Review Problem 28

- Given what we know about pipelining, assume in a widget factory it takes 40 minutes to make 1 widget. If we pipeline the process into S stages, how long will it take to make N widgets?

† assumes all stages the same length

- Time taken by each stage?

$40/S$ †

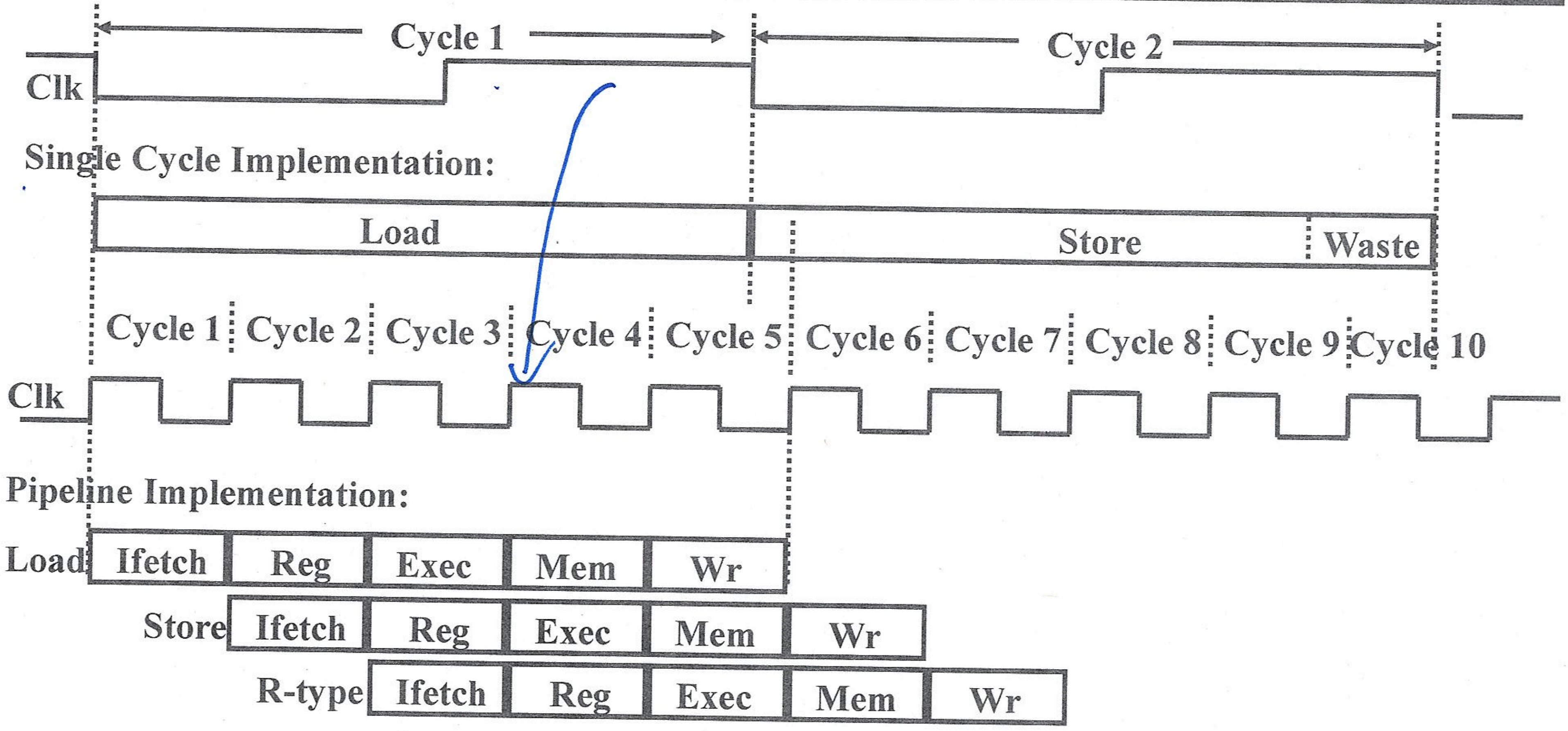
- Time to finish first stage for N widgets?

