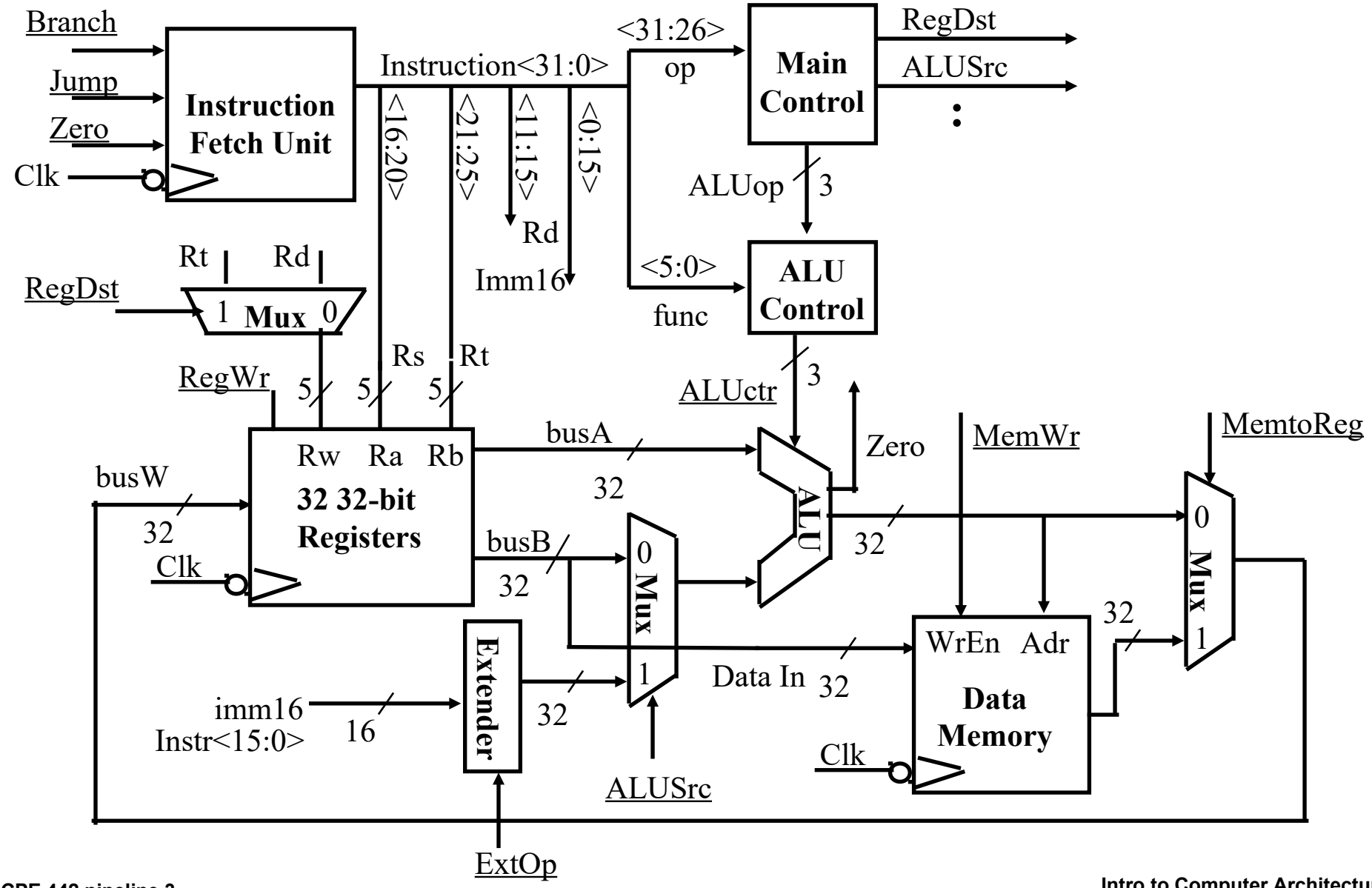
CpE 242
Computer Architecture and Engineering
Designing a Pipeline Processor

Outline of Today's Lecture

- Recap and Introduction (5 minutes)
- Introduction to the Concept of Pipelined Processor (15 minutes)
- Pipelined Datapath and Pipelined Control (25 minutes)
- How to Avoid Race Condition in a Pipeline Design? (5 minutes)
- Pipeline Example: Instructions Interaction (15 minutes)
- Summary (5 minutes)

A Single Cycle Processor



Drawbacks of this Single Cycle Processor

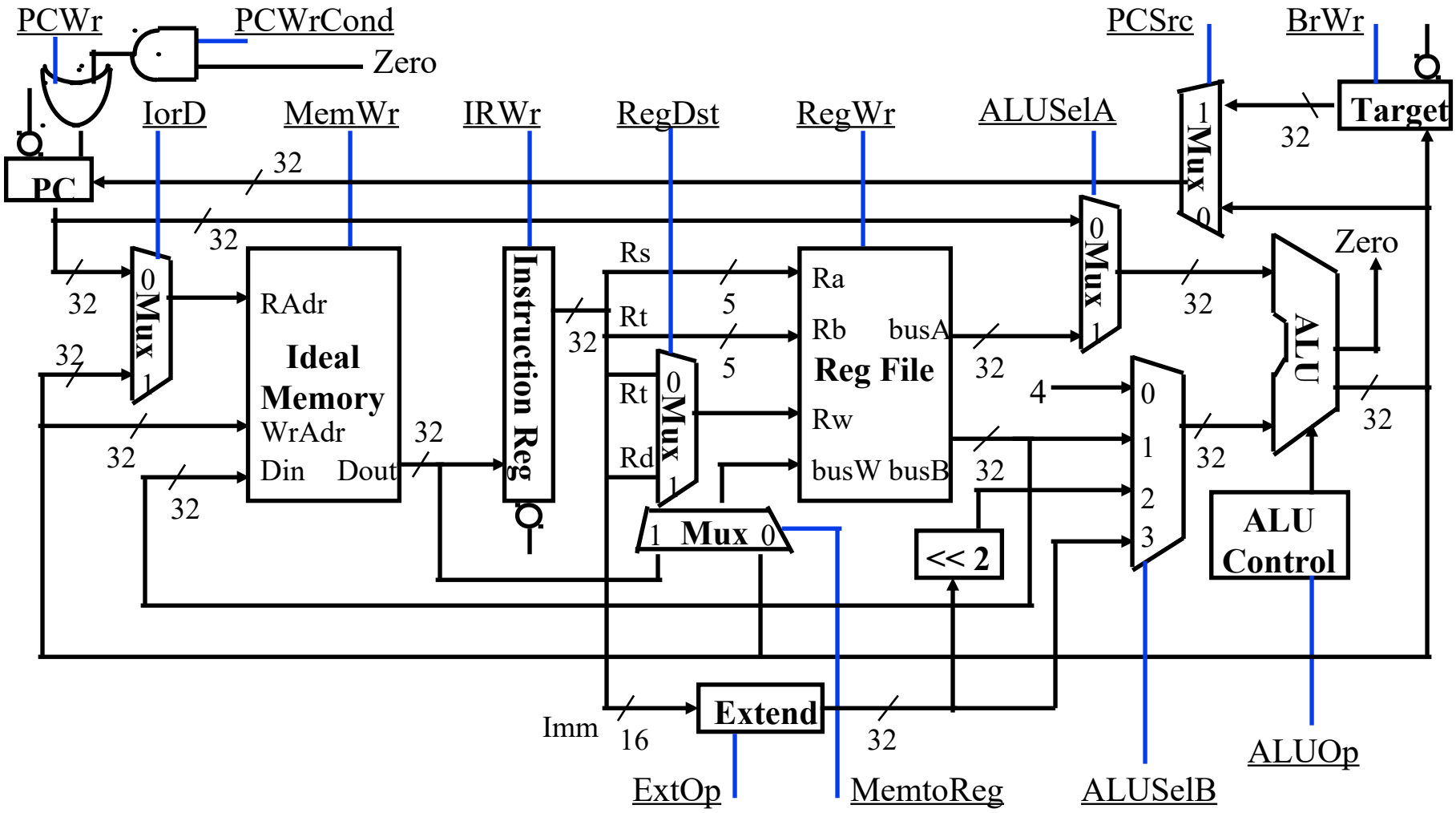
- Long cycle time:
  - Cycle time must be long enough for the load instruction:
    - PC's Clock -to-Q +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time +
    - Clock Skew
- Cycle time is much longer than needed for all other instructions. Examples:
  - R-type instructions do not require data memory access
  - Jump does not require ALU operation nor data memory access

Overview of a Multiple Cycle Implementation

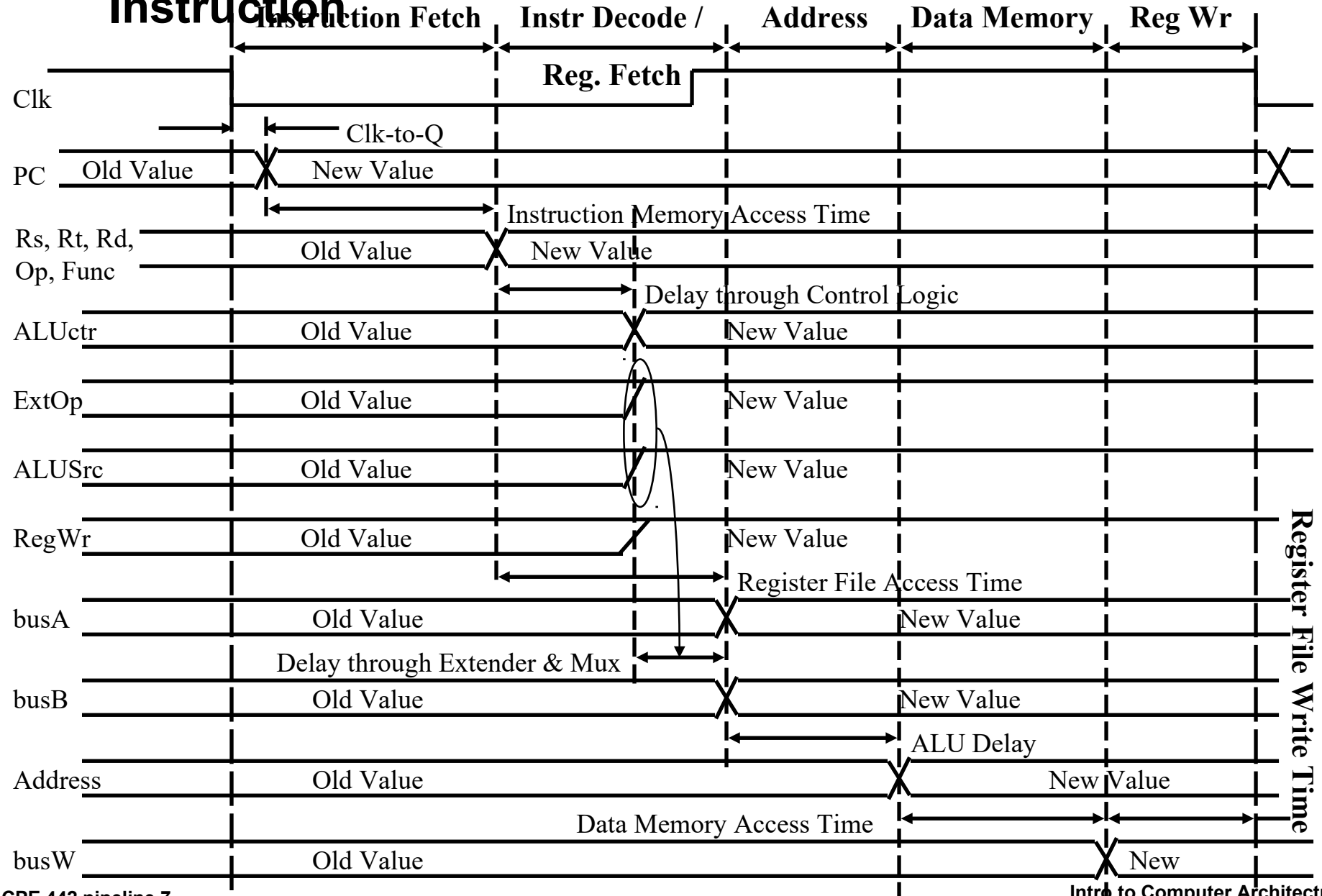
- The root of the single cycle processor's problems:
  - The cycle time has to be long enough for the slowest instruction
- Solution:
  - Break the instruction into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
    - Cycle time: time it takes to execute the longest step
    - Keep all the steps to have similar length
  - This is the essence of the multiple cycle processor
- The advantages of the multiple cycle processor:
  - Cycle time is much shorter
  - Different instructions take different number of cycles to complete
    - Load takes five cycles
    - Jump only takes three cycles
  - Allows a functional unit to be used more than once per instruction

Multiple Cycle Processor

° MCP: A functional unit to be used more than once per instruction



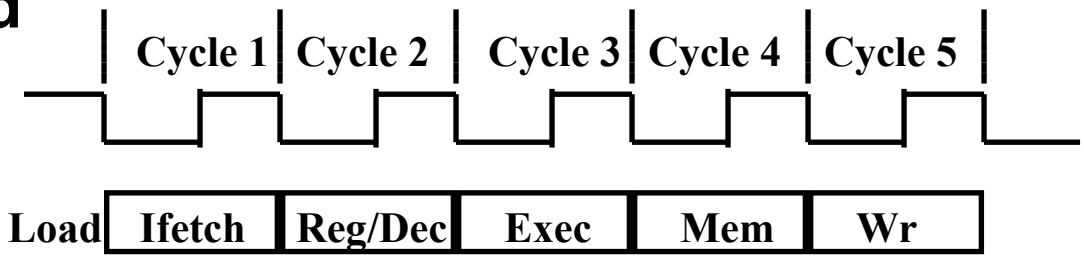
Timing Diagram of a Load Instruction



Outline of Today's Lecture

- Recap and Introduction (5 minutes)
- Introduction to the Concept of Pipelined Processor (15 minutes)
- Pipelined Datapath and Pipelined Control (25 minutes)
- How to Avoid Race Condition in a Pipeline Design? (5 minutes)
- Pipeline Example: Instructions Interaction (15 minutes)
- Summary (5 minutes)

