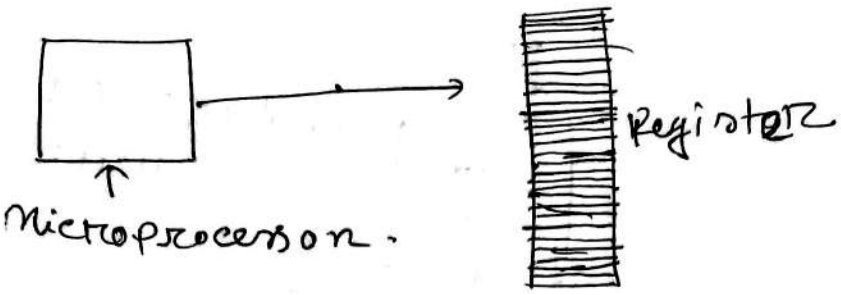
101

CSE-3109
BP

2A day
05.02.2018



→ Register microprocessor এর ভেতর embedded থাকে।

Register → AX, BX, CX, DX.
16 bit

AX (16bit) { 8 bit high byte : AH
8 bit low byte : AL

1. Assembly Language programming & organization of IBM PC
   → Ytha Yle
   → charles Marut.
2. The intel microprocessor
   → Barry B Brey

203 CSE-3109 2cday

BP 07.02.2018

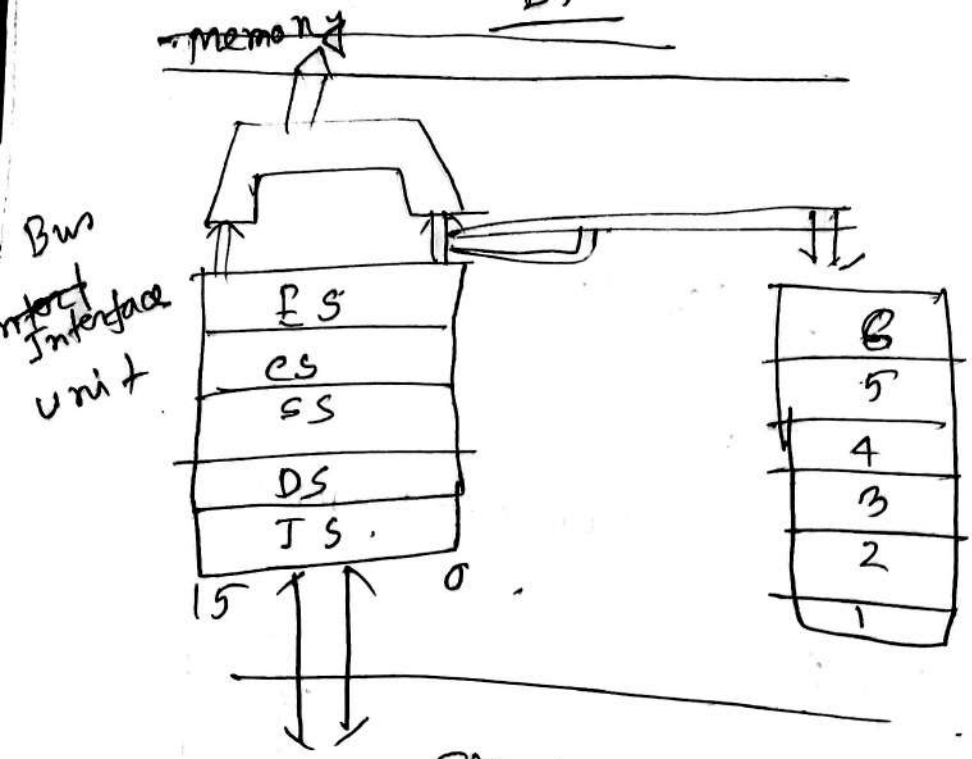


Fig-1

8086 → ১৬ বিটের microprocessor

8088 → ৮ বিটের Version

Bus → It is generally used to transfer data from one to another

- Address bus
- data bus
- control bus

→ 3 Types of Bus

Different microprocessor এ bus এর size vary করে

For figure-1

Address bus → 20 bit

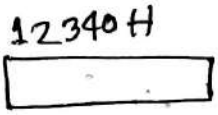
Data bus → 16 bit

→ A For Add bus → 20 bit we can use memory = 2^{20}.

ES → Extra Segment.
CS → code "
SS → Stack "
DS → Data "

} 4 type of Segment.

B ES, CS, SS, DS → One type of register.



⇒ code segment এর first byte এর address

12340H → 1234H = (segment address)

Segment : offset.

1234H : 0000H

→ Segment × 10 + Offset. (Logical → physical convert)

Instruction pointer (IP)

IP → 16 bit.

→ code segment এর IP এ 16 bit data insert হয়।

Instruction Queue (6 byte)

→ প্রতিবার এক্সিকিউশন point করে next IQ এ যায়।

Q1 Why microprocessor doesn't delay?

Pi Pelining → It get ready data for next stage.

2B Refilled → It refill the after excecution of 2 Byte from the queue.

Branching Problem →

```
if (        ) {
      1.
      2.
      ...
      10.
}
else {
      1
      2.
      ...
      10.
}}
```

→ 6B memory তে থাকে

⇒ If condition run না করে যদি else condition - এ যায় Then we need to clear if condition data (6 byte) and reload 6 byte data for else condition.

This is called branching problem.

→ A Typical Microprocessor's Assembly Language code

```
.MODEL SMALL
.STACK 100H
.CODE                → code segment
MAIN PROC
  -
  -
MAIN ENDP
END MAIN
```

MAIN PROC … MAIN ENDP → mainfunction define

Between MAIN PROC and MAIN ENDP → Here, we write code.

→ code model

→ If we insert a decimal value 1101.

Binary value 1101 B.

Hexadecimal value 1ABC H

MOV Destn, Source.

ADD Destn, Source

[8086 Emulator / MASM]

{Here one must be variable and one register

if MOV Destn = Var, then source = register}

```
MOV AX, 2
MOV A, AX
```

A=2
A = A+4 ;

If we want to operation through register ,

MOV Ax, 4.

ADD A , Ax n .

Here will be stored in A] A=variable

CSE-3109

BP

2D day

10.02.2018

DB → B

DW → 2B.

DD → 2W

DQ → 4W

DT → 10B

Var 1 DB 20H

Var1 DB ? [→ (For blank variable)]

Var1 DB 10H, 20H, 30H

↑ ↓ ↓ ↓

101H (address) 101H 102H 103H

(address)

SUB D, S.

ADD D, S

MOV D, S.

XCHG D, S

INC D

DEC D

70.

$7 \times 16^1 =$

Interrupt

INT 21h

↳ Interrupt এর মধ্যে যে fn -G আছে.

↳ Microprocessor কে interrupt করে.

* INT 21h (console থেকে input নেওয়া & অন্যান্য কাজ করে)

* AH ← 1.

