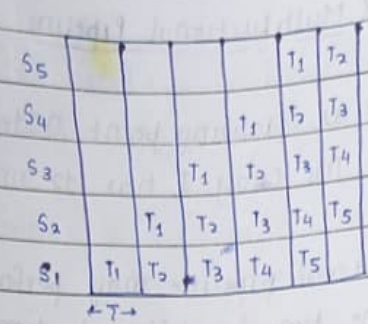
Date
Page No.

Pipelining

Necessarily, there is an overlapping in between 2 stages of Instruction Execution.

Diagram:



For Non-Piplined System (Time) = $kn\tau$
For Pipelined System (Time) = $(k+(n-1))\tau$

Classification of Pipelined Processor

According to the level of processing Händler (1977) has proposed the following:

1) Arithmetic Pipelines - The Arithmetic logic Unit of a computer can be segmentized for pipeline operation in various data format.
Eg- The 4 stage pipeline used in Star-100, 8 stage pipeline used in TIASC (Texas Instrument Advanced Science Computer)

2) Instruction Pipeline - The exeution of a stream of instructions can be pipelined by overlapping the execution of the current execution with the Fetch, Decode, Operand-fetch of subsequent instruction

3) Processor Pipelining - Processor Pipelining refers to the pipelined processing of the same data stream by a cascade of processors each of which process a speafic task

According to Pipelined Configuration & Control Strategies Ramamoorthy & Li (1977) had proposed the following pipline classification scheme.

1) Unifunctional & Multifuctional Pipeline - A piplined unit with a fixed and dedicated function such as floating point adder is called unifunctional pipeline. Eg- The Cray-I has 12 unifunctional pipeline units

A multifunctional pipeline may perform different function either at same time or different time. Eg - By interconnecting different subsets of stages TIASC has 4 multifunctional pipeline units.

2) Static & Dynamic Pipeline - A static pipeline may assume only one functional configuration at a time. Pipelining is made possible in static pipelines only if instruction of the same type are to be executed continuously. The function performed by a static pipeline should not change frequently.

A dynamic pipeline processor permits several functional configurations to exist simultaneously.

3) Scalar & Vector Pipelined Processor - A scalar pipeline process a sequence of scalar operation under the control of a loop.

Vector pipeline are specially designed to handle vector instruction over the vector operands.

H/W → Read → Throughput (ω, η)
Speed Up (S_k)

Date 8/2/22
Page No.

Architectural classification (Also known as Flynn's classification)

Based on the multiplicity of instruction streams and data streams in a computer system: (1965)

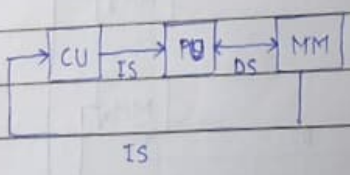
* Stream - is used to denote a sequence of items as executed or operated by a single processor.

* Instruction Stream - is a sequence of instructions as executed by the processor.

* Data Stream - is a sequence of data including input, partial and or temporal result called for the instruction stream.

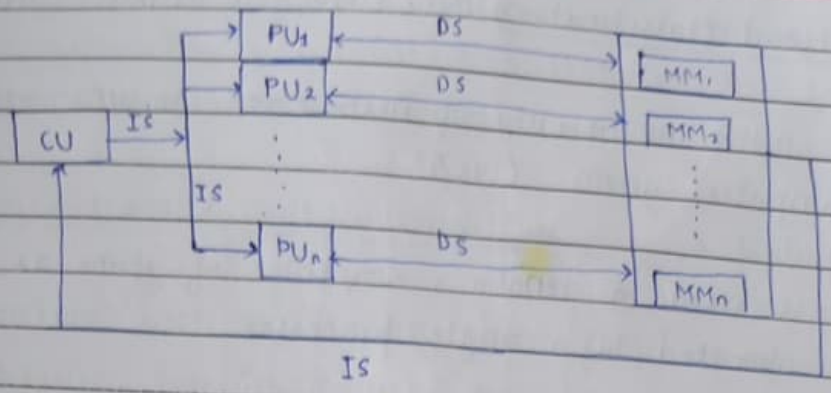
1) Single Instruction Single Data Stream (SISD) - A single processor executes a single instruction to operate on data stored in a single memory. Instructions are executed sequentially but may be overlapped in their execution stages.

Eg- IBM Uniprocessor - 701, IBM - 1620



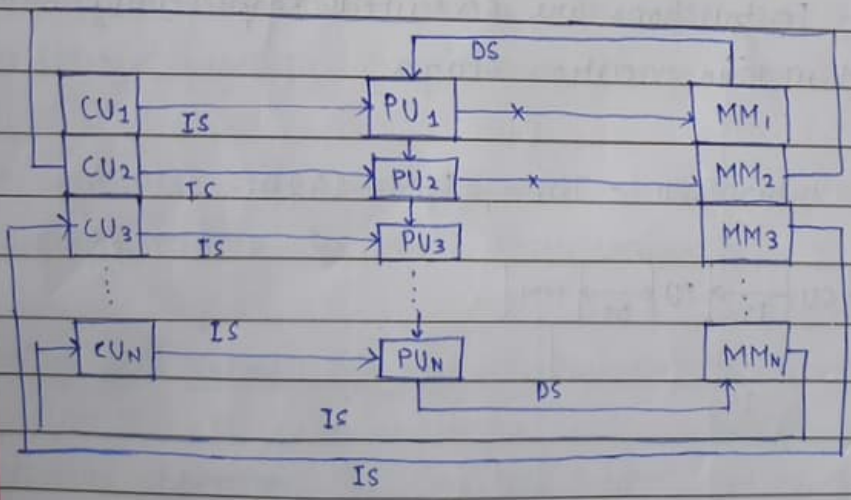
2) Single Instruction Multiple Data Stream (SIMD) - A single machine instruction controls the simultaneous execution of processing element (PE)

Eg- Vector Processor - ILLIAC - IV

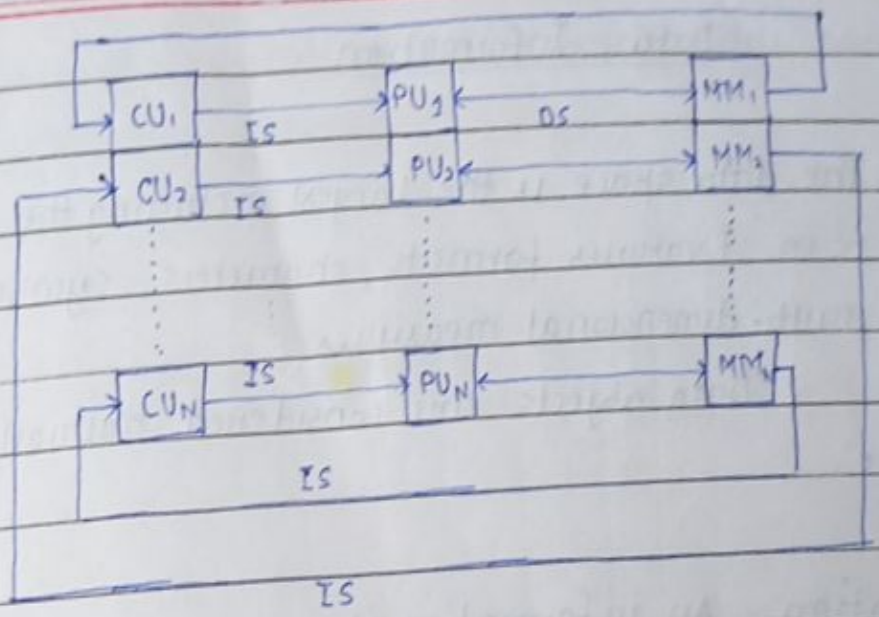


A[30]
B[30]
C[30] = A+B

3) Multiple Instruction Single Data Stream (MISD) - A sequence of data is transferred to a set of Processor each of which execute a different instruction sequence. Their n Processing Unit each receiving distinct instruction operating on same data and output of one processor becomes input of next processor.



4) Multiple Instruction Multiple Data Stream (MIMD) - A set of processor simultaneously execute different instruction sequence on different data set.



Example - IBM - 370 , Cray - II, IBM - 3081 , UNIVAC - 1100/80

Date 9/2/22
Page No.

Data, Information

- **Data** - The data space is the largest including the numeric numbers in various formats, characters, symbols and multi-dimensional measures.
  Data objects are considered mutually unrelated in space.

- **Information** - An information item is a collection of data objects that are related by some syntatic structure or relation. This derives into three architectural configuration.

- **Knowledge** - Knowledge consists of information contents plus some semantic meanings.

- **Intelligence** - Intelligence is derived from a collection of knowledge items.

- **Parallel computer Structures**

Parallel computers for computing - Parallel computer are those system that emphasize parallel processing.

