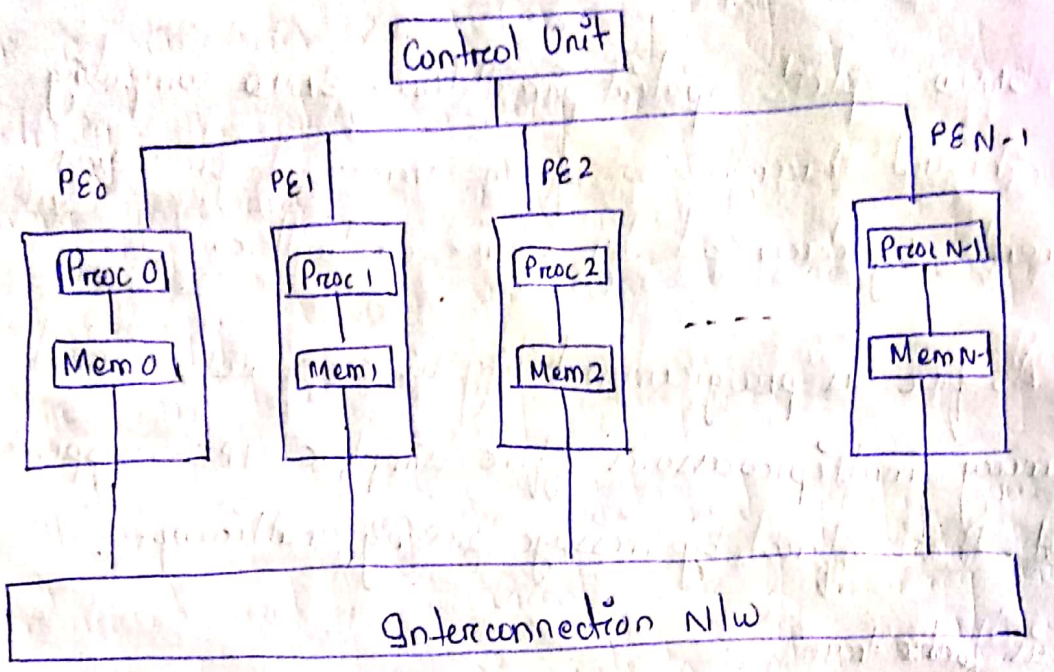
SIMD Supercomputer

8/8/19



(Operational model of SIMD computers)

Proc i : Processor i

Mem i : Memory i

Control unit has direct control on PE.

SIMD M/c model

M/c. $\leftarrow M = (N, C, I, M, R)$ → masking sch.

N : No. of PEs

C : set of instruc. directly executed by CU

I : " " " broadcasted " "

M : set of masking schemes

R : set of data routing func?

- Here, N is no. of PEs in the m/c.

eg: Illiac has 64 PEs,

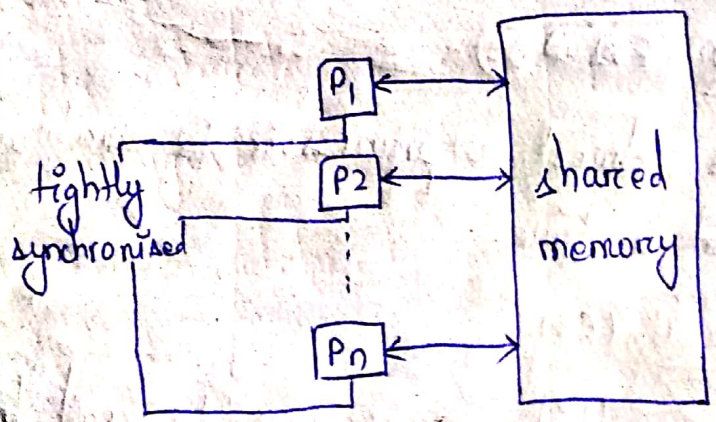
and CM-2 has 65536 PEs.

- C is set of instruc. directly executed by CU including scalare and program flow instructions.
- I is set of instruc. broadcasted by CU. to all PEs for parallel processing. These includes arithmetic, logic, data routing, masking, and other local operations. executed by each active PE over the data within the PE.
- M is the masking schemes where each mask partitions the set of PEs into enabled & disabled subsets.
- R is the set of data routing func. specifying various patterns to be set up in the interconnection N/w for inter PE comm^n.

PRAM Models

↓

~~Proattel~~ Parallel Random Access M/c.



Pi - processors.

(PRAM model of multiprocessor system)

- PRAM Model can be used for parallel algorithm development and for scalability & complexity analysis.
- An n-processors PRAM has a globally addressable memory. The shared memory can be distributed among processors or centralized at one place.
- The PEs operates on a synchronized read memory, Compute and write memory cycle.
- With shared memory, the model must specify how concurrent read & concurrent write of memory are handled. (bez many Pi wants to read or write, ∴, a mechanism must be provided).
- There are 4 memory update options are available :-

i) Exclusive Read (ER)

It allows atmost 1 processor to read from memory loc in each cycle.

ii) Exclusive Write (EW)

It allows atmost 1 processor to write into memory loc at a time.

iii) Concurrent Read (CR)

It allows multiple processors to read from same memory loc.

iii) Concurrent write (CW)

→ It allows multiple processors to write into same memory loc^n.

→ Some policies are required to avoid the write conflicts

Prepare note for below topics:-

a) EREW - PRAM Model
b) CREW
c) ERCW
d) CRCW

} PRAM Variants.

09/08/19

a) EREW

- most restrictive model as it forbids more than one processor from reading or writing same memory cell simultaneously.

b) CREW

- conflicts are avoided by mutual exclusion. Concurrent reads to same m/m loc^n are allowed.

c) ERCW

exclusive read & concurrent write.

d) CRCW:

concurrent read & write are allowed.

In what way, we can define parallelism, is discussed here.

1) Data dependency: (DD)

Ordering relationship blw statements is given by data dependency graph. (DDG)

The nodes of dependency graph represents the prog statements (instruc^n).

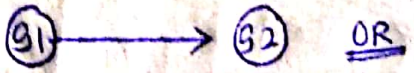
The directed edges with diff labels represents the relation blw diff statements.

The data dependency graph is used to exploit ~~red^n~~ blw parallelism of the prog^n.

