Date ....
Page No. ....

Pentium Multimedia Extended Instruction (MMX)

In 1996, Intel introduced multimedia extension instruction technology into its Pentium product line.

Multimedia Extension Instructions or multimedia task. There are 57 new instruction that treat data in a SIMD fashion, which makes it possible to perform the same operation such as addition or multiplication on multiple data elements at once. Each instruction typically takes a single clock cycle to execute. For the proper application this parallel operation can speed up of 2 to 8 times over comparable algorithms that do not use the MMX instruction.

The MMX technology use 3 new data types -

1. Packet Bytes (that is 8 bit packet).
2. Packet Words (four 16 bit word)
3. Packet Double Word (2× 32 bit word).

	Category.	Instruction	Description
1.	Arithmetic	PADD (Parallel Add).	Parallel addition of 8 bit packet, 16 bit Word, 32 bit double word.
		PMULL (Parallel Multiplication	Parallel Multiplication of 4 signed 16 bit words.
2.	Comparision	PCMPGT.	Parallel compare for greater than result is 1 or TRUE and less than result is 0 or FALSE
3.	Data Transfer	MOV	MOVE double word or word word to or from MMX regis

4.	logical	POR (Parallel OR)	Parallel logical OR 64 bit Bitwise
5.	Shift	PSAR	Parallel Arithmetic Right shift of Packet word or double word.

Improve Processor Performance

Pentium Processor with MMX technology can provide a 10 to 20 % performance boost over classical pentium processor over same freq- -uency. In addition, the MMX technology versions of the processor double on chip code and data cache to 16 kilobytes and feature improved Branch Predition and enhance pipeline and deeper write buffer for improved performance.

* Advantages of MMX technologies

MMX technology provides 57 new instruction sets to improve processor performance in traditional digital signal Processor (DSP) application including graphics, voice and audio processing capabilities emerging as value added features in High-Performance embedded products.

MMX technology can potentially eliminate the requirement for DSP chips in embedded applications such as tele communication devices.

Real Problems solved using Vector Processor

$$S_i = \sum_{i=0}^{k} A_i$$

1
2
3

In Uniprocessor system, clock cycles Required = 28

$S0 = A0$

$S1 = A0 + A1$

$S2 = A0 + A1 + A2$ ~~+ A3 +~~

$S3 = A0 + A1 + A2 + A3$

$S4 = A0 + A1 + A2 + A3 + A4$

$S5 = A0 + A1 + A2 + A3 + A4 + A5$

$S6 = A0 + A1 + A2 + A3 + A4 + A5 + A6$

$S7 = A0 + A1 + A2 + A3 + A4 + A5 + A6 + A7$

~~S8~~

$\frac{8 \times 7}{2}$

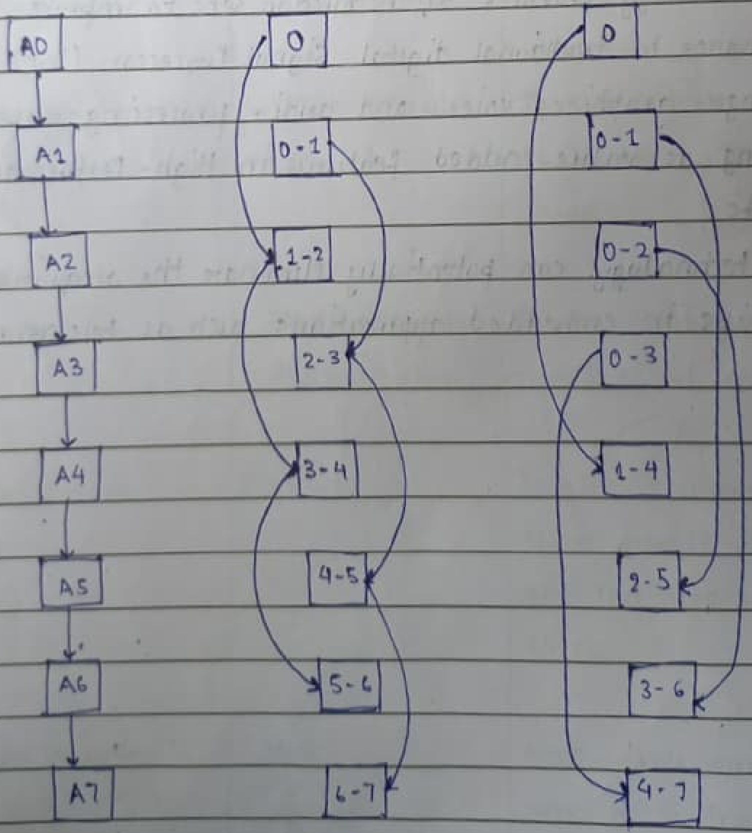
28

$8 + 7 +$

$\frac{8 \times (8-1)}{2} = \frac{8 \times 7}{2}$

$4 \times 7 = 28$

In Multiprocessor system, No. of clock cycles = 3.



Step 1 $R_i = R_{i+1}$

Step 2 $R_i = R_{i+2}$

Step 3, $R_i = R_{i+4}$.

Date
Page No.

