Charlie
Date
Page No.

$$5 = \frac{1}{(1-0.8) + \frac{0.8}{N}}$$

$$1 = 0.2 + \frac{4}{N} \quad \text{(Not possible)}$$

~~$1 = 4$~~

Tutorial

Consider a hypothetical processor which runs crystal oscillator on 1MHz frequency of a code with 2M instructions in it is executed in 2.5 sec. Then find the CPI.

$f = 10^6$ Hz.

Execution time = IC × CPI avg × τ.

$$2.5 = 2 \times 10^6 \times CPI \times \frac{1}{10^6}$$

$$2.5 = 2 \times CPI.$$

$$CPI = 1.25.$$

What is the mix rating (MIPS)

$$MIPS = \frac{f}{CPI \times 10^6}$$

$$= \frac{10^6}{1.25 \times 10^6} = \frac{4}{5} = 0.8$$

Page No.

Q:- This microprocessor can be used to execute two types of instructions class A and class B. If the code contains 40% class B instructions then what will be the overall speedup if class A instruction execution module is enhanced with the speed up of 3.

Speedup = $\frac{T_{old}}{T_{new}}$

$3 = \frac{1}{(1-f) + \frac{f_A}{N}}$

$3 = \frac{1}{(1-f) + \frac{f_A}{0.4N}}$

class B → N

$f_A = 0.4N$

Speedup B = $\frac{1}{(1-f) + \frac{f_B \cdot f}{N \text{ Sm}}}$

$= \frac{1}{0.4 + \frac{0.6}{3}}$

$= \frac{1}{0.4 + 0.2} = \frac{10}{06}$

$= 1.66$

Charlie

Date

Page No.

(ii) What is the new execution time after speed up.

Speed = $\frac{T_{old}}{T_{new}}$

$3 = \frac{1.66}{T_{new}}$ $1.66 = \frac{2.5}{T_{new}}$

$T_{new} = \frac{2.5}{1.66}$

Q:- Consider a hypothetical microprocessor on which a code with 10 M instructions are executed in time of 8.5 sec. A code has two types of instructions in it - integer instruction type and floating point. If a newer microprocessor is introduced with an enhancement in the integer type instruction module such that its execution time of these instructions is reduced by the factor of 0.75 (75%). If floating point instructions has participation of 70% in the overall code then find the overall speedup.

$S_{en} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{0.75 \times T} = \frac{4}{3}$

$Speed\ up = \frac{1}{(1-f) + \frac{f}{S_{en}}} = \frac{1}{(1-0.3) + \frac{0.3 \times 3}{4}}$

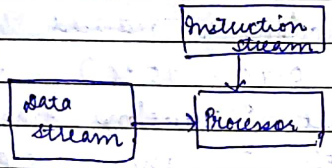
$= 1.08$

Flynn's Taxonomy for parallel computing Architecture.

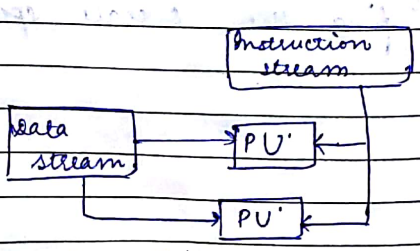
Based on no of Instruction stream and data stream

1) Single Instruction Single Data (SISD)
2) " " Multiple " (SIMD)
3) Multiple " Single " (MISD)
4) Multiple " Multiple " (MIMD)

(1) SISD:

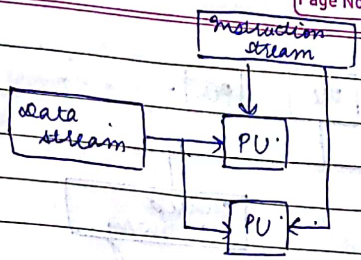


(2) SIMD.