1) Pipeline computer
2) Array Processor
3) Multiprocessor system.

(1) A pipeline computer performs overlapped computation to explore temporal parallelism.

(2) An array processor uses multiple synchronized ALUs to achieve parallelism.

③ A multiprocessor system achieves asynchronous parallelism through set of interactive processor

Date
Page No.

Computer Architecture Prac 1

ASIC - Application Specific IC
FPGA - Field Programmable gate array.

- Language - Verilog. File Extension - .vl

```
Module name (input a; input b; output c);
begin
assign c = a & b;
end
end module
```

For testing verilog code we use test bench.

Eg -
```
module testabc
    a = .x ;
    b = .y ;
    c = .z ;
    initial
    x = 1 ;
    y = 0 ;
    # 100;
module name a1 (a(x); b(y); c(z));
```

* At the end we have to call the module
* end & end module.

Date
Page No.

AND Gate

Theory :- The AND gate is a basic digital gate that implements logic conjunction (&) from mathematical logic - AND. A High output (1) results only if all the inputs to AND gate are HIGH (1). If none or not all inputs to the AND gate are HIGH, LOW output results.

Truth Table :

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

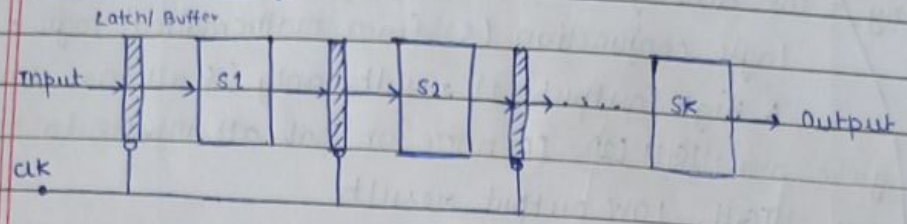
Code :

```
`timescale
```

Date 15/2/22
Page No.

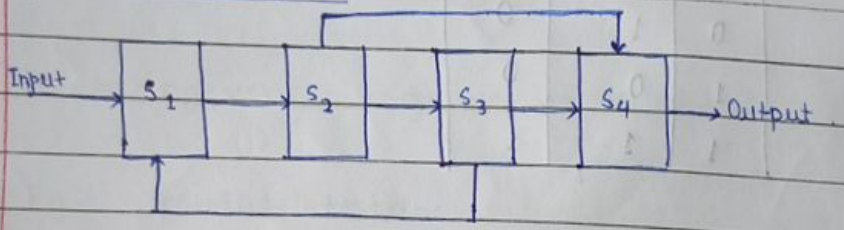
Non-Linear Pipeline

- Linear Pipelining



In case of linear Pipeline there is no feed forward or feed back connection available

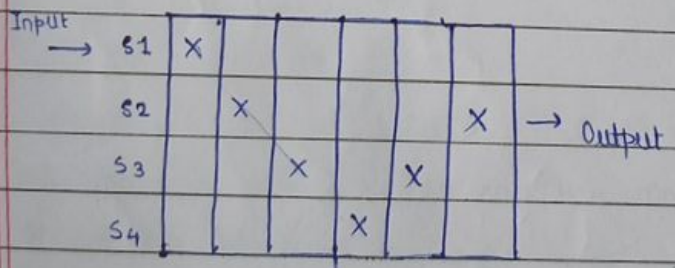
- Non-Linear Pipeline



Non-Linear Pipeline

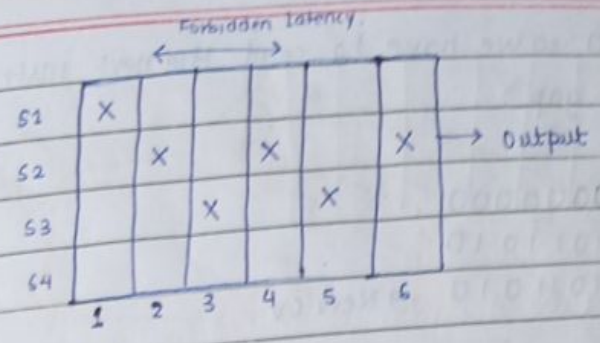
After getting executed in S2 whether the data will move to S3 or S4? There is an amiguity as a result reservation table is used

Reservation Table



Collision - When more that one instruction is placed in the reservation table in the same stage then there is a collision.

Forbidden Latency - Number of clock-cycle difference b/w two colliding instructions.



FL = 2, 4.

PL = 1, 3, 5, 6 (Permissible Latency).

Collision Vector

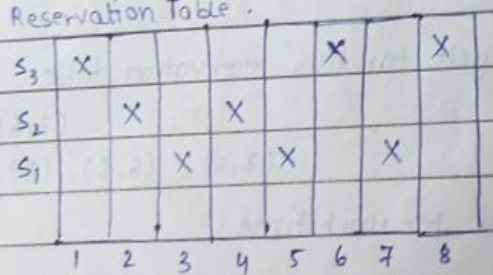
$C = C_6\ C_5\ C_4\ C_3\ C_2\ C_1$

We have to look at the values of the forbidden latency and place 1 at place where the latency is forbidden.

C = 001010 = 1010.

16/2/22.

Reservation Table.



Forbidden Latencies = 2, 4, 5, 7.

Collision Vector = $C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1$

= 1011010.

If we send a task in machine after 1 latency, new collision vector will be. (For T_1).

C_2' = 0101101 (After shift operation on inital CV).

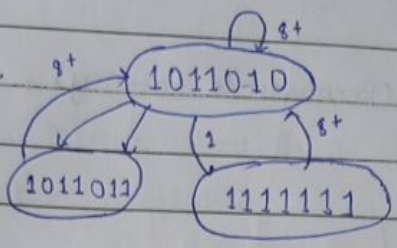
1011010 (For T_2).

0 1 1 1 1 1 1 1 (New CV).

As C_8 is 0 so we have to send the next instruction I_3 at 8 latency gap.

```
0000000
1011010
1011010  → New CV.
```

State Diagram.



After 3 latency gap if we insert another instruction

```
0001011  (for T1)          0001011
OR 1011010            OR   1011010
   1011011  (for T2)       1011011  (for T3).
```

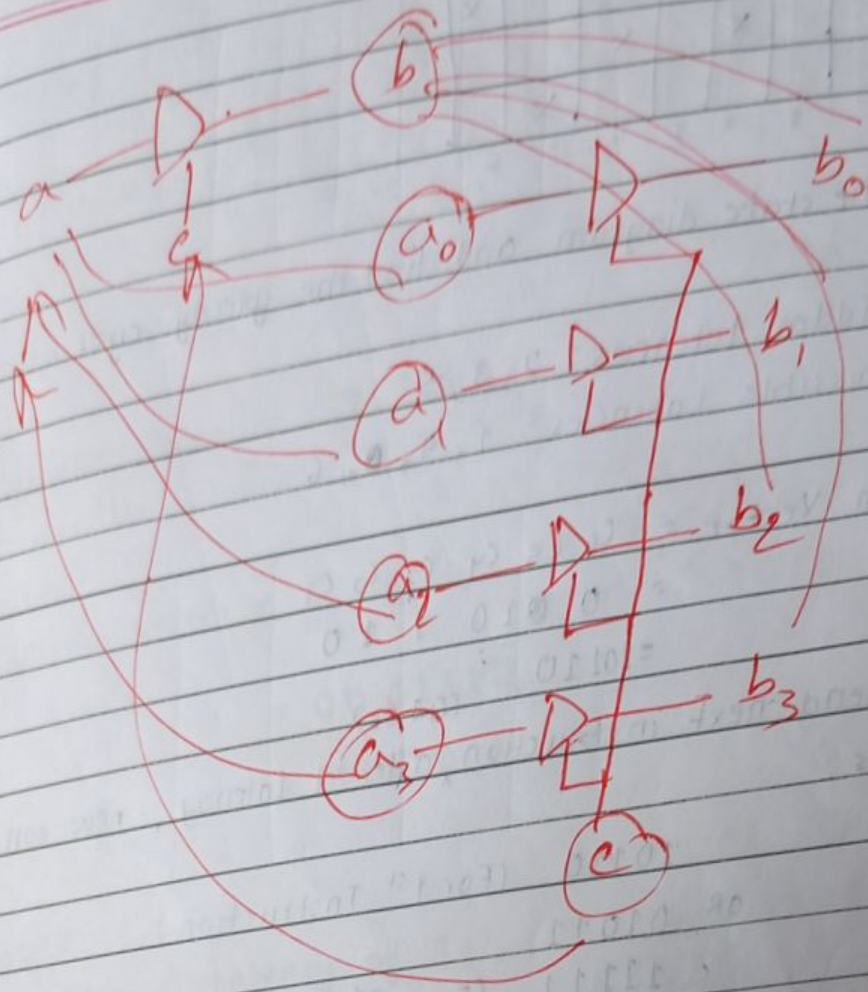
Permissible latency cycle for this reservation table - (1,8) (3,6) (3) (6) (3,8); (6,8), (8).

