Page No:
Date: / /

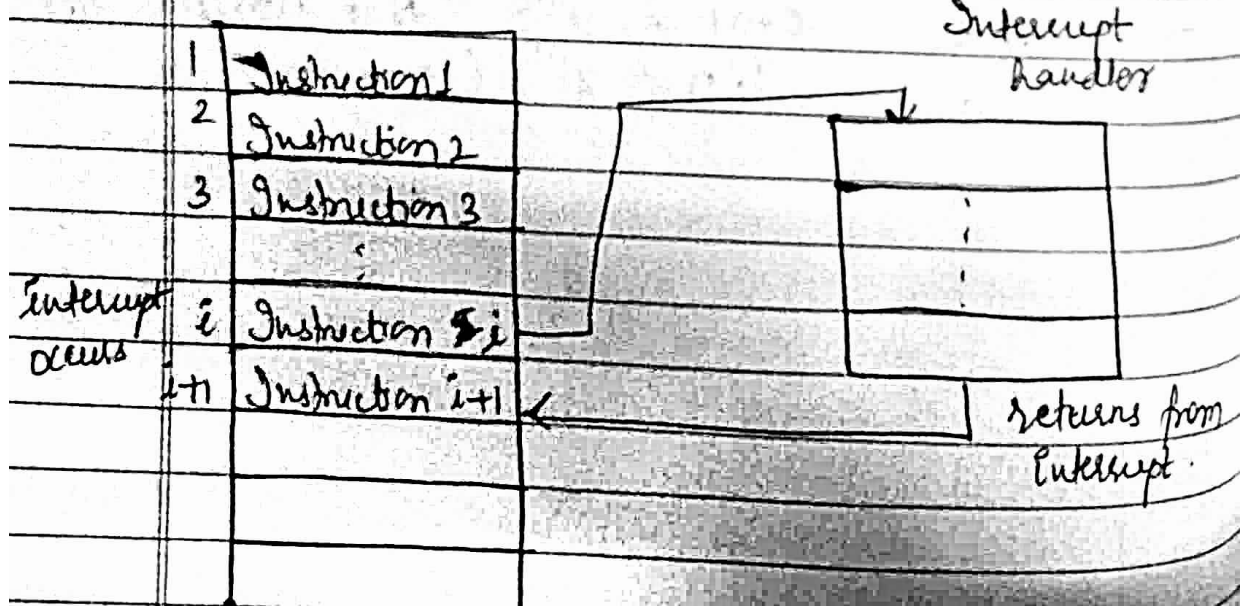
Interrupt and Its Types

Program Interrupt :- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.

* Control returns to the original program after the service program is executed.
* When an interrupt is initiated, the state of CPU containing following information is saved.

1) Content of program counter (PC)
2) Content of all CPU register
3) Content of Status bit conditions (PSW)

* CPU does not respond to the interrupt until the execution of currently running instruction ends.



Date: / /

Transfer of Program Control via Interrupt

* Before going to next fetch cycle, It checks for an interrupt signal, If there is an interrupt.

(1) Save CPU state (PC, CPU register, PSW), in memory stack
(2) PC ← interrupt branch address
PSW ← Status bits of interrupt service program.
(3) Last ← instruction of service program is return from interrupt.
(4) PSW ← old PSW, PC ← old PC, CPU registers ← old CPU registers

Types of program Interrupts.

1) External Interrupts :- External interrupts are generated by input output devices, by a timing device, from a circuit monitoring the power supply or from any external source. They are "asynchronous".

Eg:- (i) I/O device requesting transfer of data.
(ii) I/O device finished transfer of data.
(iii) Timeout interrupt is generated from program that goes into endless loop.

* *

(iv) power failure circuit also causes interrupt

(2) Internal Interrupt :- Internal interrupts are generated from illegal or erroneous use of an instruction also called "Traps"

Eg:- (i) Due to register overflow error
(ii) Divide by zero error
(iii) Invalid operation code (opcode).
(iv) Stack overflow.

Internal interrupts are "synchronous" with the program.

"Both External and internal interrupts are initiated from signal that occur in the hardware of the CPU".

(3) Software Interrupt :- Software interrupt is a special case instruction that behaves like an interrupt rather than a subroutine call.

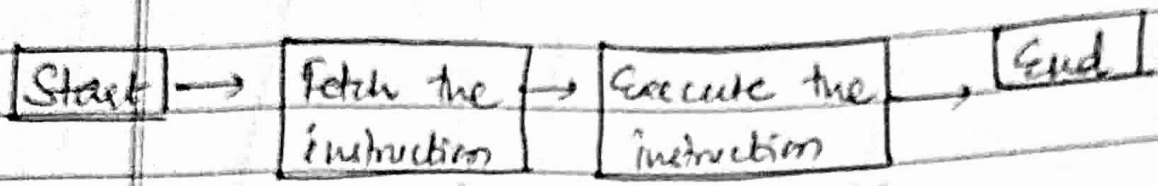
* It is initiated by executing an instruction.
* It is used by the programmer to initiate an interrupt procedure at any desired point in the program.

Eg:- Supervisor call instruction generates a software interrupt to switch

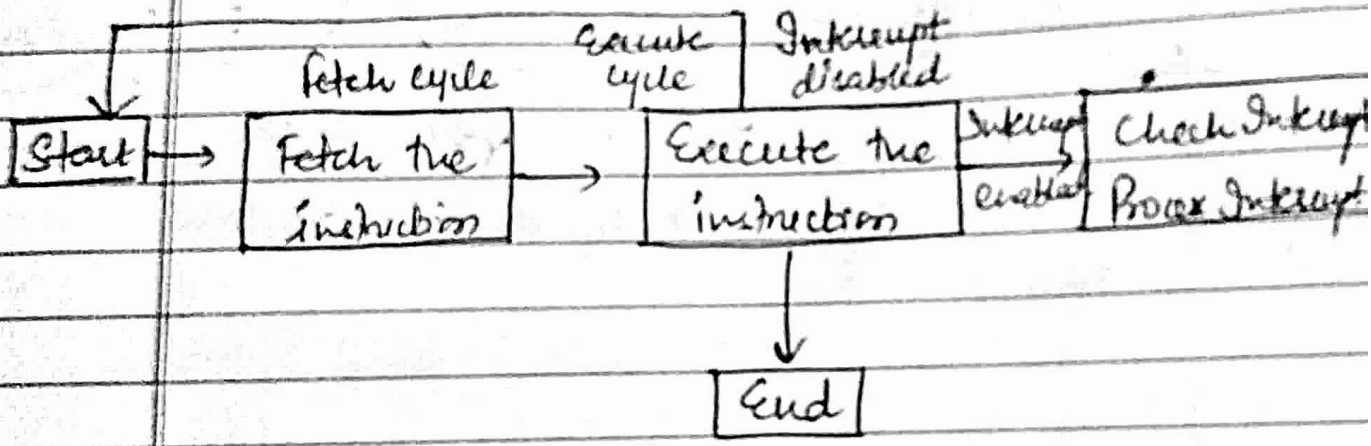
from user mode to supervisor mode.