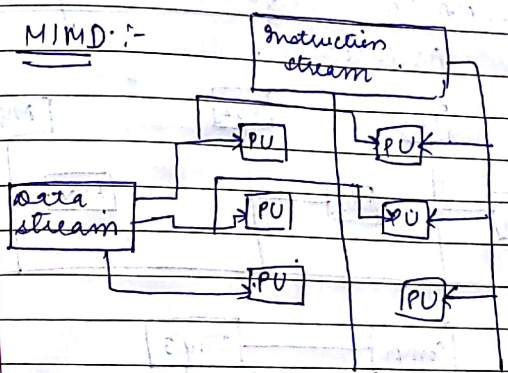
vectors and matrix operation may super computers.

(3) MISD



Ex:- $y = \sin(x) + \cos(x) + \tan(x)$.

(4) MIMD:-

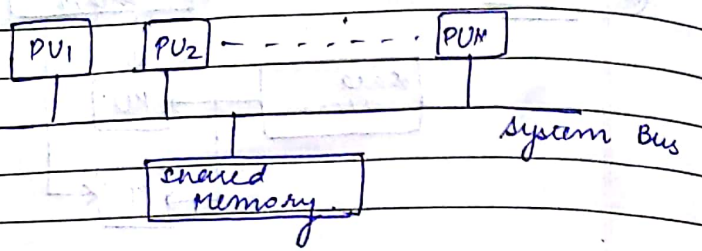


Ex:- Multiprocessors Multicomputers.

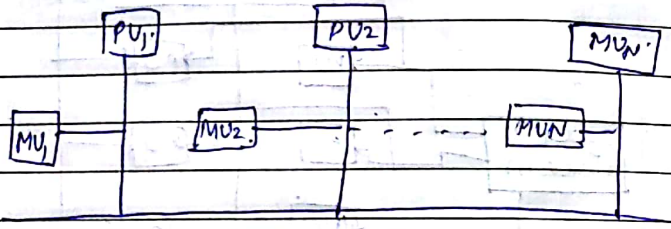
Modern architecture classification.

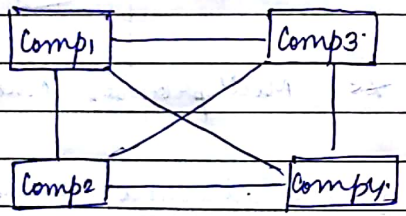
1) Shared memory multiprocessor.
2) Distributed " "
3) Message passing Multicomputer.

UMA (Uniform Memory Access Time) Architecture.



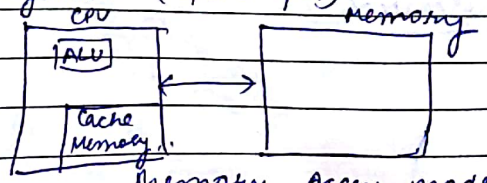
NUMA (Non uniform Memory Access) Architecture.





MIPS Architecture. ~ 200 Instructions.
RISC architecture → reduce instruction set, all instructions are of same length (32 bit)
ISA (Instruction set Architecture).

32 Registers ($0 - $31)



Memory Access modes.

Register Name	Number
$zero	$0
$at	$1
$v0, $v1	$2, $3
$a0 - $a3	$4 - $7
$t0 - $t7	$8 - $15 (temporary variable)
$s0 - $s7	$16 - $23 (saved variable)
$t8, $t9	$24, $25
$k0, $k1	$26, $27
$gp	$28
$sp	$29
$fp	$30
$ra	$31

Date
Page No.

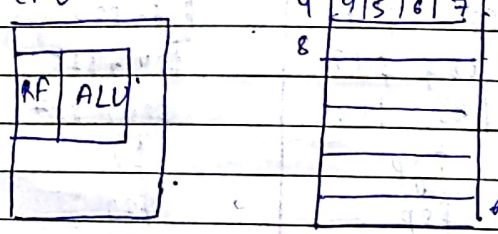
Charlie

MIPS 32-bit ISA.

(1) $f = h + A[i]$.

$\$t0$ $\$t1$ $\$t2$ (shifting by 2)

$\$t2 = \&(Mem\,A[0])$ 32 bits



Memory is byte addressable.

01000_2 (8)

10000_2 (16).

← Shift (left logical).

```
sll  $t4, $t3, 2   // $t4 = $t3 * 4
                   = i * 4.
add  $t5, $t2, $t4 // $t5 = $t2 + $t4
                   = addr(Mem(A[0])) + i * 4.
lw   $t6, 0($t5)
add  $t0, $t1, $t6 // $t0 = $t1 + $t6
                   // f = h + A[i]
```

load byte
load half word
load word

Shamt → shift amount.