5 classes of DD are:

a) Flow dependency:

The statement S_2 is flow dependent on statement S1, if an execution path exist from S1 to



OR $S_1 \rightarrow S_2$

and atleast one o/p of S1 feeds in as i/p to S2.

b) Anti dependence:

cond^n 1

S2 is anti dependent on S1, if S2 follows S1 in program order and if o/p of S2 overlaps the i/p to S1.

cond^n 2

09/08/19

cussed here

by data

prog.

nts the

bleit ~~not~~"

t on

from S1 to S2

llp to S2.

cond^n 1

2 follows S1

overlaps

$S_1 \dashrightarrow S_2$

c) o/p dependence:

$S_1 \text{ o} \longrightarrow S_2$

2 statements are o/p dependent from S_1 to S_2, if the procedure (write) the same o/p ~~value~~ variable.

d) i/o dependence: $S_1 \xrightarrow{i/o} S_2$

Read and write are i/o statements. i/o dependence occurs not bcz same variable is involved, but bcz the same file is referenced by both i/o statements.

e) Unknown dependence:

The dependence b/w 2 statements cannot be determined in some situations. for eg-

→ subscript of a variable is ~~other~~ itself subscript

→ " doesnot contain the loop index variable.

→ A variable appears more than once with subscripts having diff. coefficients of loop variable.

→ Subscript is non-linear in loop index variable.

eg

```
S1: Load  R1, A      // R1 ← M[A] /
S2: ADD   R2, R1     // R2 ← (R1) + (R2) /
S3: MOV   R1, R3     / R1 ← (R3) /
S4: STORE B, R1      / M[B] ← (R1) /
                       ↑ contents of R1
```

→ S_1 and S_2 is flow dependent

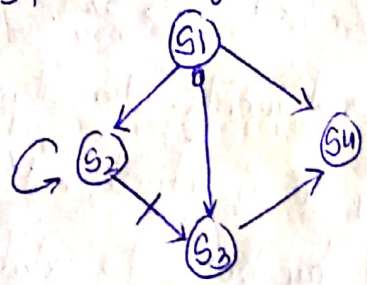
→ S_2 & S_3 are anti " bcz of registers potential ~~content~~ conflict.

→ S_3 & S_1 is o/p dependence.

→ S_1 & S_4 are flow dependence.

→ S_2 & S_4 is indepen.

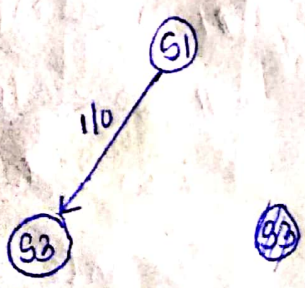
→ S_3 & S_4 is flow dependent.



→ S_2 is flow dependent on itself bcz (R_1) is I/p as well as o/p in same statement.

eg: Code fragment with I/O operation

```
S1 : Read (4), A(I)        / Read array A from file 4 /
S2 : process               / Process data /
S3 : write (4), B(I)       / write array into file 4 /
S4 : close (4)             / close " " /
```



14/8/19

2) Control Dependency (CD)

A situation where order of execution of statements cannot be determined before runtime. (because of some branch statements)

eg IF statement of FORTRAN will not be resolved until runtime.

Diff branches of IF, may introduce or eliminate data dependencies among instruc.

Eg1 Control independent loop

```
DO 10 I=1,N
    A(I) = C(I)
IF (A(I).LT.0)
    A(I) = 1
10 CONTINUE
```

Control dependent loop

```
DO 20 I=1,N
IF (A(I-1).EQ.0)
    A(I) = 0
20 CONTINUE
```

↓

In this, where control dependency exists, parallelism is not exploited.

- Better compilers are needed to exploit more parallelism.

3) Resource Dependency (RD)

It means the conflicts that arise when using shared resources like int units, floating pt units, registers & memory area.

* If conflicting resource is ALU, it is known as ALU dependence.

* If the conflicts involve workplace storages then it is called as storage dependence.

→ The transformation of a sequentially coded program into a parallel executable form is done either manually by programmer using explicit parallelism or by compiler detecting implicit parallelism auto.

However, some prog are inherently sequential in nature. So, they cannot be decomposed into ll branches.

H/w and S/w parallelism

H/w parallelism	S/w
1) It is build into the m/c's architecture & h/w multiplicity. It is known as m/c parallelism.	1) It is exploited by concurrent execution of m/c lang instrn. in a prog as coded by a programmer or as generated by HLL compiler.
2) It is func of cost & performance trade offs	2) It is a fun of algo, programming style & compiler optimization.
3) It displays what is resource utilization.	3) It displays patterns of simultaneously executable operations.
4) It can also indicate peak performance of processor resources.	

a) Grain size or granularity: It is measure of amount of computation involved in the process.

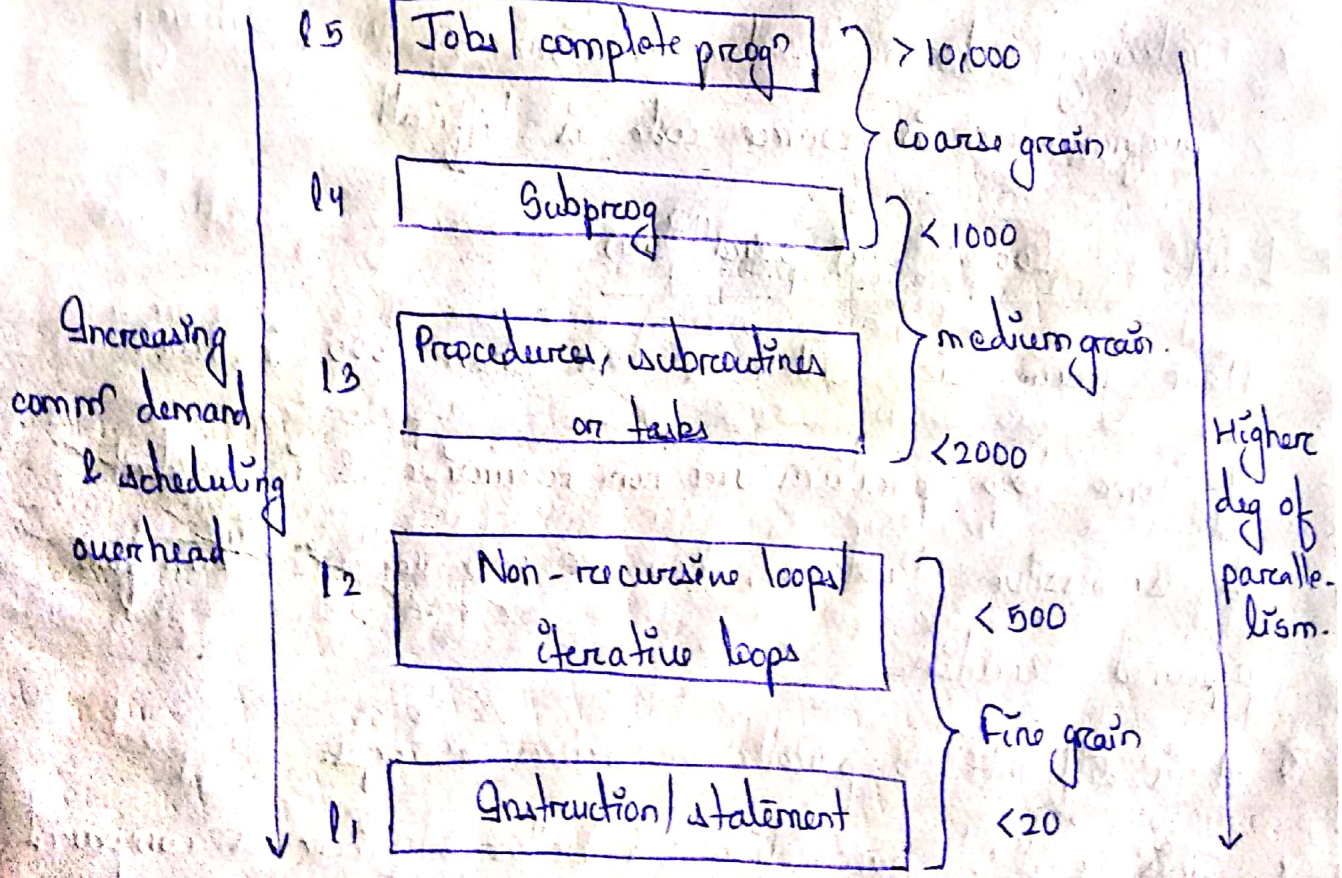
b) Grain: It is basic prog segment that is chosen for ll processing

