrow 0$

$0.50 \times 2 = 1.00 \rightarrow 1$

$(1259)_{10} = (10011101011)_2$ $(0.125)_{10} = (0.001)_2$

Binary numbers : 10011101011 + 0.001

10011101011.001

Step: 2 Normalise the number

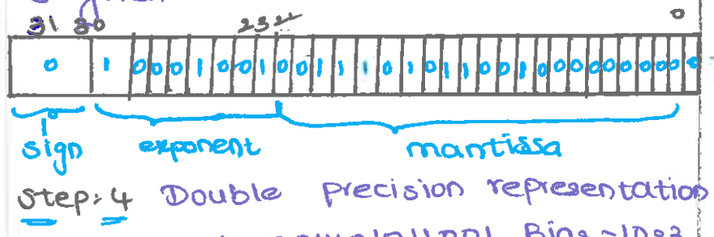
$10011101011.001 = 1.00111010111001 \times 2^{10}$

Step: 3 Single precision representation

S=0, E=10, M=00111010111001, Bias=127

$E^1 = E + 127 = 10 + 127 = 137_{10} = 10001001_2$

number in single precision format is given as.

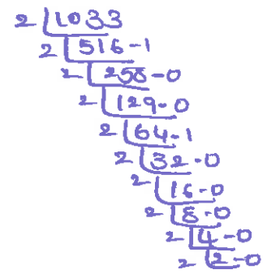


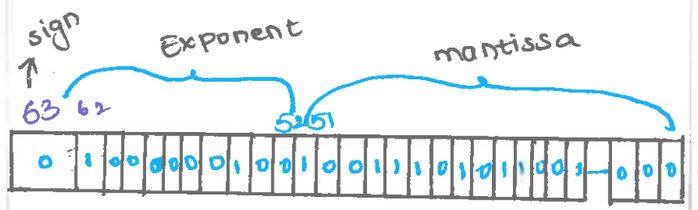
Step: 4 Double precision representation

S=0, E=10, M=00111010111001 Bias=1023

$E^1 = E + 1023 = 10 + 1023$ $E^1 = 1033_{10}$

$= 10000001001_2$





Floating point Operations [4]

* Addition
* Subtraction
* multiplication
* division

Floating point Operations :-

It consists of 4 types

Addition, Sub, multiplication, Div

Floating Point Arithmetic operation

Floating point Addition:

1) Add the following two decimal num in scientific notation: 0.70×10^{-1} with 9.95×10^{1}

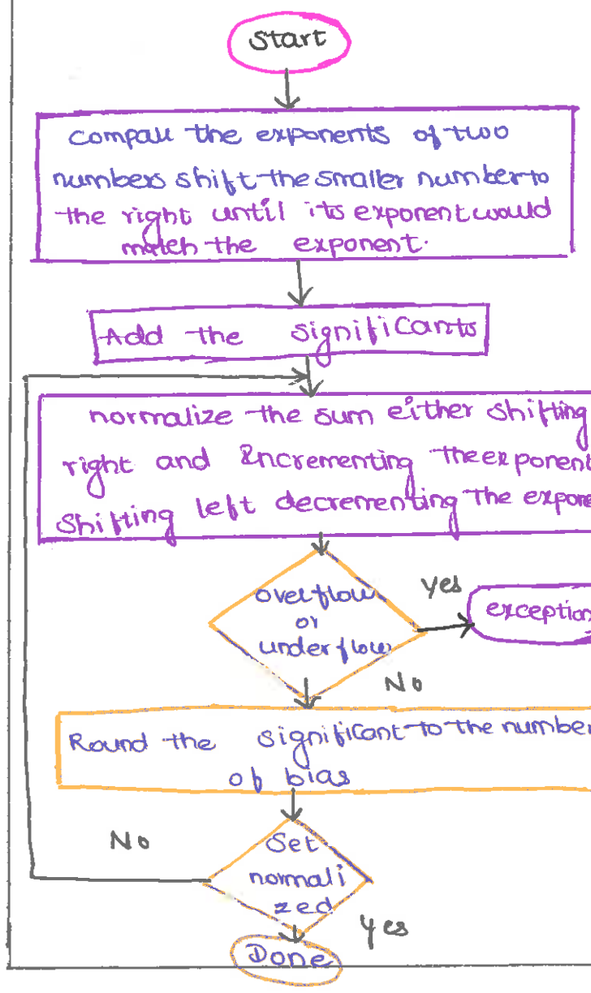
Step:1 Rewrite the smaller number such that its exponent matches with the exponent of larger number

$8.70\times10^{-1} = 0.087\times10^{1}$

Step:2 Add the mantissas

$9.95 + 0.087 = 10.037$

Sum $= 10.037\times10^{1}$



Step:3 put the result in normalised form

$10.037\times10^{1} = 1.0037\times10^{2}$ [shift mantissa, adj. exponent]

Check for overflow/underflow of the exponent after normalisation

Step:4 Round the result

If the mantissa does not fit in the space reserved for it, it has to be rounded off.

Eg: If only 4 digits are allowed for mantissa

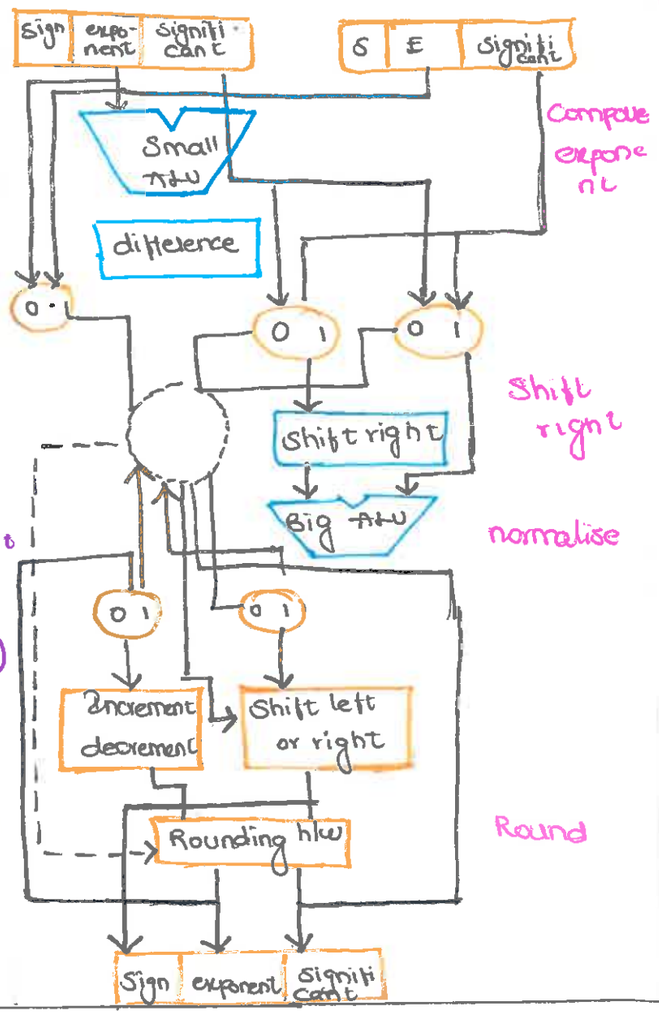
$1.0037\times10^{2} \rightarrow 1.004\times10^{2}$.

Example 1: Addition in binary:

Perform $0.5 + (-0.4375)$ $0.5 = 0.1\times2^{0} =$

1.000×2^{-1} [normalised]

Floating Point Addition/subtraction hardware:



$-0.4375: -0.0111\times2^{0} = -1.110\times2^{-2}$ [normalised]

Step:1 Rewrite the smaller num such that its exponent matches with the exponent matches with exponent of lauge number

$-1.110\times2^{-2} = -0.1110\times2^{-1} = 0.001\times2^{-1}$

Step:2 Add the mantissas:

$1.000\times2^{-1} + -0.110\times2^{-1} = 0.001\times2^{-1}$

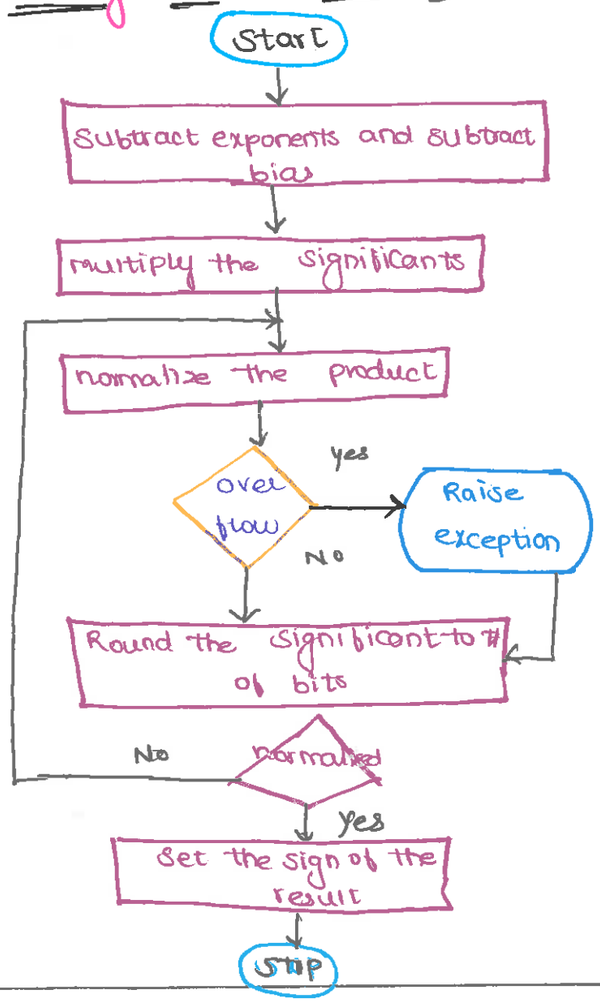
Step:3 normalise the sum, checking for overflow/underflow. $0.001\times2^{-1} = 1.000\times2^{-4}$

$-126 <= -4 <= 127 \Rightarrow$ no overflow or underflow.

Step:4 Round the sum, The sum fits in 4 bits so rounding i.e not required

Check: $1.000\times2^{-4} = 0.0625$ which is equal to $0.5 - 0.4375$ Correct!

Floating Point Subtraction:



Step:1 Subtract the following two decimal numbers in scientific notation

8.70×10^{-1} with $9.95\times10^{+1}$

Step:2 Rewrite the smaller number such that its exponent matches with the exponent of lauger number

$-1.110\times2^{-2} = -0.1110\times2^{-1}$

Step:3 normalise the sum, checking for overflow/underflow

$0.001\times2^{-1} = 1.000\times2^{-4}$

a) Sub the mantissas

$9.95 - 0.087 = 9.863\times10^{1}$

$= 9.863\times10^{1}$

Step4 Round the result

9.863×10^{1}.

Floating point multiplication

Step:1 Add exponents & subtract bias

Step:2 multiply the significants

Step:3 Normalize the product

Step:4 overflow if yes, raise exception

Step:5 Round the significant to appropriate # of bits

Step:6 If not still normalized to back to step 3

Step:7 Set the sign of the result

Example-1

multiply the following two numbers in scientific notation

$1.110\times10^{10}\times9.200\times10^{-5}$

FLOATING POINT MULTIPLICATION

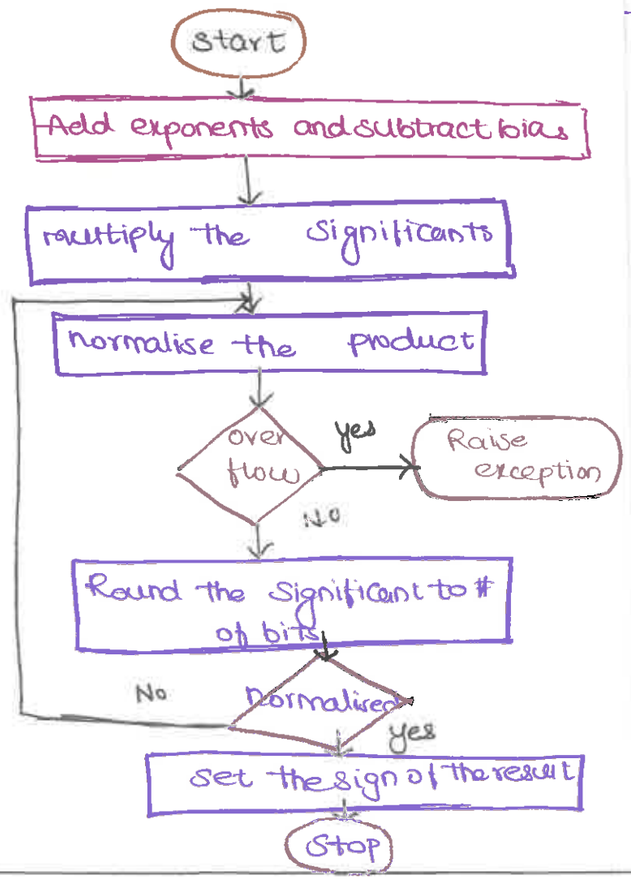
1. Add the exponents to find new exponent = 10+(-5) = 5
If we add biased exponents, bias will be added twice Therefore we need to subtract it once to compensate: (10+127) + (-5+127) = 259
259-127 = 132 which is (5+127).
bias new exponent