T - Target
D - destination

Date
Page No.

Charlie

```
A[i] = h + A[i]
sll  $t4, $t3, 2.
add  $t5, $t2, $t4
lw   $t6, 0($t5)
add  $t0, $t1, $t6
sw   $t0, 0($t5)
```

R format; I format; I format.

MIPS 32 bit architecture.

Type	opcode	5 bits	5 bits			
R	6 bits	RS	Rt	RD (5 bits)	Shamt (5 bits)	function (6 bits)
I	6 bits	RS	Rt	Immediate (16 bits)		
J	0 bits	address (26 bits)				

1 word = 32 bits = 4 Bytes

Load and store.

Mnemonic	Format	Encoding	Offset (16 bits)
	R	32 10 rs rt	
lb	I	32 10 rs rt	u
lh	I	33 10 rs rt	u
lw	I	35 10 rs rt	u
lbu	I	36 10 rs rt	u
sw	I	43 10 rs rt	u

store word

load byte unsigned.

sll → logical shift left
srl → " " right
sra → arithmetic shift right (sign extended)

Charlie
Date
Page No.

Arithmetic and logical Instructions.

Mnemonic	Format	Encoding
add	R	0 rs rt rd 0 32
addu (add unsigned)	R	0 rs rt rd 0 33
sub	R	0 rs rt rd 0 34
and	R	0 rs rt rd 0 36
Or	R	0 rs rt rd 0 37
XOR	R	0 rs rt rd 0 38
addi	I	8 rs rt ← Immediate →
andi	I	12 rs rt ← Immediate →

Shift.

Mnemonic	Format	Encoding
sll	R	0 0 rt rd sa 0
srl	R	0 0 rt rd sa 2
sra	R	0 0 rt rd sa 3

Jump and Branch.

Mnemonic	Format	Encoding
j (jump to add.)	J	2 ← Immediate address →
jal (jump & link)	J	3 ← →
beq (branch if equal)	I	4 rs rt ← offset →
bne (branch if not equal)	I	5 rs rt ← offset →

Q:- if (i == j)
{
f = g+h
}
Translate in MIPS assembly code.

Suppose $f0 → i, $f1 → j
bne $s0, $s1, end
add $s2, $s3, $s4
end

Q:- if (i == j)
{ f = g+h; }
else
f = g - h; }

bne $s0, $s1, else
add $s2, $s3, $s4
j end
else sub $s2, $s3, $s4
end;

Q:- for (i = 0; i < 100; i++)
{
a[i] = b[i] + 10;
}

$s0 → $to
$s1

It will be changed to byte address
so, i < 400 ap i+4

sll → multiply opr.

addi $t2, $0, 100 // $t2 = 100 → Iteration limit

loop: sll $t1, $t0, 2 // $t4 = $t0 * 4
= i * 4

add $t3, $s1, $t1 // $t3 = $s1 + $t1.
addr (b[0]) + i*4 = addr (b[i])

lw $t4, 0($t3) // $t4 = b[i].
addi $t5, $t4, 10 // $t5 = b[i] + 10
add $t6, $s0, $t1 // $t6 = addr
(a[0]) + i*4 = addr (a[i])

sw $t5, 0($t6) // a[i] = b[i] + 10
addi $t0, $t0, 1 // $t0 = $t0 + 1 = (i+1)
bne $t0, $t2, loop.