The Five Stages of Load



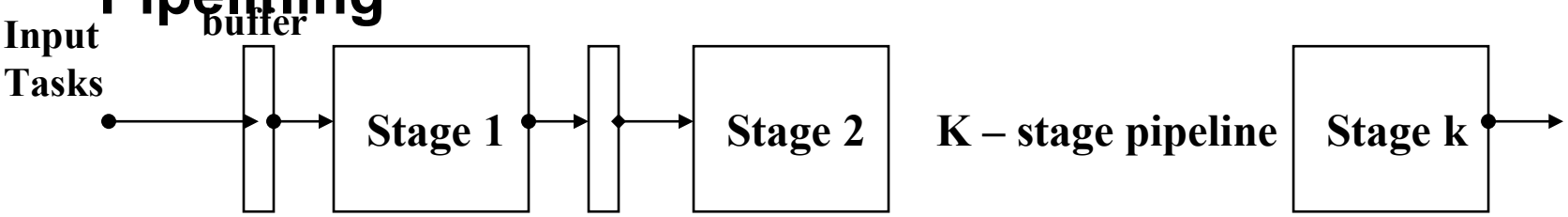
- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Calculate the memory address
- Mem: Read the data from the Data Memory
- Wr: Write the data back to the register file

Key Ideas Behind Pipelining

- Grading the Final exam for a class of 100 students:
  - 5 problems, five people grading the exam
  - Each person ONLY grade one problem
  - Pass the exam to the next person as soon as one finishes his part
  - Assume each problem takes 12 min to grade
    - Each individual exam still takes 1 hour to grade
    - But with 5 people, all exams can be graded five times quicker

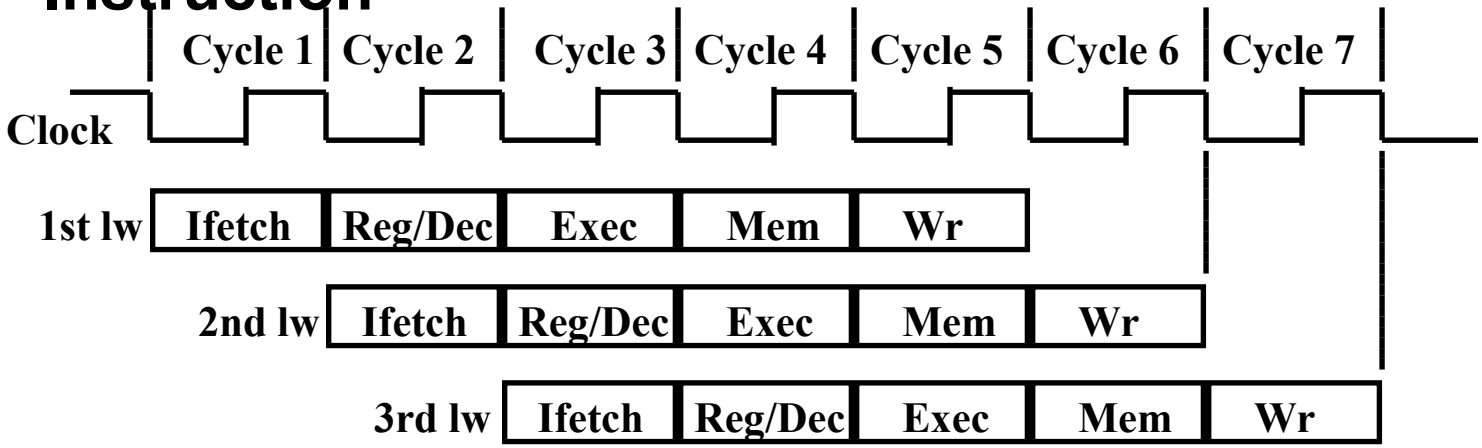
- The load instruction has 5 stages:
  - Five independent functional units to work on each stage
    - Each functional unit is used only once
  - The 2nd load can start as soon as the 1st finishes its Ifetch stage
  - Each load still takes five cycles to complete
  - The throughput, however, is much higher

Key Ideas Behind Pipelining



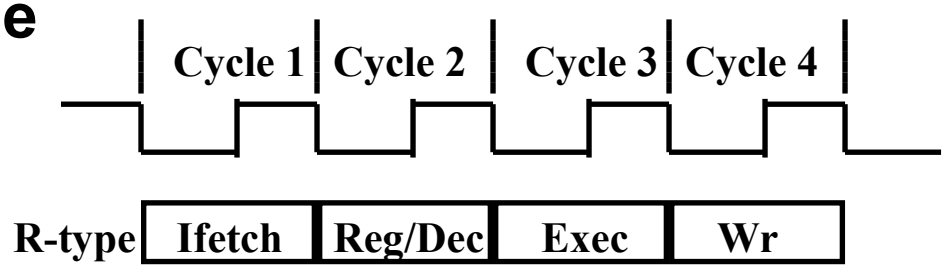
- Let n be number of tasks or exams (or instructions)
- Let k be number of stages for each task
- Let T be the time per stage
- Time per task = T . k
- Total Time per n tasks for non-pipelined solution = T . k . n
- Total Time per n tasks for pipelined solution = T . k + T . (n-1)
- Speedup = pipelined perform/ non-pipelined performance
  = Total Time non-pipelined/ Total Time for pipelined
  = k . n / k + n-1 = k approx. when n >> k

Pipelining the Load Instruction



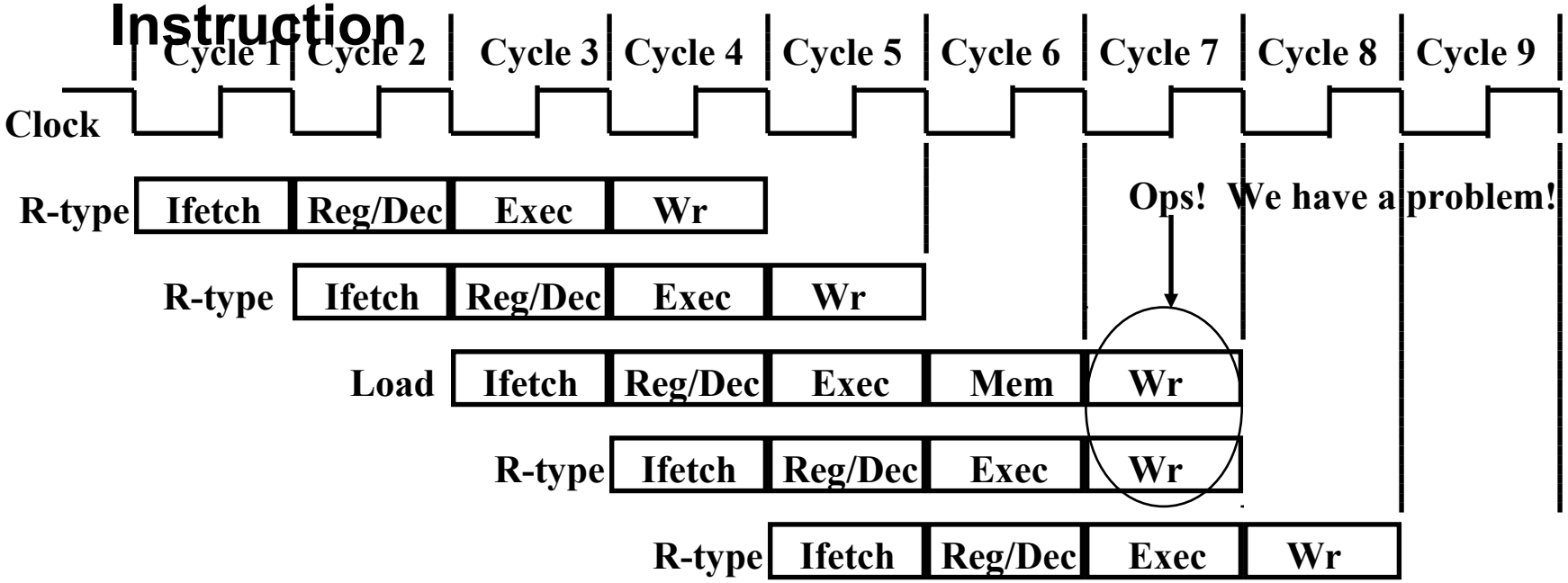
- The five independent functional units in the pipeline datapath are:
  - Instruction Memory for the Ifetch stage
  - Register File's Read ports (bus A and busB) for the Reg/Dec stage
  - ALU for the Exec stage
  - Data Memory for the Mem stage
  - Register File's Write port (bus W) for the Wr stage
- One instruction enters the pipeline every cycle
  - One instruction comes out of the pipeline (complete) every cycle
  - The "Effective" Cycles per Instruction (CPI) is 1

The Four Stages of R-type



- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: ALU operates on the two register operands
- Wr: Write the ALU output back to the register file

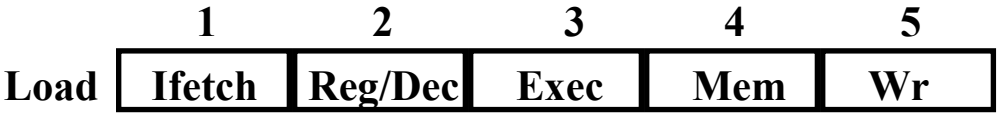
Pipelining the R-type and Load Instruction



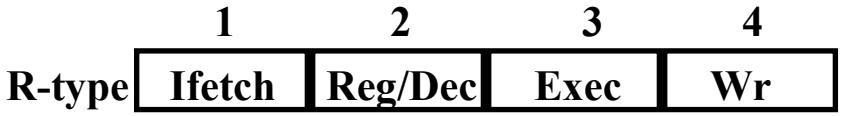
- We have a problem:
  - Two instructions try to write to the register file at the same time!

Important Observation

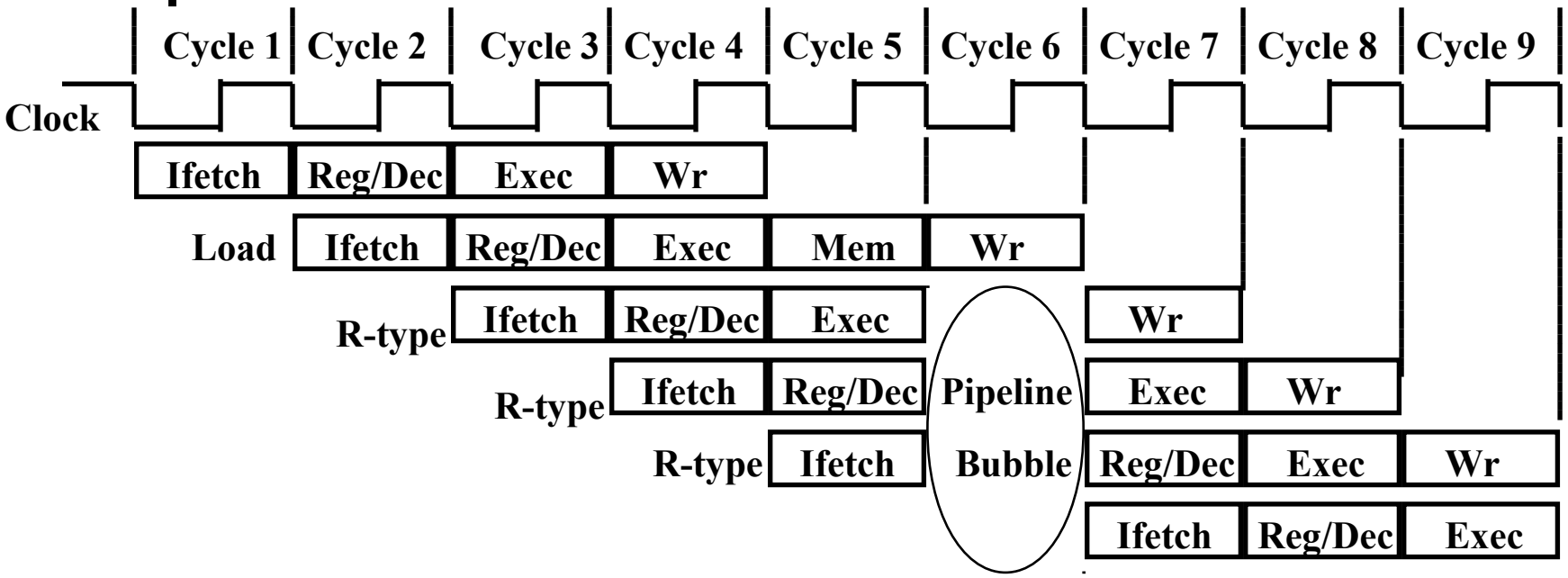
- Each functional unit can only be used once per instruction
- Each functional unit must be used at the same stage for all instructions:
  - Load uses Register File's Write Port during its 5th stage