2. Multiply the mantissas:
$1.110 \times 9.200 = 10.212000$

Can only keep three digits to the right of the decimal point so the result is 10.212×10^5

3. Normalise the result
1.0212×10^6

4. Round it 1.021×10^6



Example: 2 Multiplication in binary

$1.000 \times 2^{-1} \times -1.110 \times 2^{-2}$

1. Add the exponents
$(-1)+(-2) = -3$
$-3+127 = 124$ which is within the limit)

2. Multiply the mantissas

```
 1.000
 1.110 x
-------
 0000
 1000
 1000
 1000
-------------
 1.110000 -> 1.110000
```

The product is 1.110000×2^{-3} need to keep it to 4 bits 1.110×2^{-3}

3. Normalise (already normalised)
At this step check for overflow/ underflow by making sure that
$-126 \leq$ exponent ≤ 127
$1 \leq$ Based exponent ≤ 254

4. Round the result 1.110×2^{-3}

5. Adjust the sign.
Since the original signs are different the result will be negative
-1.110×2^{-3}

Floating Point Division

Step:1 Subtract the exponents and add bias

Step:2 Divide the significants

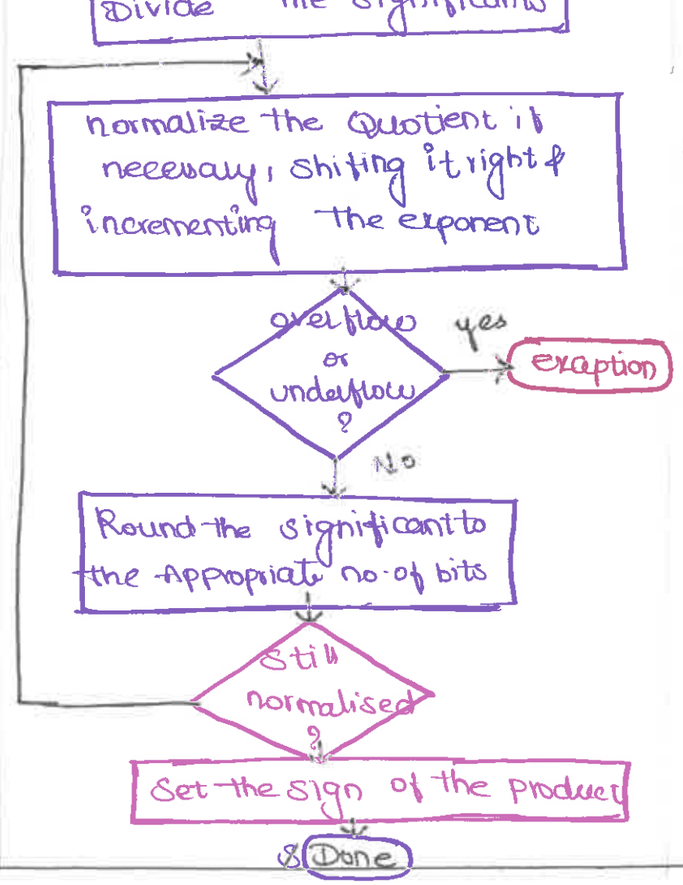
Step:3 Normalise the Quotient

Step:4 If there is a overflow, raise exception

Step:5 Round the significant to the Appropriate number of bits.

Step:6 If not still normalised go back to step 3

Step:7 Set the sign of the result



Example:

Divide the following two numbers in scientific notation
$1.110 \times 10^{10} \div 9.200 \times 10^{-5}$

Step:1 Subtract the exponent to find the new exponent
$10-(-5) = 10+5 = 15$

Step:2 Divide the significants
$1.110 \div 9.200$
$= 0.120652173$

Step:3 Normalise the result
$0.120652173 \times 10^{15}$
$\Rightarrow 1.206\,52173 \times 10^{14}$

Step 4: Round it
1.2065×10^{14}

Fundamental Concepts of Processing

* A Unit which executes Machine instructions and Coordinates the activities of other units is Called Instruction set Processor (ISP)

Steps Involved in Instruction Execution:

1. Fetch the Contents of Memory location Pointed to by the Pc.

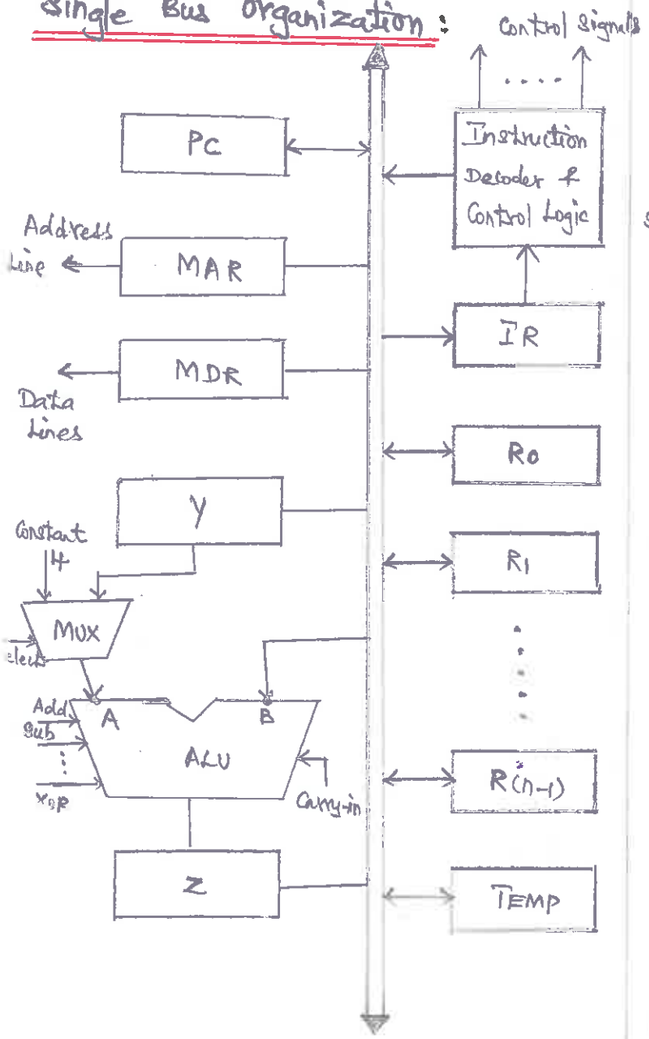
   IR ← [[Pc]]

2. Increment the Value of Pc by 4.

   Pc ← [Pc] + 4

3. Carryout the action acording to IR.

Single Bus organization:



1. Register Transfers

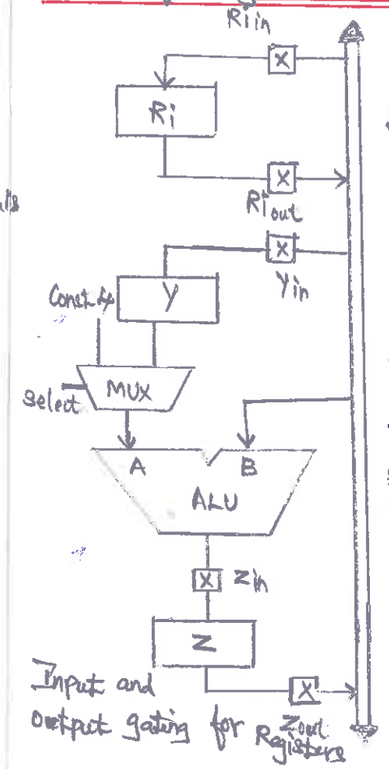
* For each register 2 Control signals are used
* Ex: R1 → R1in, R1out.
  MAR → MARin, MARout.
* Example for Register Transfer: 1- open 0- close

  Move R1, R4

* Equivalent Control signals are

  $R1_{out}^{=1}$, $R4_{in}^{=1}$

2. Performing an ALU operation



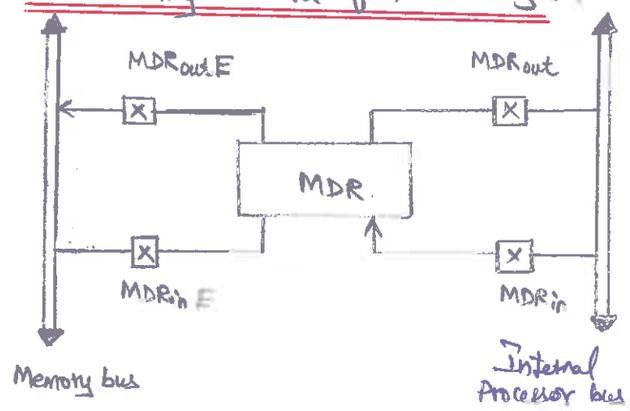
Example:

Add R1, R2, R3

Control Sequence:

1. R1out, Yin
2. R2out, SelectY, Add, Zin
3. Zout, R3in

3. Fetching a Word from Memory:



Example: Move (R1), R2

Control signals:
1. R1out, MARin, Read
2. MDRinE, WMFC
3. MDRout, R2in

4. Storing a Word in Memory:

Example: Move R2, (R1).

Control signals:
1. R1out, MARin
2. R2out, MDRin, Write
3. MDRoutE, WMFC

Execution of a Complete Instruction:

Example: Add (R3), R1

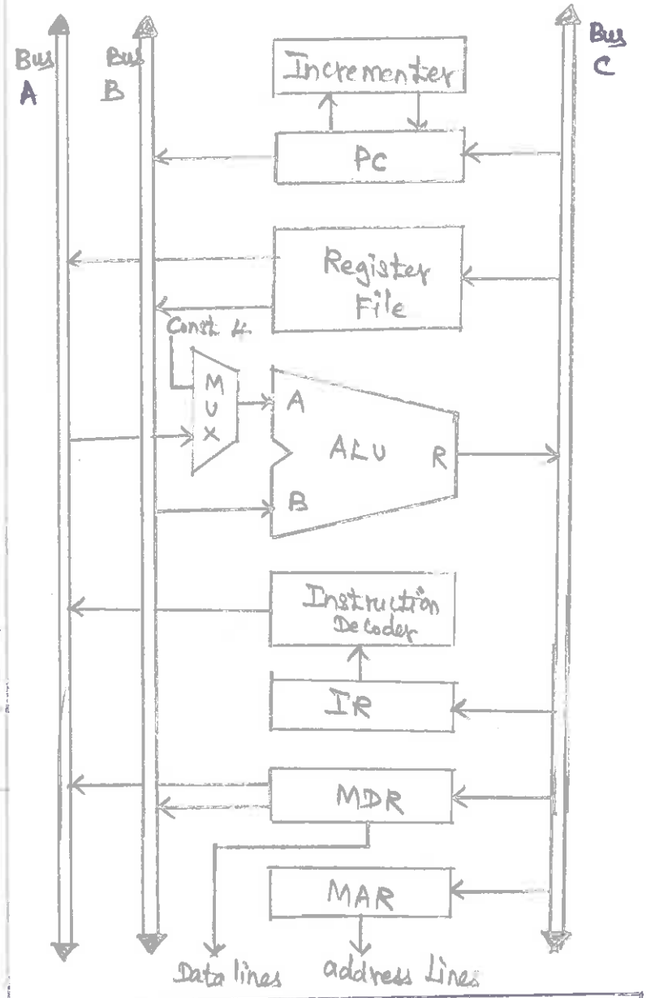
Steps involved:
1. Fetch the instruction
2. Fetch the first operand
3. Perform the addition
4. Load the result into R1.

Control Sequence for instruction Add (R3), R1

Step	Action
1.	Pcout, MARin, Read, Select4, Add, Zin
2.	Zout, Pcin, Yin, WMFC
3.	MDRout, IRin
4.	R3out, MARin, Read,
5.	R1out, Yin, WMFC
6.	MDRout, SelectY, Add, Zin
7.	Zout, R1in, End.

Multiple Bus Organization

* One bus — One data transfer.
* Multiple bus — Multiple data transfer act Same time.
* Register File — All general-Purpose registers are Combined into a Single block.

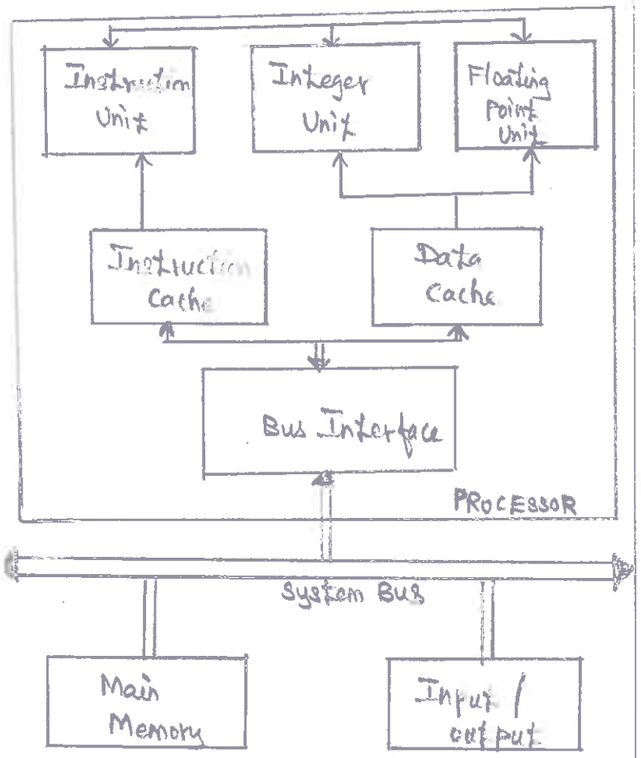


Control Sequence for Instruction- Add R4, R5, R6

Step	Action
1	PCout, R=B, MARin, Read, Inc Pc
2.	WMFC
3	MDRoutB, R=B, IRin
4	R4outA, R5outB, SelectA, Add, R6in, End.