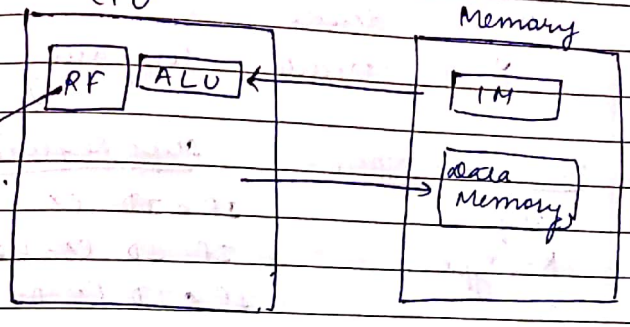
Q: while (i != j)
{
if (i > j)
i = i - j;
else
j = j - i;
}

beq $t0, $t1, END.
slt $t2, $t0, $t1 // if $t0 is less than $t1 than $t2 = 1 otherwise $t2 = 0.
beq $t2, $0, Else.
sub $t0, $t0, $t1
J END
Else: sub $t1, $t1, $t0.
END.

sll → set less than

Pipelining (MIPS - 5 stage)

assembly is in instruction memory.



add R1, R2, R3 // R1 = R2 + R3

(1) Instruction fetch
(2) " decode
(3) " execute ie. R2 + R3
(4) IDLE
(5) WriteBack ie. R1 = R2 + R3

II lw R1 32(R2).

(1) Instruction fetch.
(2) " decode
(3) " execute (32 + R2) (add. calculation)
(4) Memory access in data memory pointing at 1000
(5) Write back ie. R1 is stored at 1000.

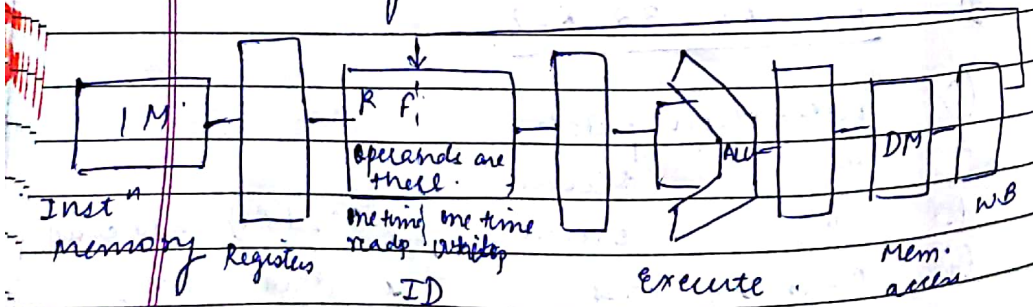
beq $r_1, r_3, 36$.

① Instruction fetch
② " decode
③ " execute - comparing r_1 & r_3.

Instruction type :	Steps Required.
beq	IF ID EX
R-Type	IF ID EX WB
sw	IF ID EX Mem
lw	IF ID EX Mem WB. (all 5 steps)

sw R_1, $0(R_2)$.

① Instruction fetch
② " decode
③ " execute : $= 0 + R_2$
④ Memory access to store



	I1	I2	I3	I4	I5
	F	D	E	M	W
		F	D	E	M	W
			F	D	E	M	W
				F	D	E	M	W
					F	D	E	M	W

→ Time (cycle)

1. Non pipeline then for 5 instruction take 25 cycles.
2. Pipe pipeline then for 5 instruction take 9 cycles.

1. CPI = 25/5 = 5
2. CPI = 9/5 = 1.8 less (Best)

no. of cycles / instruction

IPC (Instruction per cycle) = Throughput

① IPC = 5/9 = 0.55 Best
② IPC = 5/25 = 0.2.

Latency is not improved in pipelining but throughput is improved.

Latency = CPI

Pipeline overhead :

IF | ID | EX | Mem | WB.

Take maxm time (from time domain by MEM).

IF | ID | EX | MEM | WB

equal time according to max time.

add R1, R2, R3	F D E M W
sub R5, R6, R7	F D E M W
lw R8, 0(50)	F D E M W.

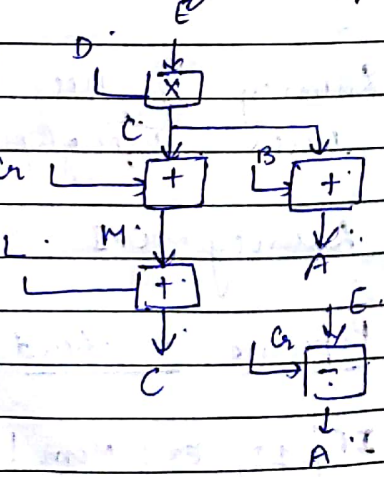
Total no of cycles taken to complete

add R1, R2, R3	F D E M W.
Sub R5, R1, R4	F - - - D E M W.
lw R2, 0(50)	

Pipeline Hazard.

