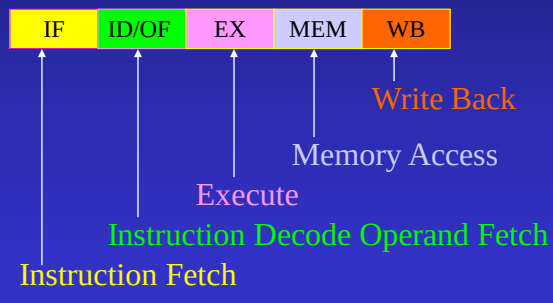
Chapter 4
(Part B)

PIPELINING

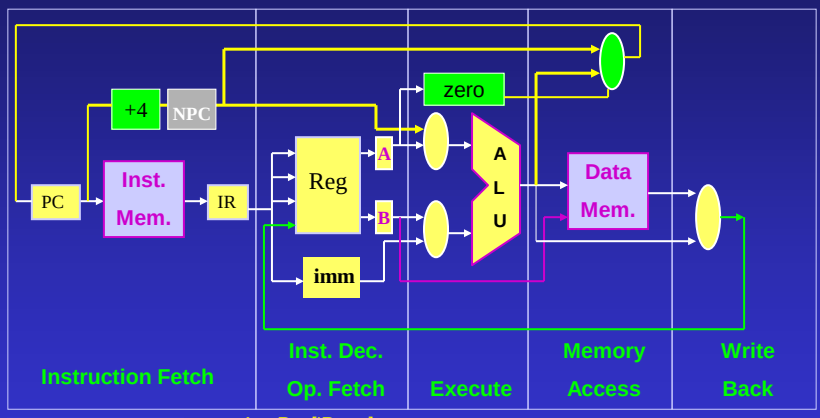
Processor

Basic steps to process an instruction



2

Datapath



IR ← Mem[PC]

NPC ← PC + 4

A← Reg[IR $_{6..10}$]

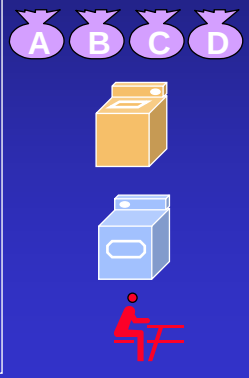
B← Reg[IR $_{11..15}$]

Imm← ((IR$_{16}$)^{16}## IR $_{11..15}$]

3

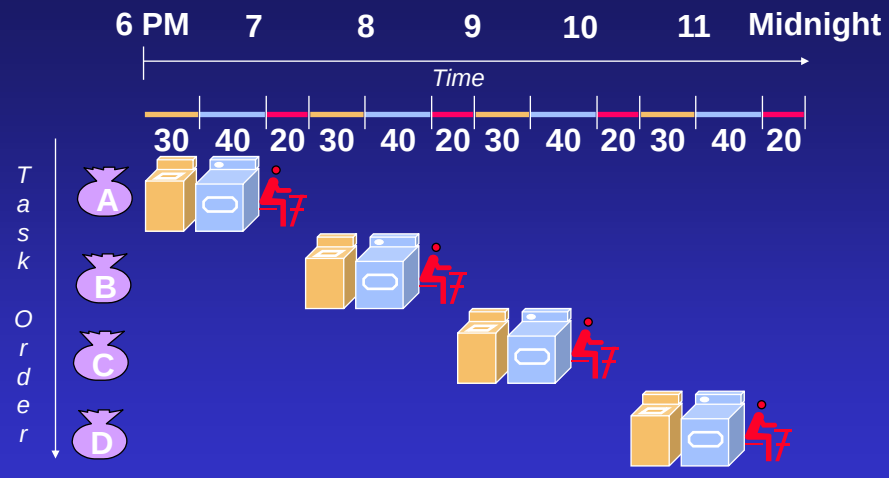
Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes



4

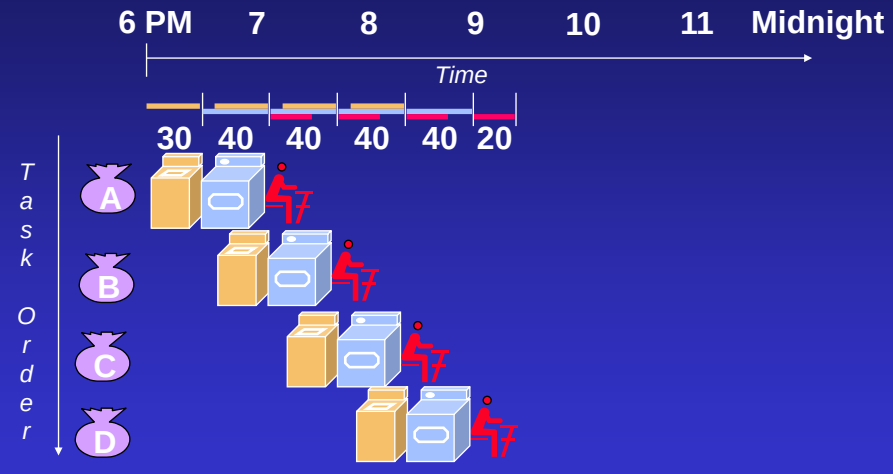
Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