c) Latency: It is time measure of the comm overhead incurred blw m/c subsys.

eg time required by PE to access the memory is known as memory latency. eg time required by 2 processes to synchronize with each other is called synchronization latency.

Levels of Parallelism

Parallelism has been explored at 5 levels of processing.



developing multicore CPU:-

Level-1 : Instruction level / statement level parallelism

At this level of parallelism, a typical grain (or prog's segment) consists of less than 20 instruc°. So, it is called fine grain parallelism.

* The lower the level, finer will be granularity of slow process.

Adv:

→ At this level there will be huge amount of parallelism available. for which we need optimizing compiler. It is compiler which automatically detects parallelism. It converts source code to a || form which can be understood by the run-time system.

* Detecting instruc° level parallelism by an ordinary programmer in source code is difficult.

Level-2: Loop level parallelism

Here grain size will have less than 500 instruc°. If some loop operations are non-recursive & independent in successive iterations, then we can vectorize them for pipelined execution.

* It is difficult to parallelize recursive loops.
* This level of parallelism is better & finer as compared to level-3 & level-5.
* This type of grain is very suitable for || comp.

Level-3: Procedure level Parallelism

→ At this level, a typical grain corresponds to medium size at task, procedural, subroutine or functional levels.

→ Grain size < 2000 instructions.

→ Detection of parallelism at this level is much more difficult than at fine-grain levels.

→ Programmers must restructure a program at this level. They need compilers.

Level-4: Subprogram level parallelism

→ Typical grain corresponds to subprog or job steps.

→ Grain size is thousands of instruc.

→ Earlier, parallelism at this level has been exploited by algo designers or programmers & not by compilers.

Level-5: Job/Program Level Parallelism

→ At this level, grain corresponds to independent jobs or programs for a II execution.

→ Grain size: from ~~10000~~ million to of instruc in a single prog.

∴ it is called coarse grain level parallelism.

→ To achieve this level of -5 or prog level IIism, we need prog loaders & an OS or space-sharing sys.

→ Time sharing sys explores this level of parallelism only.

Conclusions

1. Fine grain parallelism is exploited at instruc or loop levels. It requires a parallelizing or vectorizing compiler.
2. Medium grain parallelism is exploited at task or job step. It requires compilers as well as significant programm rules.
3. Coarse grain parallelism is exploited at prog level. It requires effect of OS & efficient algo. Message-passing multicomp have been used for medium & coarse-grain computations.

* The finere the grain size, the higher is the potential for parallelism & higher is comm and scheduling overhead.
Fine grain provides a higher degree of parallelism but more will be comm overhead. This may or may not be true with coarse grain computations.
Massive parallelism is often found at fine grain levels like data parallism on SIMD or MIMD comp.

→ Shared variable comm is often used to support fine grain & medium grain computations.

Grain Packing?

→ Introduced by Lewis in 1988 for ll programming.

→ The idea of grain packing is to apply fine grain first in order to achieve a higher degree of parallelism. Then, we combine (pack) multiple fine-grain nodes into a coarse-grain node if it can eliminate unnecessary comm delays or reduce scheduling overhead.

→ Usually, all fine-grain operations within a single coarse-grain node are assigned to same CPU for execution

→ Fine grain partition of prog demands more & IPC than that required in coarse grain partition.

→ Thus, grain packing offers a trade off blw parallelism & comm overhead. Grain packing removes comm delays.

→ Internal delays among fine grain operations within the same coarse-grain node are negligible. This is due to fact that comm delay is contributed mainly by inter PE delays and not by the delays within same processor. Hence, the choice of optimal grain size is must. It is only then we can achieve the shortest schedule for nodes on ll comp.

Eg: Consider following program -

Begin

1) a := 1
2) b := 2
3) c := 3
4) d := 4
5) e := 5
6) f := 6

} grain size = 1 as there is one variable / constant

7) g := a * b
8) h := c * d
9) i := d * e
10) j := e * f
11) k := d * f
12) l := j * k
13) m := 4 * l
14) n := 3 * m
15) o := n * i
16) p := o * h
17) q := p * q

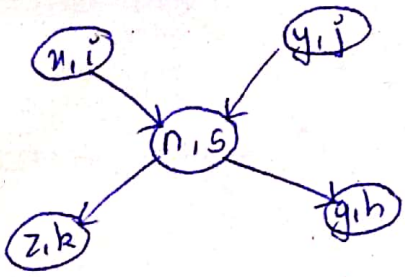
} grain size = 2 as there are 2 variables / constants on RHS

End

Show fine grains prog graph before packing.
Convert it to coarse grain prog graph after packing.

Soln Prog graph shows structure of prog. Each node corresponds to computational unit in prog.

Grain size is measured by no. of basic m/c cycles - both processor & memory cycles that are needed to execute all operations within node.