Non - ~~John~~ Vonn Neumann Architecture

- Data Driven Computing - Data flow computers are based on the concept of data driven computational non - Von Neumann Architecture. The fundamental difference is that instruction execution in a conventional computer is under program flow control whereas, $\overset{\text{that}}{,}$ in a data flow computer is driven by the data availability. Jack Dennis in 1979 of MIT has identified 3 basic issues towards the development of an ideal architecture for future computer. The first is to achieve a high performance per cost ratio. The second is to match the ratio with technology progress and the third is to offer better better programmability in the application areas. The data flow model offers an approach to meet these demands.

* Special Features of Control $\overset{\text{Data}}{\wedge}$ Flow Computers

- Data is passed b/w instructions via reference to share memory

- Flow of control is implicitly sequential but special control operators can be used explicitly for parallelism.

- Program counters are used to sequence the execution of instructions in a centralized control environment.

* Features of Data Flow Computers

- Intermediate / Final Results are passed directly as data token b/w instructions.

- There is no concept of shared data storage as embodied in the traditional notation of variable.

- Program Sequencing is constrained only by data dependency among instructions.

Note - Data driven concept means asynchroning which means that many instructions can be executed simultaneously and asynchronously. A high degree of asynchronous parallelism is expected in a data flow computer. Because there is no use of shared memory cells, data flow programs are free from side effects.
Information Items in a data flow computer appear as operation packet and data tokens. An operation packet is composed of the opcode, operands and destination of its successor instruction. A data token is formed with result value and its destination.

~~Data~~

Data Flow Machine Architecture

Depending on the way of Handing data tokens, data flow computers are divided into static and dynamic model.
In a static data flow machine data tokens are assumed to move along the arcs of the data flow program graph to the operator nodes. The ~~total~~ nodal operation gets executed when alls its operand data are present in the input arcs. only one token is allowed to exist on any arc at a given time, otherwise the successive sets of token cannot be disting-uished. This architecture is considered static because tokens are not labelled and control tokens must be used to acknowledge the proper timing in transferring data tokens from node to node

Dynamic Data Flow Machine - Dynamic Data Flow Machine uses tagged token so that more than one token can exist in an arc. The tagging is achieved by a label with each token which uniquely identifies the context of that particular token. This dynamically tagged data flow model suggest that maximum parallelism can be exploited from a program graph. If the graph is cyclic, the tagging allows dynamically unfolding of the iterative computation.

Data Flow Graph

A data flow graph is a directed graph whose nodes corresponds to operators and arcs points to data token.

$P = X + Y$

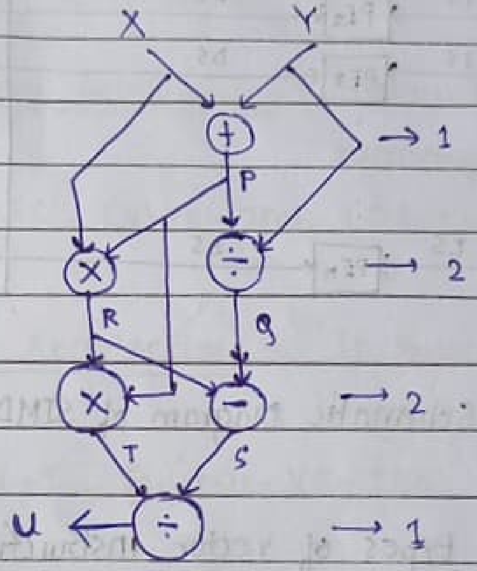
$Q = P \div Y$

$R = X \times P$

$S = R - Q$

$T = R \times P$

$U = S \div T$



Maximum Degree of Parallelism = 2.

Sequence of Instruction - (Possible)

S1) $I_1\ I_2\ I_3\ I_4\ I_5\ I_6$.

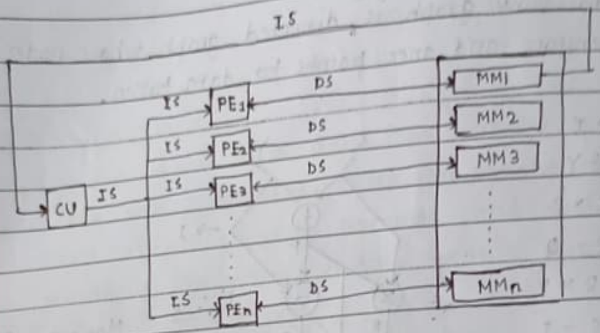
S2) $I_1\ I_3\ I_2\ I_4\ I_5\ I_6$

S3) $I_1\ I_3\ I_2\ I_5\ I_4\ I_6$

S4) $I_1\ I_2\ I_3\ I_5\ I_4\ I_6$.

Vector Processing (Practice for Exam).

Under Flynn's classification, vector Processors can be categorized as SIMD (Single Instruction Multiple Data Stream) computers.



Schematic Diagram of SIMD Architecture.

There are four types of vector instruction -

i) f1 : V → V - These type of instructions take a vector as an input and give a vector as an output.

Eg - $A = \{2, 4, 5, 6\}$ ↳ Input VSQRT $= \{\sqrt{2}, \sqrt{4}, \sqrt{5}, \sqrt{6}\}$ ↳ output.

ii) f2 : V → S - These type of instructions take a vector as an input and gives a scalar as an output.

Eg - $A = \{2, 3, 5, 6\}$ ↳ input VSUMS $= 2+3+5+6 = 16$ ↳ output.

iii) f3 : V×V → V - These type of instructions take two vectors as input and give a vector output.

