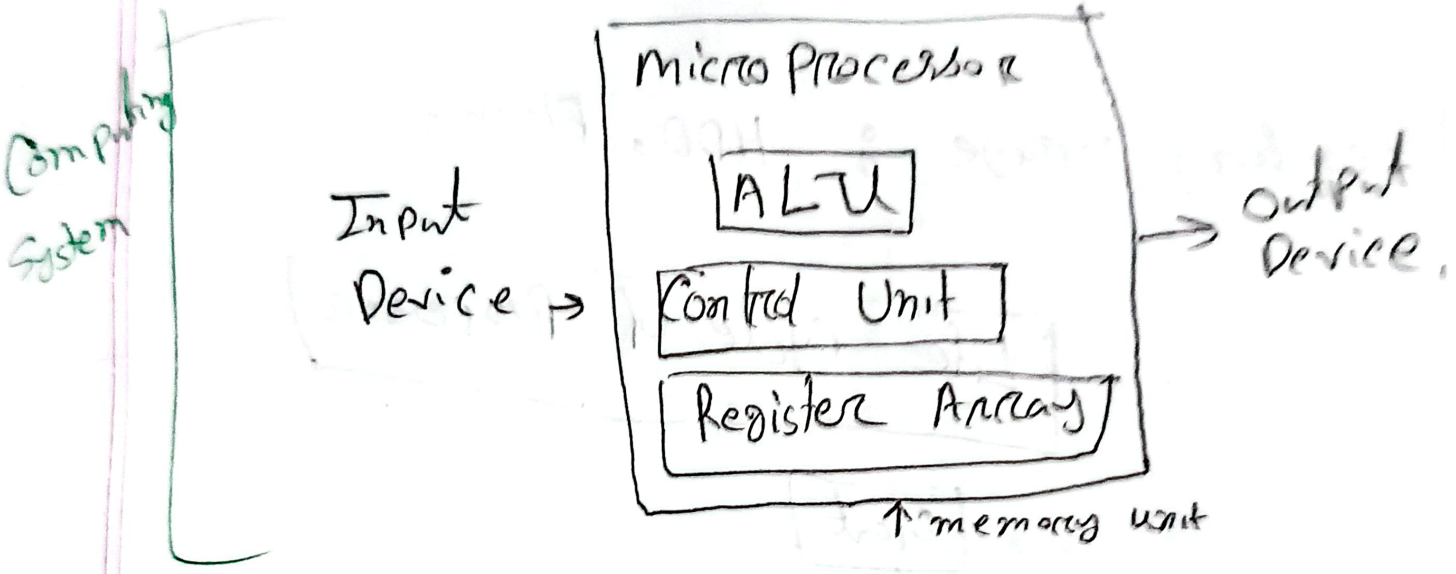
02/09/24

The integrated circuit which contain all the function of the unit CPU (central processing unit) of a computer is known as micro processor



1) ALU → Arithmatic Logic
Calculative works
conditional

ii) CU → generates control signal to alu to performing a specific operation

iii) Register → To store Data, results, commen

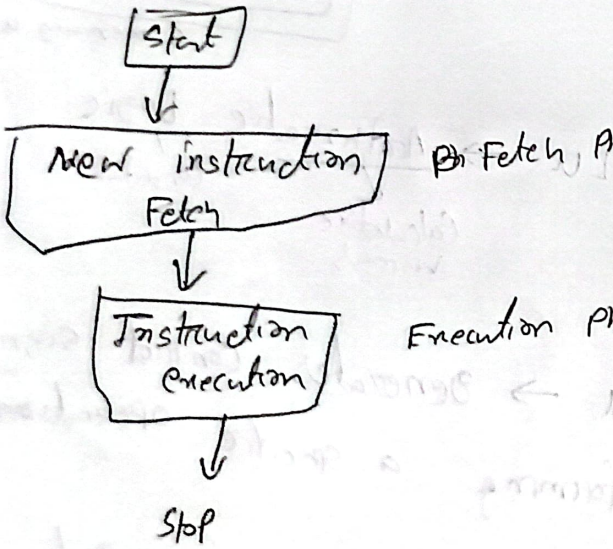
Memory:

① Primary storage (Processor can directly access)

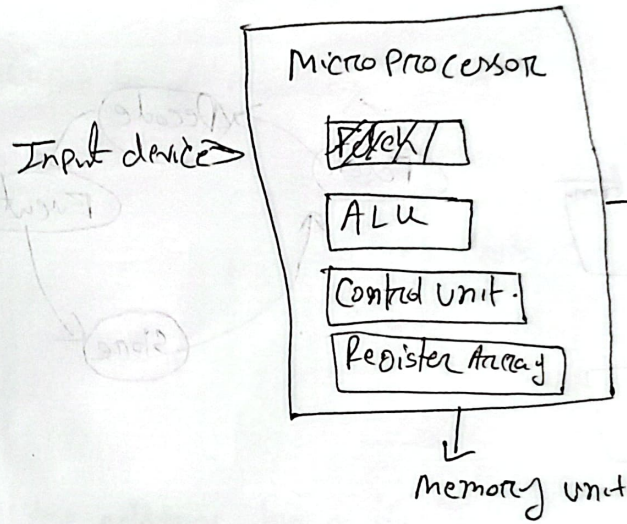
→ Cach, Ram, Rom (Non volatile)
Volatile