Instruction Cycle

- An instruction cycle specify the various steps required during the processing of an instruction.



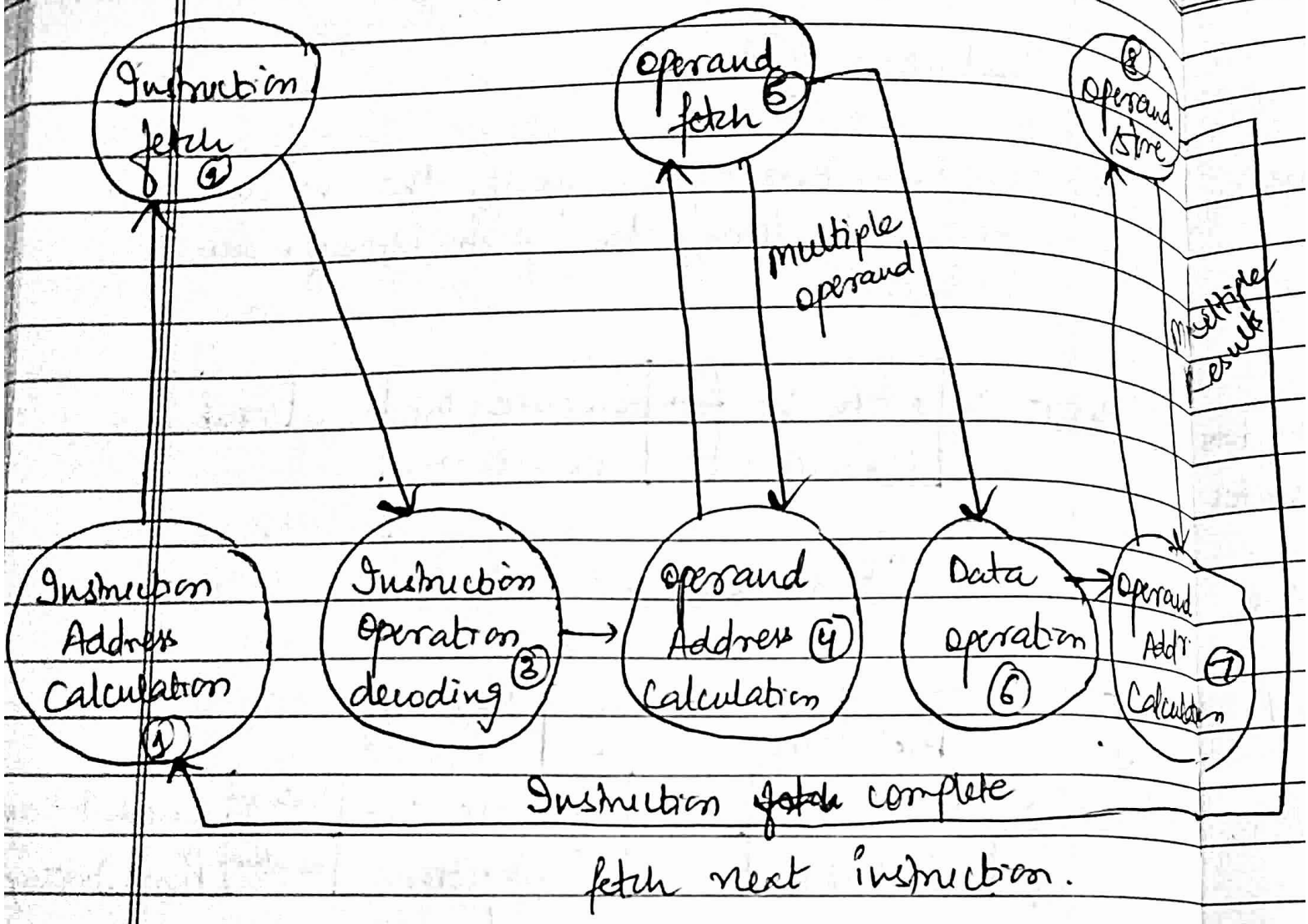
(Basic Instruction cycle)