Eg - $A = \{1, 2, 3, 4\}$
$B = \{8, 7, 6, 5\}$ ↳ inputs VSUMV (A, B) $= \{9, 9, 9, 9\}$. ↳ output

iv). f4 : V·S → V - These type of instructions take a vector and a scalar as an input and produce a vector output.

Eg - A = { 5, 6, 7, ~~9~~ 9} → Vector Input. VSUMVS (A, B) = {10, 11, 12, 14}
B = 5 → Scalar Iput. ↳ vector output.

⊛ Architectural Classifications of Vector Processor

• Memory to Memory Architecture - Instruction is fetched from memory and stores data in memory. Eg - TIASC, Cyber - 200, CDC - 6600.

• Register to Register Architecture - Instruction is fetched from register and stores data in register. Eg - Cray - I, Fujitsu - VP - 200.

⊛ Special Types of Vector Instructions

① Boolean Instruction ② Compress Instruction ③ Merge Instruction

.- Eg of Compress Instruction - A = { 10, 15, 17, 12} → vector i/p
C = { 1, 0, 1, 0} → masking vector.
O = { 10, 17} → compressed Vector.

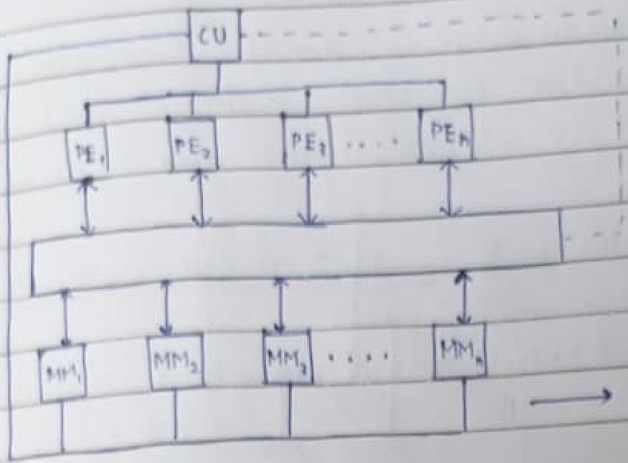
- Eg of Merge Instruction. - A = {11, 16, 18, 20} → vector i/p s.
B = {15, 17, 19, 22}
C = { 1, 0, 0, 1} → masking vector.
M = { 11, 17, 19, 20} → merged vector.

⊛ Types of Vector Operations/Processing

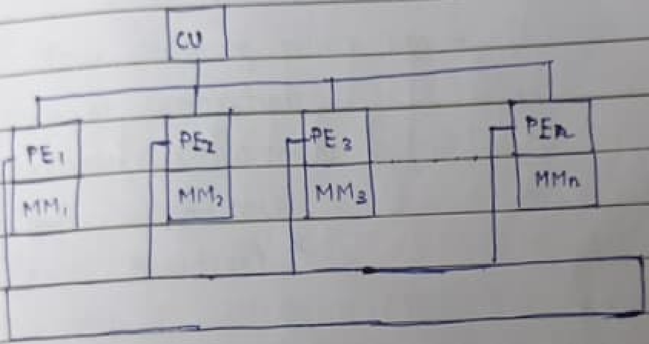
• Horizontal Processing.- In case of Horizontal Processing, elements in the vector are processed from left to right.

- Vertical Processing - In case of Vertical Processing, elements in the vector are processed from top to down.

- Vector Looping - In case of vector looping, elements are processed from both left to right and top to bottom.



→ Vector Processors.



→ Array Processor.

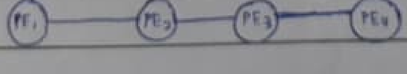
Centralized Procedure

Decentralized Procedure

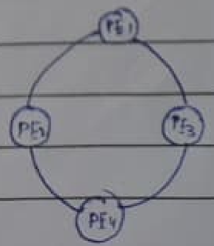
Switching Methodology → Circuit Switching, → Packet Switching.

Network Topology → Static → 1D, 2D, 3D; → Dynamic.

① 1D - Bus Topology -



② 2D - a) Ring Topology -



A single stage network is a switching network with n input selectors (IS) and n output selectors (OS). Each IS is a 1 to D Multiplexer and each OS is an m : 1 multiplexer where $1 \leq d \leq n$ and $1 \leq m \leq n$.

Multistage Network Topology - Many Stages of Interconnected switches forms a multistage SIMD network.

Multistage networks are described by the following features -
i) switch Box.
ii) Network Topology
iii) Control Structure.

Four states of switch box -
- Straight Connection.
- Exchange Connection.
- Upper Broadcast.
- Lower Broadcast.

A multistage network is capable of connecting an arbitrary i/p terminal to an arbitrary o/p terminal.

Multistage Network can be $\longrightarrow$ 1 sided
$\longrightarrow$ 2 sided.

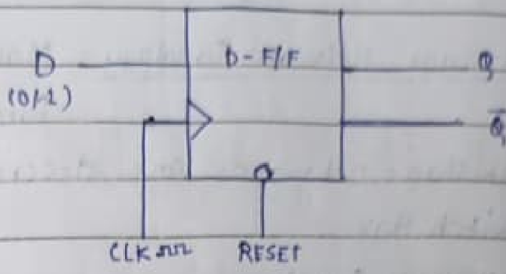
The 2 sided multistage network which usually have an i/p side and an o/p side can be divided into 3 classes -

① Blocking.
② Rearranging.
③ Non - Blocking.