5

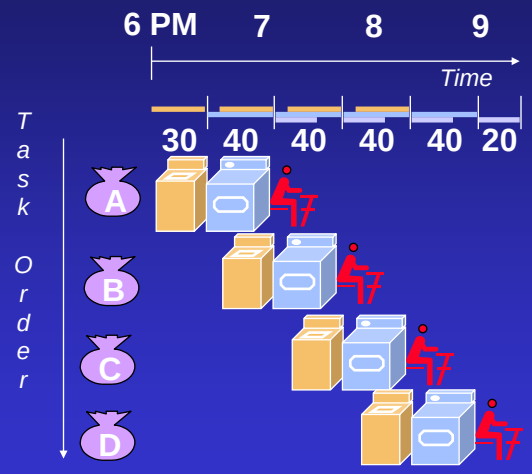
Pipelined Laundry
Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

6

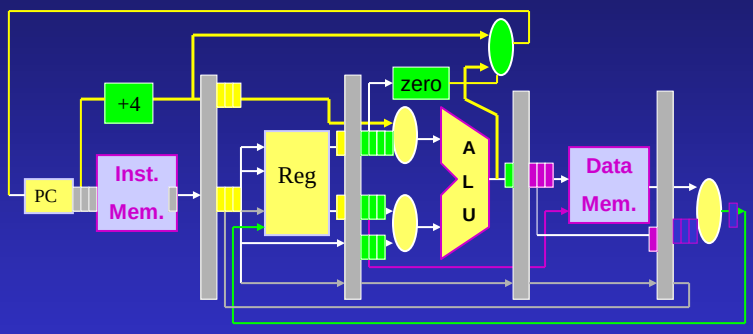
Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