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



Solution 1: Insert "Bubble" into the Pipeline

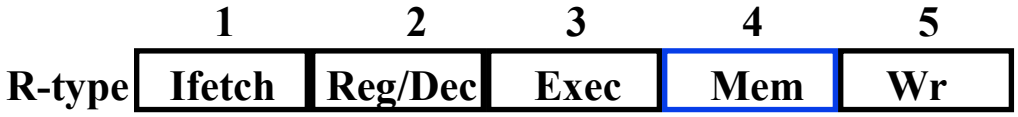


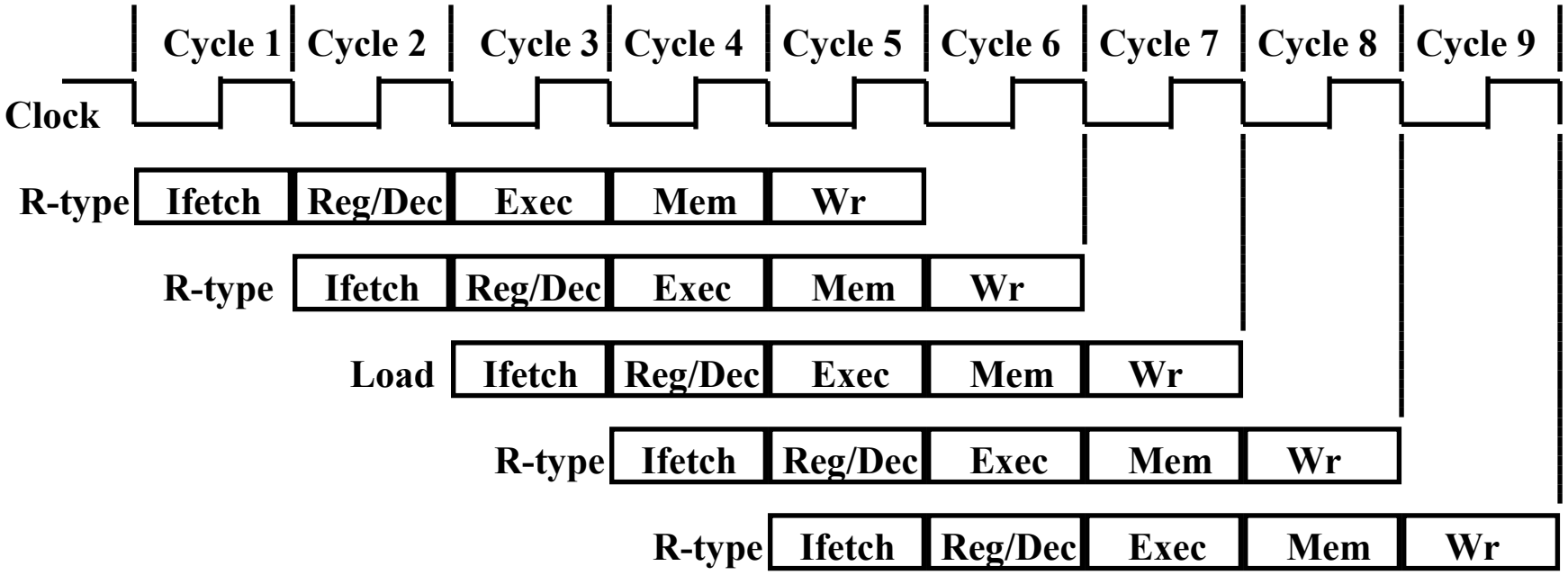
- Insert a "bubble" into the pipeline to prevent 2 writes at the same cycle
  - The control logic can be complex
- No instruction is completed during Cycle 5:
  - The "Effective" CPI for load is 2

Solution 2: Delay R-type's Write by One Cycle

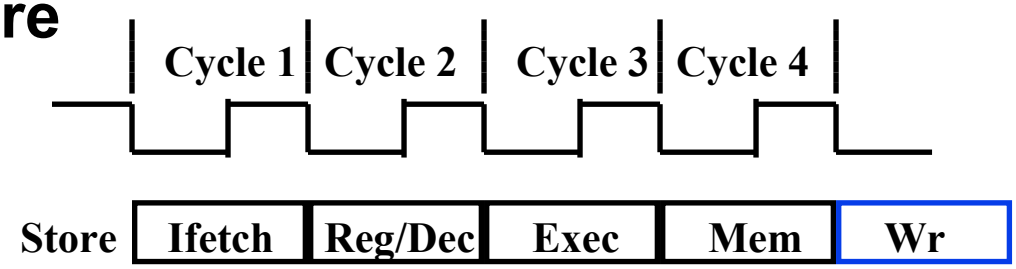
° Delay R-type's register write by one cycle:

- Now R-type instructions also use Reg File's write port at Stage 5
- Mem stage is a NOOP stage: nothing is being done



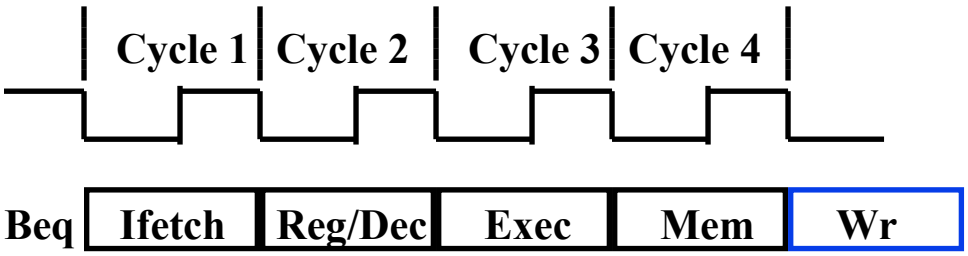


The Four Stages of Store



- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Calculate the memory address
- Mem: Write the data into the Data Memory

The Four Stages of Beq

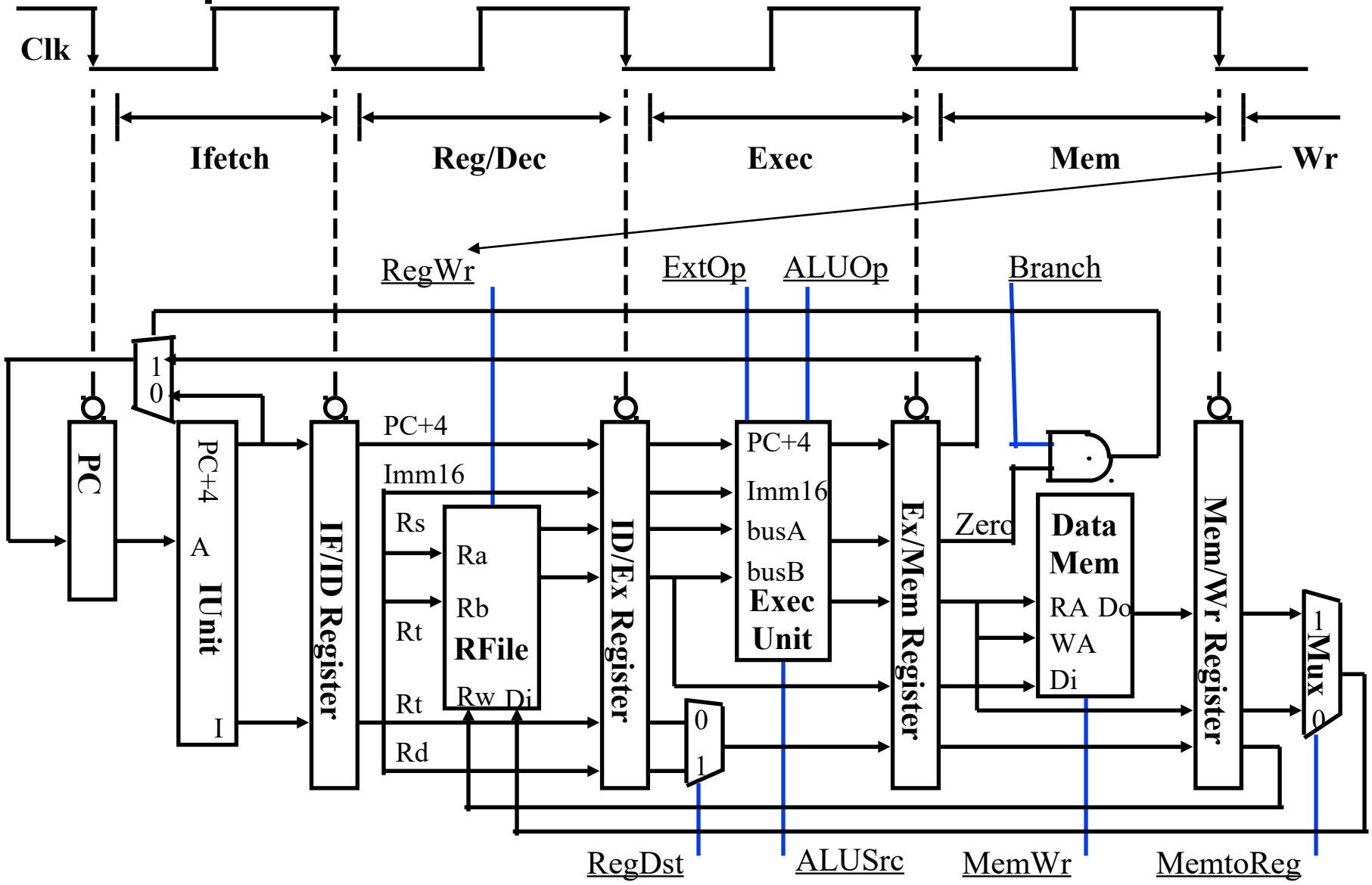


- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: ALU compares the two register operands
  - Adder calculates the branch target address
- Mem: If the registers we compared in the Exec stage are the same,
  - Write the branch target address into the PC

Outline of Today's Lecture

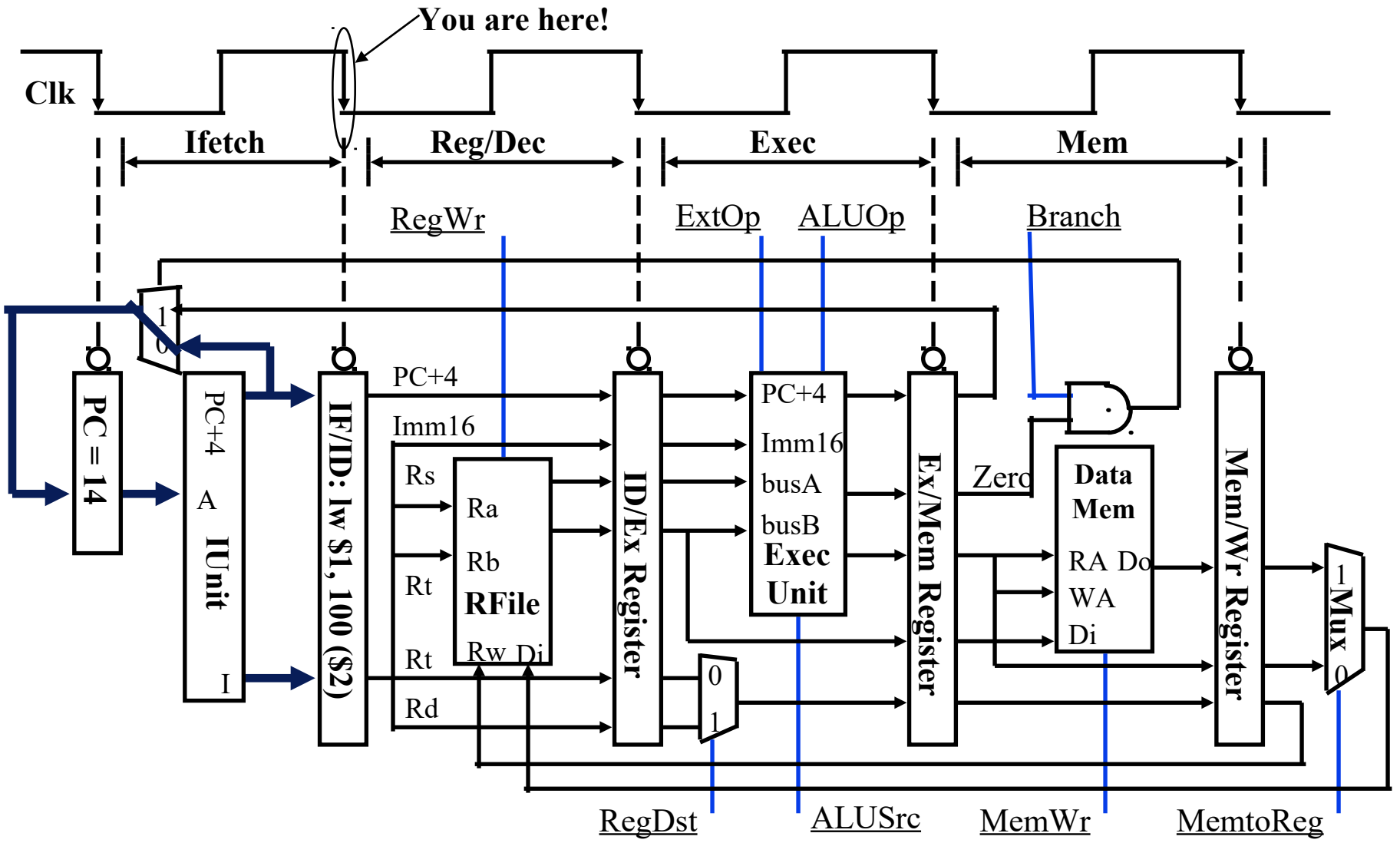
- Recap and Introduction (5 minutes)
- Introduction to the Concept of Pipelined Processor (15 minutes)
- Pipelined Datapath and Pipelined Control
- How to Avoid Race Condition in a Pipeline Design? (5 minutes)
- Pipeline Example: Instructions Interaction (15 minutes)
- Summary (5 minutes)

A Pipelined Datapath



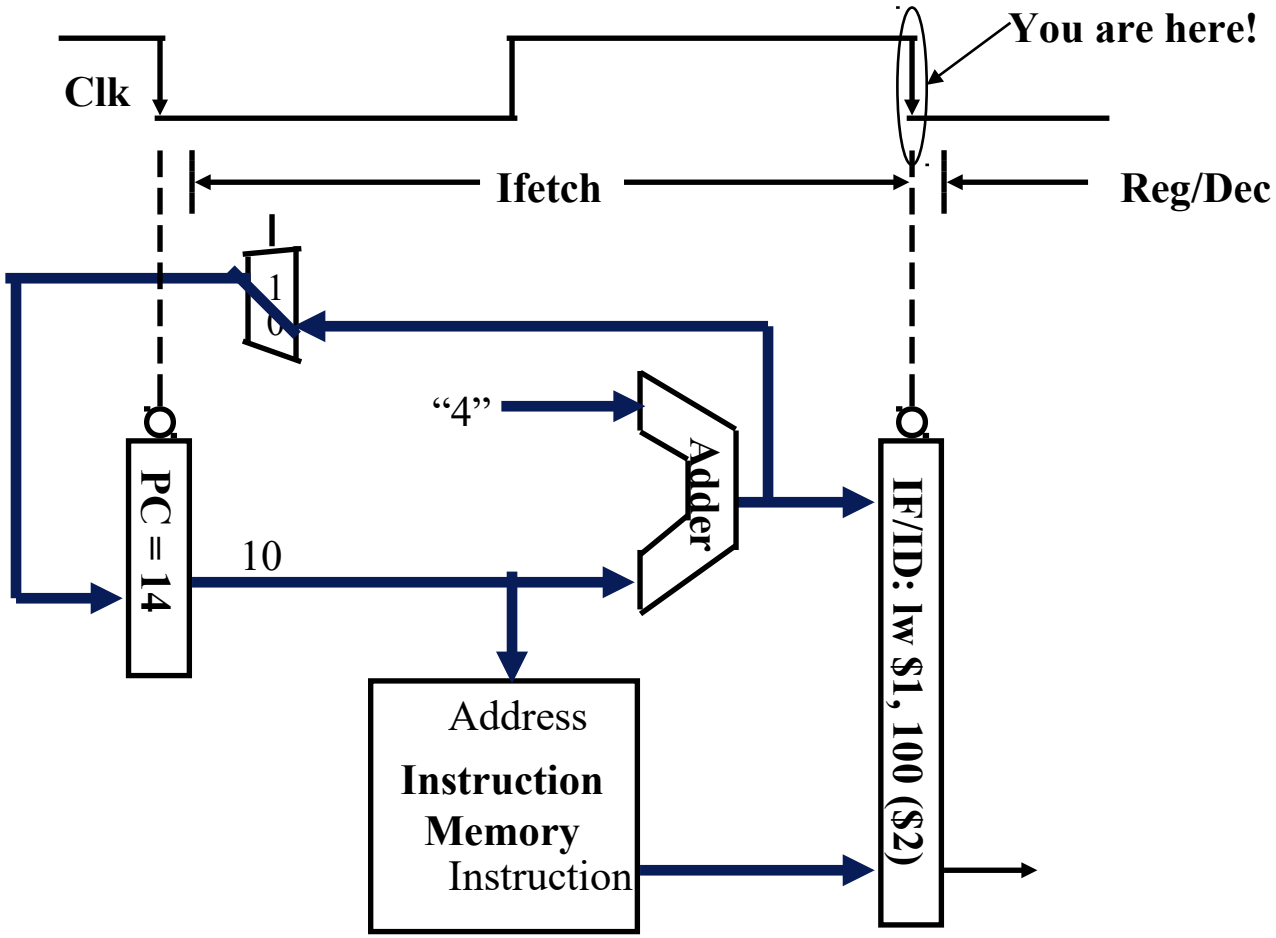
The Instruction Fetch Stage

- Location 10: lw $1, 0x100($2) $1 <- Mem[($2) + 0x100]