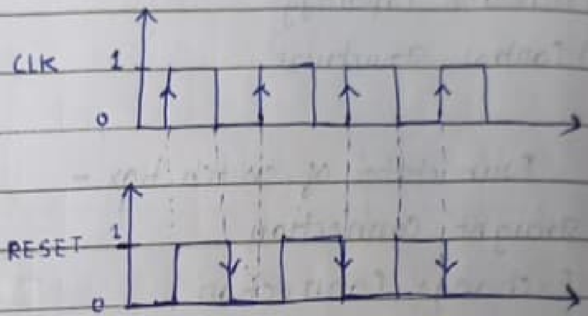
Flip - Flop

Asynchronous Active Low

```
module dff (input d, input rstn, input clk, output reg
always @ (posedge clk  or negedge rstn) .
if (! rstn) .
  q <= 0 ;
else
  q <= d ;
endmodule .
```



Test Bench



```
module tb_dff
reg clk ;
reg d ;
reg rstn ,
reg [2:0] delay ;
  dff uut (.d(d), .rstn(rstn), clk (clk), .q(q)) ;
always begin #10; clk = ~ clk ; end
initial begin
   clk <= 0 ;
   d <= 0 ;
   rstn <= 0 ;
   #15 ; d <= 10 ;
   #10 ; rstn <= 1 ;
for (int i = 0 ; i < 5 ; i = i + 1) begin
   delay = $random ;
   # delay ; d <= i ;
end  end
endmodule .
```

Synchronous Active Low Reset

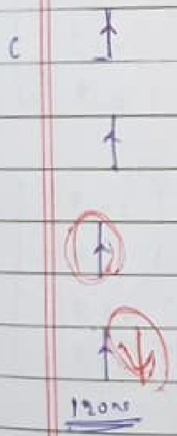
```
module dff (input d, input rstn, input clk, output reg q);
always @ (posedge clk).
  if (!rstn).
    q <= 0;
  else
    q <= d;
endmodule.
```

SR Flip Flop.

!b'+1

Data Driven computing.- Data flow computers are based on the concept of Data Driven non-Von Neumann architecture.

if (!rst).



10 count = 0
11 count++.
00 = 0
01 ____ =

(Routing Info) ILLIAC - IV (PE Communication Architecture)

General Routing Function

$R_{+1}(i) = (i+1) \bmod N$

$R_{-1}(i) = (i-1) \bmod N$

$R_{+r}(i) = (i+r) \bmod N$

$R_{-r}(i) = (i-r) \bmod N$

$N = PE$

$r = \sqrt{n}$

if,

$N = 64$, $r = \sqrt{64} = 8$

and if,

$N = 16$, $r = \sqrt{16} = 4$

1, 15, 4, 12

$R_{+1}(0) = (0+1) \bmod 16$

$R_{-1}(0) = (0-1) \bmod 16$

$R_{+4}(0) = (0+4) \bmod 16$

$R_{-4}(0) = (0-4) \bmod 16$

$R_{+1}(1) = (1+1) \bmod 16 = 2$

$R_{-1}(1) = (1-1) \bmod 16 = 0$

$R_{+4}(1) = (1+4) \bmod 16 = 5$

$R_{-4}(1) = (1-4) \bmod 16 = 13$

$R_{+1}(2) = (2+1) \bmod 16 = 3$

$R_{-1}(2) = (2-1) \bmod 16 = 1$

$R_{+4}(2) = (2+4) \bmod 16 = 6$

$R_{-4}(2) = (2-4) \bmod 16 = 14$

$R_{+1}(3) = (3+1) \bmod 16 = 4$

$R_{-1}(3) = (3-1) \bmod 16 = 2$

$R_{+4}(3) = (3+4) \bmod 16 = 7$

$R_{-4}(3) = (3-4) \bmod 16 = 15$

$R_{+1}(4) = (4+1) \bmod 16 = 5$

$R_{-1}(4) = (4-1) \bmod 16 = 3$

$R_{+4}(4) = (4+4) \bmod 16 = 8$

$R_{-4}(4) = (4-4) \bmod 16 = 0$

$R_{+1}(5) = (5+1) \bmod 16 = 6$

$R_{-1}(5) = (5-1) \bmod 16 = 4$

$R_{+4}(5) = (5+4) \bmod 16 = 9$

$R_{-4}(5) = (5-9) \bmod 16 = 1$

R+1 (6) = (6+1) Mod 16 = 7
R-1 (6) = (6-1) Mod 16 = 5
R+4 (6) = (6+4) Mod 16 = 10
R-4 (6) = (6-4) Mod 16 = 2

R+1 (7) = (7+1) Mod 16 = 8
R-1 (7) = (7-1) Mod 16 = 6
R+4 (7) = (7+4) Mod 16 = 11
R-4 (7) = (7-4) Mod 16 = 3

R+1 (8) = (8+1) Mod 16 = 9
R-1 (8) = (8-1) Mod 16 = 7
R+4 (8) = (8+4) Mod 16 = 12
R-4 (8) = (8-4) Mod 16 = 4

R+1 (9) = (9+1) Mod 16 = 10
R-1 (9) = (9-1) Mod 16 = 8
R+4 (9) = (9+4) Mod 16 = 13
R-4 (9) = (9-4) Mod 16 = 5

