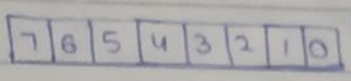
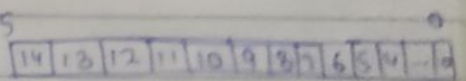
Unit - 1 Register Transfer language (RTL)

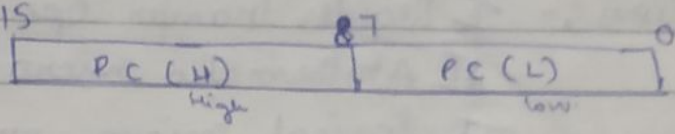
I ⇒ Register - collection of flip-flops (store 1 bit) used to store data ; represented by 'R' / R1, R2

Diff Difference b/w Register and memory

- Various types :- PC (Program Counter), IR (Instruction Reg), MAR (Memory address register), DR (Data Register)

8 bit R 

; 16 Bit 

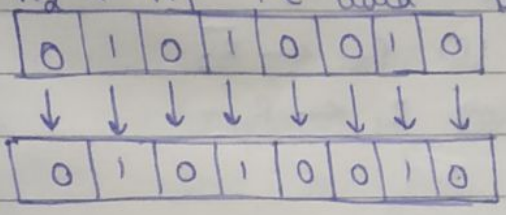
16 bit - 

:. Representation

⇒ Register Transfer - transfer of data from one to another Reg

Let R1, R2 are two registers

$R_2 \leftarrow R_1$ i.e data transfered from R_1 to R_2



⇒ Micro operations :- Operations executed on data stored in registers are called Micro operations

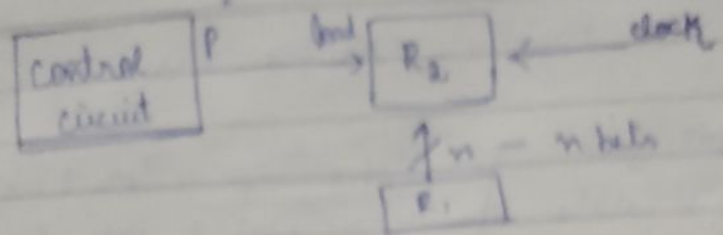
for eg :- Shift (1 bit from L→R / R→L), load, clear

$R_1 \leftarrow R_2$, $R_3 \leftarrow R_4$

if (P = 1) then $R_2 \leftarrow R_1$ /OR/ P : $R_2 \leftarrow R_1$

control function

that controls the transfer



⇒ Memory Transfer operations :- memory Address register
1. Read operation $DR \leftarrow M[AR]$
Transfer data to DR whose address is in AR.
2. Write operation $M[AR] \leftarrow DR$

9-August

Microoperations :- I Register Transfer ope.
II Arithematic micro ope.
III logical micro ope.
IV shift micro ope.

II Arithematic (a) $R_3 \leftarrow R_1 + R_2$
(b) $R_3 \leftarrow R_1 - R_2$ (c) $R_2 \leftarrow \overline{R_2}$
(d) $R_2 \leftarrow \overline{R_2} + 1$ (e) $R_1 \leftarrow R_1 + 1$
(f) $R_1 \leftarrow R_1 - 1$

Ques. 8 Bit registers AR, BR, CR, DR initially have the foll. values: 11110010, 11111111, 10111001, 11101010 resp. Determine 8 bit value in each register after execution of the foll. microoperations.

1. $AR \leftarrow AR + BR$; 2. $CR \leftarrow CR \wedge DR$
3. $BR \leftarrow BR + 1$; 4. $AR \leftarrow AR - CR$

Ans. 1. AR + BR

```
  11110010
  11111111
 (1)11110001
```

= AR = 11110001 Ans.

not to consider

III- logical AND ope - L.M.O specifies binary ope. for string of bits stored in register. list of L.M.O :- logical AND, logical OR

(a) $F \leftarrow 0$ (bits of F will be clear = 0)
(b) $F \leftarrow A \wedge B$ (AND ope)
(c) $F \leftarrow A \vee B$ (OR ope)
(d) $F \leftarrow \overline{A \vee B}$ (NOR ope)
(e) $F \leftarrow \overline{A \wedge B}$ (NAND ope)

Ques: (a) $CR \leftarrow CR \wedge DR$

$CR \wedge DR$ -

$CR = 10101000$

$$\begin{array}{r} 10\ 111001 \\ \times\ 11101010 \\ \hline 10101000 \end{array}$$

→ Applications of L.M.O

For manipulating individual bits or a portion of a word stored in a register. They can be used to change bit values, delete a group of bits, or insert new bit values in a register by performing

we can modify the bits of 1 register for eg:-

1. Selective Set :- it modify bits of A using bits of B and only that bit will be modify which has corresponding 1 bit only. 0 bit has no effect on anyone for eg.

$$\begin{array}{rl} A = & 1\ 0\ 0\ 1 \\ B = & 1\ 0\ 1\ 0 \quad \text{(no effect)} \\ \hline A = & 1\ 0\ 1\ 1 \end{array}$$

S.S. - It set the bits in register A to 1, where there are 1's in corresponding bits of register B. This doesn't affect the (a) bits of register B and (b) bit value in A where there are 0's in corresponding bit of register B.

2 Selective Complement
eg:- A - 1010
B - 1100
A - 0110

Here, when the corresponding register has 1 bit as value then its corresponding value will be changed.

3 Selective clear
eg:- A - 1010
B - 1100
A - 0000

Here, it clears the corresponding bits having value 1.

Imp IV Shift Microoperation :- To move bits either to left side or right side by 1 bit. S.M.O are used for serial transfer of data.
Contents of register can be shifted to the left or to the right. There are 3 types of shift:
(a) logical shift ; (b) Circular shift ; (c) Arithmetic shift

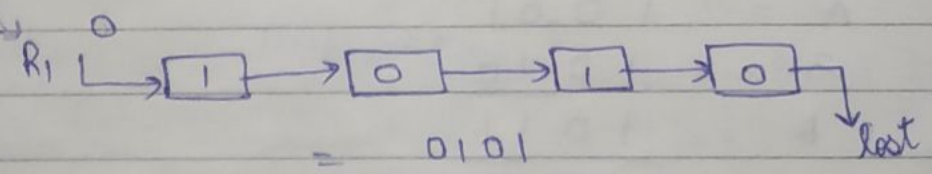
3-aug (a) logical shift

shl
shr

$R_1 \leftarrow shl\ R_1$

$R_1 \leftarrow shl\ R_1$

eg:

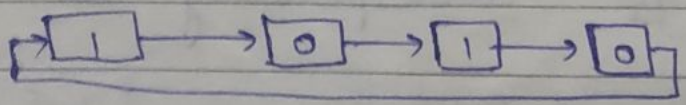


= 0101

(b) Circular Shift (rotate ope.) - Circulates the bits of the register around the 2 end without loss of info. Symbol used is cir. (right), cil (left)

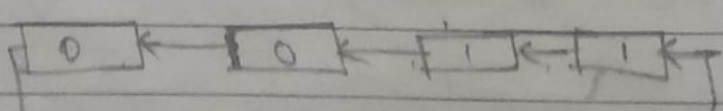
eg:-

$R_1 \leftarrow cir\ R_1$



0101

$R_1 \leftarrow cil\ R_1$



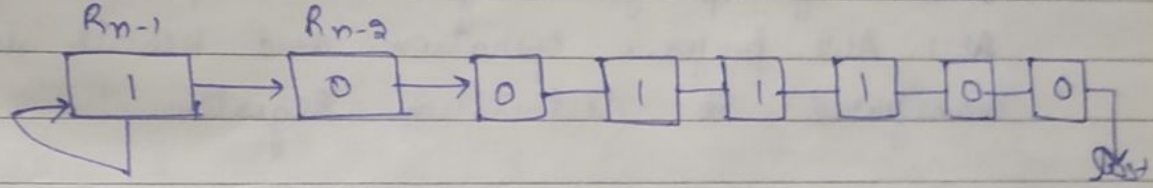
0110

Various shift operations

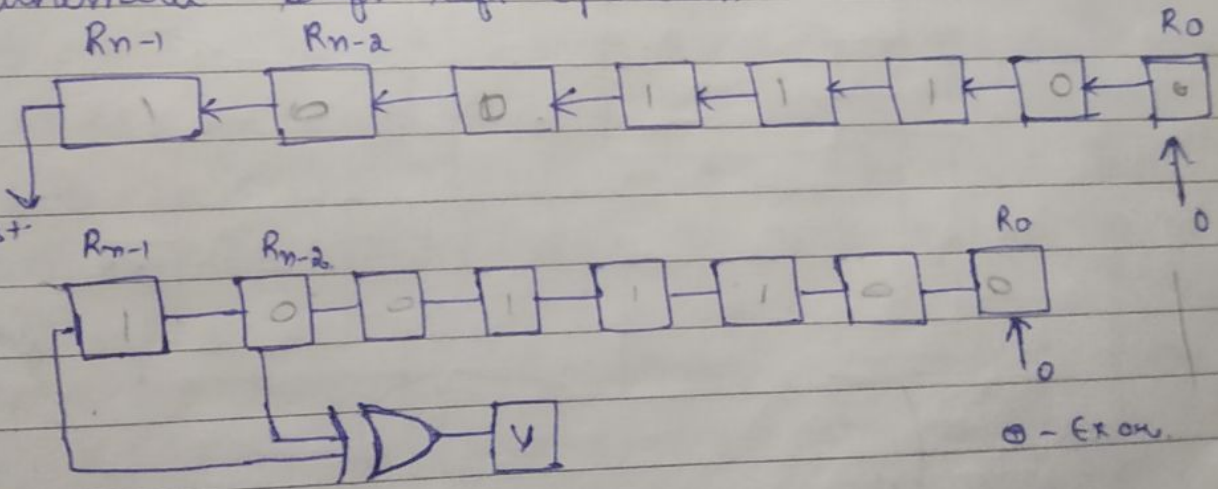
(a) R ← shl R ; (b) R ← shr R
(c) R ← cir R ; (d) R ← cil R
(e) R ← ashl R ; (f) R ← ashr R

(c) Arithmetic Shift - It is a microoperation that shift sign binary number to the left or right The left most bit in the register holds the sign bit and remaining bits hold the number Sign bit is 0 for +ve & 1 for -ve

(i) Arithmetic shift right operation - It leaves the sign no unchanged and also shift it to the Right If register R has n bits then the bit R_{n-1} remains unchanged; R_{n-2} receives the bit from R_{n-1} And so on for other bits in the register. Bit R_0 is lost



(ii) Arithmetic shift left operation



$V = R_{n-1} \oplus R_{n-2}$

→ A.S.L.O inserts a 0 into R_0 and shifts all other bits to the left. Initial bit of R_{n-1} is lost and replaced by bit from R_{n-2}. An overflow occur after an arithematic shift left, if initially, before the shift, R_{n-1} is not equal to R_{n-2}. An overflow flip-flop V can be used to detect an Arithmetic shift left overflow.

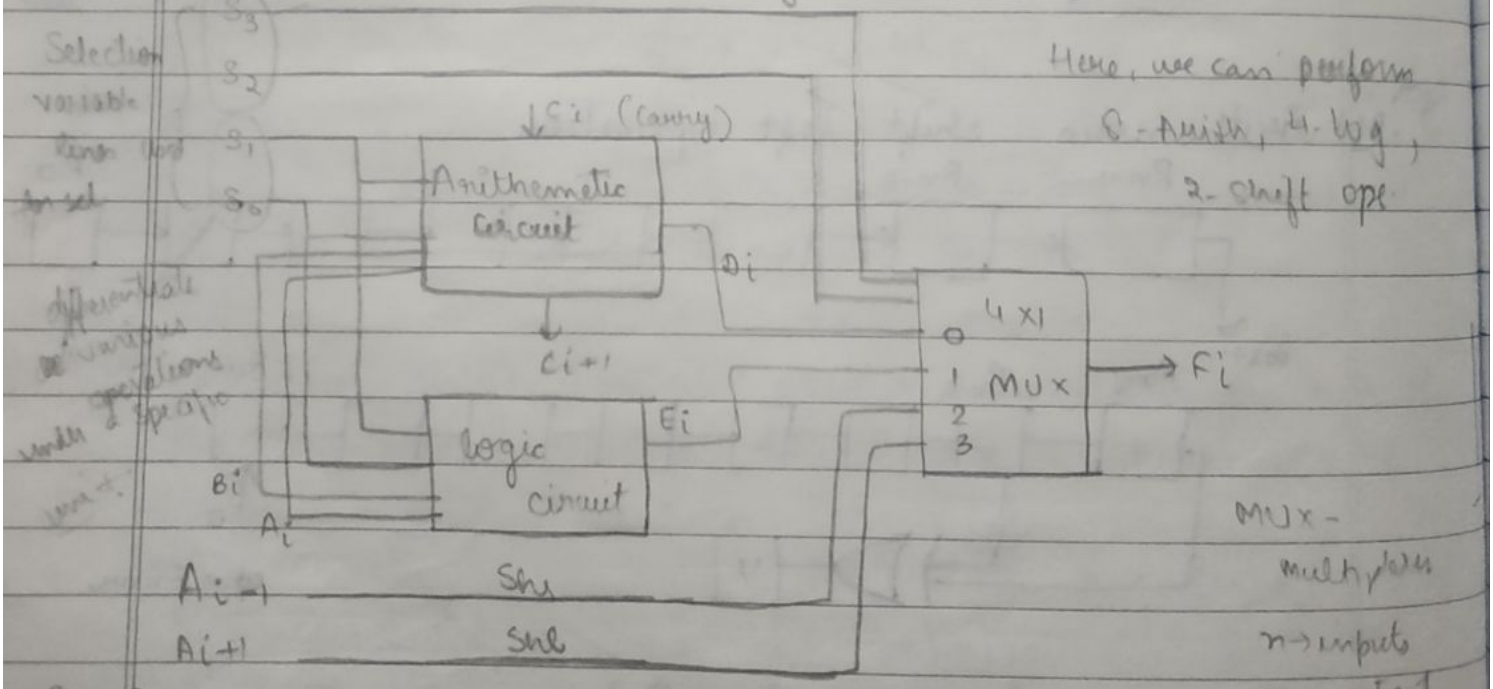
After this draw fig. (b) from previous page. If V = 0, there is no overflow but if V = 1, there is an overflow which indicates sign reversal after performing shift operation.