1. Structural hazard
2. Data Hazard (data depending)
3. Control or Branch Hazard

Q:- S1 : C = D X E
S2 : M = Cr + C
S3 : A = B + C
S4 : C = L + M
S5 : R = A ÷ E
3 depending



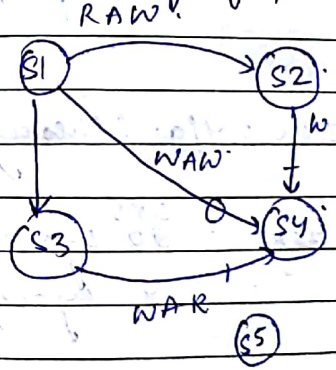
RAW (Read after write). →
WAR (Write after Read) ⊣→
WAW (Write after write) ⊸→

RAW
C = D X E. M = Cr + C → Read

WAR. M = Cr + C → Read, C = L + M (Anti dependency)
Write

WAW. C = D X E. C = L + M
called o/p dependency.

Dependency graph.



Q:- S1: load R1 // R1 = A
S2: add R2, R1 // R2 = R2 + R1
S3: move R1, R3 // R1 = R3
S4: store B, R1 // B = R1

Data hazard is avoided by forwarding

Date
Page No.

Charlie

Bernstein condition of parallelism.

I_1, I_2, O_1, O_2.

$I_1 \cap O_2 = \phi$, $I_2 \cap O_1 = \phi$, $O_1 \cap O_2 = \phi$

S1: add r1, r2, r3 // $r_1 = r_2 + r_3$

S2: sub r4, r1, r3 // $r_4 = r_1 - r_3$

Load use hazard.

```
lw  R1, 0(R2)        F D E M W
add R2, R1, R3       F D - E M W
lw  R1, 0(R5)          F D E M W
sw  R1, 0(R6)          F D E M W
```

CPI = 9/4 , IPC = 4/9. (throughput).

Non pipeline will take = 4×5 = 20 cycles.

pipeline " " = 9 cycles

speedup = 20/9 = 2.2

Q:- add r1, r2, r3. F D E M W.

sub r4, r1, r3 F D E M W

and r6, r1, r2 F D E M W

OR r8, r1, r9. F D E M W.

XOR r10, r1, r11. F D E M W

CPI = 9/5

IPC = 5/9.

Charlie

Date
Page No.

Q:-
```
lw  R1, 0(50)
lw  R2, 0(51)
add R3, R1, R2
sw  R3, 0(52)
lw  R4, 0(52)
lw  R5, 0(53)
sub R6, R4, R5.      14 cycles
sw  R6, 0(54).
```

code reordering.

```
lw  R1 0(50).
lw  R2 0(51)
lw  R4 0(52)
lw  R5 0(53)          12 cycles.
add R3, R1, R2.
sw  R3, 0(53)
sub R6, R4, R5
sw  R6, 0(54)
```

1.	F	D	E	M	W							
2.		F	D	E	M	W						
3.			F	D	-	E	M	W				
4.					F	D	E	M	W			
5.						F	D	E	M	W		
6.							F	D	E	M	W	
7.								F	D	E	M	W
8.									F	D	E	M

Date
Page No.

MIPS 5 stage pipeline.

```
lw    R1, 0(R2)
lw    R3, 0(R4)
add   R5, R1, R3.
sw    R5, 0(R6)
lw    R7, 0(R8)
lw    R9, 0(R10)
sub   R11, R4, R9.
sw    R11, 0(R12)
```

```
F D E M W.
  F D E M W.
    F D - E M W.
      I F D E M W.
        F D E M W
          F D E M W.
          F D - E M W'
          F F D E M W
```

14 cycles.

Code Reordering

```
lw R1, 0(R2)
lw R3, 0(R4)
lw R7, 0(R8).
lw R9, 0(R10)
add R5, R1, R3.
sw R5, 0(R6)
sub R11, R7, R9.
sw R11, 0(R12)
```

F D E M W.
F D E M W.
F D E M W
F D E M W
F D E M W
F D E M W.

