TOPIC NAME : W-1 DAY : L-1

TIME : DATE : 4 / 12 / 24

Mark Distribution

Attendance + Class Performance → 20

Online (11) + Offline (11) → ( ) will be provided later

Mid (written) → ( )

Final (code) → ( )

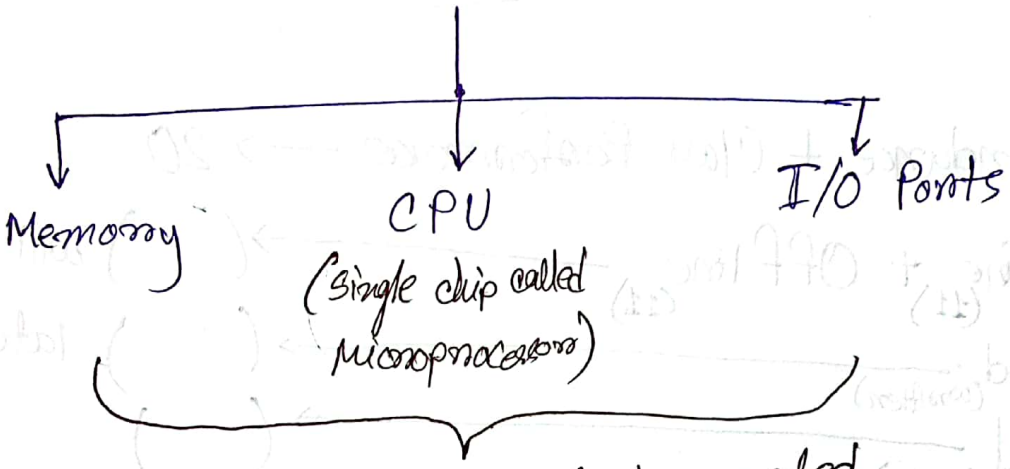
Offline:

- Cover Page will be provided
- Write from book
  -- in your own way
- Both hard copy and soft copy
  -- hard copy will be hand written

GOOD LUCK

TOPIC NAME : _____ DAY : _____ TIME : DATE : / /

Micro computer



Memory:

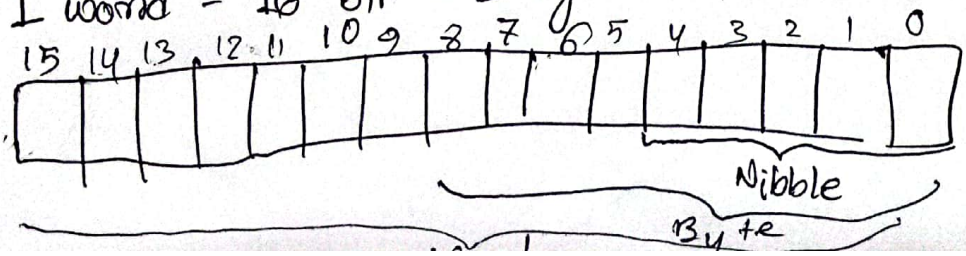
- stores one bit
- 8 bits make a byte
- first memory byte is stored in 0
- address is fixed, content is not
  - -- Address is bits
  - -- Content is bytes

MSB changes ~~most~~ most

LSB changes last

- Operations:
  - ~~to~~ Read / Fetch
  - + write / Add

- 1 word = 16 bit = 2 byte = 4 nibble



GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

- Bus : ——→ Connection
  - → Address Bus
  - → Data Bus
  - → Control Bus

- Cache Hit : Memory is found immediately in cache

  " Miss : " " " not " " " "

- 2^{10} → KB

  2^{20} → MB

  2^{30} → GB

  .

  .

GOOD LUCK

TOPIC NAME : _____ DAY : _____ TIME : DATE : / /

CPU :

- Execution Unit
- Bus interface unit
- while Execution Unit performs, BIU fetches 6 bits of the next instruction (in a queue) → Pre fetch
  - ↓ instruction queue

- Execution Unit :
  - ~~Circuit~~ ALU has all the instructions
    - AX : Accumulator } ~~Registers~~ Register
    - BX : Base
    - CX :
    - DX :
    - SI : Source } Index
    - DI : Destination
    - BP : Base } Pointer
    - SP : stack

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

- FLAG register : reflects result of a computation

- BIU :

- connects

- Transmits

- CS
  DS
  ES
  SS
  IP : Instruction Pointer

GOOD LUCK

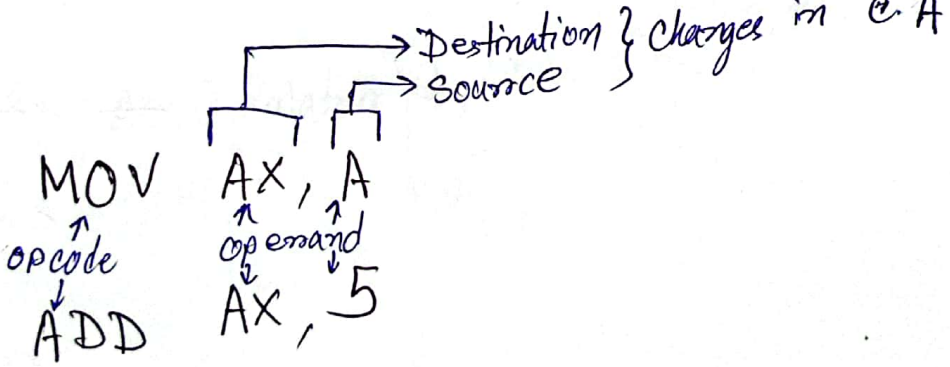
TOPIC NAME : _____ DAY : _____ TIME : _____ DATE : / /

I/O Ports :

- Input → Input Port → Do something
  Output ← Output Port ←

Instruction Execution :

Opcode → instruction
Operand → instructing things



~~Fetch memory Decode~~

Fetch

Fetch instruction → Decode instruction → Fetch data from memory if necessary

Perform Operation → Store the memory if needed

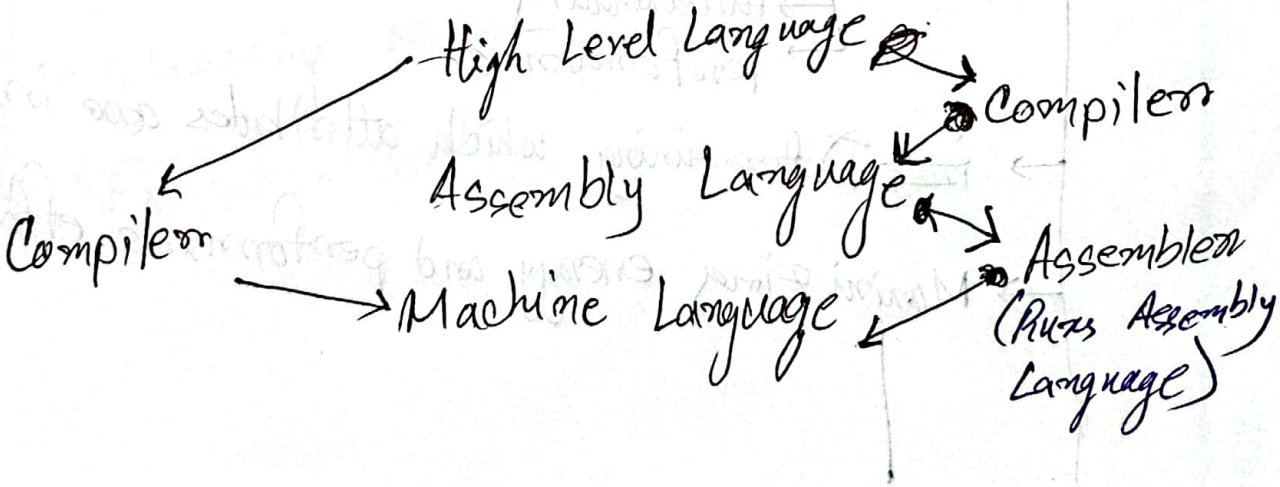
Execution Cycle

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Machine Language
(0/1)
↓
Assembly Language
(Converts Machine language to a more understandable language)
↓
High level language



GOOD LUCK

TOPIC NAME : W-1

DAY : C-1

TIME : 9 DATE : 4 / 12 / 24

Comp Arch

* Classroom : wh4yhad
* "Core courses are important" ~ Miss
* "Fully theoretical . Less Math"
* 4 Quizzes

Computer Architecture:

- Design
  - Functionality
  - performance
- Determining which attributes are important
- Maximizing energy and performance efficiency
- Optimizing design

GOOD LUCK™

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Computer Architecture = ISA + CO

ISA → Instruction Set Architecture

- Interface bet^n Hard and Software
- Instruction set
- " format
- Data storage
- Addressing Modes
- Exceptional Conditions

CO → Computer Organization

- Hardware level design

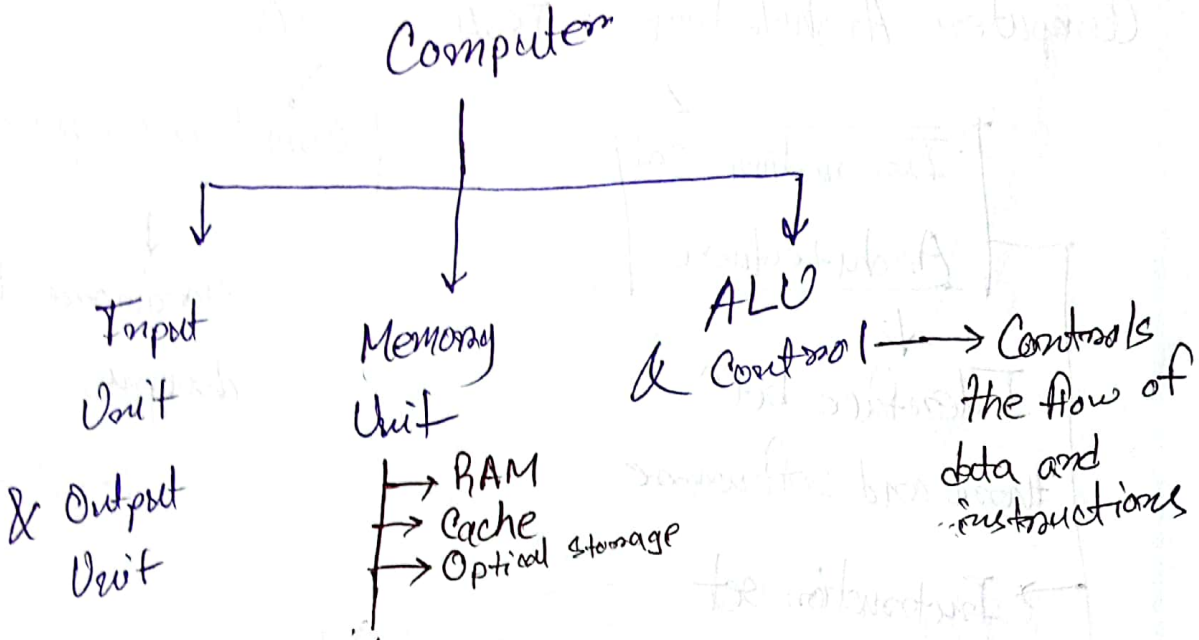
GOOD LUCK™

TOPIC NAME :

DAY :

TIME :

DATE : / /



* I/O unit ≠ I/O device

* Cache is fastest
Main Memory is slowest

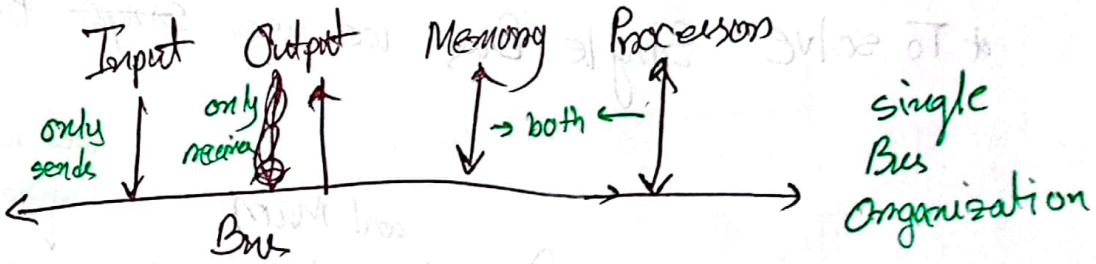
* Immediate use = Register
Frequent " = Cache
Everything else = Main Memory

GOOD LUCK

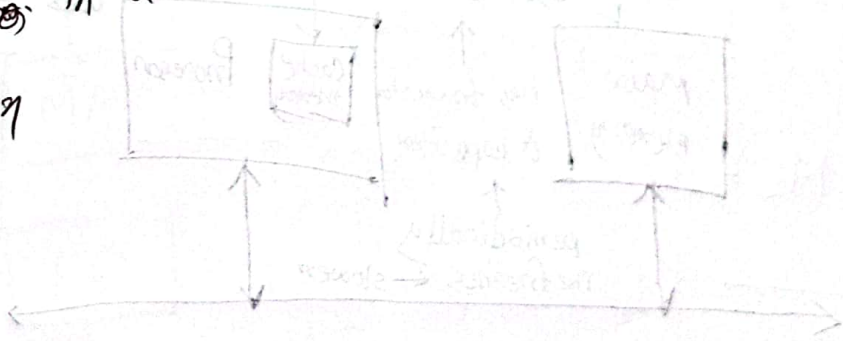
TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Bus: Group of parallel wires



- Each wires in a bus can transfer one bit of information



TOPIC NAME : W-1 DAY : C-2

TIME : DATE : 5 / 12 / 24

Bus Structures:

* To solve Single Bus we use ~~single~~ Registers

Organization of Cache and Main Memory:

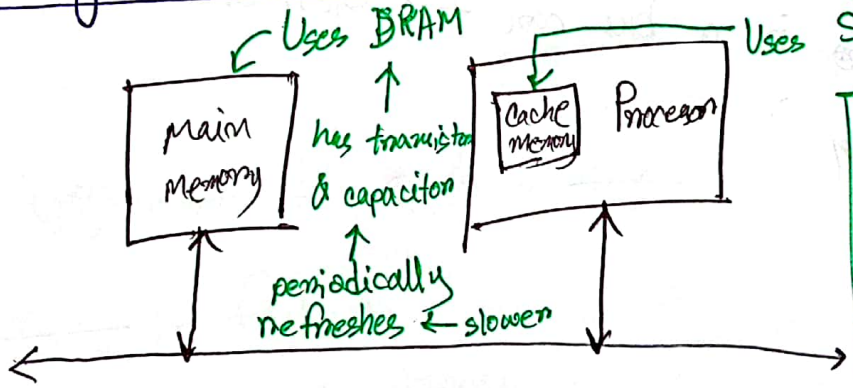
does not periodically refresh ← faster

↓

uses 6 transistors

↓

SRAM



* RAMs contain Memory Cells with Reg which contains 1 bit

Access time of ~~the~~ Cache is less:

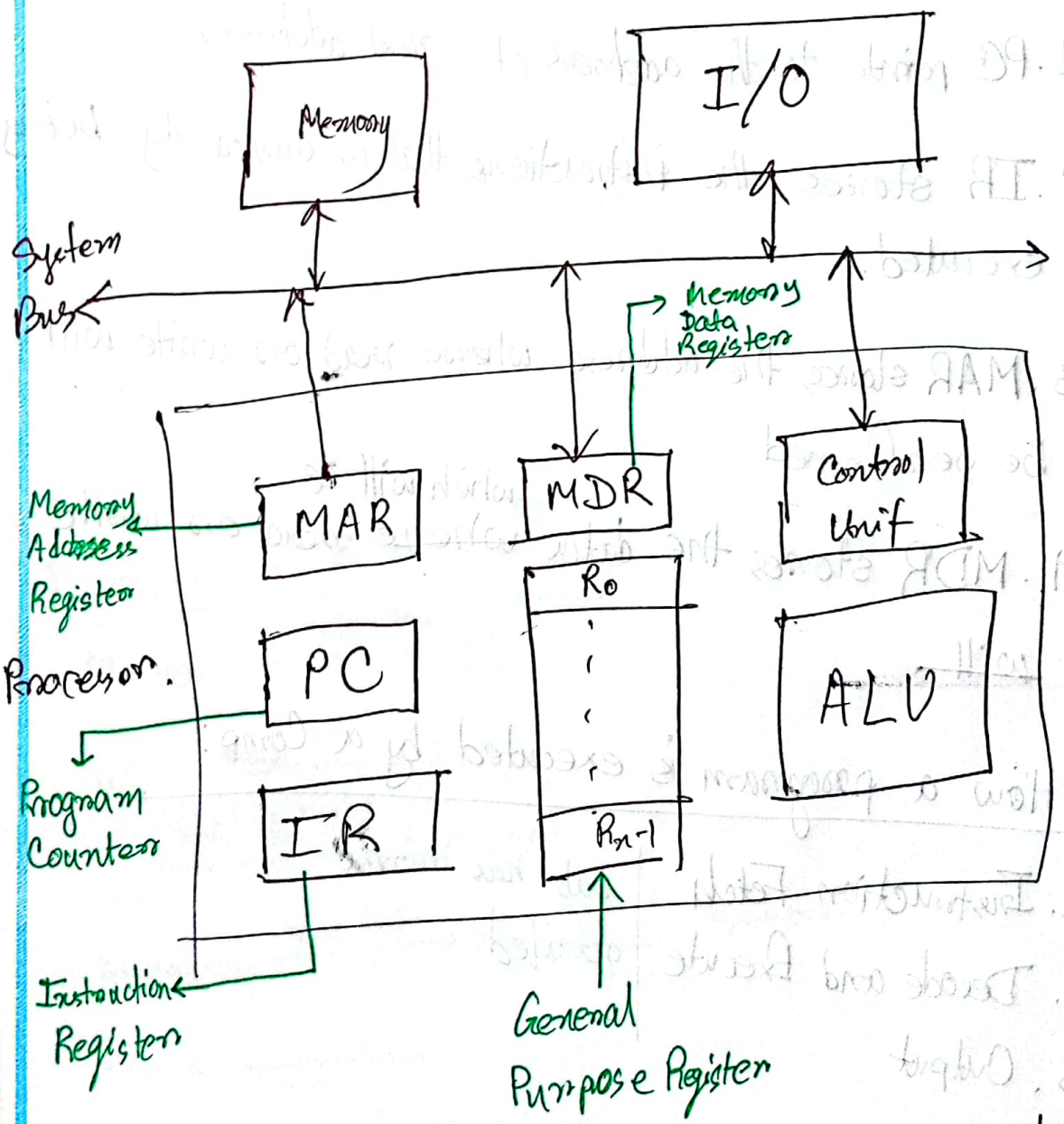
1. Closer to Processor
2. ~~Has~~ Uses SRAM
3. Cache Memory uses pipelining ← will be explained later (last weeks)
4. Size is less and so searching is easier

GOOD LUCK™

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Computer Components : Top Level View :



PC. points to the next instruction to be executed

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Running a Program:

1. PC points to the address of next address
2. IR stores the instructions that is currently being executed.
3. MAR stores the address where read or write will be performed
4. MDR stores the data which will be ~~where~~ read or write ~~will~~

*How a program is executed by a Comp:

1. Instruction Fetch
2. Decode and Execute
3. Output

Lab has more detailed

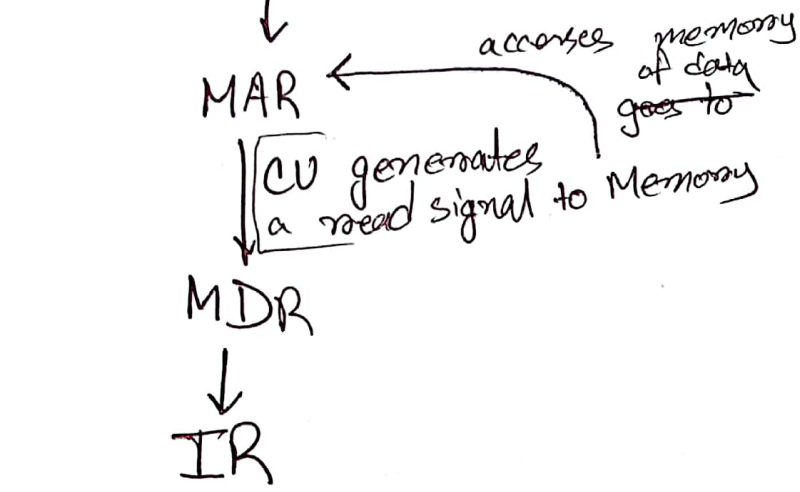
GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Memory has instructions