Throughput - Output per unit time.

Average latencies - 4.5, 4.5, 3, 6, 5.5, 7, 8.

Minimum Average Latency = 3.

Greedy cycle - If each latency contained in the cycle is minimal latency and form a state in the cycle then this state is known as greedy cycle.

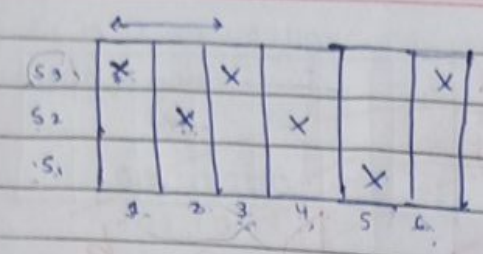


b[3:0] a[0]

a[3:0]

c

Q



Q) Draw the state diagram and find the greedy cycle.

Ans): Forbidden latencies : 2, 3, 5

Permissible latencies : 1, 4, 6.

Collision Vector = $C_6\ C_5\ C_4\ C_3\ C_2\ C_1$

= 0 0 1 0 1 1 0

= 10110. (F1 & F2)

(I) If we send next instruction after 1 latency, new collision vector is

10110 (For 1st Instruction)

OR 01011

11111 (For 2nd Instruction)

(I_2)

If we send next instruction after 6 latency, new collision vector is :-

000000

10110

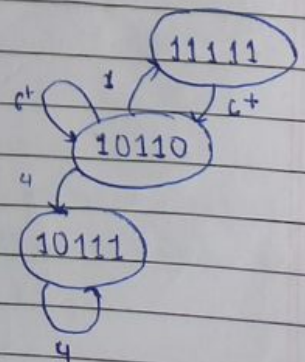
010110 → For (I_3).

(II) If we send ~~seco~~ I_2 after 4 latency. State Diagram then new collision vector will be

10110

OR 00001

10111



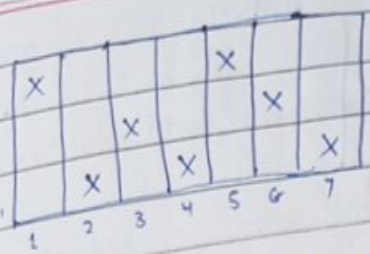
(III) Permissible latency cycle : (1, 6), (4).

Avg. latency cycle : (3.5), 4.

min

Greedy cycle = (1, 6).

Q)



State Diagram & Greedy Cycle

A) Forbidden Latencies = 2, 3, 4, 5

Permissible Latencies = 1, 6, 7.

Initial Collision Vector = $C_7 C_6 C_5 C_4 C_3 C_2 C_1$

$= 00\ 11110$.

$= 11110$.

Case I

T_2 after 1 Latency of T_1.

```
  11110
OR 01111
  11111  → New Collision Vector.
```

T_3 after 6 latency of T_2.

```
  11110
OR 00000
  11110  → New Collision Vector.
```

State Diagram

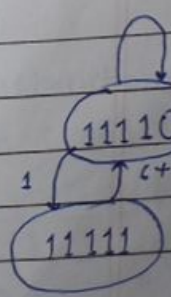
~~Case II~~ Permissible latency cycles - (1, 6), (6)

Avg. Latency Cycle = (3.5) 6

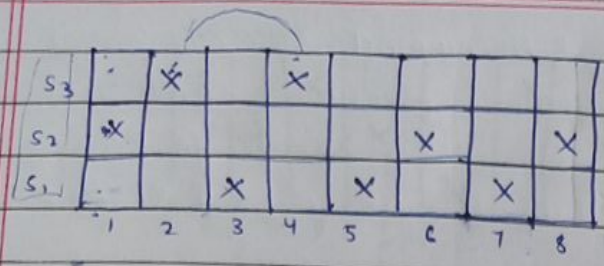
min.

~~T_3 after~~

Greedy Cycle = (1, 6).



Q)



FL = 2, 4, 5, 7

PL = 1, 3, 6, 8

CV = $C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1$

= 0 1 0 1 1 0 1 0

= 1011010

Case I

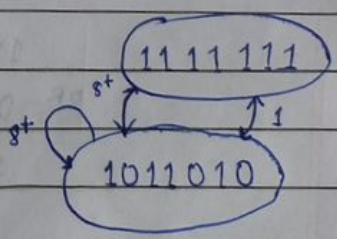
T_2 after 1 Latency of T_1.

```
  1011010
OR 0101101
  1111111  → (New Collision Vector).
```

T_3 after 8 Latency of T_2.

```
  1011010
OR 0000000
  1011010  (New CV)
```

State Diagram



Hazard

Any condition that causes the pipeline to stalled is called a hazard.

- Data Hazard - Data Hazard is a condition in which either the source or destination of operands of an instruction are not available at the expected time as a result some operation has to be delayed and causes the pipeline stall.

Technique to Handle Hazard

The domain D(I) of an instruction I is a set of resource object whose data object may affect the execution of an instruction. The range R(I) of an instruction I is a set of resource object whose data object may be modified by the execution of instruction I. Obviously, the operands to be used in an instruction execution are retrieved from its domain and the result will be stored in its range.

The are three class of data hazard according to various data update patterns:

i) Read after Write - Hazard between two instructions I and J may occur when J attempts to read some data object that has been modified by I.

ii) Write after Read - Hazard between two instructions I and J may occur when J attempts to modify some data object that is read by I.

iii) Write after Write - Hazard between two instructions I and J may occur if both I and J attempt to modify the same data object.

* In case of Read after Write $R(I) \cap D(J) \neq \phi$.
* Eg- I A: B+C
  J D: A+M
  } → Read after write.

* In the example Domain of I is D(B,C) and range of I is R(A). Similarly. Domain of J is D(A,M) and range of J is R(D). So, if we find R(I) ∩ D(J) = {A} ≠ ∅ So, the Hazard occurs.

* In case of write after read R(J) ∩ D(I) ≠ ∅

* In case of write after write R(I) ∩ R(J) ≠ ∅.

Methods for Removing / Dealing with Data Hazard

(1) Internal Forwarding - Internal Forwarding refers to a "short-circuit" technique for replacing unecessary memory access for register to register transfer in a sequence of Fetch, Arithmetic and Store operation.

(I) Store Fetch Forwarding - Eg -

1 { $M_i \leftarrow R_1$; $R_2 \leftarrow M_i$ | $M_i \leftarrow R_1$; $R_2 \leftarrow R_1$ } 2

The operation 1 requires comparatively more time for execution compared to operation 2. So 2nd one is better option and is known as Store fetch forwarding

(II) Fetch Fetch Forwarding -

1 { $R_1 \leftarrow M_i$; $R_2 \leftarrow M_i$ | $R_1 \leftarrow M_i$; $R_2 \leftarrow R_1$ } 2

Since, 1st operation requires to access memory more number of times compared to 2nd operation, 2nd operation requires less time to execute and this is known as fetch-fetch forwarding.

(III) Store Store Forwarding - 1 { $M_i \leftarrow R_1$; $M_i \leftarrow R_2$ | $M_i \leftarrow R_2$ } 2

Since R_2 is overwriting the data on M_i created by R_1 so we can overlook the 1st step of 1st operation to

reduce the time required making the execution faster. This method is known as store-store forwarding.

(2) Register Tagging - Register Tagging refers to the use of tagged registers, buffers and reservation station for concurrent activity among multiple arithmetic unit.

Set of Instruction -

$R_0 \leftarrow M_1$ → I_1

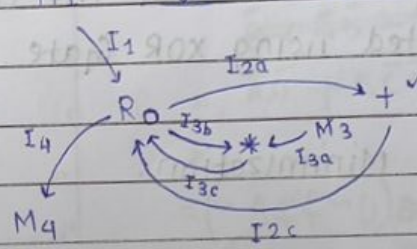
$R_0 \leftarrow R_0 + M_2$ → I_2

$R_0 \leftarrow R_0 * M_3$ → I_3

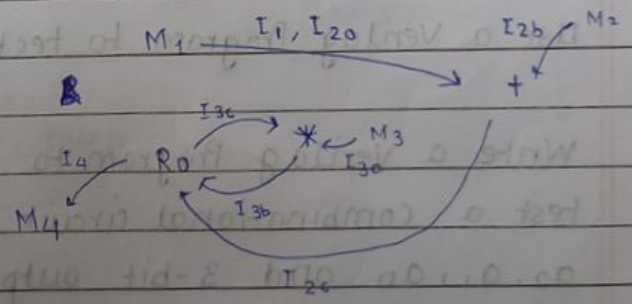
$M_4 \leftarrow R_0$ → I_4

intersection b/w range and domain of multiple instructions occur so hazard occurs.

