E85 Digital Design & Computer Engineering

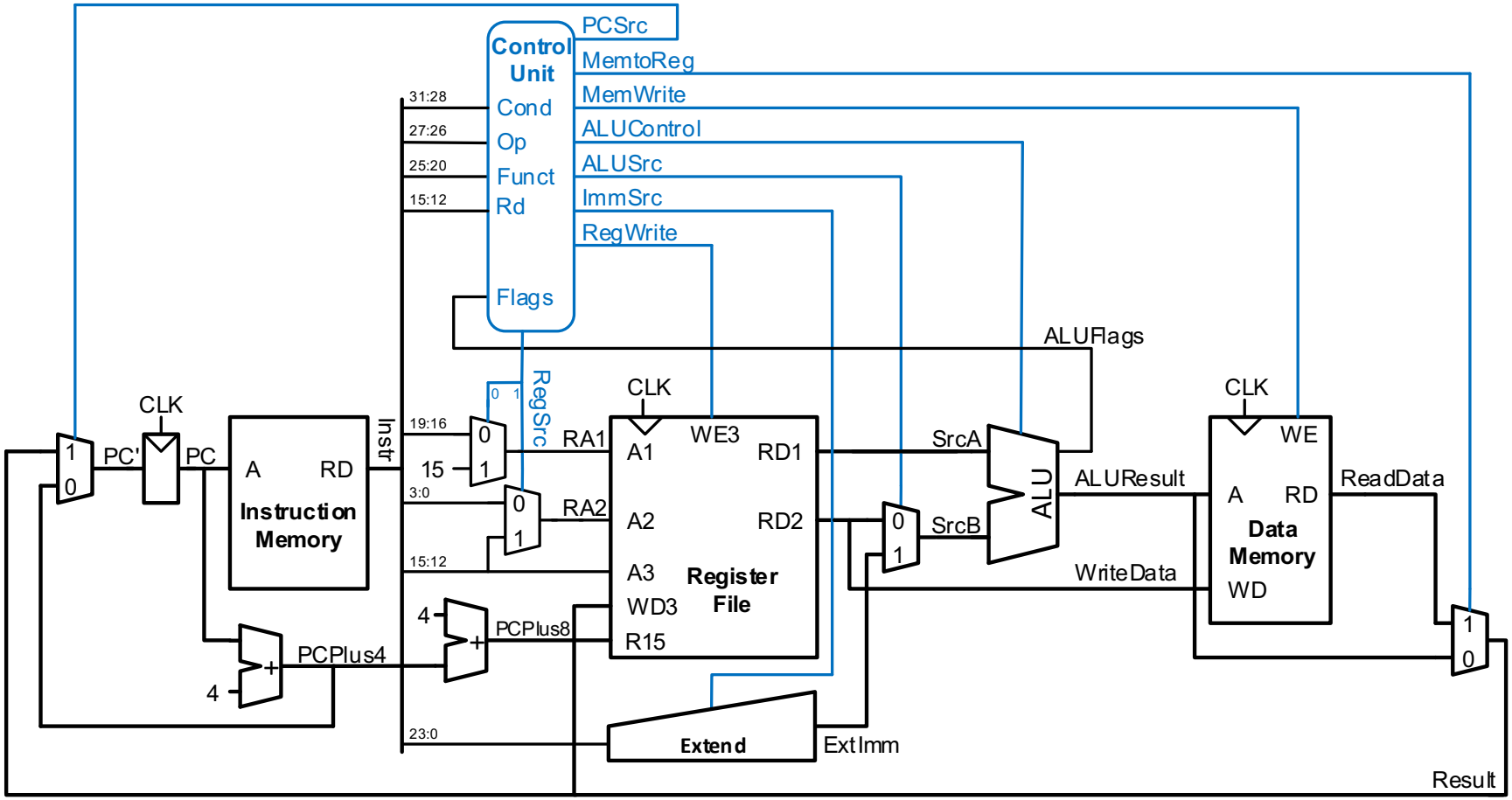
Lecture 22:
Pipelined Processor

HARVEY MUDD COLLEGE

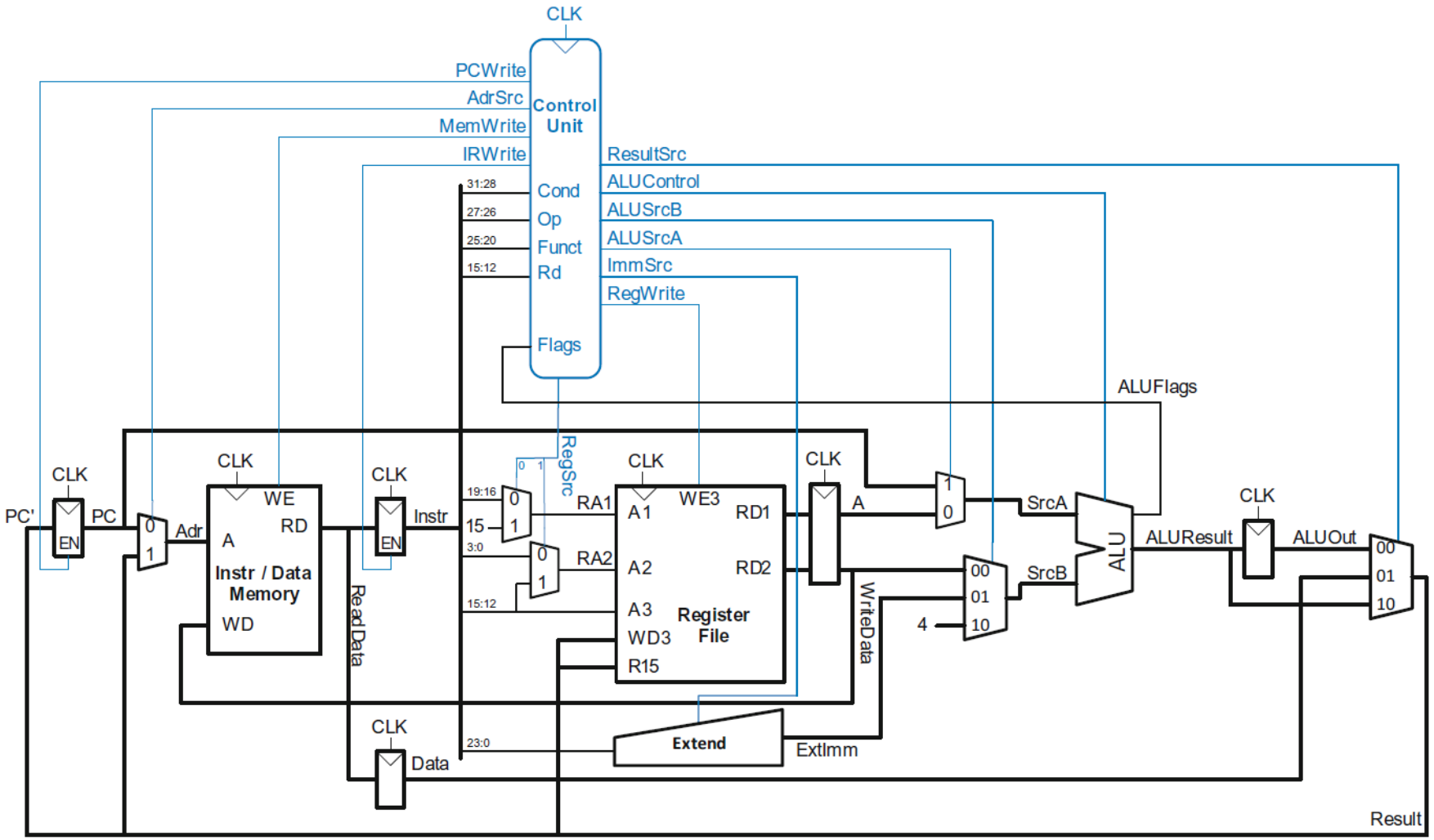
Lecture 22

- **Pipelined Processor**

Review: Single-Cycle ARM Processor