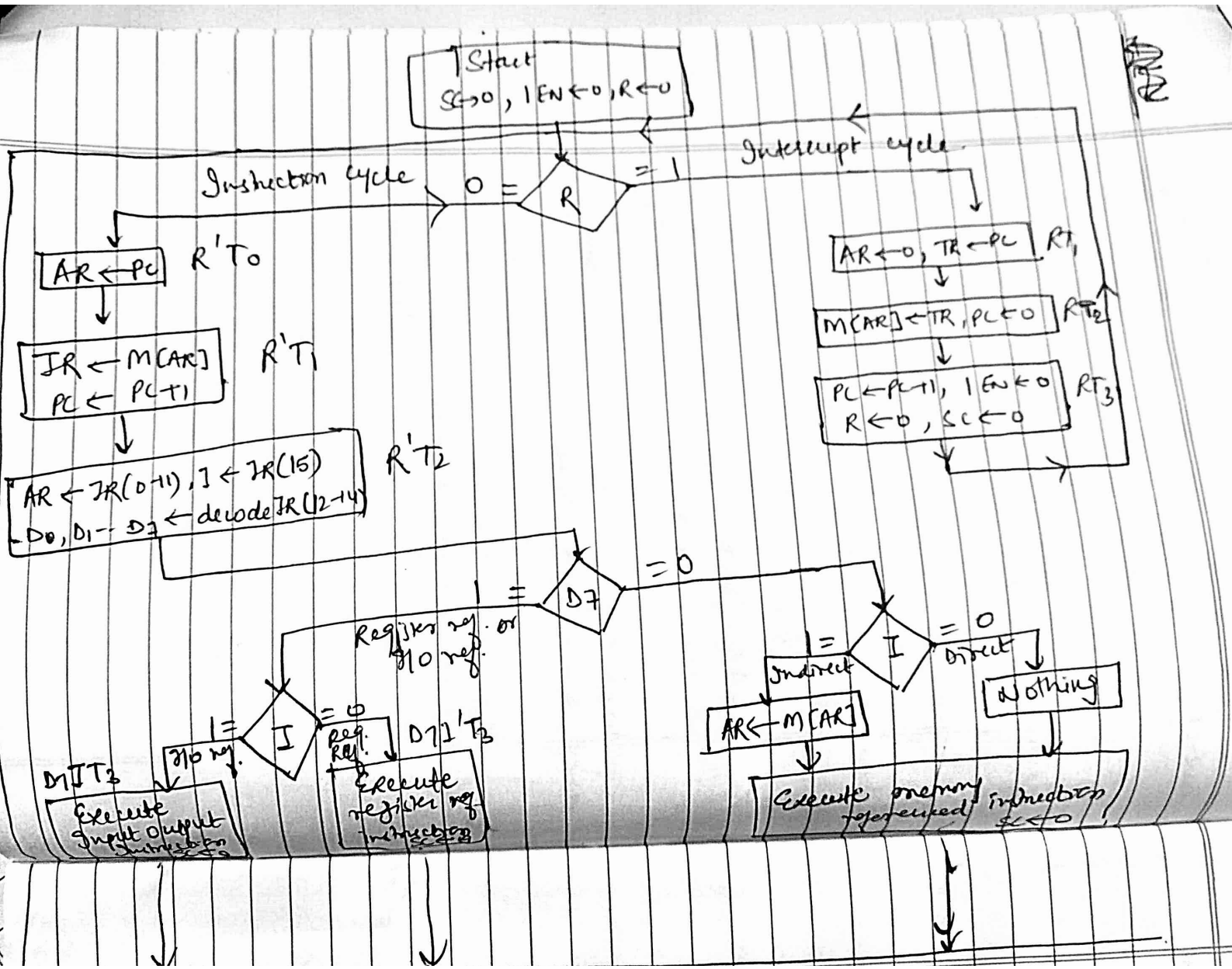


(Basic Instruction cycle with Interrupt)

Page No.
Date: / /

Instruction cycle State diagram.




Start
SC←0, IEN←0, R←0
R
0
1
Instruction cycle
Interrupt cycle
AR←PC
R'T0
IR←M[AR]
PC←PC+1
R'T1
AR←0, TR←PC
RT1
M[AR]←TR, PC←0
RT2
PC←PC+1, IEN←0
R←0, SC←0
RT3
AR←IR(0-11), I←IR(15)
D0, D1 ... D7←decode IR(12-14)
R'T2
D7
1
0
Register ref. or I/O ref.
I
1
0
I/O ref.
Reg. Ref.
D7I'T3
D7IT3
Execute input output instruction
Execute register ref. instruction
I
1
0
Indirect
Direct
AR←M[AR]
Nothing
Execute memory reference instruction
SC←0

AR = Address Register

IR = Instruction Register.

SC = Sequence Counter.

R = 1 bit flip flop.

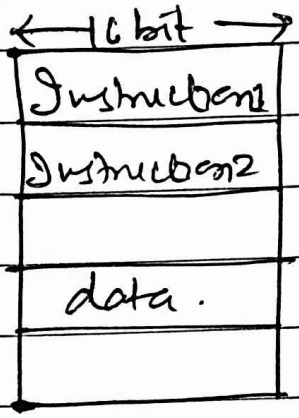
IEN = Interrupt Enable.

PC = Program Counter

I = 1 bit flip flop.

TR = Temporary Register.

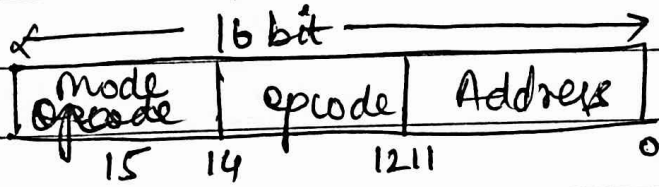
$T_0, T_1, T_2, T_3 \ldots$ = Timing signal.

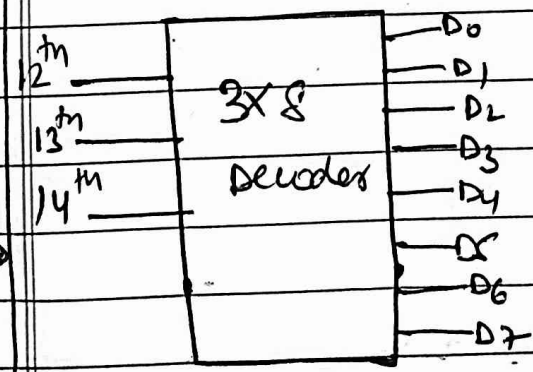


AR ← 0 IR ← M[0]

PC ← 1

Main Memory





000 to 110 → D0 to D6 = 1

111 → D7 = 1

Date
Page No.