8-August-2018

Arithematic Logic Shift Unit

ALU is a common operational unit connected to number of storage registers to perform microope. Content of specified registers are placed in input of ALU. ALU perform operation and result is then transferred to destination Register.

tell which unit is working out of 3



arith ope. are selected with $S_3, S_2 = 0,0$
logic ope. are selected with $S_3, S_2 = 0,1$
shift ope are selected with $S_3, S_2 = 1,0$ or $1,1$

	S_3	S_2	S_1	S_0	C_{in}	operation	Funct. with
	0	0	0	0	0	$F = A$	T/FA
	0	0	0	0	1	$F = A+1$	Increment
	0	0	0	1	0	$F = A + B$	Addition
	0	0	0	1	1	$F = A + B + 1$	Add with carry
	0	0	1	0	0	$F = A - B$	Sub
	0	1	0	0	x	$F = A \wedge B$	log AND
log	0	1	1	0	x	$F = A \vee B$	log OR
	0	1	0	1	x	$F = A \oplus B$	XOR
	0	1	1	1	x	$F = \overline{A}$	complement
shr	1	0			x	$F = shr\ A$	shr.
	1	1			x	$F = shl\ A$	shl.

The inputs A_i and B_i are applied to both arithmatic & logic unit - particular microope is selected with inputs S_1 and S_0. A 4x1 MUX at the output chooses b/w arith output in D_i and logic output in E_i. The data in MUX are selected with inputs S_3 and S_2. The other two data inputs to the MUX receives the inputs A_{i-1} and A_{i+1} for shr and shl respectively. The circuit shows 8 arith. ope., 4 logic ope., & shift ope.

9 Aug-18

Unit 2 Basic Computer Organisation and design

Instruction - which instructs the comp to do something
Instruction Code - is a group of bits which instructs the comp to (do something) perform specific operation. Each inst is divided into two parts each having its own meaning. The most basic part of instruction code is the operation code - (set of bits which operates various operation

Part of inst code — Operation Code of an instruction is a group of bits that define such operations as add, subtract, clear, shift etc. The no of bits required for opcode of an instruction depends on total no. of operations available in computer

For Opcode — eg. A comp. perform 16 operation then no of bits req. to perform 1 operation is 4 ($2^4 = 16$)
Similary for 8 op. → no of bits → 3 ($2^3 = 8$)

Instruction code → data

MVI	A,05

→ Instruction

4096 X 16 (each word is 16 bit)

Instruction (prog)
Operand (data)

Stored Program Organisation

$4096 = (2)^{12}$

To represent memory add. we need 12 bits

(mode bit used to specify bit mode)

15	12	0
I	Opcode	Address

I = 0; direct addressing mode
I = 1; indirect addressing mode

0 - 11 - represent dat Address of memory (data)
12 - 15 - 4 bits used to represent OP code
In case of memory data can be accessed directly - indirectly

Opcode	Data

— Immediate case

Opcode	Address

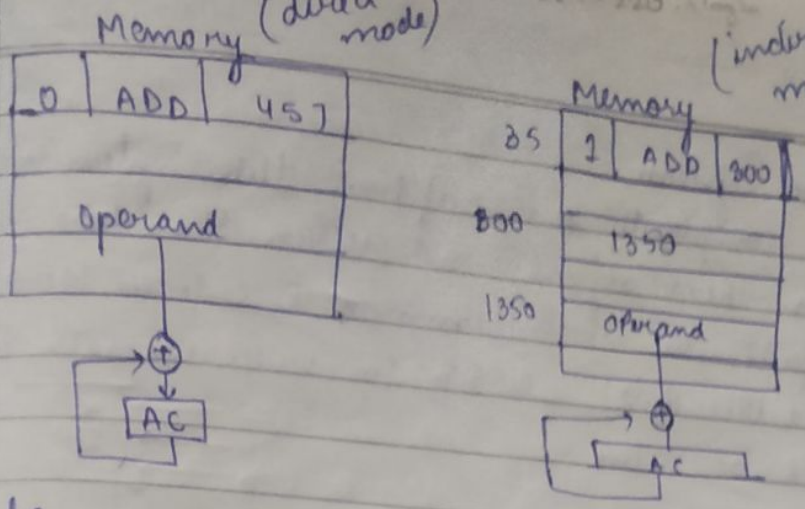
— direct mode

Address	Opcode

— ~~indirect mode~~

Opcode	Add

Add → operand → indirect mod



Control Unit fetch instruction and Put into memory instruction word register

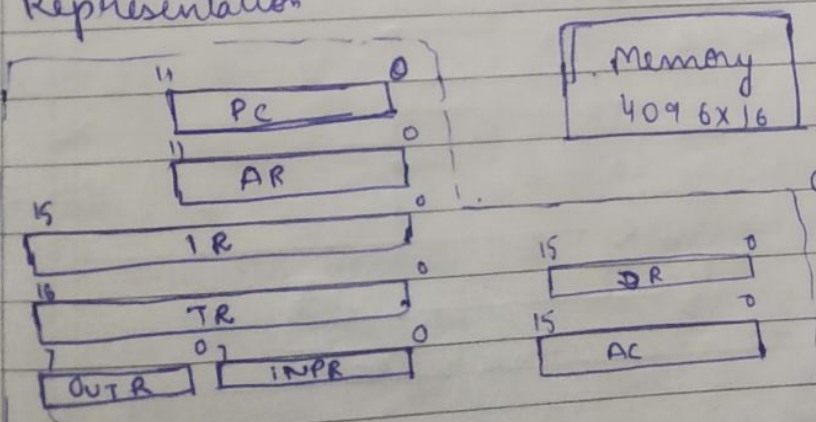
10/Aug/18

Computer Registers :- Registers are used in Control Unit for storing instruction code after it is read from memory

Basic Computer Registers are as follow

DR	Data Register	16	Holds memory operand
AR	Address Register	12	Holds address for memory.
AC	Accumulator	16	Processor register
IR	Instruction Register	16	Holds instruction code
PC	Program Counter	12	Holds address of instruction
TR	Temporary Register	16	Holds temporary data
INPR	Input Register	8	Holds input character
OUTR	Output Register	8	Holds output character

Representation



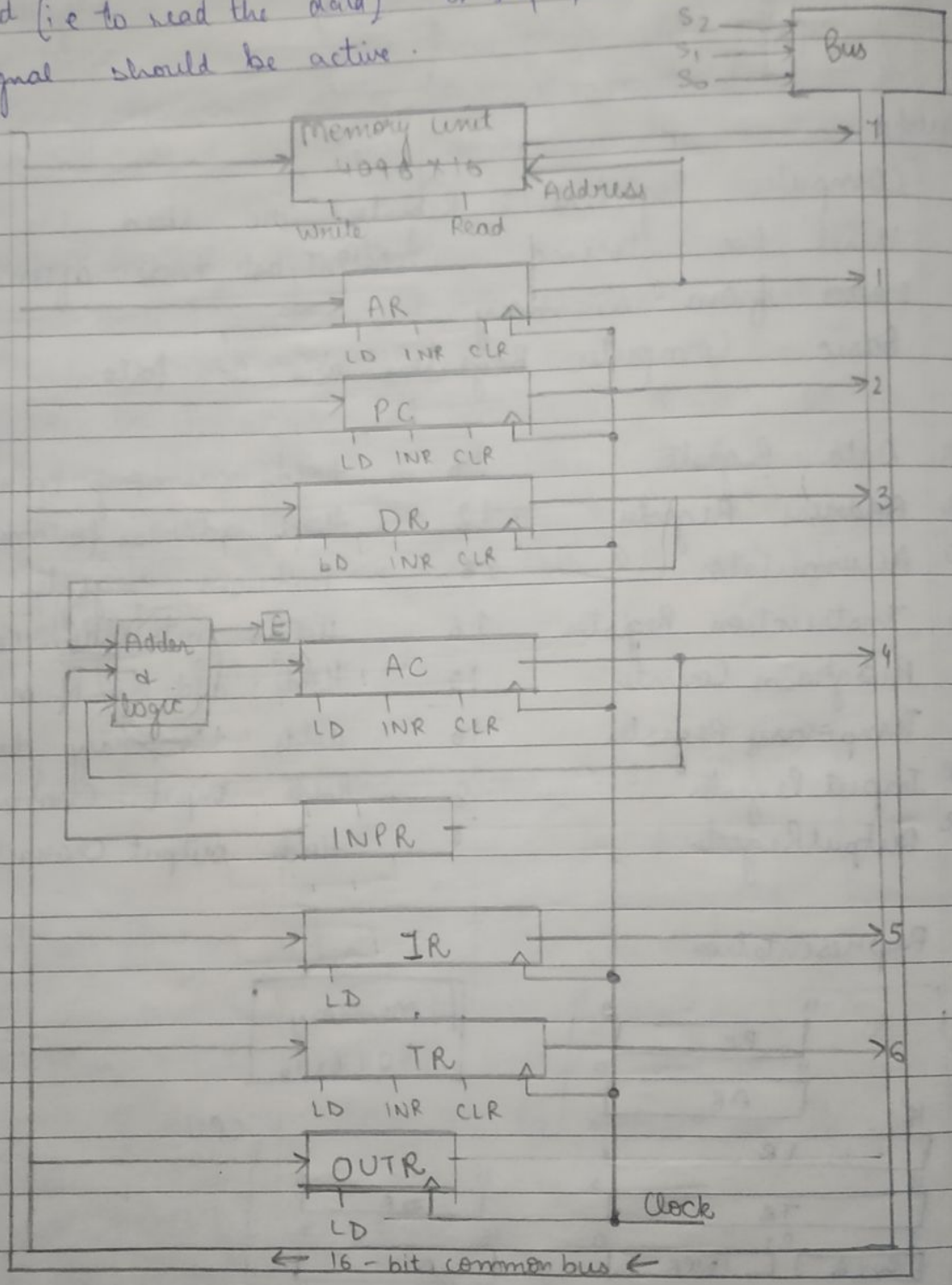
4096 X 16 = 4096 words can be hold by memory and each having 16 bit size.

Bus is a common system that transfers data b/w components inside a computer or b/w computers

Page No. 10.
Date:

Common Bus System

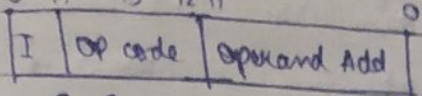
to decrease complexity & no's of lines this System is used. Until or Unless Read signal is not active the data will not be transfered to 16 Bit Common Bus from Memory Unit (i.e to read the data) or $14 for write the data write signal should be active.



Computer Instruction :-

Instruction format : Basic computer has 3 inst code format. Each Format has 16 bits :-

(a) Memory reference ; (b) Register ref ; (c) I/O Ref

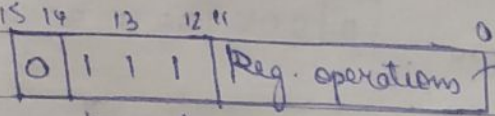
(a) 

I - mode bit

[0-6] , 0, for direct mode
[8-E] , 1, indirect mode
⇒ memory reference

[7, F] → ?

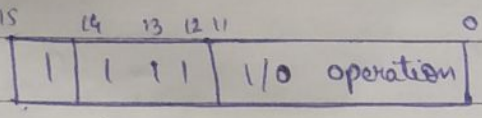
(b) No need to access the memory as data is present in reg.



→ to specify op of reg.

Thus → 7 → reg. ref.

Thus to conclude if 12, 13, 14 bits are combs of
- 0-6 & 8-E ⇒ memory ref.
- (0 1 1 1) 7 ⇒ Reg. ref.
- F (1 1 1 1) ⇒ to I/O ref.

(c) 

Thus F → I/O ref

(a) Memory Reference - This instruction uses 12-bits to specify an add of operand and 3-bits used (12-14) used to specify op code and bit no 15 is used mode bit.

(b) Register Reference - This instruction specifies an operation on accumulator register. An operand from memory is not needed. Therefore, 12 bits (0-11) used to specify op-code ref. to a

(c) I/O Reference - This instruction doesn't need a memory and is recognized by op code (111) with a 1 in a left most bit of the instruction.

The type of instruction is recognised by computer control from 4 bits in position 12-15 of the inst.

* If the 3-op code bit is in position 12-14 are, ≠ 111, instruction is a memory ref ins reference type and bit in position 15 is taken as addressing mode bit (I) * If the 3-op-code bit = 111 control inspects bit in position 15; if it is 0 inst is register reference; if it is 1 inst in I/O reference.

Basic Computer Instructions :

eg (a) AND. 0XXX → Add. of operand. → Add. specifier

0 - logical and.

0	000	0011 0010 0001

direct → AND 8XXX

indirect → 0XXX

1	000	0011 0010 0001
0	000	0011 0010 0001

(b) ADD 1XXX (15 14 13 12) → 0 0 0 1

9XXX (15 14 13 12) → 1 0 0 1

eg (b) CLA (Clear Accumulator)

CLA – (7)8 00 (code) 1000 0111 0000 0000

8 → 7

0	111	1000 0000 0000

15 14 12 11 0

Two digit is always 7

CMA – CMA – (7)2 00

0	111	0010 0000 0000

15 14 13 12 11 0

eg (c) INP F800

1	111	1000 0000 0000

Decoder → n inputs → 2^n outputs

Page No. 17.
Date: 24 | 8 | 18

Timing and control Unit :-
The clock pulse are applied to all flip flops & registers in the control unit The clock pulse do not change the state of register Unless reg is Enabled by control Signal. There are 2 types of Control organisation.