PROCESSOR ARCHITECTURE

* It has instruction unit which fetches instructions from main memory.
* Has Separate Processing units to deal with integer data & floating-Point data.
* Data Cache & instruction Cache is inserted between these units & main memory.
* Processor is Connected to the System bus.
* Can include several units to increase the potential for Concurrent operations.

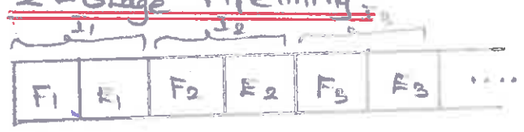


* Most of the modern Processors use Separate Caches for instruction and data.
* Bus interface used to Connect the Processor with external environment.

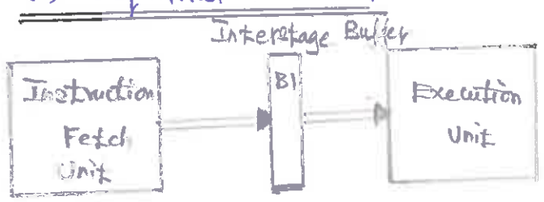
Basic Concepts of Pipelining:

* One way to improve Performance is to use faster circuit technology to build Processor.
* Another way – to arrange the hardware, so that more than one operation Can be performed.
* Pipelining – Effective way of organizing Concurrent activity in a Computer System.

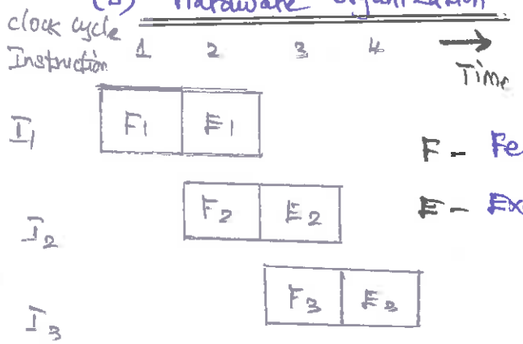
2-Stage Pipelining:



(a) Sequential Execution

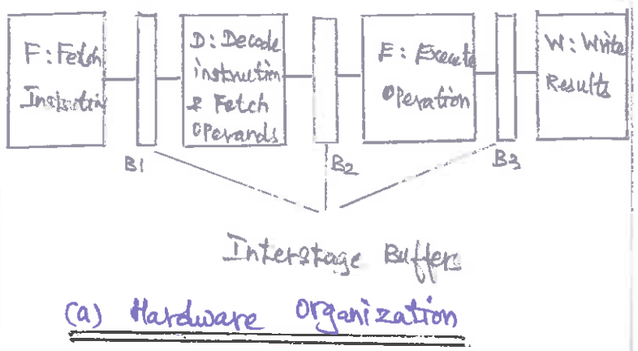


(b) Hardware organization

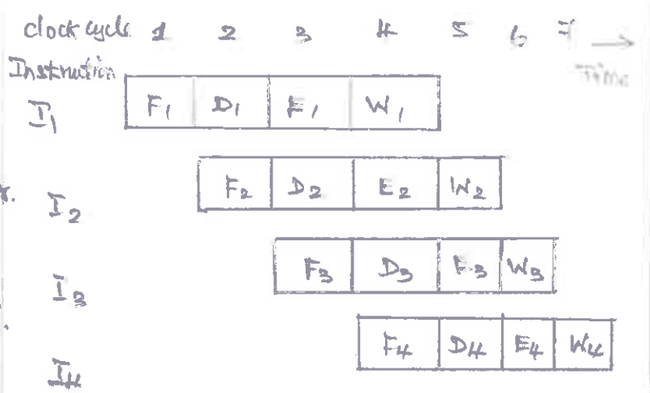


(c) Pipelined Execution

4-Stage Pipelining



(a) Hardware Organization



(b) 4-stage Pipelined Execution

Pipeline Performance:

* Pipeline Performance is directly Proportional to the number of stages.

Hazard:

* For a Variety of reasons, one of the Pipeline stages may not be able to Complete its Processing tasks in the time allotted.

Types of Hazards:

1. Data Hazard
2. Instruction Hazard
3. Structural Hazard.

Structural Hazard:

* It will occur when two instructions require the use of a given hardware resource at the same time.

Example – 1. Access to memory
2. Printing multiple documents using a shared Printer.

Data Hazard:

* It is a situation in which the Pipeline is stalled because the data to be operated on are delayed for some reason.

Example: 1

A ← 3 + A
B ← 4 × A

Example: 2

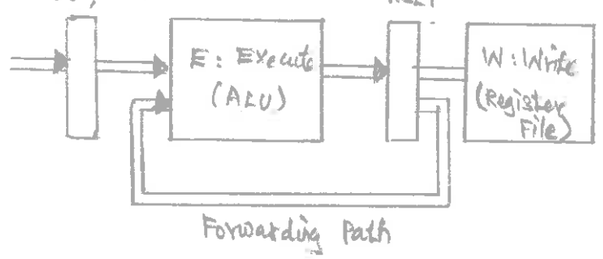
A ← 5 × C
B ← 20 + C

Example: 3

Mul R2, R3, R4
Add R5, R4, R6.

Operand Forwarding Technique

* Delay can be reduced, or possibly eliminate if we arrange for the result of one instruction to be forwarded directly for use in Step E2 of 2nd instruction.



Handling Hazards using Software

* Compiler automatically introduces NOP (No operation) when it detects a dependency between instructions.

Side Effects:

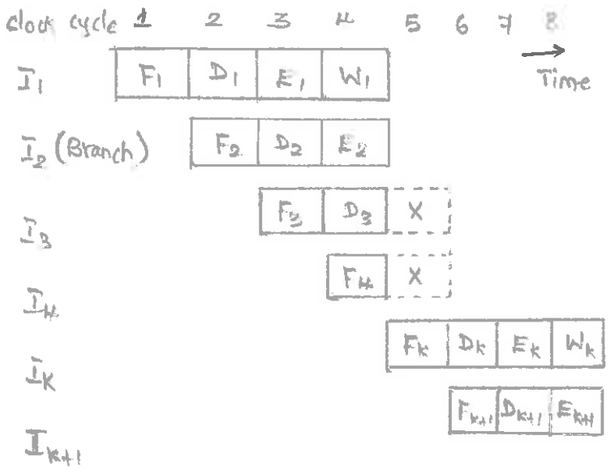
* When a location other than one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect.

INSTRUCTION HAZARDS

* Occurs if instruction Stream is interrupted.
* Branch instruction may also Cause the pipeline to stall.

Unconditional Branches:

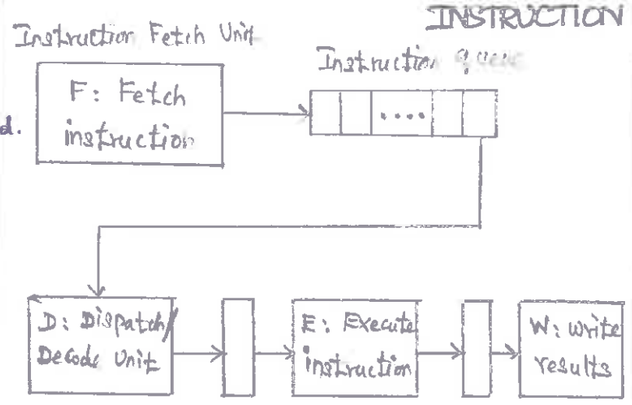
* Time lost as a result of branch instruction is known as branch Penalty.
* Reducing the branch Penalty requires the branch address to be computed earlier.
* IF Unit has dedicated hardware to identify branch instruction & Compute target Address.



Branch address Computed in Execute stage

Instruction Queue and Prefetching Technique:

* Instruction queue can store several instructions.
* Dispatch unit takes instructions from the front of the queue and Sends them to the execution unit.
* IF unit keeps the instruction queue filled at all times to reduce delay.



* **Branch Folding**: IF will execute the branch instruction Concurrently with the execution of other instructions.
* Instruction queue helps in dealing with Cache misses.

Conditional Branches & Prediction:

* Introduces added hazard Caused by the branch Condition & evaluation.
* Branch instructions occur frequently (20%), which increases branch Penalty.

1. Delayed Branching Technique:

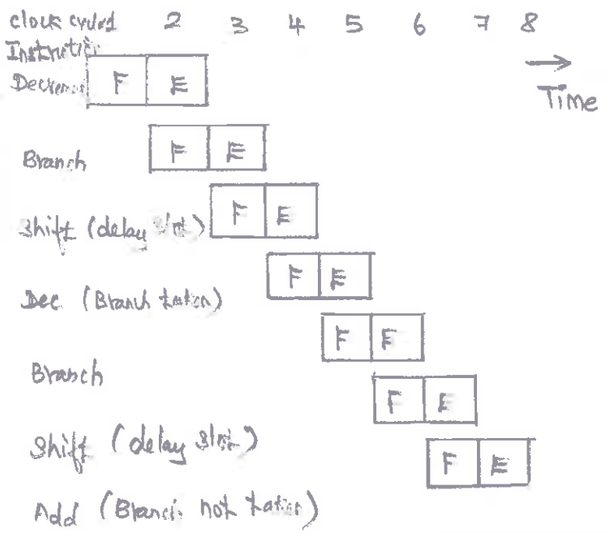
* Location following a branch instruction is Called a branch delay slot.
* Delayed branching Can minimize the Penalty occurs due to Conditional branches.
* Instructions in the delay slots are always fetched. Allow them to be fully executed whether or not the branch is taken.

Loop	shift_left	R1
	Decrement	R2
	Branch ≠ 0	Loop
Next	Add	R1, R3

(a) Original Program Loop

INSTRUCTION HAZARDS (b) Reordered Instructions

Loop	Decrement	R2
	Branch ≠ 0	Loop
	Shift_left	R1
Next	Add	R1, R3



* Effectiveness of delayed branching depends on how instructions are reordered.

2. Branch Prediction Technique:

* Predict whether or not a Particular branch will be taken.
* Simplest form of branch Prediction is to assume that branch will not take Place & Continue to fetch instructions in order.
* Speculative Execution — Instructions are executed before the Processor is certain that they are in Correct execution Sequence.

A. Static Branch Prediction:

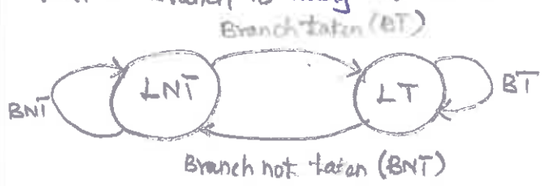
* Branch Prediction decision is always the same every time a given instruction is executed.

B. Dynamic Branch Prediction:

* Prediction decision may change depending on execution history.
* Objective: To reduce the Probability of making a wrong decision.

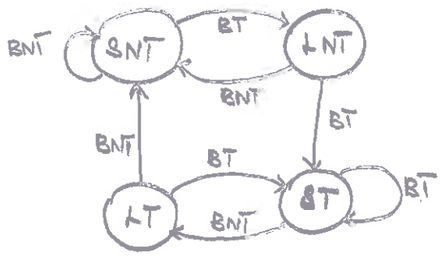
B1 - 2 State Algorithm

LT - Branch is likely to be taken
LNT - Branch is likely not to be taken.



B2 - 4 State Algorithm

ST - Strongly likely to be taken
LT - Likely to be taken
LNT - Likely not to be taken
SNT - Strongly likely not to be taken.



Influence on Instruction Sets:

* Two key aspects of Machine Instructions.

1. Addressing Modes:

* Must Consider the effect of addressing modes on instruction flow in the pipeline.

2. Condition Codes:

* Either set or cleared by many instructions.
* Provides flexibility in reordering instructions.
* Effectively utilized by Compiler.

SUPERSCALAR PROCESSING

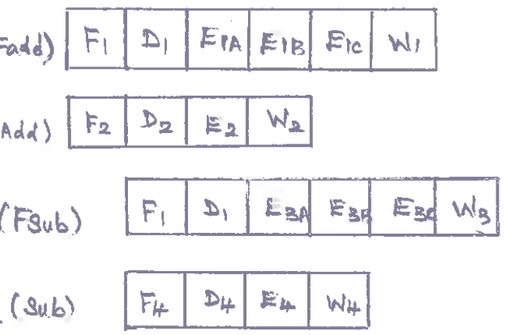
* Equip the Processor with multiple Processing units to execute in Parallel.
* **Multiple Issue**: Several Instructions start execution in the same clock cycle.
* **Superscalar Processor**: Instruction execution throughput more than one instruction per cycle.
* Many modern high-Performance Processors use this approach.
* Instruction queue should be filled always
* Processor should be able to fetch more than one instruction at a time.

Instruction Fetch Unit – Capable of reading 2 instructions & store in IQ.