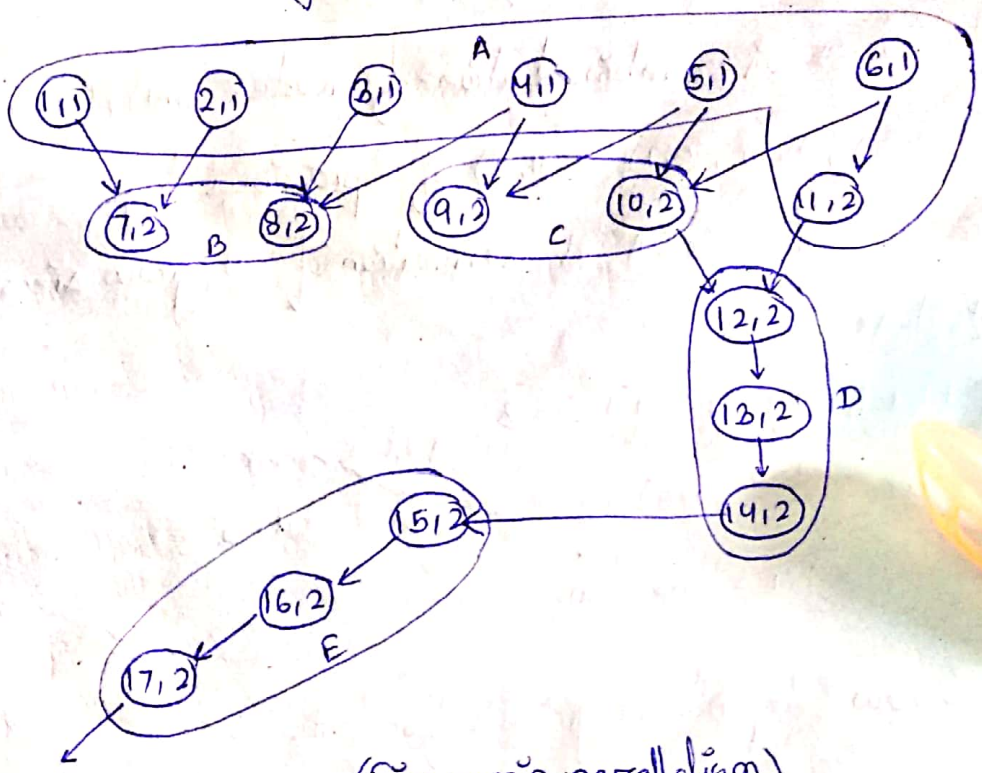


denotes?

(n,s)	(node-no, grain size)
(x,i)	(I/P, delay)
(z,k)	(O/P, delay)
(y,j)	I/P
(g,h)	O/P

So, grain size reflects the no. of computations involved in program segment.

* Fine grain nodes have a smaller grain size & coarse grain nodes have larger grain size.



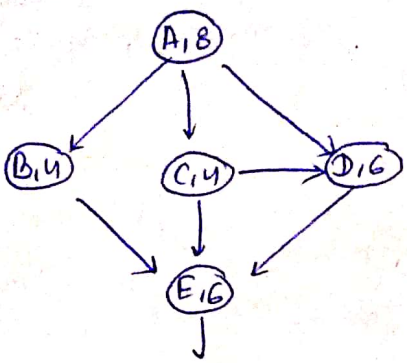
(Fine grain parallelism)

Here,

→ Nodes 1, 2, 3, 4, 5 and 6 are data fetch operations from I to memory that takes one cycle.

→ All other nodes (7 to 17) are CPU operations, each requiring 2 cycles to complete.

→ for developing embedded software



(Coarse grain parallelism)

Here,

→ we do grain packings which forms coarse grain node with larger grain sizes (from 4, to, 8).

→ Node (A,8) is combination of nodes (1,1), (2,1), (3,1), (4,1), (5,1), (6,1) & (1,2) of previous fig.

→ Grain size of node-A is summation of all grain sizes of all these nodes i.e.

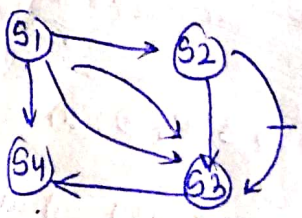
$1+1+1+1+1+1+2 = 8$

So, we get (A,8).

→ Similarly, others are done.

Eg-2 Draw dependence graphs for foll.

a) S1 : A = B+D
S2 : C = A*3
S3 : A = A+C
S4 : E = A/2

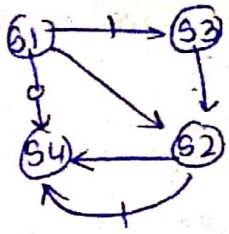


b) S1 : X = SIN(Y)

S2 : Z = X + W

S3 : Y = -2.5 * W

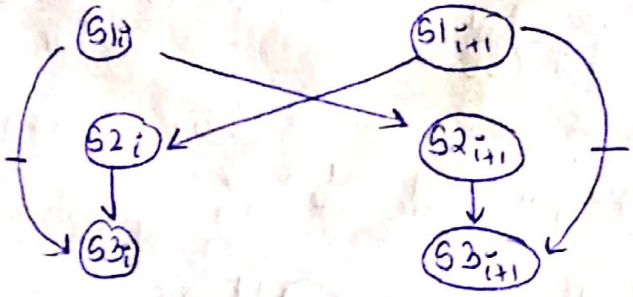
S4 : X = cos(Z)



Eg 3 Considere FORTRAN Code:

```
DO 10 I = 1, N
S1: A(I+1) = B(I-1) + C(I)
S2: B(I) = A(I) X K
S3: C(I) = B(I) - I
10 Continue.
```

Determine data dependencies of this.



Eg 4

S1: load R1, 1024

S2: load R2, M(10)

S3: Add R1, R2

S4: Store M(1024), R1

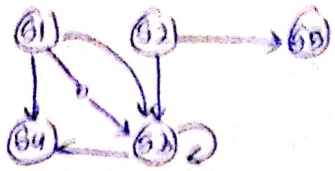
S5: Store M(IR2)), 1024.

(Ri) - content of Ri

Memory (10) contains 64 initially.

→ for developing embedded software

a) Draw dependence graph to show all dependencies



b) Are there any resource dependencies if only on copy of each functional unit is available in CPU?

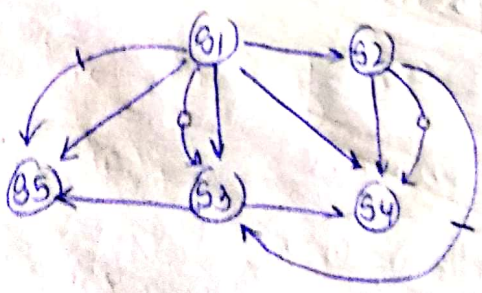
→ Yes, there are storage dependencies blw instruc pairs - S2, S5) & S4, S5).