Control unit is of 2 types.
Hard wired C.U & Microprogram C.U

Decoder - is a circuit that converts binary information from n inputs to max. of 2^n unique outputs.

Control Unit - It consists of 2 decoders, Sequence counter & no. of control logic gates. An inst read from memory is placed in instruction reg. which is divided into 3 parts i.e
① Bit I ; Operation code & Bit (0-11).
Operation Code bits (12-14) are decoded with 3 × 8 decoders. 8 o/p of the decoder are represented by symbols D_0 --- D_7. 4 Bit Sequence counter can count in binary from 0-15 The output of counter are decoded into 16 timing signals ($T_0 - T_{15}$). Sequence counter (SC) can incremented or cleared Syncronically. Once a counter is clear, causing next active timing signal D_2 to T_0
For eg :- Consider SC is inc. to provide timing signal T_0, T_1, T_2, T_3, T_4 in Sequence. At T_4 S.C is cleared to 0. If decoded output D_3 is active

$D_3 T_4 : SC \leftarrow 0$

Instruction Cycle

A program in memory consists of sequence of instructions. In Basic computer each instruction cycle consists of following phases:

- Fetch an instruction from memory
- Decode the instruction
- Read the effective address from memory if instruction has indirect address.
- Execute the instruction

T_0 : $AR \leftarrow PC$

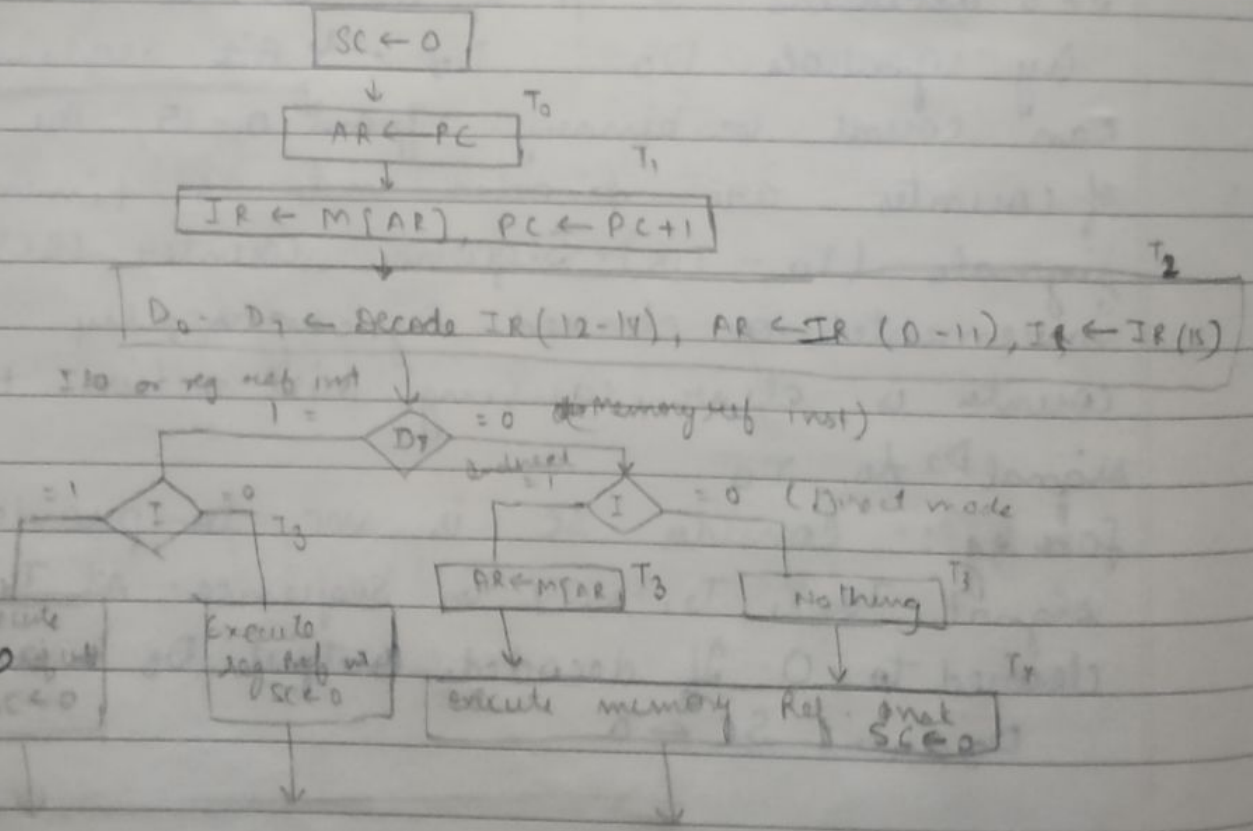
T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

T_2 : $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12-14)$,
$AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

Imp — Determine the type of instruction

At T_3, clock cycle control unit determines the type of instruction that was read from memory.

SC - seq. counter



$D'_7 I T_3$ - $AR \leftarrow M[AR]$

$D_7 I' T_3$ - Nothing

$D_7 I' T_3$ - Execute Reg. inst $\left\{ \begin{matrix} D'_7 = 0 \\ D_7 = 1 \end{matrix} \right\}$

$D_7 I T_3$ - Execute I/O Reg. inst

- Register Reference Instructions :-

① CMA - Complement content of Acc. - 7200

D_7 I 0111 0010 0000 0000

$\hookrightarrow B_9$

$D_7 I' T_3 = r$

$IR(.) = B_i$ (bit in IR(0-11) that specify the operation

$rB_i = rB_9$

② CLE = 7400 0111 0100 0000 0000

$\hookrightarrow B_{10}$

$rB_i = rB_{10}$

③ CLA - 7800 0111 1000 0000 0000

$\hookrightarrow B_{11}$

$rB_i = rB_{11}$

30/August/2018

- Memory Reference Instructions :-

address of operand - effective address

various op.

1. AND - D_0
2. ADD - D_1
3. LDA - D_2
4. STA - D_3
5. BUN - D_4
6. BSA - D_5
7. ISZ - D_6

1. AND to AC

This perform an AND operation on pairs of bits in accumulator at memory word specified by effective address. Result is transfer to accumulator

$D_0\ T_4$: $DR \leftarrow M[AR]$

$D_0\ T_5$: $AC \leftarrow AC \wedge DR$, $SC = 0$

(2) ADD to AC

This add the content of memory word specified by memory address to the value of accumulator

$D_0\ T_4$: $DR \leftarrow M[AR]$

$D_1\ T_5$: $AC \leftarrow AC + DR$, $SC = 0$, $E \leftarrow$ Carry

The sum is transfered to the accumulator and output carry out is transferred to E-flipflop.

(3) LDA : Load to AC

This transfer the memory word specified by effective address to accumulator.

$D_2\ T_4$: $DR \leftarrow M[AR]$

$D_2\ T_5$: $AC \leftarrow DR$, $SC \rightarrow 0$

(4) STA : Store to AC

This inst stores the content of accumulator into memory specified by effective address.

$D_3\ T_4$: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

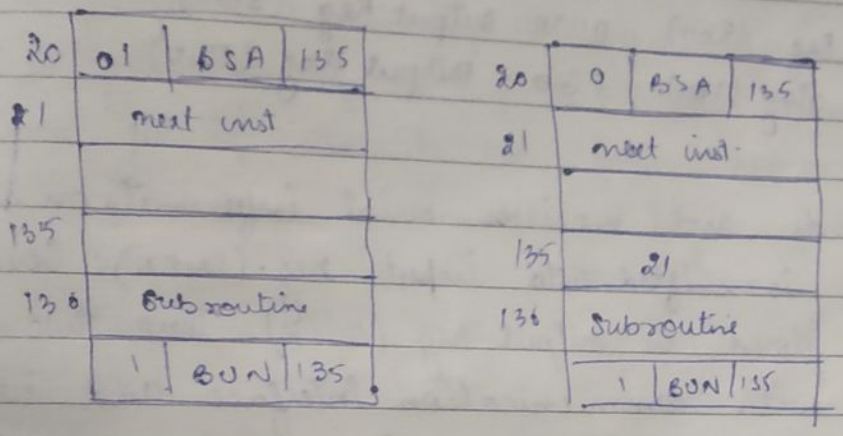
(5) BUN : Branch uncondition

This inst transfer the program to the instruction specified by effective address. P.C called in the add. of the inst to be read from memory in the next inst cycle. PC is incremented at time t_1. to prepare it for the address of next instruction in sequence. The BUN inst. allow the programmer to specify an instruction out of sequence & To say that Program branch is unconditioned.

$D_4\ T_4$: $PC \leftarrow AR$, $SC \leftarrow 0$ (or jump)

(6) BSA - Branch and save return address.
This inst is useful for branching to a portion of prog. called subroutine (set of inst.) when executed BSA inst stores the address of next instruction in sequence (which is available in AR) into a memory location specified by effective address. Suppose EA is then transfer to the PC, to serve as address of 1st inst in subroutine.

M[AR] ← PC , PC ← AR+1



$D_5 T_4$: M[AR] ← PC , AR ← AR+1

$D_5 T_5$: PC ← AR , SC ← 0

31/Aug/2018

(7) ISZ. (increment and skip if Zero)

$D_6 T_4$: DR ← M[AR]

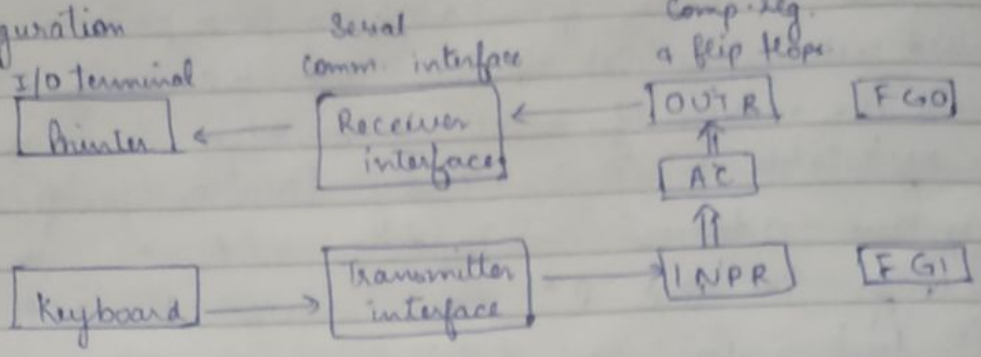
$D_6 T_5$: DR ← DR+1

$D_6 T_6$: M[AR] ← DR , if (DR=0) then (PC ← PC+1), SC←0

This inst increment the word specified by effective address & if ↑ment value = 0 ; PC is incremented by 1. It is not possible to increment a word inside the memory, it is necessary to read the word into DR., increment DR and store the word back into memory.

— INPUT - OUTPUT

• Configuration



⟶ Serial Comm. Path ⟹ Parallel Comm. Path

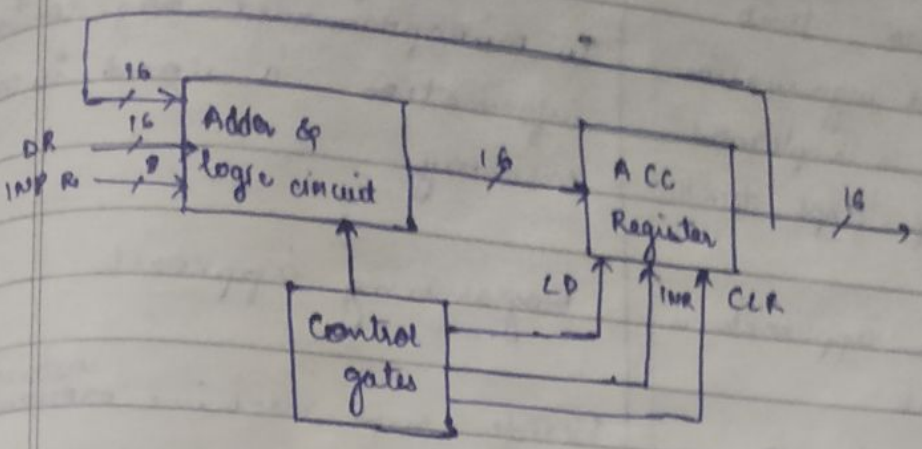
control flow of info. → [INPR - Input Reg. (8bit), OUTR - Output Reg. (8bit)
FGI - Input flag (1bit), FGO - Output flag (1bit)

Terminals sends and receives serial information. Serial info from Keyboard is shifted into input reg. (INPR). Serial info for printer is stored in output reg. (OUTR). These two registers communicate with communication interface and with accumulator in parallel. The transmitter interface receives serial info. from Keyboard and transmit it to input register. Receiver interface receives info. from OUTR and sends it to the printer serially.

5-Sept-2018

Program interrupt.

— Design of Accumulator



One set of 16 bits input comes from output of accumulator & Another set of 16 inputs comes from data register (DR) Third set of 8 inputs comes from INPR. In addition, it is necessary to include logic gates for controlling the signal LD (load), INR (increment) and CLR (clear) and for controlling the operation of adder and logic circuit.