$N \times 40/S$

- Time to finish all N widgets?

$$\underbrace{N \times 40/S}_{} + \underbrace{(S-1) \times 40/S}_{\text{Drain Time}}$$

Single Cycle vs. Pipeline



Cycle 1

Cycle 2

Clk

Single Cycle Implementation:

Load

Store

Waste

Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5 Cycle 6 Cycle 7 Cycle 8 Cycle 9 Cycle 10

Clk

Pipeline Implementation:

Load Ifetch Reg Exec Mem Wr

Store Ifetch Reg Exec Mem Wr

R-type Ifetch Reg Exec Mem Wr

Why Pipeline?

Suppose we execute 100 instructions

Single Cycle Machine

45 ns/cycle x 1 CPI x 100 inst = 4,500 ns

Ideal pipelined machine

10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1,040 ns

104

CPI for Pipelined Processors

Ideal pipelined machine

10 ns/cycle x (1 **CPI** x 100 inst + 4 cycle drain) = _____ ns

CPI in pipelined processor is “issue rate”. Ignore fill/drain, ignore latency.

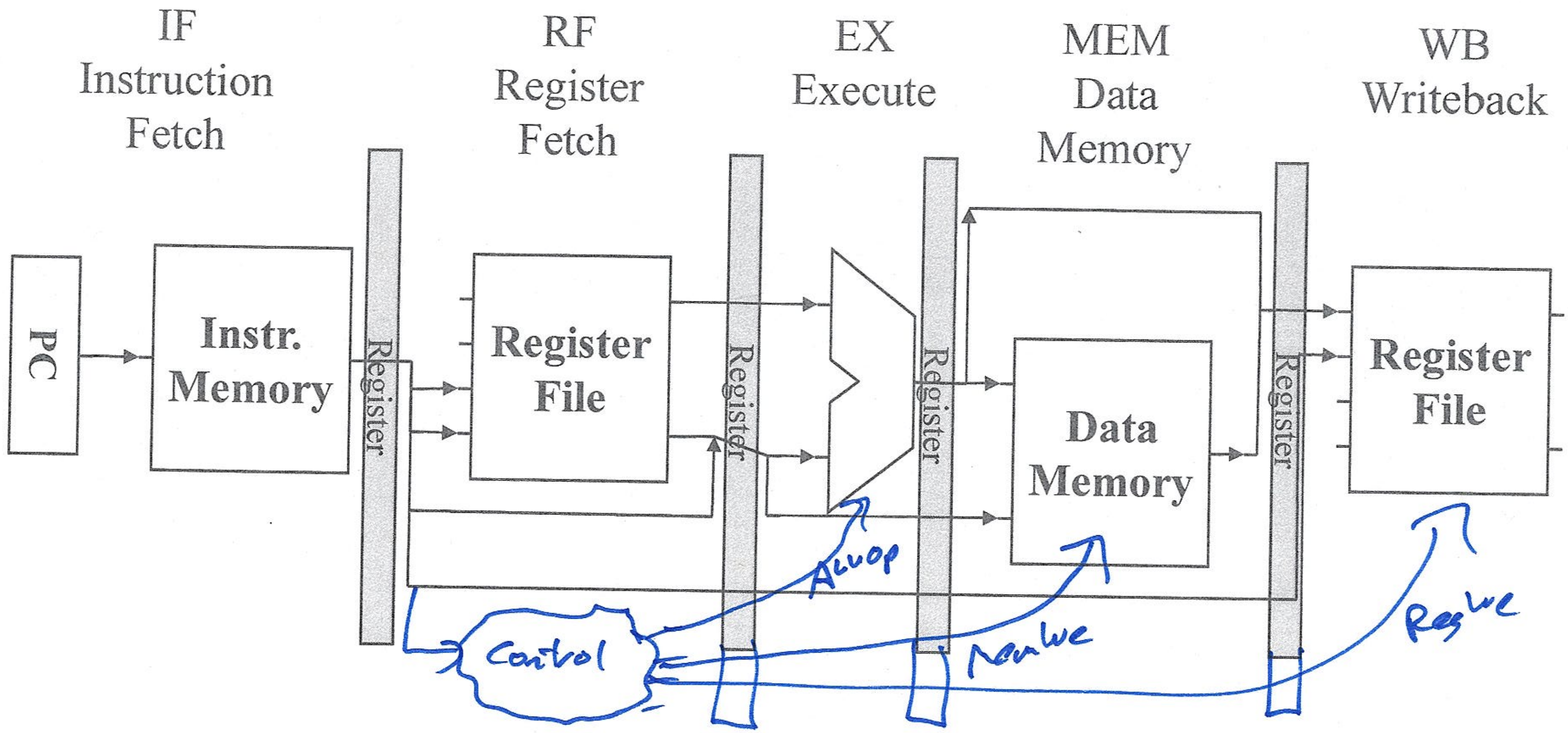
Example: A processor wastes 2 cycles after every branch, and 1 after every load, during which it cannot issue a new instruction. If a program has 10% branches and 30% loads, what is the CPI on this program?