(যদি আমরা AH ← 1 assign করে 21h fn call করি তাহলে single character input নিবে & AL এ রাখবে).

↳ Single character input to AL.

* AH ← 2.

↳ Single character output from DL & copy to AL.

* AH ← 9.

↳ String output from DX

DX should contain the offset of string to be printed.

Write a program Input a character and output the character

```
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    → (code) ——————→
```

MAIN

MAIN END MOV AX, @DATA
END MAIN MOV DS, AX.

LEA DX, MSG.
MOV AH, 9
INT 21h

MOV AH, 1
INT 21h

MOV DL, AL

MOV AH, 2.
INT 21h.

→ MOV AH, uCH.
INT 21h

MAIN ENDP
END MAIN.

LEA → Load Effective Address.

• DATA MSG DB 'please enter a char $'

→ Program Segment Prefix 256 B. ['It is a one kind of memory']

DS, ES ↕ MOV DS, @DATA

CSE 2109

BP Sir

12.02.2018

4 bit → 4004 (1971) → 6800 → 6808

8 bit → 8008, 8080, 8085

16 bit → 8086, 8088, 80186 - 286.

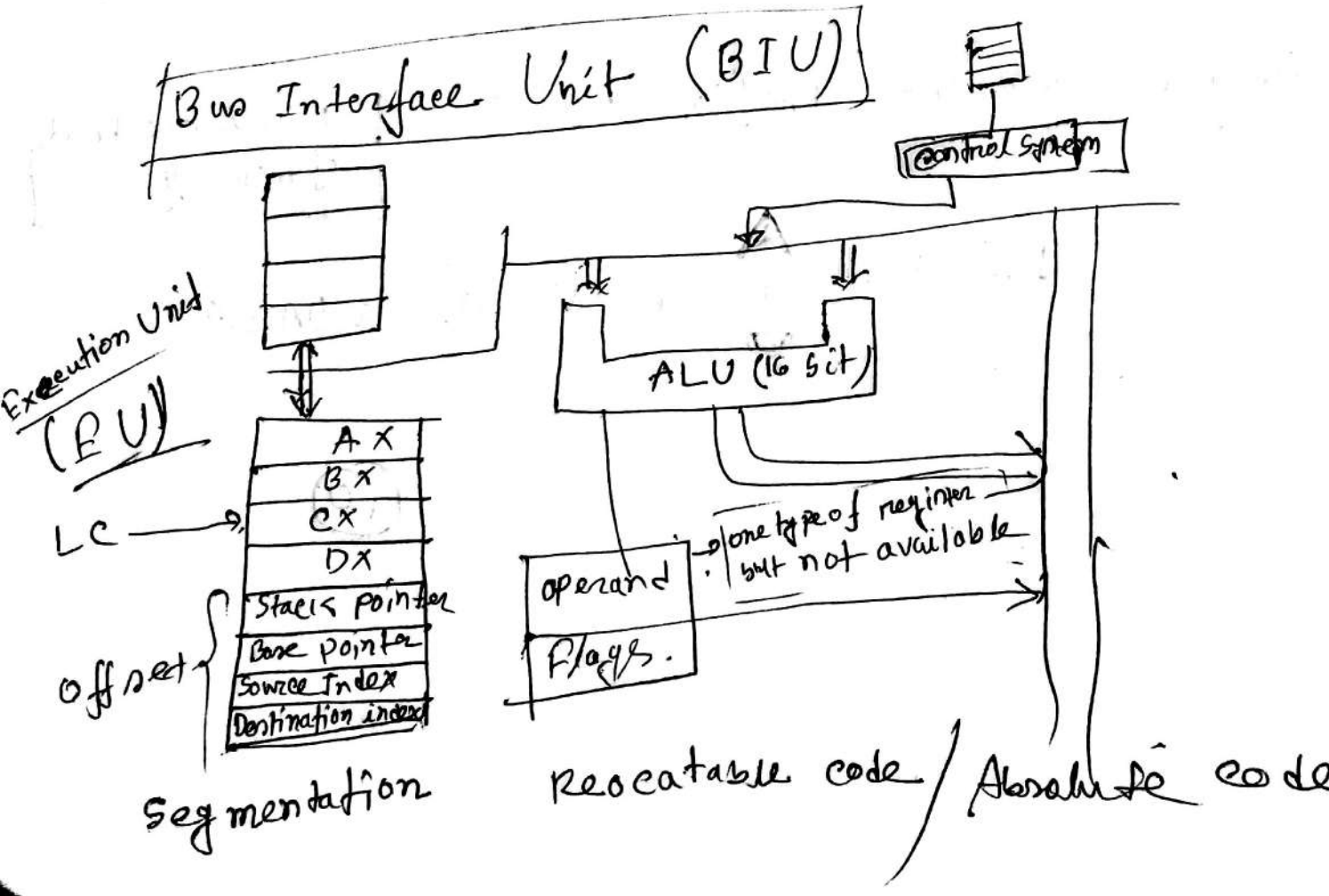
32 bit → 80386, 486.

64 bit → pentium, PI → IV,
core 2DVO, Dual core,
xeon, core i3 - i7. → Intel

→ power pc, G3, G4, G5.

x86 → 8086 architecture based microprocessor

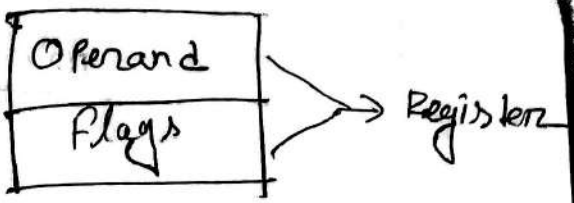
motorola



Segmentation

Relocatable code / Absolute code

ADD Ax, BX.
Opcode Operand



Stack Segment : Stack pointer / Base pointer.
Extra Segment (ES) : Destination Index (DI).
Data Segment (DS) : Source Index (SI)

Assignment : Relocatable Code.
Absolute machine Code
} Assembler

203

CSE — 3109

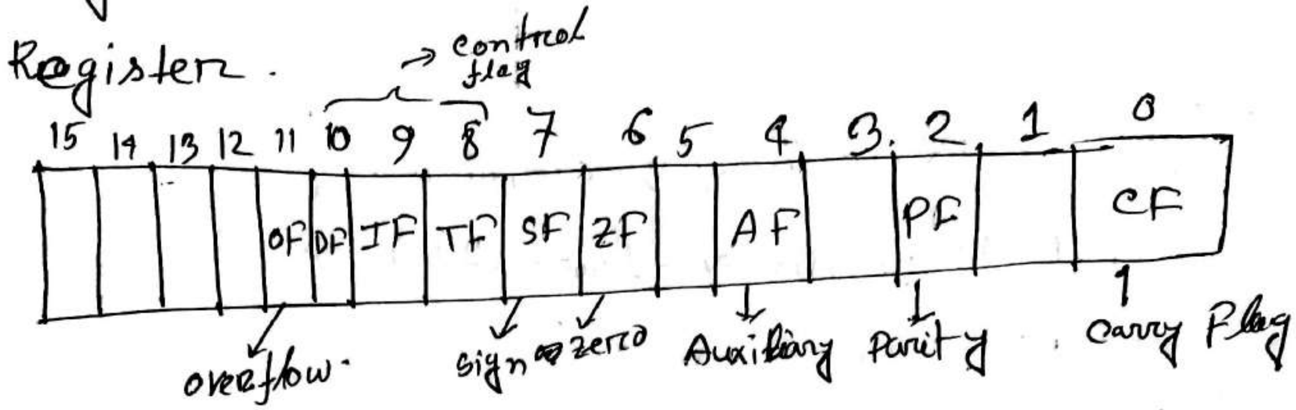
BP Sir

3C day

14.02.2018

Flag

Register.