② Secondary storage : HDD, Flash,

Life cycle of Processor



Draw a Block diagram of a Computing Syste



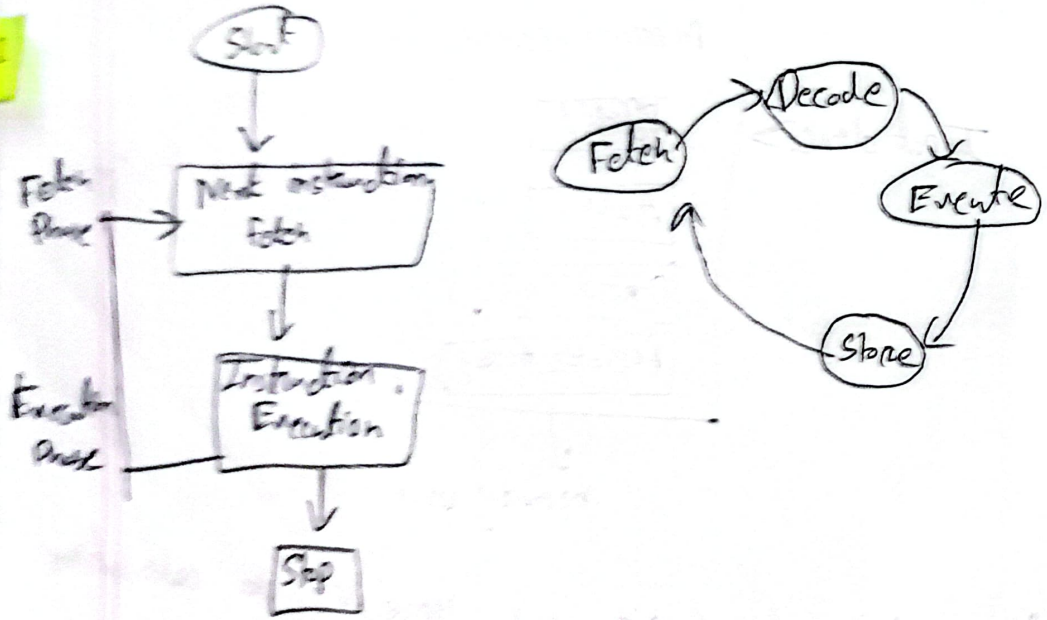
ALU: Arithmatic Logic Unit . Here we do calculat
Work with condition

Control Unit: generates control signal to alu to pe
a specific operation.

Register Array: Store, Data, results Comand.

Memory: Primary: Processor can directly access (volatile) cach, Ra
Secondary : HDD, Flash

Life Cycle of a microprocessor. Differentiate between fetch & execution.



Fetch: CPU Fetches instruction & Data from Ram

Decode: Control unit decode the instruction.

Execute: ALU executes instruction & operates on data.

Store: The Processed data is stored in Ram

If the address is 16-bit find out the capa of memory, organized as byte.

Capacity of memory $= 2^n$ [n = address of b

$= 2^{10}$

$= 1024$ byte

$= 1kB$

1kB = 102
8bit = 1

If the address bus is 12-bit find out the capacity organized as bytes.

Capacity of Ports $= 2^{\text{no of bus}}$

$= 2^{12}$

$= 4096$

$= 4096 / 1024$

$= 4kB$

Differentiate Fetch & Execution.

Fetch	Execution
(I) Retrives instruction from memory	(I) Performs the operation of instruction
(II) Involves Program Counter(PC) and memory Unit	(II) Involves ALU control Unit
(III) Read instruction & the instruction register	(III) Executes operation(logic, Arithmatic
(IV) loads the instruction for Processing	(IV) Produce results
(V) First stage of the instruction Cycle	(V) 2nd stage of instruction cycle

responsible for fetching instruction from memory is decoding while EU executes.

What are the transaction of BIU?

Definition: BIU (Bus Interface Unit) handles all the transaction of Data and addresses of EU

Operation of BIU:

(I) Instruction fetching
(II) Reading & writing operands for mem
(III) Calculating address of memory operand
(IV) Transfer instruction byte to the instruc

List the elements of BIU?