Diagram - Graph M_1 Register Tagging



STEP-1



STEP 2

Date
Kabita Page No.

Computer Architecture (Assignment-3)

Expt-3 - Minimization Techniques.

Objective - Implementation of Boolean function and Logic Equations.

Problem statement - **Majority Circuit** - A majority circuit is a combinational circuit whose output is 1 if input has more 1's than 0's. Design the majority circuit.

(2) **2's Complement**

Design a 4-bit combinational circuit for implementing 2's complement so that the circuit could be ~~comple~~ constructed using XOR Gate.

(3) Function Minimization.

(a) Implement the Boolean Function. $F(A, B, C, D) = (0, 1, 3, 4, 8, 9, 15)$. Use a Verilog Program to test the ciruit.

(b) Write a Verilog Program to design, simulate and test a combinational circuit with 3-bit input a_2, a_1, a_0 and 3-bit output b_2, b_1, b_0. When binary input is 0, 1, 2, 3 the binary output is 1 greater than the input. When input is 4, 5, 6, 7 in binary output is 1 less than input.

Questionnaires

Q) Explain the difference b/w $ monitor and $ display

Q) Given the following Verilog code, what value of a is

displayed :-

```
always @ clk begin.
a = 0
a <= 1 ;
$ display (a);
end
```

Q3) For the following Verilog code segment, if the initial value of IR is ABCD 3456 in Hexadecimal, the value of "data" in decimal will be _____ (Note that data is a 4-bit variable.

```
wire [
```

Ans)3b)

CD \ AB	$\bar{A}\bar{B}$ 00	$\bar{A}B$ 01	AB 11	$A\bar{B}$ 10
$\bar{C}\bar{D}$ 00	X	X	X	
$\bar{C}D$ 01	X			
CD 11			X	
$C\bar{D}$ 10	X	X		

$Y = \bar{A}\bar{D} + \bar{C} + B$

$f = \bar{A}\bar{D} + B\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} + ABCD.$

$= (\sim A \&\& \sim D) \,||\, (B \&\& \sim C \&\& \sim D) \,||\, (\sim A \&\& B \&\& \sim C) \,||\, (A \&\& B \&\& C \&\& D)$

$Q1 = \bar{A}\bar{D}$ $P2 = B\bar{C}\bar{D}$

$P1 = \bar{A}B\bar{C}$ $S1 = ABCD$

$F =$

$1111 + 1 = 10000$

2) 0 0 0 0 ⊕ 1 →

0 0 0 1

0 0 0 1 0

0 0 1 1

0 1 0 0

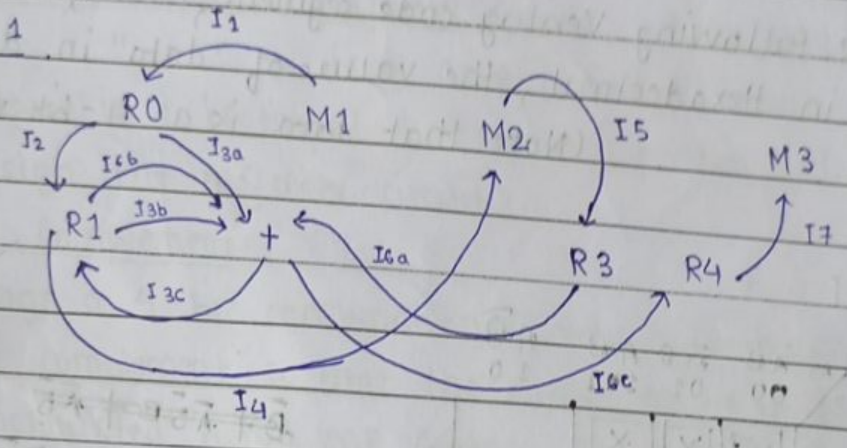
HAZARD REMOVAL

Q) I_1 : $R_0 \leftarrow M_1$
I_2 : $R_1 \leftarrow M_1$
I_3 : $R_1 \leftarrow R_0 + R_1$
I_4 : $M_2 \leftarrow R_1$
I_5 : $R_3 \leftarrow M_2$
I_6 : $R_4 \leftarrow R_3 + R_1$
I_7 : $M_3 \leftarrow R_4$

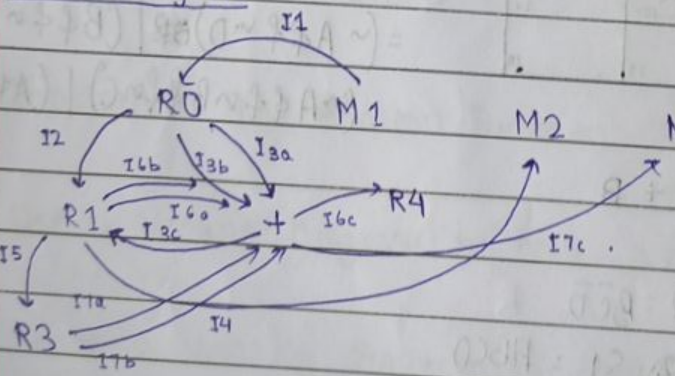
$\rightarrow$

I_1 : $R0 \leftarrow M1$
I_2 : $R1 \leftarrow R0$
I_3 : $R1 \leftarrow R0 + R0$
I_4 : $M2 \leftarrow R1$
I_5 : $R3 \leftarrow R1$
I_6 : $R4 \leftarrow R1 + R1$
I_7 : $M3 \leftarrow R3 + R3$

Step 1



Final Diagram



Q) I_1 : $R_0 \leftarrow M_1$

I_2 : $R_1 \leftarrow M_1$

I_3 : $R_1 \leftarrow R_0 + R_1$

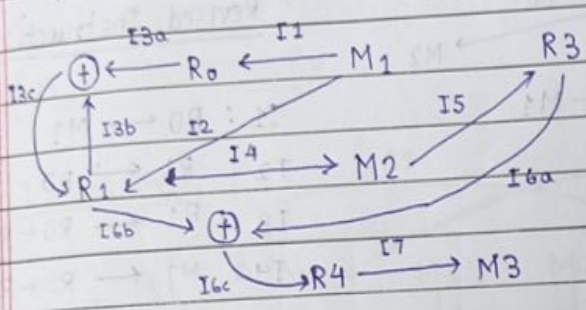
I_4 : $M_2 \leftarrow R_1$

I_5 : $R_3 \leftarrow M_2$

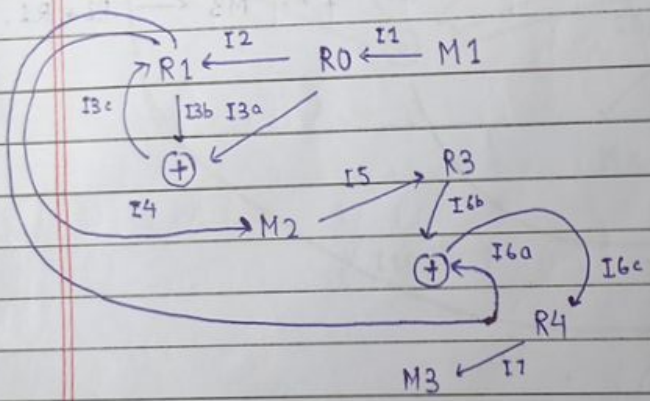
I_6 : $R_4 \leftarrow R_3 + R_1$

I_7 : $M_3 \leftarrow R_4$.

Step 1



Step 2



Revised Instructions

I_1 : $R0 \leftarrow M1$

I_2 : $R_1 \leftarrow R0$

I_3 : $R1 \leftarrow R0 + R1$

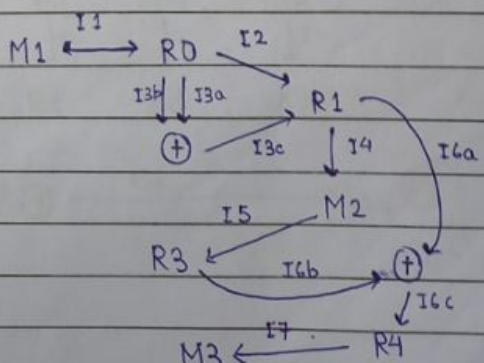
I_4 : $M2 \leftarrow R1$

I_5 : $R3 \leftarrow M2$

I_6 : $R4 \leftarrow R3 + R1$

I_7 : $M3 \leftarrow R4$.