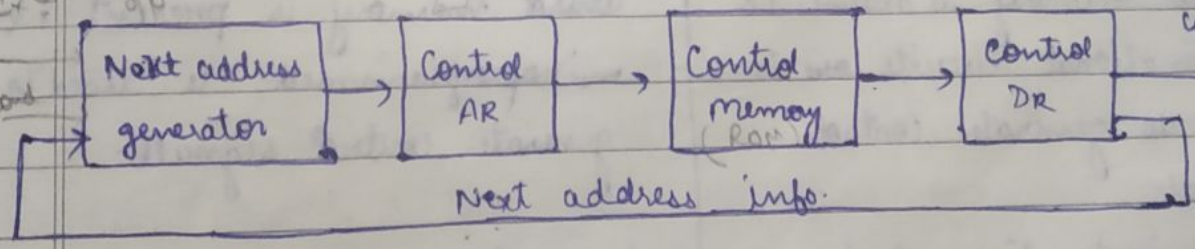
$D_0 T_5$: $AC \leftarrow AC \wedge DR$ AND with DR

$D_1 T_5$: $AC \leftarrow AC + DR$ Add with DR

$D_2 T_5$: $AC \leftarrow DR$ Transfer from DR

— Control Unit

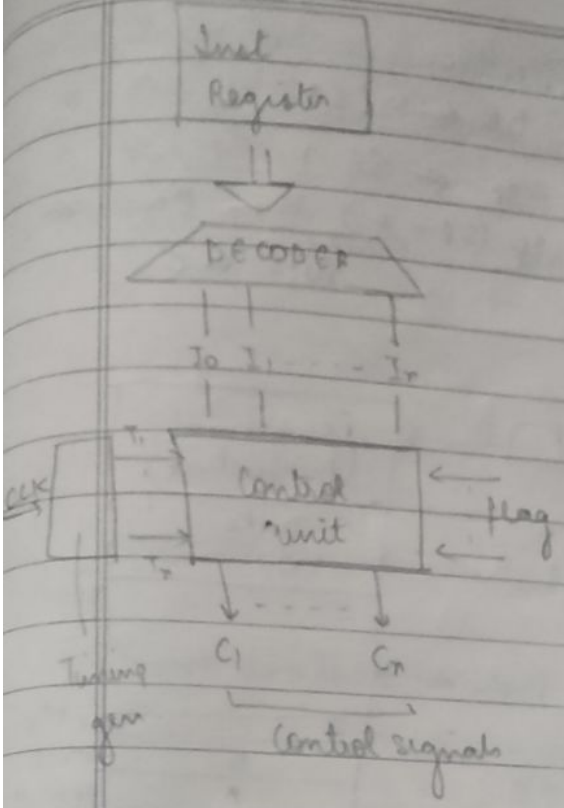
Control word: Set of bits which perform particular op.



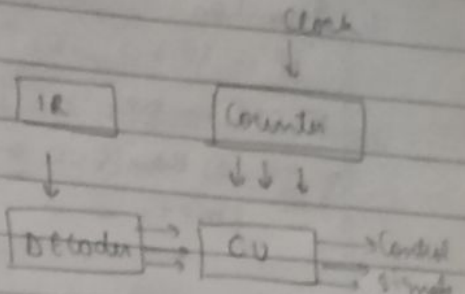
Ans: Diff. b/w Hardwired and Microprogrammed Control Unit.

Date: 07 09 2018

Hardwired control Unit	microprogrammed control unit
1. In a Hardwired organisation control logic is implement with gates, flip flops, decoders etc	In microprogrammed org. control information is stored in control memory.
2. Implementation approach is Squential circuit	Programming approach
3. Not flexible, difficult to modify for new instruction	Flexible, new machine instruction can easily be added
4. It can be optimise to perform fast mode of operation.	It is slower.
5. In Hardwired control, if design has to be modify & changed, it requires changes in the wiring among various components.	Any change or modifications can be obtained by changing or updating microprogram in control memory
6. Risc Microprocesser	Cisc microprocessor
7. Eg:- CPU is with a hardwired logic control are INTEL 8085, MOTROLA 6802.	Eg:- CPU with microprogram control unit are INTEL 8086, MOTROLA 68000
8. Control memory is absent i.e Combinational circuits are is used to generate control signals	Control memory is present i.e microprogrammed is used to generate control signals.
9. Small instruction set size	Large instruction set size.
10. Cost of implementation is higher	Cost of implementation is lower
11. Very difficult to detect fault	Faults can be detected in control unit



OR



14-Sept-2018

Stack Organisation

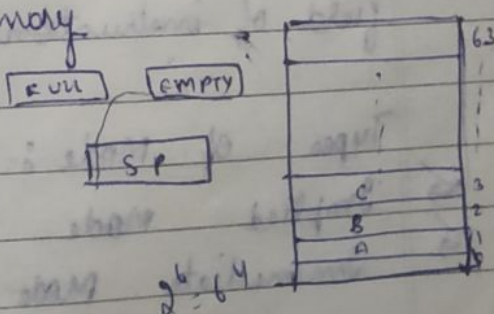
A useful feature that is included in CPU is a stack or LIFO. Stack is a storage device that stores information in such a manner that items stored last is the first item retrieved.

A register that holds address for the stack is called <u>stack Pointer</u> because its value always point at the top item in the stack. Two operations of a stack are insertion and deletion of items. The operation of insertion is called PUSH op. and operation of deletion is called POP op.

Types of stack :- Register, Memory

(a) Register stack :-

Full & empty check and of overflow & underflow bit



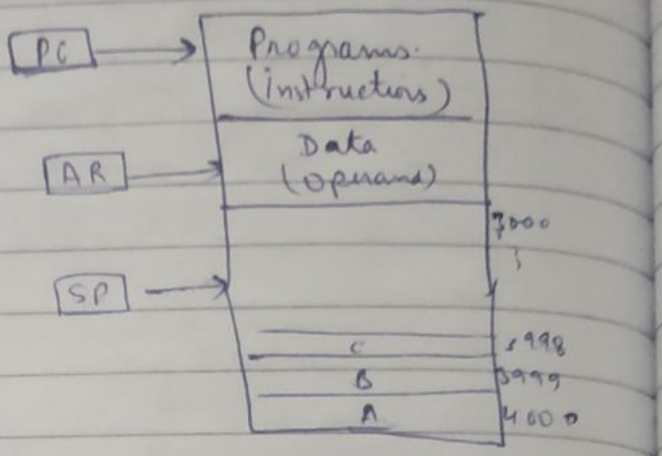
Operations :-

1. Insertion
   SP ← SP + 1
   M[SP] ← DR
   If (SP = 0) then (FULL ← 1)
   EMPTY ← 0

2. Deletion
   DR ← M[SP]
   SP ← SP - 1
   If (SP = 0) then (EMPTY ← 1)
   FULL ← 0

(b) Memory stack :-
Operations :-

1. Insertion SP ← SP - 1
   M[SP] ← DR
2. Deletion DR ← M[SP]
   SP ← SP + 1



20-Sept-2018

Addressing Mode - The way operand is given during execution depends on addressing mode. (way how to access the operand.)
Effective address - address of operand

Comp. uses addressing mode for the purpose of accomodating one or more of the foll. provision-

1. To give prog. versatality to the user by providing such facilities as pointer to memory, counters for loop control, indexing of data and program deallocation
2. To reduce the no. of bits in the addressing field of instruction.

Types of Mode :-

- Implied Mode
- Immediate Mode
- Register Mode
- Register indirect mode

→ Auto increment or Auto decrement mode
→ Direct access mode → Indirect access mode
→ Relative addressing mode → Index addressing mode
→ Base register addressing mode

1. Implied Addressing Mode – Only opcode present, no operand, everything is specified inside the inst no need to give address or data seperately.
for eg :- CMA, HLT, etc. [All inst are of 1 byte]

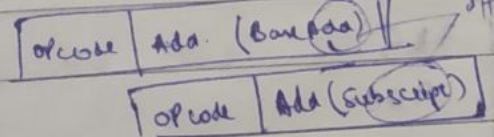
2. Immediate Mode – when data is given in the inst itself.
for eg :- MVI B32H – 2 Byte instruction / 3 Byte (8 bit content / 16 bit content)
LXI 2500H – 3 –

3. Register Mode :- Operand is present in register ∴ Dred.
:- Indirect :-

4. Auto increment or decrement mode

21-sept-2018

5 Direct ac A.M.
6 Indirect A M
7. Relative A.M – E.A = Add. in the inst + P.C
8 Index. A.M. E.A = Add + X R
XR - Index Reg.
A[20] (20 index)



9. Base A.M

Base Reg.
E.A = Sub. + B.R

Ques.

Auto increment - ① fetch ② increment
Auto decrement - ① dec ② fetch

Page No. 27
Date:

Numerical -

200	load to AC	Mode
201	Address = 500	
	Next Instruction	
	⋮	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

PC = 200

XR = 100 Index Reg.

R_1 = 400 Reg. AC

A.M	E.A	Operand
Direct	500	800
Indirect	800	300
Immediate	201	500
→ Relative	702	325
Index.	600	900
Reg. pass	—	~~900~~ 400
Reg. indirect	400	700
Base	800	300

Bit Program Control :-

- Branching Cond – BZ, BNZ, BC, BNC, BP, BM
- Subroutine – insertion
  SP ← SP - 1
  M[SP] ← PC
  PC ← EA

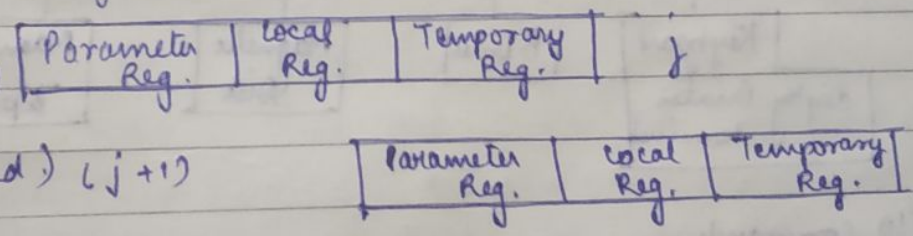
  deletion
  PC ← M[SP]
  SP ← SP + 1
- Program Interrupt

27 Sept - 2018

RISC (Reduced instruction set computer)

charact.

- Realitively fewer instructions
- Realitively fewer addressing mode
- memory access are limited to load and save inst.
- all operations are done within the register of CPU.
- fixed length and easily decoded instruction format
- Single cycle instruction execution
- Hardwired rather than microprogram contrd.
- Overlap register windows.



CISC (Complex inst set computer)

charact.

- Instructions are complicated.
- To represent one instruction large no. of bits are req.
- Decoding of Hardware is complicated so more space & more cost is required.

- fg's – Diff.

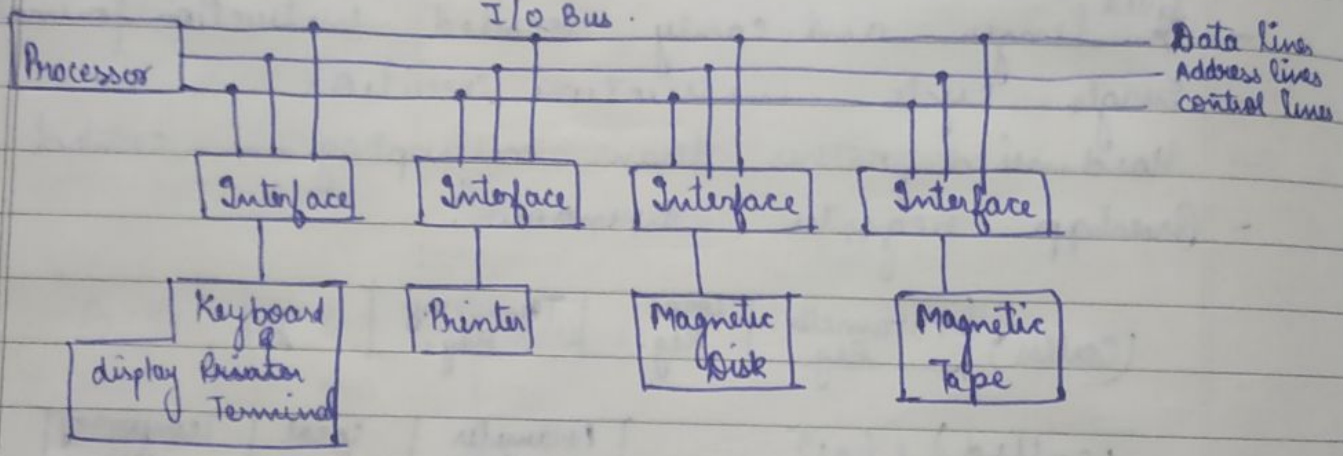
CISC	RISC
1. Data transfred from memory to memory.	Data transfered from register to register
2.	

Ch - INPUT - OUTPUT ORGANISATION

- Peripheral - I/O Interface
- Peripheral — CPU

electromag./electromech devices	Electronic devices
Rate of data transfer slow	fast
Data form - codes.	words

- I/O Bus & Interface Module.



I/O Commands.

- Control commands.
- status command
- Output data
- Input data.