→ There is a resource dependence blw S1 & S2 on load unit and another blw S4 & S5 on store unit.

→ There is an ALU dependence blw S3 & S4 & a storage dependence blw S1 & S5.

c) Repeat above for full code:

S1 : load R1, M(100)
S2 : Move R2, R1
S3 : Inc R1
S4 : Add R2, R1
S5 : Store M(100), R1



Module - 2 16/08/19

Properties of N/w

i) N/w size : No of nodes present in the graph.

ii) Graph : A graph of a n/w consists of nodes that represent switching pts and edges that represents comm^n links.

iii) Node degree (d) : The no. of edges or links that are incident on a node is called node degree. (node deg = in deg + out deg)

iv) Diameter (D) :

→ d reflects no of i/o ports required for node.

→ d shld be kept as constant or as small as possible to reduce cost & to achieve modularity.

v) Diameter (D) : The diameter of a N/w is the largest dist blw 2 nodes.

→ A low diameter is better bcz it puts lower bound on complexity of N algo. requiring comm^n blw arbitrary pair of nodes.

v) Bisection width (b) : It is defined as the min^n no of edges that must be removed in order to divide the n/w into 2 half.

→ The higher bisection width is better bcz in algo requiring large amount of data movement, the size of data set divided by "b" puts a lower bound on the

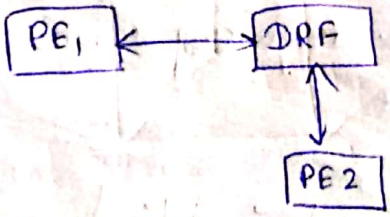
complexity of ll algo.

(ii) No. of edges per node : A node represents a PE and an edge represents comm^n path blw 2 PE. So, it represents the no. of comm^n paths per processor.

(iii) Max^n edge(path) length : It is based if the nodes (PE) and edges (path) represent a 3D space Nlw so that this max^n edge length is a constant, independent of Nlw size.

Routing

A data routing Nlw is used for inter-PE data comm^n. There are data routing func^n (DRF) that are implemented on an inter-PE routing Nlw.



(Role of DRF)

Some of DRFs:

1. Permutations
2. Perfect Shuffle & exchange
3. Hypercube routing func^
4. Broadcast & multicast

1) Permutation

If we have small t objects, then there are t! permutations by which t obj can be reordered.

We use a cycle notation to specify a permutation func?.

The permutation

eg $\pi = (a,b,c).(d,e)$ stands for

$a \to b$, $b \to c$, $c \to a$, $d \to e$, $e \to d$ in a circular fashion.

Also note that the cycle (a,b,c) has a period of 3 and the other (d,e) has period of 2.

So, total no. of permutation, π has a period of

$3 \times 2 = 6$

i.e. π: $3 \times 2 = 6$ period.

Now, if we apply permutation 6 times, then this many times data routing can be performed. and Identity mapping, $I = (a), (b), (c), (d), (e)$ is obtained.

Permutations can be implemented by using a crossbar switch, a multistage nlw, a shifting or broadcast operations.

→ permutation capability of a nlw is often used to indicate data routing capability.

When n is large, permutation speed often dominates the performance of data routing nlw.

2) Perfect shuffle and exchange

In 1971, Harold Stone suggested that if we are given a binary string as

$a_n a_{n-1} \cdots a_2 a_1$ then

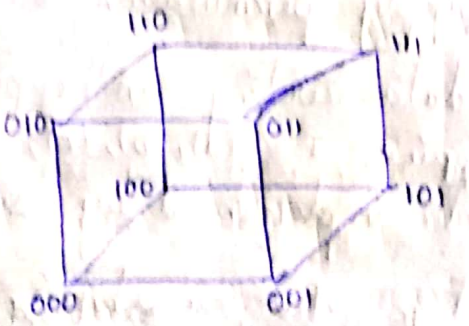
$\text{shuffle}(a_n a_{n-1} \cdots a_2 a_1) = (a_{n-1} a_{n-2} \cdots a_2 a_1 a_n)$

eg shuffle (001) = 010

shuffle (101) = 011

* The shuffle transform is a left shifting of binary strings by 1 bit

3) Hypercube routing function



(3-D binary cube)

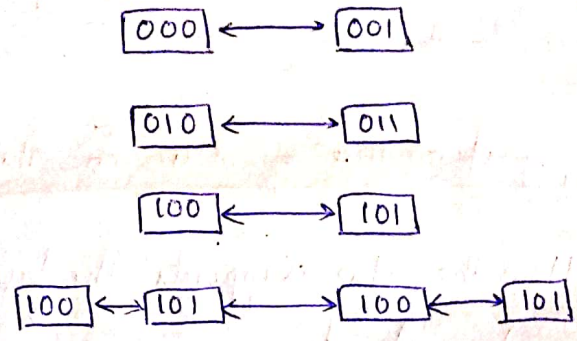
We consider a 3D binary cube as shown in fig

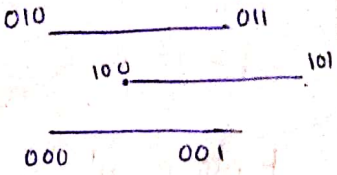
Say node address is 3 bit long (c_2, c_1, c_0)

Since, there are 3 bits in node add, we can have 3 routing func^n. Ex:

These 3 routing func^n will be on the basis of the routing of 3 bits i.e. either c_0 or c_1 or c_2.

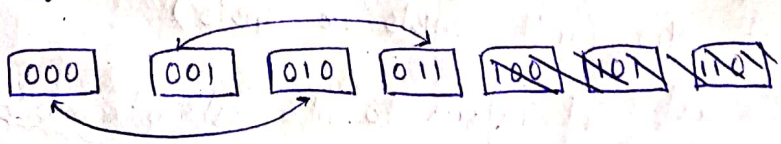
Case-I : Routing on c_0 (LSB)





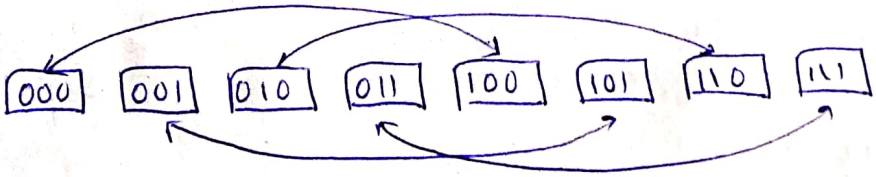
Case-II : Routing on c_1 (middle bit) 20/8/19

This type of routing is possible iff the middle bit i.e. c_1 differ on their node address.