R+1 (10) = (10+1) Mod 16 = 11
R-1 (10) = (10-1) Mod 16 = 9
R+4 (10) = (10+4) Mod 16 = 14
R-4 (10) = (10-4) Mod 16 = 6

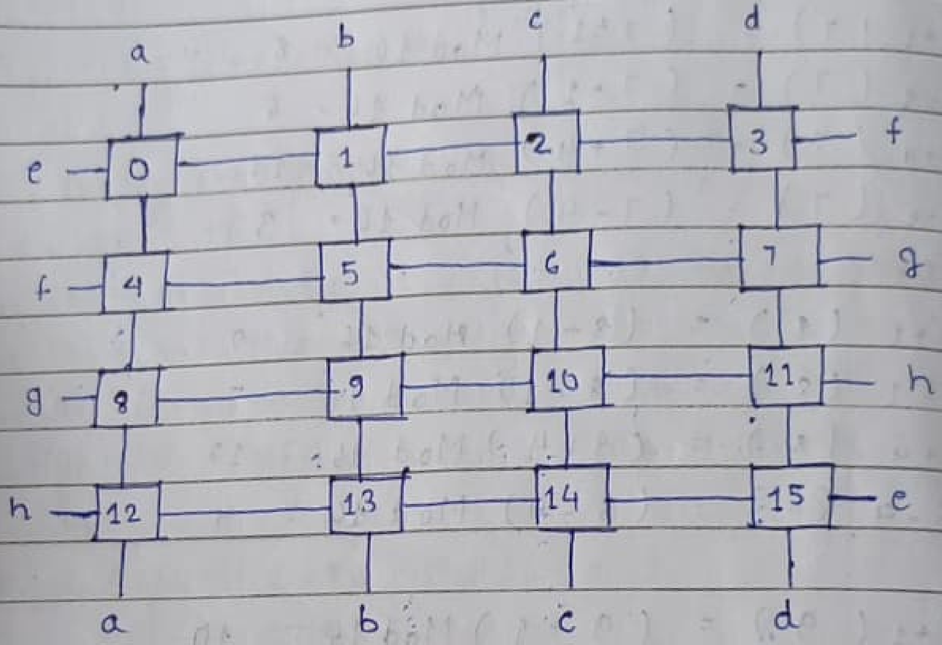
R+1 (11) = (11+1) Mod 16 = 12
R-1 (11) = (11-1) Mod 16 = 10
R+4 (11) = (11+4) Mod 16 = 15
R-4 (11) = (11-4) Mod 16 = 7

R+1 (12) = 13
R-1 (12) = 11
R+4 (12) = 0
R-4 (12) = 8

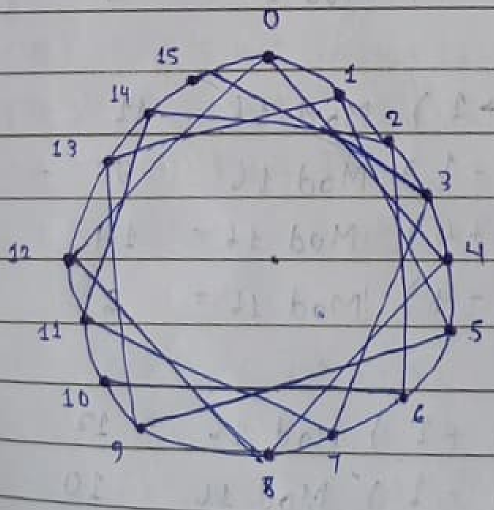
R+1 (13) = 14
R-1 (13) = 12
R+4 (13) = 1
R-4 (13) = 9

$$\begin{cases} R+1\ (14) = 15 \\ R-1\ (14) = 13 \\ R+4\ (14) = 2 \\ R-4\ (14) = 10 \end{cases}$$

$$\begin{cases} R+1\ (15) = 0 \\ R-1\ (15) = 14 \\ R+4\ (15) = 3 \\ R-4\ (15) = 11 \end{cases}$$



Ring Topoly From Above Diagram.



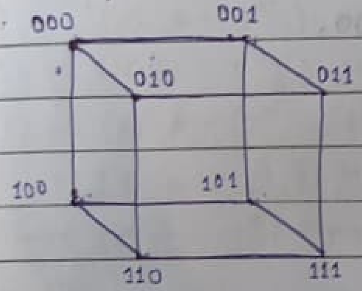
Inter PE Communication

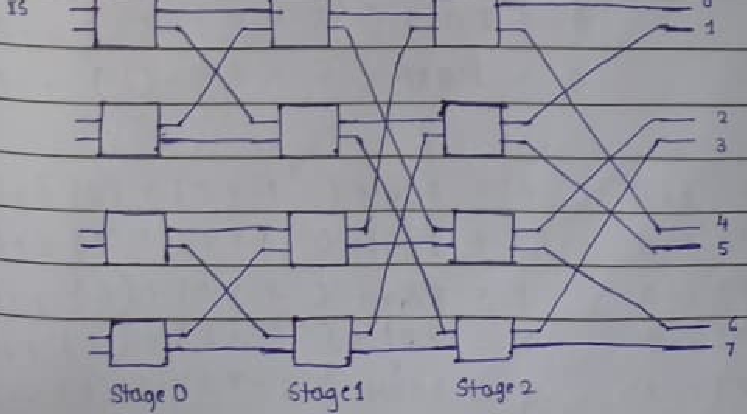
Bits used to rep. PE.

let n = 8. $\boxed{n = \log_2 N}$ $(i(a_{n-1} \cdots \bar{a}_i\, a_{i-1} \cdots a_1 a_0)$.
$\downarrow$ $\hookrightarrow \{0 \leq i \leq n-1\}$.
3 bits to represent.