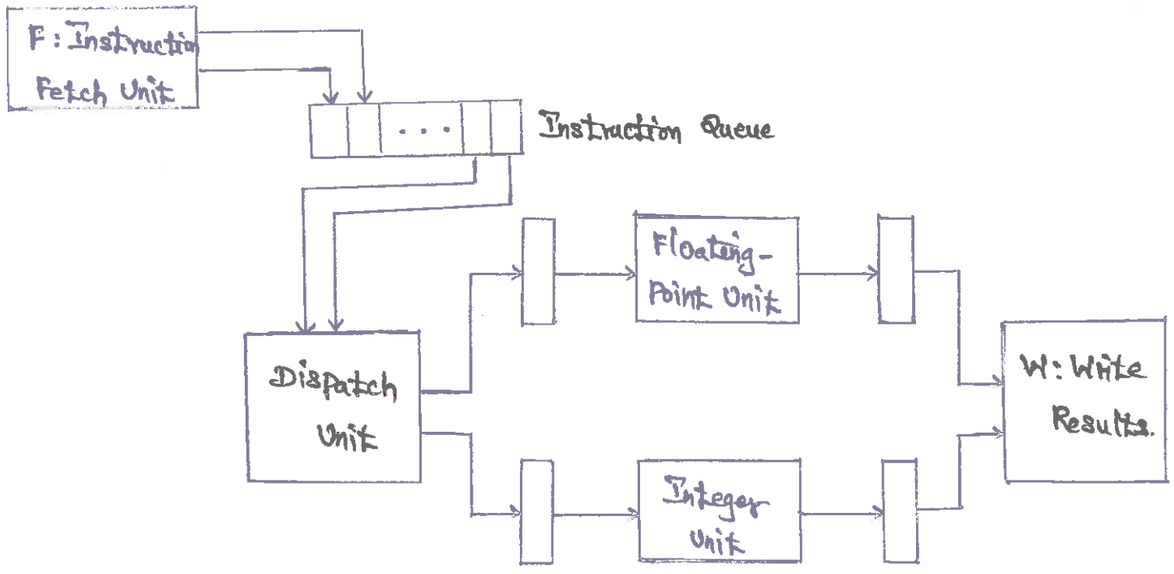
Dispatch Unit – Retrieves and decodes upto 2 instructions from the front of the Queue.

* No Hazards if more Processing units are there.

clock cycle 1 2 3 4 5 6 7 → Time



* **Instruction Execution Flow in the Processor**
* Compiler can avoid many Hazards through Judicious selection & ordering of Instructions

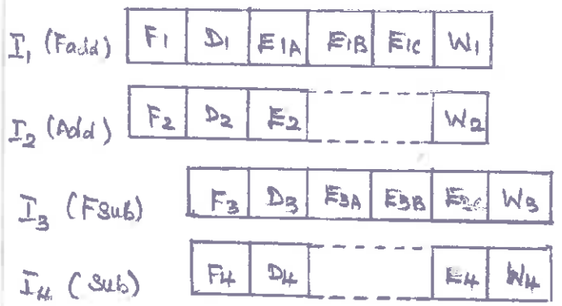


A Processor with Two execution Units

Out-of-order Execution:

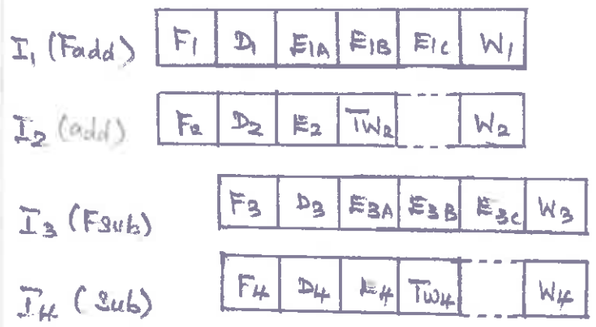
* Instruction execution is done out of order.
* Problems / Side effects ?.
* Independent instructions – No Problem.
* Dependent instructions – Creates Hazards.
* Exceptions will arise < Imprecise Exception / Precise Exception

clock cycle 1 2 3 4 5 6 7 → Time



(a) Delayed Write

clock cycle 1 2 3 4 5 6 7 → Time



(b) Using Temporary Registers

* To guarantee Consistency – Results of instructions must be written in Program order.
* By using Delayed write technique (or) by using temporary Registers, we can execute out-of-order & update results in-order.

Execution Completion:

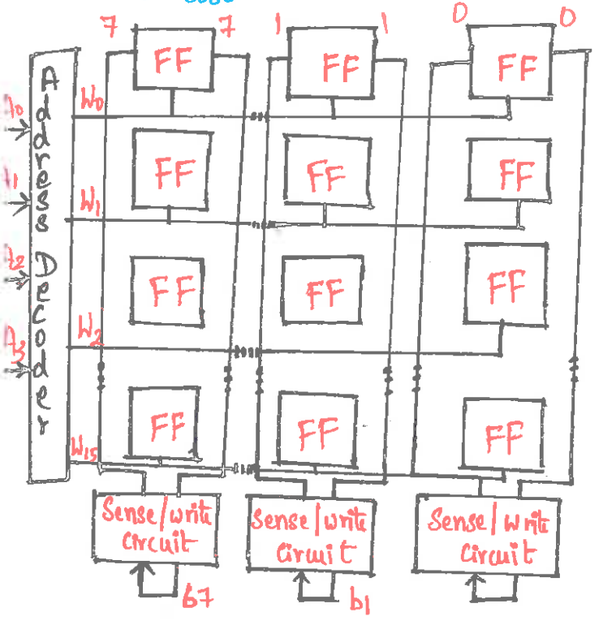
* Instructions are executed out-of-order to make effective use of Processing units.
* But instructions must be Completed in Program order to allow Precise exception.
* Commitment step – Usage of Tag.
* **Commitment Unit**:
  * Special control unit to guarantee in-order Commitment.
* **Reorder Buffer**:
  It is a Queue to determine which instructions should be commited next.

Dispatch operations:

* Will ensure whether all the resources needed for the execution of an instruction are available.
* Should instructions be dispatched out-of-order ?.
* If allowed – deadlock will arise.
* Issuing instructions out-of-order → increases the complexity of dispatch unit.
* Most Processors use only in-order-dispatching
* **Conclusion** – Superscalar Processing increases the throughput of the execution of instructions which in turn increases the speed of the Computer

RANDOM ACCESS MEMORY:

* Accessed for a Read or Write operation.
* Consists of memory cells.
* Semi-Conductor Materials (VLSI)
* More - reliable.
* Works faster.
* More Cost.



Internal Organization of Memory chips:

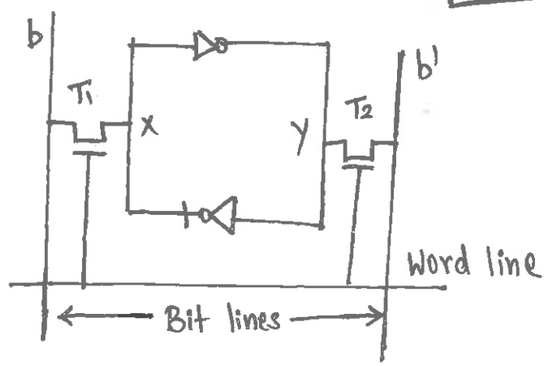
* Memory cells organized as arrays.
* Each cell stores one-bit information.
* Row - Word Line
* Column - bit line
* Sense/write circuit - Sense/read data
* Control signals - R/W, Cs.

Types Of RAM

1) Static RAM (SRAM)

* Retains their state as long as power is applied.
* Two inverters are cross-Connected - Latch.

MEMORY



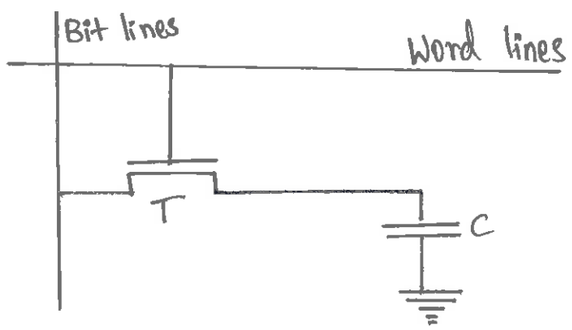
* Read operations.
* Write operations.

Advantages of SRAM:

* Consumes very low Power.
* Refreshing Not required.
* Can be accessed very quickly.
* 8-16 times faster and costlier than DRAM.

2) Dynamic RAM (DRAM)

* Do not retain their state indefinitely.
* Less Expensive - Simpler cells are used.
* Information stored - charge on Capacitor.
* Periodic refreshing is required.
* Problem - Capacitor charge Leakage.
* Read operation.
* Write operation.



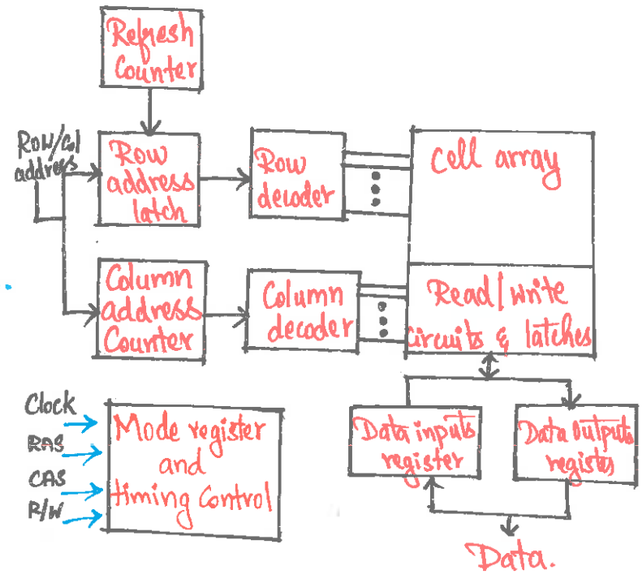
Types of DRAMS:

1) Asynchronous DRAM:

* operations not Synchronized with clock.
* Special memory Controller - RAS, CAS.
* Need refresh circuit.
* High density and low Cost.

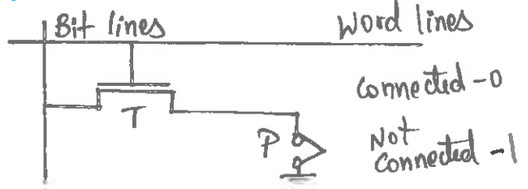
2) Synchronous DRAM:

* operations directly Synchronized with clock.
* Has built-in refresh Circuit.
* Different modes of operation.



READ-ONLY MEMORY (ROM)

* Retain information even it is switched off.
* Special writing process is needed.
* Used to "boot" the system.



Types of ROM

1) ROM:

* WORM device - Write Once Read Many.
* Data is Written by Manufactuer.
* Not reversable - break the fuse once.

2) PROM: Programmable ROM

* Data can be loaded by the User.
* Initially empty - All cells Contains '0'
* Break the fuse as per requirement
* Requires high-Current pulse

3) EPROM: Erasable Programmable ROM

* Stored data can be Erased.
* New data Can be loaded/written.
* For erasing - expose to UV light.
* physically removed from the circuit.

4) EEPROM: Electrically Erasable PROM.

* Electrically Erasable
* Need different voltages for erasing, writing and reading stored data.
* No need to remove physically.
* Selective Contents can be erased
* More flexible and costly.

CACHE MEMORY :-

- Speed of processors are high when compared with speed of main memory.
- The processor speed is the good performance parameter, so reduce waiting time by main memory
- Cache memory is the efficient solution

EFFECTIVENESS OF CACHE MECHANISM

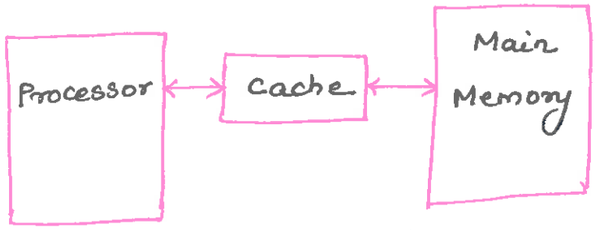
* Locality of reference (LOR) property of computer programs.
* Many instructions in localized areas of program executed repeatedly during time period.
* Remainder of the program accessed infrequently.

* Types of LOR
1. Temporal LOR
2. Spatial LOR.

* Temporal - Recently executed instruction is likely to be executed again soon.
* Spatial - Instructions in Close proximity to recently executed Instruction will be executed soon.
* By placing active segments of program in fast cache memory, then total execution time is reduced.

OPERATION OF A CACHE MEMORY:

* The memory control circuit is designed to use LOR property.
* Temporal LOR - When instruction/ data needed, bring from Cache & Hopefully remain.
* Spatial LOR - Fetching several items reside at adjacent addresses (Blocks of data)
* Cache block / Cache line.



→ Read request made by processor
→ Contents of a memory block with a specified location transferred to cache one word at a time
→ Program references this block's location, then it is read from cache.
→ cache memory stores less number of blocks.
→ correspondence between main memory blocks and cache - mapping function.
→ Cache is full, memory word not in cache referenced, cache control hardware decide which block to be removed from cache. (replacement Algorithm) → Least Recently used (LRU)
→ Processor issues Read/write requests → If it is available in cache read hit / Write hit.

READ / WRITE IN CACHE:

* Main Memory not involved in Read.
* Two methods in Write

① Write-Through Protocol
- Cache and Main Memory updated simultaneously.