8/2018

Synchronous Data Trans. - In CS, Synchronization refers to the data or idea, keeping multiple copies of data set in coherence with one another or the maintain data

integrity

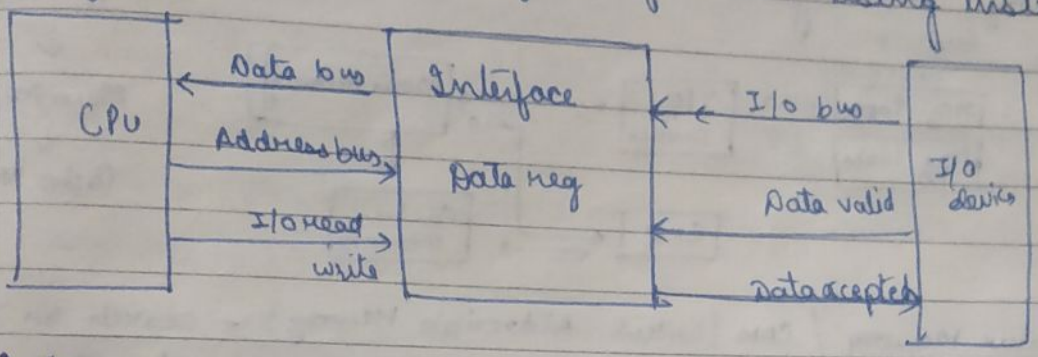
Page No. 27
Date:

Asynchronous data Transfer

1. A type of comm. that sends data using flow control rather

Modes of Transfer 3-Oct-2018

1. Program I/O - transfer of data using instructions



2. Interrupt Initiative

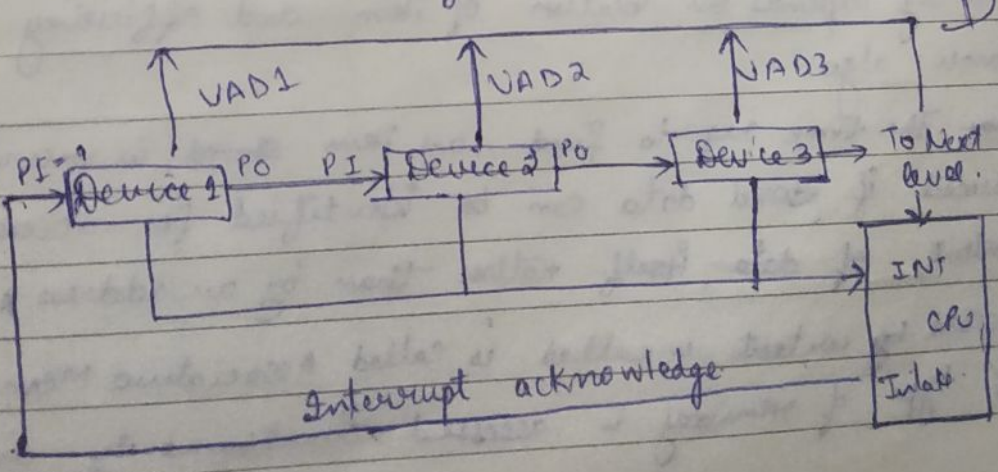
CPU just busy in I/O peripheral devices To overcome this I.I come CPU just do his work and when peripheral devices has to work with CPU they just send a signal to CPU and CPU works accordingly Thus Time is save and CPU works for both.

Priority Interuppt : First that instruction with be executed first which is having highest priority.

Non-Vector Int → control & transfer directly to memory [no Add. is req.]

Vector Int → source gives addres to reach the loc.

Daisy Chain



Page No.
Date: 4 | 10 | 18

- DMA (Direct Memory access)

5-10-18

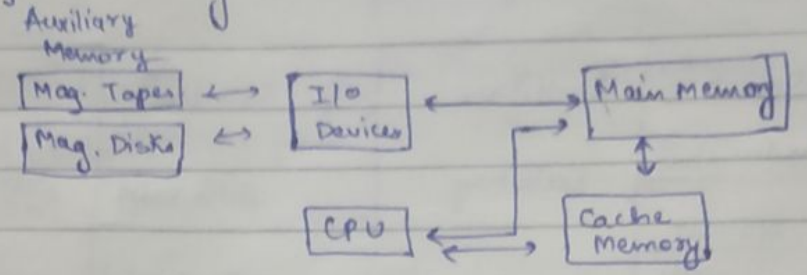
- Input-Output Processor

10-Oct-2018 Section-4

Ch. - Parallel Processing and Pipelining
Lecture

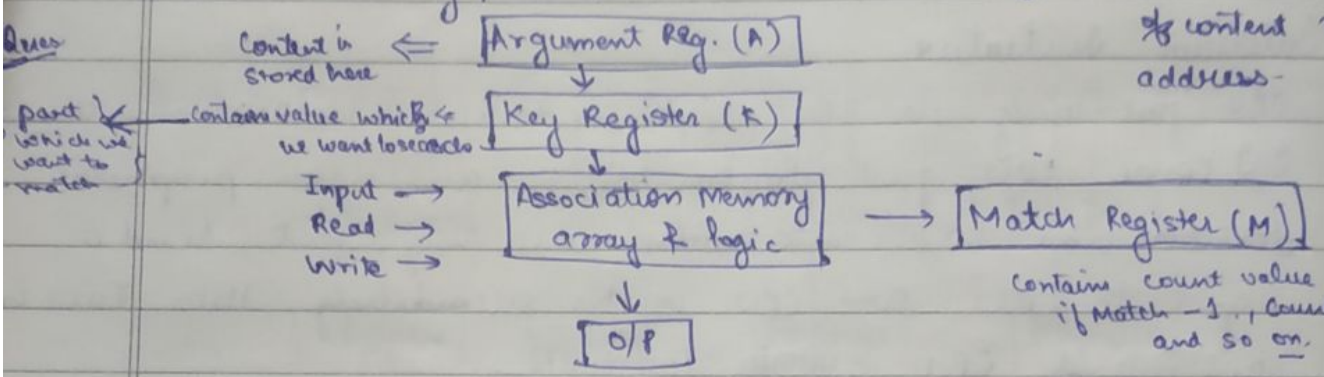
⇒ Intro to Memory

⇒ Memory Hierarchy



2° Memory
↓
Main Memory
↓
Cache Memory

⇒ Associative Memory / CAM (Content Addresible Memory) - Search on the basis of content not address.



The search procedure is ~~stat~~ stargey for choosing sequence of addressing, reading the memory content & compare info, read with the item being search until a match occur. The number of access to memory depends on location of item and efficiency of the search algo.

~~Hem~~ The time req. to find an item stored in memory can be reduced if stored data can be identified for access by the content of data itself rather than by an address. A memory unit accessed by content is called is called Associative Memory or as CAM. This type of memory is accessed simultaneously and in parallel or

the basis of data content rather than by specific address or location. When a word is written in an CAM no address is given. When a word is to be read to from CAM, the content of the word or a part of word is specified.

CAM is more expensive than RAM because each (memory) cell must have storage capability as well as logic circuit for matching its content with an external argument for this reason CAM is used in application where search time is very critical and must be very short.

eg: A = 101 111100
K = 111 000000

[That position will be search only 1 which is assigned at 1 only
assign 1 which u want to match
assign 0 which u don't want to match]

Word 1 = 100 111100 ✗
Word 2 = 101 000001 ✓
} compare with A

⇒ (Match Reg) M = 1

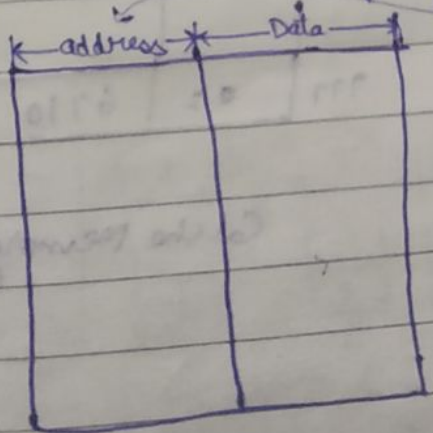
⇒ Catche Memory

12-Oct-2018

If req. Data is present in memory - Hit
not present - Miss. transfer from

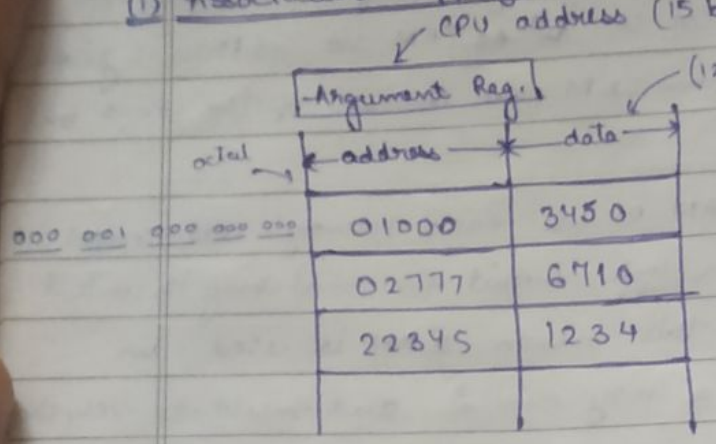
The transferation of data from Main Memory to Catche Memory is referred to as Mapping process. There are 3 types of Mapping:-

(i) Associative Mapping
(ii) Direct Mapping
(iii) Seat - Associative Mapping



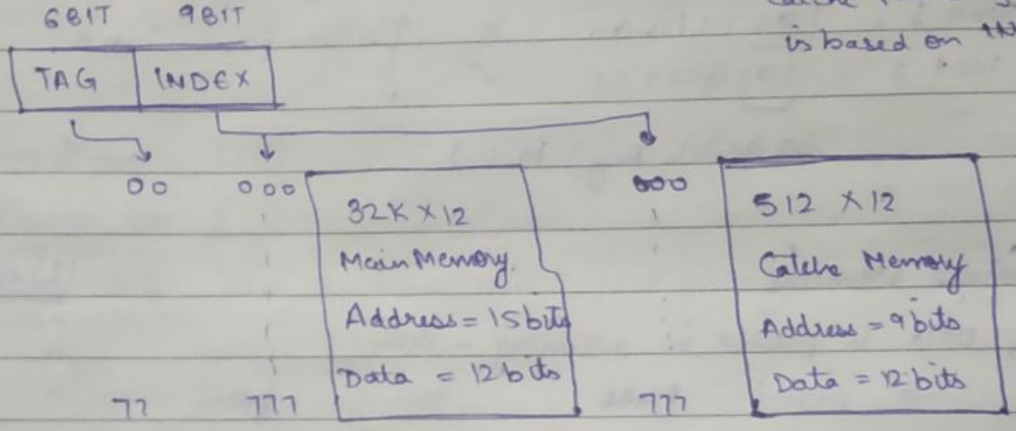
CPU address (15 bit)
Argument Reg

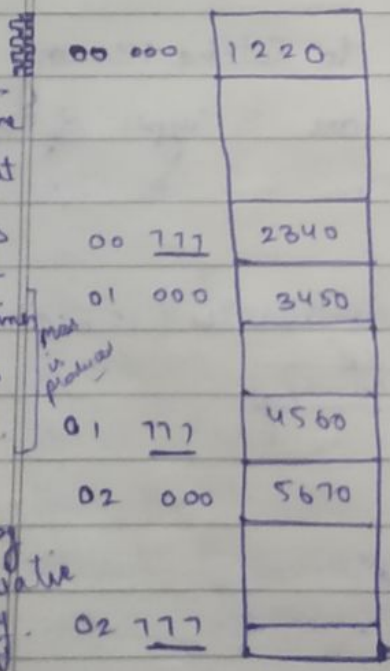
(i) Associative Mapping :- It stores both address and content of memory word

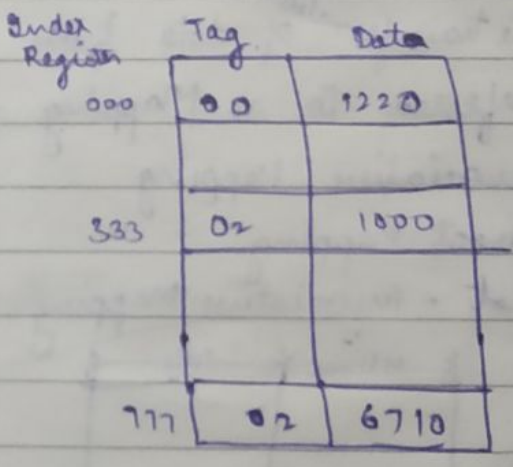


(ii) Direct Mapping :- Associative Mem are expensive comp to RAM. The CPU address is divided into two parts INDEX (9 bits) & TAG (6 bits)

Catche Memory searching is based on INDEX field.







(iii) **Set-Associative Mapping** :- Here ... index, diff Tag are stored in one memory word i.e.

	Tag	Data	Tag	Data
000	01		02	5670
777	02	6710	00	2340

Disadv.: Max size i.e. n word becomes 36 byte

if catche becomes full then on the basis of Algos Replacement Algo's we replace the data.

18-Oct

VIRTUAL MEMORY

Virtual Memory is implemented by Demand page

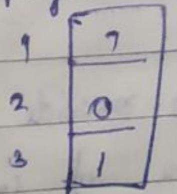
Page fault - If CPU refers to any page in memory but doesn't get it → known as page fault.

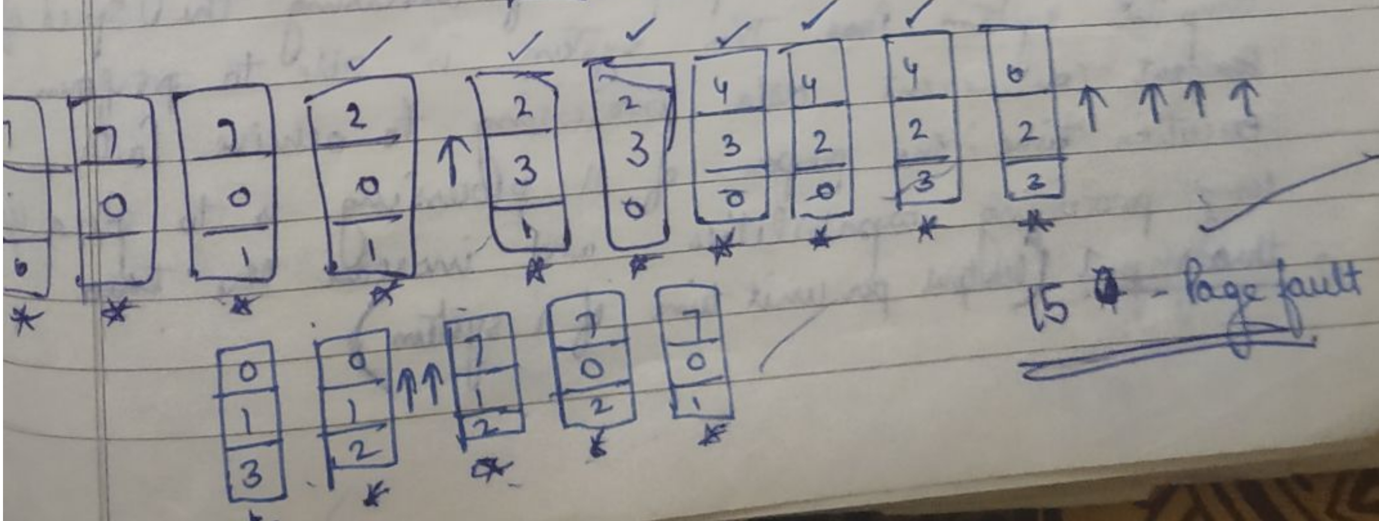
To remove this error - (steps to handle page fault) :-

Numericals of Algos Imp!