Save Pc at memory 0 & Interrupt address at memory 1

	Save Pc
1	Interrupt Address

Page No:
Date: / /

RISC V/S CISC

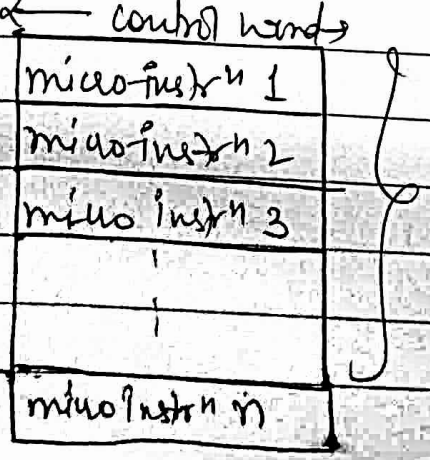
	RISC (Reduced Instruction Set Computer	CISC (Complex Instruction set compute
1.	Simple Instructions	1. Complex Instructions
2.	Less no. of instructions	2. Large no of instructions
3.	Hardwired Control Unit	3 Microprogrammed control unit
4.	≃ 1 cycle/ instruction	4. 2 or more cycle/ instruction.
5.	Fixed Instruction format.	5. Variable instruction format.
6.	High pipelined.	6. Sequential / less pipelined.
7.	Fewer addressing modes.	7. More addressing modes.
8.	only load & store instructions can access memory.	8. Many instructions can access memory.
9.	Uses more registers	9. Uses less registers.
10.	Used in mobile phones and tablets.	10. Used in desktop & laptops.

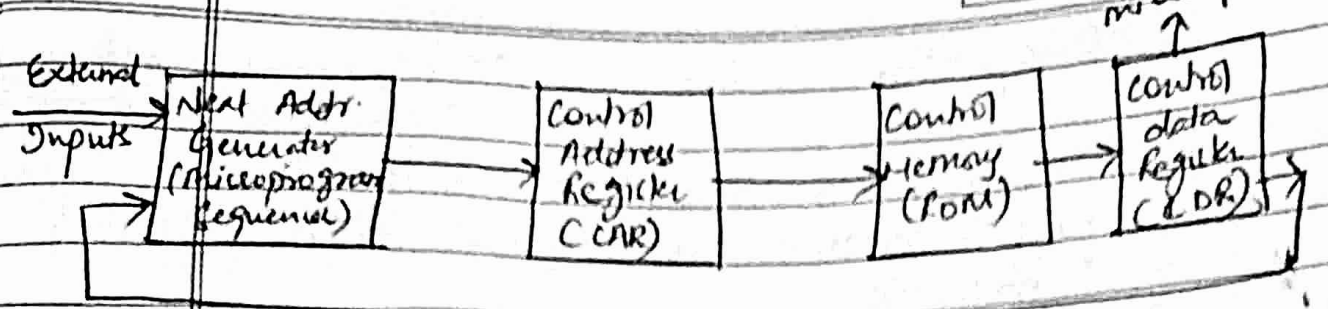
Page No:
Date: / /

Microprogrammed Control Unit

* A microprogrammed control unit is a control unit in which control signals are stored in binary form (0s & 1s) as control word in a ROM chip called control memory.

* Each control word consist of microinstructions
* Each microinstruction when executed, generate sequence of micro-operations to fetch instruction from memory, calculate effective address, fetch operand etc.
* A set of ~~instructions~~ microinstructions is called a microprogram.
* A set of microinstructions belonging to a machine instruction is called a micro routine.
* A microinstruction is a control word stored in control memory (ROM chip).
* A control memory is a ROM chip which stores microinstructions as control word in it.
* Microprogramming is the art of writing microprogram (micro-code) for the control unit in a CPU.





General Configuration of Microprogrammed Control Unit

1. **Microprogram Sequencer** :- It is also called next address generator as it determines the address of next microinstruction that is read from control memory.

2. **Control Address Register** :- It holds the microinstruction in the control memory. It is represented by CAR.

3. **Control Memory** :- The control memory is actually a ROM chip where all the microinstructions are stored.

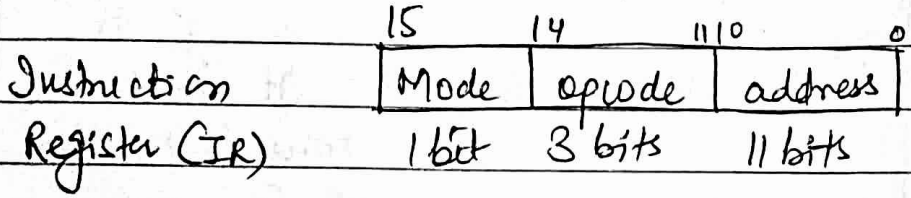
4. **Control data Register** :- It stores the microinstruction after reading from control memory. The microinstruction when executed, perform a set of micro-operations along with providing the next address.

Page No:
Date: / /

Microprogram Sequencer :- It is called next address generator. Can generate the next address of the microinstruction in control memory using 4 methods.

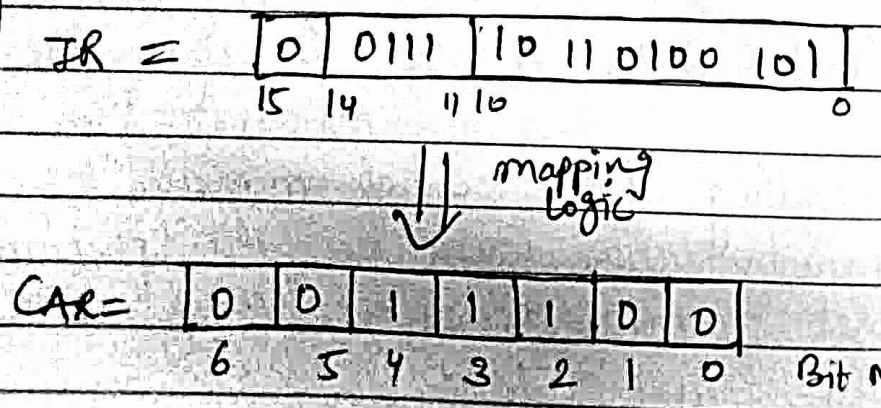
(i) Increment the control address Register.
(ii) Unconditional / conditional branch depending on ~~status~~ status bit flags.
(iii) Subroutine call & return.
(iv) Mapping from machine instruction to microinstruction address.

Mapping from machine instruction to micro-instruction address



Mapping Logic
CAR (0, 1, 6) ← 0
CAR (2-5) ← IR (11-14)

Let CAR = 7 bits
IR = 16 bits.