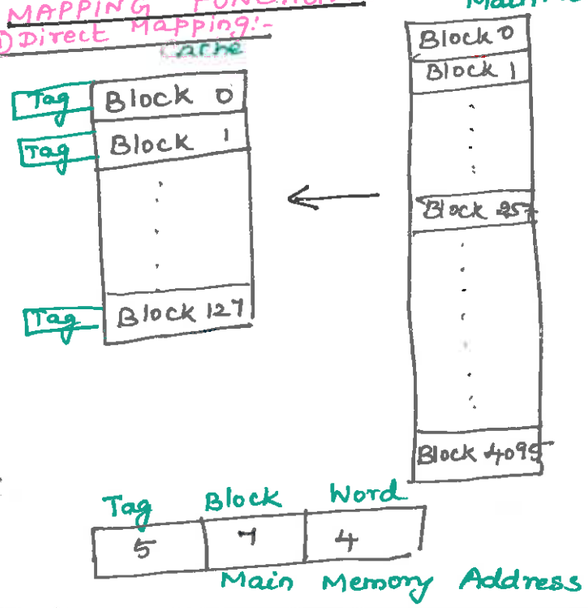
② Write-back / copy-back protocol:
- update only in cache
- Mark as updated using dirty or modified bit.
- Later updated in Main memory

READ MISS / WRITE MISS:-

(Read Miss)
* request word not in cache (read)
* Block of words in requested word copied to cache.
① * Particular word sent to processor.
② Particular word sent to processor immediately read from Main memory (Load through / Early restart)
* If requested word not in cache during write - Write Miss
* Writeback / Write-Through is used

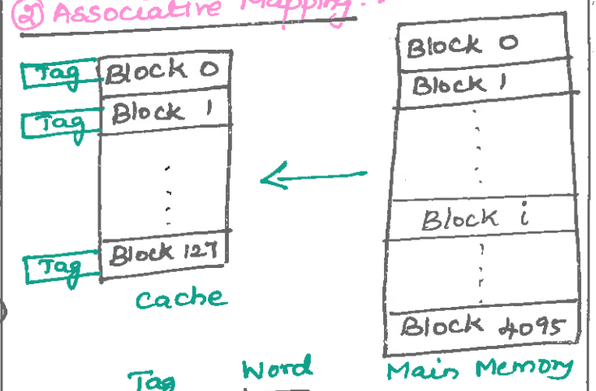
MAPPING FUNCTIONS:-

① Direct Mapping:-



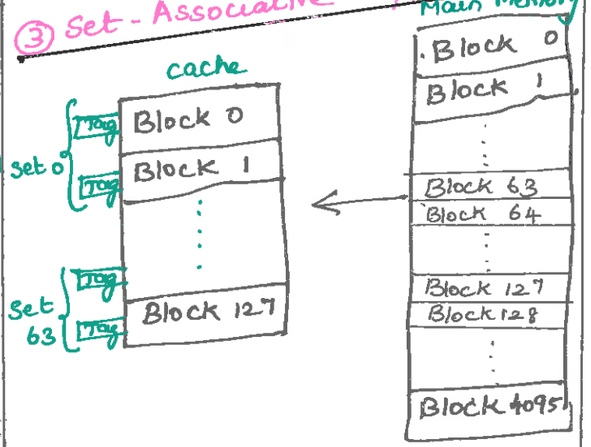
* consider cache has 128 blocks main memory 4096 blocks.
cache memory value = block j % 128
* 0, 128, 256 - block 0 of cache
* 1, 129, 257 - block 1 of cache.

② Associative Mapping:-



* Tag bits
* cost is higher than direct mapped
* Associative search method used

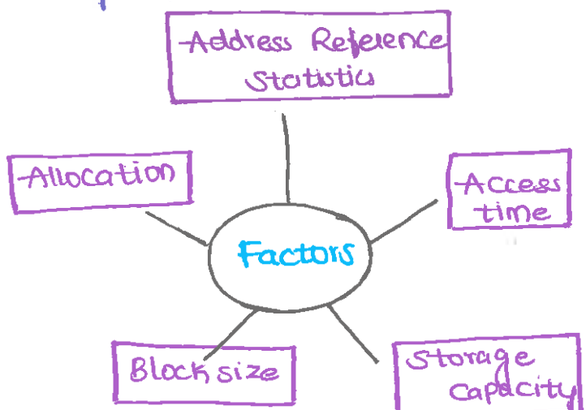
③ Set-Associative Mapping:-



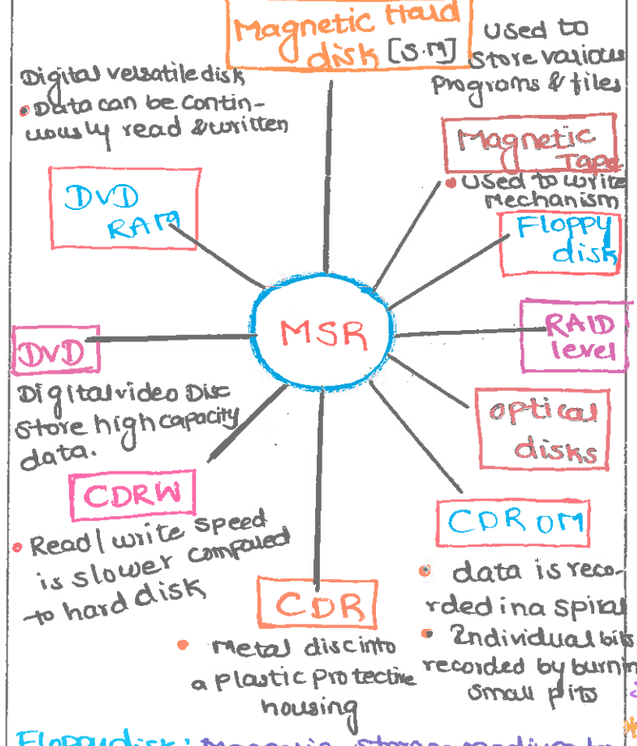
* Combination of Direct + Associative
* Simple method
* valid bit.

Performance Measures:

*memory peformance captured by two parameters, latency and bandwidth.
*latency is the time from the issue of a memory request to the time the data is available at the processor.
* Bandwidth is the rate at which data can be pumped to the processor by the memory system.

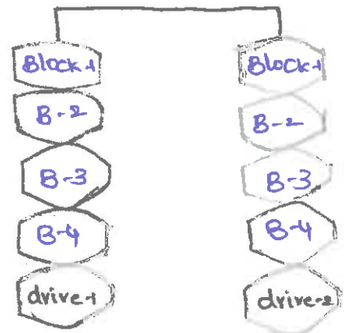


Magnetic Surface Recording



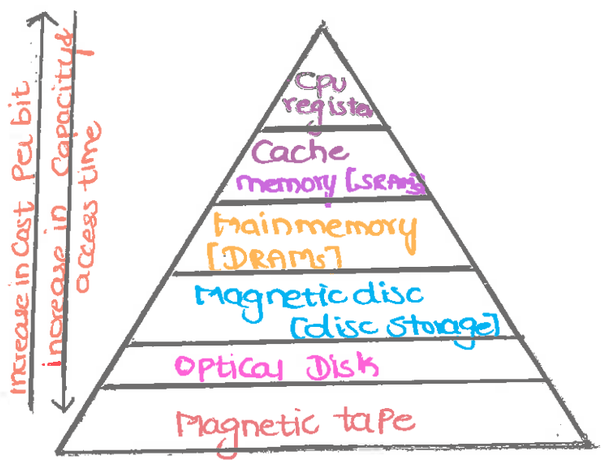
Floppy disk: Magnetic storage medium for computer systems.
Composed of a thin, flexible magnetic disk sealed in a square plastic carrier

RAID level: Redundant array of independent disks
Used to protect data in the case of a drive failure RAID 1



Optical Disks: A storage medium from which data is read and to which it is written by lasers.

MULTI LEVEL MEMORY (or) MEMORY HIERARCHY



MEMORY Hierarchy Design

*Memory with different speeds & sizes
*Faster memories are expensive
*Distances from processor increases
*Size and access time also increases

Memory Allocation

A simple way to allocate memory

First fit: The process is allocated the first available memory space.
Best fit: The process is allocated the smallest memory block that is large enough for its purposes.
Worst fit: The process is allocated the largest memory block that is available.

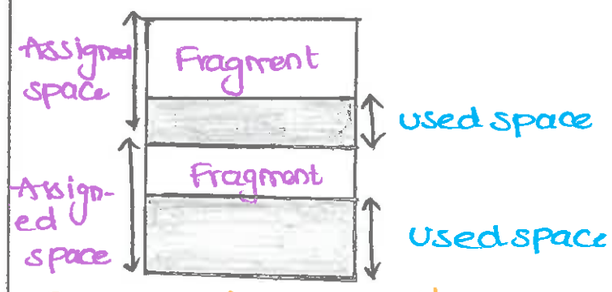
• Replacement policy
FIFO : first in, first-out
LRU : least recently used.

External fragmentation

* Dynamic memory allocation to allocate some memory
* leave a small amount of unusable memory
* memory available is reduced substantially

Internal Fragmentation

• memory is distributed into fixed sized blocks



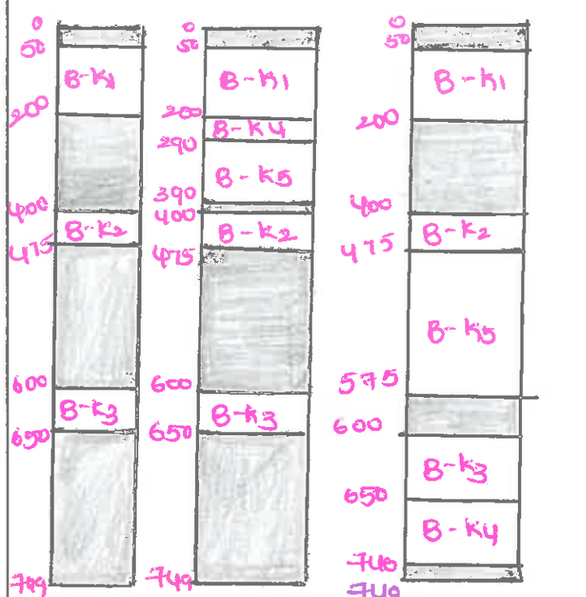
Internal fragmentation

Types:

Non pre-emptive allocation

* memory allocation is necessary to find or create an available region.

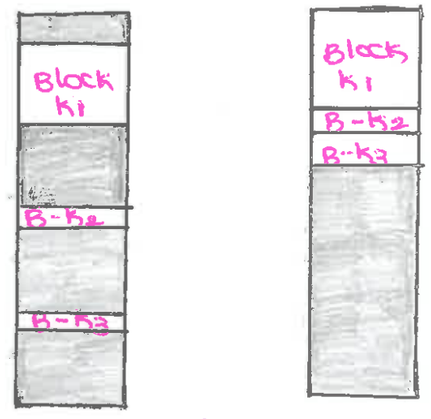
Two algorithms are first fit & Best fit



a) Intial memory state
b) Allocation of K4 & K5 by Hotfit
c) Allocation of K4 & K5 by bestfit

Pre-emptive allocation

* Re-allocation of the blocks by Compaction.
* memory are compressed into a single contagious group.
* creates on available region of maximum size



a) memory allocation before compaction
b) memory allocation after Compaction

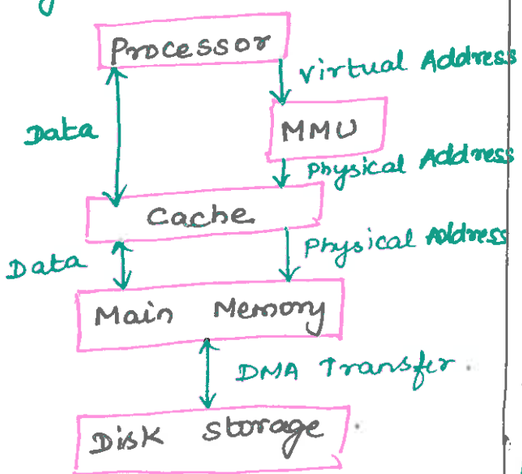
VIRTUAL MEMORY

VIRTUAL MEMORIES:-

* Main Memory is small compared to processor.
* When program does not completely fit into main memory, not executed parts stored in secondary storage devices.
* When new segment of program coming inside main memory, need to replace existing content
+ Technique to more program & data into physical main memory is virtual Memory.
* Binary addresses generated by processor is virtual/logical address.

Virtual Address —(CHW/SW)→ Physical Address

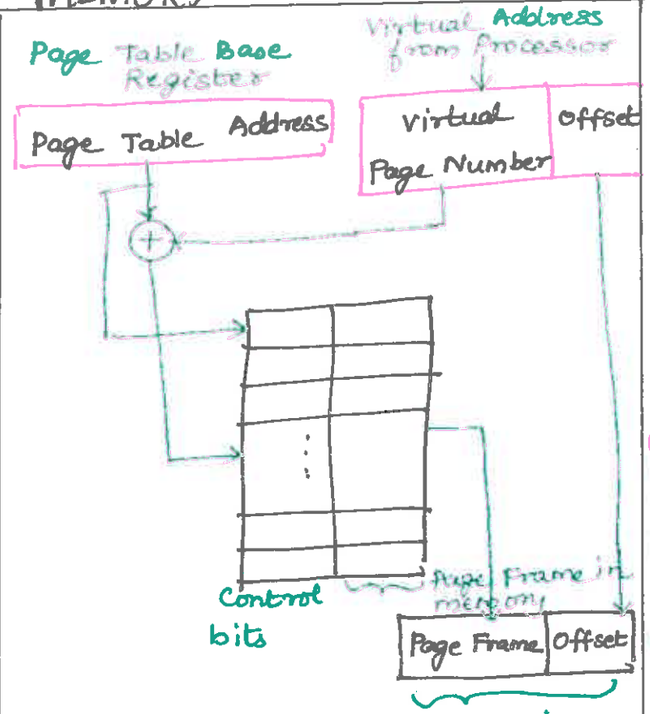
* If virtual address refers to program, currently available in physical memory, accessed immediately or it should be brought into main memory.