$$\begin{array}{lccccc} & \underline{\text{branch}} & & \underline{\text{load}} & & \underline{\text{others}} \\ & 10\% \times (1+2) & + & 30\%(1+1) & + & 60\%(1) \\ = & 0.3 & + & 0.6 & + & 0.6 \\ = & 1.5 & & & & \end{array}$$

Pipelined Datapath

Slow: mem, RF, adder, ALU, control
Fast: registers, muxes, etc..
Never do 2 slow things in series within a cycle

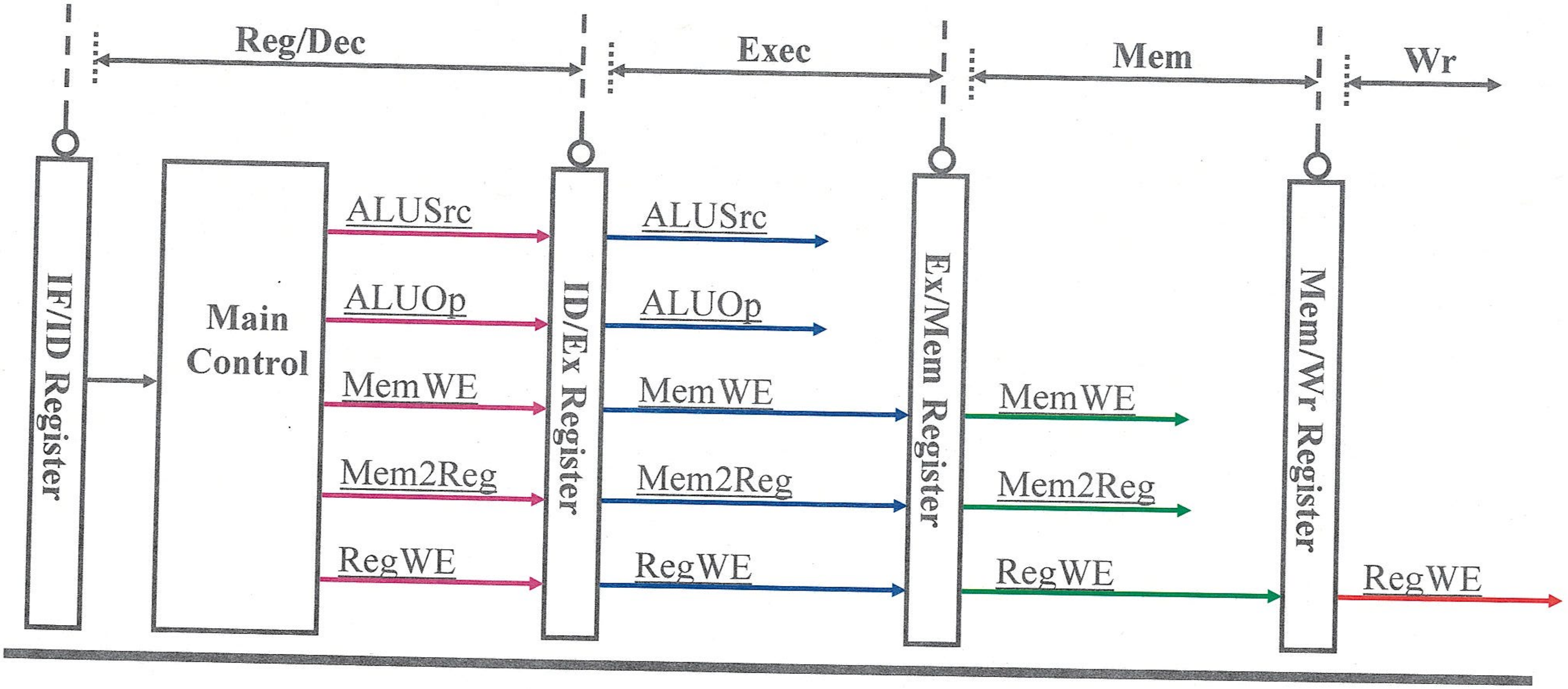
Divide datapath into multiple pipeline stages