PC is pointing at instructions



1. Instruction Fetch Cycle:

Memory —instructions→ PC

PC —~~reads~~ sets→ MAR

CU —sends Read signal→ Memory

Memory —reads instr.→ MAR —sends ~~value~~ address→ Memory

~~MAR~~ ⟶

Memory —sends value→ MDR

MDR ⟶ IR

GOOD LUCK™

TOPIC NAME : W-2 DAY : C-3

TIME : DATE : 8 / 12 / 24

~~De~~ 2. Decode & Execute : → happens in IR

i) Decodes the instruction

ii) Memory $\xrightarrow{\text{location}}$ MAR

~~MA~~ CU $\xrightarrow{\text{read signal}}$ ~~MA~~ Memory

MAR $\xrightarrow{\text{data}}$ MDR

Happens when ! register

WMFC → Wait for Memory Function Completion

MDR $\xrightarrow{\text{data}}$ GPR

3. Output :

Store in

→ Register

→ Memory Location

↳ MAR. $\xrightarrow{\text{data}}$ MDR

CU $\xrightarrow{\text{write}}$ Memory

Memory → MAR

~~Memory~~ $\xrightarrow{\text{write}}$ ~~MDR~~

MDR $\xrightarrow{\text{write}}$ Memory

~~WFM~~ WMFC

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

A Typical Instruction :

* MOV LOCA, R0 ← * lab has dest src
theory " src dest

↳ puts value of LOCA in R0

* Load = Memory in register → MOV LOCA, R0
Store = Register " Memory → MOV R0, LOCA

↓

LOAD LOCA, R1
STORE R2, LOCB

* MOV R0, LOCA :

1. Instruction fetch :

i) Memory —instructions for MOV→ PC

PC —sets points at→ MAR

CU —sends read signal for MOV→ Memory

Memory —reads instruction→ MAR —sends address of MOV→ Memory

Memory —sends instruction data→ MDR

MDR —sets instruction→ IR

2. Decode and Execute :

i) Decodes the instruction
ii) Memory —location of LOCA→ MAR

CU —sends read signal for LOCA→ Memory

Memory —reads data→ MAR

MAR —sends data of R0 to LOCA→ Memory

Memory —moves data→ MDR

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

3. Output :

i) ~~MAR~~ ~~data~~ (of LOCA) ~~MDR~~

CU —write signal→ Memory

Memory —writes data→ MAR

MDR —writes in→ LOCA

~~WFM~~ WMFC

i) ~~MAR~~ —sets address→ ~~M~~

* try at home

GOOD LUCK

TOPIC NAME : 10-3

DAY C-5

TIME

DATE 18 / 12 / 24

MULTICOMPUTER & MULTIPROCESSOR

Quiz - 1
on 26th
Chap - 1

M. Processor
↓
One memory
Multiple Processor

M. Computer
↓
Network connected
Communication through
messages

GOOD LUCK

TOPIC NAME _____ DAY : _____

TIME : DATE : / /

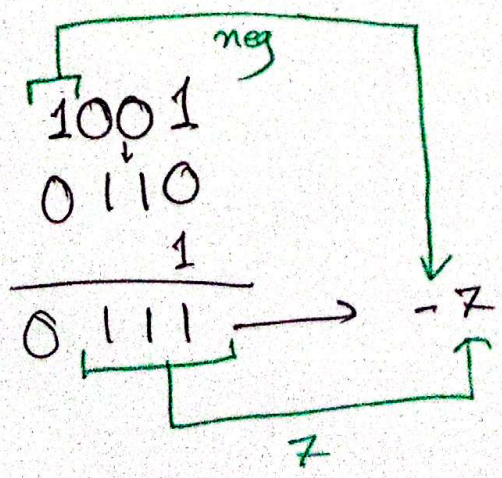
CH-2

- Signed numbers (Numbering system)
-

Number System

-5

1. Sign & Magnitude - 1101
2. 1's complement - 1010 $(0101)_5$
3. 2's complement - 1011



GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

1000

Sign and Magnitude : 1 000 = -0

1's complement : 0 111 = -7

2's " : (0111 + 1) = 1000 = -8

4 - (-2)

4	0100 = 4
-(-2)	0010 =
6	0110

-2 = 0010's 2's complement

= 1101

\+ 1

1110

-(-2) = 1110's 2's complement

= 0001

\+ 1

0010

Self Study: why 2's complement is preferred

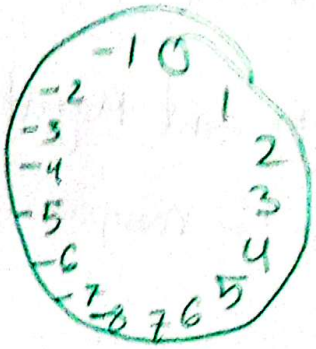
"GOOD LUCK"

TOPIC NAME :

DAY

TIME : DATE : / /

Overflow:

In 2's complement clock, → 

$3 + 5$

$= 0011 + 0101$

$= 1000 = -8$

which also counts as overflow

i) Acquired result above range

ii) Opposite signs after adding same signed numbers

iii) MSB $C_{in} \neq$ MSB C_{out}

TOPIC NAME : W-3 DAY : C-6

TIME : DATE : 18 / 12 / 24

Machine Instructions and Programs

Sign and Mag $\longrightarrow -2^{n-1}+1 \longrightarrow 2^{n-1}-1$

One's comp $\longrightarrow$

Two's comp $\longrightarrow -2^{n-1} \longrightarrow 2^{n-1}-1$

Memory Location Addresses and Operation

① There are 2^n ~~bit~~ addresses that can be accessed

2^{10} = kilobyte

2^{20} = megabyte

2^{30} = gigabyte

GOOD LUCK

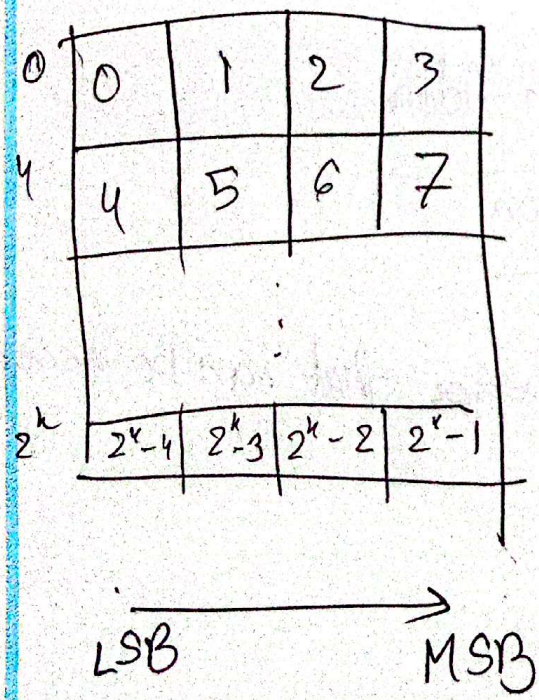
TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

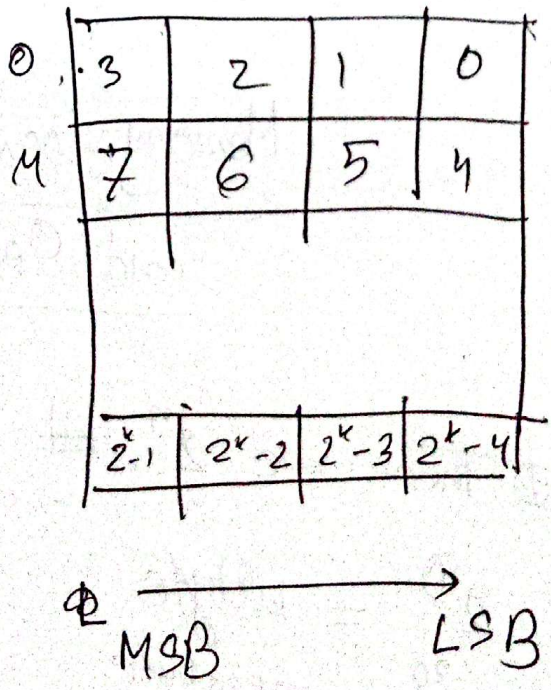
Byte-Addressable:

Each byte is given a definite address This addressing is referred to as byte addressable memory

Big Endian



Little Endian



GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

Word Alignment:

If words are aligned, addresses will be ~~word length~~ ~~they~~ multiple of bytes.

* Register Nam = Word Length

Instruction and Instruction Sequencing

i) Data transfer between memory and registers
ii) Arithmetic and logical ~~Unit~~ Operation
iii) sequencing
iv) I/O Data Transfer

R1 ⟶ address
[R1] ⟶ content

ADD R_1, R_2, R_3
↳ $R_3 \leftarrow [R_1] + [R_2]$

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Basic Instruction Type : $C = A + B$

* Three address instruction : OP A, B, C
* 8086 has 20 address bit
* Two address instruction : Uses 2 address
* One Address " : Uses AC/AX ← is used implicitly

↓

LOAD A ; AC ← M[A]

ADD B ; AC ← [AC] + M[B]

STORE C ; M[C] ← [AC]

GOOD LUCK

TOPIC NAME U-3 DAY C-7

TIME DATE 13/12/24

CPU Organization

based on memory

1. Single Accumulator
   - Uses accumulator
   - Uses AC
2. Registers - Memory
   - Uses at least one register
3. Register - Register / Load-Store
   - ALU can only work on registers
4. Stack CPU

Stack Memory

TOS → Stack Memory

Top of Stack

When ALU uses zero memory instruction, it uses the top 2 data in the stack.

GOOD LUCK

TOPIC NAME : DAY :

TIME : DATE : / /

#

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

Three address instructions :

R1

ADD A, B, R1; M[X] ← [A] + [B]

ADD C, D, R2; R1 ← [C] + [D]

MUL R1, R2, X; M[X] ← [X] * [R1]

Two address :

MOV A, R1 ; R1 ← [A]

MOV C, R2 ; R2 ← [C]

ADD B, R1 ; R1 ← [B] + [R1]

ADD D, R2 ; R2 ← [D] + [R2]

MUL R1, R2 ; R2 ← [R1] * [R2]

MOV R2, X

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

One address :

```
LOAD  A  ;  AC ← [A]
ADD   B  ;  AC ← [A] + [B]
STORE X
LOAD  C  ;  AC ← [C]
ADD   D
MUL   X
STORE X
```

TOPIC NAME : W-4 DAY : C-8

TIME : DATE : 22/12/24

#2000 address : $X = (A+B)^* (C+D)$

PUSH A

8

TE 22/12/24

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

RISC Instruction:

- Can use 3 registers in a single instruction
- Only LOAD and STORE

```
LOAD  A, R1
LOAD  B, R2
LOAD  C, R3
LOAD  D, R4
ADD   A, B  R1, R2, R1
ADD   R3, R4, R3
MUL   R1, R3, R1
STORE R1, X
```

GOOD LUCK

TOPIC NAME

DAY

TIME DATE / /

Instruction Execution:

- Straight Line Sequencing → reads line by line
- Branching → may skip lines if necessary
  - Conditioned Branching → goto inside if-else

Straight Line Sequencing:

Assumptions:

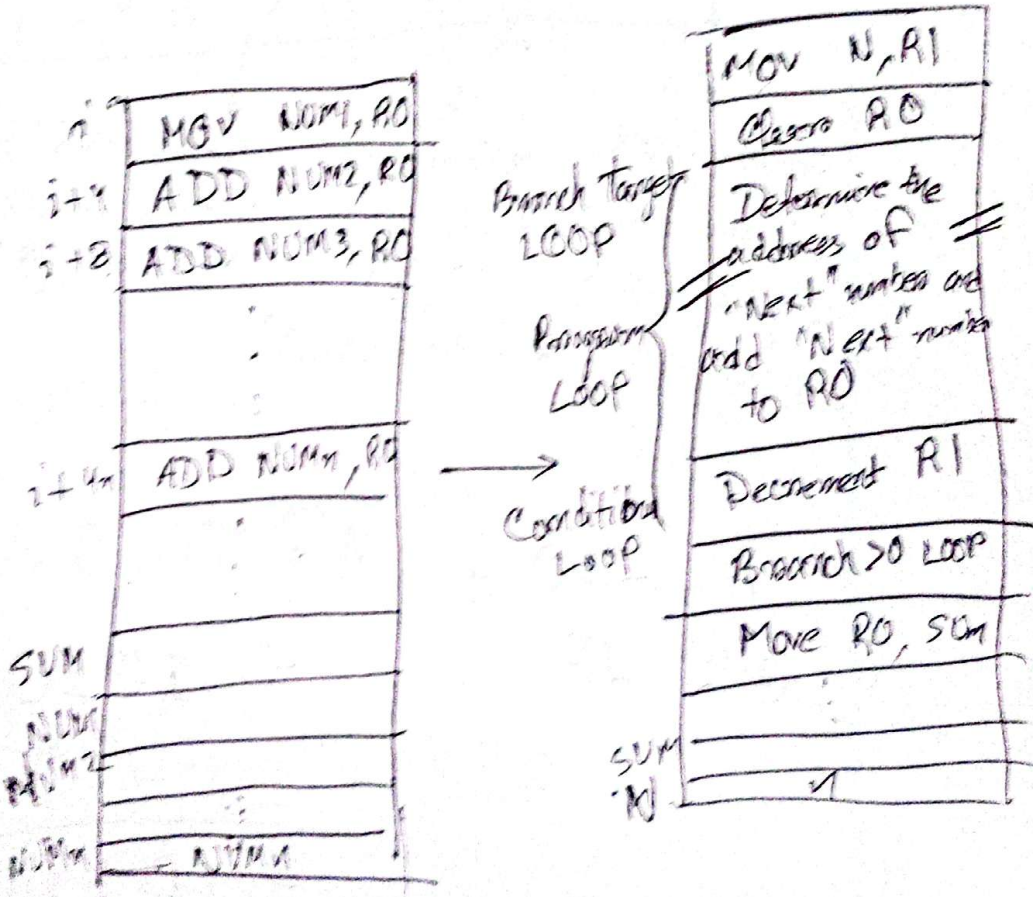
- One memory operand per instruction
- 32-bit word length → 32 bit = 4 byte = i, i+4, i+8...
- Memory byte addressable
- Each instruction fits one word

GOOD LUCK

TOPIC NAME : _____ DAY : _____ TIME : _____ DATE : / /

Branching :

The PC can point towards another address that's NOT the immediate next address. The other address is called branch target

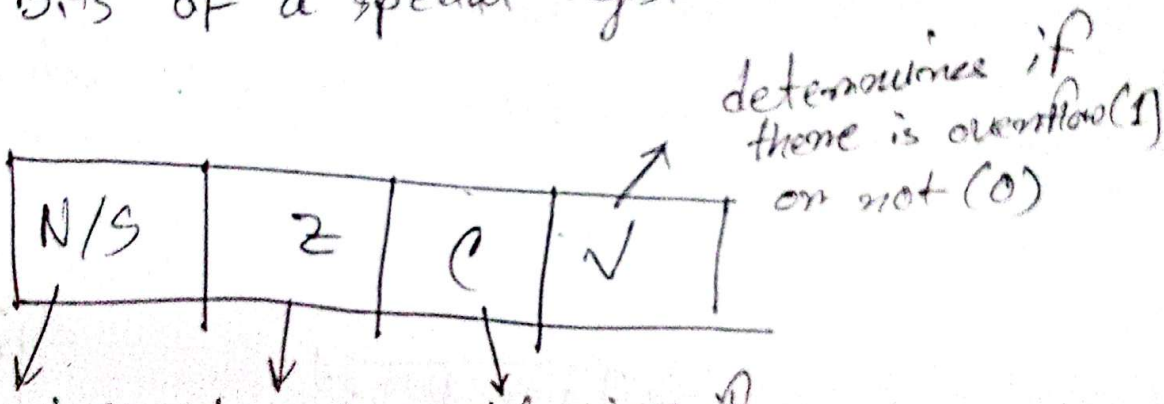


GOOD LUCK

TOPIC NAME : DAY
TIME : DATE

Condition Codes / Status Flags :

Bits of a special register



A : -20
B : 16

A = Two's Complement (0 10100)
= 101011
+ 1
= 101100

-B = Two's complement (0 10000)
= 1 01111
+ 1
1 10000

GOOD LUCK

TOPIC NAME : _____ DAY : _____

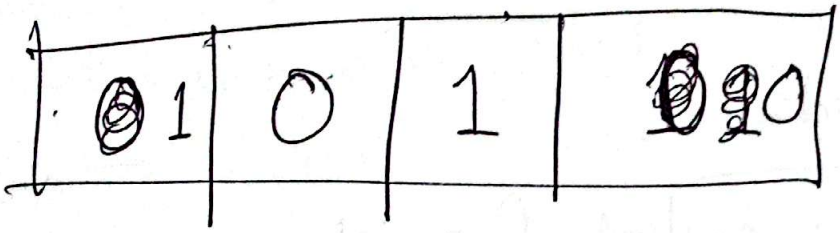
TIME : DATE : / /

$\therefore A-B = 101100$

110000 → This is wrong

1011100

User 8 bit.



~~1000~~

~~10001100~~

GOOD LUCK

TOPIC NAME: W-5 DAY: C-9 10(9 we quiz)

TIME: DATE 29/12/24

Addressing Mode

→ Ways to address operands

Register Mode:

i) Operand is content of register

Absolute Mode: Operand is in memory

* Also called direct mode

* Register and absolute modes can be used to represent variables

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

Immediate Mode: Represents constants

* Operand is given explicitly in the instruction
* # is used before value → #500

* Effective Address: Actual address of operand
  - For Register: EA = Ri;
  - " Absolute: EA = LOCA

Indirect Mode: Address is the content of operand

→ Pointer | EA = [Ri] / [LOCA]

* Expressed with () → (R1)
  - → R1 = B implies
    (R1) → Operand in B

[Check Page 55 of Slide
→ 54: Move #NUM1, R2
→ address of NUM1

GOOD LUCK

TOPIC NAME

DAY :

TIME : DATE : / /

Index Mode: operand is generated by adding constant → offset

* $X(R_i)$
* EA = $[R_i] + X$

GOOD LUCK

TOPIC NAME : W-5 DAY : L-3

TIME : DATE 29/12/24

Organization of the IBM Personal Computer

* Protected Virtual Address → Address is stored elsewhere

* 80486 is strongest in 8086 family

Registers:

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

16 bit is divided into 2 8bit

GOOD LUCK

TOPIC NAME : _____ DAY :

TIME : DATE : / /

AX: ~~Used~~ Accumulator Register

* Arithmetic, logic and data transfer instn.

BX: Base Register

* Stores addresses

CX: Count Register

* Loop Counter
* CL is used to shift and rotate bits

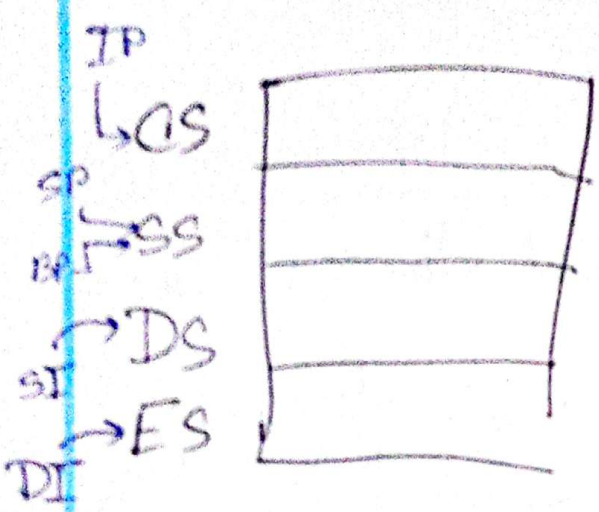
DX: Data Register; (stores remainder)

* Used in MUL and DIV
* " " I/O Operations

GOOD LUCK

TOPIC NAME : DAY
TIME DATE / /

* Segmentation :



* SP → points at the top
* BP → " " any index

* 64 bit segment can be given as 16 bit number
  * Think DLD encoder/MUX

* Physical Address = Segment × 10H + offset
  → Logical Address = segment:offset

* Always do it in Hex

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

* Segment registers should not be used for data transfer.