Page No:
Date: / /

Micro-instruction format

f_1	f_2	f_3	CD	BR	AD
3bits	3bits	3 bits	2bit	2 bit	7bits

f_1, f_2, f_3 → micro-operation fields.
CD → Condition for branching.
BR → Branch field.
AD → Address field

f_1 (3 bits) → specify a total of 8 micro-operations (including No-operation)
f_2 (3 bits) → specify a total of 8 micro-operation (including No-operation)
f_3 (3 bits) → specify a total of 8 micro-operations (including No-operation)

Total micro-operations = 24 (21+3 (No-op))

CD (Condition field)

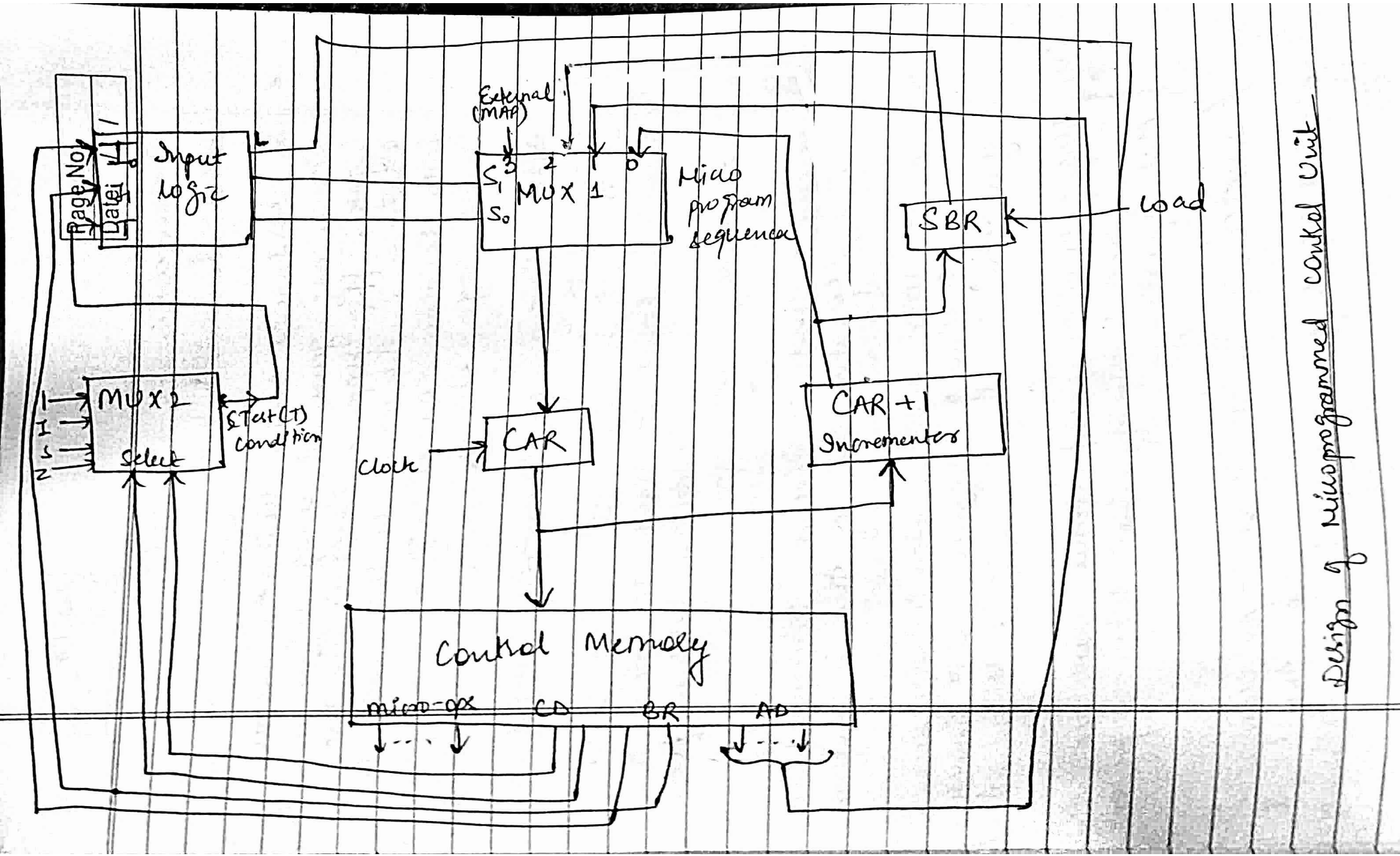
CD	Condition	Symbol	Comment
0 0	Always = 1	U	Unconditional Branch
0 1	JR (15)	I	Indirect Addr bit
1 0	Ac (15)	S	Sign bit of Ac
1 1	Ac = 0	Z	Zero value in Ac

Page No:
Date: / /

BR (Branch field)

BR	Symbol	Function
0 0	JMP	CAR ← AD, if condition = 1 CAR ← CAR+1, if condition = 0
0 1	CALL	CAR ← AD, SBR ← CAR+1, if condition = 1 CAR ← CAR+1, if condition = 0
1 0	RET	CAR ← SBR (Return from subroutine)
1 1	MAP	CAR(2,5) ← IR(11-14), CAR(0,1,6) = 0

B	R	T	S_1	S_0	L
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	x	1	0	0
1	1	x	1	1	0


Design of Microprogrammed control Unit
Page No.
Date
Input logic
External (MAP)
MUX
Micro program sequencer
SBR
Load
MUX 2
Select
Status (T) condition
Clock
CAR
CAR + 1 Incrementer
Control Memory
micro-op
CD
BR
AD

Page No.
Date: / /

~~Status bonds~~

Hardwired Control Unit

* In hardwired control unit, the control signals are generated using logic circuits (gates, flip-flops & decoders) in the hardware. Hardware control unit is faster because control signals are generated using logic circuit.