FIFO (Algo.

- Reference string :- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.
- 3 frames (3 pages can be in memory at a time per process).





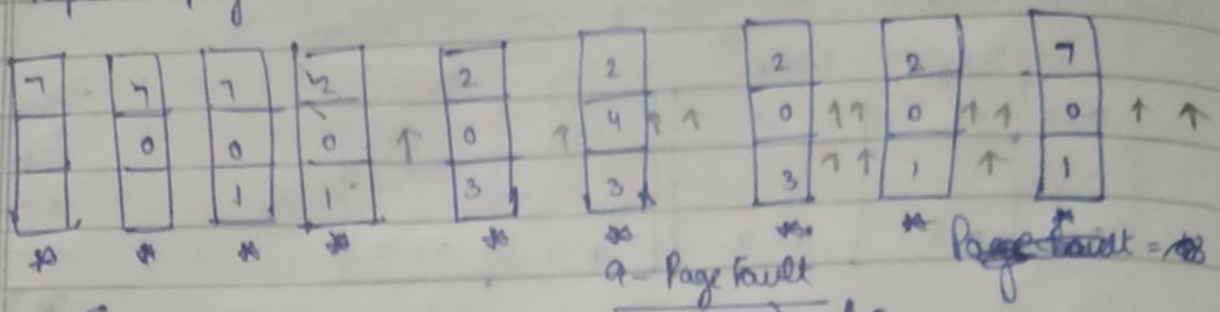
15 - Page fault

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Page No.
Date:

Optimal Page Replacement Algo.

Replace page that will not be used for longest period of time



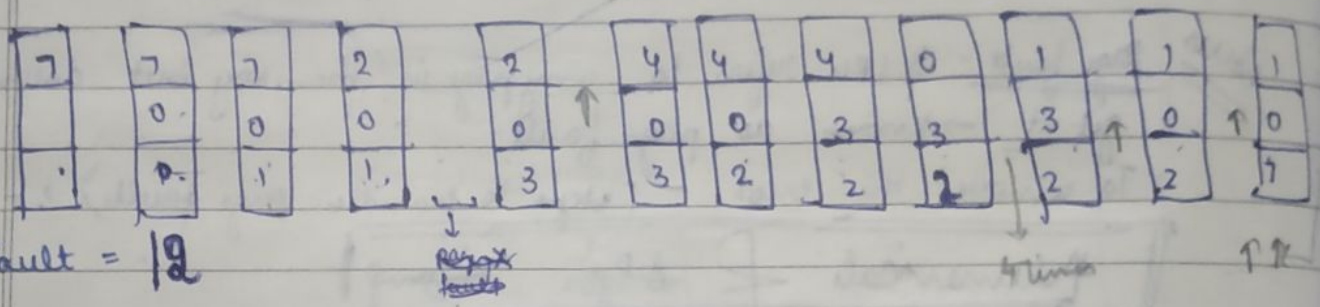
9 - Page fault

Page fault = 9

Least Recently Used (LRU) Algo.

Use past know - rather than future

Replace page that has not been used in the most amount of time



Page fault = 12

At last - calculate Page Fault ? [To be solved]

(10 Page fault)

25-Oct-2018

Pipeline & Vector Processing

Parallel Processing : It is used to provide simultaneously data processing task for the purpose of increasing the speed of computer system. Here This system is able to perform constant concurrent data processing to acheive faster execution time. The purpose of || processing is to speed up comp. processing capabilities and increase the throughput (output per unit time of a system)

Page No.
Date:

Amount of processing that can be completed during a given interval of time - Throughput.

There are various ways that II processing can be classified:

1. Single Instruction Single data
2. Single Instruction Multiple data
3. Multiple Instruction Single data
4. Multiple Instruction Multiple data

II processing comes under following topics:

1. Pipeline processing
2. Vector processing
3. Array processing

- Pipeline Processing :- { just divide a task into subtasks }

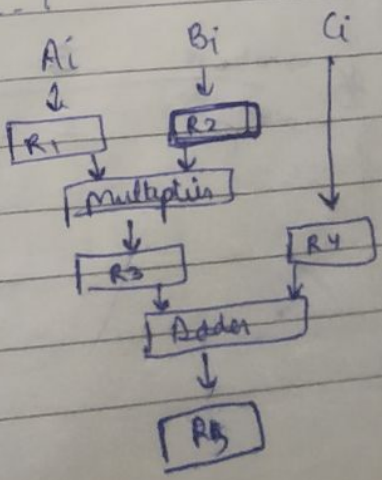
- It is a technique of decomposing a sequential process into suboperations & each subprocess being executed in a segment that operate concurrently with all other segments. Each segment performs partial processing by ___ the task is partioned. Result obtained from each segment is transport to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

Eg. $A_i * B_i + C_i$ $i = 1, 2, 3 \ldots 7$

$R_1 - R_5$

3 segments

1. $R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$
2. $R_3 \leftarrow R_1 \times R_2$, $R_4 \leftarrow C_i$
3. $R_5 \leftarrow R_3 + R_4$



clk Pulse No	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
1	A_1	B_1			
2	A_2	B_2	$A_1 \times B_1$	C_1	
3	A_3	B_3	$A_2 \times B_2$	C_2	$A_1 \times B_1 + C_1$
4	A_4	B_4	$A_3 \times B_3$	C_3	$A_2 \times B_2 + C_2$
5	A_5	B_5	$A_4 \times B_4$	C_4	$A_3 \times B_3 + C_3$
6	A_6	B_6	$A_5 \times B_5$	C_5	$A_4 \times B_4 + C_4$
7	A_7	B_7	$A_6 \times B_6$	C_6	$A_5 \times B_5 + C_5$
8			$A_7 \times B_7$	C_7	$A_6 \times B_6 + C_6$
9					$A_7 \times B_7 + C_7$

31 | Oct | 2018

- Instruction pipelining
- Arithmetic pipelining

Suboperations that are performed in 4 segments are

1. Compare components by subtracting.
2. Align the mantissa
3. Add or Sub the mantissa
4. Normalise the result.

eg.

$X = .9504 \times 10^3 \qquad Y = .8200 \times 10^2$

$3 - 2 = 1$

$\therefore Y = .0820 \times 10^3.$

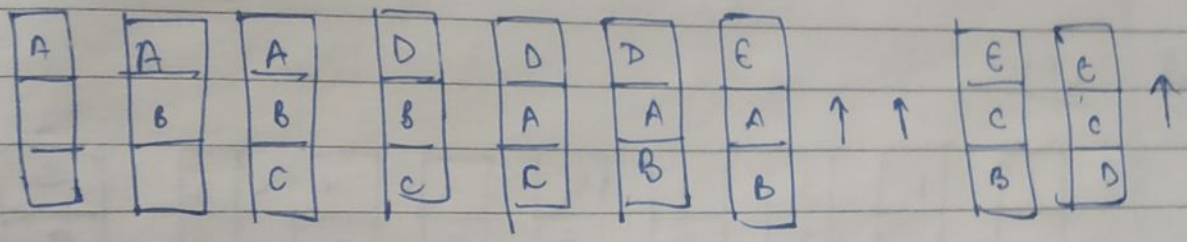
make it non-zero.

$Z = 1.0324 \times 10^3$

$Z = .1034 \times 10^4$

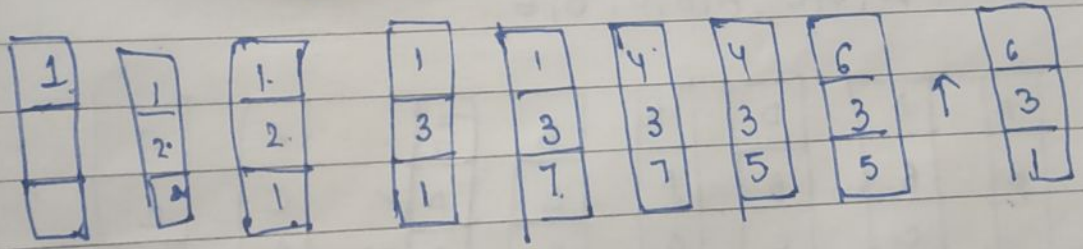
Memory Organisation
Associative Memory - diag, eg. - Intro.
Catche Memory
Virtual Memory
Demand paging - Page fault (How to Handle)
Page Replacement Algo's
Concept of pipelining - II processing, Arith & Inst pipelining

A, B, C, D, A, B, E, A, B, C, D, E FIFO



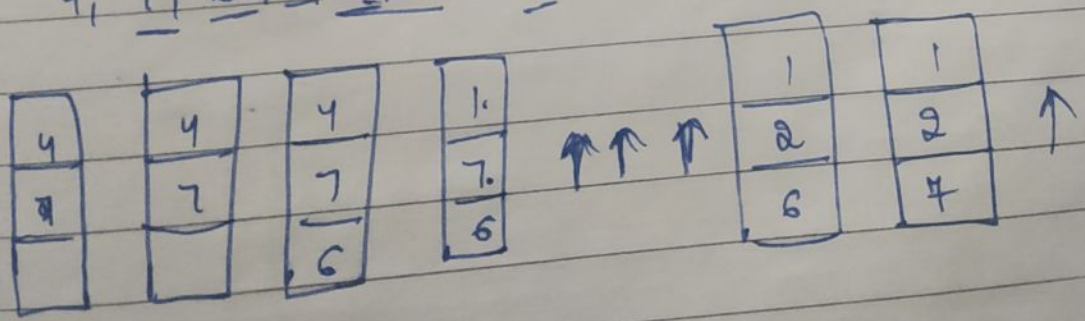
Optimal PR

1, 2, 1, 3, 7, 4, 5, 6, 3, 1



LRU

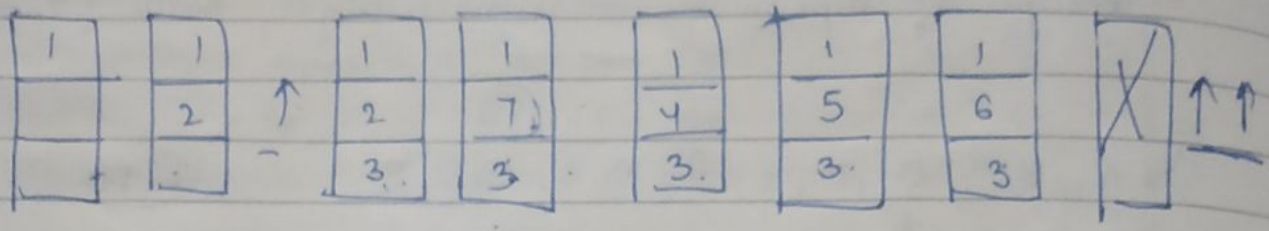
4, 7, 6, 1, 7, 6, 1, 2, 7, 2



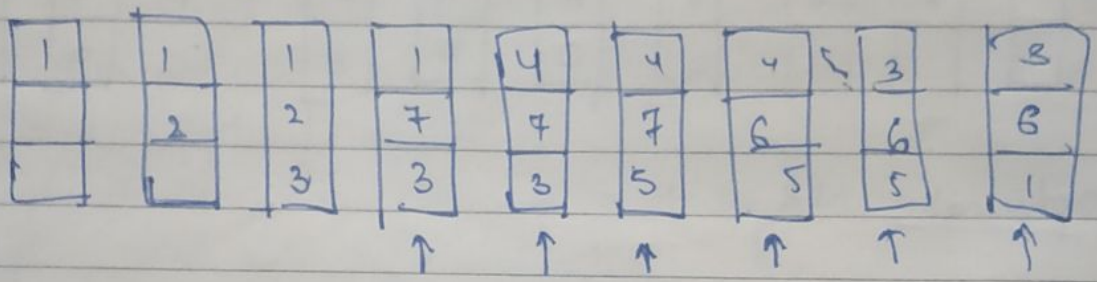
FIFO → easy to implement but pages are removed & loaded from memory very frequently.

Future Optimal page R.

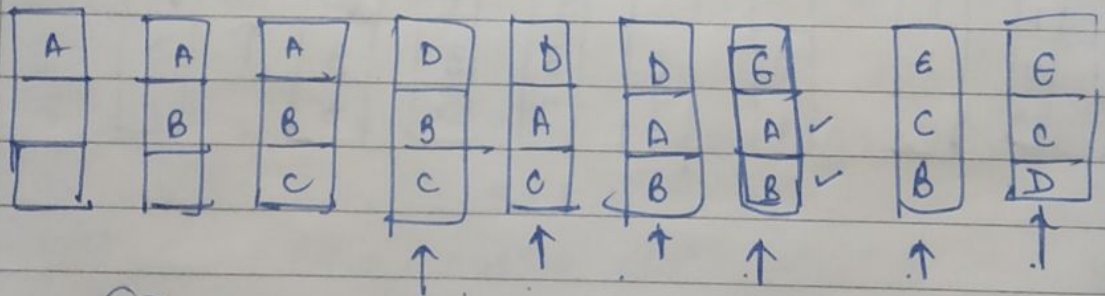
1, 2, 1, 3, 7, 4, 5, 6, 3, 1



Past LRU.