For 12345 h as physical address

1230h → DS → Segment

0095h → SI → Offset

* IP contain offset of CS

Flag Register :

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

ASSEMBLY LANGUAGE

Assembler;

* Not Case sensitive

Statements;

* Each statement is one line
* PROC MAIN
* Can have upto 4 operands
* ; to start comments.
* ~~name~~ :

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Name Field

* labels, procedures, nameres and variables
* can be 1 to 31 chars long

Operation Field

* opcodes are used in Operation field
* pseudo op codes are used for mentioning stuffs

Program Data:

* for hex, use a digit before numbers starting with A – F (0ABCh ✓ ABCh ✗)
* "A" = 65d = 41h = 10000 1b

GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

Constants:

name EQU address/value

LF EQU 0Ah

Funcs:

* MOV ⟶ cannot be performed between two addresses ⟶ an intermediate register is to be used

* XCHG ⟶ swaps the values of two addresses
  ⟶ ~~same~~ cannot do for mem, mem

* NEG ⟶ does two's complement

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Program Structure :

- MODEL SMALL
- STACK 100h
- DATA

- CODE

MAIN PROC

XMAIN ENDP

END MAIN

"GOOD LUCK"

TOPIC NAME : DAY
TIME : DATE : / /

甲I/O

```
MOV AH, 1
INT 21H
```
} input ASCII CODE
↳ stored in AL

```
MOV AH, 2
INT 21H
```
} output ASCII CODE
↳ which is stored in DL

```
MOV AH, 9
INT 21H
```
} output string
↳ which is stored in DX
↳ strings must end with 拿$

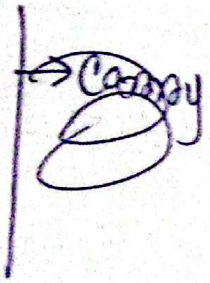
GOOD LUCK

TOPIC NAME: 10-5 DAY: L-4

TIME: DATE: 01/01/24

Flags

- Status (Status of any operation) — 2, 4, 7, 8, 11
- Control (Controls operation) — 8, 9, 10.



1] Status Flags:

Carry flag: 0 when carry 0
1 " " 1

Parity: 0 when low byte has odd no. of 1
1 " " " even no. " 1

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Auxillary Carry Flag: 0 if BCD addition carry/borrow = 0

1 " " " " = 1

Zero Flag: 0 if result is (not ~~zero~~ 0)

1 " " " (0)

Sign Flag: 0 if MSB ~~(SF)~~ is 0 (or +ve)

1 " " " 1 (or -ve)

Overflow Flag: 0 if there is overflow

1 " " " "

→ If acquired answer is higher than the range

- → No
- → ~~Sign~~ Unsigned → if CF = 1
- → Signed → if SF changes
- → Both signed and Unsigned

*OF = Carry In MSB ⊕ Carry Out MSB

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Instructions effect on flags:

MOV / XCHG —— None

ADD / SUB —— All

INC / DEC —— All except CF

NEG —— All

CF = 1 unless result = 0

OF = 1 if word operand = 8000h

or byte " = 80h

2's complement of -128 is 128

where range is between -127 to 128

*If CF = 1 with all trailing 0s, ZF = 1

GOOD LUCK

TOPIC NAME : W-5 DAY : C-11

TIME : DATE 01/01/25

Addressing Mode

Relative Address: X(PC) → Used for loops

X → offset

PC → Address of instr.

-16(PC) → go to 16 bytes before PC's pointing

Auto increment: (R1)+ → [R1] = [R1] + 1

* Adds later

Auto decrement: -(R1) →

used in stack

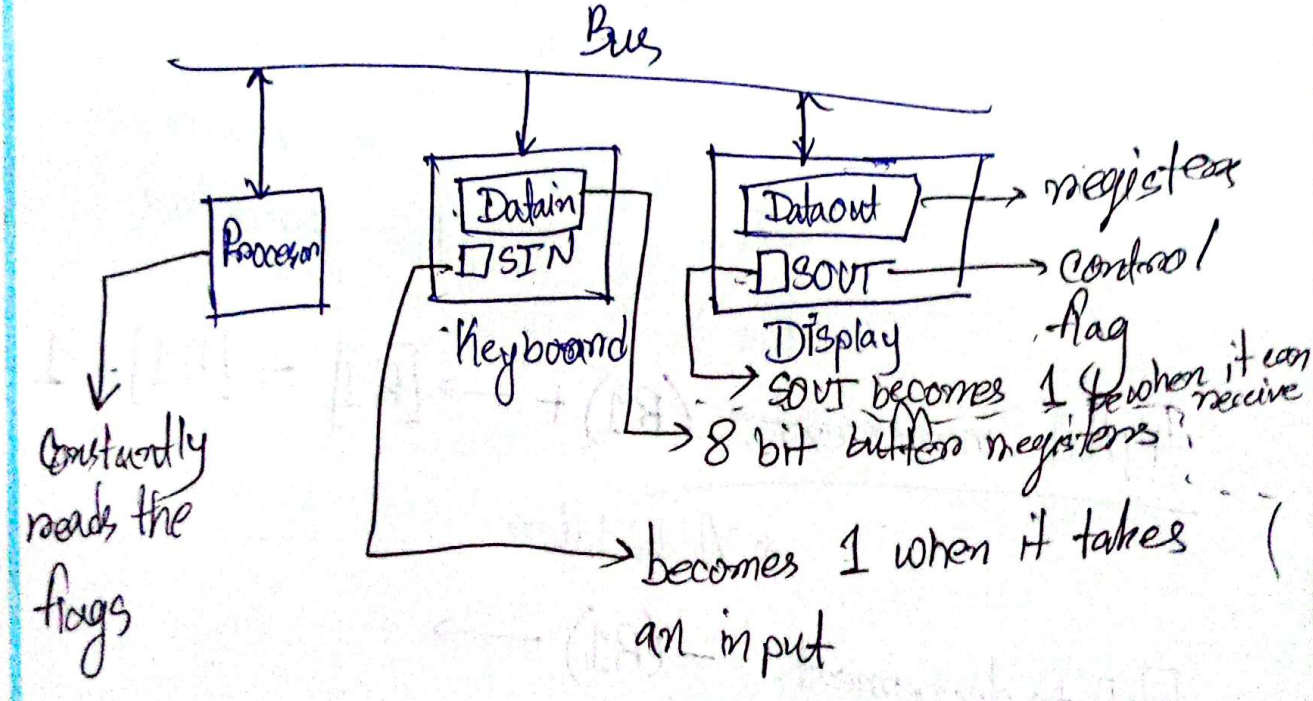
* Subs first

GOOD LUCK

TOPIC NAME : DAY
TIME : DATE : / /

Ch - 3

I/O stuffs



GOOD LUCK

TOPIC NAME : W-5 DAY : C-12

TIME : DATE 02/01/25

I/O Devices or Something

Basic Input output Operation

* DataIn / Data Out ⟶ 8 bit Buffer register

// In single bus, the speed of processor is faster, there is a chance of data getting replaced

* SIN/SOUT ⟶ Control flags

Keyboard → value → Processor → value → Mem

Mem → ~~reads~~ value → Processor → value → Display Device

** When SIN = 1, Data can be read from DATAIN

** " SOUT = 1, " " sent to DATAOUT

GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

READWAIT Branch to READWAIT if SIN=0

Input from DATAIN to R1

WRITEWAIT Branch to WRITEWAIT if SOUT=0

Output from R1 to DATAOUT

*this is not efficient and as it keeps the processor busy.

*SIN and SOUT are kept along with other control/status flags

GOOD LUCK

TOPIC NAME : DAY
TIME : DATE / /

To read a character

```
READWAIT  Testbit    #3, INSTATUS
          Branch=0   READWAIT
          MoveByte   DATAIN, R1

WRITEWAIT Testbit    #3, OUTSTATUS
          Branch=0   ~~READ~~ WRITEWAIT
          MoveByte   R1, DATAOUT
```

To read a line (NOT storing)

```
          Move       #LOC, R0

READ      Testbit    #3, INSTATUS
          Branch=0   READ
          MoveByte   DATAIN, (R0)

ECHO      Testbit    #3, OUTSTATUS
          Branch=0   ECHO
          MoveByte   (R0), DATAOUT

          Compare    #CR, (R0)+ ; Compare A, B = B-A
                                  #CR = ~~Last~~ Value of Carriage Return
          Branch≠0   ~~Read~~ READ
```

↳ Start of line

GOOD LUCK

TOPIC NAME: W-B 6 DAY: C.13 TIME: DATE: 01/25

Subroutine: Thing that needs to be executed multiple times

* Instruction that performs the branching operation is named as Calling Instruction

* Link Register is used to store value of PC when branching.

* Subroutine Linkage → whole method

* Subroutine Nesting:

→ Subroutine inside subroutine

→ Makes link register hard

→ It is better to use SP (Stack Pointer) here

→ Stack Pointer:

→ Decreasing Memory Location ↓

→ Push() → SUB #4, SP ; MOV NI, (SP)

1024 [TOS]
1020 [ ]
↳ we need to move TOS to 1020 and move data there

↳ MOV NI, -(SP)

→ Pop() → MOV (SP), LOC ; ADD #4, SP } MOV (SP)+, LOC

GOOD LUCK

TOPIC NAME : W-6 11 DAY : C-13

TIME : DATE : 05/01/25

Parameter Passing :

Using Register

Calling Program

```
Move   N, R1 ; R1 is a counter
Move   #NUM1, R2 ; R2 pointing to list (first number)
Call   LISTADD ; Call subroutine
Move   R0, SUM ; Save result
```

Subroutine

```
LISTADD  Clear R0 ; Initialize sum as 0
LOOP     Add (R2)+, R0 ; Add entry from list
         Decrement R1
         Branch >0 LOOP
         Return ; Return
```

GOOD LUCK

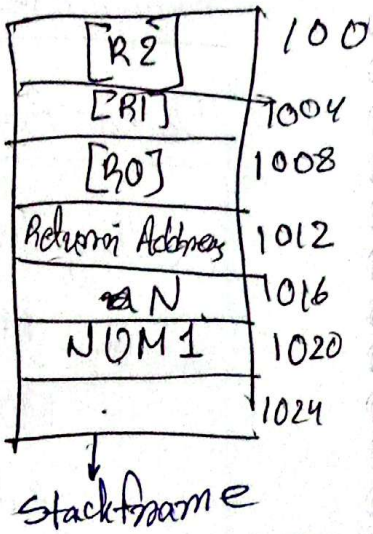
TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Using Stack

SP at 10204

```
Move    #NUM1, -(SP); 1020
Move    N, -(SP); 1016
Call    LISTADD; 1012 ← Ret Add
                 → SP is at 1016
Move    4(SP), SUM; Index 1020
Add     #8, SP; Back to 1024
```



Stackframe

```
LISTADD  Move Multiple  R0→R2, -(SP); 1008→1000
         Move  16(SP), R1; 1016 (SP is at 1000)
                        ↳ this just offsets
         Move  20(SP), R2; 1020 ( " )
         clear R0
LOOP     Add   (R2)+, R0
         Decrement R1
         Branch>0  LOOP
         Move  R0, 20(SP); Replace 1020
         Move Multiple (SP)+, R0→R2; 1000→1008
                        ↳ actually R2→R0
         Return; 1012 is removed → Move SP
```

GOOD LUCK

TOPIC NAME: W-6 DAY: L-5

TIME: DATE: 8/1/25

Label:

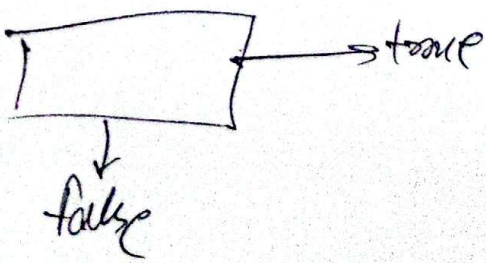
* Refers to another line
* Label_Name:

Conditional Jump:

* JMP → unconditional jump → no limitations
  - can only jump between 126 bytes
* JG/JNLE → dest > src
  JGE/JNL → "≥"
  JL/JNGE → "<"
  JNE/JGL → "≤"

Use Slide Page for → 6-8

* Jump → LABEL if true
  LABEL if false



GOOD LUCK

TOPIC NAME : DAY :

TIME : DATE : / /

CMP : Compare

* Dest - Src

- 0 → Equal
- >0 → Greater
- <0 → Smaller

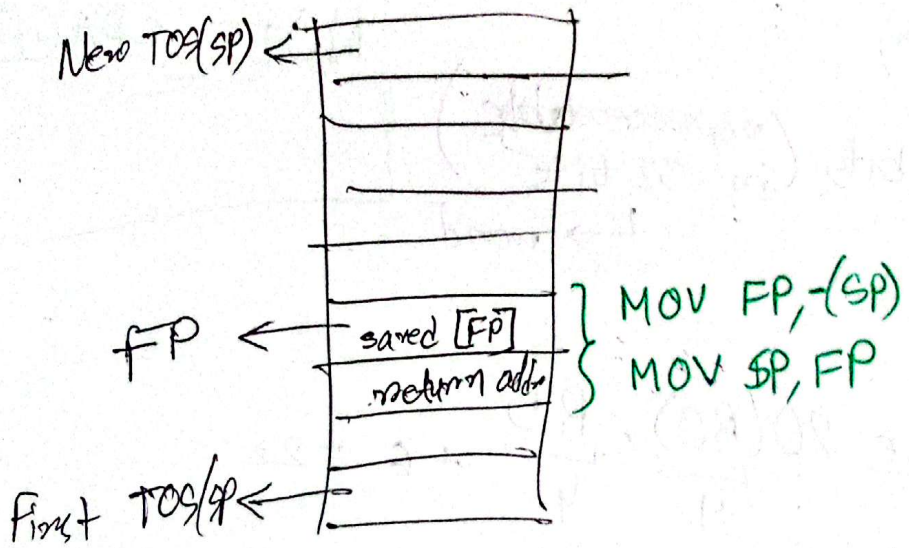
GOOD LUCK

TOPIC NAME : W-6 DAY : C-14

TIME : DATE : 8 / 1 / 24

Frame Pointer, Stack Pointer

→ points to the address right above the return address → Good for indexing



#Self Study : Carl Hamacher book page (78-80)

GOOD LUCK™

Encoding Instruction: Converting instr. into binary

* 3 bits to represent addressing mode

ADD R1, R2 → TO ADDRESS NOT SPACE

ADD: 6 bits (3 for each), 8 bits
R1: 4 bit
R2: 4 bit

= 22 bits (representable in 32 bits → word)

Move 20(R0), R5 + 6 = 22

Move: 8
20(R0): 4
R5: 4

20 can only be expressed in 2^{10} range

10 bits remaining

* 1 WORD is used in memory address
* RISC can use only 1 word

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

To solve the issue of RISC using memory locations,

~~we set~~ → We ~~set~~ memory locations near instructions
↳ let it stay in 1016

→ We ~~call~~ set the value using relative address
MOV 16(PC), R2

→ We move that to a register using indirect
↳ MOV R1, (R2)
address

→ win

GOOD LUCK™

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

4. Arithmetic

* Multiplier → Right shift
  ~~Multip~~
  Product → Left shift

} Until Multiplier is 0

OR

* Multiplicand → Left shift

```
  1000
 x1001
------
  1000
 0000
0000
1000
------
1000000
```

* For n bit multiplicand
  m " multiplier

Final shifted multiplicand will be (n+m) bits

GOOD LUCK

TOPIC NAME: 10-6

DATE: 9/1/25

3 basic steps for each bit

[do n times] → no. of bits in multiplier

1. Check LSB of multiplier and decide whether to add the multiplicand to product

   If LSB = 1 → add

   If LSB = 0 → do nothing

2. Left shift the multiplicand by 1 bit

3. Right " multiplier by 1 bit

↓ (i) (ii) (iii) (iv)

\# 1000
   1001

10000
 0100

100000
  0010

1000000
   0001

10000000
    0000

Product: 0

0 + 1000

0 1000

Do Nothing

1000

Do nothing

1000

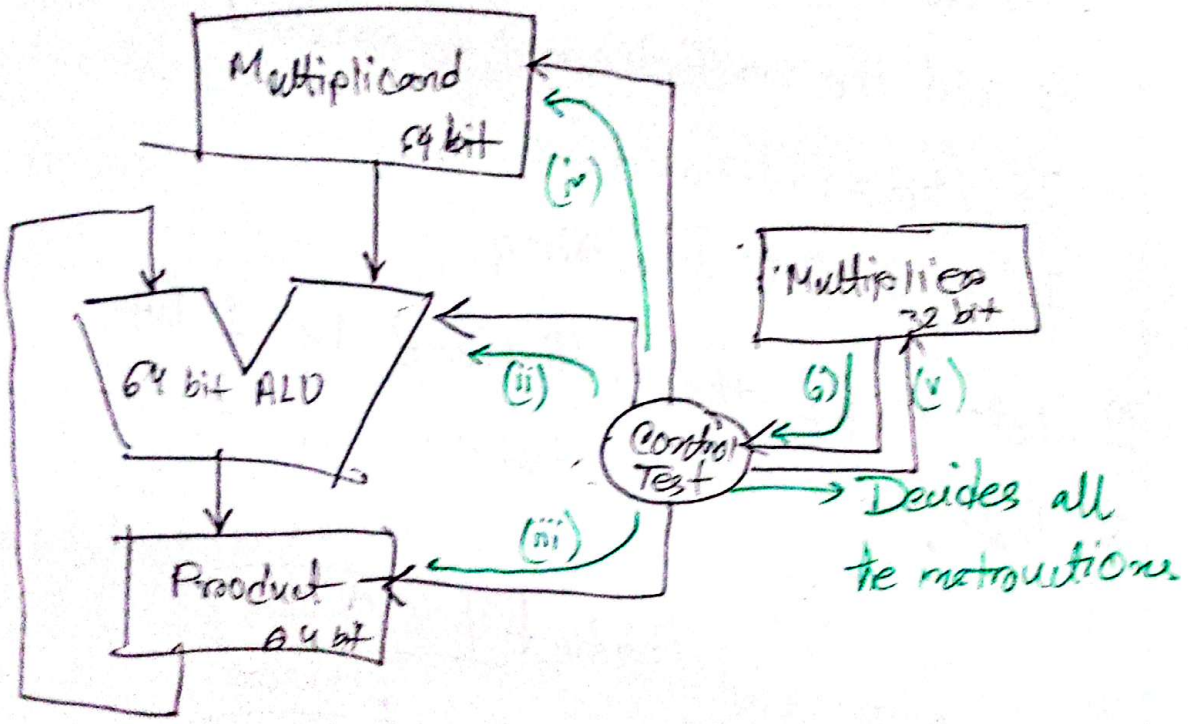
1000 + 1000000

1001000

GOOD LUCK

TOPIC NAME

Hardware:



(i) Control test decides checks LSB

(ii) Lets addition happen

(iii) Write the acquired result in Product

(iv) Left shifts Multiplicand

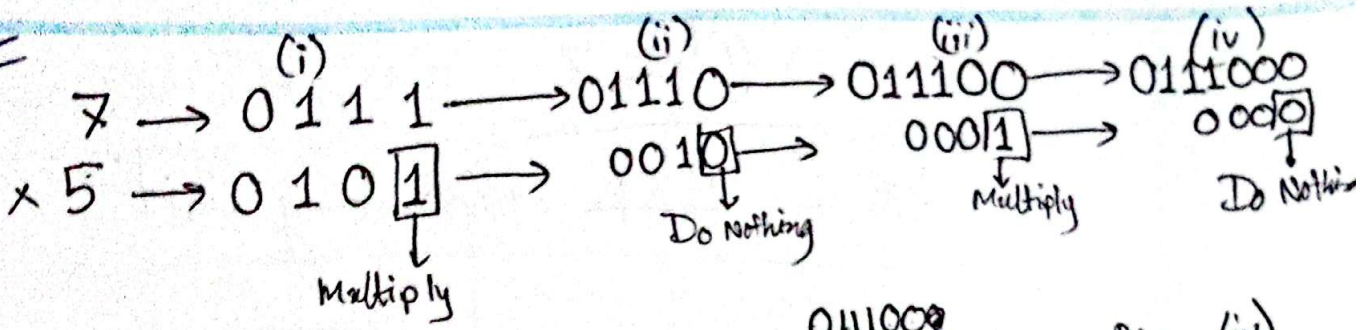
(v) Right shifts Multiplier

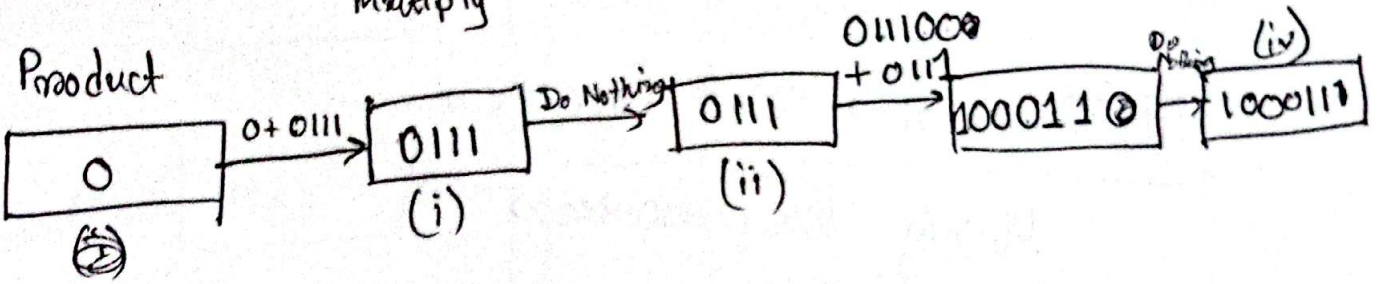
GOOD LUCK

TOPIC NAME : DAY :