12 cycles

Control or Branch Hazard.

```
loop: beq  r1, r3, 36   ' F D E '
      and  r2, r3, r5   '
      OR   r6, r1, r7   '
      add  r8, r1, r9   '
      XOR  r10, r1, r11 '
      beq  r12, r13, r9, loop '
```

1. Branch prediction
   (i) Branch Taken
   (ii) Branch Not Taken.
2. Provision of delayed.

Control a branch hazard.

```
for ( i = 1000 ; i > 0 ; i = i - 1).
{
    x[i] = x[i] + s
}.
```

```
1.  addi $t1, $0, $0  // t1 = 0.
loop: 2. sll $t2, $t0, 2   // t2 = t0 x 4 = 7 x 4.
3.  add  $t3, $t2, $t0  // t3 = addr(x[i])
4.  lw   $t4, 0($t3)  // t4 = x[i] + i x 4.
5.  add  $t5, $t4, $f1  // t5 = x[i] + s
6.  sw   $t5, 0($t3)  // x[0] = x[i] + s
7.  subi $t0, $t0, 1  // t0 = t0 - 1 = i - 1
8.  bne  $t0, $0, loop.
```

1. F D E M W
2. F D E M W
3. F D E M W
4. F D E M W
5. F D E M W
6. F D E M W
7. F D E M W
8.

Charlie
Date
Page No.

Loop unrolling / Register Renaming

```
for ( i = 1000 ; i>0; i = i - 4)
{   x[i]   = x[i] + s;
    x[i-1] = x[i-1] + s;
    x[i-2] = x[i-2] + s;
    x[i-3] = x[i-3] + s;
}
```

250 times.

Register Renaming:-

```
addi  $t1,  $s0,  $0  // t1 = 0.
sll   $t2,  $t0,  2   // t2 = to * 4 = i * 4.
add   $t3,  $t2,  $s0 // t3 = addr(x[0] + i*4 = addr(x[i])
lw    $t4,  0($t3)    // t4 = x[i]
add   $t5,  $t4,  $s1 // t5 = x[i] + s
sw    $t5,  0($t3)
lw    $t6,  -4($t3)   // t6 = x[i-1].
lw    $t7,  -8($t3)   // t7 = x[i-2] + s
lw    $t8,  -12($t3)  // t8 = x[i-3] + s
```

Q:

```
for ( i = 1000; i > 0; i = i - 1)
{
    x[i] = x[i] + s.
}
```

$s0 → x[i]
$s1 → s

→ Superscalar computer
→ VLIW processor.

Charlie
Date
Page No.

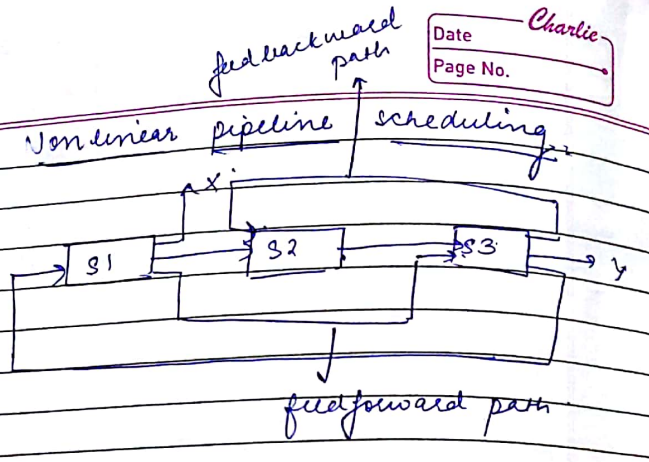
```
(1)  add   $t1, $0, $1000   // t1 = 0
loop
2.   sll   $t2, $t0, 2      // t2 = i * 4.
3.   add   $t3, $s0, $t2    // t3 = addr(x(i))
4.   lw    $t4, 0($t3)     // t4 = x[i]
5.   lw    $t5, -4($t3)    // t5 = x[i-1]
6.   lw    $t6, -8($t3)    // t6 = x[i-2]
7.   lw    $t7, -12($t3)   // t7 = x[i-3]
8.   add   $t8, $t4, $s1   // t8 = x[i] + s
9.   add   $t9, $t5, $s1   // t9 = x[i-1] + s
10.  add   $s2, $t6, $s1   // s2 = x[i-2] + s
11.  add   $s3, $t7, $s1   // s3 = x[i-3] + s
12.  sw    $t8, 0($t3)     // x[i] = x[i] + s
13.  sw    $t9, -4($t3)    // x[i-1] = x[i-1] + s
14.  sw    $s2, -8($t3)    // x[i-2] = x[i-2] + s
15.  sw    $s3, -12($t3)   // x[i-3] = x[i-3] + s
16.  subi  $t0, $t0, 4     // i = i - 4
17.  bne   $t0, $t9, loop.
```