CF → If carry out from msb or borrow into msb

PF → Even no. of 1 in low byte.

AF → If carry out from 3rd bit or borrow into 3rd bit.

ZF → If zero result.

SF → If sign bit 1.

8 – 10 → Control Flag.

0–7 & 11–15 → Status Flag.

** CF এর ক্ষেত্রে msb থেকে শুরু হয় but AF এর ক্ষেত্রে শুরু করে 3rd bit দিয়ে এটা হল এদের basic difference.

AX = 1111 1111 1111 1110 (FFFE)

BX = 1111 1110 1111 1111 (FEFF)

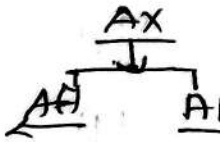
(1) 1111 1110 1111 1101 (1FFFD)

MOV AX, FFFE

MOV BX, FEFF

ADD AX, BX.

CF = 1.
PF = 0
AF = 1
ZF = 0
SF = 1



CT-1 : Chapter - 3, 4, 5

BLSI
APGS

Jump Instructions:

Conditional Jump instruction

unsigned representation

JA	→	CF = 0 , ZF = 0
JB	→	CF = 0
JAE	→	CF = 1.
JBE	→	CF = 1 , ZF = 1

JMP destination Label.

CMP D, S

SUB AX, 10 ≡ CMP AX, 10

JA mypos; JA --

JMP destination Label
↳ jump to this label

↳ need not to be in same segment.

exit

If we want to print character from A to Z.

- CODE

```
MOV DL, 65
PRINT:
MOV AH, 2
INT 21h
ADD DL, 1
JMP PRINT
```

A → 65

Level

```
CMP DL, 75
JA exit
exit:
MOV AH, 4CH
INT 21h
MAIN ENDP
END MAIN
```

Compare Two character

```
MOV AH, 1
INT 21h
AL ← C1
MOV BL, AL
INT 21h
AL ← C2
MOV AH, 2
CMP AL, BL
JA printit
MOV DL, BL
JMP PRINT:
```

```
printit:
MOV DL, AL
INT 21h
PRINT:
INT 21h
```

Unsigned

11 → 3
01 → 1
100 → 4

} memory তে থাকবে

Signed

11 → −1
01 → 1
00 → 0

} memory তে থাকবে না,

✓ Unsigned overflow ← CF = 1

CF = carry flag

✓ Signed overflow

7FFF → 0111 1111 1111 1111
7FFF → 0111 1111 1111 1111

Signed

0111 → 7
0111 → 7
1110 → 14

↙

Signed overflow

Signed bit

↓

range, −8 → 7.

✓ OF = 1 → when signed of overflow occurs

OF = 0 → when signed overflow ! occur (not occur)

msb_in ≠ msb_out → OF = 1

msb_in = msb_out → OF = 0

JA (Unsigned) JG / JNLE
JGE / JNL
JL / JNGE
JLE / JNG
} Signed Jump.

ASCII

JC → jump if carry. ← CF = 1.

If CF = 1 or not, then we can check it by JC

AX = 7FFF
BX = 8000

CMP AX, BX
JA label-big

If Ax is bigger

JG label-big

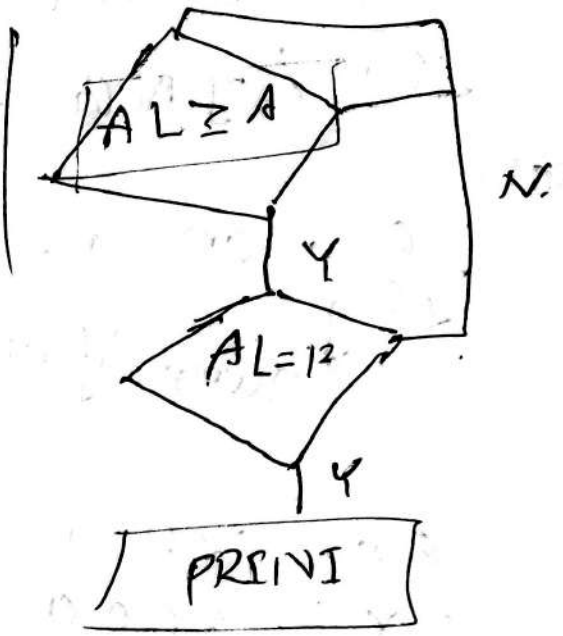
If signed

ASCII → 27 Extended ASCII → 28

• CODE.

```
MOV   AX, @ DATA
MOV   DS, AX
Try_again:
MOV   AH, 1
INT   21h ; Scan

CMP   AL, 'A'
JGE   chk-2.
JMP   try_again
```




```
chk-2:
  CMP  AL, 'P'
  JLE  cPRINT.
  JMP  Try_again
cPRINT:
  MOV  AH, 2
  MOV  DL, AL / '1'
  INT  21h
  SUB  BL, 'A'
  MOV  AL, BL
  INT  21h.
```

MOV BL, AL ←

LOOP

Label:

Loop Label;

mov CX, 10

```
MOV AL,'0'
TOP:-
MOV AH,2
INT 21h
INC AL
JMP TOP
CMP AL,'9'
JGE out
JMP TOP
```

Chapter 6 → The End

8086 → Architecture → BIU, EU → already

PIN DIAGRAM → must read

101 CSE-3109 5 A day

BP 27.02.20

AND destination, source. } Mask / Filteri
OR destination, source }
XOR destination, source

SHL destination, CL
SHR destination, CL

NOT destination.