TIME : DATE : / /

#





Product = 00100011

#Self Study → Patterson & Hennessy

GOOD LUCK

TOPIC NAME 10-7 C-16

DAY TIME DATE 12/01/25

Ch-7

Basic ~~Unit~~ Processing Unit

Instruction Set Processor:

Mainly the processor

- Control Unit Generates the signals of instructions

$IR \leftarrow [[PC]]$

PC → address

[PC] → address value

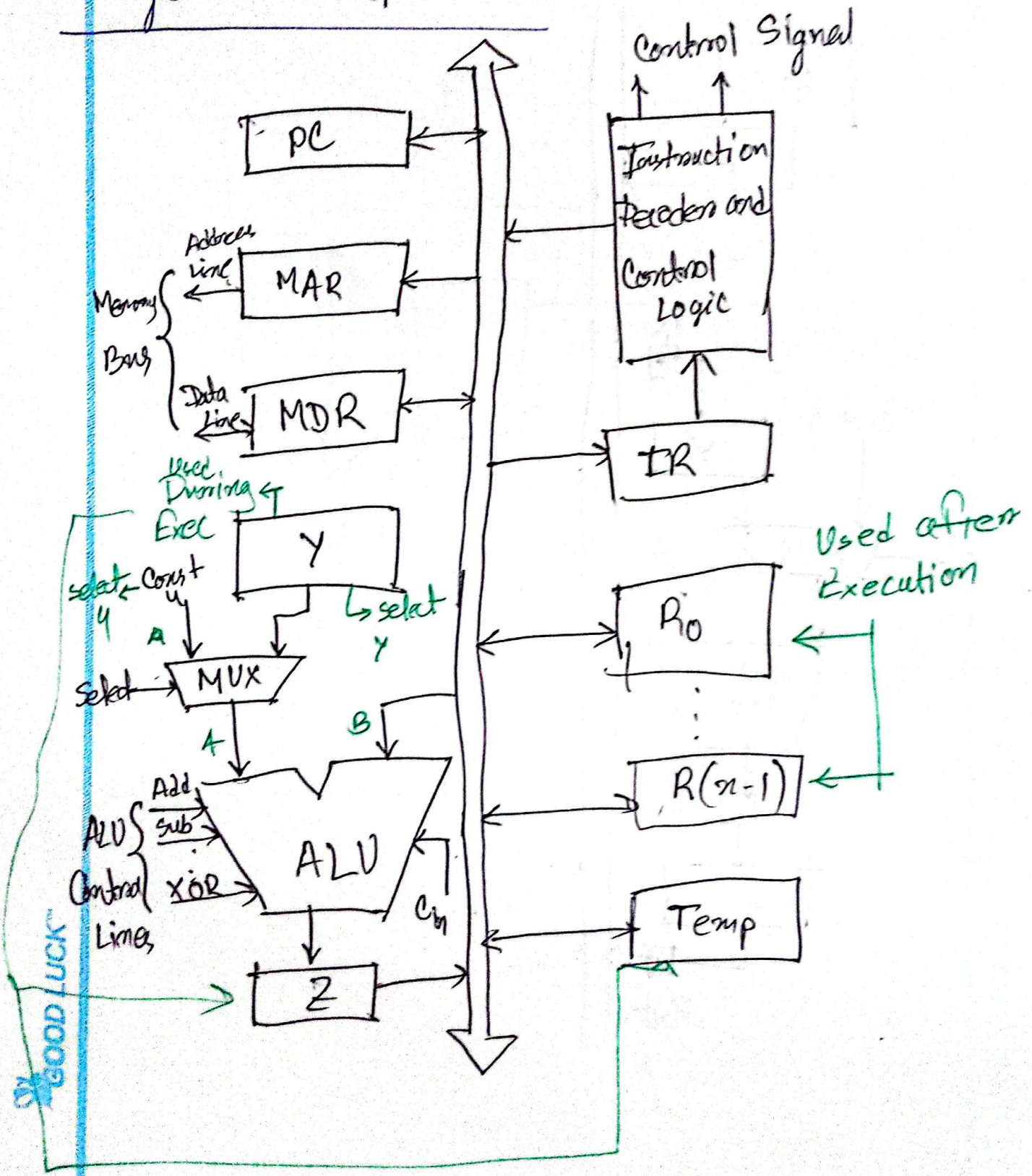
[[PC]] → value inside the address value of PC

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

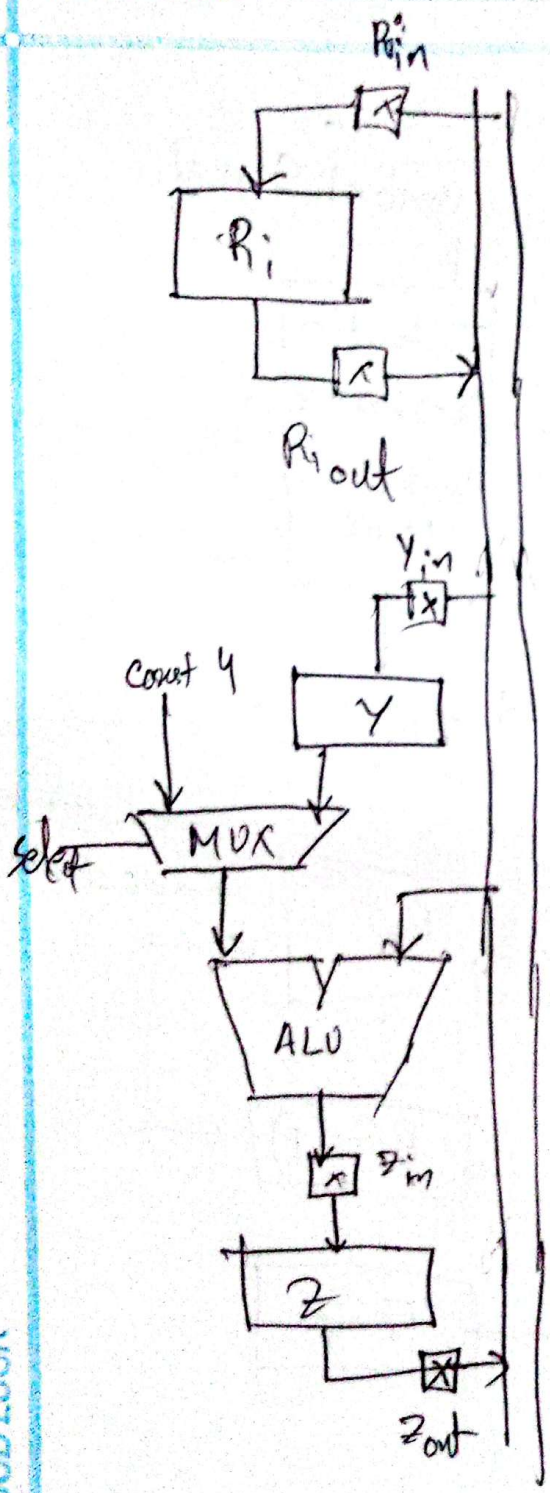
Single Bus Datapath :



TOPIC NAME :

DAY :

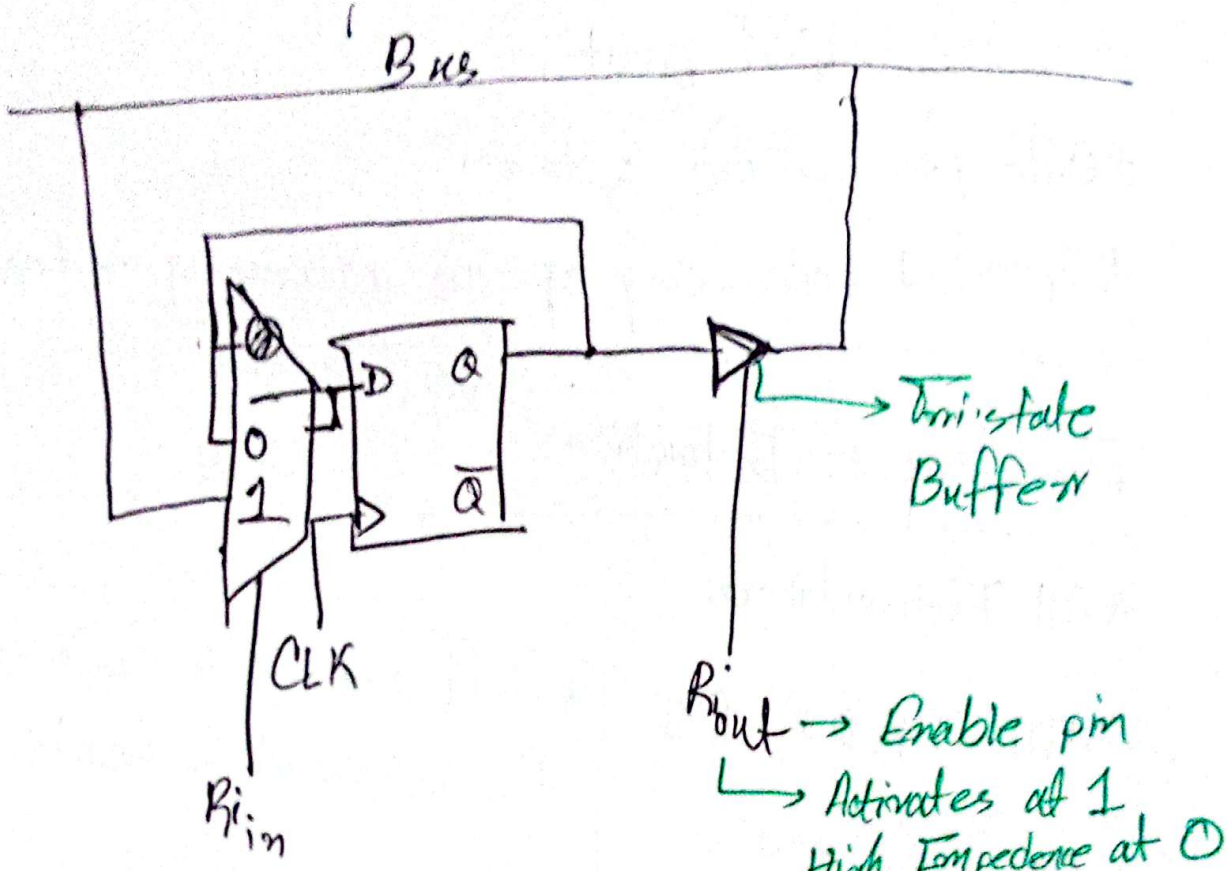
TIME : DATE : / / /



"GOOD LUCK"

TOPIC NAME : DAY :
TIME : DATE : / /

Single bit representation of Register:



*ALU is a combinational ~~Ci~~ Circuit

GOOD LUCK

TOPIC NAME: W-7 DAY: C-17 DATE: 13/1/25

CA Quiz-2

*No theoretical part
*Code for specific systems
*Specified code using specific addressing modes

Executing an Instruction:

*All Instructions

#ADD R1, R2, R3

*Only one clock cycle is needed for internal instructions

1. R1out, Y in
   ↕ → Different clock cycles to prevent errors
2. R2out, SelectY, Add, Zin
3. Zout, R3in

GOOD LUCK

TOPIC NAME

DAY

TIME : DATE / /

Fetching a word from Memory

1. $R1_{out}$, MAR_{in}
2. Read

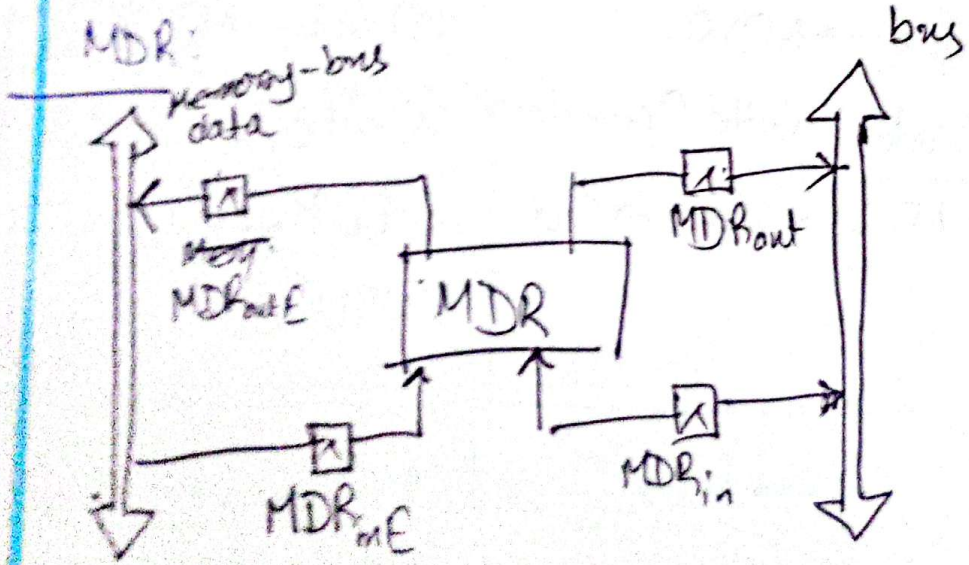
→ Can be combined → One clock cycle

3. WMFC
4. MDR_{inE}

→ Can be combined → One clock cycle

5. MDR_{out}, $R2_{in}$

1. $R1_{out}$, MAR_{in}, Read
2. WMFC, MDR_{inE}
3. MDR



GOOD LUCK

TOPIC NAME : DAY :
TIME : DATE : / /

MOV R1, (R2)

Step 1: ~~R1 o~~ Place R2 onto the ~~internal~~ bus
- Load contents onto MDR
- Place

Step 2: ~~WMFC~~

Steps:

1. R2 → bus → MAR
2. R1 → bus → MDR
3. ~~Wr~~ Activate Write Operation
4. ~~WMFC~~ MDR → Mem → Bus
5. WMFC

Control sequence

R2out, MARin
R1out, MDRin
Write
MDRoutE
WMFC

→ MDRin and MDRoutE & Write and WMFC CANNOT be in the same clock cycle

GOOD LUCK

TOPIC NAME: W-7 DAY C-19

TIME: DATE 16 / 1 / 25

Mov (R1), R2

Steps:

1. R1 → Bus → MAR
2. Activate Read Operation
3. WMFC
4. Mem bus → MDR
5. MDR → Bus → R2

Control Sequence:

R1out, MARin
Read
WMFC
MDRinE
MDRout, R2in

(On the end of MFC, we can activate MDR)

Restrictions for combining:

1. Multiple out switches cannot be ~~open~~ 1
2. Read/Write and MFC cannot be in the same cycle
3. Same clock cycle cannot have any registers in and out switch activated.

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

ADD (R1), R2 → Complete Execution

PC.

Steps :

1. PC → Bus → MAR : PC_{out}, MAR_{in}
2. Activate Read Operation : Read
3. PC ← [PC] + 4 : Select 4, Add, Z_{in}
4. WMFC : WMFC → Can be multiple on single step (Z_{out}, PC_{in})
5. Mem Bus → MDR : MDR_{inE}, Y_{in}
6. MDR → Bus → IR : MDR_{out}, IR_{in}
7. R1 → BUS → MAR : $R1_{out}$, MAR_{in}
8. Read signal : Read
9. WMFC : WMFC
10. Mem bus → MDR : MDR_{inE}

1 = 1, 2, 3, 4

2 = 4

3 = 5

4 = 7, 8, 9, 10

5 = 5+6, 5+7, 5+8, 5+9, 5+10

6 = 5+6

7 = 5+7, 5+8, 5+9 (Select Y, Add)

8 = 5+10. Z → bus → R2 : Z_{out}, $R2_{in}$

"GOOD LUCK"

TOPIC NAME : U-8 DAY : C-19

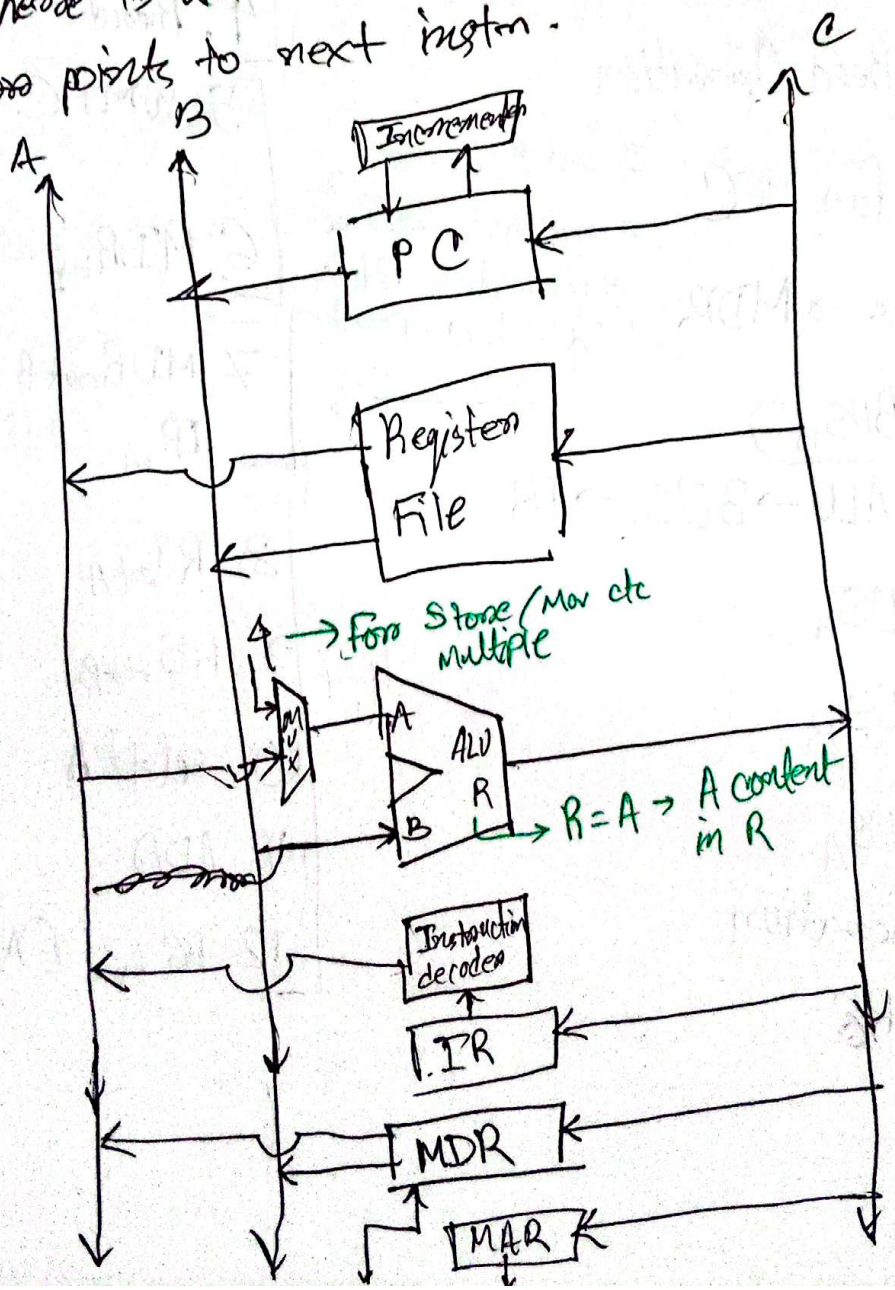
TIME : DATE : 26/01/25

* JMP → ~~for~~ for Branch : Self Study

Multiple Bus Organization

3-bus Organization :

* There is an incrementer which automatically stores points to next instn.