Now → 5 + 250 × 18 = 4505
earlier → 5 + 1000 × 10 = 10005

Super scalar computer.

2 fetch, decode, execute.
Multiport memory where 2 read and write can be done simultaneously.

Nonlinear pipeline scheduling



Reservation Table for function X.

	1	2	3	4	5	6	7	8
S1	X			I_2		X		X
S2		X		X	I_2		I_2	
S3			X		X	I_2	X	I_2

Forbidden latencies.

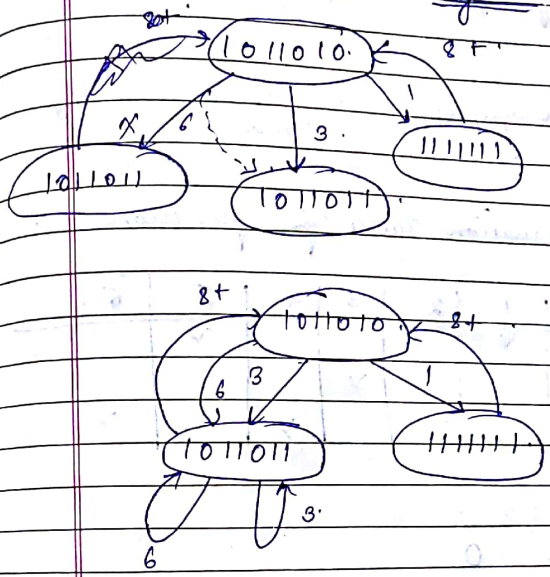
Stage 1 :- 2, 5, 7.
Stage 2 :- 2.
Stage 3 :- 2, 4.
2, 4, 5, 7.

Collision vector (7 bits).

C_7	C_6	C_5	C_4	C_3	C_2	C_1
1	0	1	1	0	1	0

1011010
OR 0101101
1111111

State Transition diagram



Simple cycles.

(1, 8), (3, 8) (6, 8) (3, 6), (3, 3), (6, 6), (6, 3).

Avg. latency :- 5, 6, 7, 5, 3.

Min. Avg latency (MAL) = 3.

Greedy cycles :- (3,3), (1,8), (3,8), (3).

Throughput = Instruction / cycle = 1/3.

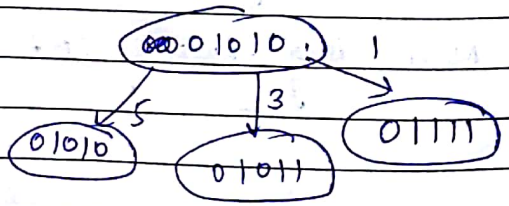
Q:- Reservation Table for function Y.

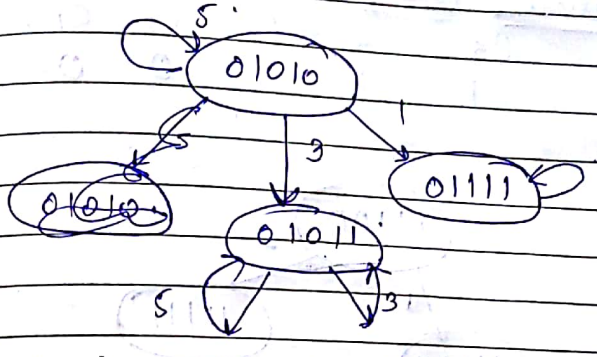
	1	2	3	4	5	6.
S1	Y				Y.	
S2			Y			
S3		Y		Y		X

Stage 1 = 4
Stage 2 = 0
Stage 3 = 2, 4.
2, 4

Collision vector

C6	C5	C4	C3	C2	C1
0	0	1	0	1	0





Simple cycles :- (1,5), (3,3), (5,5).
3, 3,

Throughput = 1/3.

greedy cycle :- (3, 3), (1,5).