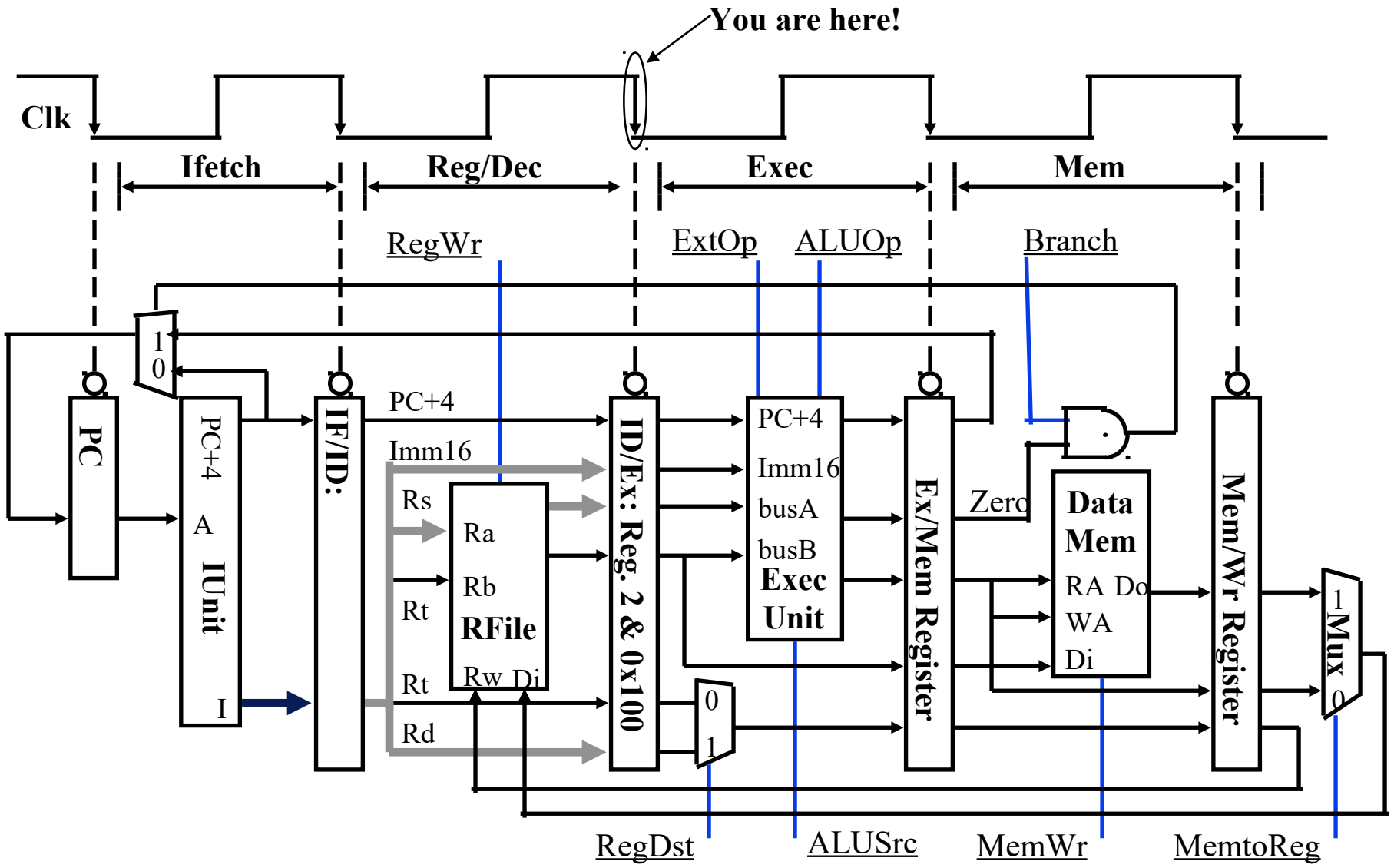
A Detail View of the Instruction Unit

° Location 10: lw $1, 0x100($2)



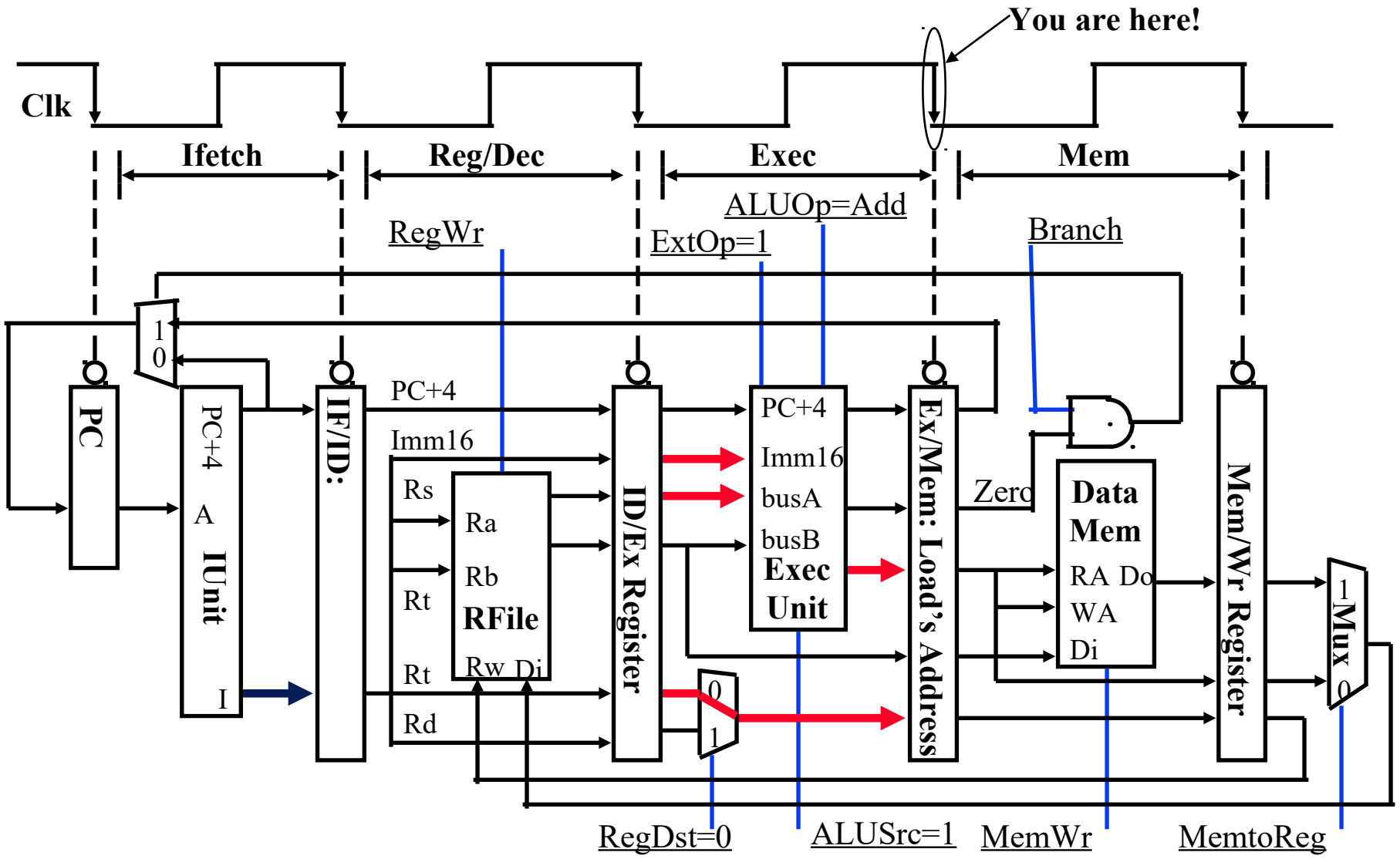
The Decode / Register Fetch Stage

- Location 10: lw $1, 0x100($2) $1 <- Mem[($2) + 0x100]

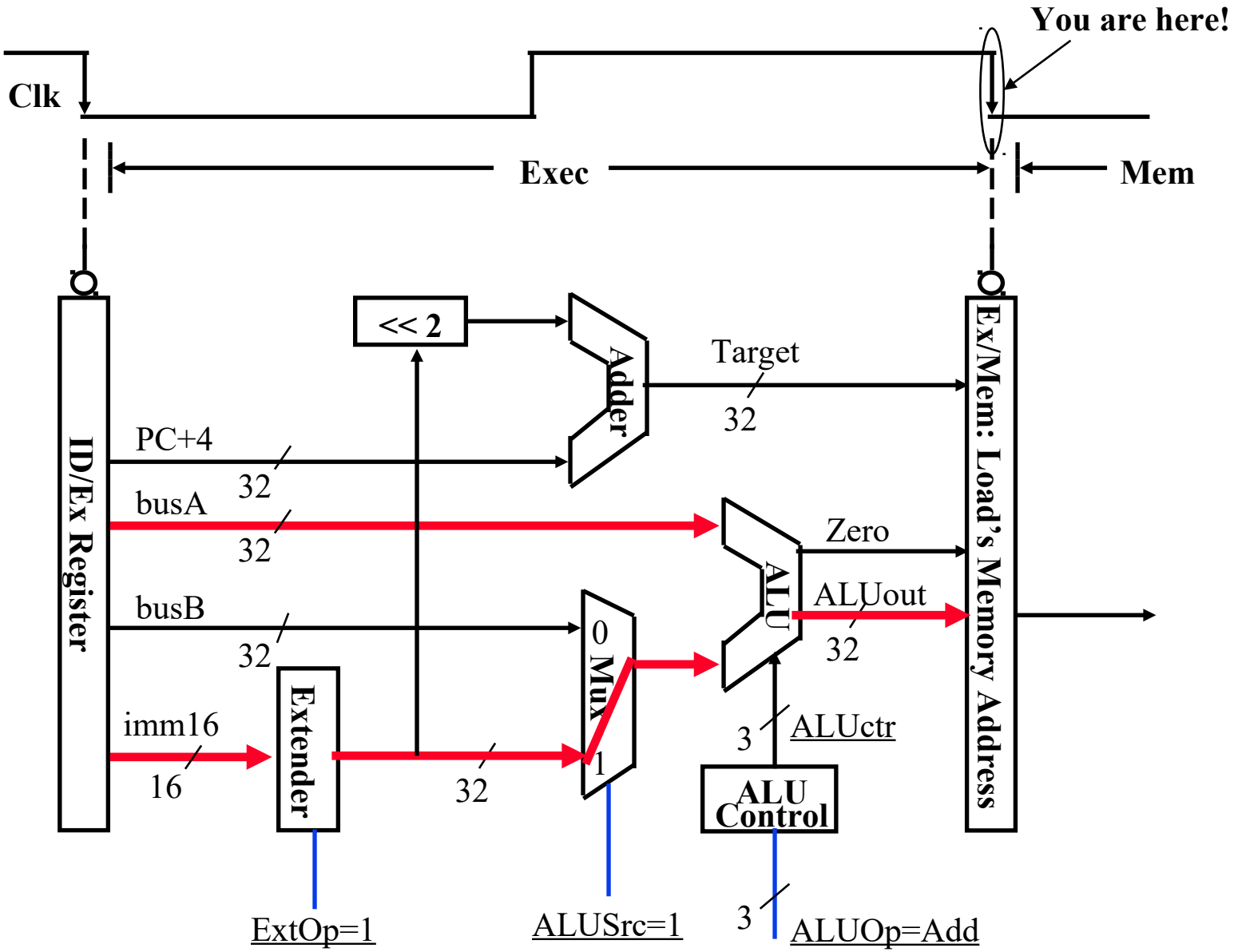


Load's Address Calculation Stage

- Location 10: lw $1, 0x100($2) $1 <- Mem[($2) + 0x100]