OR 1101 → AL.
OR AL, 02h

0000 1101
0000 0010
2
→ mask

XOR

1 0
1 0
0 0

MOV AL, 02h
XOR AL, AL

AND

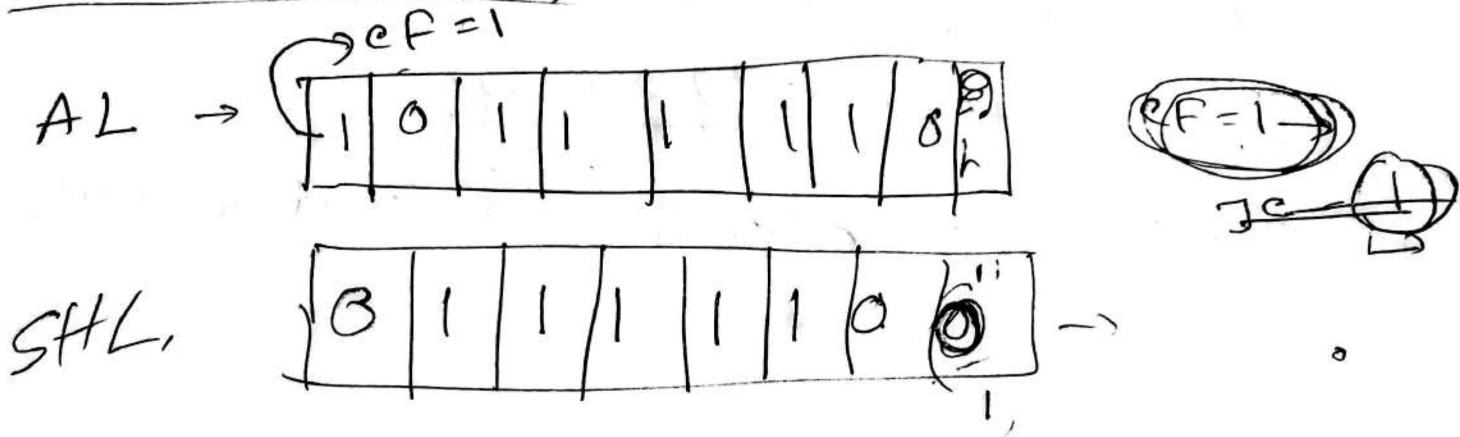
'a' → 0100 0001 → 41h
0010 0000 → 20h

'A' → 0110 0001 → 61h

OR AL, 20h

(i) No carry bit
(ii) No overflow
} masking's Advantages

SHL (Shift left)



JC → Jump in carry → এর মাধ্যমে আমরা
জানতে পারি যে CF এ কি আছে।

Not

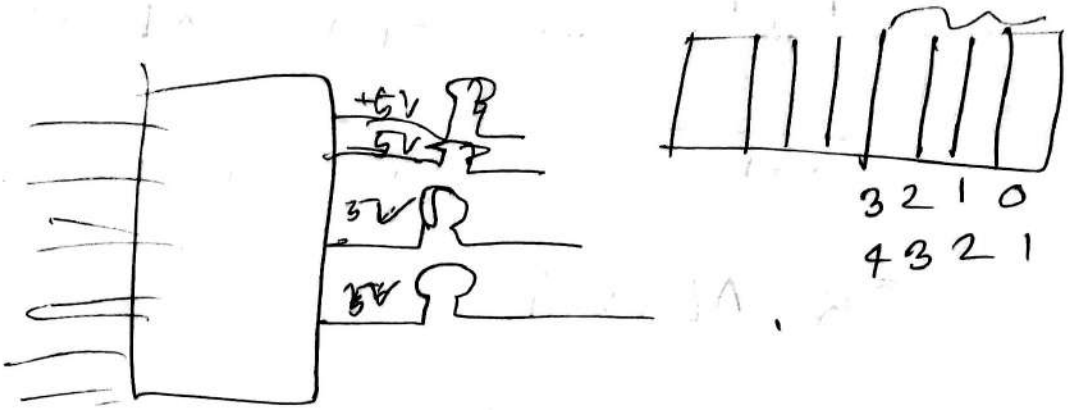
1's complement

0 11111 00
→ 1 00000 11

ODD
even

AND, D

3 0000 00
00



MOV AL, 0Fh

Assignment :- → Can we run/execute a 64 bit software / program in a 32 bit CPU? Why?

উত্তর: [illegible] 64 bit এ চালানো যাবে,

Controll Unit

→ Control Bus
→ Hard ware controlled
→ Micro program Control.

} controll unit দুইপ্রকার control করে থাকে।

RISC Architecture

RISC & CISC Architecture

RISC: Reduced Instruction Set computer.
CISC: Complex Instruction Set computer.

→ Microprocessor design এর concept এই দুইটি।

ISA : Instruction Set Architecture → Microprocessor এর জন্য Vital.

MOV AX, Var1
(AX → register)
→ memory → Single cycle

10	20
1234A	1234B

→ Address.

LOAD AX, [1234A]
LOAD BX, [1234B]
ADD AX, BX
STORE [1234

} MOV
RISC

≈ → multiple ~~cycle~~ cycle

ADDITION [1234A], [1234B]

↳ CISC (Complex)

আমাদের 4টা step আবার এই common। step কে কমিয়ে করতে পারি

RISC → Hardware use করে থাকে

CISC → Microprogram use করে থাকে।

MULTI~~P~~PLICATION

IF : Instruction Fetch

ID : Instruction Decode

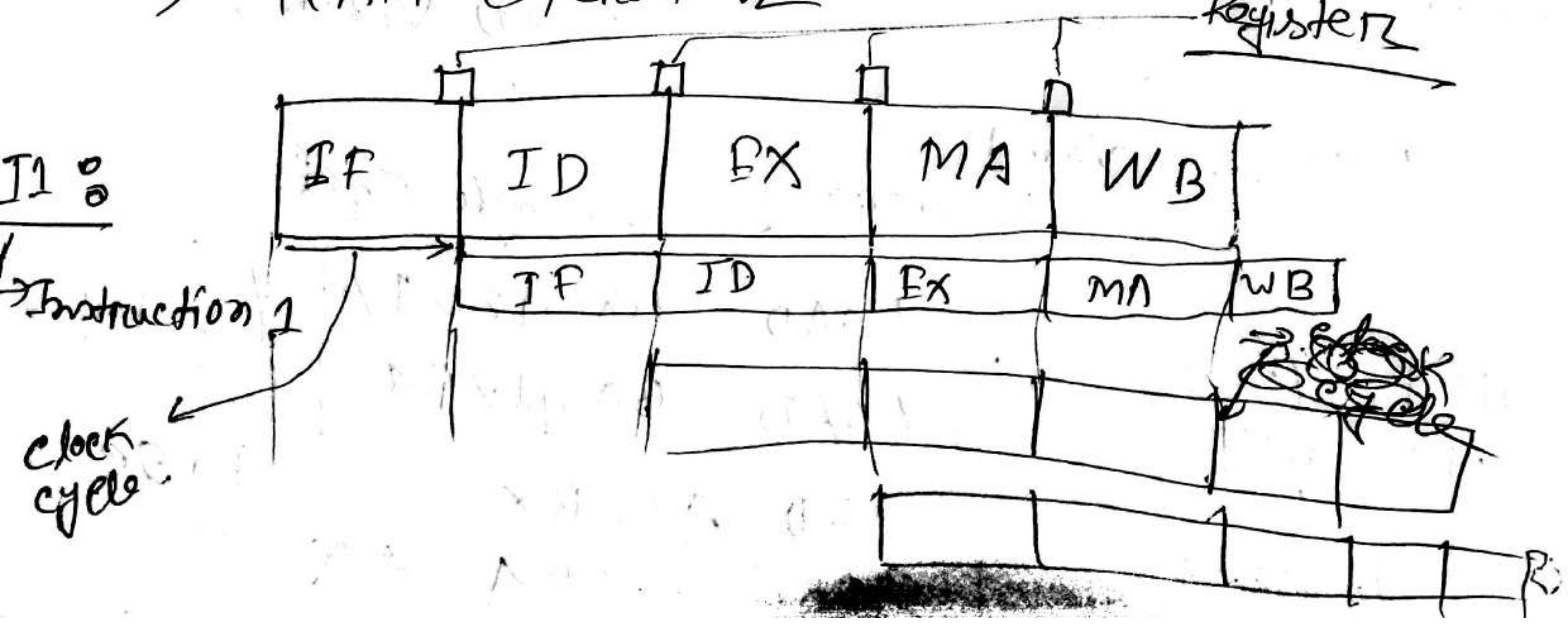
Ex : ~~Execute~~ ex Execute.