Pipelined Control

The Main Control generates the control signals during Reg/Dec

- Control signals for Exec (ALUOp, ALUSrc, …) are used 1 cycle later
- Control signals for Mem (MemWE, Mem2Reg, …) are used 2 cycles later
- Control signals for Wr (RegWE, …) are used 3 cycles later



Can pipelining get us into trouble?

Yes: **Pipeline Hazards**

- **structural hazards**: attempt to use the same resource two different ways at the same time
  - E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)
- **data hazards**: attempt to use item before it is ready
  - E.g., one sock of pair in dryer and one in washer; can't fold until get sock from washer through dryer
  - instruction depends on result of prior instruction still in the pipeline
- **control hazards**: attempt to make decision before condition evaluated
  - E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in
  - branch instructions

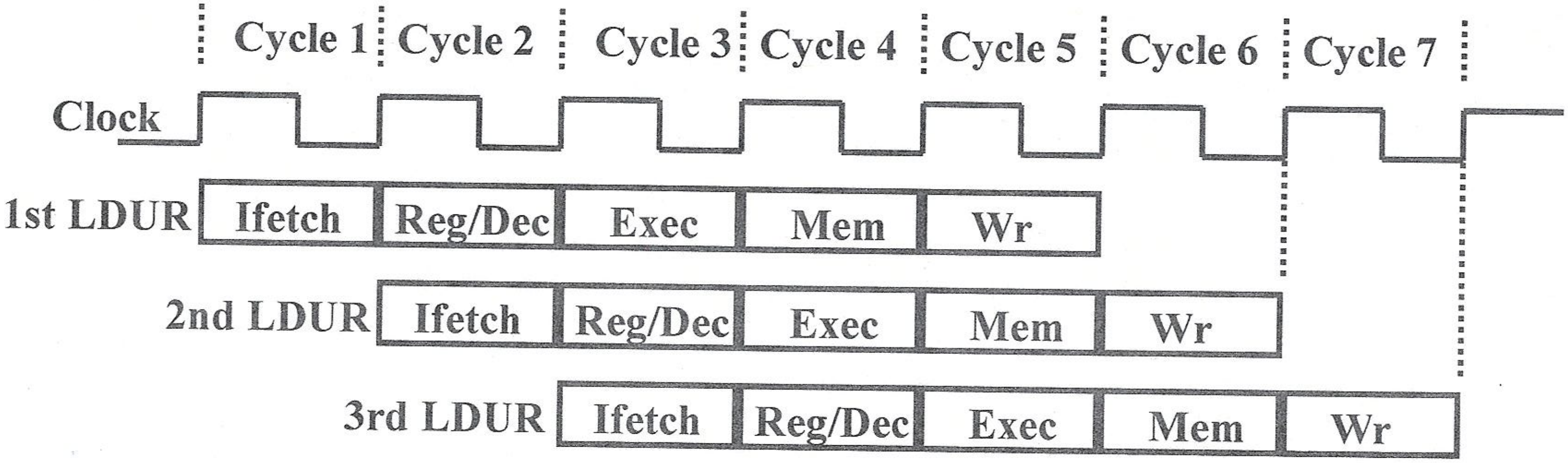
Can always resolve hazards by **waiting**