Block diagram of Hardwired Control Unit consists of :-

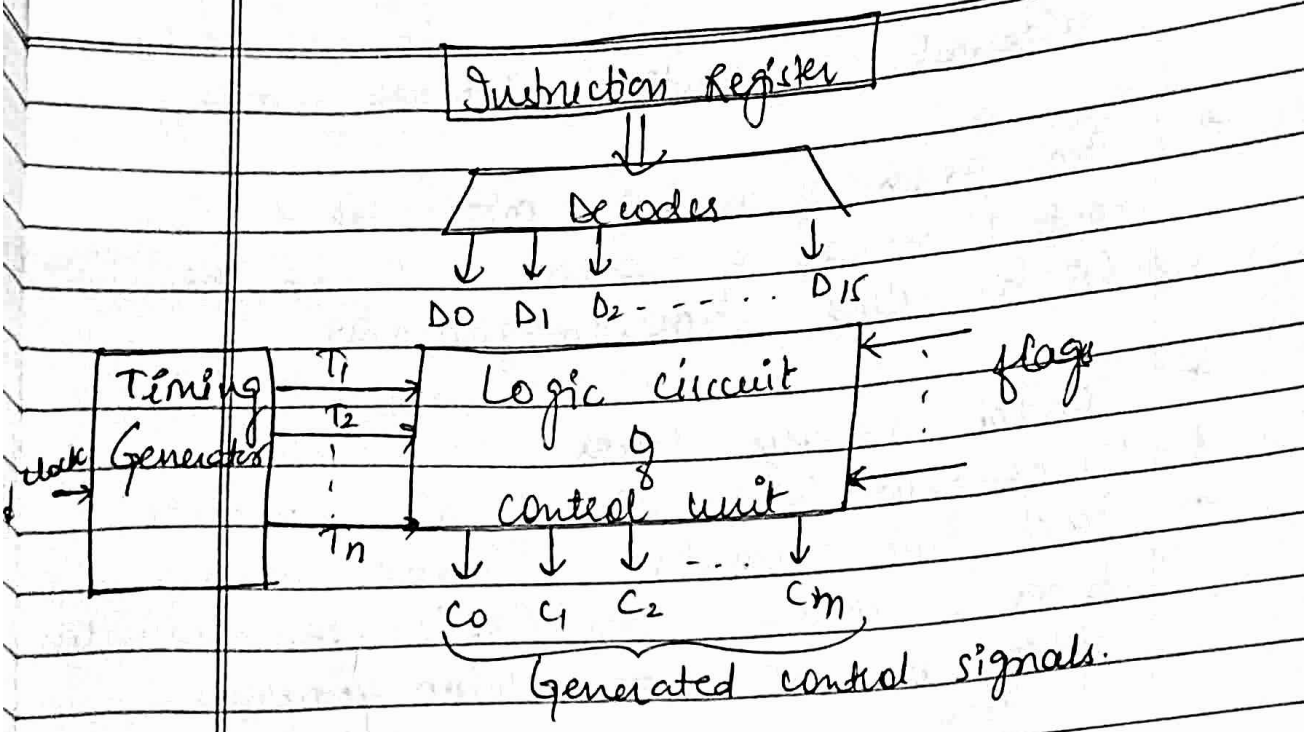
1) Instruction Register :- Used to store the instruction fetched from the memory.

2) Decoder :- Used to decode (interpret) the operation code of the instruction.

3) Clock :- Used to generate a sequence of timing signals using timing generations.

4) Flag :- Condition codes/ status flags are used to specify the status of previous ALU operation.

Generation of control signals :- An example of how the control signals are generated is as shown :-



Consider C_5 : MBR ← Memory

Cycle	Micro-operations	Control signals
Fetch	t_1:- MAR ← PC	C_2
PQ=00	t_2 :- MBR ← memory	C_5
Indirect	t_1: MAR ← IR (address)	C_8
PQ=01	t_2: MBR ← memory	C_5
Interrupt	t_1: MBR ← PC	C_4
cycle	t_2: MBR ← memory	C_5
PQ=10		

execute cycle PQ=11

$C_5 = P'Q' t_2 + P'Q t_2 + PQ' t_3$

Let only LDA, ADD, AND instruction read from memory

$$C_5 = \bar{P}\bar{Q}t_2 + \bar{P}Qt_2 + P\bar{Q}t_3 + PQ(LDA + ADD + AND)t_2$$

Page No:
Date: / /

Horizontal vs Vertical Microprogramming.

* The process of writing micro code for the control memory of control unit in the CPU is called microprogramming.

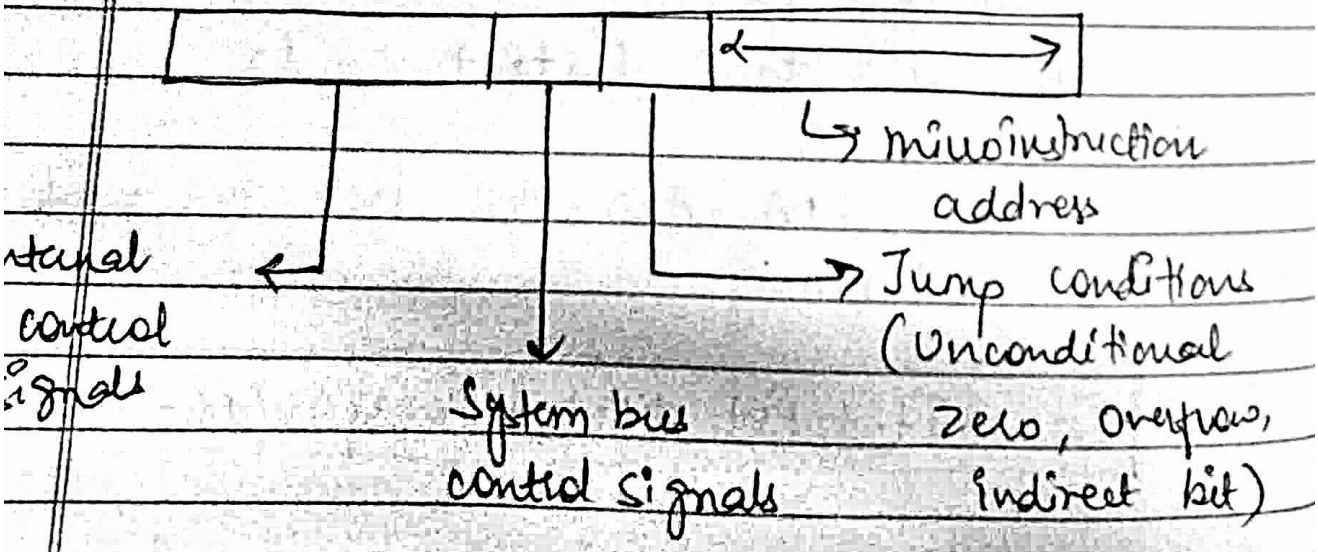
* Control memory stores microprogram.
* Microprogram is a set of microinstruction
* Each line (location) of control memory stores a microinstruction (μI). Microinstruction specify one or more micro operations.