7

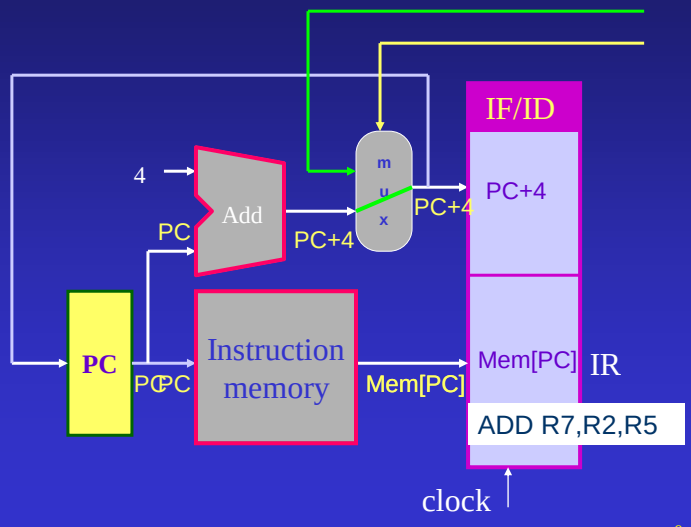
Datapath w/ pipeline



8

0

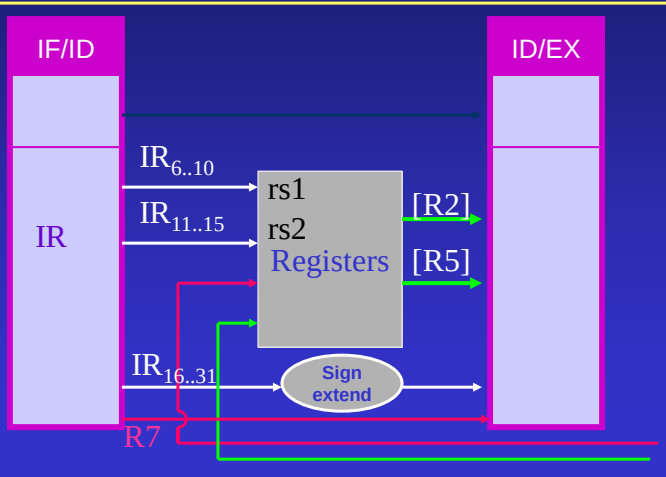
Instruction Fetch (IF) stage



9

1

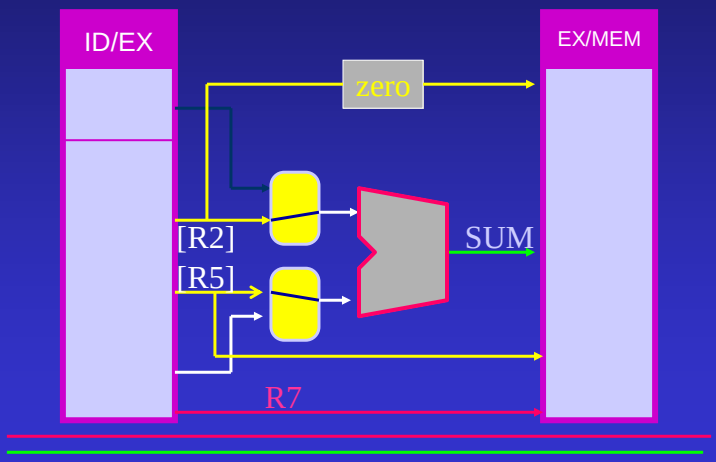
Instruction Decode (ID) stage



10

2

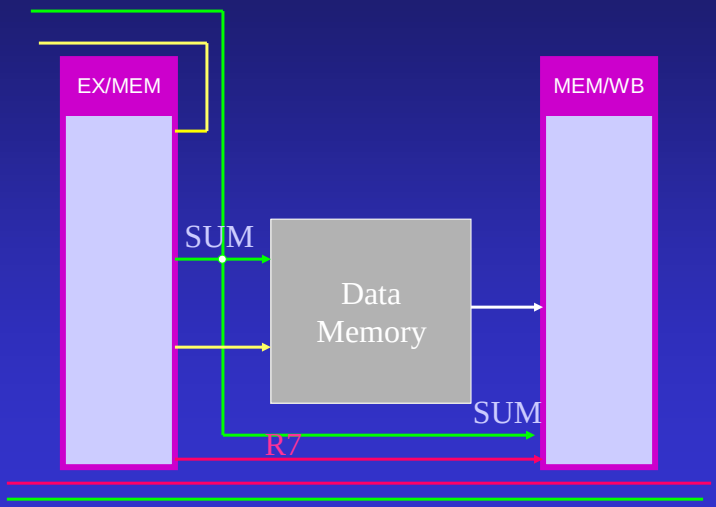
Execution (EX) stage



11

3

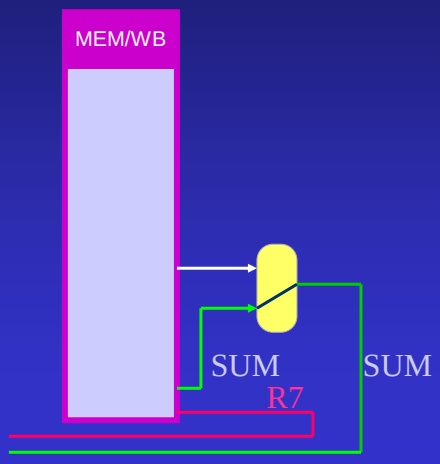
Memory (MEM) stage



12

4

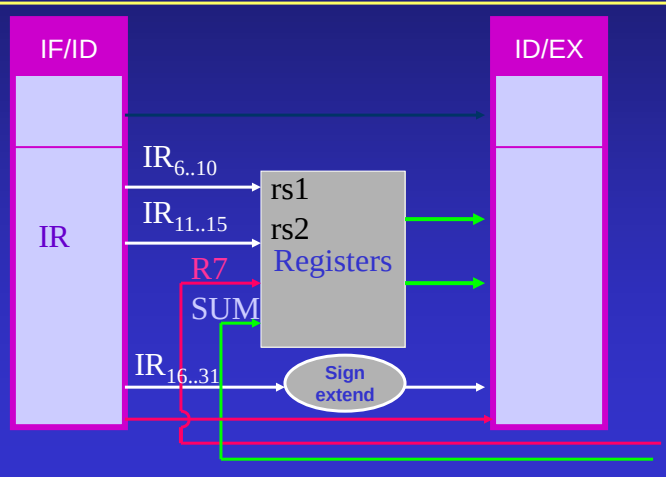
Write Back (WB) stage



13

4

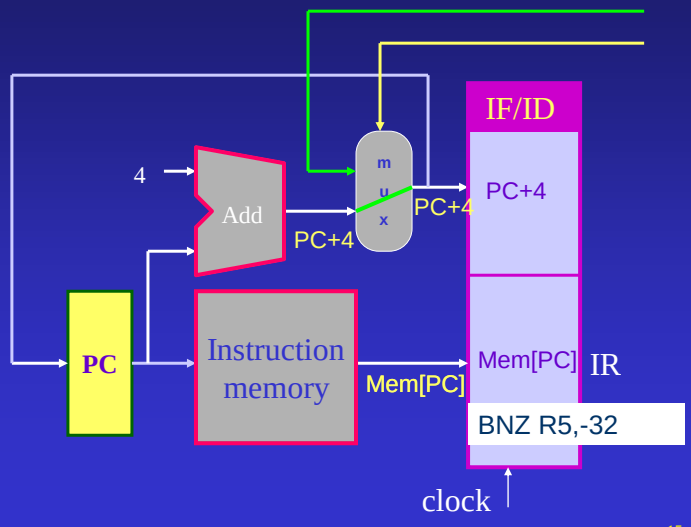