A, B, C, D, A, B, E, A, B, C, D, E



(6)

Date:

SYLLABUS

1. Register Transfer & Microoperations :
Register transfer language & operations, Bus & Memory Transfer, Arithematic Microoperations, logic microoperations, shift microoperations, arithematic logic shift unit.

2. Basic Computer Organisation & Design :
Registers, Buses, Instruction Formats, Instruction Set, Instruction Cycle, Timing & Control, I/O & Interrupt w.r.t. 8085, Instruction codes, Computer Instructions, Control memory, Design of control unit - microprogrammed, hardwired, & their comparitive study.

3. Central Processing unit :
Stack organisation, Addressing Modes, Program control w.r.t. 8085, RISC & CISC architecture, I/O organisations: Peripheral devices, I/O Interface, asynchronous data transfer, modes of transfer, priority interrupt, DMA, Architectures 8237A, I/O processor & serial communication

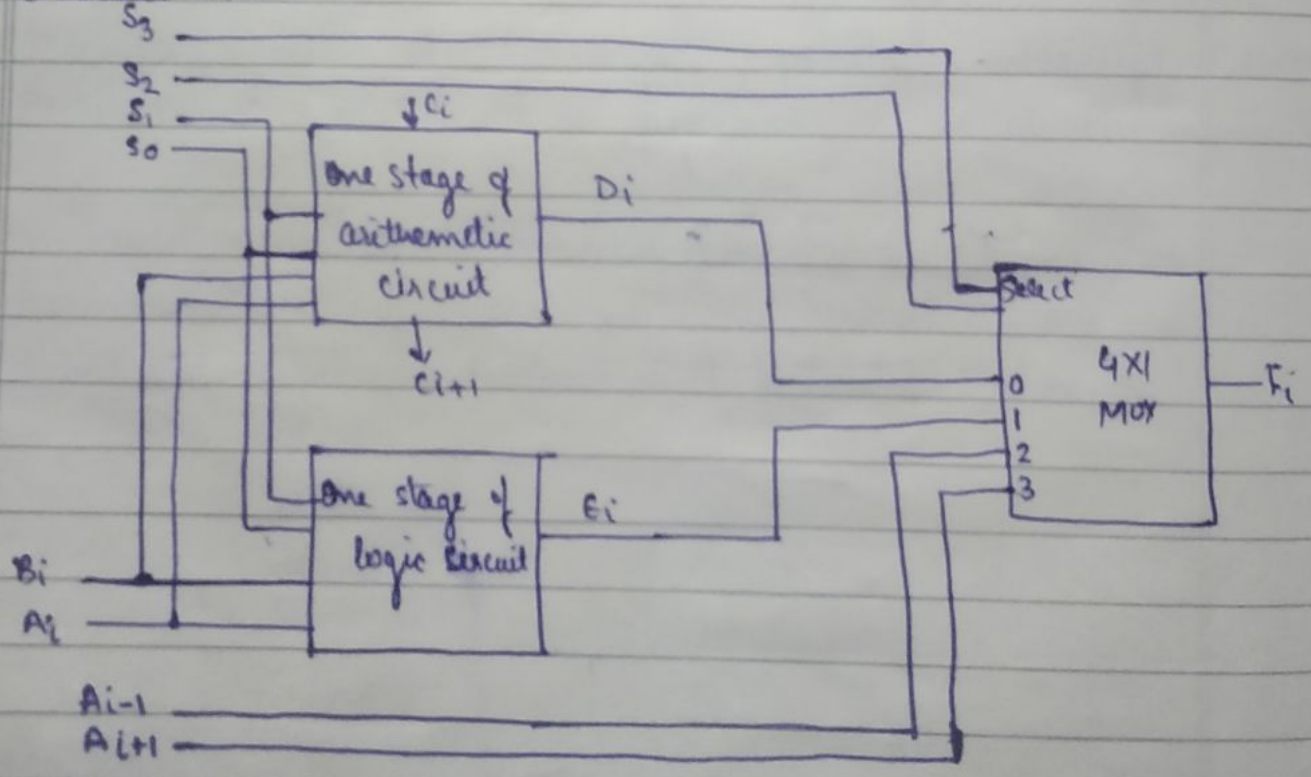
4. Memory Organisation & Advanced concepts of CA.
Associative Memory, Cache memory, Virtual Memory, Concept of pipeline, Arithmetic & Instruction Pipeline.

Page No.
Date:

Arithmetic logic shift unit

Instead of having individual registers performing the microoperation computer system employs a number of storage registers connected to a common operational unit called an arithmetic logic unit (ALU)

To perform a microoperation the contents of specified registers are placed in the inputs of common ALU. The ALU peforms an operation and the result of the operation is then transfered to a destination register.
The ALU is a combinational ~~ree~~ circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be peformed during one clock pulse period. The shift microoperations are often performed in a seperate unit, but sometimes the shift unit is made part of the overall ALU.



One stage of ALU Arithmetic Logic shift unit.
Subscript i designates a typical stage.

Input A_i and B_i are applied to both the arithmetic and logic unit. A particular microoperation is selected with inputs S_1 and S_0.

A 4x1 mux at the o/p chooses b/w an arithmetic output in E_i & a logic o/p in E_i. The data in the MUX are selected with inputs S_0 & S_1. The other two inputs to the MUX receive inputs A_{i-1} for the shift right operation & A_{i+1} for the shift left operation.

The ckt must be repeated n times for n bit ALU. The o/p carry C_{i+1} for a given arith stage must be connected to the i/p carry C_i of the next stage in sequence. The i/p carry to the first stage is the i/p carry C_{in} which provides a selection var. For the ckt, In the given ckt, there are 8 arith. operation selected with $S_3 S_2 = 00$ 4 logic op selected with $S_3 S_2 = 01$ & 2 shift op i.e 10 & 11 other 3 selection i/p's have no effect on shift.

Operation Select				o/p	Fxn
S_3	S_2	S_1	S_0	C_{in}	Operation
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A+1$	Increment A
0	0	0	1	0	$F = A+B$	Addition
0	0	0	1	1	$F = A+B+1$	Add with carry
0	0	1	0	0	$F = A+\bar{B}$	Subtraction with borrow
0	0	1	0	1	$F = A+\bar{B}+1$	Subtraction
0	0	1	1	0	$F = A-1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	x	$F = A \wedge B$	AND
0	1	0	1	x	$F = A \vee B$	OR
0	1	1	0	x	$F = A \oplus B$	EXOR
0	1	1	1	x	$F = \bar{A}$	Complement A
1	0	x	x	x	F = shr A	shift right A into
1	1	x	x	x	F = shl A	Shift left A into

Page No.
Date:

Instruction Cycle - Prog consist of a sequence of instruction. The prog is executed by going through a cycle for each inst. Each inst cycle is subdivided into a sequence of subcycles or phases :-

1. Fetch the inst from the memory
2. Decode the inst.
3. Read the effective add from memory if the inst has an indirect address.
4. Execute the inst

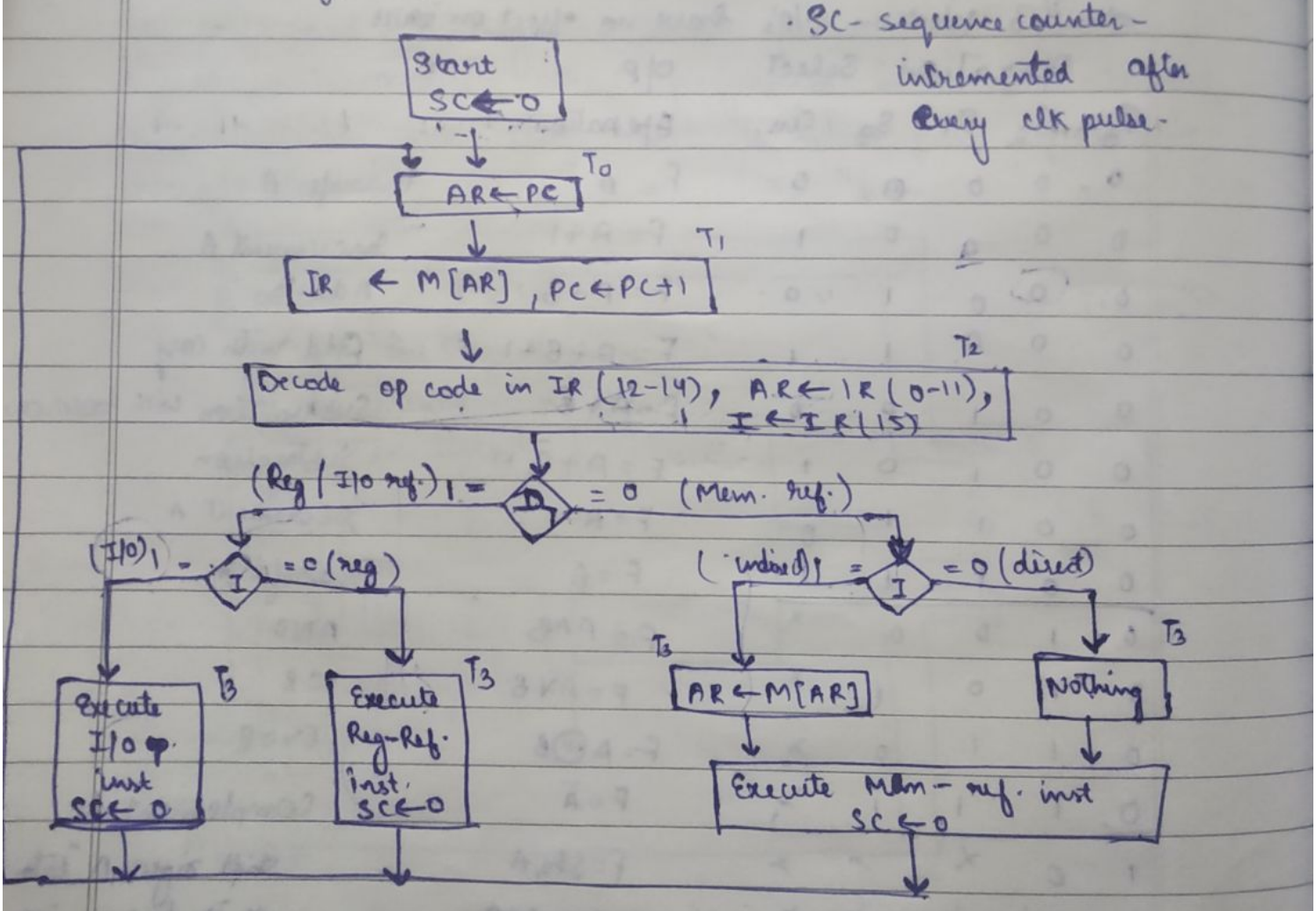
Fetch and Decode

T_0 : $AR \leftarrow PC$

T_1 :- $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$

T_2 :- $D_0 \dots D_7 \leftarrow$ Decode $IR(12-14)$,
$AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$.

Determine the type of inst



- SC - sequence counter - incremented after every clk pulse.

Date:

Various Addressing Modes

The way operands are chosen during prog. execution OR. rule for interpreting or modifying the address field of the inst before the operand is actually referenced.

(1) Implied Mode :-

In this operands are specified implicitly in the definition of inst. All the register reference inst that use an accumulator are Implied Mode inst. Zero inst in a stack organised comp are implied mode inst. Since the operands are implied to be on top of the stack.

eg. "complement accumulator" CMA, HLT,

i.e only opcode is present, no need to specify data or address seperately (all these inst of are of 1 Byte)

(2) Immediate Mode :-

In this the operand specified in the instruction itself. It has an operand field rather than addres field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction

eg:- LXI 2501H, MVI B32H

3 Byte inst 2 Byte inst

(3) Register Mode. :- direct / Indirect

Page No.
Date

Direct Address Mode

effective add = address part of inst

Indirect :- eff add = add. Part of inst → (content of CPU eg.

Relative :- eff add = add. (inst) + Reg. counter (PC)

Indexed. :- eff add = add (inst) + Index Reg (XR)

Base. :- eff add = add (inst) + Base Reg

Eg. PC = 200 R1 = 400 XR = 100

AC

direct EA :- 500 operand 500

indirect :- 200 + 500 (+PC) = 702 325

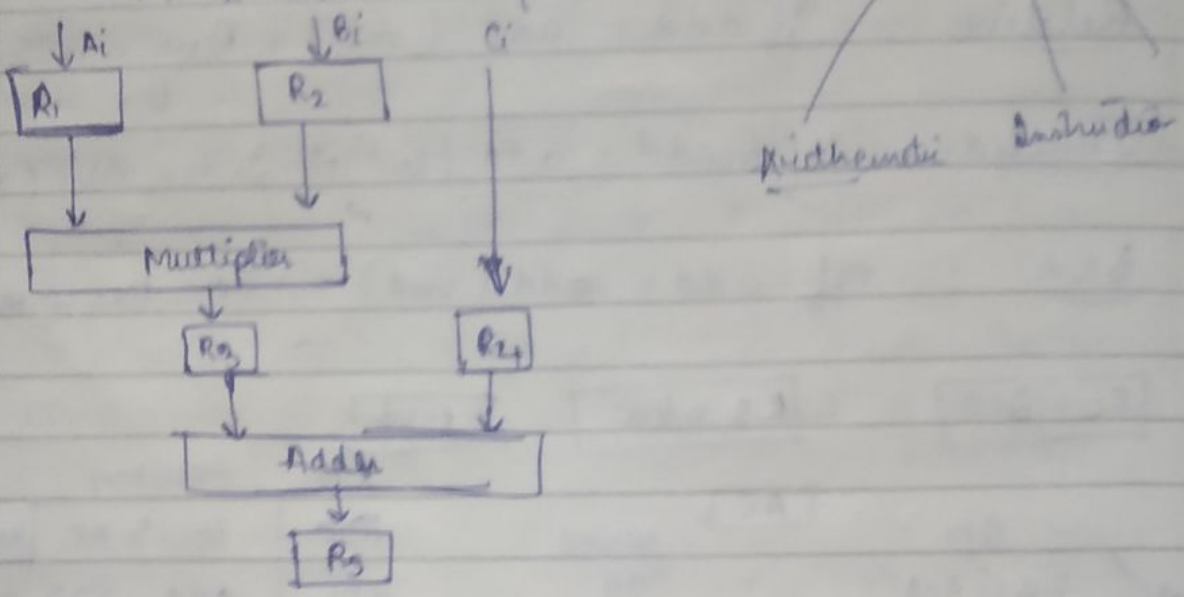
Relative :- 202 + 500

	E. A	Content of AC
direct	500	800
immediate	201	500.
Indirect	800	300 .
Relative	500 + 202 = 702	325
Indexed	500 + 100 = 600	900
Reg.	—	400
Reg. ind.	400	700
Auto inc.	400	700
Auto dec.	399	450

Add.	Memory	
200	Load to AC	Mode
201	Add = 500	→ add of (inst)
PC 202	Next inst	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.