* Logical address is converted to Physical address by combination of software & hardware
* While accessing main memory, if data available, it is hit.
* A special hardware Memory Management Unit (MMU) translates Virtual to Physical address.
* If data available, MMU tell OS to bring data into memory from disk. This is performed by Direct Memory Access (DMA)

ADDRESS TRANSLATION:

* All programs and data are composed of fixed length units → pages (Blocks of words) placed continuously in main memory
* Pages should not be too small or too large.
+ cache bridges gap of speed between the processor and main memory by hardware.
+ Virtual Memory bridges the speed gap between main memory & secondary memory by Software.
* Virtual address generated consists of Virtual Page Number, Offset
* Information about main memory location of each page is in Page Table.
* Page Table information are used by MMU. So it should be in MMU only. MMU is implemented as processor chip along with primary cache so it is kept in main memory.



CONTROL BITS:

* Describes the status of the page in main memory.
* Describes validity of a page

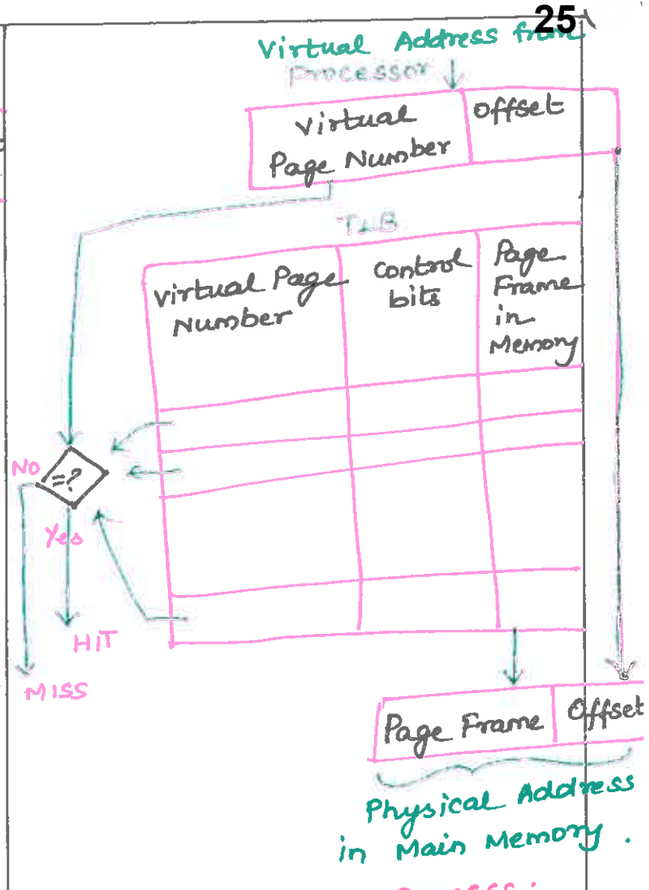
MODIFICATION BIT:

+ Convey whether the page is modified or not

RESTRICTION BITS:

+ Describes the access permission whether full read/write

+ copy of small version of page table placed in MMU.
* This access the most recently accessed pages.
+ It is implemented by a small cache called Translation look Aside buffer (TLB)



ADDRESS TRANSLATION PROCESS:

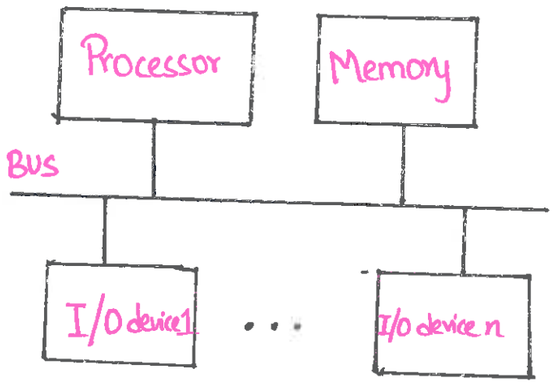
① Given the virtual address, MMU looks TLB for referenced page, If it is available, physical address got immediately.
② If miss in TLB, get content from page Table & update TLB.
③ When program request a page not in main memory, page fault occurs, whole page has to be brought from disk to main memory.
④ MMU asks OS to raise exception, OS copies requested page and place in main memory
⑤ To bring new page, if main memory is full, one of the resident page has to be replaced by LRU (Least Recently used)
⑥ Modified pages has to be updated before removing it from main memory by write-through protocol.

Accessing I/O devices.

- Connect I/O devices
- Three sets of lines

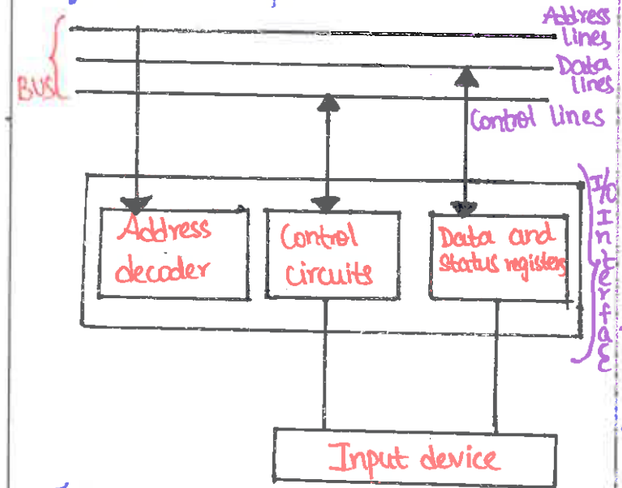
a) Carry address,
b) Data,
c) Control signals

- Requested read or a write operation
- Requested data are transffered over the data lines.



I/O Interface.

Required to coordinate I/O transfers.



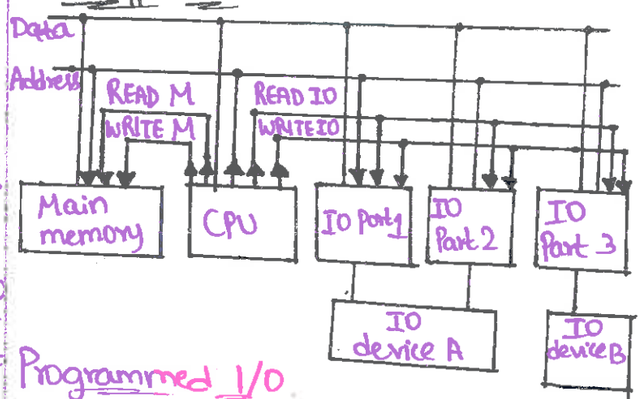
Two methods are used to address the device:

1. Memory-mapped I/O
2. I/O Mapped I/O.

Memory Mapped I/O.

- Portions of address space are assigned to I/O devices -- reads and writes to those addresses are interpreted as commands to the I/O devices.
- I/O devices and the memory share the same address space, - any machine instruction that can access memory.
- To transfer data to or from an I/O device.
- Memory load and store instuctions are used
- Transfer data words to or from I/O ports.

I/O Mapped I/O.



Programmed I/O

- Processor repeatedly checks a status flag
- To acheive the required synchronization.
- Between the processor and an I/O device.
- Dedicated instruction is used
- To give a command to an I/O device
- Specifies both the device number and the command word.

Interrupts

- Signal generated by an external device requiring some urgent action to be taken by the CPU.
  - Enabling and Disabling Interrupts
  - Handling Multiple Devices
  - Vectored Interrupt
  - PCI Interrupt
  - Pipeline Interrupt
  - Interrupt Nesting
  - Interrupt Service Routine
  - Controlling Device Request
  - Use of Interrupt in OS.

Enabling & Disabling Interrupts

- Programmer complete control over the events.

Handling Multiple Devices.

- Capable of initiating interrupts are connected to the processor.

Polling Scheme.
- Processor poll all the I/O devices
- Periodically checking status bits.

Vectored Interrupt.

- To reduce the time involved in the polling Process
- Identify itself directly to the processor.

PCI Interrupt.

- Four interrupt pins defined for PCI
- PCI devices uses the PCI-Bus.

Pipeline Interrupt.

- Interrupt pipelining is a lightweight approach.
- High priority execution stage
- Running out-of-band interrupt handlers.

Interrupt Nesting.

- Long delay in responding
- Interrupt request may lead to error in operation of Computer.

Interrupt Service Routine.

- Multiple-level priority organization.
- Depending upon the device's priority.
- Assign a priority level to the processor.

Controlling Device Request.

- Interrupt-enable a bit
- Control register determines.
- Allowed to generate an interrupt request.
- Interrupt enable bit in the Ps register.

Use of Interrupts in Operating System.

- Coordinating all activities within a computer.
- Execution of user programs
- Implementing security and protection feautures.

Exception.

Event causes the execution of one program to be suspended.

Recovery from Errors.

- Faults, Traps and Aborts etc.
- Computers include error checking code.
- Processor by raising an interrupt.

Debugging.

- Trace and Breakpoints etc.
- System software includes a program called Debugger.

Privilege Exception.

- To protect the operating system.
- Being corrupted by user programs.
- Processor is in the supervisor mode.
- Instruction will produce a privilege exception.

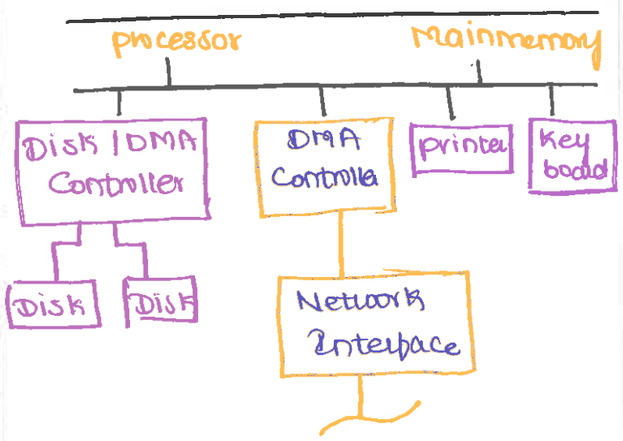
Direct Memory Addressing

* Instruction to transfer I/O deficis ready
* Transfer a block of data
* Without Continuous Intervention by the Processor.

DMA Controller

* Processor When accessing the main memory, keeps tracks of num of transfers

Diagram



DMA Registers

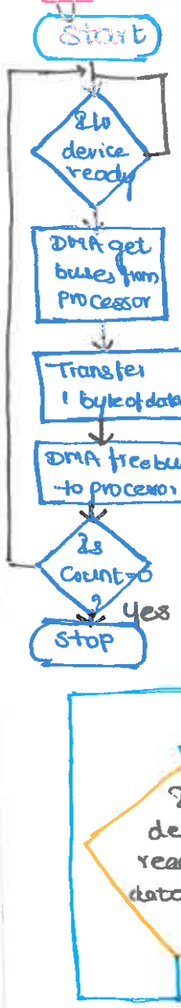
Address Register : To specify the desired location in memory
cpu writes data from MDR to mem location

Word Count Register:
* 16 bit register stores initial Address.
* data transfer takes place in a DMA Controller.

Control Register: Specifies the transfer mode
* Interrupt Control
* Switching the Addressing mode
* paging Control
* Coprocessor Control.