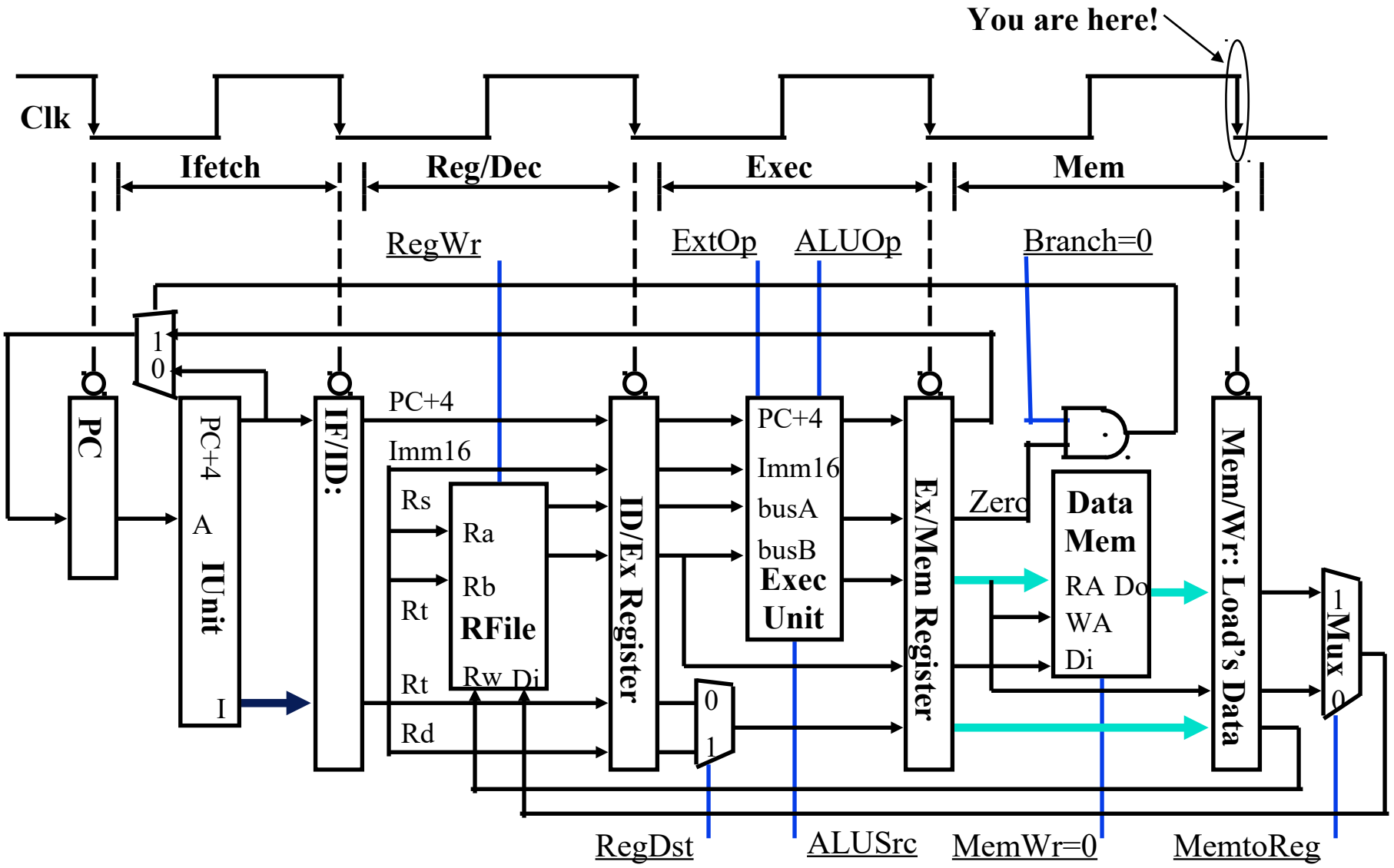


A Detail View of the Execution Unit



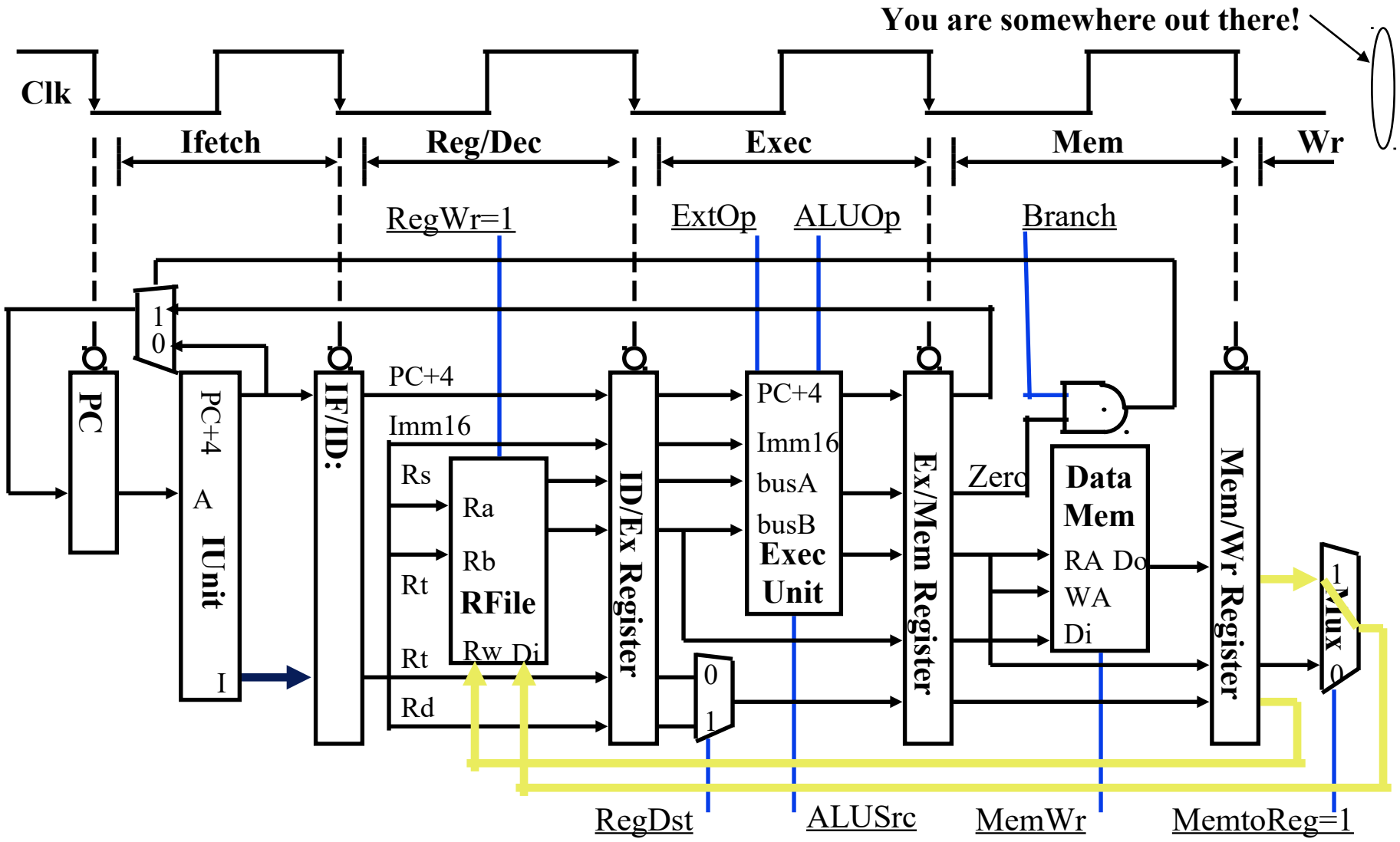
Load's Memory Access Stage

° Location 10: lw $1, 0x100($2) $1 <- Mem[($2) + 0x100]



Load's Write Back Stage

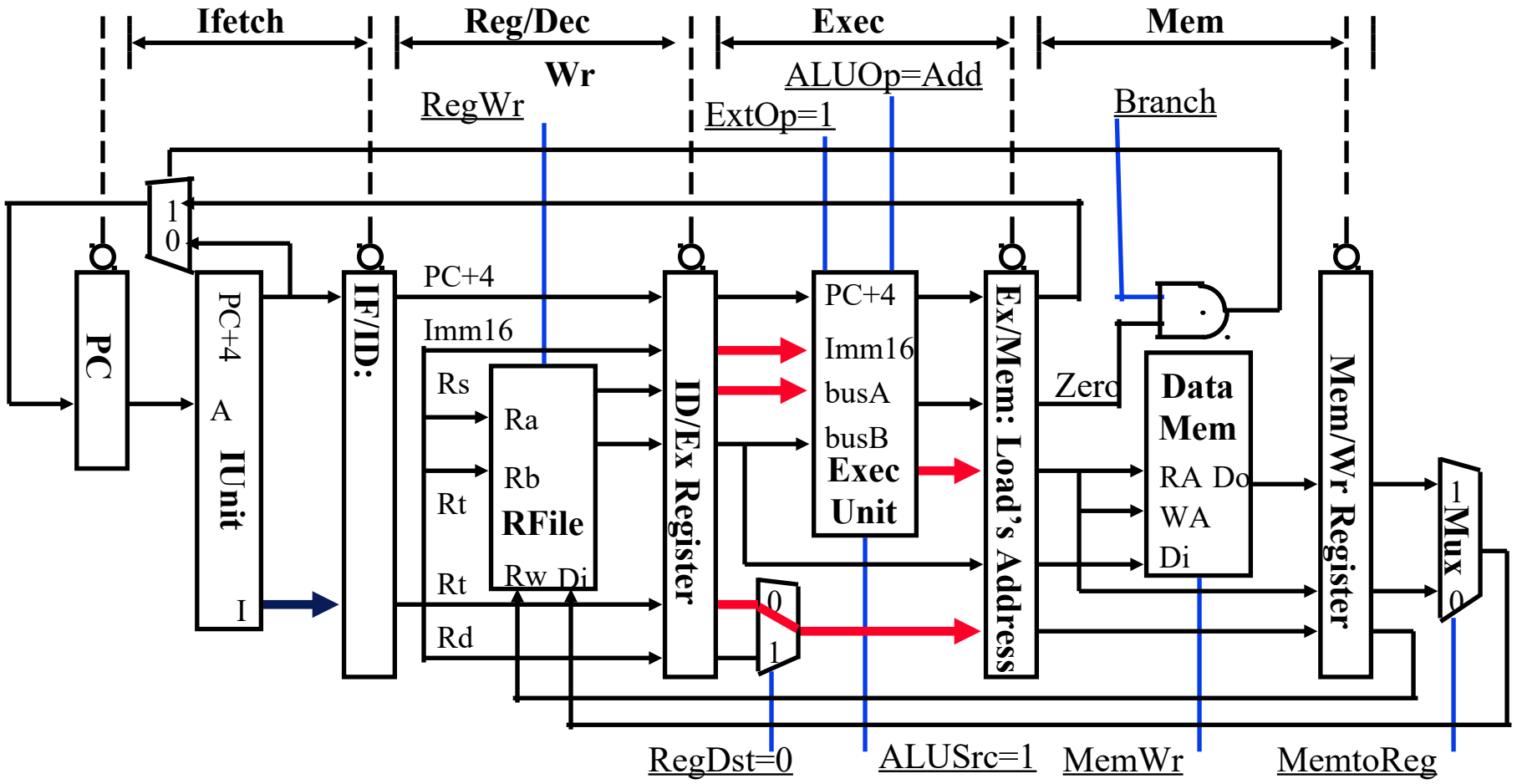
- Location 10: lw $1, 0x100($2) $1 <- Mem[($2) + 0x100]



How About Control Signals?

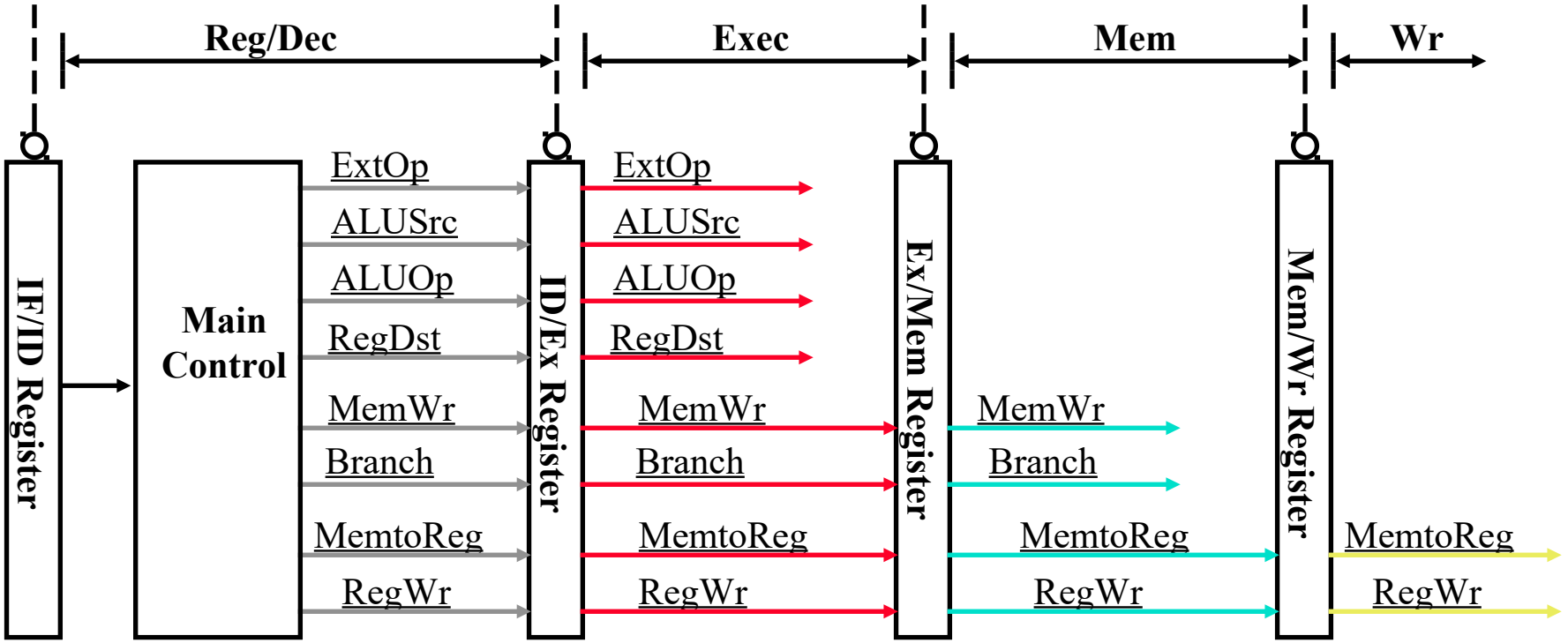
- Key Observation: Control Signals at Stage N = Func (Instr. at Stage N)
  - N = Exec, Mem, or Wr

- Example: Controls Signals at Exec Stage = Func(Load's Exec)