Step 3



Revised Instructions

I_1 : $R0 \leftarrow M1$

I_2 : $R1 \leftarrow R0$

I_3 : $R1 \leftarrow R0 + R0$

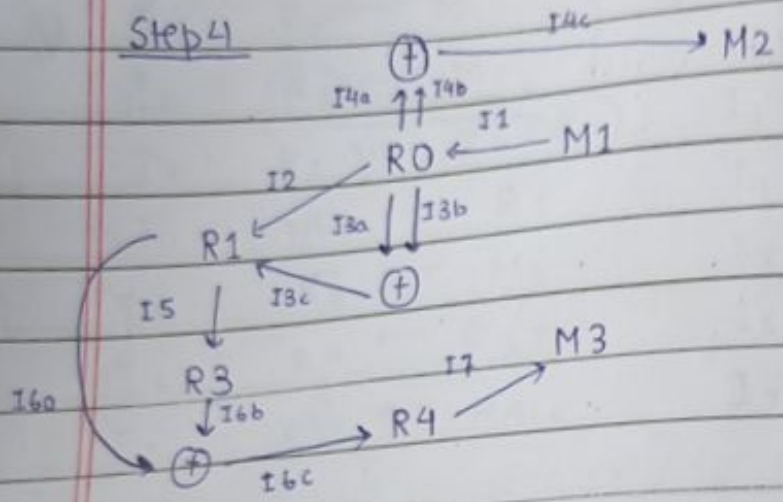
I_4 : $M2 \leftarrow R1$

I_5 : $R3 \leftarrow M2$

I_6 : $R4 \leftarrow R3 + R1$

I_7 : $M3 \leftarrow R4$

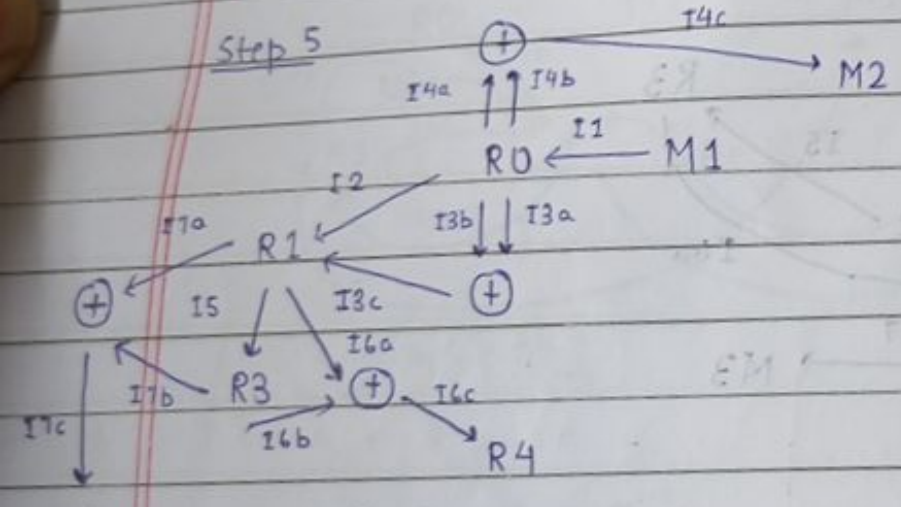
Step 4



Revised Instructions

I1 : R0 ← M1
I2 : R1 ← R0
I3 : R1 ← R0 + R0
I4 : M2 ← R0 + R0
I5 : R3 ← R1
I6 : R4 ← R3 + R1
I7 : M3 ← R4.

Step 5



Revised Instructions

I1 : R0 ← M1
I2 : R1 ← R0
I3 : R1 ← R0 + R0
I4 : M2 ← R0 + R0
I5 : R3 ← R1
I6 : R4 ← R3 + R1
I7 : M3 ← R3 + R1.

I_1 : $R_0 \leftarrow M_1$.

I_2 : $R_1 \leftarrow M_2$.

I_3 : $R_2 \leftarrow M_3$

I_4 : $R_3 \leftarrow M_4$.

I_5 : $R_4 \leftarrow R_0 + R_1$.

I_6 : $R_5 \leftarrow R_4 * M_5$.

I_7 : $M_5 \leftarrow R_4$

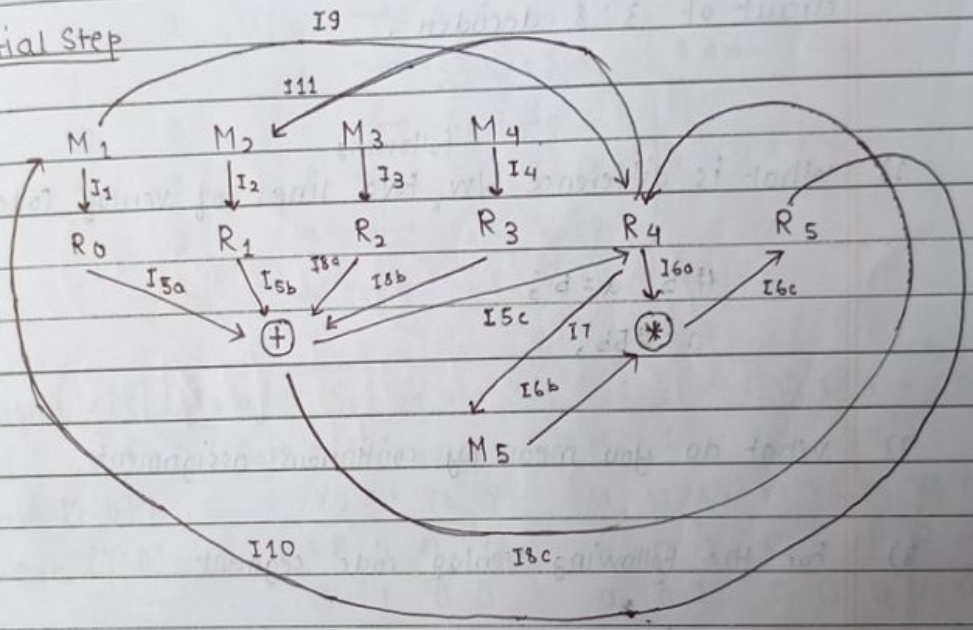
I_8 : $R_4 \leftarrow R_2 + R_3$.

I_9 : $M_1 \leftarrow R_4$

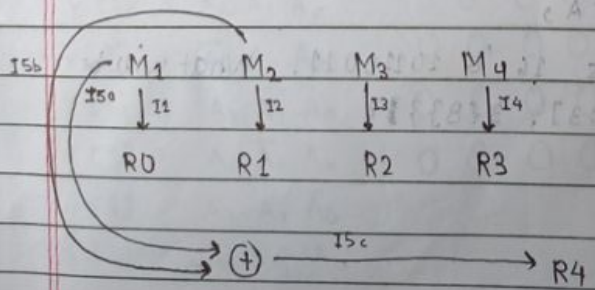
I_{10} : $M_1 \leftarrow R_5$.

I_{11} : $M_2 \leftarrow R_4$.

Initial Step



Step 2



New Instructions

I_1 : R0 $\leftarrow$ M1

I_2 : R1 $\leftarrow$ M2

I_3 : R2 $\leftarrow$ M3

I_4 : R3 $\leftarrow$ M4

I_5 : $R_4 \leftarrow M_1 + M_2$

I_6 : $R_5 \leftarrow$

I_7 :

I_8 :

:

To I_{11}

Combinational Circuit

Objective - Implementation of combinational circuit.

Q1) Write a verilog program that implements multiplexer module 4×1.

Q2) Write a verilog program that includes above module (4×1 MUX) to describe the functionality of 8×1 MUX.

Q3) Write a verilog code to design, simulate and test the circuit of 3:8 decoder.

1) What is difference b/w following two lines of verilog code.

```
#5, a=b;
a=#5b;
```

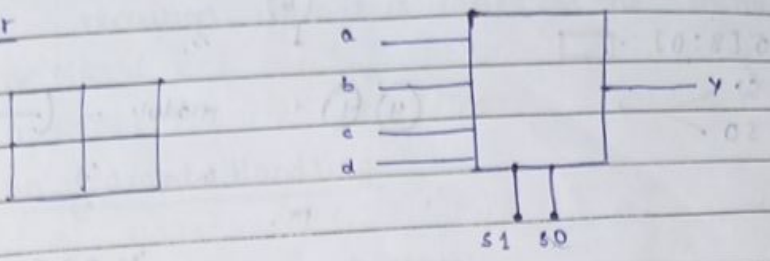
2) What do you mean by continuous assignment.

3) For the following verilog code segment

```
wire [7:0] A;
wire B;
assign B = ^A;
```

if the value of A is 16'b 10110011. What will be the value of {A[4:3], 3{B}}?