Instruction Decode (ID) stage



14

100

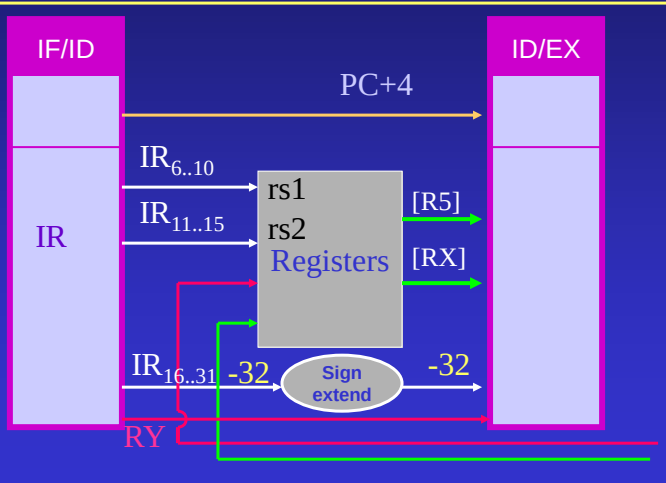
Instruction Fetch (IF) stage



15

101

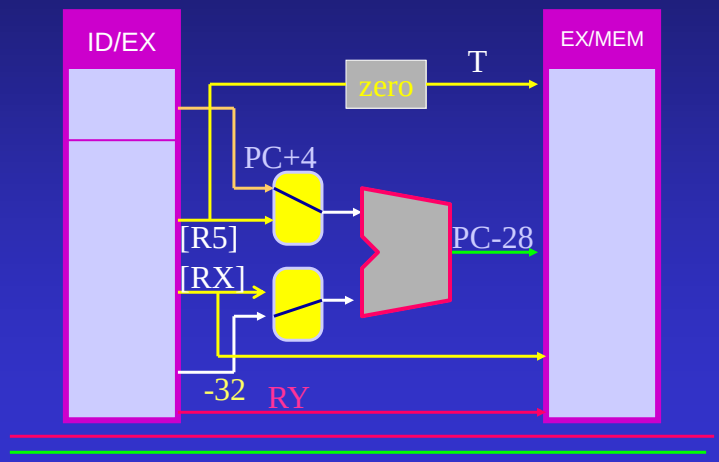
Instruction Decode (ID) stage



16

102

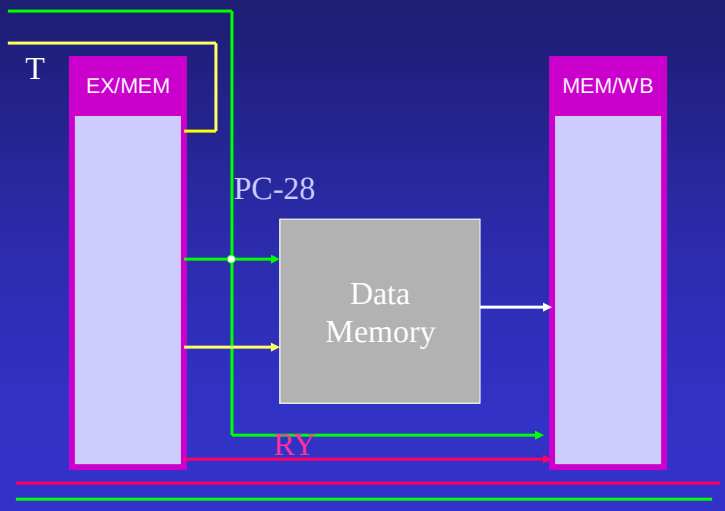
Execution (EX) stage



17

103

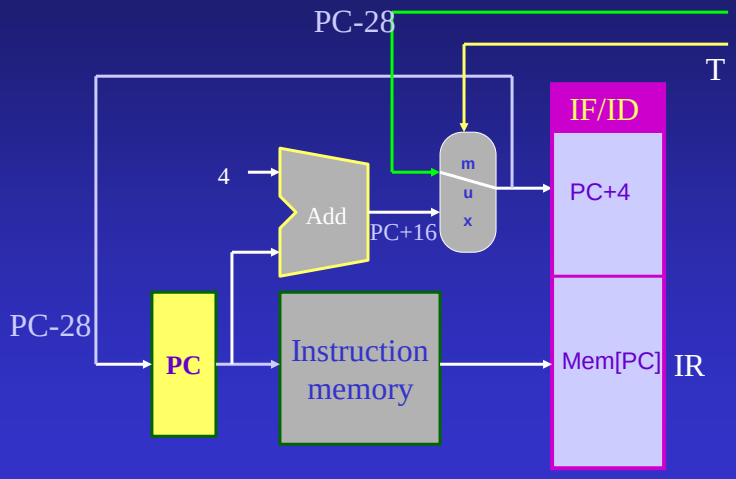
Memory (MEM) stage



18

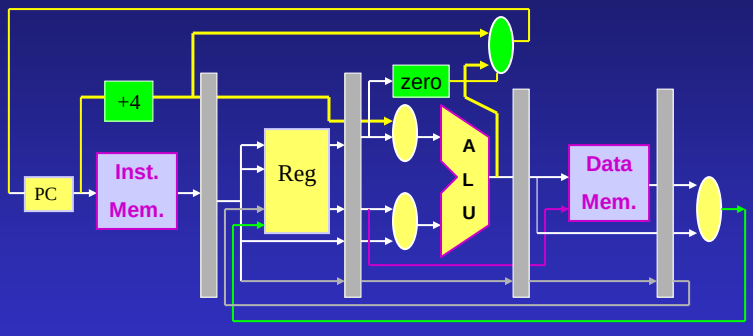
103

Instruction Fetch (IF) stage



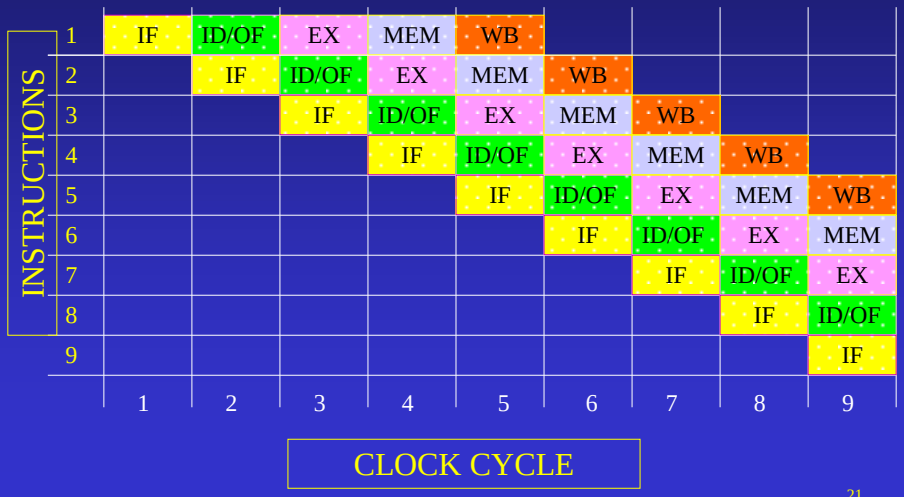
19