GOOD LUCK

TOPIC NAME : DAY :
TIME : DATE : / /

ADD R4, R5, R6

Steps:

1. PC → BUS_B
2. BUS_B → ALU → BUS_C → MAR
3. Incrementer
4. Activate Read Operation
5. Wait for MFC
6. Mem Bus → MDR
7. MDR → BUS_B
8. BUS_B → ALU → BUS_C → IR
9. R4 → BUS_A
10. R5 → BUS_B
11. Select BUS_A
12. ADD operation
13. Res → R6

7 and 8 cannot be merged because 7 is instruction fetch and 8 is for operation

Control Signal

1
1. PCout
2. R=B, MARin
3. Inc PC
4. Read

2
5. WMFC

3
6. MDRinB

4
7. MDRoutB, R=B, IRin

4
8. R4outA
9. R5outB
10. selectA
11. ADD
12. R6in, END

GOOD LUCK

TOPIC NAME : DAY
TIME : DATE : / /

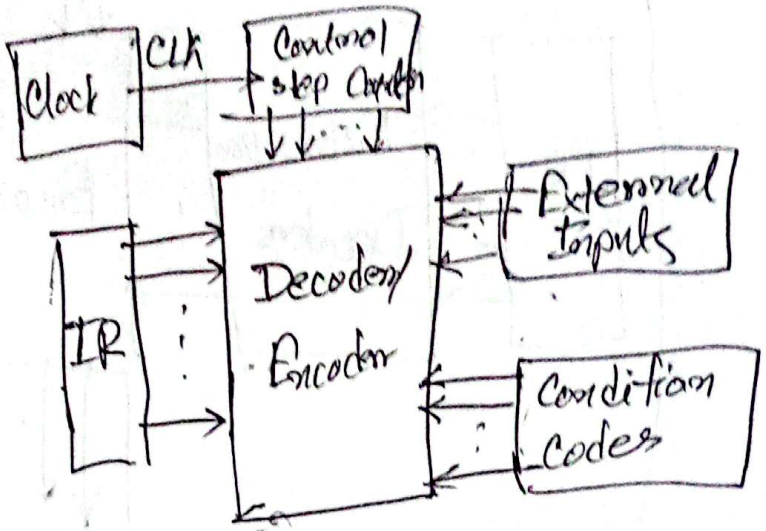
Hardwired Control Unit

CU

- Hardwired
- Microprogrammed

HCU:

- Step Counter
- Instruction Register
- Condition Code Flags
- MFC

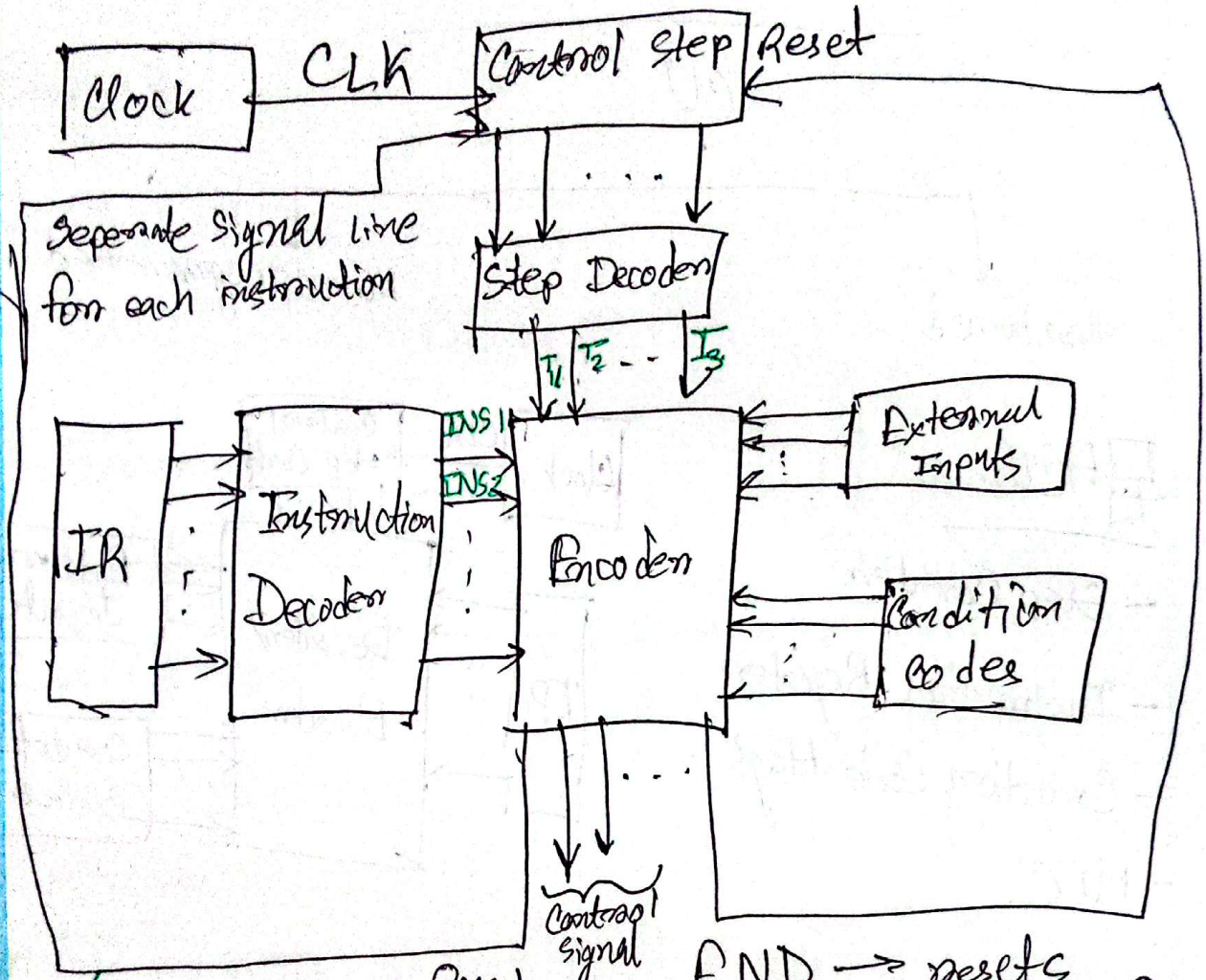


PC → bus → MAR
MA Read WMFC
2.
PC → IBUS → MAR : PCout MARin
: Read
: Select 4, Add2in
: WMFC
PC ← [PC] + 4
WMFC
MDR → Bus → MDR : MDRinE

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /



↳ Turn make sure clock cycle doesn't increase step counter

GOOD LUCK

L-7

TOPIC NAME : W-8 DAY L-~~6~~ (~~Tuesday~~)

TIME DATE 29/01/2025

Logic Instructions:

AND: Works like multiplication

- CF = OF = 0
- Effects SF, ZF and PF

	128	64	32	16	8	4	2	1
'a' = 97 =	0	1	1	0	0	0	0	1
'A' = 65 =	0	1	0	0	0	0	0	1
to convert =	1	1	0	1	1	1	1	1

∴ AND 'a', 0DFH = 'A'

*AND is used for clearing Bits

~~∴ To make~~

*Use AND to go lowers

> * Use the bits to convert it to hex
> AND/OR that hex

OR: Works like

- Same effect as AND

*OR is used to check sign or register value

GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE / /

XOR: $0 \oplus x = x$

$1 \oplus x = x'$

* Used for clearing registers

$x \oplus x = 0$

NOT: 1's complement any numbers

XOR complements bit

NOT " whole thing

TEST: dest AND src

↳ CMP but AND

Check what bit

GOOD LUCK

TOPIC NAME : _____ DAY : _____ TIME : DATE : / /

Shift Instructions:

SHL = Left shift
SHR = Right shift
} For !(1) shift, use CL
→ this moves directly

SHL / SAL

SHL dest, src → dest = dest × 2^{src}

For Multiplication of non 2^n,
Do it separately

* Shifting multiple times might not give the right CF and OF

GOOD LUCK™

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

SHR + SAR:

* CF = LSB
* MSB = 0
* CF =

SHR dest, src ⇒ dest / 2^{src}

SAR MSB = MSB

1000

SAR = 1100

GOOD LUCK

TOPIC NAME : DAY TIME : DATE / /

Rotate Instructions :

Rotate AND sends value to CF

ROL: CF = MSB
LSB = MSB

RCL: LSB = CF
CF = MSB

ROR: CF = LSB
MSB = LSB

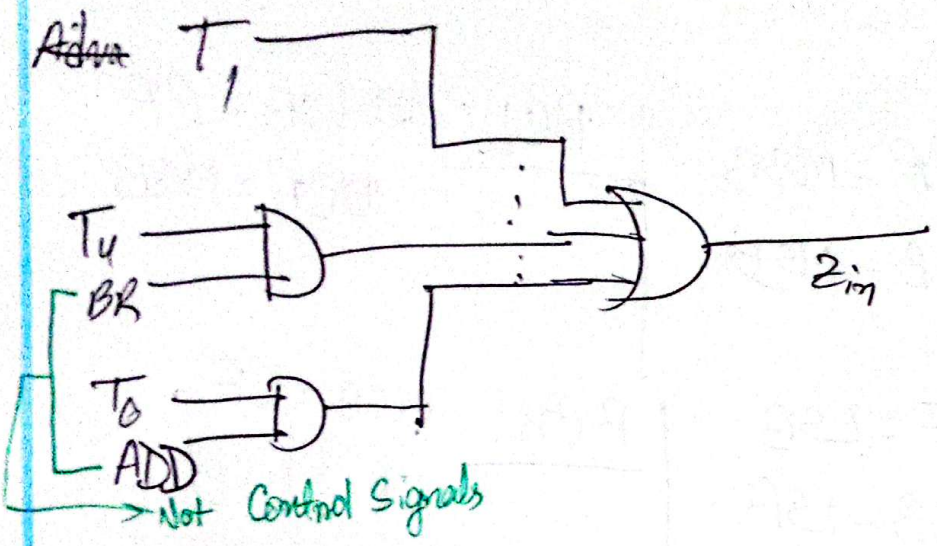
RCR: MSB = CF
CF = LSB

GOOD LUCK

TOPIC NAME: 10-8 DAY: C-20

TIME: DATE 30 / 1 / 25

Hard Wired Control Unit



Advantagee : Fast

Disadvantagee : Complex

GOOD LUCK

TOPIC NAME :

DAY
TIME
DATE / /

Microprogrammed CU:

* Control Signals are generated using Program
* A control word is used where each individual bit represent various control signals

Micro Instruction	R_{in}	R_{out}	. .	:	:	.
1	0 if not used in step	1 if used in step				
2						
.						
.						
n						
.						
.						

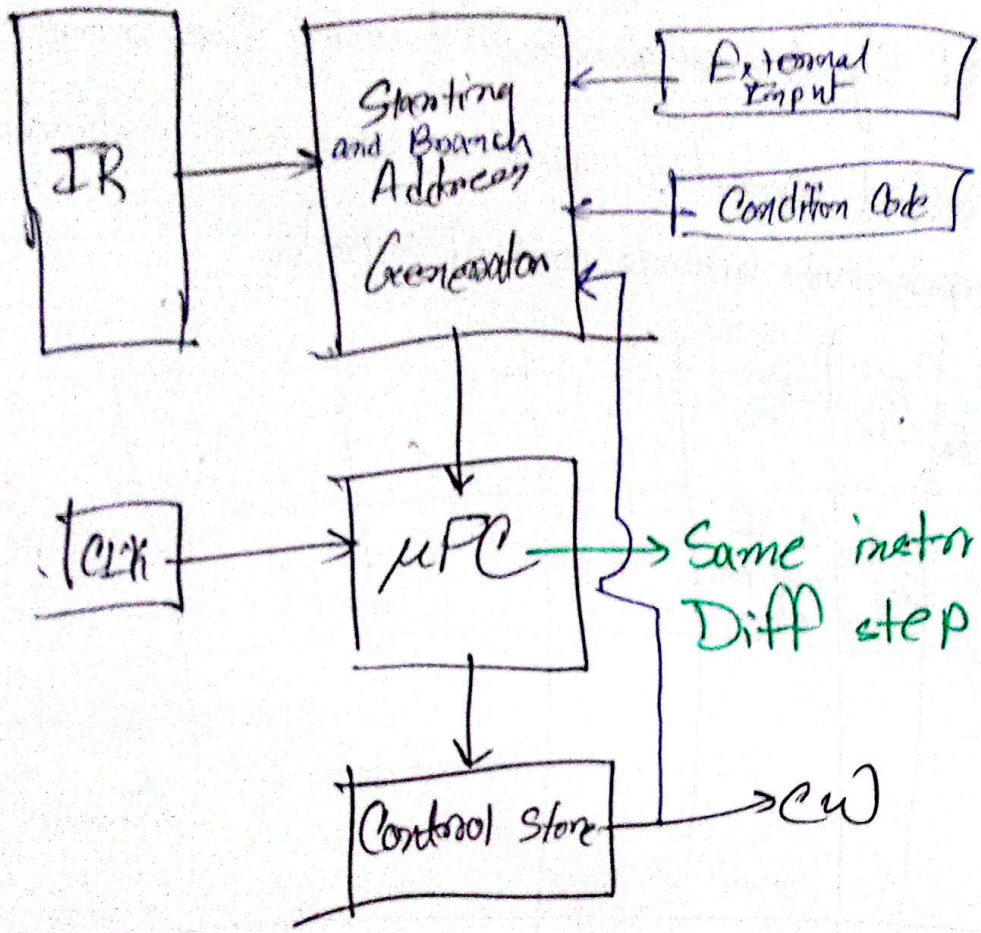
Each sequence of control word is microroutine

Micro routines are stored in ~~micro s'~~ control Store

GOOD LUCK

TOPIC NAME : DAY
TIME : DATE : / /

Updated



+ Control Signal of instruction → Execute it
+ Microinstructions

GOOD LUCK

TOPIC NAME: W-9 DAY: L-8
TIME: DATE: 5/2/25

Online - 7

AND

0	0	0
0	1	0
1	0	0
1	1	1

XOR

0	0	0
0	1	1
1	0	1
1	1	0

F4

1 → 31 H

1 H

1 1 1 1
0 1 0 1
AND 0 1 0 1

XOR X'

2H

0101 0101
5

9 → 39 H

9H

128 A n.

00

1000 0010 1000
0101 0101

0111 0001
0010 0000

18 H

1001
9H

128 D → 1000 0000
40H

0110 0110
0010

128

3C

1000 0000
XOR 0010 0010

1000 0010 1000

TOPIC NAME : DAY :
TIME : DATE : / /

Online 7 Soln:

→ Mov 12BD in BH (2H → 40H)
~~XOR 0AAH~~. 0AAH ⊕ BH (2AH)
→ CX, 8
COUNT
→ SHL BH, 1;
JNC ZERO
CONTINUE

ZERO:
INC ZER
CONTINUE
LOOP COUNT.

(4X)

"GOOD LUCK"

TOPIC NAME : DAY
TIME : DATE : / /

STACK:

- Each interaction reduces SP
- PUSHF → PUSHes all the flag value
- Const PUSH # [const]
  PUSH AL/BL/CL/DL...
- Can PUSH AX

PROCEDURE:

[Name] PROC [type] → we use small model so by default its NEAR

.
.
.

ENDP [Name]

CALL PROC → Return address is stored in stack

GOOD LUCK

TOPIC NAME 10-9 DAY C 21+22

TIME DATE 6/2/25

Ch-8

Pipelining

Speed can be increased in two ways

- Faster circuit technology
- Arrange hardware to simultaneously run multiple instruction

Throughput: Instr. exec/~~sec~~ unit time

Latency: Time taken to complete instn.

* Throughput improves in pipelining latency does not change

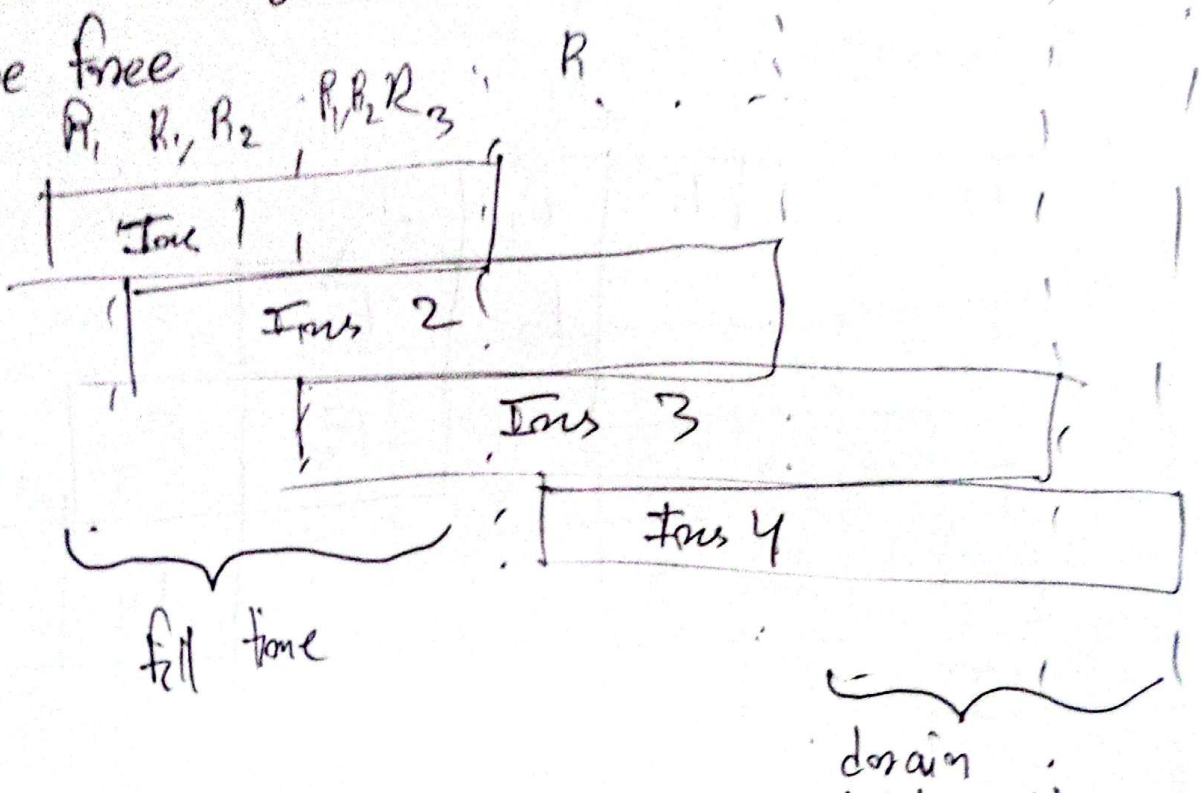
GOOD LUCK

TOPIC NAME : DAY :
TIME : DATE : / /

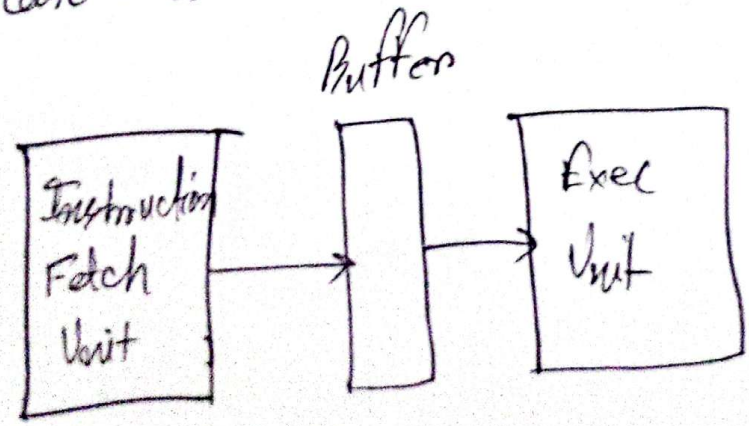
Pipelining :