Modes of DMA Transfer

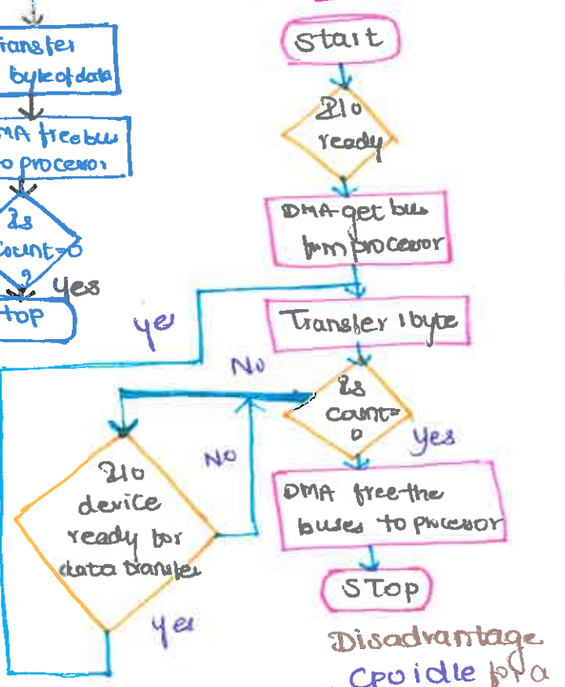
Single Transfer mode



Advantage
* Accuracy
* CPU not idle for long

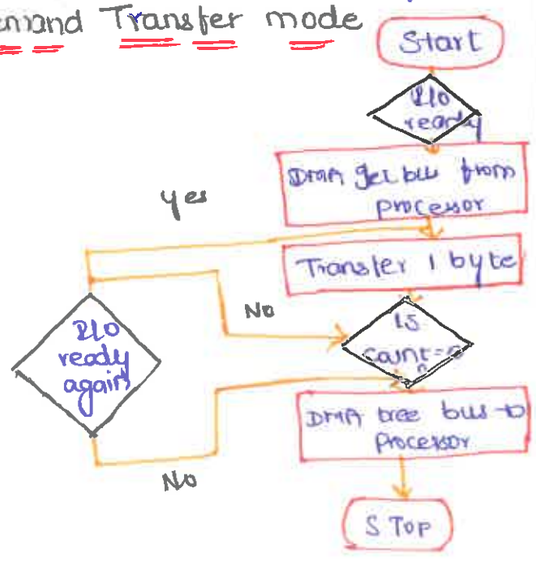
Disadvantage
* data-transfer
* rate-low

ii) Block Transfer Mode



Adv :- fast data Transfer
Disadv: CPU idle for a long

iii) Demand Transfer mode



Adv : very fast data transfer

Disadv : CPU not idle for long

Burst mode:
* exclusive access to the main memory.
* Transfer a block of data without Interruption

Cycle Stealing mode:
* Interweaving Technique called cycle Stealing

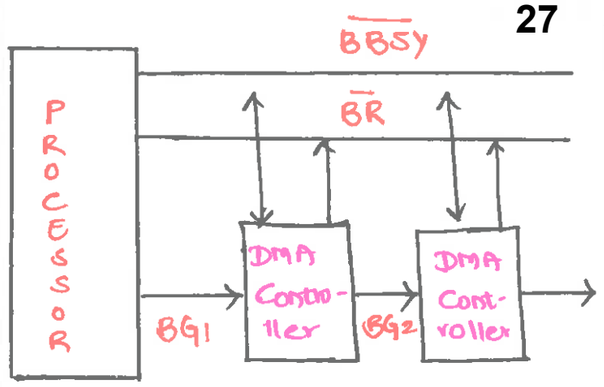
Interleaving Mode :
* Design Compensates Slow speed of DRAM
* Technique divides memory into a number of Modules.

Transparent mode:
* Transfer of block data
* Most efficient mode
* DMA Controller Transfers data CPU is performing operations

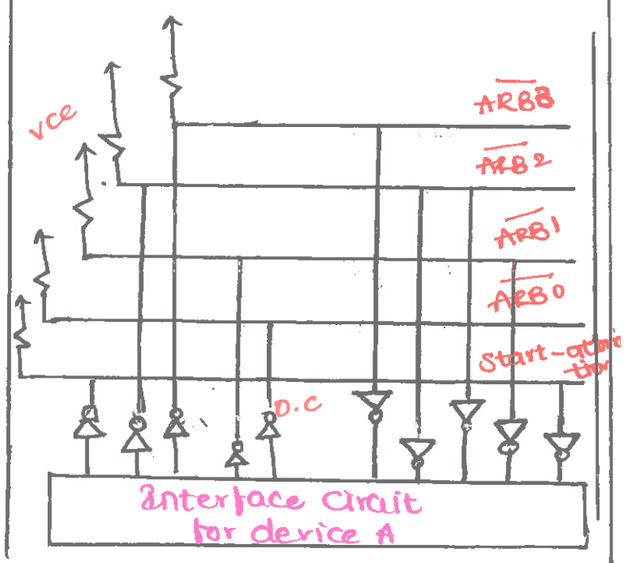
Bus Arbitration

* The device allowed to initiate data transfers on the bus at any given time is Called an bus matter

Centralised Arbitration
* Single bus arbiter performs the required arbitration
* The bus arbiter may be the processor or a seperate unit Connected to the bus.
* processor is normally the bus matter



Distributed Arbitration
* All devices waiting to use the bus have equal Responsibility
* Carrying out the arbitration process without using a central arbiter
* all devices participate in the Selection of the next bus.
* each device on the bus is assigned a 4-bit Identification number



CPU PERFORMANCE

1) If a computer A runs a program in 10 sec and computer B runs same pgm in 25 sec. who is faster and how much

$$\frac{\text{performance A}}{\text{performance B}} = \frac{\text{Execution time B}}{\text{Execution time A}} = n$$

$$= \frac{25}{10} = 2.5$$

performance of computer A is 2.5 greater than computer B

A is 2.5 times faster than B

2) consider 3 processors P_1, P_2, P_3 executing same instruction set

Processor	clock rate	CPI
P_1	3GHz	1.5
P_2	2.5GHz	1
P_3	4GHz	2.2

(i) Which processor has highest performance?

(ii) All the processor executes a program in 10 sec. Find the no. of cycles required and no. of instruction required for same program.

Sol) $\text{CPU Time} = \frac{N \times CPI}{R}$

P_1 = CPU Time = $\frac{N \times 1.5}{3GHz}$ = 500 PicosecXN

P_2 = CPU Time = $\frac{N \times 1}{2.5GHz}$ = 400 PicosecXN

P_3 = CPU Time = $\frac{N \times 2.2}{4GHz}$ = 550 PicosecXN

∴ Processor P_2 has highest performance since it has lowest CPU execution time 400PS

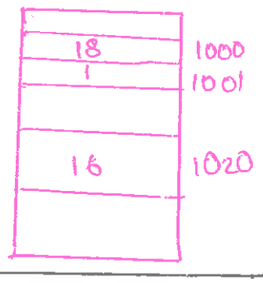
Addressing modes:

The memory location 1000, 1001 and 1020 have data value 18, 1 and 16 respectively before program is executed.

MOV I	R5, 1	move immediate
Load	Rd, 1000(R5)	Load from memory
ADD I	Rd, 1000	Add Immediate
store	0(Rd, 20	store immediate

Find memory location and its value

Before execution memory is



Lets execute:

1) MOV I, R5, 1

(*) Immediate mode

(*) R5 ← 1

(*) value = 1, moved to R5

Load Rd, 1000 (R5)

* user displacement addressing
* Rd ← 1000 + [R5])
* value = [1000 + [R5]] = [1000+1]
  = [1001] = 1
* ∴ value = 1, moved to Rd

ADD I Rd, 1000

(*) Immediate Addressing

(*) Rd ← [Rd] + 1000

(*) value = (Rd) + 1000 = 1 + 1000
= 1001

(*) ∴ value = 1001 moved to Rd

STORE I 0(Rd, 20

(*) Displacement Addressing

(*) 0 + (Rd) ← 20

(*) value of destination address
= 0 + [Rd] = 0 + 1001 = 1001

(*) value = 20 moved to 1001

∴ Program execution is completed memory location is 1001 has value 20

Instruction format

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

ZERO ADDRESS FORMAT

PUSH A TOP = A
PUSH A TOP = B
ADD TOP = A+B
PUSH C TOP = C
PUSH D TOP = D
ADD TOP = C+D
MVL TOP = [C+D] * [A+B]
POP X M[X] = TOP

ONE ADDRESS FORMAT

LOAD A AC = M[A]
ADD B AC = AC + M[A]
STORE T M[T] = AC
LOAD C AC = M[C]
ADD D AC = AC + M[D]
MVL T AC = AC × M[T]
STORE X M[X] = AC

TWO ADDRESS FORMAT

MOV R1, A R1 = M[A]
ADD R1, B R1 = R1 + M[B]
MOV R2, C R2 = C
ADD R2, D R2 = R2 + D
MVL R1, R2 R1 = R1 * R2
MOV X, R1 ⇒ M[X] = R1

THREE ADDRESS FORMAT

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

NUMBER SYSTEM

1) Convert number 5062_{10} to Binary System

2 | 5062	
2 | 2531	0
2 | 1265	1
2 | 632	1
2 | 316	0
2 | 158	0
2 | 79	0
2 | 39	1
2 | 19	1
2 | 9	1
2 | 4	1
2 | 2	0
1	0

$\therefore$ ANS = $(1001111000110)_2$

2) Convert 159_{10} to octal Number

159 to octal

8 | 159	
8 | 19	7
2	3

ANS = $(237)_8$

3) Convert 380_{10} to Hexadecimal

380_{10} to Hexadecimal

16 | 380	
16 | 23	12
1	7

ANS = $(17C)_{16}$

4) Convert 11001011_2 to decimal number System

11001011

$1 \times 2^0 = 1$
$1 \times 2^1 = 2$
$0 \times 2^2 = 0$
$1 \times 2^3 = 8$
$0 \times 2^4 = 0$
$0 \times 2^5 = 0$
$1 \times 2^6 = 64$
$1 \times 2^7 = 128$

203

ANS = $(11001011)_2 = (203)_{10}$

5) Convert octal Number 714_8 to decimal Number

714

$4 \times 8^0 = 4$
$1 \times 8^1 = 8$
$7 \times 8^2 = 448$

460

ANS = $(714)_8 = (460)_{10}$

6) Convert Hexadecimal Number 2C4 to decimal numberSystem.

2 C 4

$4 \times 16^0 = 4$
$C \times 16^1 = 192$
$2 \times 16^2 = 512$

708

ANS = $2C4_{16} = 708_{10}$

7) Convert 1001_2 to octal Number System.

$(1001)_2 = (\ )_8$

Step1 : Convert Binary to decimal.

1001

$1 \times 2^0 = 1$
$0 \times 2^1 = 0$
$0 \times 2^2 = 0$
$1 \times 2^3 = 8$

$(9)_{10}$

Step2 : Convert decimal to Octal Number.

8 | 9	
1	1

ANS = $(11)_8$

8) Convert $(672)_8$ to Hexadecimal Number.

Step1 : Octal to decimal.

6 7 2

$2 \times 8^0 = 2$
$7 \times 8^1 = 56$
$6 \times 8^2 = 384$

442

Step2 : Convert Decimal to Hexadecimal.

16 | 442	
16 | 27	10 (A)
1	11 (B)

ANS = $(1BA)_{16}$

9) Convert $(2020.65625)_{10}$ to ~~decimal~~ Binary

2 | 2020	
2 | 1010	0
2 | 505	0
2 | 252	1
2 | 126	0
2 | 63	0
2 | 31	1
2 | 15	1
2 | 7	1
2 | 3	1
1	1

$= (11111100100)_2$

Take decimal part.

- $.65625 \times 2 = 1.3125$
- $.3125 \times 2 = 0.625$
- $.625 \times 2 = 1.25$
- $.25 \times 2 = 0.5$
- $.5 \times 2 = 1.0$

$.65625 = .10101$

$\therefore 2020.65625 = 11111100100.10101$

Floating Point Operations:

1. Represent 1259.125_{10} in single Precision and Double Precision formats

(i) Integer = 1259_{10}

Fraction = 0.125_{10}

2	1259	
2	629	1
2	314	1
2	157	0
2	78	1
2	39	0
2	19	1
2	9	1
2	4	1
2	2	0
	1	0

$\Rightarrow 10011101011_2$

$0.125 \times 2 = 0.25 \Rightarrow 0$

$0.25 \times 2 = 0.50 \Rightarrow 0$

$0.50 \times 2 = 1.00 \Rightarrow 1$

$1259_{10} = 10011101011_2$

$0.125_{10} = 0.001_2$

Binary = 10011101011.001

(ii) Normalize the number

$10011101011.001 = 1.0011101011001 \times 2^{10}$

(iii) Single Precision

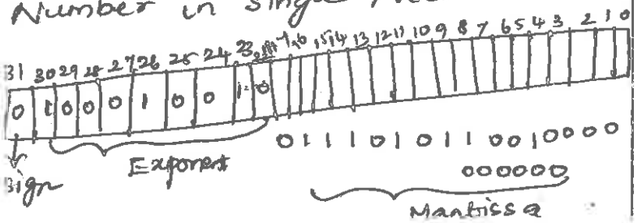
S = 0 E = 10 M = 00111010111001

Bias = 127

$E' = E + 127 = 10 + 127 = 137_{10}$

$= 10001001_2$

Number in single Precision Format



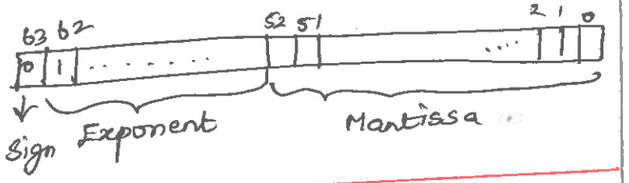
(iv) Double Precision

S = 0, E = 10, M = 0011101011001

Bias = 1023

$E' = E + 1023 = 10 + 1023 = 1033_{10}$

$= 10000001001_2$



Booth ALGORITHM

① Multiply (+11) & (−13) by binary multiplication method:

(−13) ⇒ convert to binary 1101

for −ve number find 2's complement

$1101 \Rightarrow \overset{1}{0010}+ \;\; 0011$

1 0 0 1 1 (−13) ⇒ (M)

1 0 1 1 (+11) ⇒ Q

0 ← sign bit

$$\begin{array}{r} 1\;0\;0\;1\;1 \quad (-13) \\ \times\; 0\;1\;0\;1\;1 \quad (+11) \\ \hline 1\;1\;1\;1\;1\;1\;0\;0\;1\;1 \\ 1\;1\;1\;1\;1\;0\;0\;1\;1 \\ 0\;0\;0\;0\;0\;0\;0\;0\;0\;0 \\ 1\;1\;1\;1\;0\;1\;1 \\ 0\;0\;0\;0\;0\;0 \\ \hline 1\;1\;0\;1\;1\;1\;0\;0\;0\;1 \end{array}$$

2's complement of answer

$00100011101+$

00100011111

$= 128 + 8 + 4 + 2 + 1 = [-143]$

② Multiply (+13) and (−6) using Booth algorithm

M = +13 (01101)

Q = −6

6 ⇒ 0110 ⇒ 2's complement of Q

$\overset{1}{1001}+ \;\; 1010$

Q = 11010 0 append 0 at end.