1) Instruction queue.
ii) Segment register
   - Ds (Data segment)
   - Stack segment (SS)
   - Extra segment (ES)
   - Code segment (CS)

III) Instruction
IV) Adress add

List the elements of EU

Definations: Eu execute instruction from the instruction system byte queue.

Elements:

(I) Control Circuit → Direct operation of EU by issuing control signals

(II) Instruction Decoder → Determines what EU perform

(III) ALU

(IV) Pointer and Index Register

1) Accumulator register (AH, AL) Ax
2) Base register (Bx = Bh + BL)
3) Counter Register (Cx = Ch + CL)
4) Data register (Dx = DH + DL)
5) Stack Pointer & Base Pointer (SP & BP)
6) Source index (SI)
7) Destination Index (DI)

} String & array

(V) Flag register

(BP) → Used to access data in SS but can also used in other segment

(V) ... to access SS.

What are the operations of EU?

(I) Decode instruction by BIU
(II) Generate control signal
(III) Executes instruction

• STACK (segment) Stack segment begins with directive S

Stack segment is used to store temporary actuly the content of register for interrup Section of Program mainly return address

- Size (size in no. of bytes)
- Stack look (allocates two bytes for segment)

→ Last in first out mechanism (LIFO)

→ main function
  → contain return address
  → Data
  → Content of Present register.

• Push → Decrese SP by 2 & store a value here
• Pop → returns a value from stack & increase S

Instruction Execution
is Addressing:
① Fetch the instruction Put to queue
② Decode
③ Execute.

. Data
memory variables will be declared in this section.

. CODE The code segment beggins with the directives.
logical portion of the code segment starts from here. Instruction goes after this Segment

MAIN PROC
Beginning of the main procedure. main Procedure instruction goes here

MAIN ENDP
Other Procedure goes here. But not part of the main Procedure.

END MAIN
Indicates the assembler that the main code of assembly code is finished

. MODEL SMALL
The model indicates size of the code & data segment.
model small indicates that it has one code segment not
Data segment size is greater than 64 kB.
Both Data & code fit whithing 64 kB segment each
Often used in small Programme that do not require large memories

Medium:
∝ code more than one segment, Data one seg

COMPACT:
∝ Code one segment, Data more than one

LARGE:
- Code more than one segment
- data more than one segment

One segment = 64 kB

A Programme Structure:

1) .Model small ;
2) .Stack 100H ; stack size. 100 bytes.
3) .Data :
4) .CODE :
5) MAIN PROC :
6) MAIN ENDP ;
7) END MAIN :

MOV / ADD / SUB / XCHG (Swap):

Syntax : Opcode, Destination, Source.

Possible Combination: **POSSIBLE COMBINATION:**

- Reg - reg
- Reg - mem
- Mem - reg
- mem - reg
- mem - Immd.
- Reg - Immd

INC / DEC / NEG / MUL / DIV:

Syntax : OPcode Operannad

Example : INC An

NEG An, 2s Complement of An

Intel 8086 Internal Architecture:

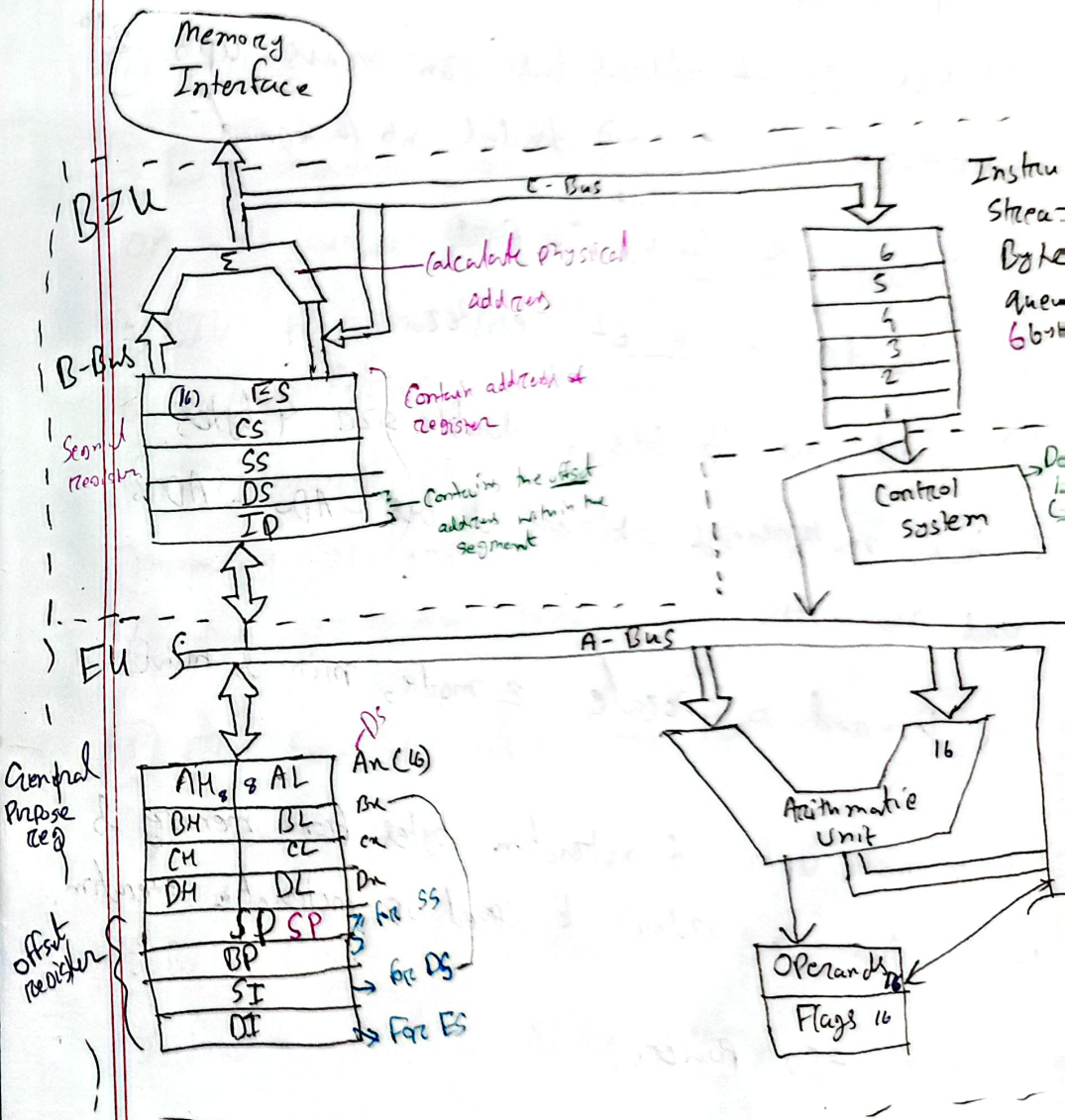


Fig: 8086 internal block Diagram

8086 Architecture Features:

(1) 64 bit up

(II) has 20 bit address bus can access upto 2^{20} memory location (1MB) divided into 16 segment.

(III) Support upto 64k (1 segment) I/O Port

(IV) Provide 14, 16 bit register.

(V) Word size 16 bits, double size 4 bytes.

(6) has multiplexed address & bus AD0 – AD15 and A16 – A19

(7) it's Designed to operate 2 mode, min & max

(8) Prefetches upto 6 instruction bytes from memory & queues them in order to speed up instruction execution.

(9) requires +5V Power.

(10) 40 pin dual line package.

(11) Address range 00000 to FFFFF

(12) 2 block BIU & EU

(13) BIU handles all transaction of data & address on the buses for EU

(14) BIU Performs all the bus operation such as instruction fetching reading & writing operand from memory and calculating the address of memory operands. The instruction bytes are to the instruction queue.

(15) EU Executes instructions from the instruction byte queue.

(16) BIU Contains instruction queue. Segment Instruction pointer Address adder.

(17) EU contains control circuit, instruction decoder, ALU, pointer, and Index register, flag register

Execution Unit:

(I) Decode instruction fetched by the BIU

(II) Generates control signal,

(III) Executes instruction,

The main Parts are:

(I) Control Circuitry

(II) instruction Decoder

(III) ALU

Directive Model: The model indicates size of the Code & Data segment

SMALL	Code → 1 segment Data → 1 segment
MEDIUM	Code → more than 1 segment Data → 1 segment
Compact	Code → more than 1 segment data → more than 1
large	more than 1 → code more than 1 → Data

Segment Directives:

8086 uses segment; Special areas defined to its Co Code, Data & Stack.

Assembly language got 3 segment: (I) stack (II) Data (III) CODE

Data Segment; begins with directive .Data

.Data is the logical definition of Data segment, me variables are declared here.

Data types : 8086 has 2 types of Data | DB, DW

There are 4 types of Data

(I) Bytes (8-bit) : can be defined as DB = Define Byte

(II) Word (16-bit) : " " " DW = Define word

(III) Quad Double word (64 bit) : " " DQ = Define Quad word

(IV) Double word (32-bit) : DD = Double word.