Overlapping instructions if previous ~~stages~~ resources

one free



Ideal case : all resources are used at a time

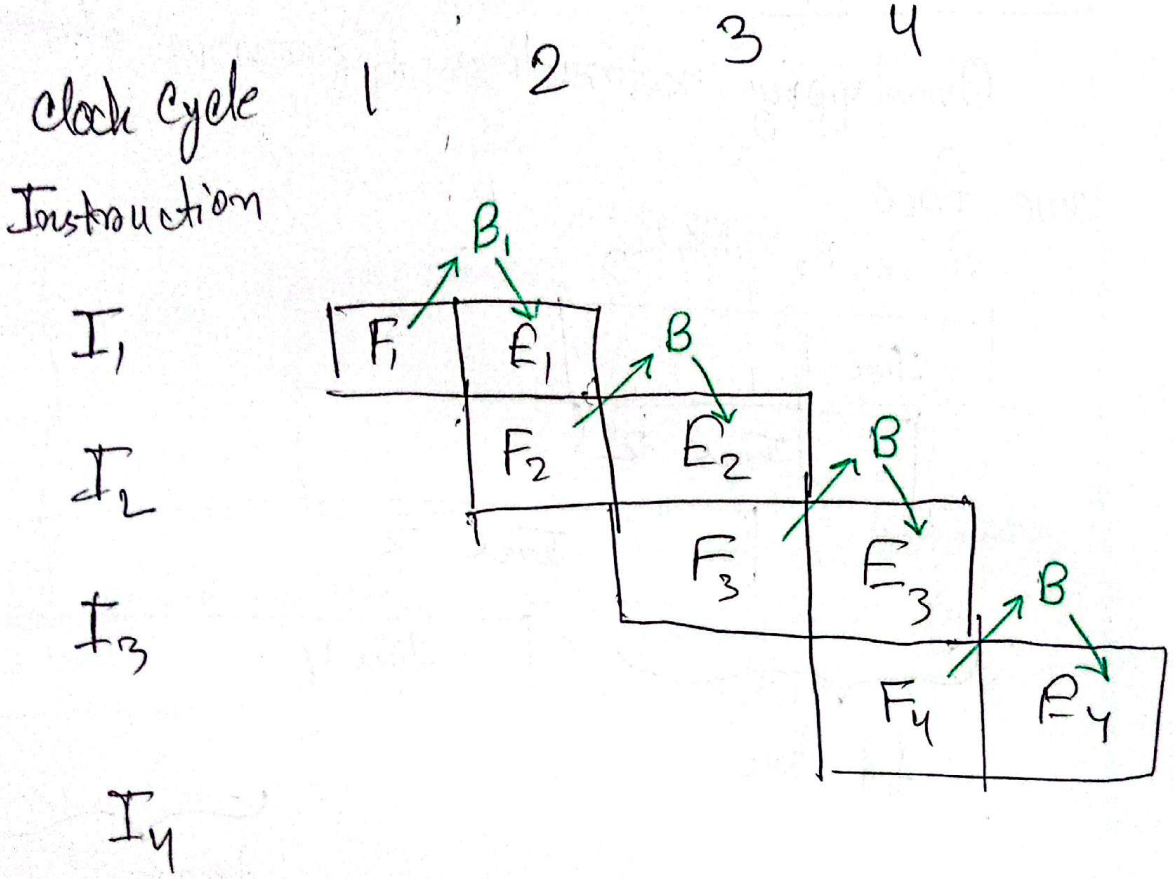


GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

For fetch and execution

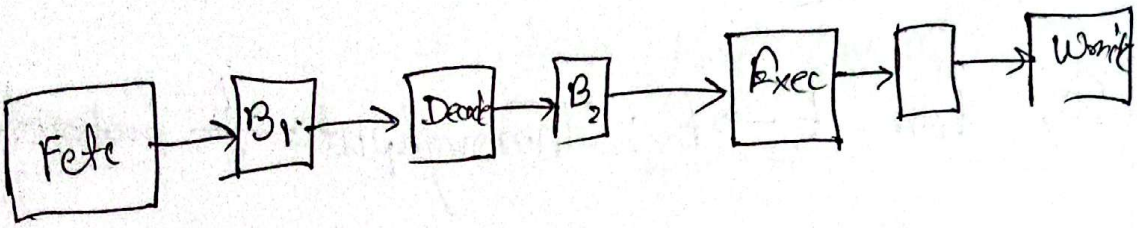


"GOOD LUCK"

TOPIC NAME : _____ DAY : _____

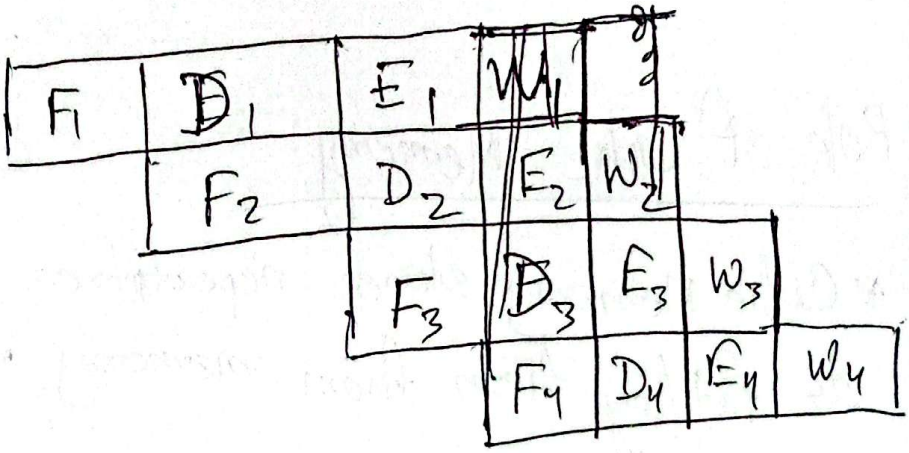
TIME : DATE : / /

For fetch → decode → Execute → Write



C.C 1 2 3 4 5 6 7

Ins



GOOD LUCK™

TOPIC NAME _____ DAY: _____

TIME: DATE: / /

Throughput is the total instruction / time

So, for F→E: Throughput = $\frac{1}{2}$ instruction/cycle

Latency = 2 cycles → 2 unit time for one F-E

Role of Cache Memory:

* Cache Memory stores repeatitive instructions as fetching them from memory requires much time

* Pipelining can increase throughput if ~~they all~~ all instructions required same time and there is no hazard.

↳ Any interruption to pipelining is ~~haza~~ "stall"

A condition which causes the pipeline to "stall" is called hazard.

TOPIC NAME

Data Hazard : Availability of sequentiality timing

Hazard:

$A \leftarrow 3 + R_1$

$B \leftarrow 4 \times A$

Occurs when :

* Dest 1 = Src 2
* Dependency

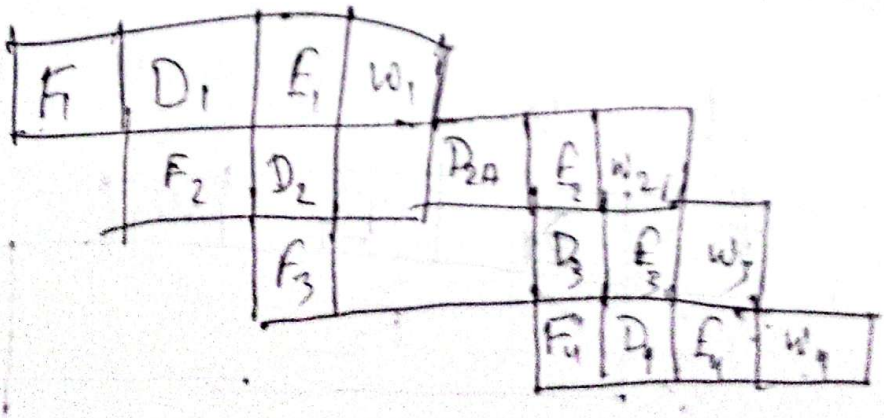
Not Hazard:

$A \leftarrow 5 \times C$

$B \leftarrow 20 + C$

Mul R2, R3, R4

Add R5, R4, R6



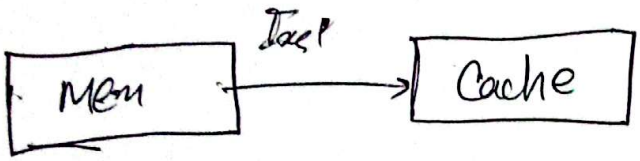
GOOD LUCK

TOPIC NAME: W-10 DAY: C-23

TIME: DATE: 9/2/25

Control on Instruction Hazard:

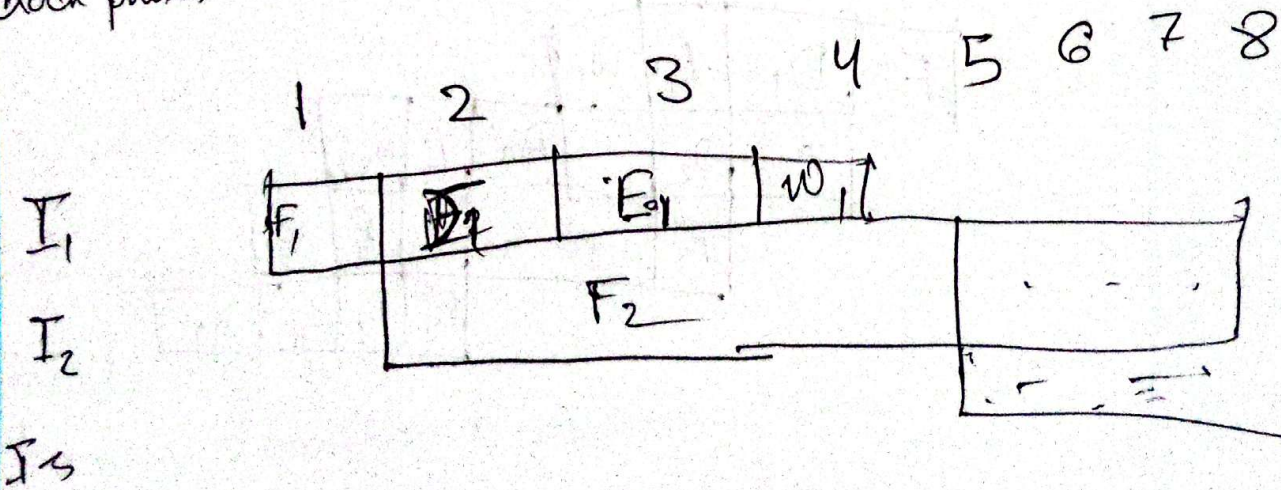
If instructions are not available



Ins 1 ✓
Ins 2 ×
Ins 3 ✓
Ins 4 ✗

If any instruction is missed it is cache miss

We need to fetch this missing instr. which takes a lot of clock pulses which causes hazard



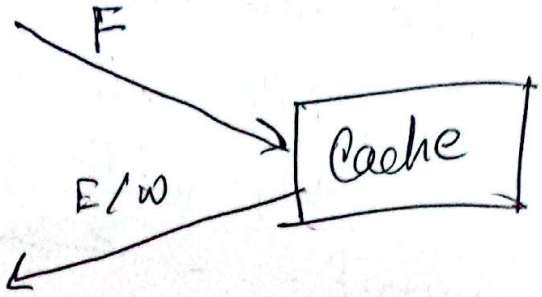
GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Structural Hazard:

If mult. ins uses the same resources.



	1	2	3	4	5	6	7
I_1	F_1	D_1	E_1	W_1			
I_2 (load)		F_2	D_2	E_2	M_2	W_2	
I_3			F_3	D_3	E_3		W_3
I_4				F_4	D_4		E_4
I_5					F_5		D_5

→ Getting from memory

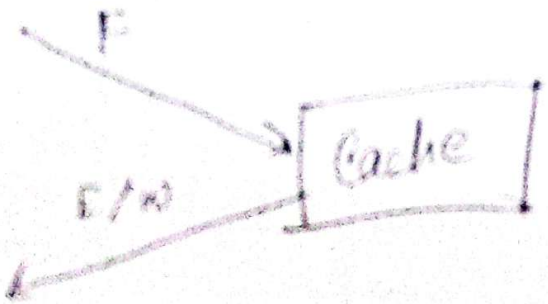
W_3 cannot happen as W_2 is under process

E_3 is holding E_3 result

GOOD LUCK

Structural Hazard:

If multiple ins use the same resource



	1	2	3	4	5	6	7
				W_1		→ Getting from memory	
I_1	F_1	D_1	E_1	E_2	M_2	W_2 → W_3 cannot happen as W_2 is under process	
I_2 (wait)		F_2	D_2				W_3
I_3			F_3	D_3	E_3	→ B_3 is holding I_3 result	E_4
I_4				F_4	D_4		D_5
I_5						F_5	

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

Handling So Data Hazard

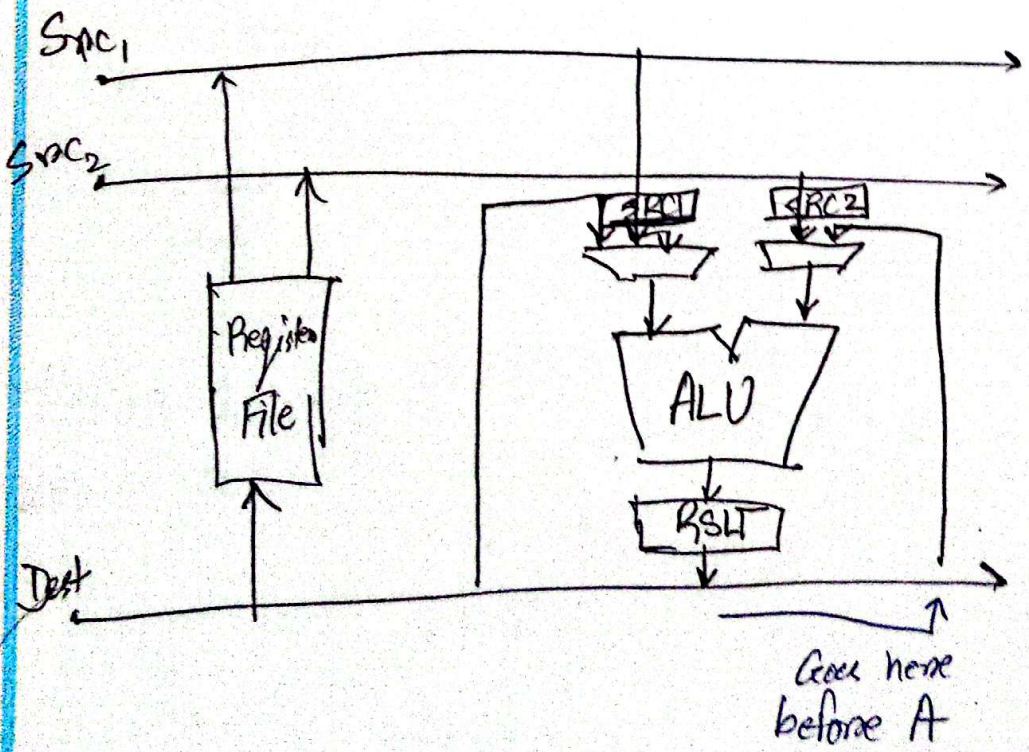
$A = 3 + R_1 \longrightarrow$ W first

$B = 4 \times A \longrightarrow$ E then

We handle it using operand forwarding

But, if we can forward the result obtained in E_1 we wont need the $W_1 - E_2$ dependency

* handled using hardware.



GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

MUL DIV

* MUL src ⟶ only 1 operand
the other is stored in AX

~~MUL~~ ; X * Y
MOV AX, X
MUL Y
} AX = [AX] * [Y]

But for words,

DX : AX
first 8 digits ~~bits~~ Last 8 digits

#For 16 bit result ⟶ AX
" 32 " " ⟶ DX : AX

FFFF
* 1
FFFF → for unsigned (Use MUL)
FFFF FFFF → for signed (Use IMUL)

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

```
  FFFF
 * 1
------
FFFF FFFF
```

this happens as DX is filled with MSB of AX

CF and OF = 0 if AX

= 1 " DX : AX

TOPIC NAME _____ DAY _____ TIME _____ DATE / /

DIV / IDIV

For 8 bit

Quotient $_{bits}$ = Remainder $_{bits}$ = Divisor $_{bits}$

↓ ↓ ↓

AL AH AL

For 16 bit:

Quotient → AX

Remainder → DX

Divisor → AX

	FD
0111 1111	1111 1101
1000 0000	0000 0010
1000 0000	0000 0011
8 1	3

Remainder sign = Dividend sign

CWD → Convert word to double word } Moves AX to DX

GOOD LUCK

TOPIC NAME: W-10 DAY: C-24

TIME: DATE: 12/2/25

Handling Data Dependency: (software edition)

I_1 : MOL R2, R3, R4

NOP } No operation
NOP } to prevent clogging

I_2 : Add R3, R4, R5

Superscalar Operation:

* Multiple processors

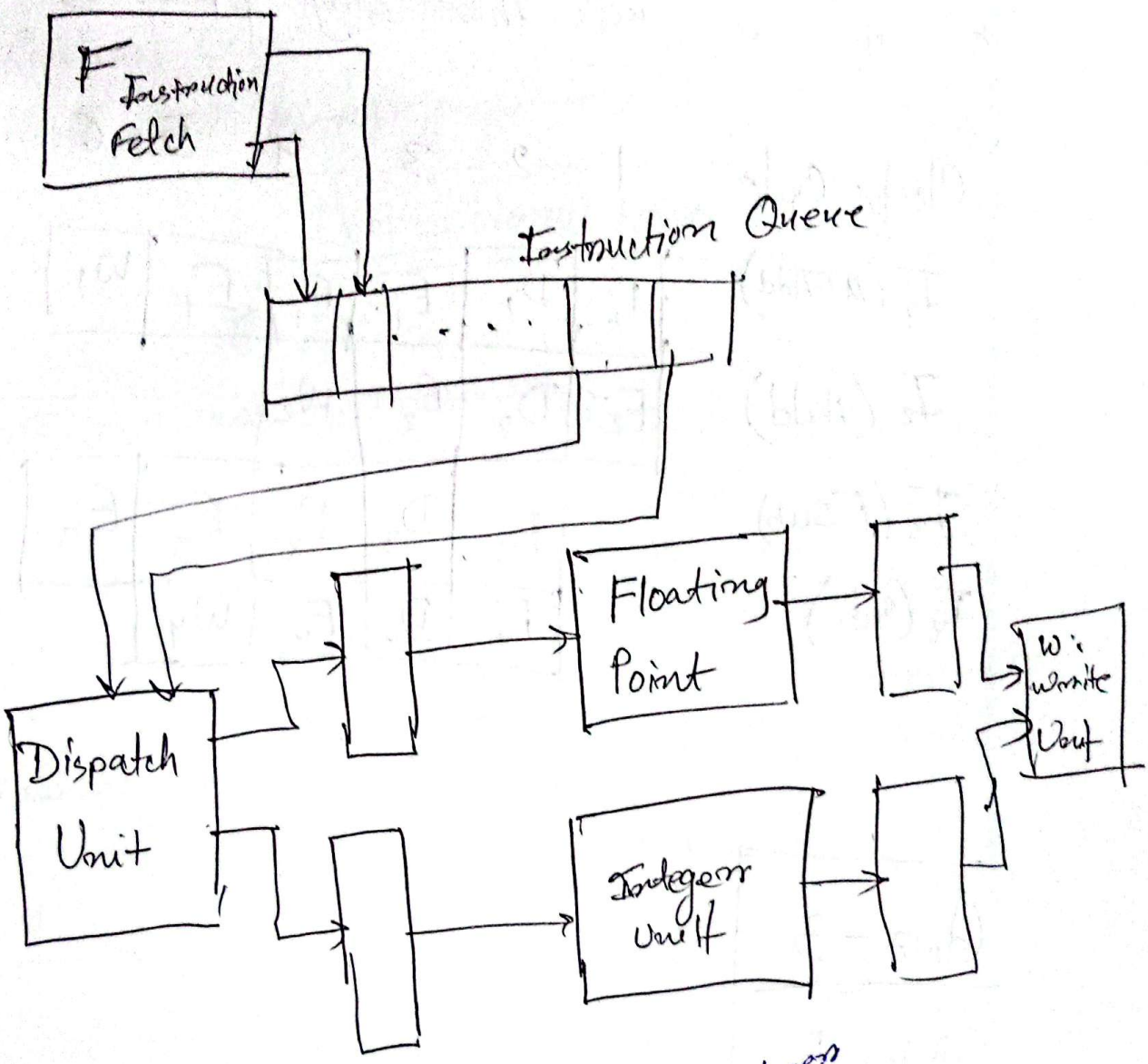
∴ Increase in throughput

* Multiple Processors = Multiple Issue handling processors

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /



* Can handle sequential floating point / Integer unit calculations

* High performance = hardware unit can be kept busy the ~~most of~~ using compilers