A1) Multiplexer



$y = \bar{S1} \cdot \bar{S0} \cdot a + \bar{S1}\, S0\, b + S1\, \bar{S0}\, c + S1\, S0 \cdot d$

S_2	S_1	S_0	Y	
0	0	0	a	$\rightarrow S_2 S_1 S_0 a$
0	0	1	b	$\rightarrow S_2 S_1 S_0 b$
0	1	0	c	$\rightarrow S_2 S_1 S_0 c$
0	1	1	d	$\rightarrow S_2 S_1 S_0 d$
1	0	0	e	$\rightarrow S_2 S_1 S_0 e$
1	0	1	f	$\rightarrow S_2 S_1 S_0 f$
1	1	0	g	$\rightarrow S_2 S_1 S_0 g$
1	1	1	h	$\rightarrow S_2 S_1 S_0 h$

} data.

assign $y = \{\bar{S_2}\bar{S_1}\bar{S_0}\, a + \bar{S_2}\bar{S_1}S_0\, b + \bar{S_2}S_1\bar{S_0}\, c + \bar{S_2}S_1S_0\, d + S_2\bar{S_1}\bar{S_0}\, e + S_2\bar{S_1}S_0\, f + S_2S_1\bar{S_0}\, g + S_2S_1S_0\, h\}$

input [2:0] A;

output [7:0] y;

$y[0] = \bar{A_2}.\bar{A_1}.\bar{A_0}$

$y[1] = \bar{A_2}\bar{A_1}A_0$

$y[2] = \bar{A_2}A_1\bar{A_0}$

$y[3] = \bar{A_2}A_1A_0$

$y[4] = A_2\bar{A_1}\bar{A_0}$

$y[5] = A_2\bar{A_1}A_0$

$y[6] = A_2A_1\bar{A_0}$

$y[7] = A_2A_1A_0$

Y0	Y1	Y2	Y3	A2	A1	A0	Y7	Y6	Y5	Y4	Y4
1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	0	1	1	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0

MUX (4x1).

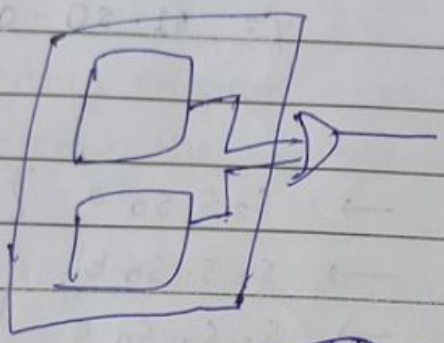
(8) o[3:0]
S1
S0

(3) s2 s1 s0 — 8x1 MUX.

(6)

(4) (4) : module .. ( ).

000 →



$S_2 = 0$? aut : out2

mux a ({s[1:0]}, {I[3:0]}, 01);

mux a2 ({s[1:0]}, {I[7:4]}, 02);

Cache Miss - When instruction is not available, at the time of execution, then it is called as the 'Instruction Hazard' / Cache Miss.

Instruction Hazard Handling.

① Perfect Target Instruction - ~~Prefetc~~

② Prefetch a Bunch of Instructions

③ High Speed Buffer

③ Control Hazard -

```
100  JUMP  200
102  ADD   A, B, C
106  SUB   A, D, E
 :    :     :
190  HLT
 :    :     :
200  ADD   M1, A, M2
204  ADD   A, B, D
208  JUMP  102
```

2 Types of Control Hazard → Conditional Control Hazard
→ Unconditional Control Hazard.

How to Deal with Control Hazard.

- Perfect Target Instruction - a) Fetch both instruction stream branch not taken and taken.

b) Both are saved until branch end.

- Branch Target Buffer - Store the address of previously executed instruction as branch target address and address of next few instruction. When decoding branch

instruction search the branch target buffer.

- Loop Buffer - Storage of entire loop that allow to execute loop without memory access.
- Branch Prediction - Predict the branch condition and fetch an instruction stream based on the prediction.
- Delayed Branch - Compiler defects the branch and rearrange the instruction sequence by inserting useful instruction that keep the pipeline busy in presence of a branch instruction.

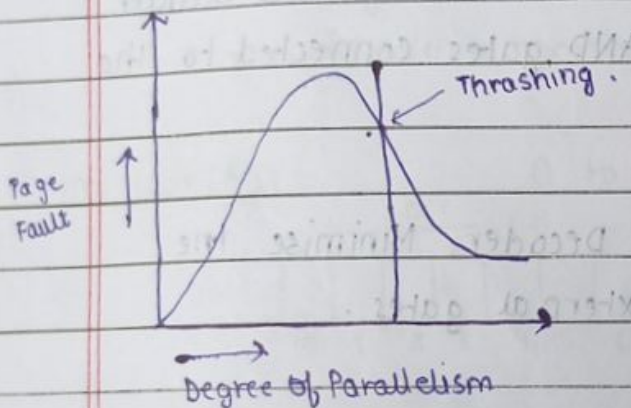
(*) Structural Hazard - When any resource is not available during the time of instruction execution, this is called structural Hazard.

Date
Page No.

Cache Memory

- **Locality of Reference** -

PID-100
PID-101
PID-102
PID-103
PID-104



Page Fault - When a certain page is not available on the RAM during requirement.

Date
Page No.

Combinational Circuit

Objection :- Implementation of Combinational

Q1) Function Implementation using decoder

$F1 = (2, 4, 7)$

$F2 = (0, 3)$

$F3 = (0, 2, 3, 4, 7)$

Write a HDL to implement the circuit with ~~when~~ a decoder constructed with NAND or AND gates connected to the decoder outputs.

Use a Block diagram for the Decoder. Minimise the number of inputs in the external gates.

Q2) Priority Encoder.

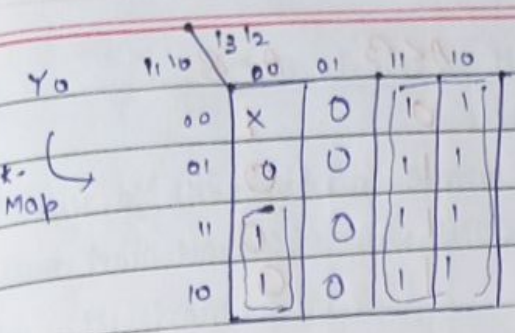
Write a verilog code to design simulate and test a circuit of a 4:2 Priority encoder.

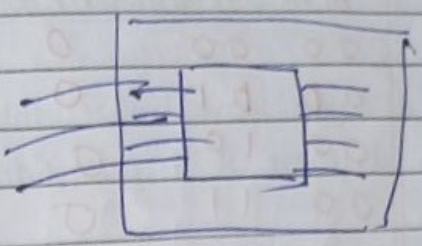
A[3]	A[2]	A[1]	A[0]	O[1]	O[0]
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1
0	0	0	0	x	x

Y1 K-Map

I1I0 \ I3I2	00	01	11	10
00	x	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

$Y_1 = I_2 + I_3$.



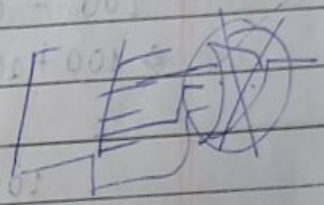


$$Y_0 = I_3 + I_2 \bar{I}_2$$

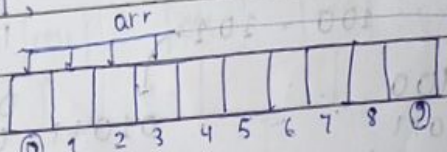
· A(A), · Y(Y)

(3) Magnitude Comparator.

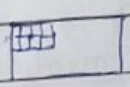
Write a Verilog code to design, simulate and test a circuit of 4 bit magnitude comparator.



int arr [10]; 0 to (n-1).



HDL → Verilog . . .



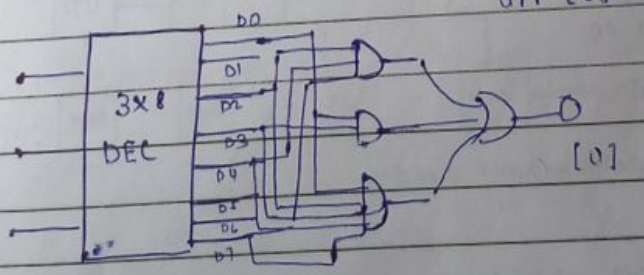
(10) (arr) arr [9:0]

arr[10] = - ;

arr [9] + . . .

arr [9] =

arr [8] =



7 6 5 4 3 2 1 0

000 → 00000001 001 → 0000

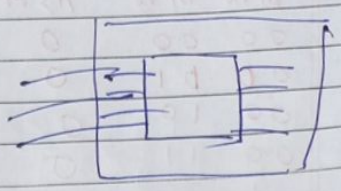
f1 = 0

f2 = 0 ⇒ 0.

f3 = 0.