Byte (80-bit) : DT = Define Ten byte

Syntax : N_name type value

8086 works with DB & DW

DB	DW
(I) Single byte (8 bit) of memory	(I) Used to define a word
(II) Can be used to store char data or small numerical value in unsigned or signed form	(II) Used to storing larger value.
	(III) 16 bit (2 byte)

Define constants :

To assign a name to a constant we use EQU()

Segment Registers :

SS → Stack segment

DS → Data segment

CS → Code segment

ES → Extra segment

Procedures : Inside a code segment instruction ar

Organised as Procedure

Tanah

Tanah A 1

Operation এর সময়
এর Data seg ও
এর Extra

Programme
- Code → Instruction
- Data → variables
- Stack — Return address
- Extra → another Data segment

* Model বলতে Code segment আর data segment কতটুকু space নিবে
* Model is a specifier tells the assembler/system about
* .Code দিবে instructions.
* Assembly only has Integer value.
* D DW 10h, 20h, 30h → array.
* A EQU 1234H — equates constant data
* .Data → data segment শুরু
* .Stack Size → Stack segment শুরু
  ↳ .STACK 100h → allocates 100 bytes for stack seg
* Instruction → MOV / Add / SUB / * CHNG

Syntax: Opcode, Destination, source

General Purpose Registers:

Possible Combination:

→ Reg – Reg
→ Reg – mem
→ mem → Reg
→ mem — Immd
→ Reg — Immd

Source & Destination size must be equal

* Mov → B8 [machine code]

* Code segment এ Data seg use করতে হলে আগে initialize করতে হবে

```
MOV AX, @data
MOV DS, AX
```

Fall-23 MiD

(3) Write assembly code for below:

(i) input the first letter of your name

(ii) Display the letter in the next line

(iii) Display your name as a string in next line with a beep.

```
.model small
.stack 100h
.data
name db 'Akhi $'
.code
main proc
    mov ax, @data
    mov ds, ax

    mov ah, 1
    int 21h
    mov bl, al

; Newline start
mov ah, 2
mov dl, 010
int 21h
; mov ah 2
; mov dl, 013
; int 21h
; Newline end
```

```
    mov ah, 2
    mov dl, bl
    int 21h
; newline start
    mov ah, 2
    mov dl, 010
    int 21h
    mov ah, 2
    mov dl, 013
    int 21h
; newline end

; Display name as a string
    mov ah, 9
    lea dx, name
    int 21h
; Beep Sound
    mov ah, 2
    mov dl, 7
    int 21h

    mov ah 4Ch
    int 21h
main endp
End main
```

Difference between Stack & queue.

Stack	queue
① linear data structure data are removed or inserted at the same end	① linear data structure but insertion & removal happens 2 different ends
② Follows Lifo	② Follows fifo
③ One pointer at the top	③ 2 Pointer front & back
④ Push & Pop	④ enqueue & dequeue
⑤ insertion & deletion from top	⑤ insertion rear, de Front

Differentiate between fetch & Execution;

The term fetch & execution refers to distinct phases in the Process of handling instruction or data in Computing.

Fetch;

Process of retrieving data or instruction from a storage location (disk, memory) to prepare for Processing In cpu the Processor fetches an instruction from memory to execute it.

Execution;

Process of actully running or acting on the fetched instruction, or data, here actual Processing happens. In cpu Processor executes the instruction, after it is fetched,

Microprocessor have -4 bit what does it mean?

It means a microprocessor can handle/transfer 4 bits of Data at a time, which is, also called nibble

Data width; It processes 4 bit of data at a time (half byte)

Operations; Any arithmatic or logical Operation Pe by the cpu will be on 4 bit values.

Memory; A 4bit address bus means the cpu acces a maximum of 16 memory location ($2^4 = 2$ directly.

Example intel 4004

A microprocessor has 64 kB memory (Ram). What is the length of address bus? [Compute from memory capacity]

Address bus (bits) $= \log_2$ (memory capacity in bytes)

We have 64 kb $= 64 \times 1024$

$= 65536$ bytes.

$\therefore$ Address bus $= \log_2(65536)$

$= 16$ bits

$2^{16} = 65536$

We know that,

size $= 2^{\text{address bus}}$

$\Rightarrow 2^{16} = 65536$ $\quad 64 \times 1024 = 65536$

$\Rightarrow 2^{\text{address bus}} = 2^{16}$ $\quad 2^{\text{add bus}} = 65536$

$\Rightarrow$ address bus $= 16$.

2^4 2^9 2^{16} 2^{23}

Stack with SP Push.