Case-III : Routing of c_2 (MSB)

This type of routing is possible iff MSB bit i.e. c_2 differs on their node add.



* In general, an n-dimensional hypercube will have n routing func^n, defined by each bit in n-bit add.

These data exchange can be used in routing msg in hypercube multicomp.

4) Broadcast and Multicast

Broadcast means from one node data may be send simultaneously to all othere nodes in the n/w.

Multicast means from many nodes data may be sent to all othere nodes in the n/w.

Factors affecting performance of interconn? N/w

→ max^ Functionality: How the n/w supports the func? like data routing, interrupts etc.

→ min^ N/w Latency: it means worst case time delay for a unit message to be transferred through the n/w.

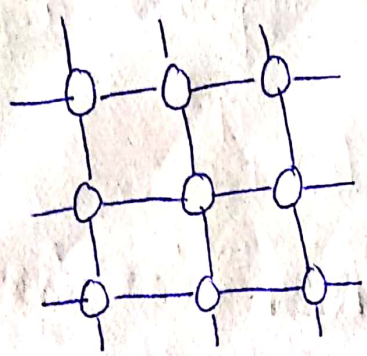
→ max^ Bandwidth: it means the max? data transfer rate in a ~~unit~~ n/w. (unit MB/s or GB/s)

→ min^ H/w cost: it refers to h/w cost like wires, switches, connectors etc.

→ max^ Scalability: it means ability of n/w to expand with scalable performance with increasing m/c resources.

Mesh N/w

A 2D (3*3) mesh n/w is shown below.

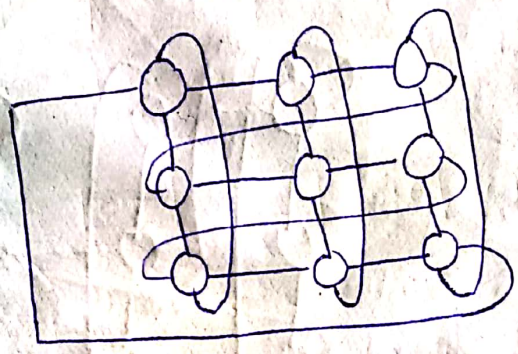


The Illiac-IV, DAP, 600 multiprocessors, CM-2 & Intel Paragon use diff types of mesh archi.

In general, a k-dim mesh has n^k nodes

n - # nodes in each direction with each node having an internal node degree of 2k.

Dally's J-m/c is 3D-mesh.



(Illiac IV mesh)

Properties:

node deg, d = 4

diameter, D = 2(r-1) where (r,r) is mesh size & r = $\sqrt{n}$

no. of links, l = 2n - 2r

Bisection width, b = r

→ for developing embedded softwa...

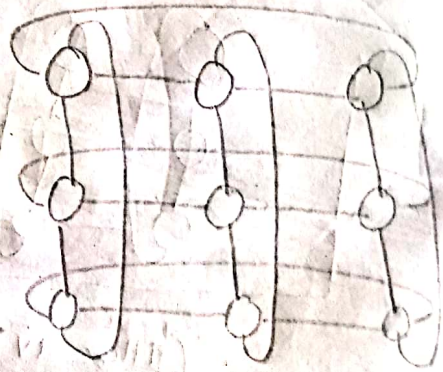
For Illiac mesh, properties are:

d = 4
D = r - 1
l = 2n
b = 2r

Torus topology

- variant of mesh archi.
- It combines ring & mesh & extends to higher dimensions.
- has ring connec along each row & along each column of array.
- a (n * n) binary torus has node deg of 4.
- symmetric topology.
- Properties:

d = 4
D = $2\lfloor \frac{r}{2} \rfloor$
l = 2n
b = r (r = √n)



torus of size (3 * 3)

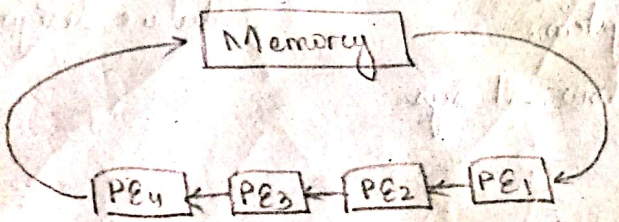
Systolic Arrays

- electronic ckt.
- It is so called bcz data from memory unit is sent in rhythmic fashion though an array similar to way in which human heart regularly pumps bl... via arteries.

- data proces...
- transform...

primary ...
flow of ...
data ca...
Each c...
process...
comm...
indivi...
cells ...
comm...

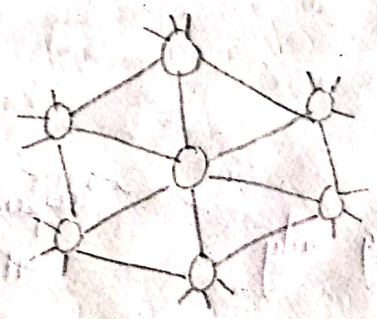
- data processed by PEs
- transformed I/P is returned to memory.



(A ~~symbolic~~ SA archi.)

Primary characteris of SA are :-

a) flow of data is rhythmic & regular.
b) data can be bidirectional
c) Each cell or PE performs identical operation.
d) processing time of all cells are equal
e) comm° blw cells is serial & periodic
f) individual cells are connected to nearest neighbours.
g) cells except those at boundary of array do not communicate with outside world.



(SA topology)

→ for developing embedded

Adv of SAS

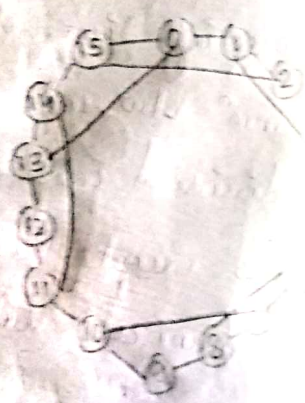
- deg of pipelining is high
- simple I/O system
- high speed & low cost sys
- synchronised multiprocessor
- modular design

Disadv of SAS

- signal delays result in global synchronization limit
- higher BW is required for inter PE comm.
- poor run time fault tolerance.

Chordal Ring

- If we incr node degree from 2 to 3 or 4, we get 2 chordal rings.
- more the links are added, higher is node degree & shorter is n/w diameter.



Barrel Shifter

- It is obtained from ring only by adding extra links.
- A node is connected to another node if they are at dist of power of 2.