Connected PEs.

000 ⟶ [001, 010, 100]
001 ⟶ [000, 011, 101]
010 ⟶ [011, 000, 110]
011 ⟶ [010, 001, 111]
100 ⟶ [101, 110, 000]
101 ⟶ [100, 111, 011]
110 ⟶ [111, 100, 010]
111 ⟶ [110, 101, 011]





Cube Network.

$(a_{n-1}, \dots \bar{a}_i, a_{i-1} \dots a_1 a_0)$.

OS $(t_{n-1} \dots t_i t_{i-1} \dots t_1 t_0)$ $\{0 \leq i \leq n-1$.

IS- $(t_{n-1} \dots \bar{t}_i t_{i-1} \dots t_1 t_0)$ $\{0 \leq i \leq n-1$.

Q) $N = 16$.

$n = \log_2 16 = 4$. (Bits).

<u>Connected PEs.</u>

0 0 0 0 → [0001, 0010, 0100, 1000]

0 0 0 1 → [0000, 0011, 0101, 1001]

0 0 1 0 → [0011, 0000, 0110, 1010]

0 0 1 1 → [0010, 0001, 0111, 1011]

0 1 0 0 → [0101, 0110, 0000, 1100].

0 1 0 1 → [0100,

0 1 1 0 → [

0 1 1 1 → [

1 0 0 0 → [

1 0 0 1 →

1 0 1 0 →

1 0 1 1 →

1 1 0 0 →

1 1 0 1 →

1 1 1 0 →

1 1 1 1 →

Barrel shifter.

$B_{+i}(j) = (j + 2^i)$ MOD N.

$B_{-i}(j) = (j - 2^i)$ MOD N.

$0 \le i \le n-1$

$0 \le j \le N-1$.

$n = \log_2 N$

$\#$ PE $= 8$

$N = 8$.

$n = \log_2 8 = 3$.

$B_{+1}(0) = (0 + 2^{1})$ MOD 8 $= 2$ MOD 8 $= 2$

$B_{-1}(0) = (0 - 2^{1})$ MOD 8 $= 2$ MOD 8 $= 2$

$B_{+2}(0) = (0 + 2^2)$ MOD 8 $= 4$ MOD 8 $= 4$

$B_{-2}(0) = (0 - 2^2)$ MOD 8 $= 4$ MOD 8 $= 4$

$B_{+0}(0) = (0 + 2^0)$ MOD 8 $= 1$ MOD 8 $= 1$

NB × ~~$B_{-0}(0) = (0 - 2^0)$ MOD 8 $= 1$ MOD 8 $= 1$~~

$B_{+0}(1) = (1 + 2^0)$ MOD 8 $= 2$ MOD 8 $= 2$

$B_{+1}(1) = (1 + 2^1)$ MOD 8 $= 3$ MOD 8 $= 3$

$B_{-1}(1) = (1 - 2^1)$ MOD 8 $= 7$ MOD 8 $= 7$

$B_{+2}(1) = (1 + 2^2)$ MOD 8 $= 5$ MOD 8 $= 5$

$B_{-2}(1) = (1 - 2^2)$ MOD 8 $= 5$

$B_{+0}(2) = (2 + 2^0)$ MOD 8 $= 3$

$B_{+1}(2) = (2 + 2^1)$ MOD 8 $= 4$

$B_{-1}(2) = (2 - 2^1)$ MOD 8 $= 0$

$B_{+2}(2) = (2 + 2^2)$ MOD 8 $= 6$

$B_{-2}(2) = (2 - 2^2)$ MOD 8 $= 6$

$B_{+0}(3) = (3 + 2^0)$ MOD 8 $= 4$ $B_{+0}(4) = 5$

$B_{+1}(3) = (3 + 2^1)$ MOD 8 $= 5$ $B_{+1}(4) = 6$

$B_{-1}(3) = (3 - 2^1)$ MOD 8 $= 1$ $B_{-1}(4) = 2$

$1-4$ $B_{+2}(3) = (3 + 2^2)$ MOD 8 $= 7$ $B_{+2}(4) = 0$

$8-3=5$ $B_{-2}(3) = (3 - 2^2)$ MOD 8 $= 7$ $B_{-2}(4) = 0$

$B_{+0}(5) = 6$	$B_{+0}(6) = 7$	$B_{+0}(7) = 7$
$B_{+1}(5) = 7$	$B_{+1}(6) = 0$	$B_{+1}(7) = 1$
$B_{-1}(5) = 3$	$B_{-1}(6) = 4$	$B_{-1}(7) = 5$
$B_{+2}(5) = 1$	$B_{+2}(6) = 2$	$B_{+2}(7) = 3$
$B_{-2}(5) = 1$	$B_{-2}(6) = 2$	$B_{-2}(7) = 3$

Barrel shift network are also known as $\pm 2^i$ network (PM2I). This type of network is based on the following routing functions.