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : _____ DATE : / /

* Floating point takes 3 cycles
* " " uses three stage pipelining

Clock Cycle	1	2	3	4	5	6	7
I_1 (FAdd)	F_1	D_1	E_1	E_1	E_1	W_1	
I_2 (Add)	F_2	D_2	E_2	W_2			
I_3 (FSub)		F_3	D_3	E_3	E_3	E_3	W_3
I_4 (Sub)		F_4	D_4	E_4	W_4		

Quiz - 3

Slide - 5

16/2/25

GOOD LUCK

TOPIC NAME : W-10 DAY : C-25 TIME : DATE : 13 / 2 / 25

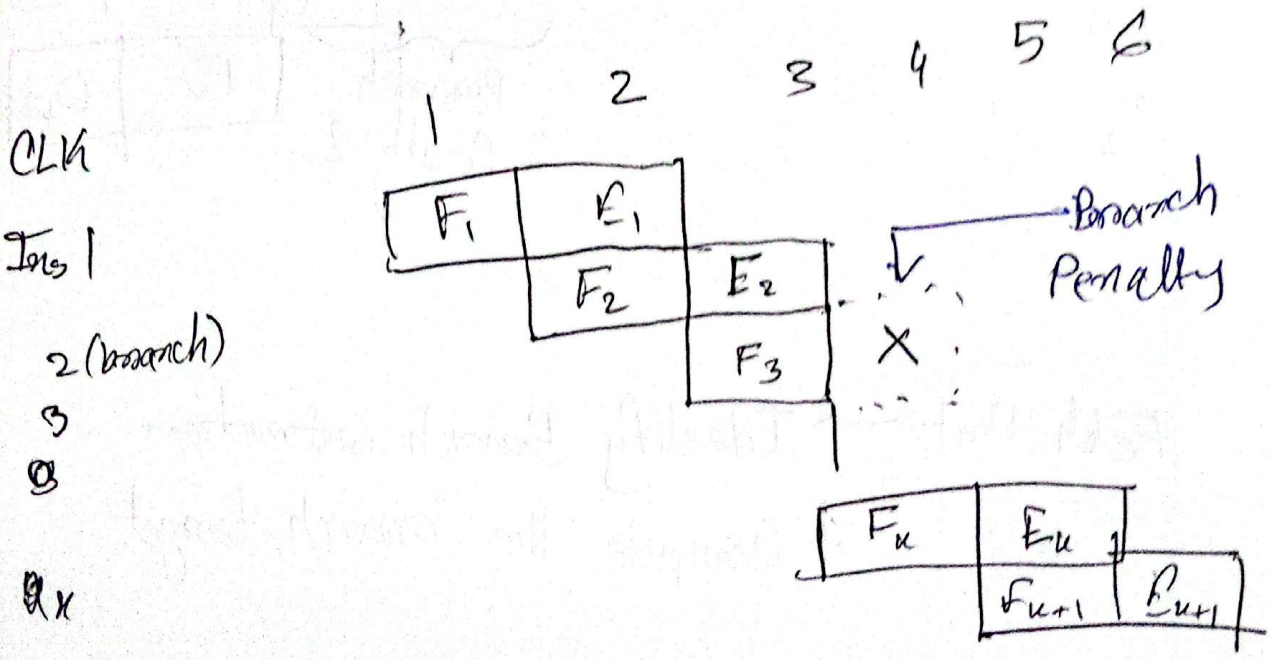
Quiz - 3 : Wednesday

→ Control sequence

→ Hard wired / Microprogrammed

Unconditional Branching

*Discard next step ~~and nop~~

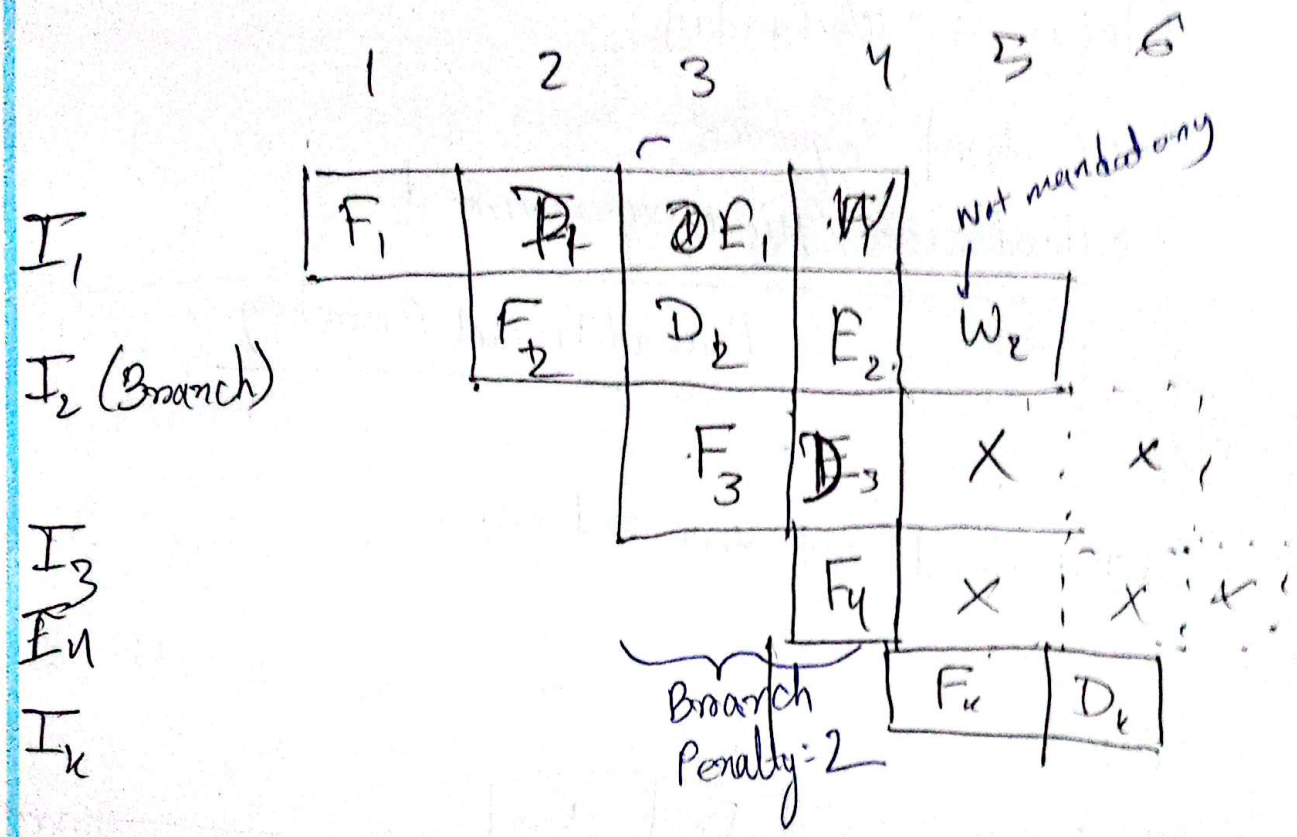


GOOD LUCK

TOPIC NAME :

DAY :

TIME : DATE : / /

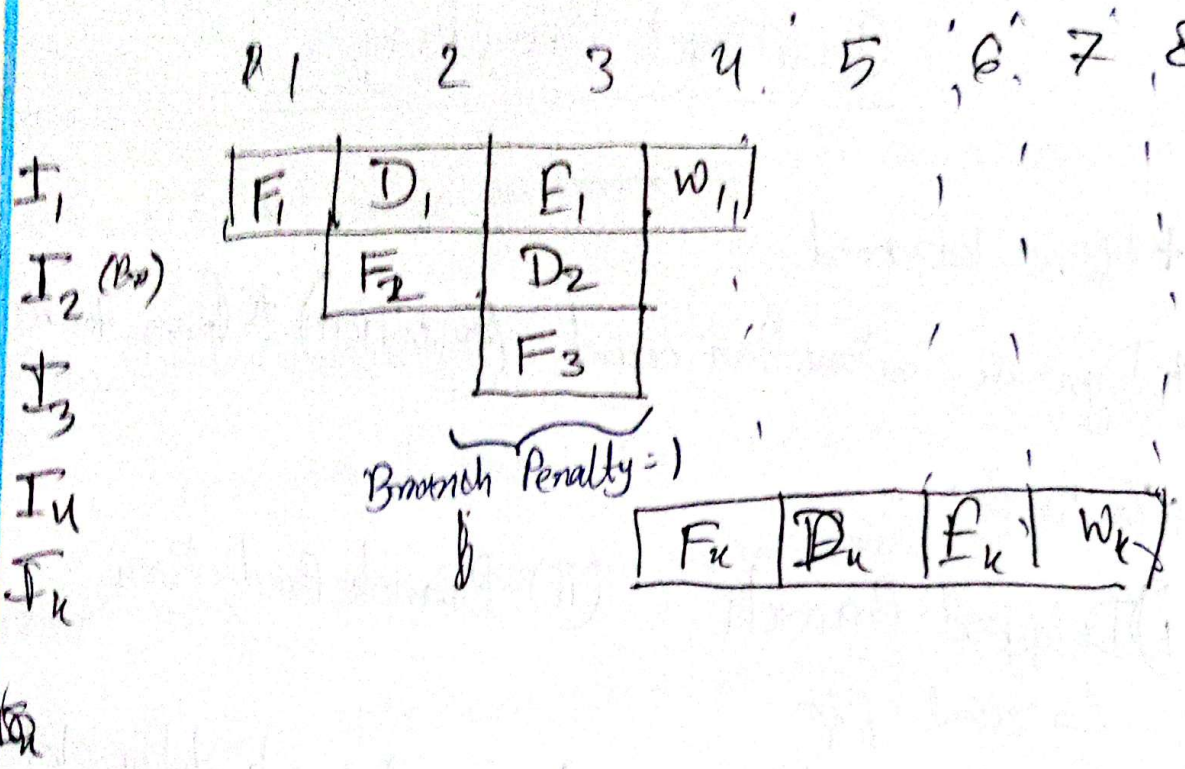


Fetch Unit → Identify Branch instruction
+ Compute the branch target

GOOD LUCK

TOPIC NAME : DAY

TIME : DATE : / /



GOOD LUCK

TOPIC NAME

DAY

TIME : DATE : / /

Conditional Branching

* More hazard
* Dynamic instruction count = (condition) * (lines to be executed)

→ To handle:

i) Delayed Branch (ii) Branch Prediction

↳ check slide

* The instruction after branch is a branch delay slot

GOOD LUCK

TOPIC NAME : W-11 DAY : C-26

TIME : DATE : 16/2/25

Branch Prediction

* Predicts if there ~~is another~~ will be a branch.

Speculative execution:

execute before the processor is certain that they are correct instruction sequence.

→ Next steps are executed assuming Branch will be ignored

→ We use some branch instructions to predict

→ If Branch condition at top → better to predict false
" bottom → " " " true
" " "

- SPARC takes a bit to check
  - → Scalable Processor ARChitecture

#dynamic prediction: Carl Hamacher - Ch-4.

GOOD LUCK

TOPIC NAME: DAY: TIME: DATE:

Performance

$$T = \frac{N \times S}{R} \qquad P_s = \frac{R}{S} = 1 \text{ in ideal case}$$

$$P_p = \frac{R}{T_1} = \frac{R}{1 + S_{miss}}$$

where,

T = Time of execution

N = dynamic instruction count

S = cycles / instruction = 1 in ideal case

R = Clock Rate = $\frac{1}{\text{Clock Period}}$

P_s = Throughput

$$S_{miss} = ((1 - h_i) + d(1 - h_d)) \times M_p$$

- S_{miss}: Average memory in T_1
- $(1 - h_i)$: instr. hit rate
- d: percentage of instructions with data operands
- $(1 - h_d)$: data hit
- M_p: cache miss penalty cycles

GOOD LUCK

TOPIC NAME : W-11 DAY : L-10 ?

TIME : DATE : / /

N!

input → n

↳ xor ah ah

↳ mov n bx

~~mul n~~

12/10 → Q = 1, R = 2

AL → 8 bits

AX → 16 bits

DX:AX → 32 bits

MUL

if multiplied ~~with~~ between bytes and answer comes out word → OF/CF = 1

in signed if MSB(AL) ≠ AH, OF/CF = 1

DIV: Divisor decides what type of division

Rem : Quotient

Divisor DB → AH : AL

Divisor DW → DX : AX

GOOD LUCK

TOPIC NAME : DAY

TIME : DATE : / /

Arrays and Addressing Mode

Arr DW _____.

SI + 1 ⟶ byte

SI + 2 ⟶ word

DUP : Duplicates

name DW times DUP(value)

If offset ⟶ segment

BX, SI, DI → DS

BP → SS

GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE : / /

~~ADD~~

pointer

LEA ~~dest~~ src

[AX]

AX → value

→ [value]

Based and Indexed Addressing Mode

Based Addressing :

Using BP, BX

MOV AX, W[BX]

Indexed Addressing :

Using SI, DI

PTR ⟶ Always moves a byte

LABEL ⟶ Helps to move stuffs automatically

XLAT ⟶ Gets ~~address~~ value from address of AL + BX

GOOD LUCK

TOPIC NAME: W-11 DAY C-27

TIME: DATE: / /

Number of Pipeline Stages

To reduce cache miss, we use secondary cache memory

→ This secondary cache stays between main and primary cache memory

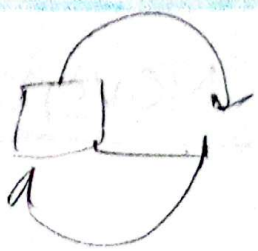
$$S_{miss} = \left((1-h_i) + d(1-h_d)\right)\left(h_s \times M_s + (1-h_s) \times M_p\right)$$

GOOD LUCK

TOPIC NAME: W-12 DAY: L-11

TIME: DATE: 26/2/25

Online-10 :



SUM =

[DI] →

MOV CX, 10

~~DW~~ ~~DX~~ + DW

Storing

DF → Direction Flag : Decides direction of storing

↓ for ABDF

DF = 0 → A (inc)

DF = 1 → E (dec)

STD → Set Direction Flag : (Set DF to 1)

CLD → Clears " " : (Sets DF to 0)

GOOD LUCK

TOPIC NAME : DAY :
TIME : DATE : / /

MOVSB : Mov str Byte

~~Source~~ ~~Dest~~

Dest Src

ES : DI DS : SI

↓

only one byte at a time

to counteer this

MOVSB

REP : repeats CX times

∴ To replicate n bytes :

```
LEA SI, ST1
LEA DI, ST2
MOV CX, N
REP MOVSB
```

For reversing :

```
STD 1
LEA SI, ST1
ADD SI, N
MOV CX, N
LOOP:
MOVSB DI
ADD DI, 2
LOOP
```

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

STOSB: Stores String Byte

AL ⟶ ES:DI

LODSB: Loads String Byte

DS:SI ⟶ AL

SCASB: Scan string Byte

Checks by comparing AL with ES:DI

if ZF ⟶ found

else ⟶ not found

moves address

REPNZ/REPNE: Repeats CX times until equal

REPZ/REPE: ∴ it CAN stop before CX is 0

CMPSB: CMP but for string :O

GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME _____ DATE : / /

General form :

Check Slide

GOOD LUCK

TOPIC NAME : 10-12 DAY : Q-28

TIME : DATE : 26/2/25

The Memory System

ch-5

Cache :

* based on "locality of reference"

- Temporal (Currently executed is most likely to be executed again)
- Spatial (Currently executed is close to the next one)
  - gets a "block" of memory
    - Contigious address locations of some size

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

16 words → 1 block

128 blocks → 2048 words (cache)

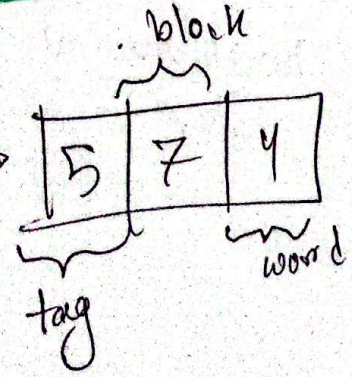
4k " → 64k " (memory)

Mapping

- Direct Mapping
- Associative
- Set Associative

Headache killing me

Of future me



GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Protocol

Write through

↓

updates both cache and main memory

Write Back or Copy Back

↓

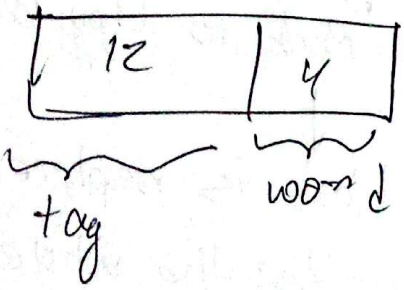
if a block is replaced, it writes the whole block to memory (Modified bit decides if it should be replaced)

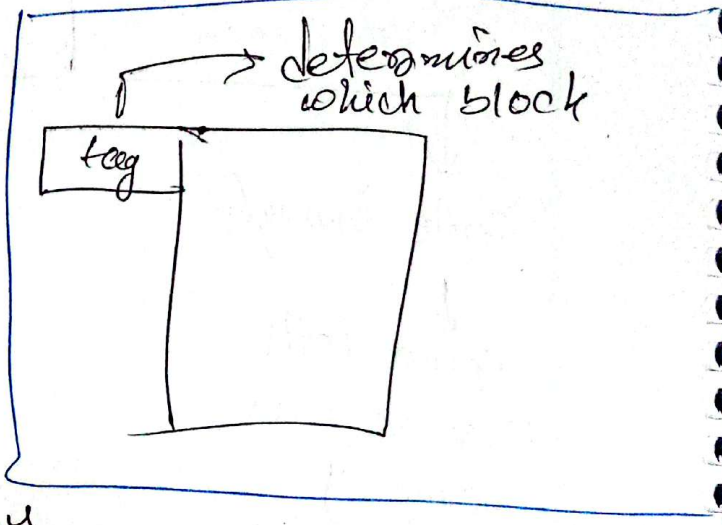
GOOD LUCK

TOPIC NAME : 10-12 DAY : C-29+30

TIME : DATE : 27/ 2 /25

Associative Mapping



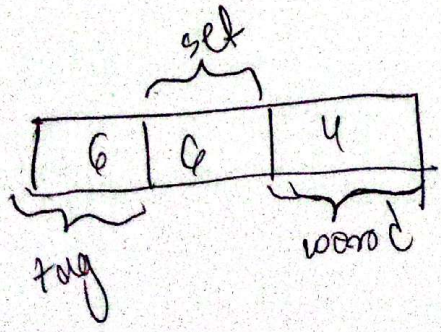


cost is higher as every tag needs to be linearly checked

Set Associative Mapping

Mixture of direct and associative mapping

2 blocks = 1 set; Each block in memory is connected to a set in cache



GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

* Valid bit: 1 if a block contains valid data

0 " " " invalid "

Cache Coherence Problem: In writeback if data in memory ≠ data in cache.

Sol^n: Cache → Mem → Cache

GOOD LUCK

TOPIC NAME: DAY:
TIME: DATE: / /

Replacement Algorithm

* LRU * Oldest

→ * Direct Mapping DOES NOT need replacement algorithm.