MA : Memory Access

WB : Write Block.

Pipelining

→ RAM Operation



Scanned by CamScanner

→ একটি ~~program~~ operation যখন একটি step থেকে অন্য step এ চলে যায় তখন, অনেক stage এ অনেকটি ~~program~~ operation run হওয়াকে বলা হলো Pipe lining. / And এই operation এ RISC better

LOAD AX, M
STORE M, AX.
ADD AX, BX

CISC-এ Function এর সংখ্যা বেশি & code এর Variation বেশি
RISC— Simple circuit দিয়ে parallel operation হয়

Power Consumption

RISC: Power consumption কম কারণ Single cycle.
~~CISC~~
CISC: power consumption বেশি কারণ instruction design complex / multiple cycle প্রয়োজন হয় তাই।

RAM ↓ memory operation

RISC → RAM এর সাথে interaction বেশি তাই memory operation বেশি (কারণ memory থেকে instruction load করতে হয়)
CISC → memory operation কম।

Which is better?

Time = I C P T_c → clock Period time (S)
- C → clk Period/cycle
- P → cycle/Instruction.
- I → No. of instruction/Program

I → 31h

0011 0001

0000 0001

AL → 1

0000 0001

0011 0010

161

CSE - 3109

BP Sir

5 D day

07.03.2018

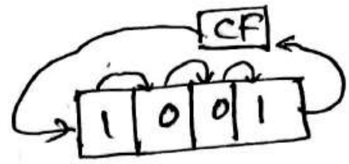
CT-1 → FLAG Register পর্যন্ত ।

ROR → Rotated Right →

ROR AX,CL

ROL → Rotated Left

RCR → Right →



যেমন last data at first CF-এ entry করে then CF এর যে value সেটা first-এ যাবে ।

RCL → Left →

Binary bit String Input

DX ← 10111100

Register 16 bit, সেক্ষেত্রে আমরা সর্বোচ্চ 16 bit input নিতে পারব

→ Data Input Loop এ নিতে হবে ।

```
# XOR DL, DL
  → MOV AH, 1
    INT 21h
→ 8times
  AND AL, 01h
  SHL DL, 1
  OR DL, AL
```

AL এর মধ্যে যেকোন Character এখানে 4 bit হবে সাথে 8 bit হবে ।

AL = 00 000001

31 h

0011 0001
AND 0000 0001
0000 0001 → 01h .

DX ← 1011

DL 0000 0000
AL 0000 0001
OR
0000 0001

Hexadecimal Input

```
MOV CL, 4
XOR DL, DL
MOV AH, 1
INT 21h
AND AL, 0Fh
SHL DL, CL
OR DL, AL
```

9192
9 → Input
3 9
AL = 0011 1001
AND 0000 1111 ←
0000 1001

DL 0000 0000
OR AL 0000 1001
DL 0000 1001

DL-এ চার বার 6ঃ Shift করতে হবে তারপর next Hex input নিতে হবে

repeat করার যদি number 0-9 এর মধ্যে হয় তাহলে

```
MOV AH, 1
INT 21h
SUB AL, 37h
SHL DL, CL
OR DL, AL
```

OABAC h
A = 65d = 41h
4 1 h
0100 0001
SUB → 0011 0111 → 37h
SUB
0000 1010

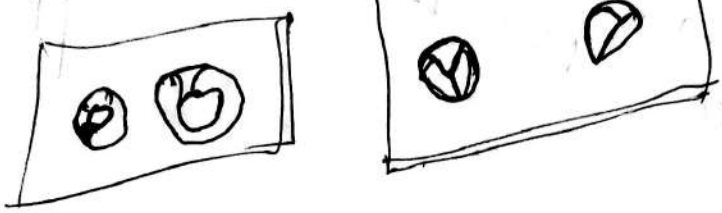
Here, A = 65 d = 41h
A = 10 বার হইলে 55 d = 37 h
Subtract করতে হবে,

41h
37h
Sub
10 d

100

1110
0010
0000

1111
0101
0100



31

DL = 10, 10

XOR DL, DL

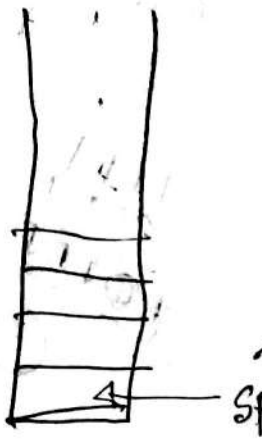
00

DL = 00

CSE-3109
BP Sir

7C day
25.03.2018

Stack operation



→ stack pointer.
16 bit operation.

PUSH source
PUSHF → Flag register.
POP destination
POP F

→ Stack এর সব operation 16 bit operation.
→ register এর মাধ্যমে অথবা PUSH করে stack-এ data.
→ PUSHF - কোন কিছু না বলা অবস্থায় flag এর মাধ্যমে push করব,

```
MAIN PROC
        } user define function এর body.
MAIN ENDP
```

```
name PROC
        } user define function.
RET
name ENDP
```

myfunc PROC
:
RET
myfunc ENDP

User define function.

parameter passing ?