- pipeline control must detect the hazard
- take action (or delay action) to resolve hazards

Pipelining the Load Instruction

The five independent functional units in the pipeline datapath are:

- Instruction Memory for the Ifetch stage
- Register File's Read ports (bus A and busB) for the Reg/Dec stage
- ALU for the Exec stage
- Data Memory for the Mem stage
- Register File's Write port (bus W) for the Wr stage



	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
Clock							
1st LDUR	Ifetch	Reg/Dec	Exec	Mem	Wr		
2nd LDUR		Ifetch	Reg/Dec	Exec	Mem	Wr	
3rd LDUR			Ifetch	Reg/Dec	Exec	Mem	Wr

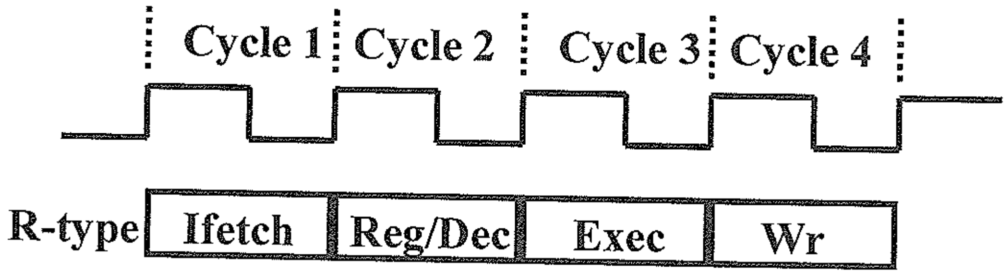
The Four Stages of R-type

Ifetch: Fetch the instruction from the Instruction Memory

Reg/Dec: Register Fetch and Instruction Decode

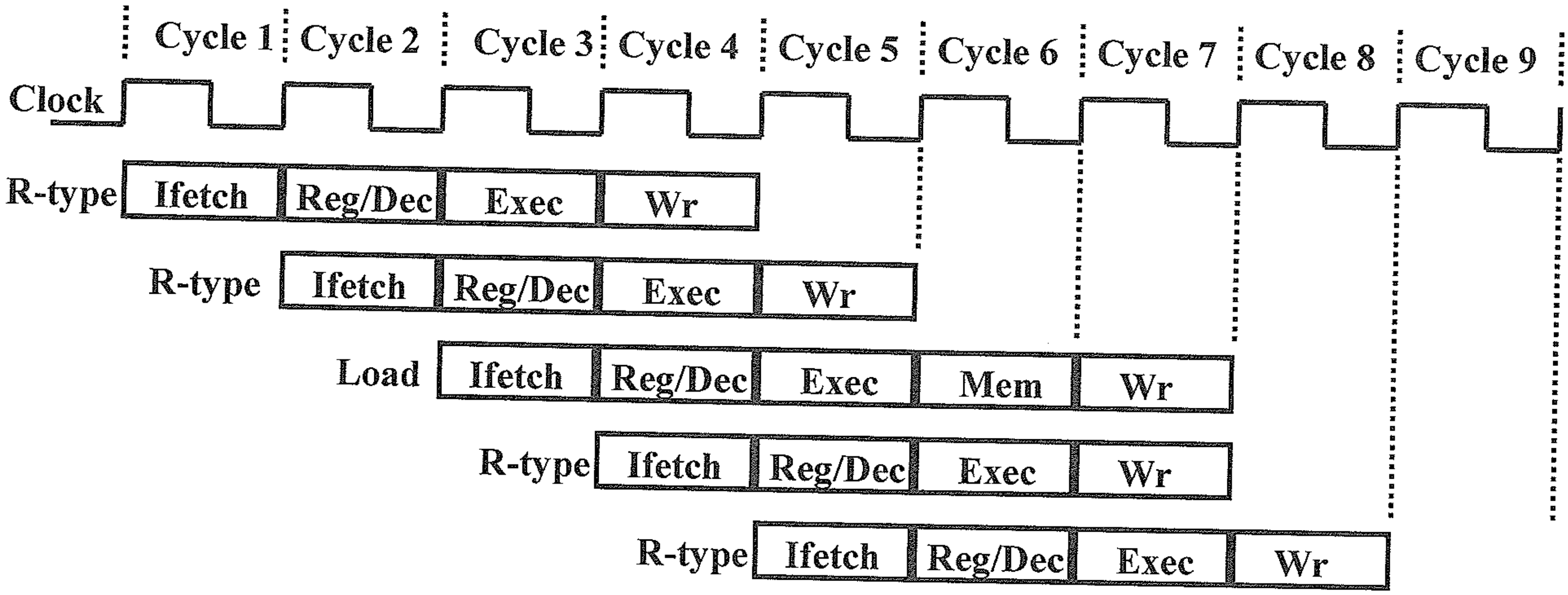
Exec: ALU operates on the two register operands

Wr: Write the ALU output back to the register file



Structural Hazard

Interaction between R-type and loads causes structural hazard on writeback



Important Observation

Each functional unit can only be used once per instruction

Each functional unit must be used at the same stage for all instructions:

Load uses Register File's Write Port during its 5th stage

	1	2	3	4	5
Load	Ifetch	Reg/Dec	Exec	Mem	Wr

R-type uses Register File's Write Port during its 4th stage

	1	2	3	4
R-type	Ifetch	Reg/Dec	Exec	Wr

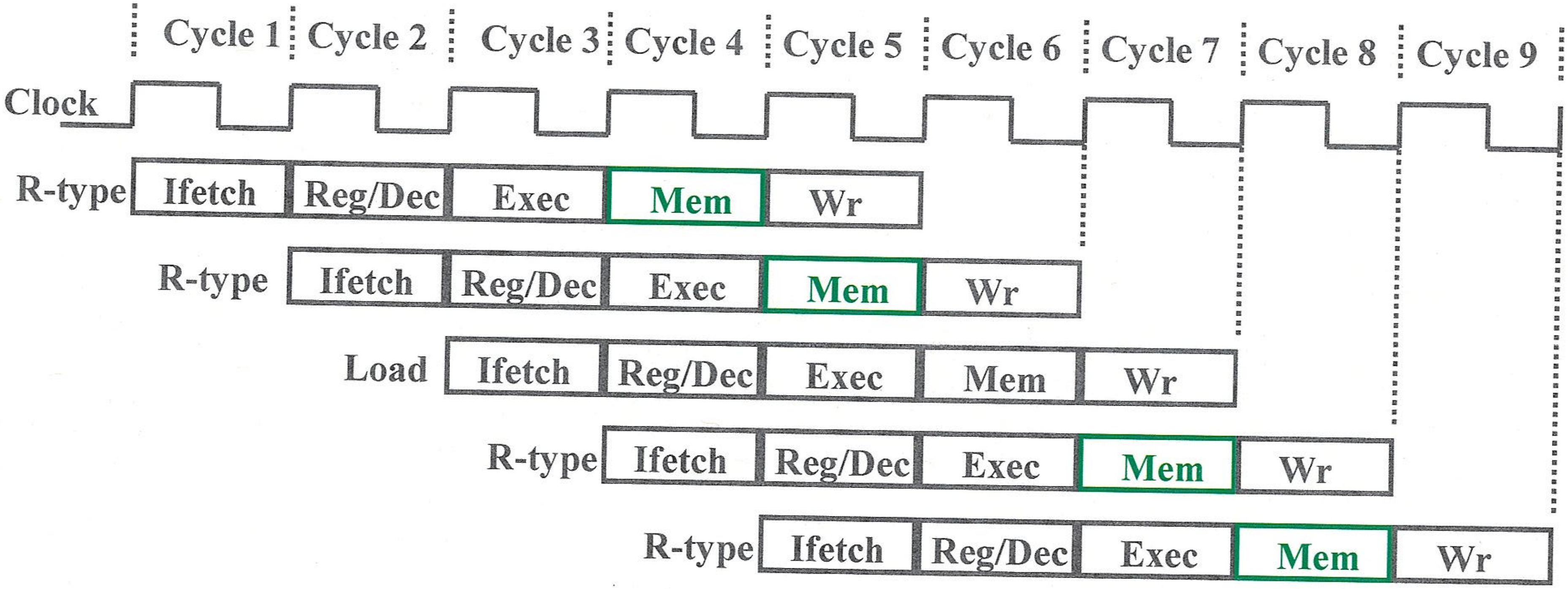
Solution: Delay R-type's register write by one cycle:

Now R-type instructions also use Reg File's write port at Stage 5

Mem stage is a NOOP stage: nothing is being done.

	1	2	3	4	5
R-type	Ifetch	Reg/Dec	Exec	Mem	Wr

Pipelining the R-type Instruction



	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8	Cycle 9