* Least Recently Used: ~~Least~~ Oldest

The oldest recent used
→ or least recently used

1, 2, 3, 1
↑

1, 2, 3, 1, 3, 2, 3
↑

GOOD LUCK

TOPIC NAME : DAY :
TIME : DATE : / /

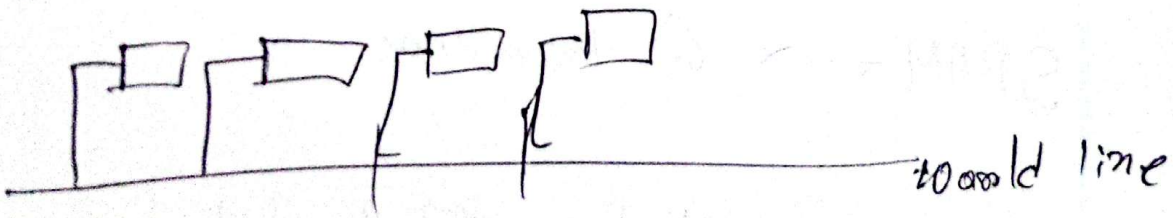
size ∝ speed

but we want higher speed with higher size

RAM :

One row = 1 memory word

~~Each~~ All cells is one connected to a single line

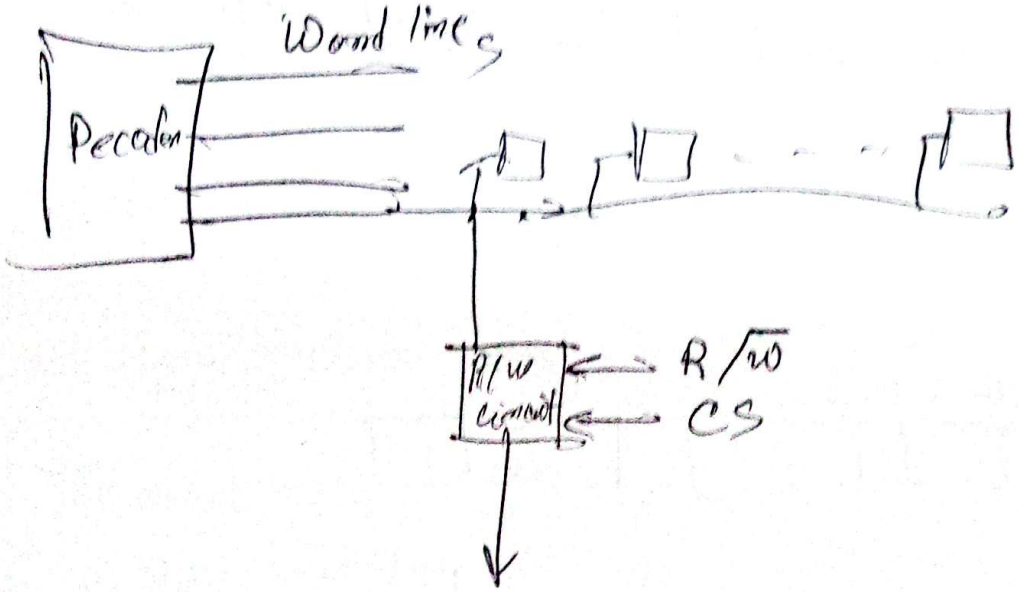


"GOOD LUCK"

TOPIC NAME :

PAGE NO. :

DATE : / /



SRAM ⟶ 6 transistors

DRAM ⟶ 1 transistor, Capacitor

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Latency:

Time taken to transfer a single word or data to or from memory

Bandwidth:

Time to transfer a single block of data

But,

- since block ~~is~~ can be of any size,

~~Time to transfer of bits~~

Number of bits transferred in one second

TOPIC NAME : DAY :
TIME : DATE : / /

ROM :

PROM → Programmable
↳ Write only once

EPROM → Erasable and Programmable

EEPROM → Electrically Erasable Programmable

~~#Qs~~

Flash → Similar to EEPROM
but has higher density meaning higher capacity
→ read one bit
→ write entire contents

"GOOD LUCK"

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Memory Hierarchy:

Speed ↑	Size ↑
1. Register	1. Sec Memory
~~Cache~~ 2. Prim. Cache (L1)	2. Main Memory/RAM
3. Sec. Cache (L2)	3. Sec. Cache (L2)
4. Main memory/RAM	4. Main Cache (L1)
5. Sec. Memory	5. Register

*SIMM → Single Inline Memory Module

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Measuring Cache Performance

For ideal :

- We assume ~~full memory~~ cache as single memory.
- We assume it will always hit

$$t_{avg} = hC + (1-h)M$$

M → Miss penalty from memory

C → Cache access time

$$t_{avg} = \cancel{h_1C_1 + (1-h)C}\ h_1C_1 + (1-h_2)C_2h_2 + (1-h_1)(1-h_2)M$$

GOOD LUCK

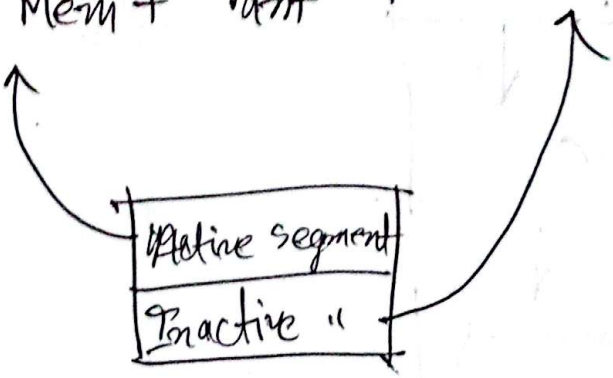
TOPIC NAME : U-13 DAY : C-31

TIME : DATE 02/03/25

#Derivation of performance

Virtual Memory

Main Mem + Part of secondary Memory



Logical Address = (seg : offset)

↑

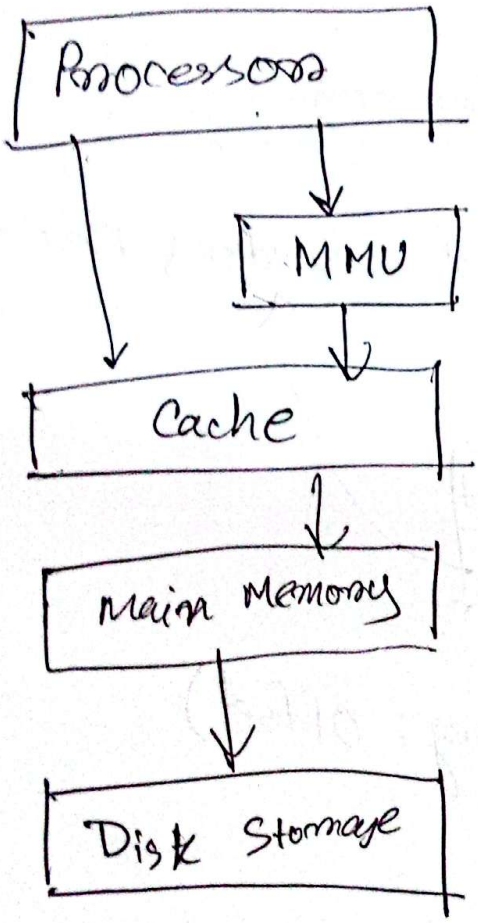
virtual address

Blocks of memories from secondary memory are called pages.

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /



"GOOD LUCK"

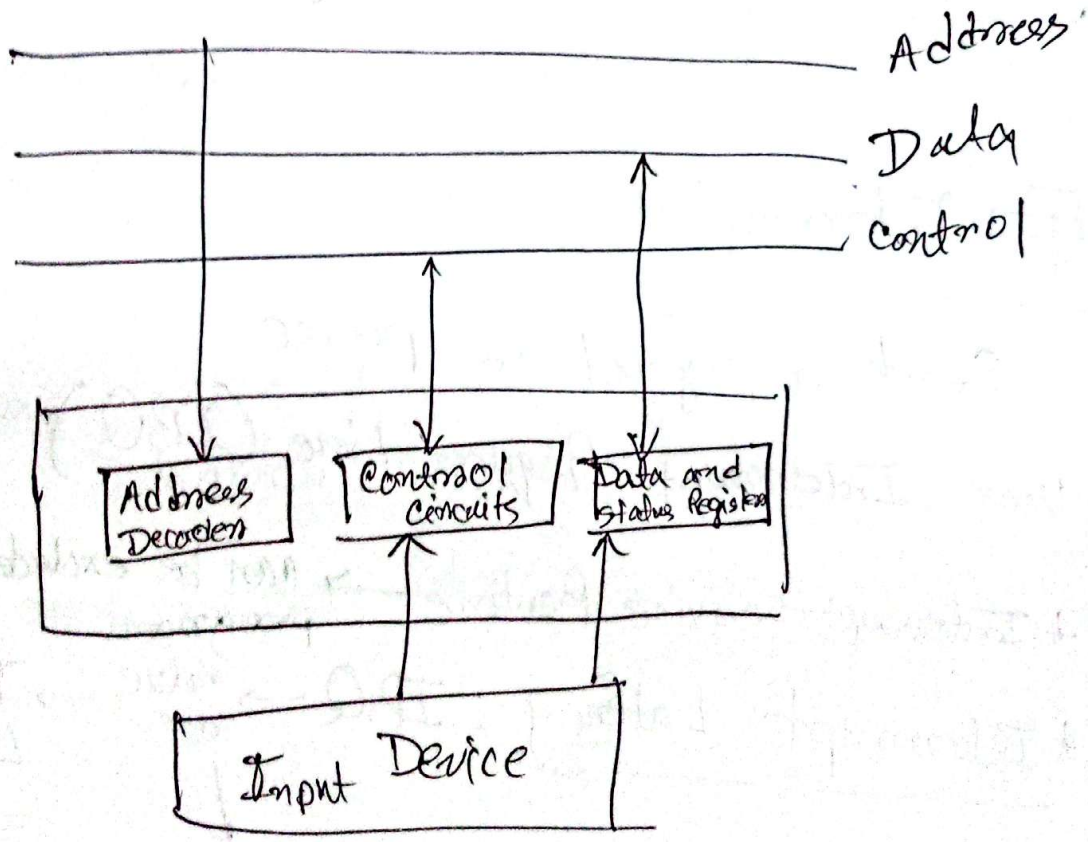
TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

Ch - 4

I/O organization

* Each I/O device has a unique address
Unique addresses are sent through address bus



GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

* Parts of memory are reserved for I/O devices

→ Memory Mapped → uses same instruction to access both

* Different addresses for I/O device

→ I/O mapped → uses less mem space

polls → continuously checks

Interrupts:

Sends a signal to pause

Uses Interrupt Request Line (IRQ)

* Interrupt service Routine → can be excluded from main program

* Interrupt Latency : IRQ → value save → ISR
  ↳ save other register values

causes time called Interrupt Latency

↓

to reduce this, we store PC and Status Flags

GOOD LUCK

TOPIC NAME: 10-13 DAY C-32

TIME DATE 5/3/23

Assignment → 4.4, 4.5

Interrupt Acknowledgement:

I/O lets the device know there is an interrupt

i) ACK/IACK control signal

ii) when addressing I/O, it lets the other know

Interrupt Enable: For letting I/O interrupt

" Disable: " not "

*three approaches from slide

GOOD LUCK

TOPIC NAME : DAY : C-33

TIME : DATE : / /

For Multiple devices:

* Which device calls interrupt:

  → Polling by ISR:
  checks IRQ for every device

  → Self Identification (vectored Interrupts):
  sends 4–8 bits of ISR using data bus
  ↳ this identifies which IRQ is called

* Can other device interrupt:
  ISR time is very small so it doesn't stop
  But some devices do

  → Priority Structure:
  Priority is set
  Higher priority interrupts lower.
  [Fig from slide]

"GOOD LUCK"

TOPIC NAME : DAY
TIME : DATE / /

How to handle simultaneous:

- **Polling:**
  Continuously checks and priority is decided on the basis of connection
- **Daisy Chain:**
  Connected like a chain
  (Sends ACK like a linked list)
  Priority based off of chain position
- **Prioritized Daisy chain**
  ~~Mixtured~~ Combined structure of Daisy and Prioritized
  ~~Indi~~ Same priority has same daisy chain

GOOD LUCK

TOPIC NAME 10-13 DAY C-34 TIME DATE 6/3/25

Direct Memory Access

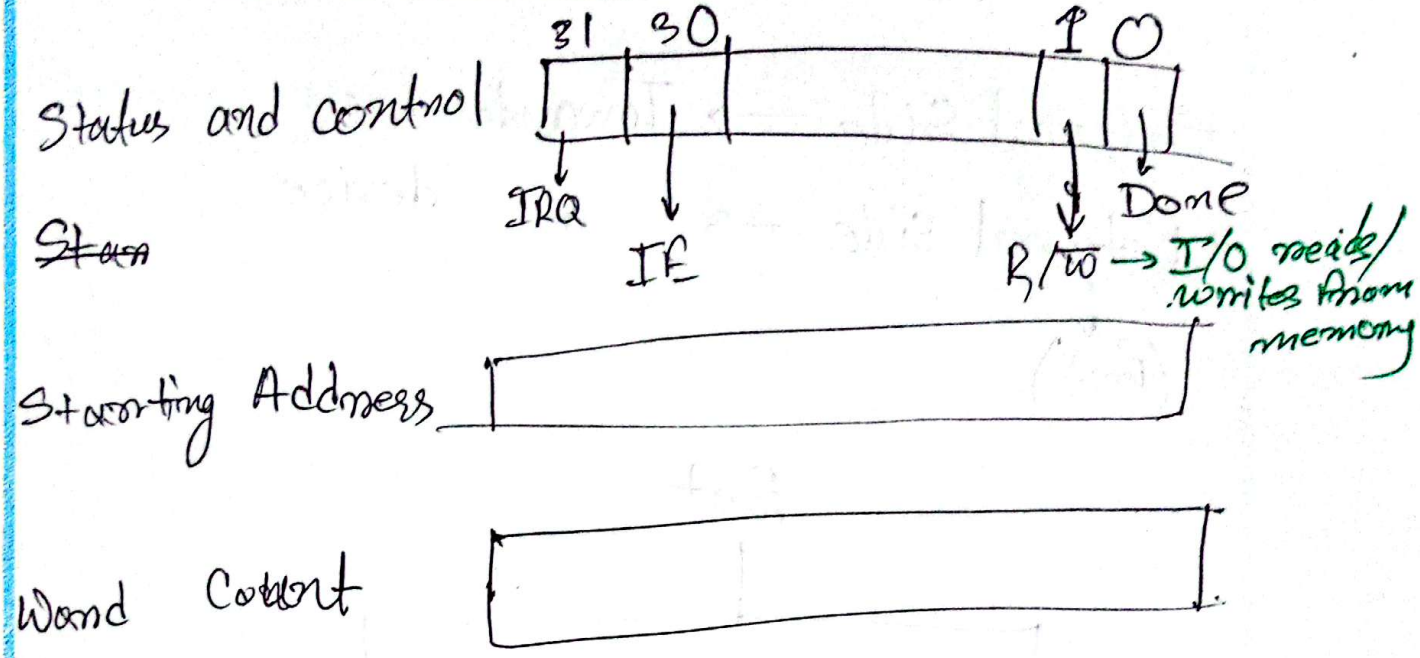
* If I/O is ready to do anything it directly accesses from memory
* Processor is not involved
* Controller Do DMA Controller does these instructions of processor
* Processor initiates DMA transfers.
  with → i) Starting address of memory location
  ii) Number of words in block
  iii) Direction of transfers
* IR knows which these things
* Something something block state
* Block St After DMA's work is finished, it sends interrupt back to processor
* Something something CPU scheming algorithm

GOOD LUCK

TOPIC NAME : _____ DAY : _____

TIME : DATE : / /

DMA has 3 registers:



* Priority of DMA > Priority of Processors
  * Thus, DMA steals cycle from Processors
* Bus Arbitration: decides which ~~bus~~ DMA is bus master

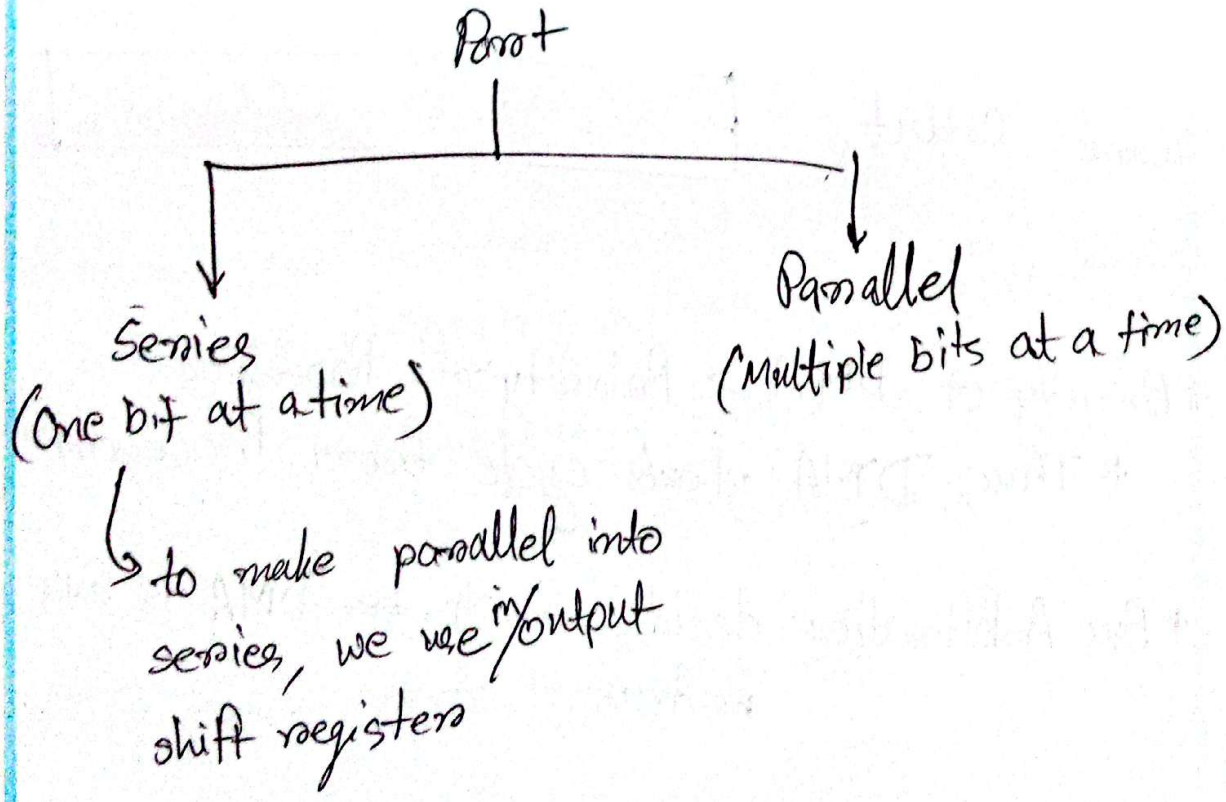
GOOD LUCK

TOPIC NAME : W-14 DAY : C-35

TIME : DATE : 2023/12/25

Interface Circuit

* Internal Side → Towards bus
* External Side → " device
  ↓
  (Port)



↳ to make parallel into series, we use input/output shift register

Sends all bit at once
↳ ~~pro~~ sends parallely

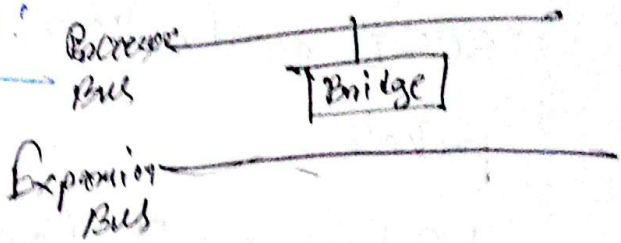
* 9 bits depending on design
* double buffering: at a time two buffers are used
* RS-323C is serial interface

GOOD LUCK

TOPIC NAME : DAY
TIME : DATE : / /

*Something slide

Standards for expansion bus:



ISA

PCI

SCSI

USB

PCI (Peripheral Component Interconnect):

* Plug and play capability
* three address spaces → memory, I/O, Configuration } 4 bits to know which space used
* address is kept until target is selected
* device configuration

GOOD LUCK

TOPIC NAME : W-14 DAY C-36

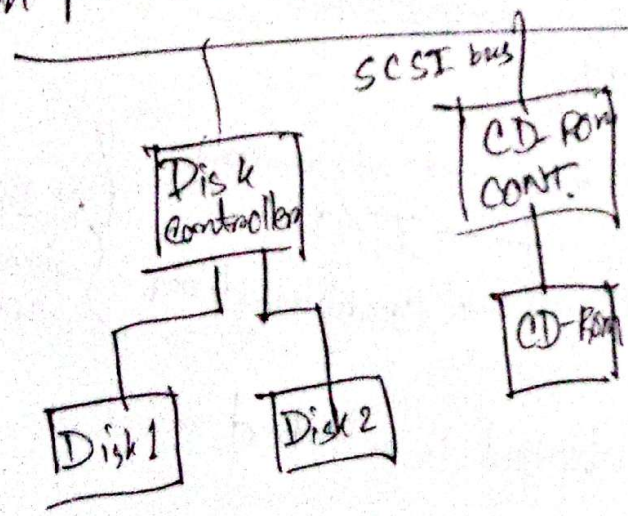
TIME DATE 13 3 25

SCSI (Small Computer System Interface):

* SCSI controllers uses DMA to send data packets

* Packets → Command
  → Block
  →

* We can overlap data transfer requests for high performance



* SCSI controller = Initiator
* " " requests for bus → using distributive
* selects target → Send Bus Control to Target
* Target does data transfer

GOOD LUCK

TOPIC NAME : _____ DAY _____

TIME : DATE / /

SCSCI Phases

i) Arbitration

ii) Selection

iii) Information Transfer

iv) Reselection

~~#to~~ read blocks of data from disk drive

*DONT DO: USB, 4.5 → Multiple Clock Cycle

GOOD LUCK