Datapath w/ pipeline



20

Pipeline

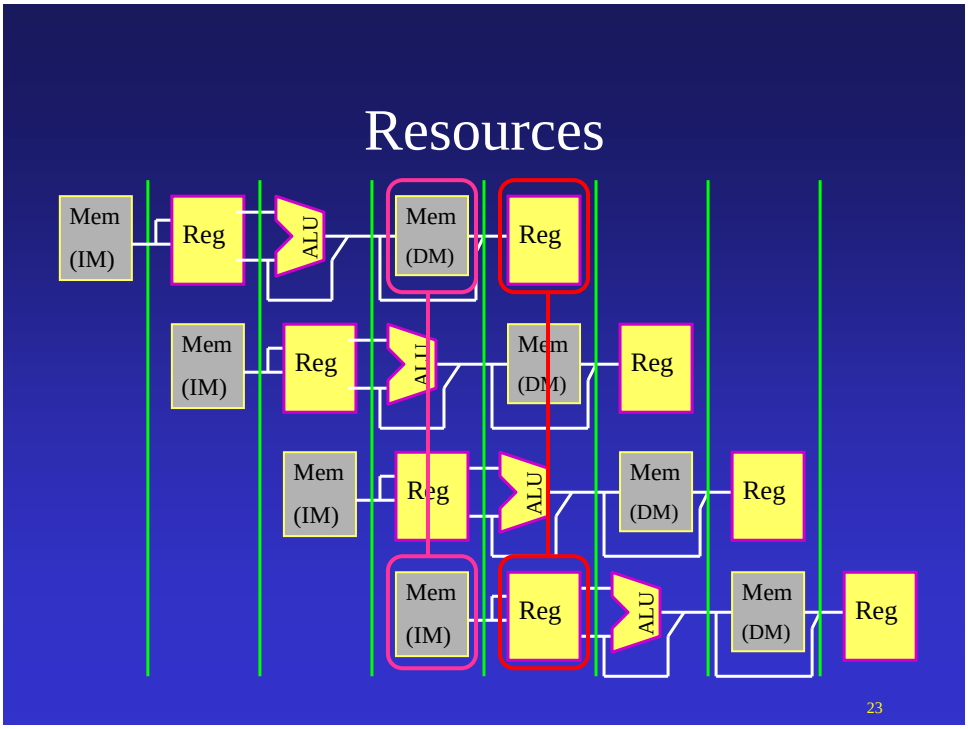


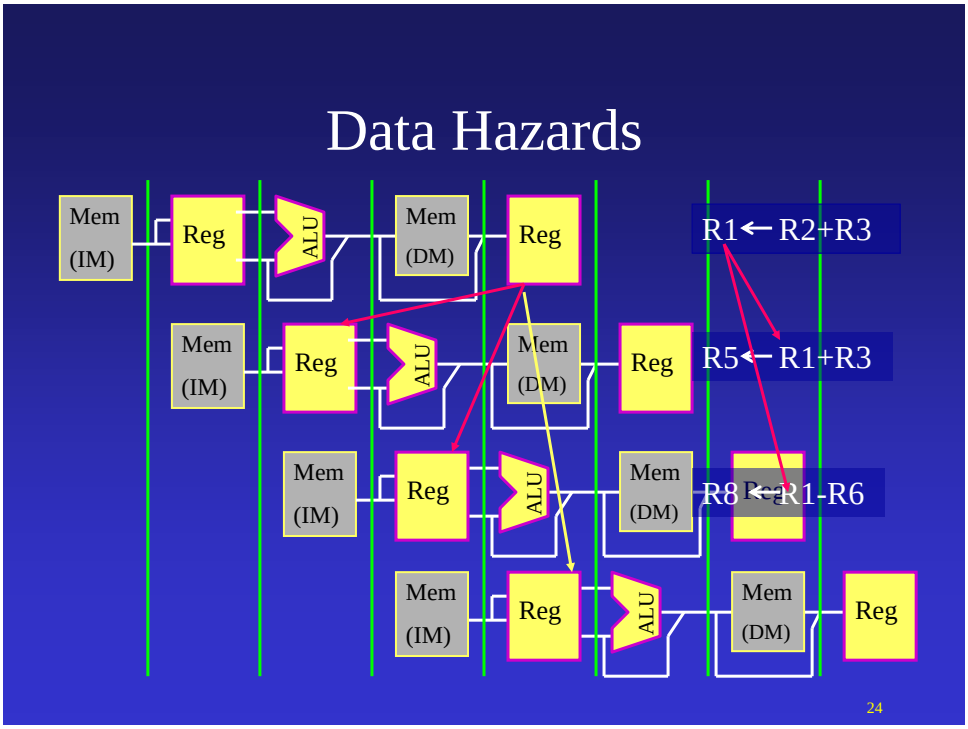
21

Pipeline Hazards

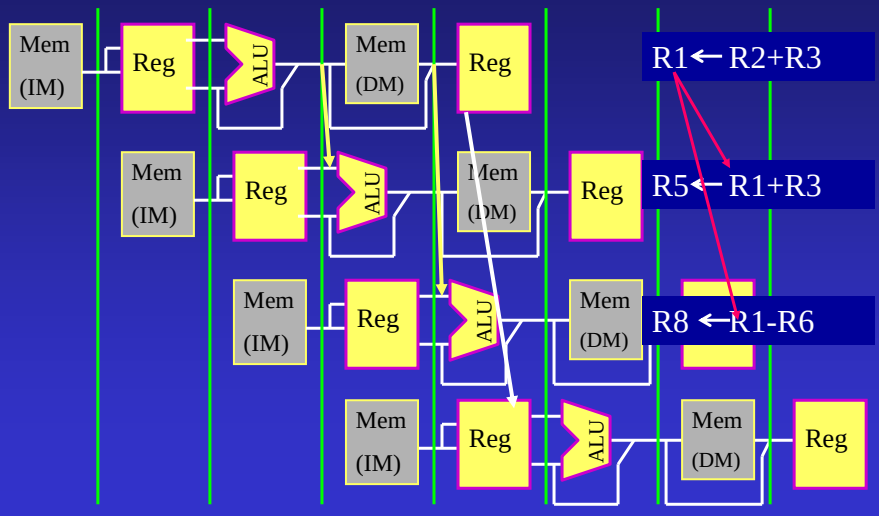
- Structural Hazards
  - two or more instructions use same hardware at the same time.
- Data Hazards
  - Data dependencies
  - Result from inst. j is needed by inst. k
- Control Hazards
  - Branch changes flow, what happen with the following instruction(s)

22


Resources
Mem (IM)
Reg
ALU
Mem (DM)