Properties:-
N/w size, N =
node degree, d =
Diameter, D =

Hypercube

- can be of diff
  we define
- symmetric
- msg transit
  when two no
  than msg tr
- n no. of
  in hyper
- transfer

Adding

to another
hypercube

ring

since, hyp
can be taper

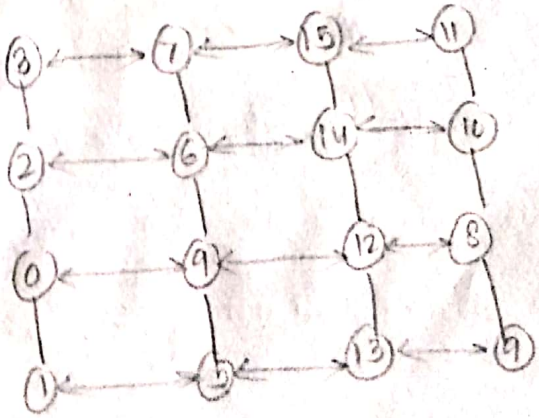
Hypercube

- can be of diff dimension.
  We define its " as node degree of each node
- symmetric topology. So, node degree is same for all nodes.
- msg ~~transit~~ transit in hops blw such node pairs.
  When two nodes or processors are not directly connected then msg transit blw them in multiple hops.
- max^n no. of hops necessary under the worst case for n-dim hypercube is n.
- msg transfer can be done either manually or auto.

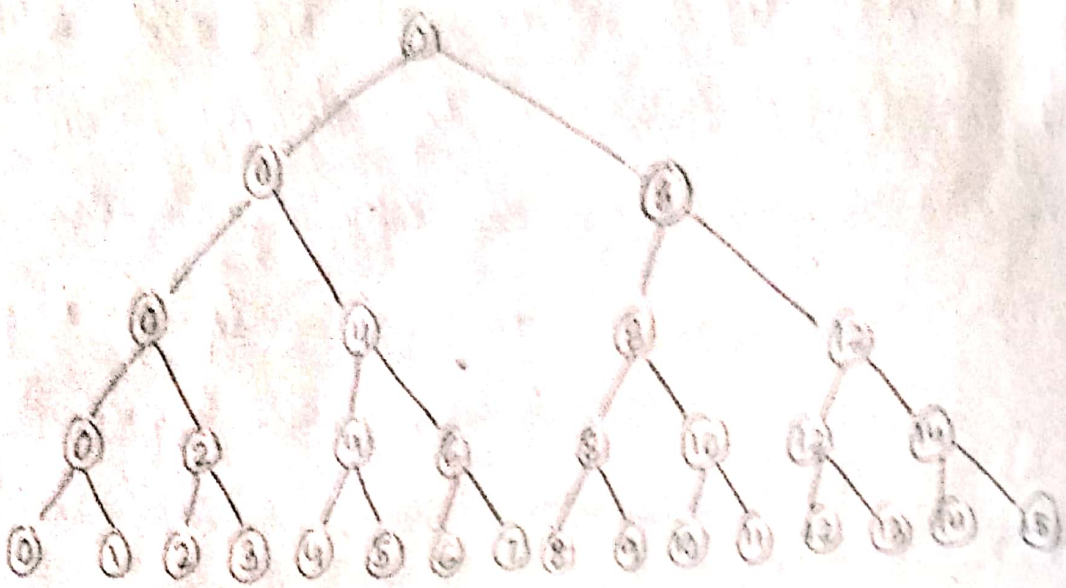
Embedding is defined as process of transformation of one N/w to another N/w.

eg hypercube can embed a mesh
" " " tree
" ring " embeded in torus (called perfect embedding)

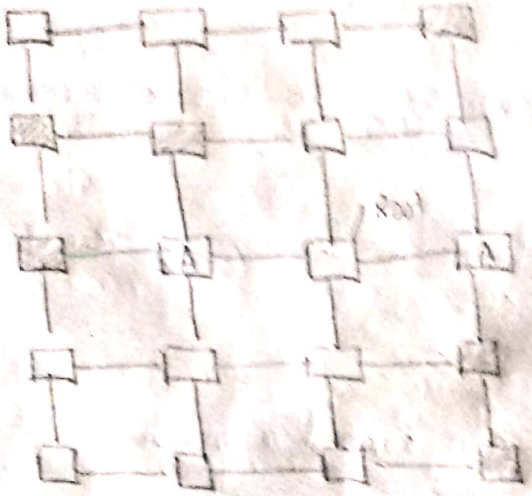
* since, hypercube is symmetric archi, any processor can be taken as root of tree.



Hypercube embedding a mesh



hypercube embedding tree.

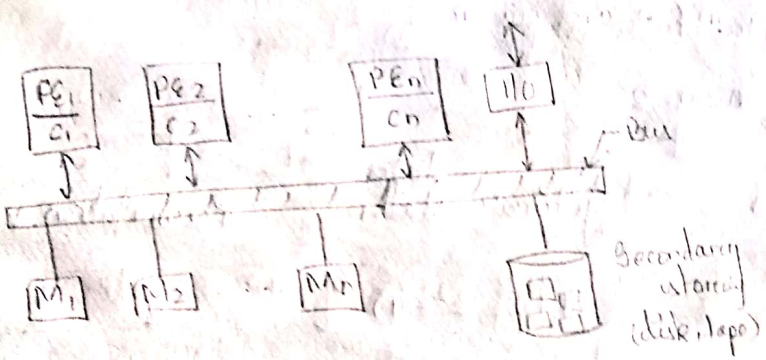


→ Quality of embedding is indicated by term- dilation
→ Dilation is defined as max^n no. of links in embedding nlw corresponding to one link in embedded n/w.

Dynamic Parallel Interconn N/w

1. Digital bus

- It is collection of wires & connectors for doing comm among PEs, memories & I/O devices. These devices are attached to bus.
- Bus contention may arise when multiple reqst arrive. So, bus arbitration logic must be able to allocate or deallocate bus service reqst one by one. That is why this bus is also called contention bus or time sharing bus among multiple functional modules.
- not costliere to implement
- disadv: limited BW



M - main memory
C - cache "
I/O - I/O subsys
PE - processing element.

(shows a multiprocessor sys with bus topology)

How this bus is practically implemented?

→ The sys bus is often implemented on backplane of a PCB (printed ckt board).

→ Other boards for processors, memories or device interface are plugged into backplane board via connector or cables.

→ The active devices (or master devices) like PEs or I/O devices generate rqst to address memory. The passive devices (or slave devices) like memories or peripherals respond to rqst.