In Assembly

CALL myfunc

In C programming

```
f(int a, int b){
    printf("%d %d", a, b);
}
main(){
    int a, b;
    f(a,b);
}
```

কোনো fn থেকে parameter call করার জন্য আমরা CALL use করি,

```
main PROC :
    popF;
    PUSHF
    CALL myfunc
```

CALL ইন্সট্রাক্টর দ্বারা mainfn এর data change হয়ে যেতে পারে CALL এর আগে PUSHF & main PROC এর শেষে POPF use করি।

MULTIPLICATION

MUL Source → Reg
IMUL Source. → AX = Source × AL
DX : AX = Source × AX

↳ Signed operation

যখন 32 bit হবে তখন DX : AX মে রেজি আপনে 16bit DX এ থাকে এবং 16 bit, AX এ থাকে।

$2^{16} - 1 = 65535$ → AX
By Source = 2 × AX.
DX : AX

[যে check করার জন্য Overflow flag (OF) check করি, DX : AX এ. কত bit থাকবে সেজন্য!]

16 16
32

AX :
AX
AX

BX :
BX
BX

D
DX : AX
16 16

203 CSE - 3109 8th day

BP sir 02.04.2018

CT#2 solve

$100000 \times 5\% \times c_g \times t_g + 100000 \times (100-20\%) \times C_c \times$

$100000 \times \frac{80}{100} \times 8 \times 100 \times 10^{-9} + 100000 \times \frac{20}{100} \times 6 \times 100 \times$

$= 64 \times 10^{-6}$

$= 64 \mu s$.

Decimal Input

0 - 9

134

$0 \times 10 + 1 = 1$

$1 \times 10 + 3 = 13$

$13 \times 10 + 4 = 134$

AX, BX, DX

→ Input নিব

→ Input টা AL এ save হবে,

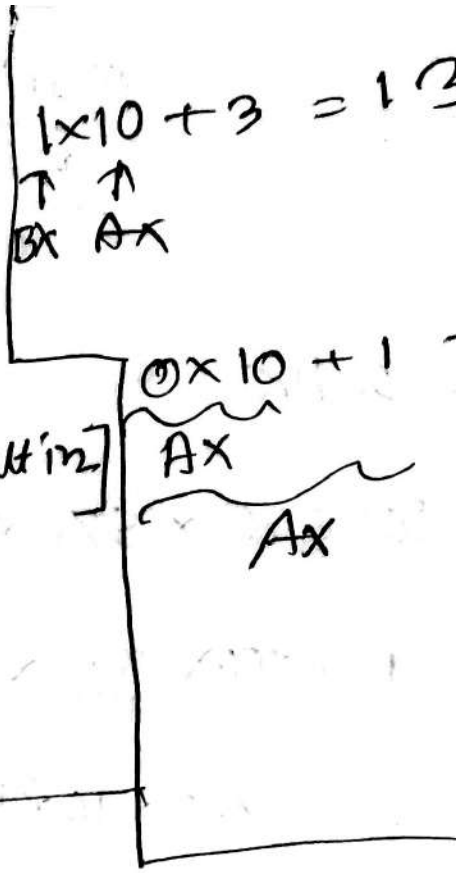
→ AX এ input কোনো আগের গুলোর সাথে Add করবো।

XOR BX, BX

Input: AND AX, 000Fh → Input: AH, 2 INT 21h

MOV DX, AX ; [1 (input)]

```
MOV AX, 10
MUL BX ; [Result in AX]
ADD AX, DX  [Result in AX]
MOV BX, AX  [First step result in BX]
JMP Input
```



Stack

```
ADD AX, DX
MOV DX, AX
```

=

```
PUSH AX
```

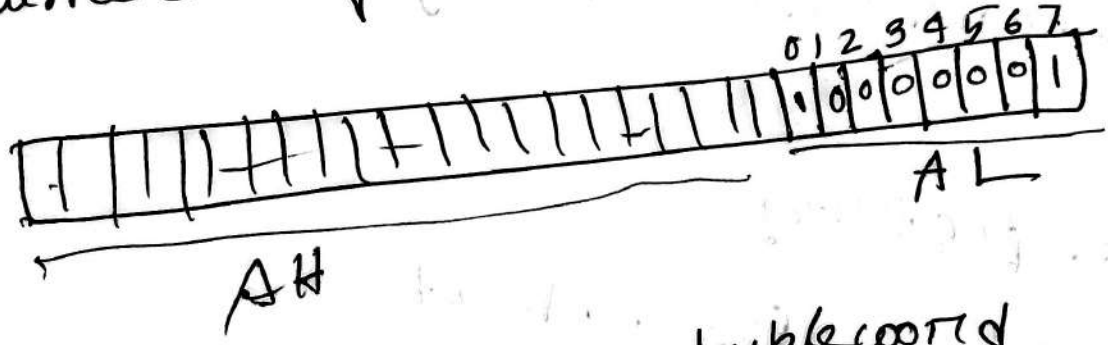
DIV Divisor

IDIV Divisor

divisor) AX (AL
AH

divisor) DX : AX (AX
DX

Remainder's Sign = dividend's Sign.



CWD → word - doubleword.
CBW → Byte-word

```
MOV  AL, -5
CBW
MOV  BL, 2
IDIV BL
```

ADDRESSING MODE

Register indirect

High Level Language

```
int * p;
p = &a;
int [10];
Printf("%d", *p);
("%d", p);
*[2]
```

→ Register contains the address of some memory.

MOV AX, [Register]

→ DS:SI ES:DI

SI → 16 bit
DI

My array DW 10, 20, 30, 40 ;
A, B, C, D

→ When we need to print one value then we move it into DL and print it.

```
mov AH, 2
LEA SI, myarray
MOV DL, [SI]
INT 21h
```

10, 20, 30, 40
↑
DS:DI
point to

ADD SI, 2

Loop print:

Based / Indexed.

~~Addre~~ Addressing mode.

[offset + displacement]

offset[displacement]

✓ mov AX, 0

mov DL, myarray [AX]

ADD AX, 2

204 CSE-3109 9th day

BP স্যার 07.04.2018

→ Based Add. mode

→ Indexed Add. mode

→ Based Indexed Add. mode

✓ Register + Displacement

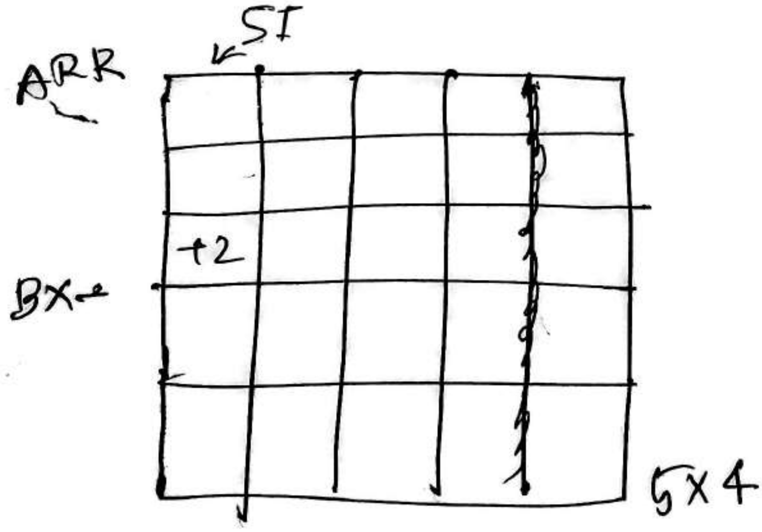
Base/I

Register হতে পারে Base add & তাকে I করে দিতে কি Type এর Add.