Reg
23

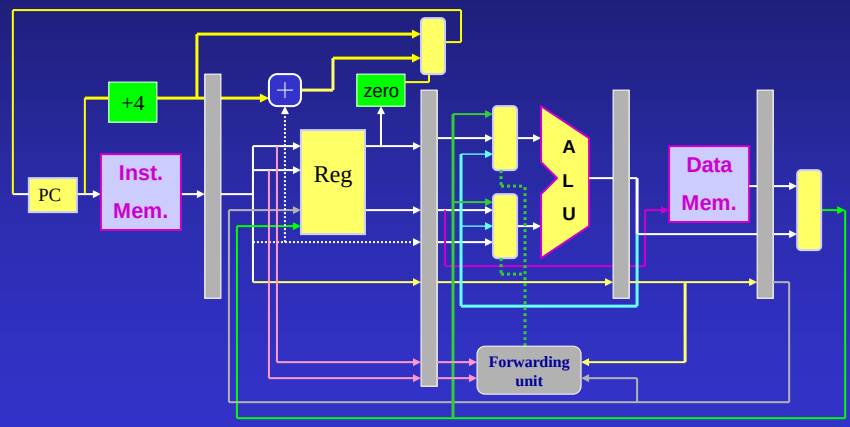

Data Hazards
Mem (IM)
Reg
ALU
Mem (DM)
Reg
R1← R2+R3
R5← R1+R3
R8← R1-R6
24

Data Forwarding



25

Datapath w/ pipeline



26

Example

ADD R1,R2,R3

SUB R4,R3,R1

XOR R7,R8,R1

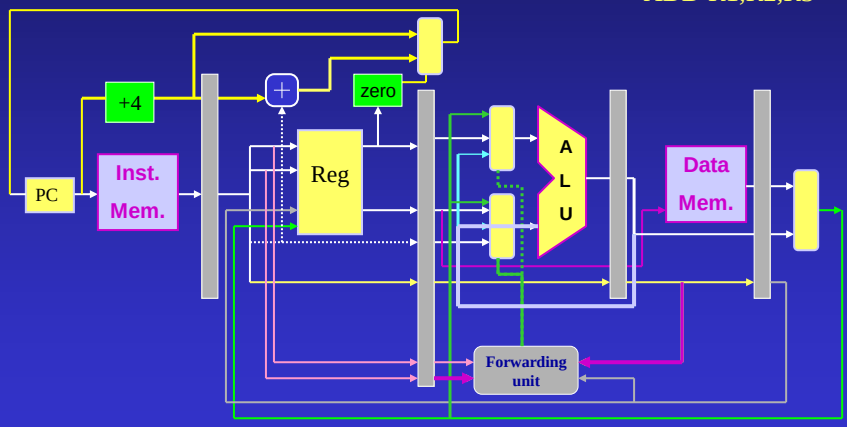
ADD R1,R2,R3

SUB R4,R3,R1

XOR R7,R8,R1

ADD R1,R2,R3

ADD R1,R2,R3

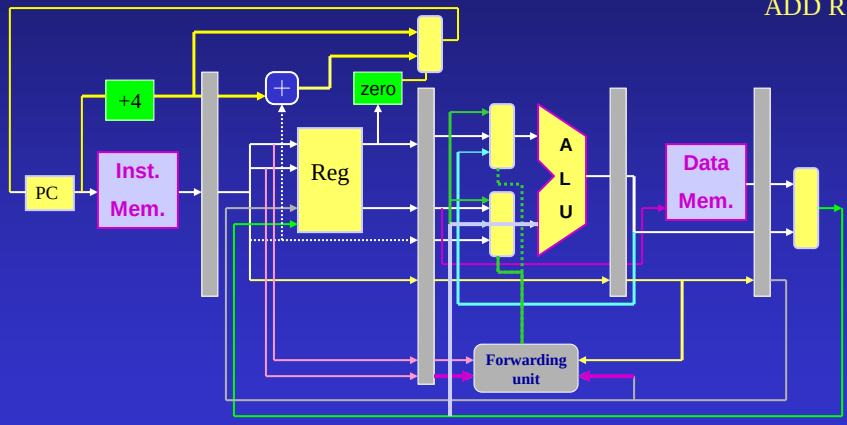


27

Example

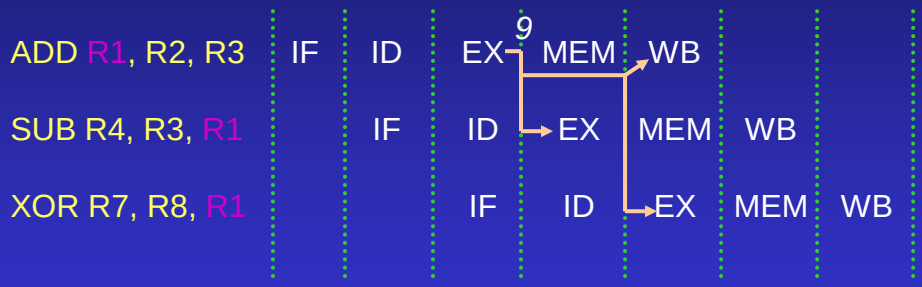
XOR R7,R8,R1 SUB R4,R3,R1

ADD R1..