Y_0 — K-Map

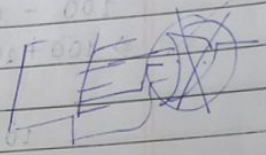
$I_1 I_0$ \ $I_3 I_2$	00	01	11	10
00	X	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1



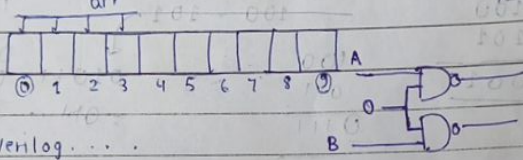
$$Y_0 = I_3 + I_2 \bar{I}_2$$

(3) Magnitude Comparator.

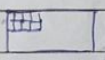
Write a Verilog code to design, simulate and test a circuit of 4 bit magnitude comparator.



int arr[10]; 0 to (n-1)



HDL → Verilog . . . .

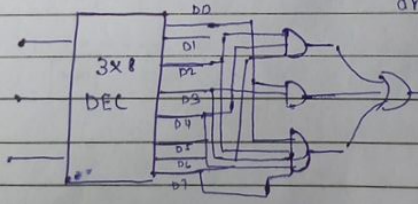


(10) (arr) arr[9:0]

arr[10] = - ;

arr[9] + . . . arr[9] =

arr[8] =



7 6 5 4 3 2 1 0

000 → 00000001 001 → 0000

f1 = 0

f2 = 0 ⇒ 0.

f3 = 0 .

A_1 A_0	B_1 B_0	$A > B$	$A < B$	$A = B$
0 0	0 0	0	0	1
0 0	0 1	0	1	0
0 0	1 0	0	1	0
0 0	1 1	0	1	0

100 - 011

⇒ 100 + 101

100 + 1 → 101

100 - 100

011 + 1

100

100 - 101

010 + 1

= 011

```
  100
+ 101
-----
 1001
```

```
  100
  011
-----
  111
```

```
  0100
- 0100
------
  1000
```

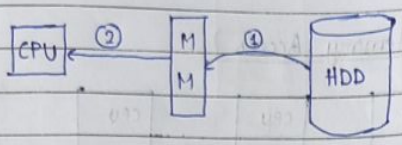
```
  1100
+ 1000
------
 10100
```

```
  1001
- 0100
------
  0101
```

Date
Page No.

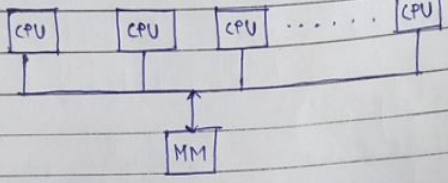
Multi-Processor Memory Management.

In case of Uni-Processor System, data is first transferred to RAM (MM) ~~from~~ from Hard Disk Drive (HDD) and then it is used or accessed by CPU from RAM.

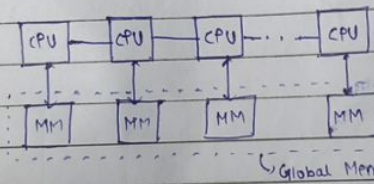


In case of Multiprocessor system, many CPU's access a particular memory module (MM) at the same time / many modules at the same time.

Uniform Memory Access (UMA)



Non-Uniform Memory Access (NUMA)



* Memory access bandwidth is reduced in case of NUMA as compared to UMA.

Local Memory of CPU_n.

* Any memory module directly attached to the CPU is called L1 cache and the global memory module is referred to as the L2 cache.

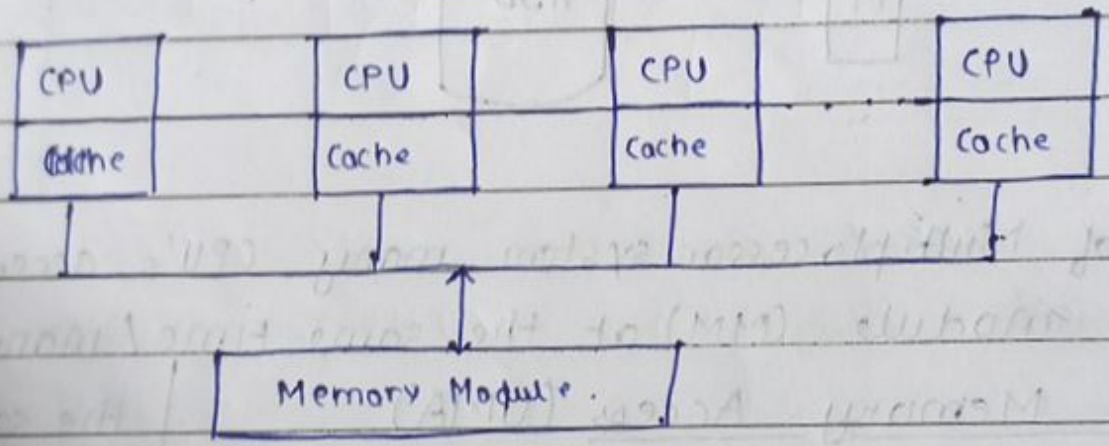
AMD (Advanced Micro devices) used shared memory module for both graphical and CPU processors.

(*) Syllabus for Internal

(1) Pipeline ( )
(2) Cache Memory ( )

→ 5 MCQs (5 marks)
3 SAQs (15 marks)
2 LAQs (30 marks)

COMA (Cache only Memory Access) :



Revision

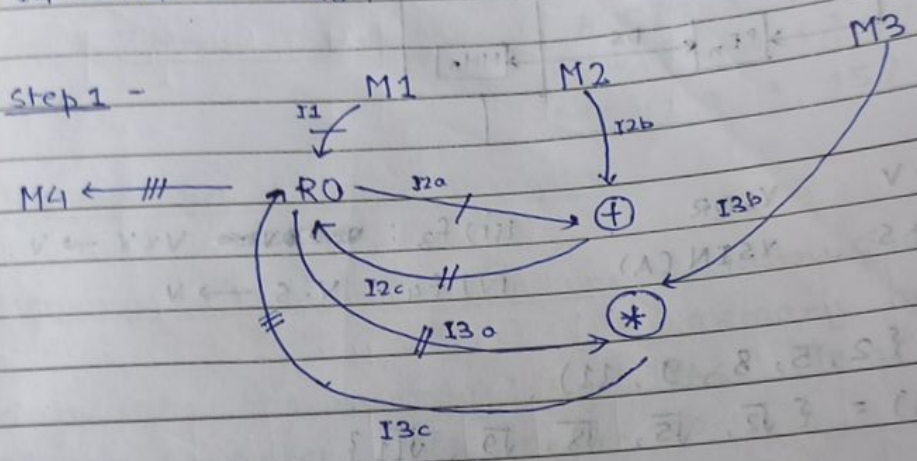
I_1 : $R_0 \leftarrow M_1$

I_2 : $R_0 \leftarrow R_0 + M_2$

I_3 : $R_0 \leftarrow R_0 * M_3$

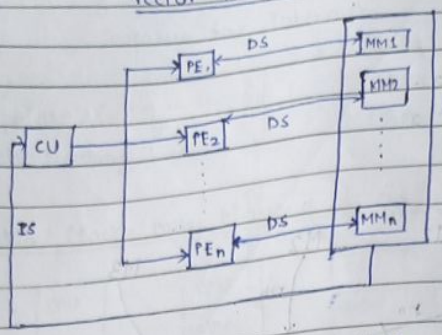
I_4 : $M_4 \leftarrow R_0$

Step 1 -



Step 2 -

Vector Processor



i) $f_1 : V \to V$ VSQR

ii) $f_2 : V \to S$ VSIN (A)

iii) f_3 : $V \times V \to V$

iv) $f_4 : V.S \to V$

Eg 1 - A = {2, 5, 8, 9, 11}.

VSQR (A) = {$\sqrt{2}$, $\sqrt{5}$, $\sqrt{8}$, $\sqrt{9}$, $\sqrt{11}$}.

Eg 2 - A = {2, 3, 6, 7}.

VSUM (A) = 15.

Eg 3 - A = {2, 8, 9, 15}. VTest (A, B) gives a masking vector.

B = {7, 10, 11, 12} C(I) = 1 if A(I) $\leq$ B(I)

C = {1, 1, 1, 0}. C(I) = 0 if A(I) $>$ B(I)

↳ Masking vector.

Type - 3 Eg - A = {2, 8, 9, 15}.

B = {7, 10, 11, 12}.

VSUM (A, B) = {9, 18, 20, 27}.

Type - 4 Eg - A = {2, 8, 9, 15}.

VSUM (A, 5) = {7, 13, 14, 20}.

Special Instruction

i) Boolean Instruction ii) Compress Instruction

(iii) Gather Merge Instruction.