Pipeline Control

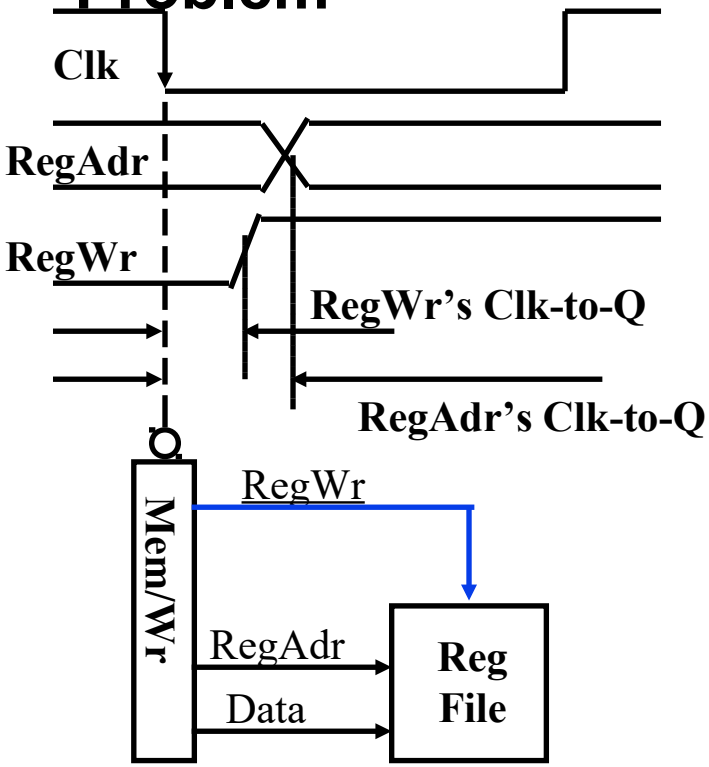
- The Main Control generates the control signals during Reg/Dec
  - Control signals for Exec (ExtOp, ALUSrc, ...) are used 1 cycle later
  - Control signals for Mem (MemWr Branch) are used 2 cycles later
  - Control signals for Wr (MemtoReg MemWr) are used 3 cycles later

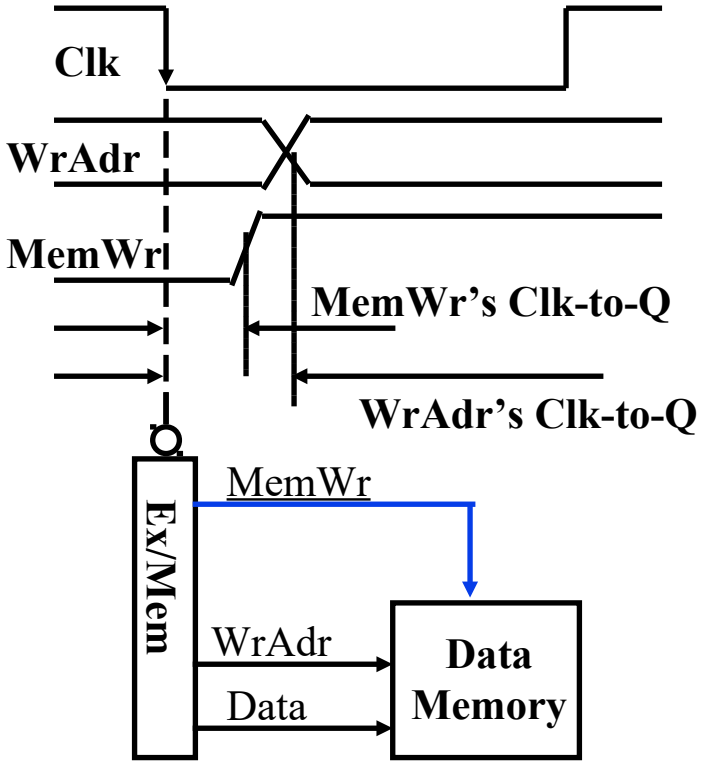


Outline of Today's Lecture

- Recap and Introduction (5 minutes)
- Introduction to the Concept of Pipelined Processor (15 minutes)
- Pipelined Datapath and Pipelined Control (25 minutes)
- How to Avoid Race Condition in a Pipeline Design?
- Pipeline Example: Instructions Interaction (15 minutes)
- Summary (5 minutes)

Beginning of the Wr's Stage: A Real World Problem



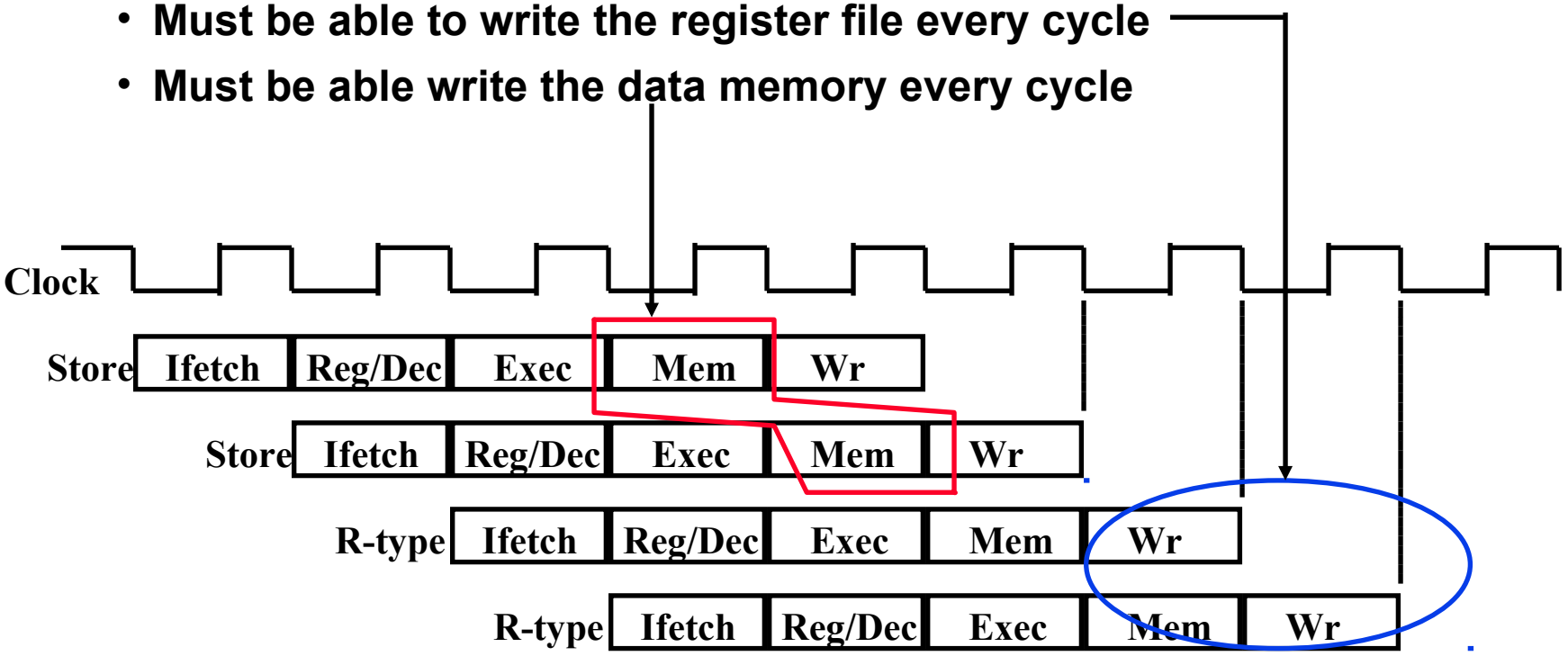


- At the beginning of the Wr stage, we have a problem if:
  - RegAdr's (Rd or Rt) Clk-to-Q > RegWr's Clk-to-Q
- Similarly, at the beginning of the Mem stage, we have a problem if:
  - WrAdr's Clk-to-Q > MemWr's Clk-to-Q
- We have a race condition between Address and Write Enable!

The Pipeline Problem

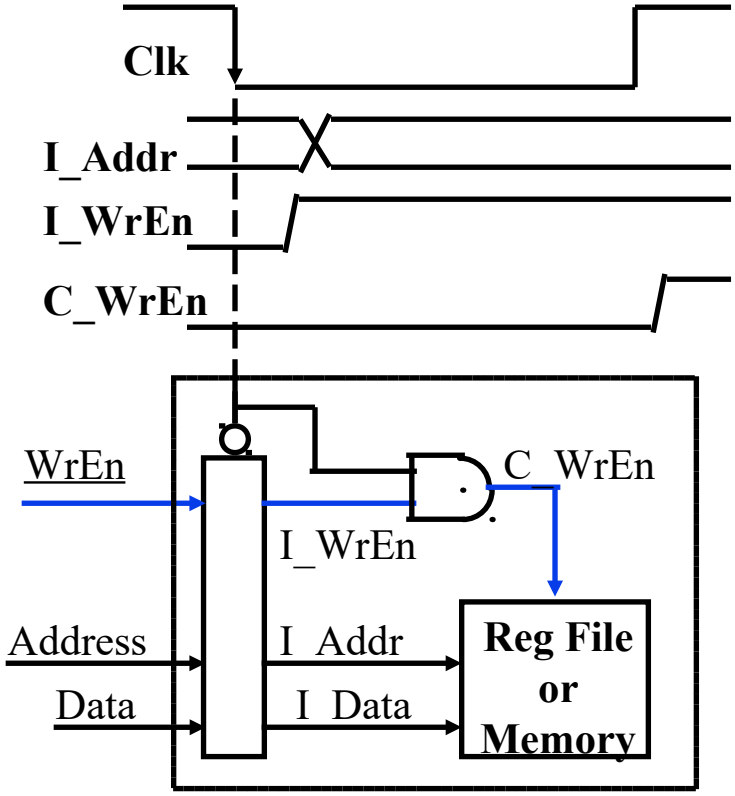
- Multiple Cycle design prevents race condition between Addr and WrEn:
  - Make sure Address is stable by the end of Cycle N
  - Asserts WrEn during Cycle N + 1

- This approach can NOT be used in the pipeline design because:
  - Must be able to write the register file every cycle
  - Must be able write the data memory every cycle



Synchronize Register File & Synchronize Memory

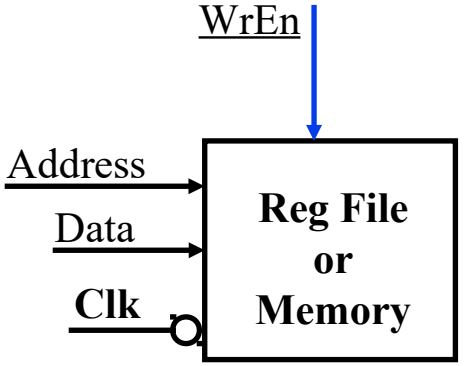
- Solution: And the Write Enable signal with the Clock
  - This is the ONLY place where gating the clock is used
  - MUST consult circuit expert to ensure no timing violation:
    - Example: Clock High Time > Write Access Delay



Synchronize Memory and Register File

Address, Data, and WrEn must be stable at least 1 set-up time before the Clk edge

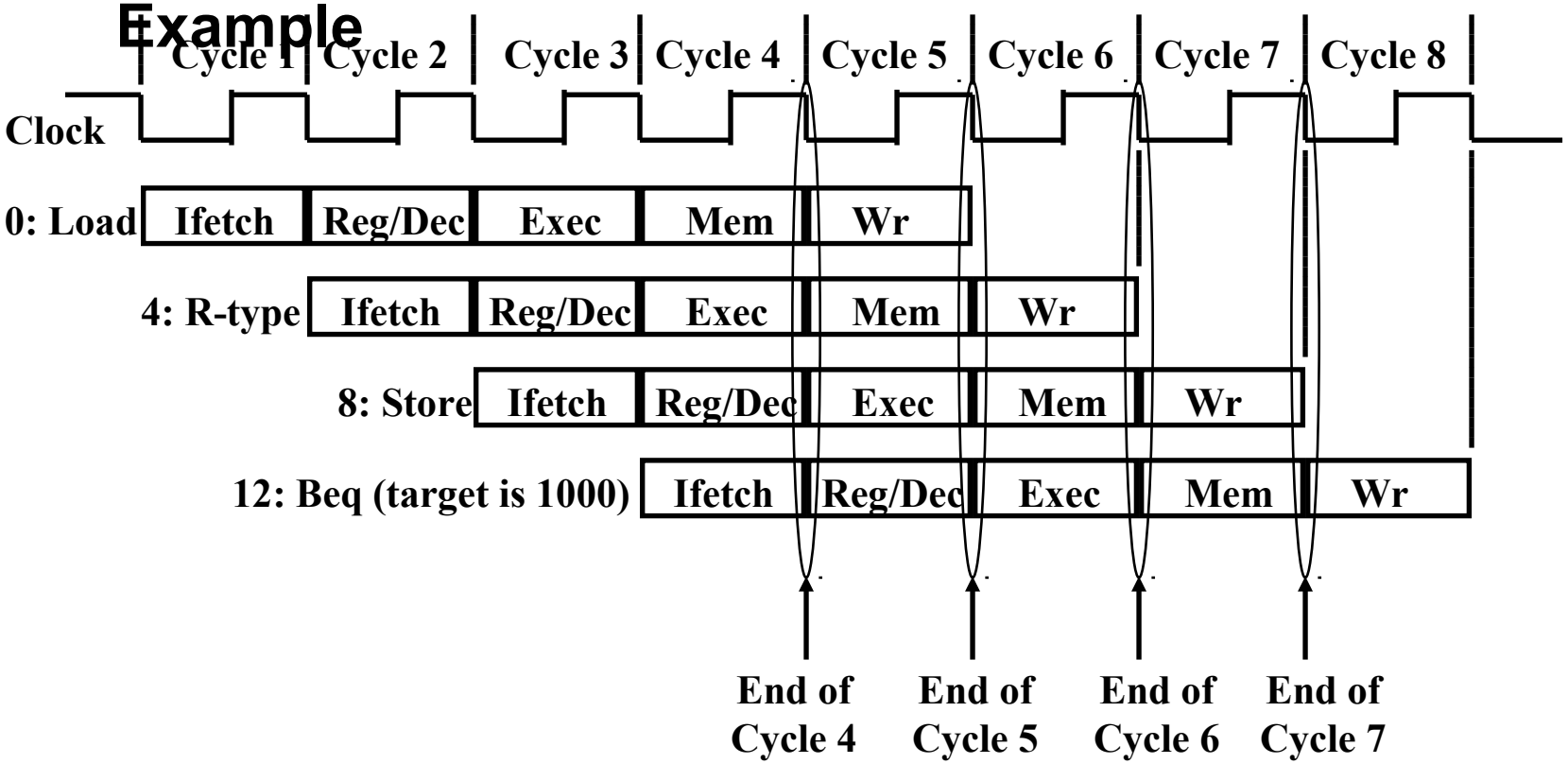
Write occurs at the cycle following the clock edge that captures the signals