28

Assume R2 & R3 values are 7&2

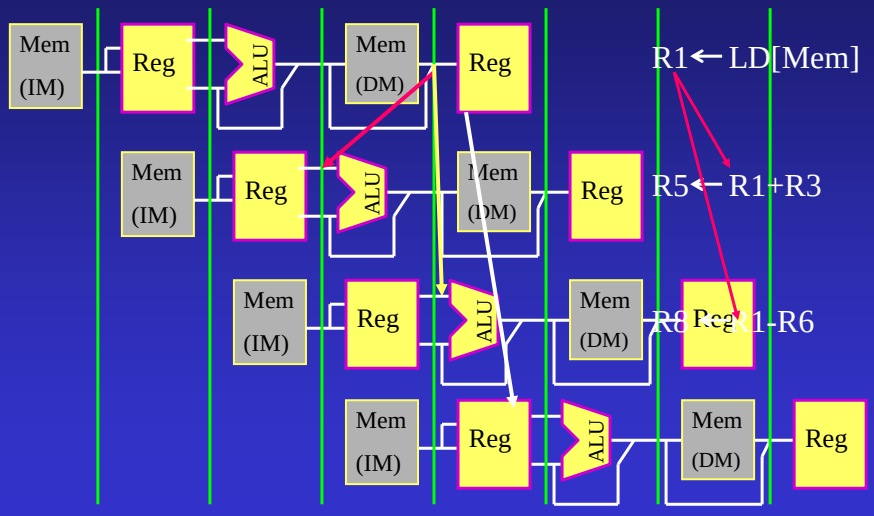


Data Hazard Classification

- RAW (Read After Write)
  - **w/ forward only load presents a problem**
- WAW
- WAR
- RAR

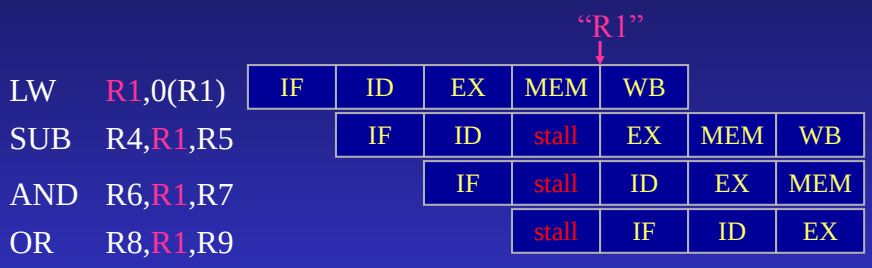
j:	R1 ←
k:	RY ← R1
j:	R1 ←
k:	R1 ←
j:	← R1
k:	R1 ←
j:	← R1
k:	← R1

Data Forwarding (load)



31

Data hazard (load)



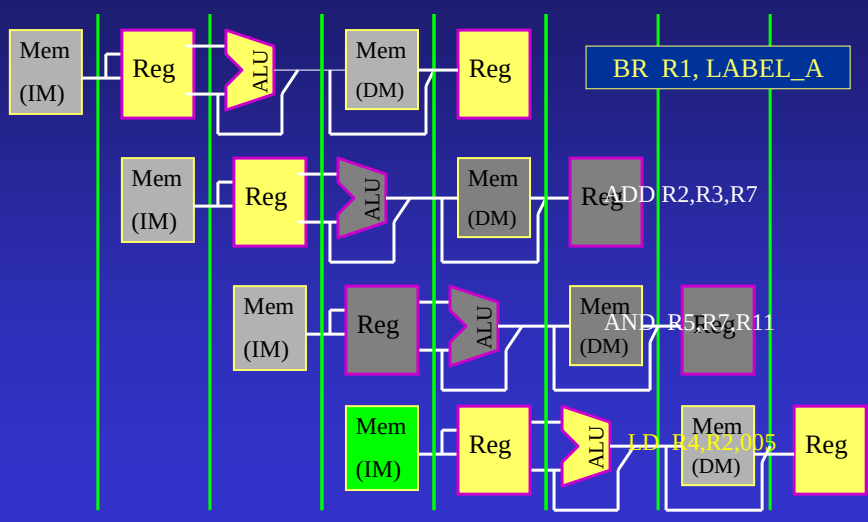
32

Branch

```
          BR  R1, LABEL_A
          ADD R2,R3,R7
          AND  R5,R7,R11
          :
          :
LABEL_A:  LD  R4,R2,005
```