Organisation of Microinstructions.

(1) horizontal microprogramming microinstruction.
(2) vertical microinstruction.

Horizontal Microinstruction.

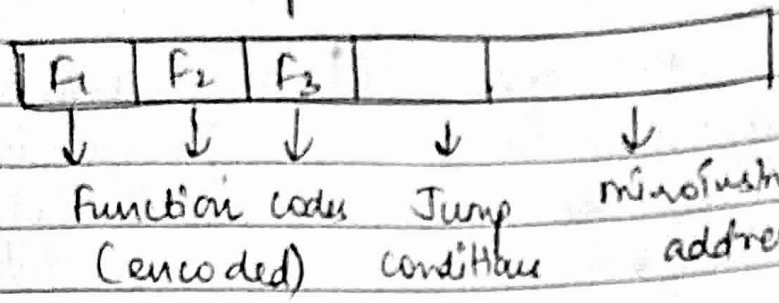
* Each horizontal microinstruction uses 1 bit/ control signal (decoded).
(8 bits for 8 control signals).
(Horizontal implies long microinstruction



Page No:
Date: / /

Vertical Microinstruction :- Each vertical microinstruction represents control signals in encoded form.
ie (3 bits for 8 control signals)
If n is the no. of control signals then $(\log_2 n)$ bits are required.

(vertical implies that microinstruction)



	Horizontal Microprogramming	Vertical Microprogramming
1.	Long microinstruction	1. Short microinstruction
2.	No encoding of control signals	2. Encoding of control signals
3	High degree of parallelism	3. Low degree of parallelism.
4.	No additional hardware	4. Additional hardware required.
5	Faster (fast execution)	5. Slower (slow execution)
6.	Less use of ROM (control memory)	6. More use of ROM (control memory).
7.	optimise performance	7. optimize programming.
8	Little or no control logic	8. Complex control logic

Page No:
Date: / /

Pipelining :- Pipelining is a technique of decomposing a sequential process into a no. of subprocesses with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.

Features of pipelining :-

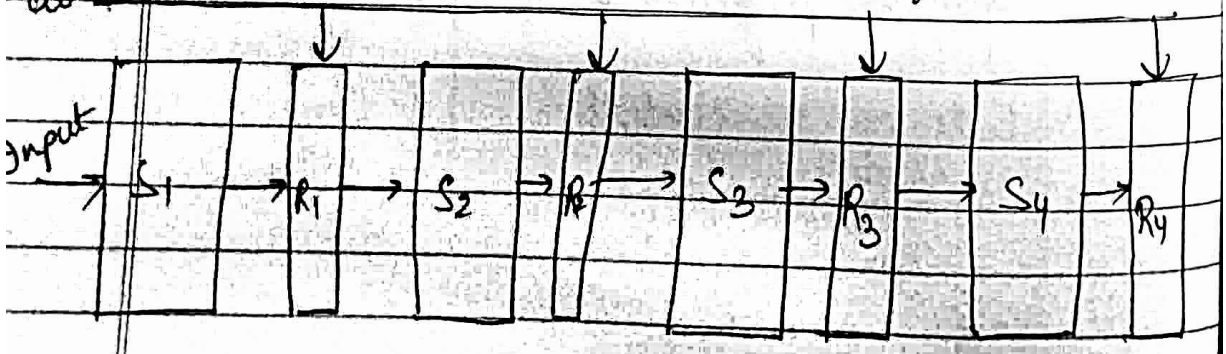
1) The partial result obtained from the computation in each segment is transferred to the next segment in the pipeline and so on.

2) The final result is obtained after the data have passed through all segments.

3) Several computation can be performed in different segments at the same time.

General structure of 4- segment pipeline.

* A 4 segment pipeline consists of 4-segments.
* The segments are separated by registers.
* Registers are used to ~~store~~ hold intermediate results. while Segments consists of combinational circuits.



Page No.
Date: / /

Segments = S_1, S_2, S_3, S_4
Registers = R_1, R_2, R_3, R_4

Space-time diagram :- is a diagram that shows the segment utilization as function of time.

Let no. of segments = 4
No. of tasks = 6

seg. No. \ clock	1	2	3	4	5	6	7	8	9
S1	T_1	T_2	T_3	T_4	T_5	T_6			
S2		T_1	T_2	T_3	T_4	T_5	T_6		
S3			T_1	T_2	T_3	T_4	T_5	T_6	
S4				T_1	T_2	T_3	T_4	T_5	T_6

Speed up ratio = $\frac{4*6}{4+5} = \frac{24}{9} = 2.67$

Instruction pipeline

* An instruction pipeline reads consecutive instructions from memory while previous instruction are executed in various segments of pipeline

Example :- 4-segment instruction pipeline.

The process of each instruction is divided into 4-segments.

Date: / /

1) FI is the segment that fetches an instruction.
2) DA is the segment that decode instruction and calculate effective address.
3) Fo is the segment that fetches the operand.
4) Ex is the segment that executes the instruction.

I \ T	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	Fo	Ex									
2		FI	DA	Fo	Ex								
3 (Branch)			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

Timing diagram at Instruction pipeline

(R)

Pipeline Hazards (Data Dependency)

* Data dependency occurs when an instruction depends on the result of the previous instruction, but this result is not yet available.

Let instruction i : $x = y + 2$
instruction i+1 : $x_1 = x + 1$

I \ Clk	1	2	3	4	5	6
Instr^n i	FI	DA	FO	EX		
Instr^n i+1		FI	DA	FO	EX	

Solutions (Hardware & Software Method)

- Hardware Interlocks
- operand forwarding

(i) Hardware Interlocks :- Hardware interlock is a circuit that detects instructions where source operands are destinations of instruction farther up in the system. Detection of such situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict.

(ii) Operand forwarding :- They use special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.

(iii) Software Method :- In the software method, the compiler is designed to detect a data conflict and reorder instructions as necessary to delay the loading of the conflicting data by inserting "No-operation" instruction. This

method is called "Delay Load".