Each PE in this network is directly connected to $2(n-1)$ PEs. Therefore, the connectivity in this network is increased from ILLIAC network by having $(2n-5)\,2^{n-1}$ more direct links. For example, the number of PEs $(N) = 16$, $n = 4$ and $r = 4$ the ILLIAC network has 32 direct links and barrel shift network has 56 direct links. The two networks are identical only when $n = 2$ or $N = 5$.

B, the minimum number of recirculation is upper bounded by $B \leq \frac{\log_2 N}{2}$. Speed up of Barrel shift network over ILLIAC network S_N is given by -

$$S_N = \frac{\sqrt{N}}{\frac{\log_2 N}{2}} = \frac{2^k - 1}{k}, \text{ where } N = 2^k.$$

Shuffle - Exchange Network.

$PE_A = a_{n-1} \cdots a_i \cdots a_1 a_0$.

$S(PE_A) = a_{n-2} \cdots a_i \cdots a_1 a_0 a_{n-1}$.

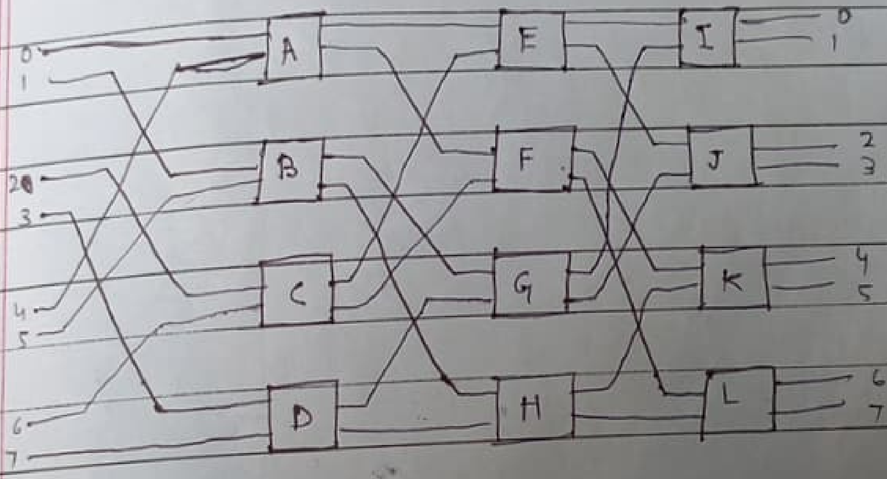
$E(PE_A) = a_{n-1} \cdots a_i \cdots \overline{a_0}$

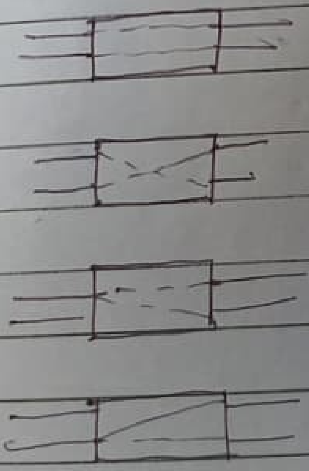
$n = \log_2 N$

$N = \# PE$.

$N = 8$

$E(PE_A) = C_0 (PE_A)$.





(0, 5) (1, 3).

RAM

16×8

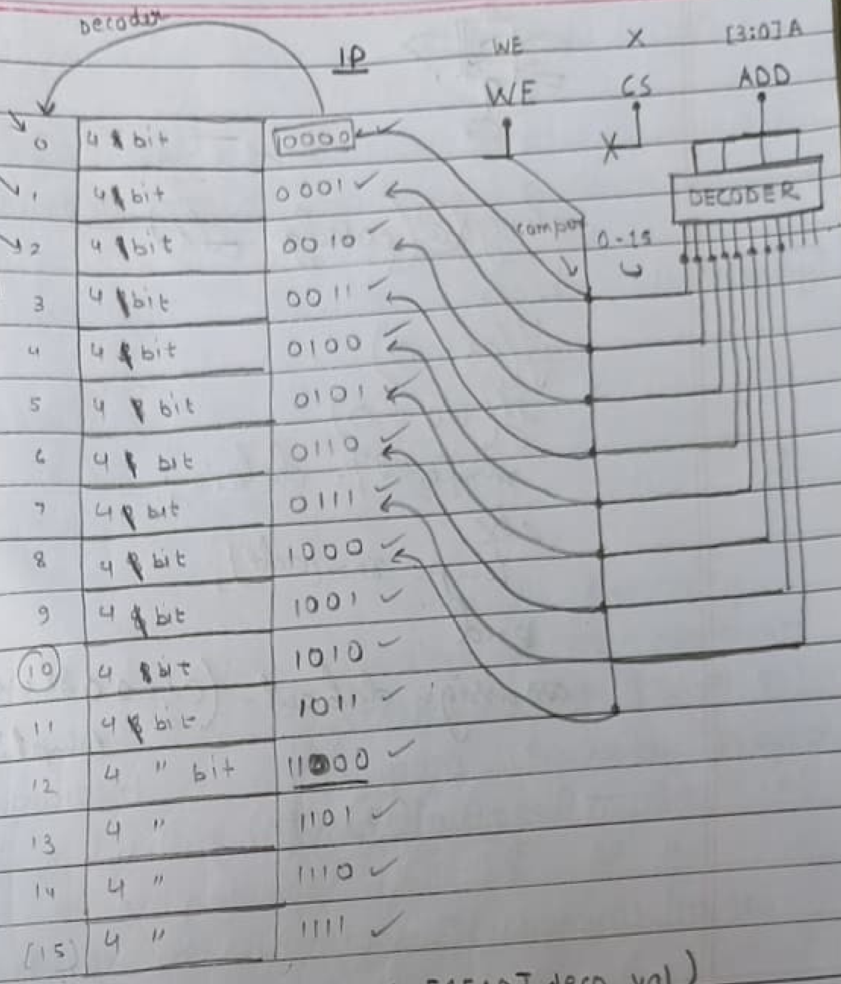
reg [3:0] abc [15:0]

reg [3:0].