Clock									
R-type	Ifetch	Reg/Dec	Exec	Mem	Wr				
R-type		Ifetch	Reg/Dec	Exec	Mem	Wr			
Load			Ifetch	Reg/Dec	Exec	Mem	Wr		
R-type				Ifetch	Reg/Dec	Exec	Mem	Wr	
R-type					Ifetch	Reg/Dec	Exec	Mem	Wr

The Four Stages of Store

Ifetch: Fetch the instruction from the Instruction Memory

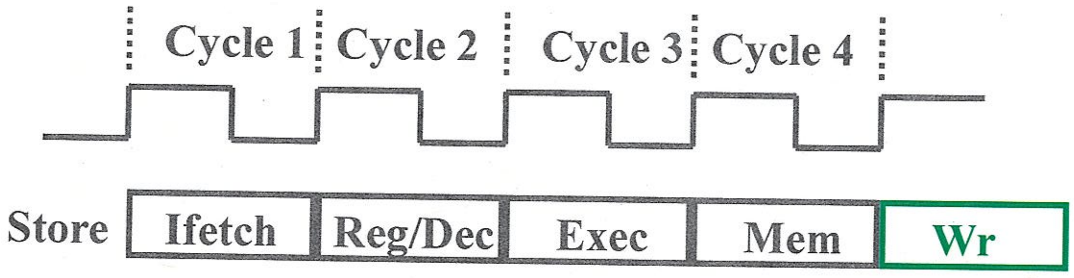
Reg/Dec: Register Fetch and Instruction Decode

Exec: Calculate the memory address

Mem: Write the data into the Data Memory

Wr: NOOP

Compatible with Load & R-type instructions


Cycle 1
Cycle 2
Cycle 3
Cycle 4
Store
Ifetch
Reg/Dec
Exec
Mem
Wr