→ This digital bus is used on time sharing basis.

→ Issues like bus ~~arbrit~~ arbitration, interrupt handling, coherance protocols & transaction processing must be handled

II. Switch Modules / Switches

→ (m x n) switch module

m - I/P
n - O/P.

eg (4x4), (8x8), ~~3x~~(4x8)

→ binary switch (m = n = 2)

→ practically, m & n are power of 2.

$$m = n = 2^k$$

Switch sizes	legitimate ~~states~~ states	Permutation conn^n
2 x 2	4 ($n = 2^2 = 4$)	2
4 x 4	256	24
8 x 8	16,777,216	40,320
n x n	n^n	$n!$

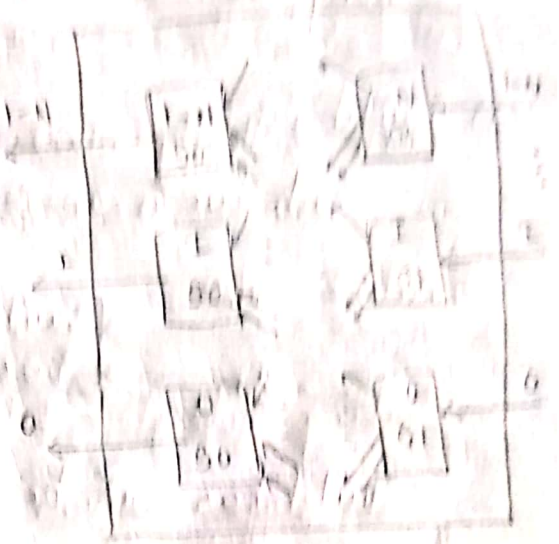
→ Here, each I/P one to one & one to many mappings are allowed but many to one mappings are not allowed due to conflicts at O/P terminal.

Dynamic n/w interconnection classes

1. Single stage N/w
2. Multi stage N/w

1. Single stage N/w

- It is switching n/w with N I/P selectors (IS) & N O/P selectors (OS).
- Each IS is 1:D mux.
  Each OS is DM:1 MUX
  where $1 \le D \le N$
  $1 \le M \le N$.
- eg crossbar ($D = M = N$).
- Single-stage N/w also called as recirculating N/w.
- higher is n/w connectivity, less no of recirculations.



(Single stage interconnection n/w - a conceptual view)

2. Multistage N/w

- also called MINS
- used in SIMD & MIMD comp.
- uses multiple (a×b) switches in each stage.
- a×b switches
  a - I/P
  b - O/P
  ISC$_i$ ($i \leftarrow 1$ to n) : Interstage connection
  S$_i$ ($i \leftarrow 1$ to n) : stages from 1 to n.

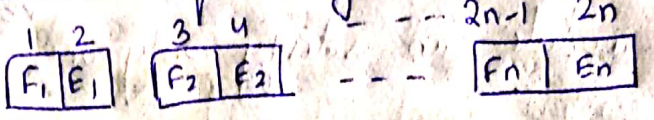
Pipelining

To incre throughput of a system, pipelining is used.

Let an instruc has 2 operations

Fetch (F) } let them take 1 CPU
& execute (E) }

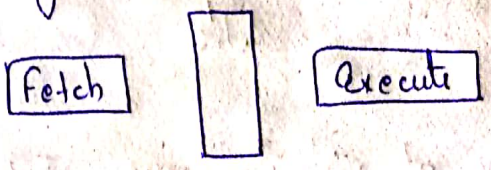
a) If we execute sequentially,



Let there are n instructions.

∴ no. of clk cycles needed = 2n

b) In pipelining.



↑ interstage buffers

↓ used to synchronise blw the 2 operations so that no conflict occurs

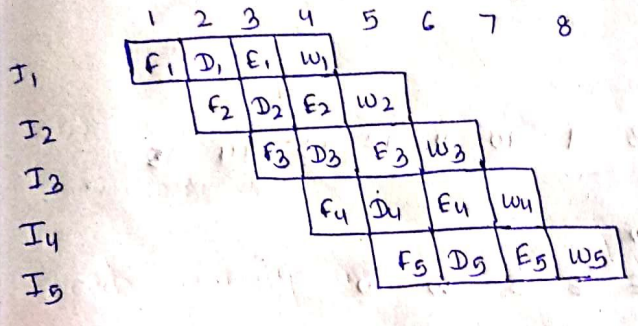
Let n = 4

	1	2	3	4	5	6	7
I_1	F_1	E_1					
I_2	-	F_2	E_2				
I_3	-	-	F_3	E_3			
I_4	-	-	-	F_4	E_4		

clk cycles needed = 5.

Let instruc has 4 segments

fetch (F)
decode (D)
Execute (E)
Write back (W)



interstage buffers required for 1 instruc = 3

speed up = $\frac{\text{total execution time in non-pipeline}}{\text{" " " in pipeline}}$

26/08/19

Q An instruc pipeline consists of 4 stages
fetch, D, E, W.
An 5 instruc. in a certain instruc. sequence need these
strategies for the # clks as shown.
Find no. of clks required to perform instruc.

	F	D	E	W
1	1	2	1	1
2	1	2	2	1
3	2	1	3	2
4	1	3	2	1
5	1	2	1	2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	F	D	D	E	W										
2		F	-	D	D	E	E	W							
3			F	-	-	D	D	E	E	E	W	W			
4				-	F	-	D	D	D	-	E	E	W		
5					-	F	-	-	-	D	D	-	E	W	W

↑ stall.

Hazards

- Control/ Instruction
- Data
- structural

* Any condn that causes pipeline to stall is called hazard.

Data Hazards

Partn. data when needed, is not available at that time - is cause of this hazard.

It is a condn in which either source or destin operands of an instruc. are not available at the time expected in the pipeline.

Due to this some operations has to be delayed & the pipeline stalls.

Control / Instru

This hazard unavailabi

eg It may o to be fetched

Structural

This condn given hlw The next com stage).

If the instruc then only on Hence, it is

∴, many pro avoid this de

Control / Instruction Hazards

This hazard is a condⁿ in which delay occures due to the unavailability of instruc.

eg It may occure due to a cache miss, requiring the instruc. to be fetched from memory.

Structural Hazards

This condⁿ occurs when 2 instruc. require the use of a given h/w at the same time.

The most common case is to access memory. (execute or write stage).

If the instruc. & data reside in the same cache unit then only one instruc. can proceed, the other will wait. Hence, it is delayed.

∴, many processors use separate instruc & data caches to avoid this delay.