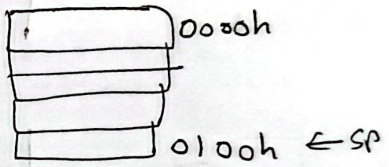
A stack operation follows the LIFO, meaning the last item pushed onto in stack is the one will be pop

First,

(I) Push $\rightarrow$ insert

(II) Pop $\rightarrow$ remove

Push

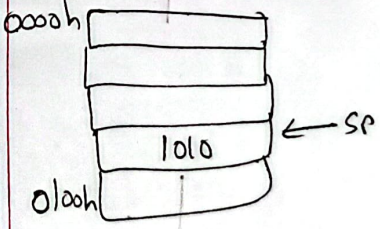


1100 → Bus

1010 → Add

0100h → SP

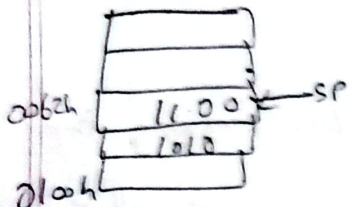
emf



1100 → Bus

0062h → Add

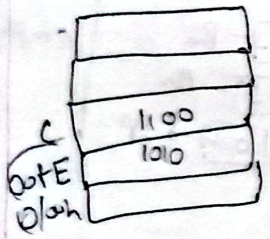
0062h → SP



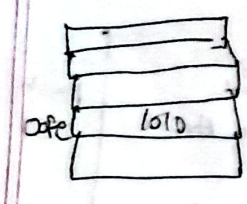
00Gh – Bx
00Fh – Ax
00Eh – SP

Push-
Bx

Pop:

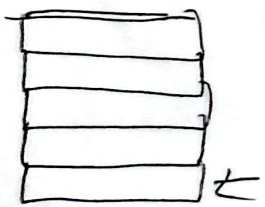


Before Pop



SP → 00Fe
Cx → 1100

Pop & Cx



SP = 100h
Bx 1010

Pop, Dx

DI (Destination Index) can be used for the destination data in string

IP (Instruction Pointer): Used to keep offset Add

Data Register:

Ax (Accumulator Regis): Used for arithmetic like mul or dividing whole number (called words, 16 bit)

Ex: if multiply 2 number, one of them will be stored in Ax register.

AL → Used for operation involve small num. (add or mul small num)

Ah →

Bx (Base register): used to save data & address

Cx (count Register): Used for loops. how many time should run.

Cl →

Dx (Data register): Dx is used when a result bigger than what a single register can hold, (

If multiplication result is too big, it'll go to Ax o
if overflow goes to Dx

Dx also holds Part of address.

Dh → higher data 8bit

Dl → lower " 8bit

Port	Register
(I) Interface for external I/O	(I) temporary storage for data
(II) External to the CPU core	(II) Internal to CPU
(III) Used for input/output	(III) Used for internal processing
(IV) CPU I/O	(IV) Ax, Bx

* SP → Points the top of the stack. Used with SS to access the stack segment

* BP (Base Pointer) → (I) Used to access data on the stack
(II) Then can be used to access data in other segment

* SI (Source Index) → Can be used in string & Array Operation

Ax, Dx for multiplication

```
mov Ax, 1234h
mov bx, 4321h
Mul Bx
```

(I) Result will be stored in Ax & Bx

(II) if result bits 16 bit Dx will be 0000h

(III) if it is more than 16 bit it will be in Dx
Stored both Ax And bx. 16 bit will be in Ax
rest in Dx

---

64 KB

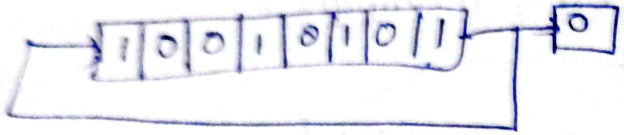
$$\text{Size} = 2^{\text{address bus}}$$

$$64 \times 1024 = 2^{\text{address bus}}$$

$$\Rightarrow 2^{65536} = 2^{\text{size}}$$

Size —

ROR AL, CL



1 1001010 1

0 1100101 0

What is Pipelining?

The Process of fetching the next instruction in advance while EU is executing current instr

Physical address = Seg × 10h + offset

16 bit
Segment address
Chapter 5

16 bit
offset add
Page 30

Seg reg → 16
offset → 16
Convert them 20 bit by sh 4 then

Assume
CS = 1000 H
IP = 2345 H

offset = BX, SI, DI, BP

Every instruction register has its own unique opcode when it into machine lang

has
ADD → Unique opcode
Sub → " "

mov bl, 254
has opcode
no
num
her

if 1 byte empty from queue it wont refill it'll wait for 2 byte empty.

As 1 byte = 8 bit & 2 byte = 16 bit.

it can transfer 16 bit at a time so rather wasting time by transfering 8 bit. it transfer 2 byte / 16 bit

But what if the next instruction is 4 byte and only 2 byte empty?

BIU will get half instruction and wait for empty.

4 Pipeline fails when there is a branch / Jump

Ax = 1234

∴ ~~AL = 12~~ ∴ AH = 12

AL = 34

CMP Instructions

CMP destination, Source

- Substract Source from destination.
- Result is not stored.
- Only flag are affected.
- Destination constant হতে পারবে না

Al Cmp equal to 5 কিনা হবে 1st এ 5 কে রাখতে

General purpose register এ রাখতে হবে

```
mov cx, Ax
Cmp Bx, cx
JLE N
mov cx, Bx
N:
```

10 - 50
Bx - cx
= -40

50 -
Bx -
⇒ 40
pos হবে

২/ JMP unconditional.