Eg $A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots$



Clock Pulse No.	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 \times B_1$	C_1	~~$A_1 \times B_1 + C_1$~~
3	A_3	B_3	$A_2 \times B_2$	C_2	$A_1 \times B_1 + C_1$
4	A_4	B_4	$A_3 \times B_3$	C_3	$A_2 \times B_2 + C_2$
5	A_5	B_5	$A_4 \times B_4$	C_4	$A_3 \times B_3 + C_3$
6	A_6	B_6	$A_5 \times B_5$	C_5	$A_4 \times B_4 + C_4$
7	A_7	B_7	$A_6 \times B_6$	C_6	$A_5 \times B_5 + C_5$
8	—	—	$A_7 \times B_7$	C_7	$A_6 \times B_6 + C_6$
9	—	—	—	—	$A_7 \times B_7 + C_7$

~~Ans~~

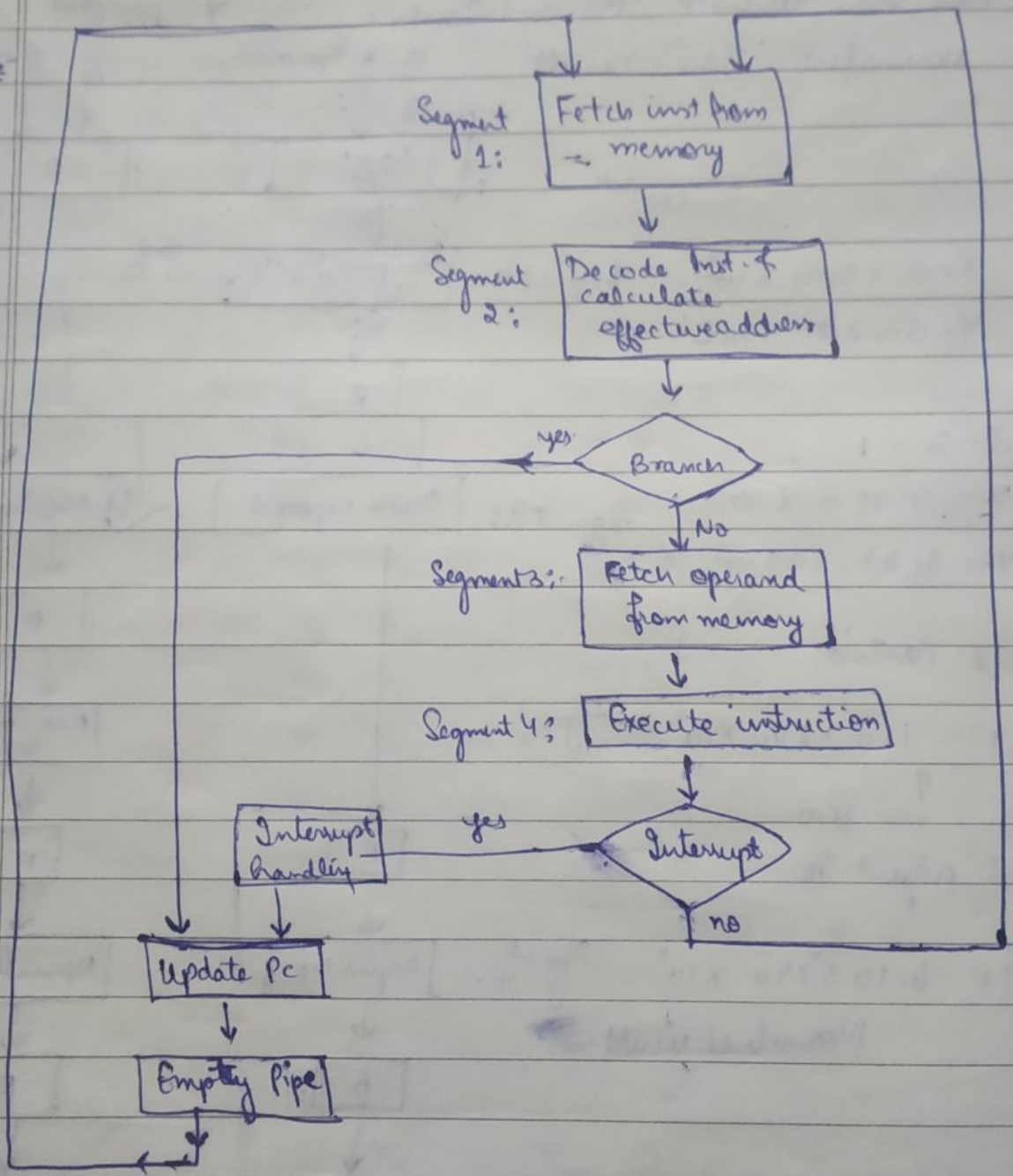
Date:

Instruction Pipeline : operates on a stream of instructions by overlapping the fetch, decode and execute phases of the instruction cycle.

Each inst is processed with the following sequence of steps :-

1. Fetch the inst from memory
2. Decode the inst
3. Calculate the effective address
4. Fetch the operand from memory
5. Execute the instruction.
6. Store the result in proper place.

eg:



1. FI is the segm that fetches the inst
2. DA is the seg. that decodes the inst & calculates eff add.
3. Fo is the segment that fetches the operand.
4. EX is the segment that executes the instruction.

Timing :-

step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Inst: 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
3			FI	DA	FO	EX							
4				FI	-	-	FI	DA	FO	EX			
5					-	-	-	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

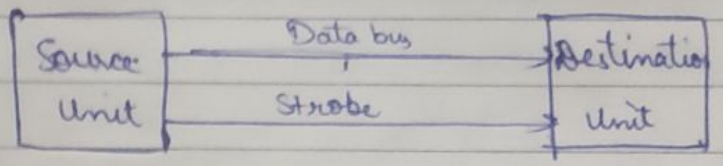
Page No.
Date:

Pipeline conflicts – that deviate the inst. pipeline from its normal operation

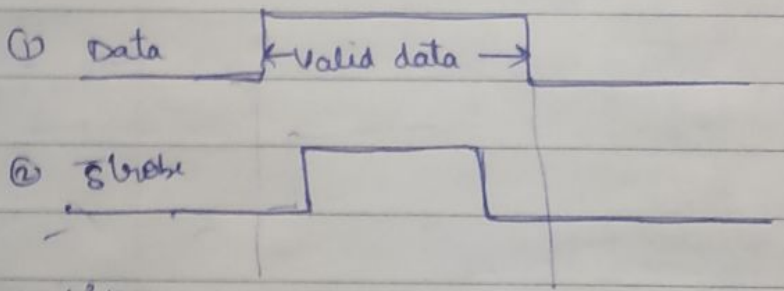
1. Resource 2. Data dependency, 3. Branch difficulties.

1. Resource conflicts caused by access to memory by two segments at the same time. Most of that conflicts can be resolved by using seperate instructions & data memories.
2. Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but, this result is not yet available.
3. Branch difficulties arise from branch and other instructions that change the value of PC.

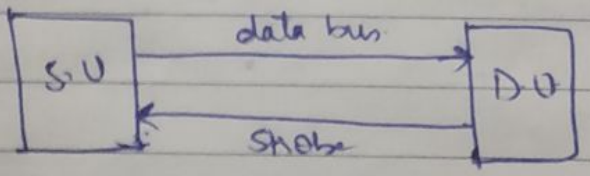
Strobe

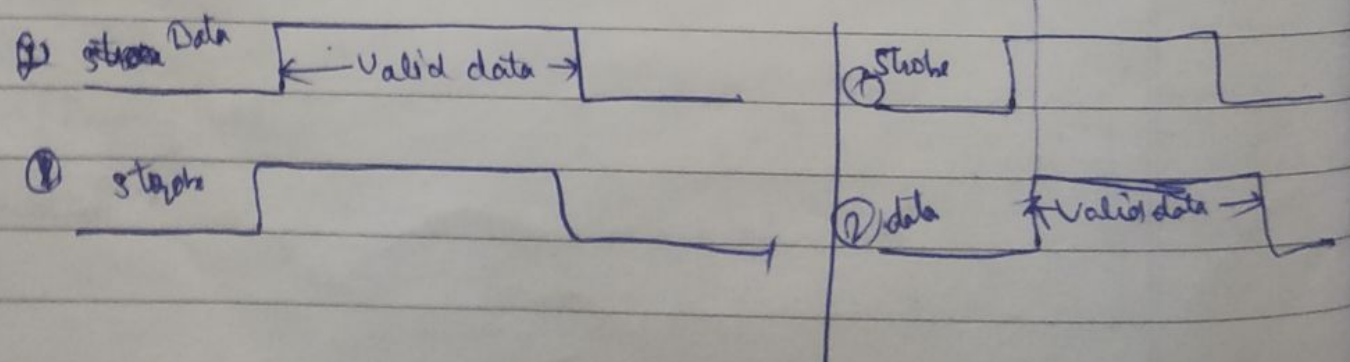


Source-initiated

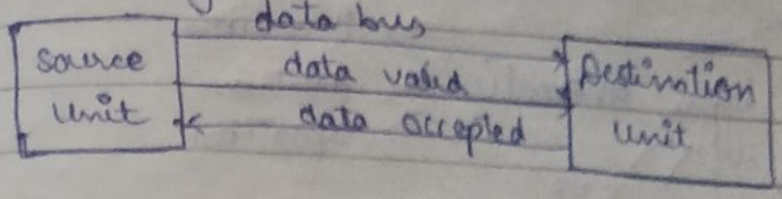


destination initiated

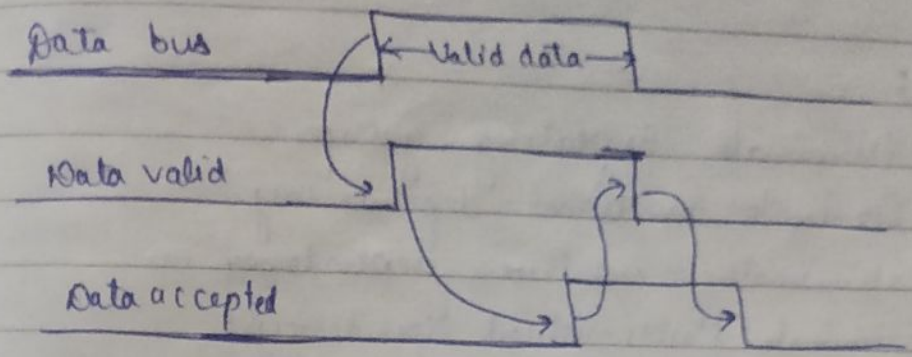




Handshaking



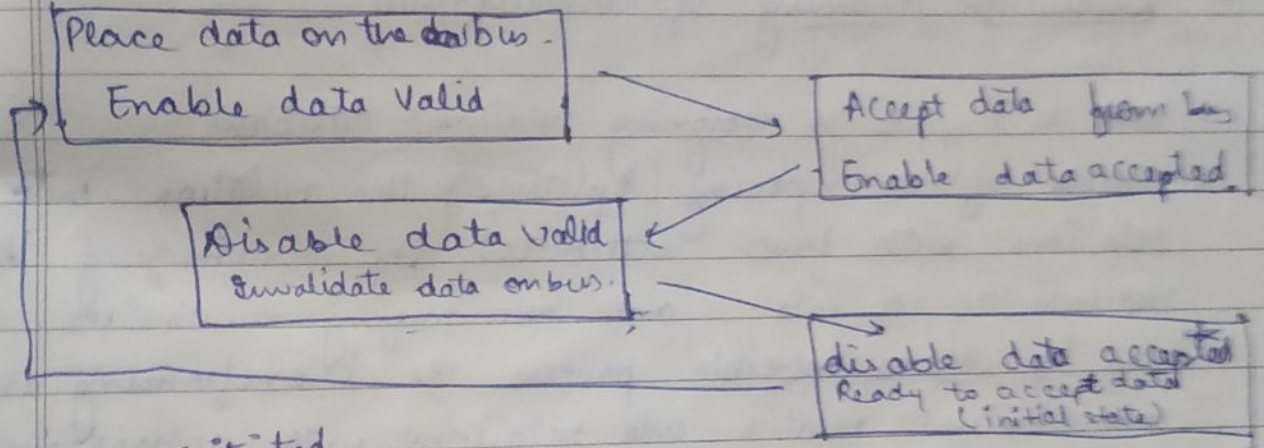
Source initiated



Timing dia

Source unit

Destination unit



Destination initiated