→ DW

✓ ARR [R] [C]

↳ 2 dimensional এর array এর ক্ষেত্রে আমরা Based Indexed Add. mode use করব।



ARR content +2 থেকে (Suppose) Array তে data কে input দিব।

BX কে Row ধরব & SI কে Column হিসেবে নিব।

Scanned by CamScanner

```
MOV CX, 3
XOR SI, SI
MOV BX, 16
ADD ARR [BX] [SI], CX
ADD SI, 2
```

→ DW → word type array

ARR [R] [C]

0 2 4 6
8 10 12 14

BX → 16

LABEL [just like C program's structure not totally]

```
STUDENT LABEL WORD
ROLL    15300
MARKS   DB 90
MOV AX, STUDENT
```

16 bit [1 byte = 2 byte = 16 bit]

AX → 16 bit → word type
Student → word type

→ AH → AL.

~~MOV AX, ST~~
MOV BH, ROLL →

উপরে যেকোন declare করেছি সেগুলোকে আমরা যে line মে করতে পারব, নাহলে not

PTR → Byte pointer → Keyword

```
MOV BYTE PTR [BX], 10
WORD
MSG DB 'RUET'
```

16 bit & 8 bit

✓ MOV AX, BL
এগুলো করা operation এ সমস্যা হবে bit not same

✓ MOV [SI], 2
PTR → just like type casting in C

LEA SI, MSG

→ SI ও msg এর address
এর assign হয়েছে

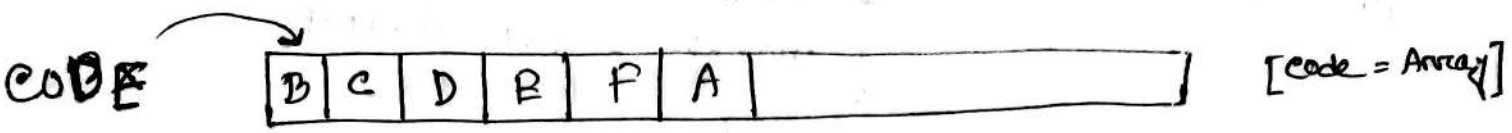
MOV WORD PTR [BX], 10
↳ word type

CSE - 3109
BIT 5TH

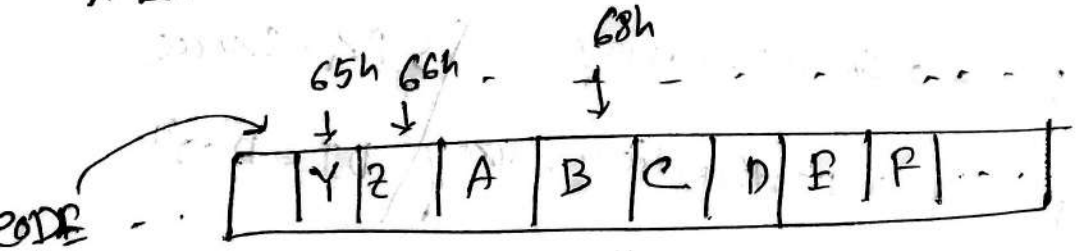
9C day
16.09.2018

XLAT → Translate করার জন্য ব্যবহার হয়।

BX ← table index [যে table থেকে value কে convert হবে]
AL ← Byte value to be converted.
XLAT [BX & AL এর value কে কিসের base এর উপর convert করবে]




```
LEA BX, CODE
MOV AL, 01h
XLAT
```




```
LEA BX, CODE
LEA SI, MSG
MOV AL, [SI]
XLAT
MOV DL, AL
MOV AH, 2
INT 21h
INC SI
```

→ BAD

~~INC SI~~.

Duplicate বুঝায়

CODE DB 65 DUP (' '), Y Z AB - -

String

✓ MOV SB/W

value move করে MOVSB এর কাজ
MOVSB → For Byte
MOVSW → For word.

ES : DI → Destination Array

DS : SI → Source Array

S | R | V | F | T | → Source Array

D | | | | | → Destination Array

SI ← Source
DI ← Des"

SI

✓ MOV AX @ DATA
MOV ES, AX
CLD
LEA SI, S
LEA DI, D
MOVSB
~~MOV~~ MOVSB

MOV SB

MOV SB

LOD SB/W → DS

STOSB/W → for string type AL/AX DS:SI.

LODS B/W → AL/Ax DS:SI

(memory to microprocessor)

STOS B/W → ES:DI AL/AX

(microprocessor থেকে memory তে)

DF → Direction flag

DF → 0 SI / DI এর value কমে যাবে

DF ← 1 SI / DI এর value বাড়তে থাকবে

DF ⇔ 0

⟶ CLD মনে রাখা

DF ← 1

↳ STD মনে রাখা

|0|

CSE-3109
BP sir

৩০ তম
17.09.2018

✓ REPE
✓ REPNZ

String operation
✓ SCASB / W
✓ CMPSB / W

A	B	C	D

SCASB → String কে check করব যেখানে এ একটি particular character আছে কিনা ?

AL / AX
↳ C

✓ CMPSB → Compare করা, SCASB একটা করে এক এর উপর operation করে কিন্তু CMPSB সম্পূর্ণ Array এর উপর operation করে,

DS : SI .

C	D

CT-3 (10A)
'Stack এর মাধ্যমে String করিস।'

REPE → (Repeat while equal)

REPNZ → (Repeat while not zero).

CX, 4 → control করার জন্য use করা হয়

✓ before we use movsb many times but it is not needed while we use REPE, it's work like multiple movsb command. REPE will controll with CX.

Interupt

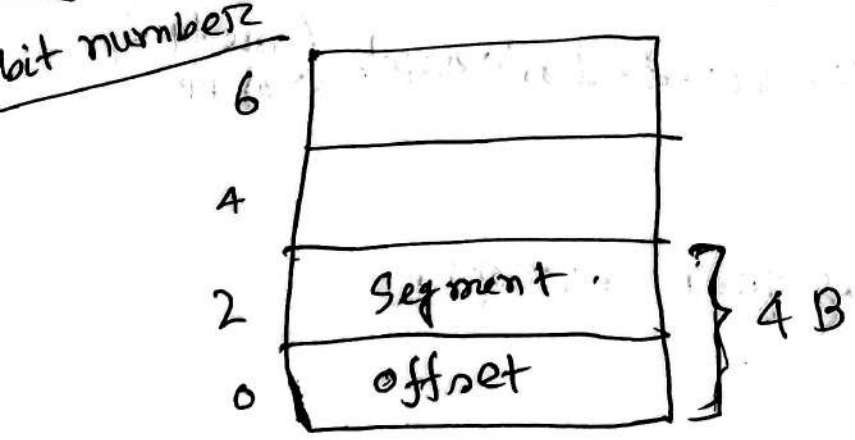
H→ Software INT.