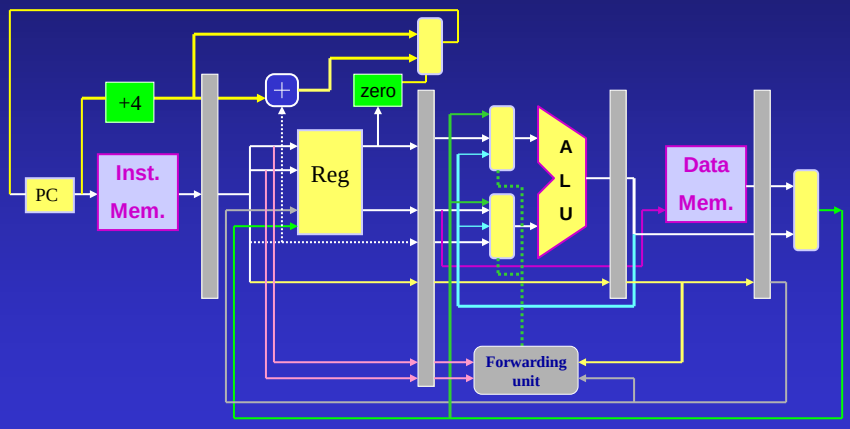
33

Branch



34

Datapath w/ pipeline



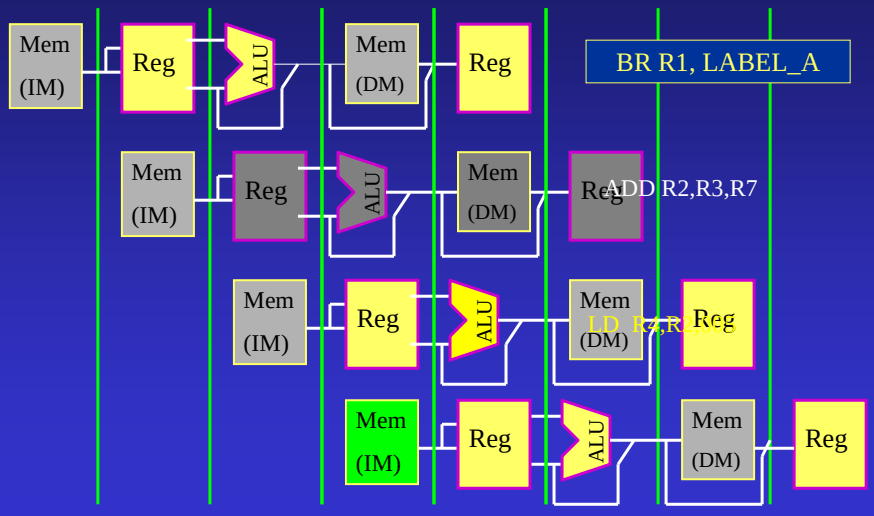
35

What to do w/ branch

- Reduce the number of cycles to decide on a branch.
- Delayed branch (Software Solutions)
  - NO-OP
  - move instructions
    - from before
    - from target
    - from fall through

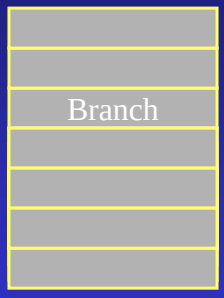
36

Branch



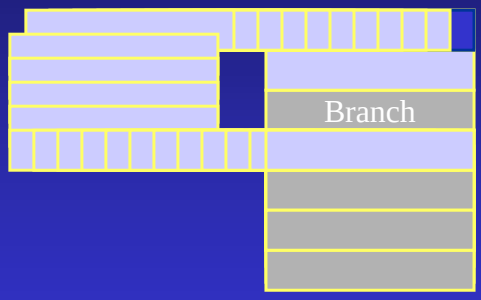
37

NO-OP



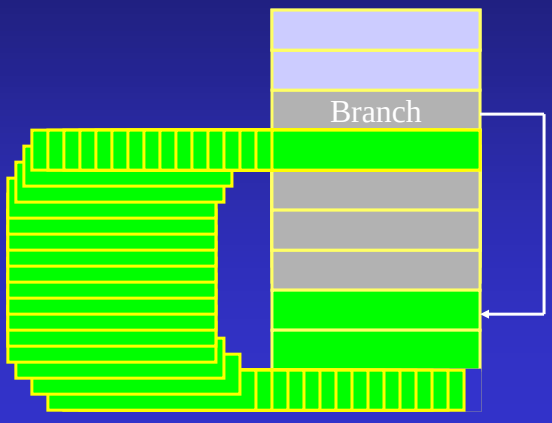
38

From Before



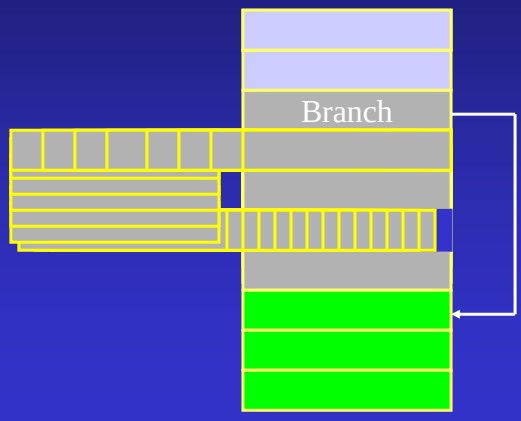
39

From Target



40

From Fall Through



41

Example

loop:	LW	$1,0($2)	R1 ← MEM[R2+0]
	ADDI	$1,$1, 1	R1 ← R1+1
	SW	$1,0($2)	MEM[R2+0] ← R1
	ADDI	$2,$2,4	R2 ← R2+4
	SUB	$4,$3,$2	R4 ← R3-R2
	BNEZ	$4,loop	IF R4 ≠ 0 GOTO loop

Initial value: R3 = R2+396

42