Outline of Today's Lecture

- Recap and Introduction (5 minutes)
- Introduction to the Concept of Pipelined Processor (15 minutes)
- Pipelined Datapath and Pipelined Control (25 minutes)
- How to Avoid Race Condition in a Pipeline Design? (5 minutes)
- Pipeline Example: Instructions Interaction
- Summary (5 minutes)

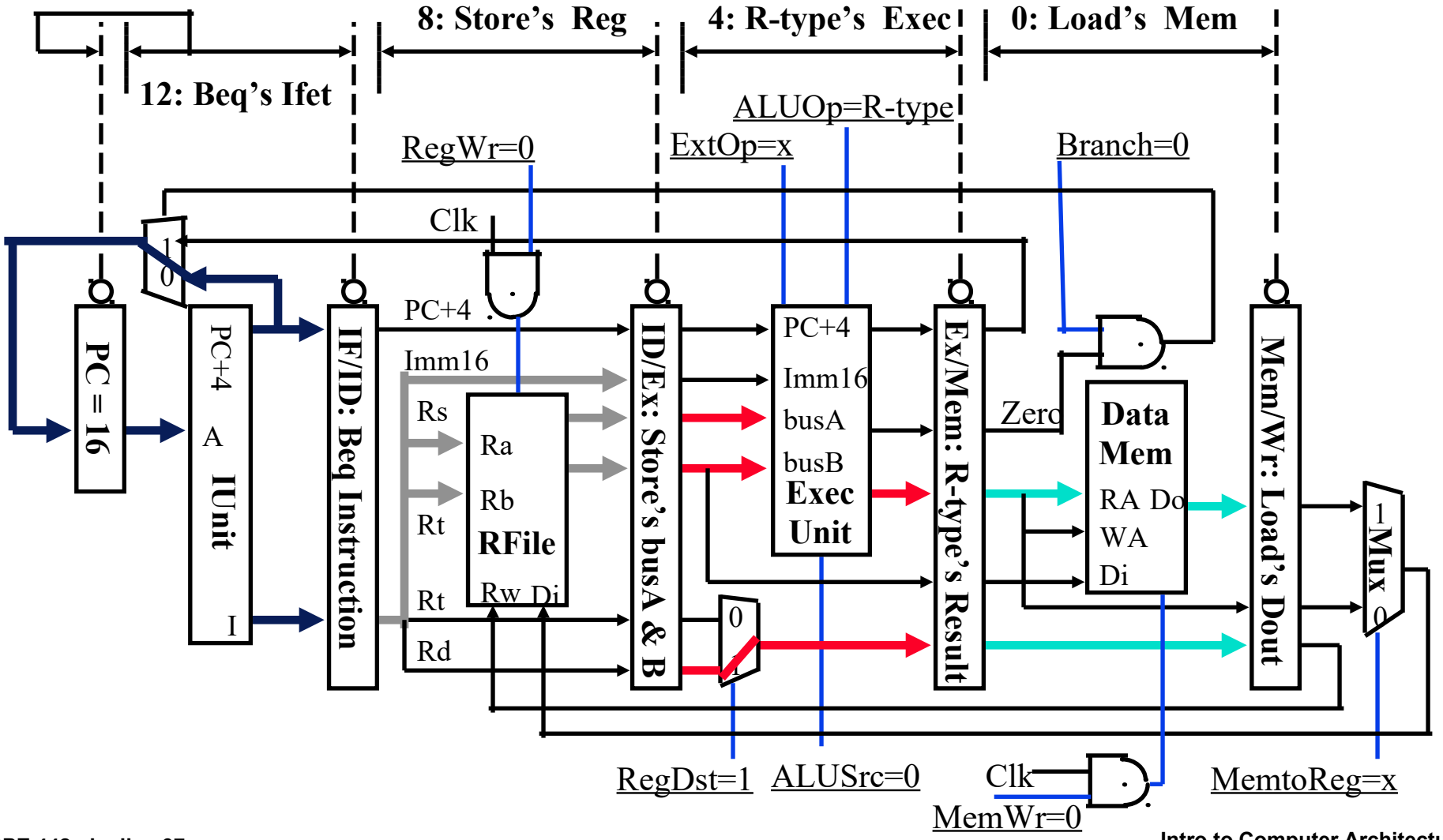
A More Extensive Pipelining Example



- End of Cycle 4: Load's Mem, R-type's Exec, Store's Reg, Beq's Ifetch
- End of Cycle 5: Load's Wr, R-type's Mem, Store's Exec, Beq's Reg
- End of Cycle 6: R-type's Wr, Store's Mem, Beq's Exec
- End of Cycle 7: Store's Wr, Beq's Mem