৩) IF - then,

Ax যদি 0 থেকে ছোট হয় positive করব ৩

CMP Ax, 0

JNL End-IF

NEG Ax

End if

Ax - 0 = -1 - 0
= -1
= -1
<
JNL মানে
Not less than
zero
যেহেতু < 0
যেতে হবে,
তাই নিচে যাবে
Direct End if ৩
সেখানে যাবে neg Ax
তারপর করবে।

But -1 0 থেকে
ছোট তাই
JNL হবে না
Neg Ax হবে
-1 C

৪) Ax NEG হলে Bx = -1
Ax 0 " " Bx = 0
Ax Positive num হলে Bx = 1

(i) Ax - 0 = -1 - 0 → Bx = -1

(ii) 0 - 0 =

```
CMP Ax, 0
JL NEGA
JE ZERO
JG POSITIVE

NEGA:
    MOV Bx, -1
    JMP End-case.

ZERO:
    mov Bx, 0
    Jmp End case

POSITIVE:
```

2) For And operation

Flag = PF, SF, ZF affected

=) CF = of = 0

AF undefined.

AND AL, BL

∴ AL = 47H = 0100 0111

BL = 36H = 0011 0110 (And)

0000 0110 = 06H

Physical Address Calculation

mov AX, [BX][SI]

PA = 10h × DS × [BX+SI]

we know,

BX and SI is under Data seg

Multiplication & Division

Instruction ← Opcode
Operand ← Source / Destination

Offset Ad = Effective Ad

Source ~~에서~~ Register 2개, Register 1개

① Register Addressing mode

~~2개~~ Source / Destination ~~중~~ 2개 ~~중~~ 1개 is Imidiate

2) Direct number then ~~1개~~ ~~imidi~~

① imidiate Addressing mode

Dest Register ← Mov BX, 1234H

Dest Memory ← mov [BX], 12H

mov AL, A

Opcode, Dest Reg
Source

Des

memory Addressing mode → 16 bit

Logical Address = Segment x offset

Physical Address = Seg x 10h + offset → 20 bit

8086 has 14 register each size 16 bit।

Ax, Cx, Dx
SP, BP, SI, DI → Can Carrying offset

IP Flag

Offset	Segment register
Bx	DS
BP/SP	SS
SI	DS
DI	DS
DI	ES
IP	CS

General Purpose register এর মধ্যে শুধু Bx offset carry করে

(Bx, DS row: Array হলে) (DI, ES row: string হলে)

Destination memory হলে immediate memory ad...

Source memory হলে register Indirect

Source যদি Direct number হয় but memor... এর মধ্যে থাকে Direct Addressing

Mov Ax, [1234h]

Base Addressing mode: 1D Array এর জন্য ব্যবহার হয়।

Base Ad mode এ offset address থেকে ২টি Data লাগে $\frac{Bx}{Ds}$ or $\frac{BP}{SS}$

8 bit sign = -128 to ...

unsign → 0-6553...

Offset = Bx + A   mov Ax, A[Bx]

Offset = BP + A

DS

(1) Mov Ax, A [Bx]

Ofset = Bx + A

Physical Ad = Seg × 10h + Ofset

= DS × 10h + (Bx + A)

=

(2) Mov Ax, [A+BP] → SS segment

Ofset = A + BP

PA = SS × 10h + (BP+A)

Base Address And INDEX Addressing mode

Same ই Just Base ~~Index~~ এর জায়গায় Index

Register use হয়

| Base reg = BP, Bx → DS (BP → SS)
| Index " = SI, DI

Based indexed / 2D Array :

(1) Mov Ax, [SI][Bx] → DS

Offset = Bx + SI + Displacement

(II) Mov Ax, [SI] [BP] → Stack

Offset = BP + SI + Disp

String Add :

Direct I/O Ad mode

In AL/AX, Port

out Port, AL/AX,

1) Out DX, AL (Port)

2) out DX, AX

Relative Addressing mode

Opcode displacement

Call

Jump L — IP এর মান change হয়

JNZ L

JMP Exit

Loop L

Int 21 — code segment

Ret — IP value change

offset: IP = IP + Exit

IP = IP + L

Segment = CS

Control Flag

Implied Addressing mode → no operand address

HLT, NOP, CLC, CLS

শুধু opcode.

৩টি Control Flag।

Trace Flag TF = Debug.