Recoding Table:

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit i−1	
0	0	0 × M
0	1	+1 × M
1	0	−1 × M
1	1	0 × M

Q ⇒ 1 1 0 1 0 0

0 −1 +1 −1 0

$$\begin{array}{r} 0\;1\;1\;0\;1 \quad \times \\ 0\;-1\;+1\;-1\;0 \\ \hline 0\;0\;0\;0\;0\;0\;0\;0\;0\;0 \\ 1\;1\;1\;1\;1\;0\;0\;1\;1 \\ 0\;0\;0\;0\;1\;1\;0\;1 \\ 1\;1\;1\;0\;0\;1\;1 \\ 0\;0\;0\;0\;0\;0 \\ \hline 1\;1\;1\;0\;1\;1\;0\;0\;1\;0 \end{array}$$

Take 2's complement as

$\overset{1}{0001001101}+$

$0001001110 \quad (-78)$

RESTORING DIVISION:-

A = 00000 M = 00011

Q ⇒ 1000

$11 \overline{)10000} \; 10$

	A	Q	
Initially	00000	1000	First cycle
Shift	00001	000□	
A−M (A+2'M)	11101		
	1 1110		
Restore	11	000 0	
A+M	00001	000 0	
Shift	00010	00□□	Second cycle
A−M	11101		
	1 1111		
A+M	11	00 0 0	
	00010		
Shift	00100	0□□□	Third cycle
A−M	11101		
	1 0001		
Restore	00001	0□□□ 1	
Shift	00010	□□□□	
A−M	11101		
	1 1111		
A+M	11	0 0 1 0	
	00010 Remainder	Quotient	

Non-Restoring Division:

Initially	00000	1000
Shift	00001	000□
A−M (A+M) set q_0	11101	000 0
	1 1110	
Shift	11100	00□□
A+M	00011	
(A+M) set q_0	1 1111	00 0 0
Shift	11110	0□□□
A+M	00011	
Set q_0	0 0001	0 0 0 1
Shift	00010	□□□□
A−M	11101	
A+M	1 1111	0 0 1 0
		Quotient

Add A+M

$\overset{1}{1111}+ \;\; 00011$

$00010 \rightarrow$ Remainder

PIPELINING

1. Consider a pipeline having 4 phases with duration 60, 50, 90 and 80 -ns Given Latch delay is 10ns Calculate

(i) pipeline Cycle time
(ii) Non pipeline execution time
(iii) Speed up ratio
(iv) pipeline time For 1000 tasks
(v) Sequential time For 1000 tasks
(vi) through put

Solution:-

Given : Stages = 4
Delay = 60, 50, 90, 80 ns
latch delay = 10ns

(i) pipeline Cycle Time
= Max. delay due to any stage + Delay due to its register
= Max (60, 50, 90, 80) + 10ns
= 90ns + 10ns
= 100 ns

(ii) Non-pipeline Execution Time
= Sum of delay of 4 phases
= 60ns + 50ns + 90ns + 80ns
= 280ns

(iii) Speed up ratio
= Non pipeline execution Time / pipeline Execution Time
= 280ns / 100ns = 2.8

(iv) pipeline Time For 1000 Tasks
= Time Taken For 1st task + Time taken for remaining 999 tasks
= 1 x 4 Clockcycles + 999 x 1 Clockcycle
= 4 x 100ns + 999 x 100 ns
= 400ns + 99900ns
= 100300ns

(v) Sequential time For 1000 Tasks
Non-pipeline time For 1000 Tasks
= 1000 x time Taken For one task
= 1000 x 280ns
= 280000 ns

(vi) Throughput
= Number of instructions executed permit time
= 1000 tasks / 100300 ns

---

2. A Four Stage pipeline has the Stage delays. Consider a non-pipelined processor with a Clock rate of 2.5 giga-hertz and average cycles per instruction of 4. the Same processor is upgraded to a pipelined processor with 5 stages but due to the interval pipeline delay, the clock Speed is reduced to 2 gigahertz. Assume there are no stalls in the pipeline. the Speed up achieved in the pipelined processor is

① 3.2 ② 3.0 ③ 2.2 ④ 2.0

Cycle Time in Non-pipelined processor = 1 / Frequency

Frequency of the clock = 2.5 Gz
Cycle Time = 1/2.5 Gz
= 1/2.5 x 10^{-9} Hertz
= 0.4ns

non-pipeline Execution Time
= Number of Clock Cycles / Instruction
= 4 Clock Cycles
= 4 x 0.4ns = 1.6ns

Cycle Time in pipelined processor
= 1 / Frequency = 1/2 Gz
= 1 / 2x10^9 hertz = 0.5ns

Pipeline execution Time
No stalls in pipeline, so one instruction) Clock cycle
= 1 Clock Cycle = 0.5ns

speed up
= Non-pipeline Execution Time / pipeline Execution Time
= 1.6ns / 0.5ns = 3.2

Option ① is the Correct answer

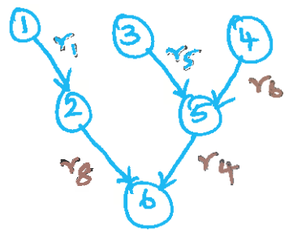
---

③ In Super Scalar processor, Find the in-order execution, out-of order execution, In-order 2 way Superscalar processor for the given Instruction

(1) r1 ← r4 / r7
(2) r8 ← r1 + r2
(3) r5 ← r5 + 1
(4) r6 ← r6 - r3
(5) r4 ← r5 + r6
(6) r7 ← r8 * r4

Solution:

Data flow graph For the given instruction



Assume that division takes 20 Cycles

In-order execution

1	2	3	4	5	6

In-order (2 way Superscalar)

1	2	4	5	6
	3			

Out-of order Execution

	1			
3	5		2	6
4				

VIRTUAL MEMORY

1. A Computer with Virtual Memory has an access time to main memory 50 ns, the time to transfer a block from the virtual into main memory is 10 ms. The Probability for the Page fault is 10^{-6}. What is the average access time, if the Page-table is in the main memory?

Solution:

$t_{ag} = 10$ ns.

$t_B = 10$ ms.

$(1-H) = 10^{-6}$.

$t_a = ?$

$t_a = t_{ag} + t_{ag} + (1-H) \times t_B$.

↙ Access to Page table in main Memory ↘ Access to the Page-frame.

$t_a = (50 \times 10^{-9}) + (50 \times 10^{-9}) + (10^{-6} \times (10 \times 10^{-3})$

$= 100 \times 10^{-9} + 10 \times 10^{-9}$

$= 110 \times 10^{-9}$

$t_a = 110$ ns.

∴ The average access time = 110 ns.

2. Computer with virtual memory has the following features.

* Length of virtual address is 38-bits.
* Page size is 16 KB.
* Length of physical address is 32-bits

(a) How many bits is the length of Page descriptor, if in addition to the frame number (FN), additional Parameters occupy another 6 bits?

Solution: $n = 38$, $f = 32$.

Page Size = 16 KB $= 2^P = 2^{14}$ B.

* No. of Pages in Virtual Memory:

$2^{n-P} = 2^n \div 2^P = 2^{38} \div 2^{14} = 2^{24}$.

* Number of Page frames in Main Memory:

$2^{f-P} = 2^f \div 2^P = 2^{32} \div 2^{14} = 2^{18}$. (FN)

Page descriptor = Frame number (FN) + 6

$= 18 + 6 = 24$ bits $= 3$ B.

6 bits	18 bits
Additional Parameters	Frame Number.

(b) What is the maximum size of the Page table in bytes?

Size of Page table = No. of Pages × Page descriptor

$= 2^{24} \times 3$ B

$= 16777216 \times 3$ B

$= 50331648$ B

∴ Size of Page Table = 49152 KB = 48 MB.

3. If a Processor has 32-bit Virtual address, 28-bit physical address, 2 KB Pages. How many bits are required for the virtual, physical Page number?

(A). 17, 21

(B). 21, 17

(C). 6, 10

(D) 10, 15

Solution:

Virtual Address = 32-bit

Virtual Address Space = 2^{32} B.

Physical Address = 28-bit

Physical Address Space = 2^{28} bit

Page size = 2 KB = 2^{11}.

No. of entries for Virtual Page Number

$= 2^{\text{Virtual Address} / \text{Page size}}$

$= 2^{32}/2^{11} = 2^{21}$.

Number of entries for physical Page Number

$= 2^{\text{Physical Address} / \text{Page size}}$

$= 2^{28}/2^{11} = 2^{17}$.

∴ No. of bits for Virtual Page Number = 21,

∴ No. of bits for physical Page Number = 17.

4. In a Paging Scheme, Virtual address space is 4 KB and Page table entry size is 8 bytes. what should be the optimal Page size?

Solution:

Given,

* Virtual address space = 4 KB. (Processor size)
* Physical Page Table entry = 8 Bytes.

Optimal Page Size

$= (2 \times \text{processor size} \times \text{Page table entry size})^{1/2}$

$= (2 \times 4\text{ KB} \times 8 \text{ Bytes})^{1/2}$

$= (2 \times 4096 \times 8 \text{ bytes})^{1/2}$

$= (8192 \times 8 \text{ bytes})^{1/2}$

$= (65536 \text{ bytes})^{1/2}$

$= 256$ bytes.

Thus,

optimal Page Size = 256 Bytes.

VIRTUAL MEMORY :-

① A 32 bit address System uses a paged virtual memory, the page size is 2 kBytes. What is the virtual Page and the offset in the page for the virtual address 0x00030f40?

For a page size of N Bytes, the number of bits in the offset field is $\log_2 N$.

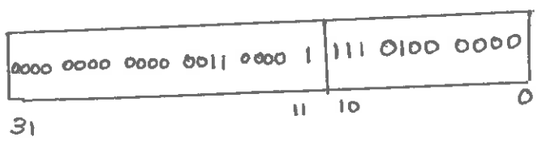
In case of a 2KBytes page,

$\log_2 2^{11} = 11$ bits.

Therefore, number of bits for the page number is

$32 - 11 = 21$, which means a total of $2^{21} = 2$ Mpages.

Binary representation of the address is



Given virtual Address identifies the virtual page Number

$0x61 = 97_{10}$.

The offset inside the page is

$0x740 = 1856_{10}$.

② A 32 bit address system, has a 4 MBytes main memory. The page size is 1 KByte. What is the size of the page Table?

The page table addressed with page number field in virtual address. Number of lines in page table equals the number of virtual pages.

$$\text{Virtual Pages} = \frac{\text{address Space [Bytes]}}{\text{page size [Bytes]}}$$

$$= \frac{2^{32}}{2^{10}} = 2^{22} = 4 \text{ M pages}$$

* Width of each line in the page table equals width of the page number field in virtual address.

* If no. of virtual pages is 2^{22}, then line width is 22 bits.

Page table size = Number of Entries * line size.

$= 2^{22} * 22$ bits

$= 11.5$ M byte.

CACHE Memory

1. A block-set Associate Cache memory Consists of 128 blocks divided into four blocks sets. The main memory Consists of 16384 blocks & each block Contains 256 eight bit words.

1. How many bits are required for Addressing the main memory?
2. How many bits are needed to represent the TAG SET & word fields?

Solution:

Given

No-of blocks in Cache memory = 128

No-of blocks in each set of Cache = 4

main memory size = 16384 blocks

Block size = 256 bytes

1 word = 8 bits = 1 byte

Main Memory Size

Size of main memory = 16384 blocks

$= 16384 \times 256$ bytes

$= 4194304$ bytes

$= 2^{22}$ bytes

The no. of bits required to address main memory } = 22

Number of bits in Block offset:

We have

Block size = 256 bytes $= 2^8$ bytes

Thus, number of bits in block offset or word } = 8 bits

Number of bits in set number:

no. of sets in Cache = no. of lines in Cache | set size

= 128 blocks | 4 blocks

= 32 sets

= 2^5 sets

Thus, no. of bits in set number = 5 bits

Number of bits in Tag number

no. of bits in tag = no. of bits in physical address − (no. of bits in set no + no. of bits in word)

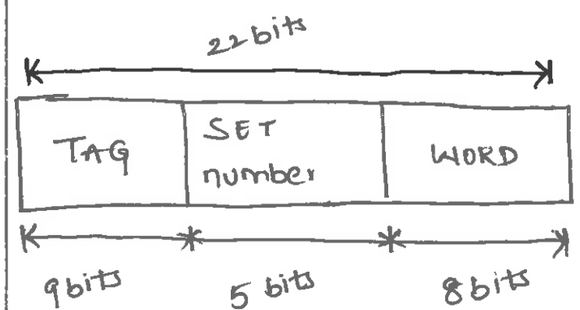
= 22 bits − (5 bits + 8 bits)

= 22 bits − 13 bits

= 9 bits

Thus, no. of bits in Tag = 9 bits

Therefore, the physical Address is



SIMATS

SCHOOL OF ENGINEERING

Approved by AICTE | IET-UK Accreditation

Engineer to Excel

Saveetha Nagar, Thandalam, Chennai - 602 105, TamilNadu, India