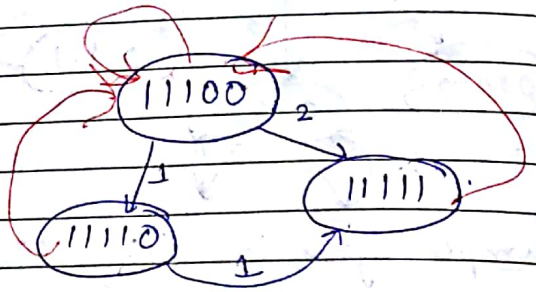
Q:-

	1	2	3	4	5	6.
S1	X				X	X.
S2		X		X	X	
S3			X			
S4				X		
S5		X			X	X.

stage 1 :- 5.
stage 2 :- 3.
stage 3 :- 0.
4 :- 0
5 :- 4
3, 4, 5.

Collision vector.

C5 C4 C3 C2 C1.
1 1 1 0 0.



01111

Code Reordering and loop unrolling.

```
for(i=0; i<1000; i++)
  {
    A[i] = h+A[i];
  }
```

```
1.  add  $t1, $0, 1000   // t1 = 1000.
2. loop: sll  $t2, $t0, 2.  // t2 = i*4.
3.  add  $t3, $s0, $t2   // t3 = addr(A[i]).
4.  lw   $t4, 0($s3)     // t4 = A[i]
5.  add  $t5, $s1, $t4   // t5 = h+A[i]
6.  sw   $t5, 0($t3)     // A[i] = h+A[i]
7.  addi $t0, $t0, 1     // i = i+1
8.  bne  $t0, $t1, loop.
```

$t4 → load use hazard.

1. F D E M W.
2. F D E M W.
3. F D E M W.
4. F D E M W.
5. - F D E M W.
6. F D E M W.
7. F D E M W.
8. F D E - -

If I do code reordering then it will solve the problem of stalling

Code Reordering:

```
1.  addi $t1, $0, 1000   // t1 = 1000.
2. loop: sll  $t2, $t0, 2   // t2 = i*4
3.  add  $t3, $s0, $t2   // t3 = addr(A[i]).
4.  lw   $t4, 0($t3)     // t4 = A[i]
5.  addi $t0, $t0, 1     // i = i+1
6.  add  $t5, $s1, $t4   // t5 = h+A[i]
7.  sw   $t5, 0($t3)     // A[i] = h+A[i].
8.  bne  $t0, $t1, loop.
```

F D E M W.
F D E M W
F D E M W
F D E M W
F D E M W
F D E M W
F D E - -

4($t3) = 4+ t3.

Charlie
Date
Page No.

loop unrolling.

```
for( i=0; i<1000 ; i=i+4).
{
    A[i] = h+ A[i];
    A[i+1] = h+ A[i+1];
    A[i+2] = h + A[i+2];
    A[i+3] = h + A[i+3];
}
```

```
addi  $t1, $0, 1000.
sll   $t2, $t0, 2 .
add   $t3, $s0, $t2 .
lw    $t4, 0($t3) .
lw    $t5, 4($t3) .
lw    $t6, 8($t3).
lw    $t7, 12($t3) .
addi  $t0, $t0, 4
add   $t8, $s1, $t4.
add   $t9, $s1, $t5
add   $s2, $s1, $t6.
add   $s3, $s1, $t9.
sw    $t5, 0($t3)
sw    $t9, 4($t3).
sw    $s2, 8($t3)
sw    $s3, 12($t3).
bne   $t0, $t1, loop.
```

$t0 = $t1.

Charlie
Date
Page No.

F D E I M W.