→ Hardware INT

↳ NMI Int.

→ Exception .(processor exception)

Interupt Vector Table



$2^8 \times 4B \longrightarrow 1KB$.

Interupt

- DOS
- BIOS

→ ~~At f~~ মূলত Int 21h দিলে সে vector table এ 21 এ যাবে সেখানে যে address আছে এই address এ গিয়ে এখানের function কে actually point করে & সেই function কে run করে।

→ Processor's Interupt Process

INTR → Interupt request (microprocessor Pin)

INTA → (Interup Acknowledge) [INTR execute হওয়ার পর INTA এ signal দিবে & তারপর INTA execute হবে।]

INTA এ থেকে যে number আসে সেই number এ যাবে & সেখানের operation করবে।

NMI → Non Muskable ~~Input~~ Interupt (one type of pin)

DF
TF
IF

| flag গুলো কাজ করে।

Interupt process এ প্রথমে TP & IF কে zero করে দেয়।

203 CSE-3109 BP 10 day 23.09.2018

Timing of 8086 for read/write cycle

Max mode
Min mode } ৪086 এর দুই mode এ কাজ করে।

Co-processor ⟵ ~~Min~~ Max (multiprocessor)

8086 এর জন্য co-processor = 8087

8087 → Math Co-processor.

→ Block Diagram

Latch
Trans receivers
clock generator
Memory
I/O

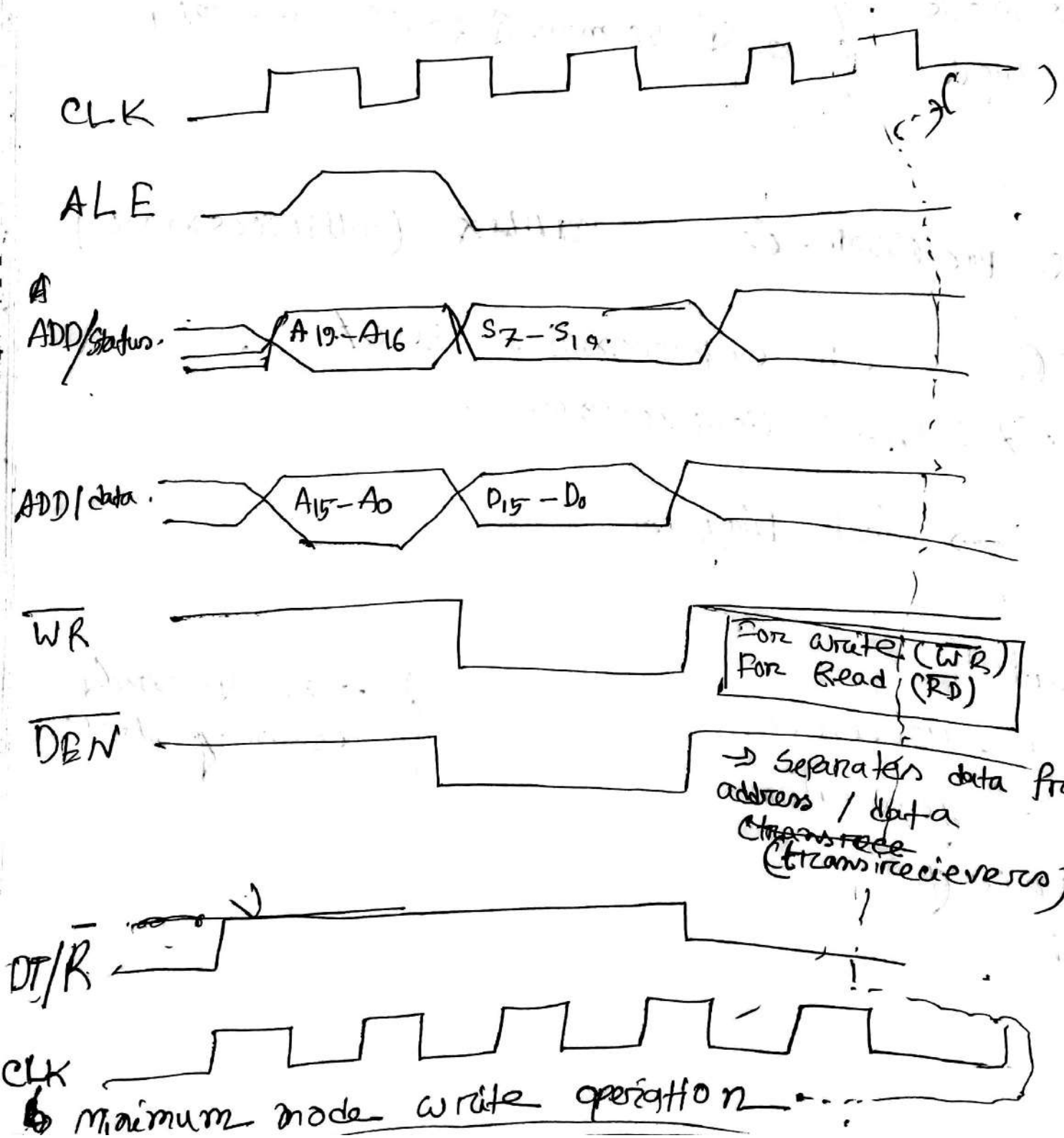
} 8086 এর কাজের necessary blocks.

3. Latch
   Transreceivers
   clock Generator
   Microprocessor – ৩টি কাজ করে বা ৩টি কি? প্রয়োজনীয়

for Microprocessor ৫ ?

Why there 3 are necessary in microprocessor?

✓ Submission date : 12.B day



Maximum mode write operation ...

✓ If Maximum mode এর operation Home work

✓ WRite এর জন্য same but $\overline{WR}$

✓ Read এর জন্য same but $\overline{RD}$

✓ Min mode → Microprocessor নিজে কন্ট্রোল করে।

✓ Max mode → অন্যান্য chip এর সাহায্য নিয়ে কন্ট্রোল করে।

→ DMA operation

DMA = Direct Memory Access.

→ SRAM
→ DRAM
} Internet :

CSE-3109

BP

10D day
24.04.2018

- Introduction to Assembly langua
- logic and Arithmetic operation,
- flag and flow control
- Array
- string manipulation
- Introduction to diff-type of micro processor
- organization of 8086 microprocessors
- component of micro computer es
- I/O device
- Phoriferal interfacing (I/O)
- DMA controller (chip). (8237)
- @ Interrupt controller
- Interupt structure.
- Interupt related PIN
- Co-Processor
- RISC & CISC
- Direct Video RAM Accessing
- memory Audio