Instruction Pipeline conflicts:- Pipeline conflicts causes instruction pipeline to deviate from its normal operation.

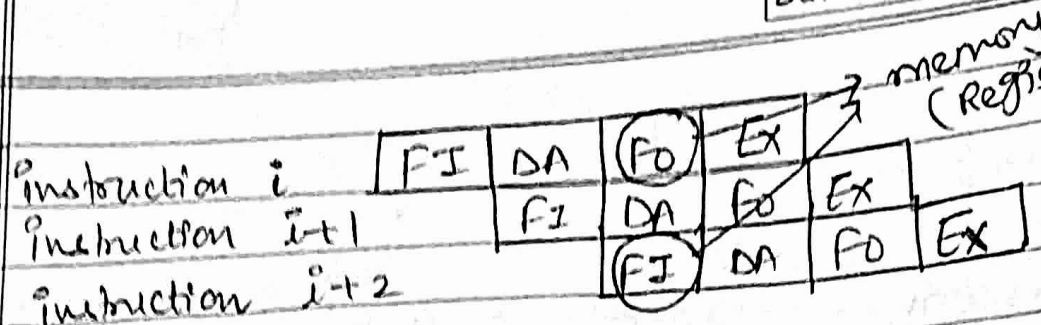
Types of conflicts

1. Resource conflicts (Structural Hazards)
2. Data dependency (Data Hazards)
3. Branch difficulty (Control Hazards)

1. Resource conflicts (Structural Hazards)

* Resource conflict is a situation when more than one instruction tries to access the same hardware resource in the same cycle.
* A resource can be a register, memory or ALU.
* It is also called structural hazards.
* Let an instruction has 4 pipeline segments
  1. Fetch instruction (FI)
  2. Decode instruction & calculate effective address. (DA)
  3. Fetch operands (FO)
  4. Execute instruction. (EX)

Date:

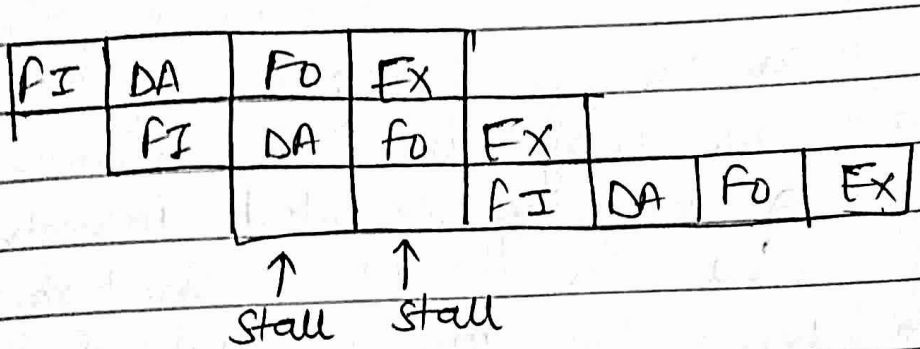


Resource conflicts (Structural hazards)

Solution to Resource conflicts

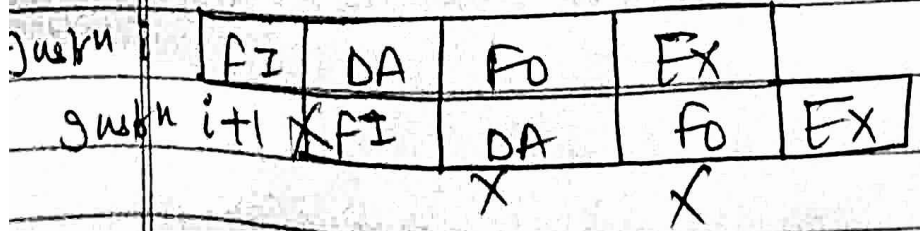
① Use of separate data memory & instruction memory resolves most of the conflicts.

② Use of pipeline stall (It is delay in the execution of an instruction in order to resolve a hazard)

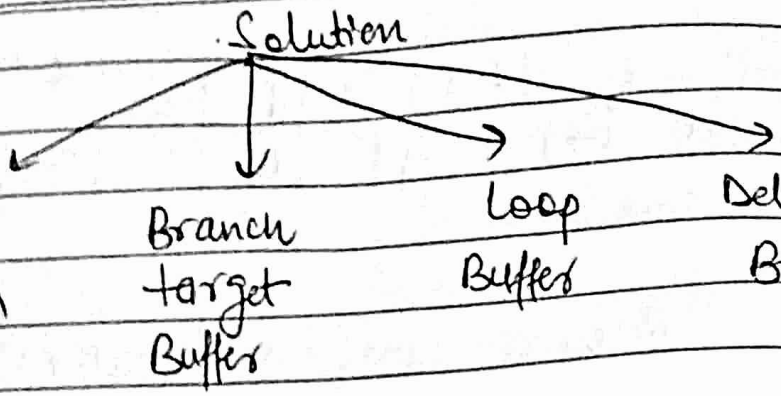


Branch difficulties

* Branch difficulties occurs in branch instructions (conditional / unconditional jump instructions) in which the branch target address is not known until the branch instruction is decoded.



Date: / /



① Prefetch Branch Target Instruction :- When a conditional branch is recognized, the target instruction is fetched in advance along with the instruction following the branch.

② Branch type Target Buffer (BTB) :- It is an associative memory included in the fetch segment of the pipeline.

* Each entry in BTB consists of address of previously executed branch instruction and its target instruction.
* When the branch instruction is decoded, it is first searched in BTB and if it is found, the instruction is available directly else the pipeline shifts to new instruction & store the target instruction in the BTB.

③ Delayed Branch :- The compiler detects the branch instructions and rearrange (Re-order) the instructions.

Page No.:
Date: / /

Original Order	Re-order
① Add R_2, R_3	① Decrement R_1
② Decrement R_1	② If $R_1=0$, goto 40
③ If $R_1=0$, goto 40	③ Add R_2, R_3
(40)	(40)

④ Loop buffer :- A loop buffer is a small, very high speed memory maintained in the FI (Fetch Instruction) stage.

* It contained the most recent fetched instruction.
* If a branch is taken, first the target instruction is searched in the load buffer
* If exist, it is fetched from the buffer to save time.