abc

abc [1100]

1010 10



		IP
0	4 bit	0000 ✓
1	4 bit	0001 ✓
2	4 bit	0010 ✓
3	4 bit	0011 ✓
4	4 bit	0100 ✓
5	4 bit	0101 ✓
6	4 bit	0110 ✓
7	4 bit	0111 ✓
8	4 bit	1000 ✓
9	4 bit	1001 ✓
10	4 bit	1010 ✓
11	4 bit	1011 ✓
12	4 " bit	1100 ✓
13	4 "	1101 ✓
14	4 "	1110 ✓
15	4 "	1111 ✓

```
module decoder ( input address, output [15:0] deco_val)
                       [3:0]

assign deco_val [0] = ~add[0] & ~add [1] & ~add [2] & ~add [3];
  "         "   [1] = ~ add [0] & ~add [1]& ~add [2] & add [3];
  "         "   [2] =
                [3] =
                [4] =
                [5] =
                [6] =
                [7] =
                [8] =
                [9] =
                [10] =
                [11] =
                [12] =
                [13] =
                [14] =
```

Instruction Level Parallelism (ILP)

1) Architectural Technique that allows the overlap of individual machine operation (ADD, Multiplication, Load, Store, ...)

2) Multiple operations will execute in parallel (simultaneously).

Goal - To speed up the Execution.

★ ILP vs Parallel Processing. (Diff).

ILP	Parallel Processing
• Overlap Individual machine operations so that they execute in parallel.	• Having seperate processors getting seperate chunk of program. (Processors programmed to do so)
• Transparent to the user.	• Non - Transparent to the user.
• Objective is to speed up execution	• Objective is to speed up and quality up.

★ ILP Challenges.

• In order to achieve parallelism, we should not have dependencies among instructions which are executing in parallel.

• Hardware Terminologies - RAW, WAR, WAW
Read ↙ ↘ Write.

• Software Terminologies Data Dependencies - ~~Types of~~ Dependencies

Types of Dependencies

(1) Name Dependencies — (i) Output Dependency.
(ii) Anti Dependency

(2) Data True Dependencies —

(3) Control Dependencies —

(4) Resource Dependencies —

Explanations

(1) NAME DEPENDENCY

(a) Output Dependency - When instruction I and J write the same register or memory location, the ordering must be preserved to leave the correct value in the register or memory location.

(b) Anti Dependency - When instruction J writes a register or memory location that instruction I reads.

Eg - I : add r6, r5, r4;
J :

(2) Data True Dependency:- An instruction J is data dependent of instruction I if either of the following hold:-

(i) Instruction I produce a result that may be used by instruction J or

(ii) Instruction J is data dependent on instruction K and instruction K is data dependent on instruction I.

```
Eg- LOOP : LD F0, 0(R1);
           ADD F4, F0, F2.
           SD  F4, 0(R1).
           SUB R1, R1, -8
           BNE  R1, R2, LOOP
            ↓
     Branch Not Equal to.
```

(3) Control Dependency - Control Dependency determines the ordering of an instruction w.r.t a branch instruction so that the instruction I is ~~e~~executed in correct program order.

Two constraints imposed by control dependency -

(i) An instruction that is control dependent on a branch cannot be moved before the branch.

(ii) An instruction that is not control dependent on a branch cannot be moved after the branch.

(4) Resource Dependency :- An instruction is resource dependent on a previously issued instruction if it requires a hardware resource which is still being used by a previously issued instruction.

ILP Architectures.

- Computer Architecture is a contract (instruction format and interpretation of bits that constitute an instruction) b/w the class of programs that are written for the architecture and the set of processors for the implementation of that architecture.

- In ILP architecture information embedded in the program pertaining to the available parallelism b/w instructions and operations in the program.

ILP Architecture classification.

1) Sequential Architectures - The program is not expected to convey any explicit information regarding the parallelism (super scalar processors)

2) Dependency Architecture - The program explicitly indicates the dependencies that exist b/w operation (Data Flow Processors).

3) ~~Dat~~ Independence Architecture - The program provides information as to which operations are independent of one another. (VLIW Processor / Architecture)

VLIW - Very Large Instruction Word.

Pipelined (Linear).

R/W					I_1	I_2	
E				I_1	I_2		
OF			I_1	I_2			
D		I_1	I_2				
F	I_1	I_2					

Super scalar.

F	D	OF	E	R/W
I_1				
I_2				
I_3				
I_4				

Q) What is Data Flow Computer? Differentiate b/w Data Flow computers from a Control Flow Computer.

Q) What are the problems with implementation of Data Flow computer.

```
1 1 0 1 0 1 1 0
0 1 0 0 1 1 1 0
        0 1

000 - 1000100 0
001 - 0100100
010 -
011 -
100 -
101 -
110 -
111 -
```