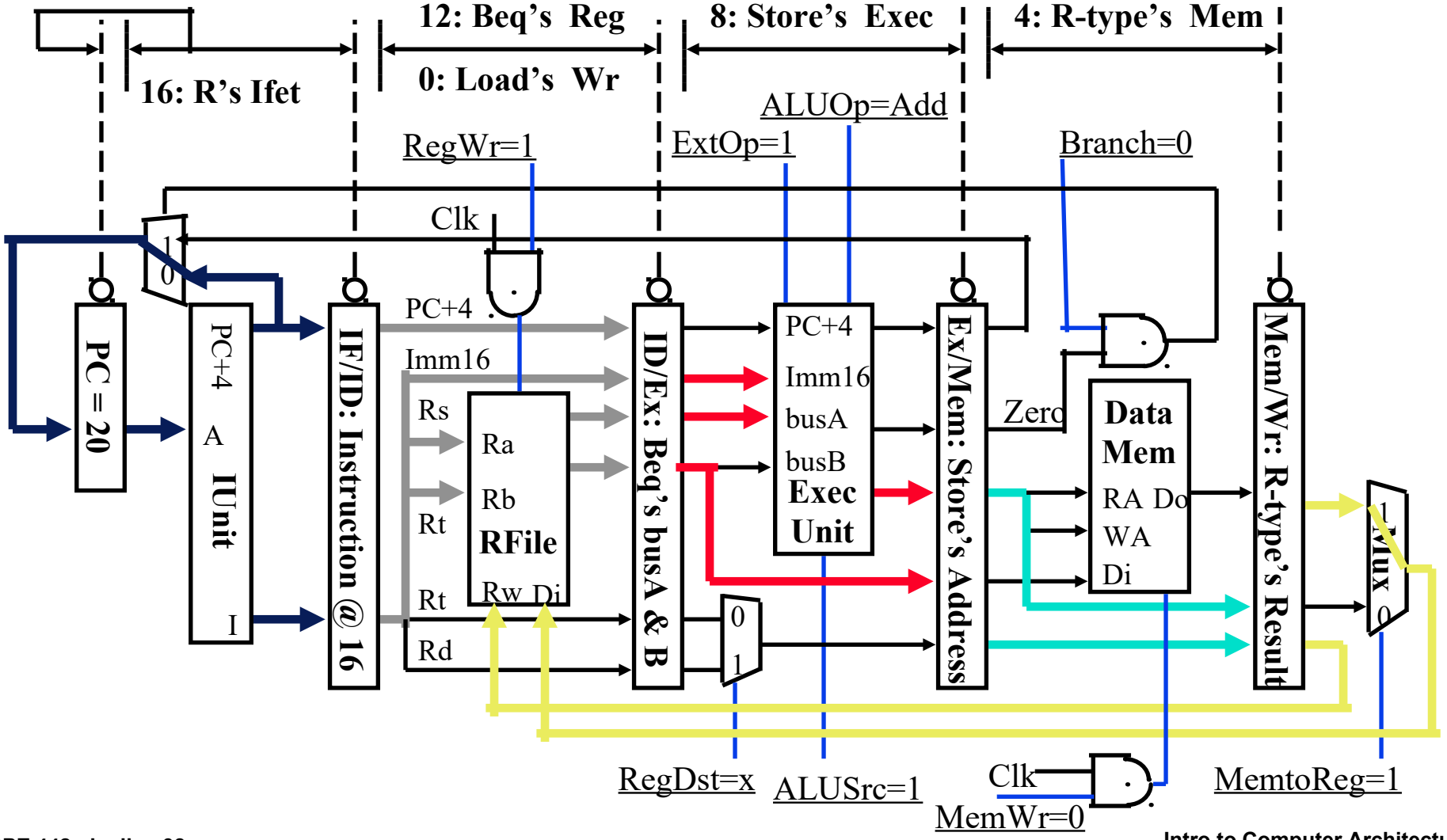
Pipelining Example: End of Cycle 4

- 0: Load's Mem 4: R-type's Exec 8: Store's Reg 12: Beq's Ifetch



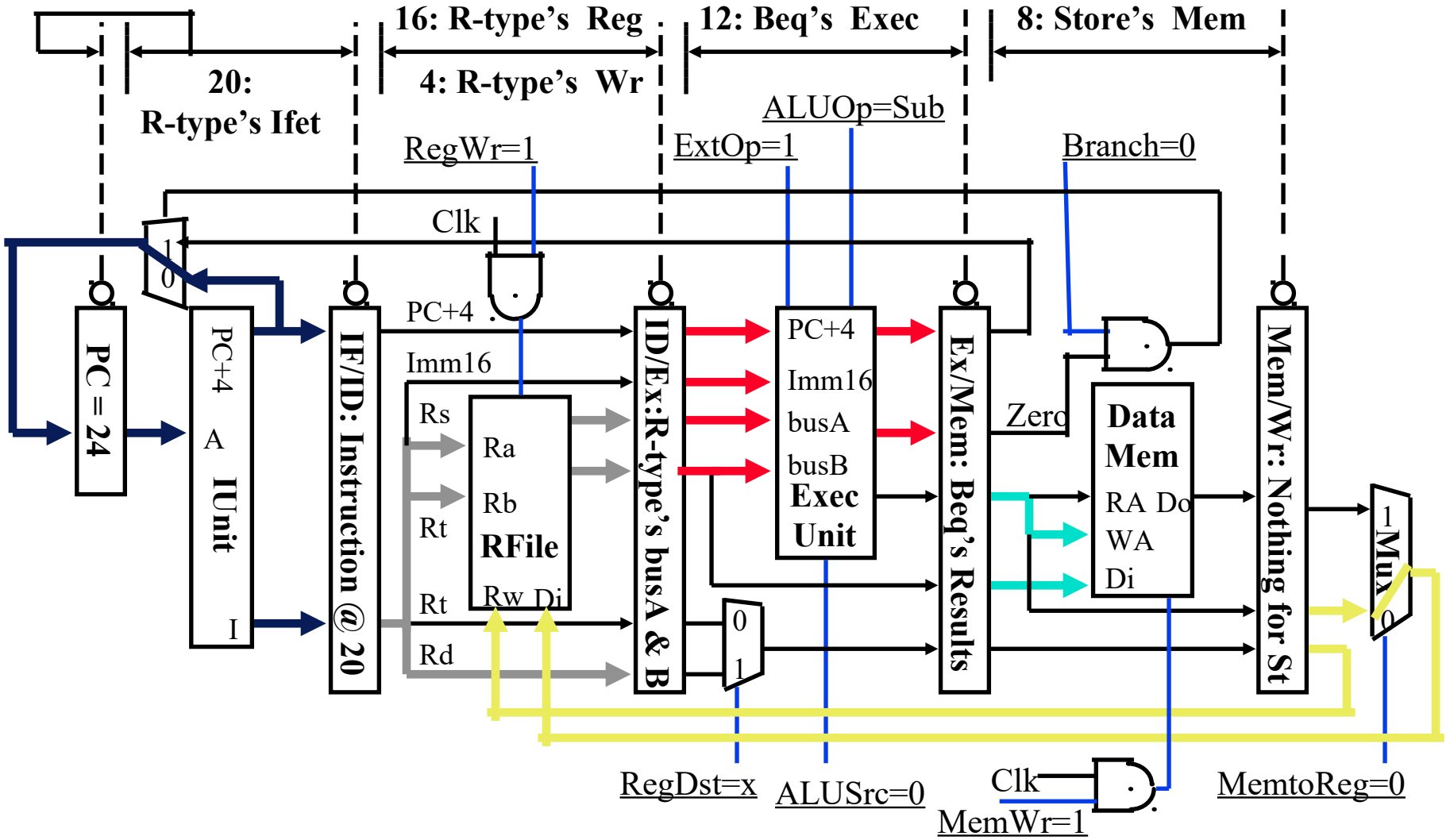
Pipelining Example: End of Cycle 5

- 0: Lw's Wr 4: R's Mem 8: Store's Exec 12: Beq's Reg 16: R's Ifetch



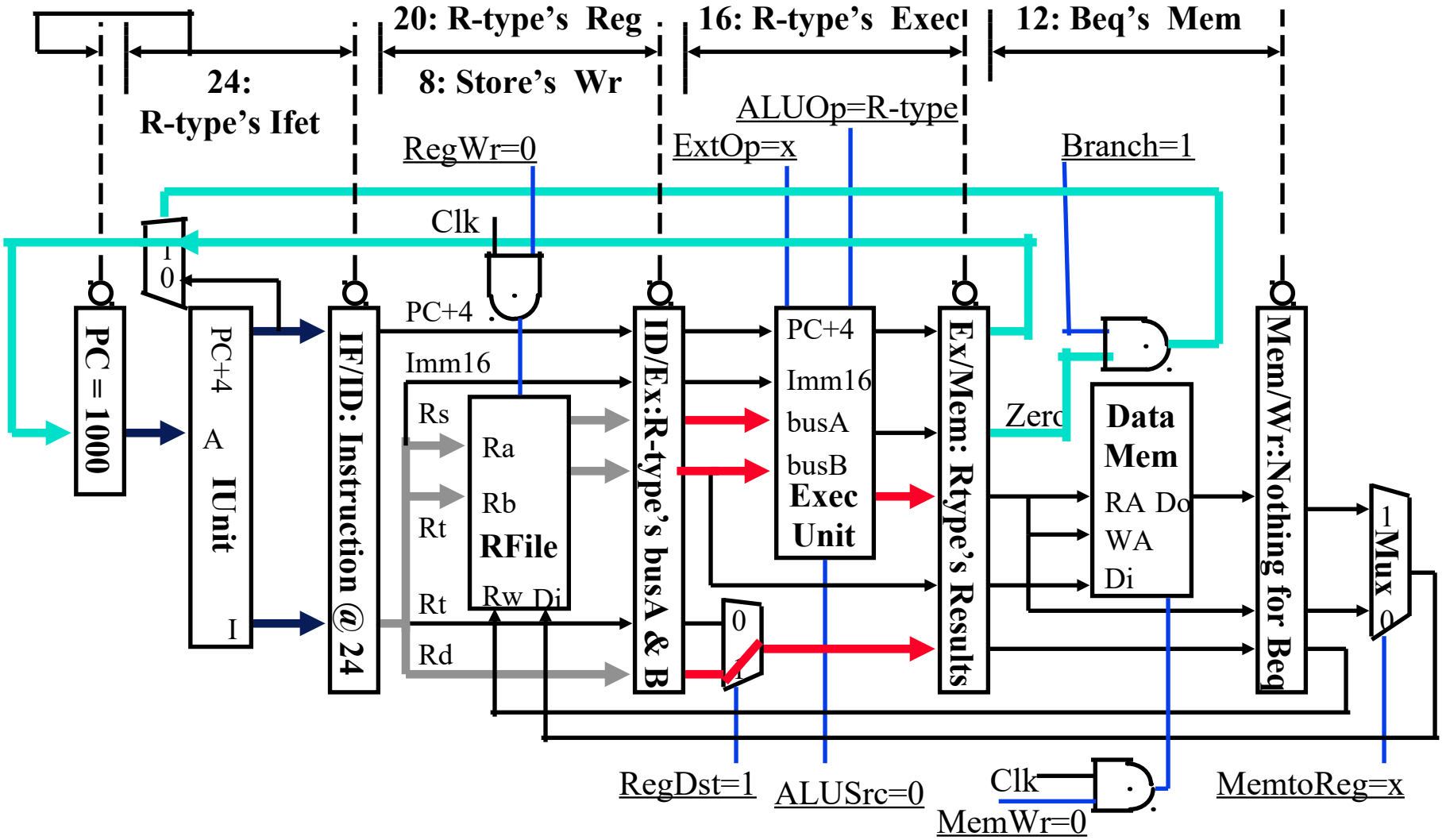
Pipelining Example: End of Cycle 6

- 4: R's Wr 8: Store's Mem 12: Beq's Exec 16: R's Reg 20: R's Ifet

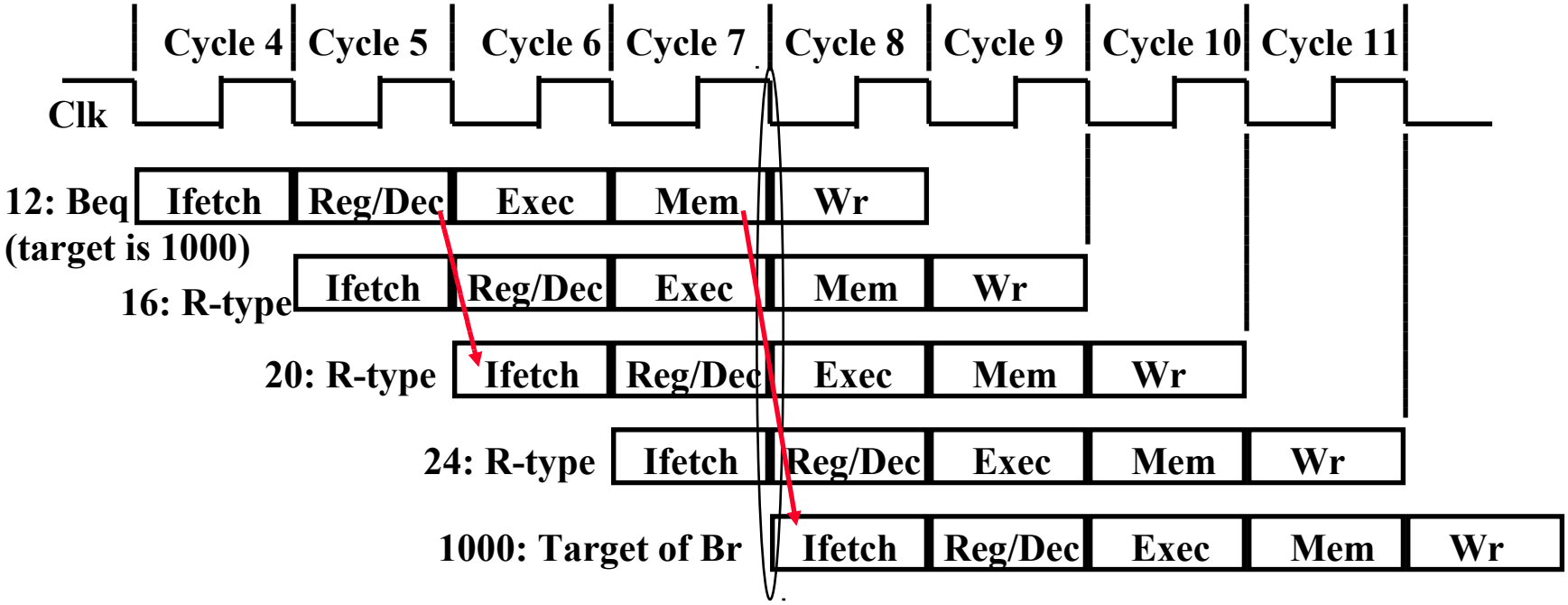


Pipelining Example: End of Cycle 7

- 8: Store's Wr 12: Beq's Mem 16: R's Exec 20: R's Reg 24: R's Ifet

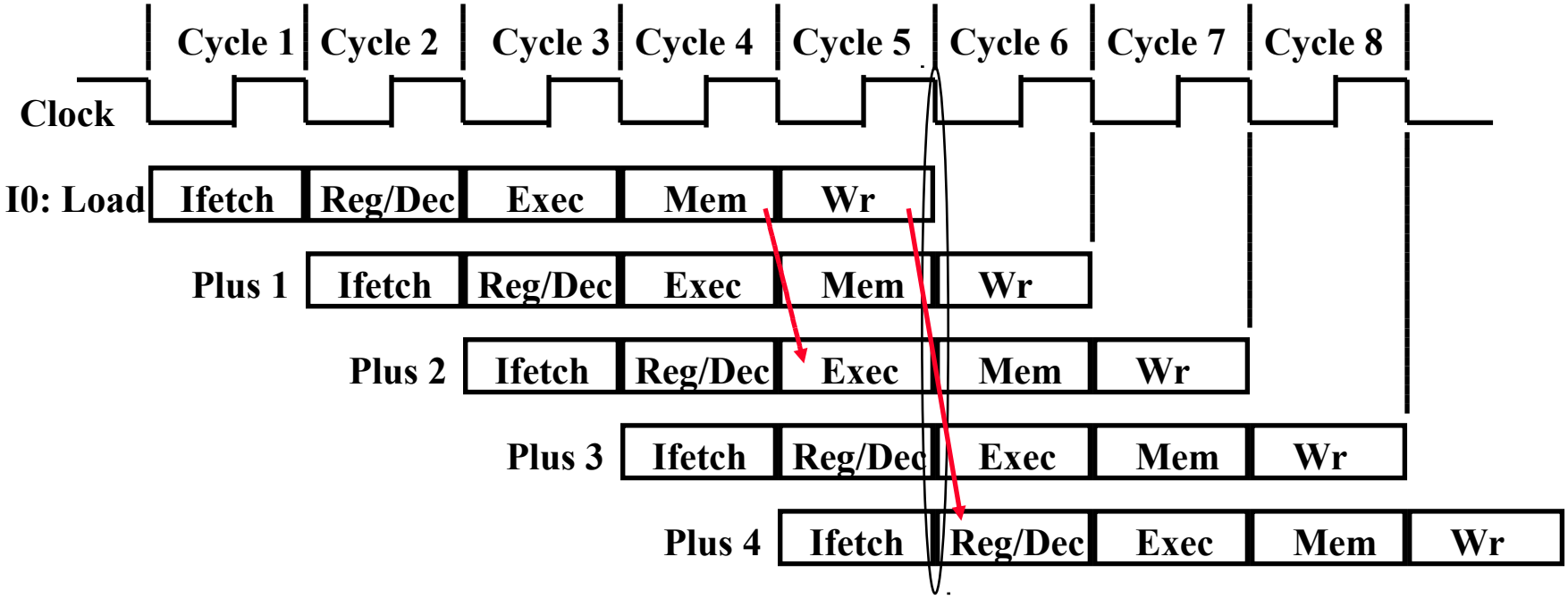


The Delay Branch Phenomenon



- Although Beq is fetched during Cycle 4:
  - Target address is NOT written into the PC until the end of Cycle 7
  - Branch's target is NOT fetched until Cycle 8
  - 3-instruction delay before the branch take effect
- This is referred to as Branch Hazard:
  - Clever design techniques can reduce the delay to ONE instruction

The Delay Load Phenomenon

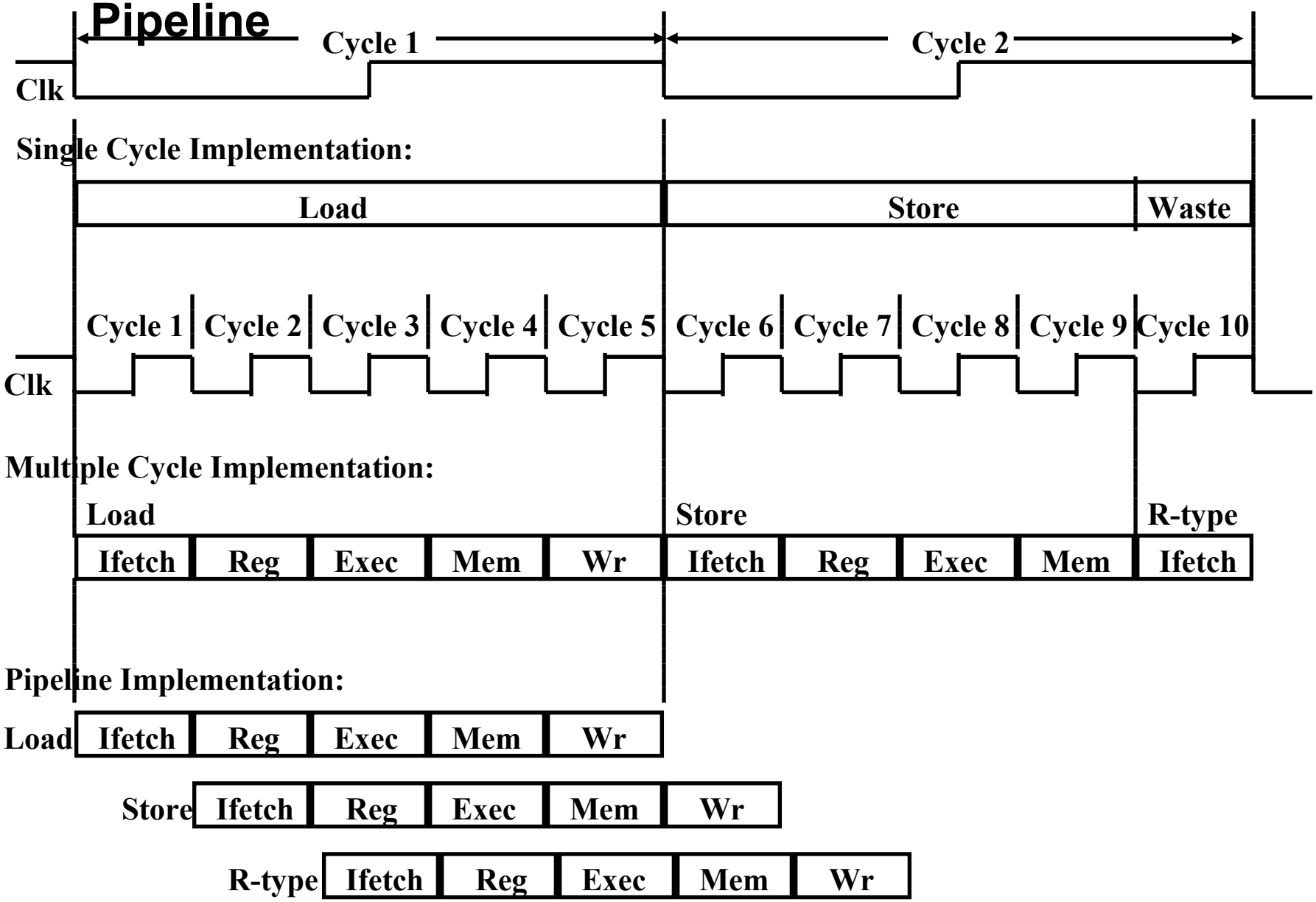


- Although Load is fetched during Cycle 1:
  - The data is NOT written into the Reg File until the end of Cycle 5
  - We cannot read this value from the Reg File until Cycle 6
  - 3-instruction delay before the load take effect
- This is referred to as Data Hazard:
  - Clever design techniques can reduce the delay to ONE instruction

Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load
- Multiple Clock Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
- Pipeline Processor:
  - Natural enhancement of the multiple clock cycle processor
  - Each functional unit can only be used once per instruction
  - If a instruction is going to use a functional unit:
    - it must use it at the same stage as all other instructions
  - Pipeline Control:
    - Each stage's control signal depends ONLY on the instruction that is currently in that stage

Single Cycle, Multiple Cycle, vs. Pipeline



Where to get more information?

- Everything You Need to know about Pipeline Computer:
  - Peter Kogge, “The Architecture of Pipeline Computers,” McGraw Hill Book Company, 1981
- Some Classic References on RISC Pipelines:
  - Manolis Katevenis, “Reduced Instruction Set Computer Architectures for VLSI,” PhD Thesis, UC Berkeley, 1984.
- Other references:
  - David. A Patterson, “Reduced Instruction Set Computers,” Communications of the ACM, January 1985.
  - Shing Kong, “Performance, Resources, and Complexity,” PhD Thesis, UC Berkeley, 1989.