STI Set TF

CLT Clear TF

DF → String

DF = 0 হলে SI/DI auto increment

DF = 1 " Decrement

Interrupt Flag →

Addressing mode অনুযায়ী

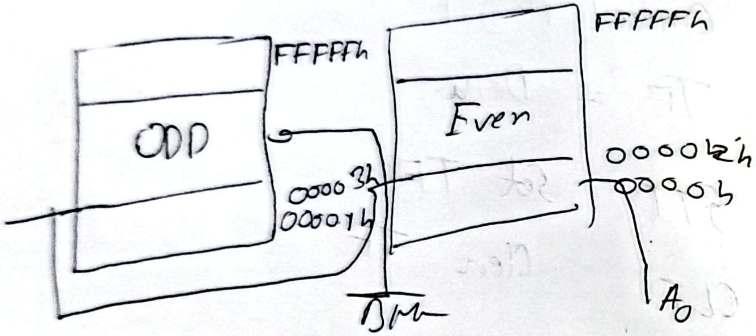
Phy Ad. Cal, logical Ad.

Memory Bank

8086 এর Address bus 20 bit

" " Data bus → 16 bit

" " memory Data bus → 8 bit



Analysis

Even bank contain even address ($A_0 = 0$) and connected to $D_0 - D_7$ and odd containing ODD ($A_0 = 1$) connected to $D_8 - D_{15}$. A particular byte is addressed by $A_1 - A_{19}$ address with A_0 as selector for even bank. BHE selector for odd address space.

সুবিধা, Data bus 16 bit but memory data bus 8 bit. So create 2 bank to fetch 16 bit data যদি Address even হয় তাহলে 16 bit আসবে, ODD হলে আসবে না, all the even address থেকে data শুরু হবে হয় তার lower byte even byte এর সাথে connected.

64 bit data bus হলে কয়টা Bank লাগবে

8 Banks.

16 bit এর জন্য লাগে 2

64 " " 8

Chapter 8 Stack

Lifo.

SP → Stack top

SP → 00000H হলে ~~empty~~ full Stack

SP → FFFF হলে empty Stack

Push হলে SP +2

Pop " SP − 2

Stack এ সবসময় word operation হয়

Stack এর আরেকটা operation Procedure?

2 type Procedure.

Same segment এ থাকে

ভিন্ন " " Far

Procedure call করলে IP change হয়ে যায়

IP = IP + Proc Nam হয়ে যায়

তখন আগের Value save করে রাখতে হবে

আগের value Stack এ Push হয় আর যখন

Procedure RET করে Then Address Pop করে.

CALL = Push

Ret = Pop

Stack এ 16 bit operation হয়

8086 এর BIU, EU, Pipeline, (que idle থাকে)

* SHL, SAR এ পার্থক্য কি)

* SAR, SHR এ পার্থক্য আছে thm bit M

* Test is logical And operation, is used to check Bit. Mask Pattern দিতে হয়

- Particular bit কে toggle করতে XOR
- bit কে set OR
- Clear And.
- Puratake compliment/toggle → NOT
- Chapter 6,

CMP, Conditional Jump

Skip a part Unconditional Jump

- loop infinite হবে যখন value 0

Flags for logic instruction

SF, ZF, PF Shows the result

AF undefined.

CF, OF = 0

Logic output has no carry. So CF = AF = 0

And it also has no sign SF = 0

Only ZF & PF useful.

NOT → No change in Status Flag

Test is logical And operation, no Result only Flag

Q:

I) Clear sign bit
II) Complement sign bit
III) set sign bit
IV) Test AL, 1
V) JZ below.
VI) is AL even?
VII) OR to below

Control Flag

TF → Trap Flag.
Enables trapping through an on-chip debugg

IF → Interrupt Flag.
Controls the operation of INTR

1) 8086 Feature

(I) 16-bit up
(II) 20 bit address bus
(III) memory 1mB 2^{20}
(IV) Support 64K I/O Port
(V) 16 bit register
(VI) has multiplexed address $AD_0 - AD_{15}$ and A
(VII) Designed to operate 2modes, Minimum and
(VII) require +5V Power.
(IX) 40 pin dual line Package
(X) Address range 00000H to FFFFFH

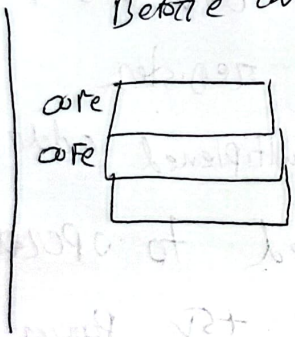
13

main Procedure is the main Procedure is the entry Point of execution. It doesn't need explicit Ret Statement The OS / emulator handles Program termination when main Procedure complete,

Call saves the return Address and Ret restore,

Before Call

Main Proc



Relative Addressing modes used for conditional and unconditional Jump. Offset can be Positive,

Jump Label

Intra seg	Inter seg
(I) Addressing within same Segment	(I) between different seg
(II) Operates within 16 bit	(II) Crosses 2016 +
(III) Short & near Jump	(III) Far Jump calls,