Eg of Compress Instruction - A = {2, 8, 9, 15}
C = {1, 0, 0, 1}
eliminate elements with
B = {2, 15}

Eg of Merge Instruction - A = {2, 8, 9, 15} B = {6, 7, 9, 17}
C = {1, 0, 0, 1}
M = {2, 7, 9, 15}

Architectures of Vector Processor

(1) Memory to Memory Architecture - Instruction is fetched from memory and stores the data into the memory. Eg- TIASC (Texas Instruments Advanced Science Computer), CTC, Cyber-200.

(2) Register to Register Architecture - Cray-1, FUJITSU-VP-200. Instruction is fetched from register and data is stored into the register.

Vector Processing Methods

(1) Horizontal Processing
(2) Vertical Processing
(3) Vector Looping

(*) In case of Horizontal Processing, elements are processed from left to right.

(*) In case of vertical processing, elements are processed from top to bottom.

(*) In case of vector looping, elements are processed from both left to right and top to bottom.

```
output reg [1:0] y;
always @clk begin
casex (x)
4'b0000 :- y <= 2'b00;
4'b001x : y <= 2'b01;
4'b01xx : y <= 2'b10;
4'b1xxx : y <= 2'b11;
default : $display ('Error');
endcase
end
endmodule.
```

x	y
0001	00
001x	01
01xx	10
1xxx	11

$AB + C(A \oplus B)$

Date
Page No.

Expt - 6

Title

Code Converter.

Objective - Implementation of code converter and Familiarization with.

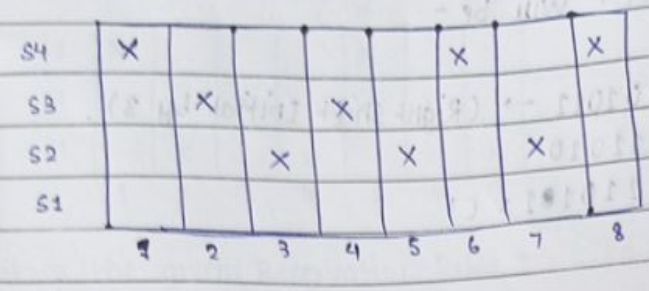
PS 1 - Binary to BCD converter.
PS 2 - BCD to Gray converter.
PS 3 - Binary to Gray Code Converter.
PS 4 - Gray Code to Binary converter.
PS 5 - Binary to Excess - 3 converter.
PS 6 - Excess - 3 to Binary Code converter.

Bin — BCD
0000 — 0000

1001 — 1001
1010 — 0001 0000

21 — 344
0010 0001
0011 0100 0100

Q)



A) From the given Reservation Table -

Forbidden Latency (FL) = 2, 4, 5, 7
Permissible Latency (PL) = 1, 3, 6, 8.

Collision Vector (CV) = $C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1$
= 01011010

∴ Initial Collision Vector is - 01011010

- Case I

If we give an instruction in the pipeline after 1 latency - new collision vector (C') will be -

00101101 → (Right Shift Initial by 1).
OR 01011010
01111111 = C'

Next instruction can only be only after a gap of 8 latency or more - so new collision vector (C") will be -

00000000 → (Right Shift Initial by 8).
OR 01011010
01011010 = C"

- Case II

If we give an instruction in the pipline after 3 latency - new

Collision Vector (C') will be :

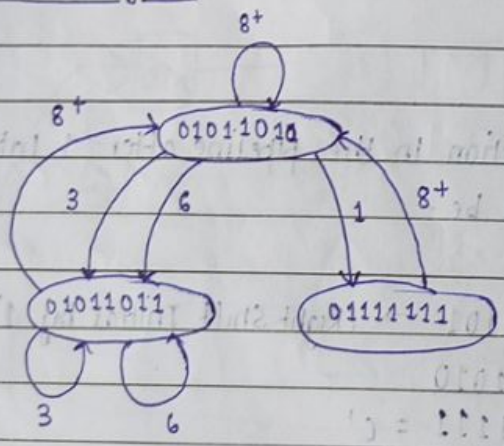
```
  00001011  → (Right shift Initial by 3).
OR 01011010
   01011011 = C'
```

- Case III

If we give on instruction in the pipeline after 6 latency gap - new collision vector (C') will be :-

```
  00000001  → (Right shift Initial by 6).
OR 01011010
   01011011 = C'
```

State Diagram

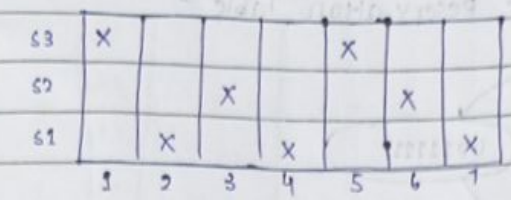


Permissible latency cycles for the Reservation Table :-
(1, 8), (8), (3, 6), (3), (6), (3, 8), (6, 8).

Average latency of cycles = 4·5, 8, 4·5, 3, 6, 5·5, 7

Greedy cycle (cycle with lowest average latency) = (3).

Q)



From the given Reservation Table -

Forbidden latencies (FL) = 2, 3, 4, 5.
Permissible latencies (PL) = 1, 6, 7.

∴ Initial collision vector (CV) = $C_7 C_6 C_5 C_4 C_3 C_2 C_1$
= 0011110

- Cases :

If we give an instruction to the pipeline after 1 latency gap - new collision vector (C') is -

0001111 → (Right shift Initial by 1 bit).
OR 0011110
0011111 = C'

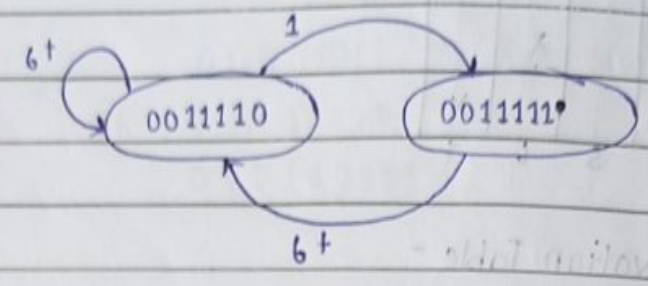
Again if we give an instruction to the pipeline after 6 latency gap - new collision vector (C'') is -

0000000 → (Right shif Initial by 6 bits).
OR 0011110
0011110 = C''.

Again if we give an instruction to the pipeline after 7 latency gap - new collision vector (C''') is -

0000000 → (Right shift Initial by 7 bits)
OR 0011110
0011110 = C'''

State Diagram for the Reservation Table -



Permissible Latency Cycles for the Reservation Table - (1,6), (6).

Average Latency of Cycles : 3.5, 6.

Greedy Cycle (cycle with lowest average - (1,6).

Dpeline Chaining

Pipeline chaining is expanded from the concept of internal forwarding. Basically, chaining is a linking process that occur when results obtained from one pipelined unit are directly feed into the operand registers of another functional pipelines. In other words intermediate results do not have to restored into the memory and can be used even before the vector operation is. Chaining permits successive operations to be issued as soon as the first result become available as an operand. Ofcourse the desired functional pipeline and operand registers must be properly reserved otherwise chaining operation have to suspended until the demanded resource become unavailable.

Eg- Design of a vector compiler.

Major optimizing function are given below according to the level of sophistication in generating efficient scalar and vector code modules.

i) General optimization - a) Common Expression Elimination
b) Invariant Expression Movement.
c) Strength Reduction
d) Register Optimization.
e) Constant Folding.

ii) Extended Optimization - a) Intrinsic Function Integration (Eg - sqrt, sin)
b) Reduction of Iteration number in nested loop.
c) Recorder of execution sequence to reduce pipeline overhead.
d) Temporal Storage Management.
e) Code Avoidance.

iii) Vector Extended Optimization - a) Full Vectorization.
b) Pipeline Chaining.
c) Pipeline Antichaining.
iv) Vector Register optimization.
v) Patta Parallelization.

- MIPS (Million Instructions per seconds).

What is MIPS?

The speed of a scalar processor is usually measured by the number of instructions executed per unit time. Such as the use of million instructions per second).

- MegaFlops

What is MegaFlops?

For a vector processor, it is universally accepted to measure the number of arithmetic operations performed per unit time such as the use of mega floating point operations (MegaFlops) per second.

iii) Vector Extended Optimization - a) Full Vectorization.
b) Pipeline Chaining.
c) Pipeline Antichaining.
iv) Vector Register optimization.
v) Patta Parallelization.

- MIPS (Million Instructions per seconds).

What is MIPS?

The speed of a scalar processor is usually measured by the number of instructions executed per unit time. Such as the use of million instructions per second).

- MegaFlops

What is MegaFlops?

For a vector processor, it is universally accepted to measure the number of arithmetic operations performed per unit time such as the use of mega floating point operations (MegaFlops) per second.