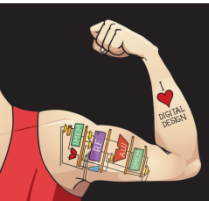


Review: Multicycle ARM Processor



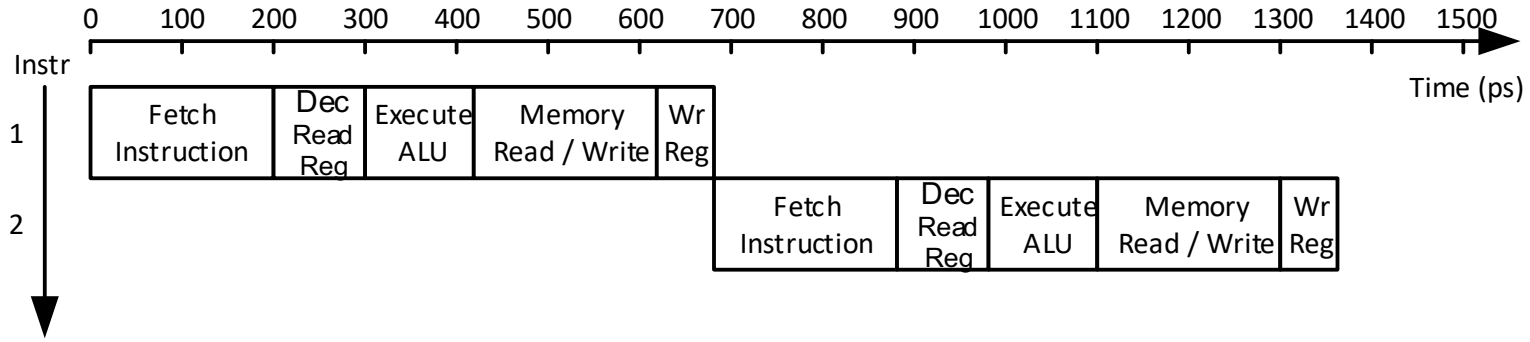
Pipelined ARM Processor

- Aim to really improve performance
- Use temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages

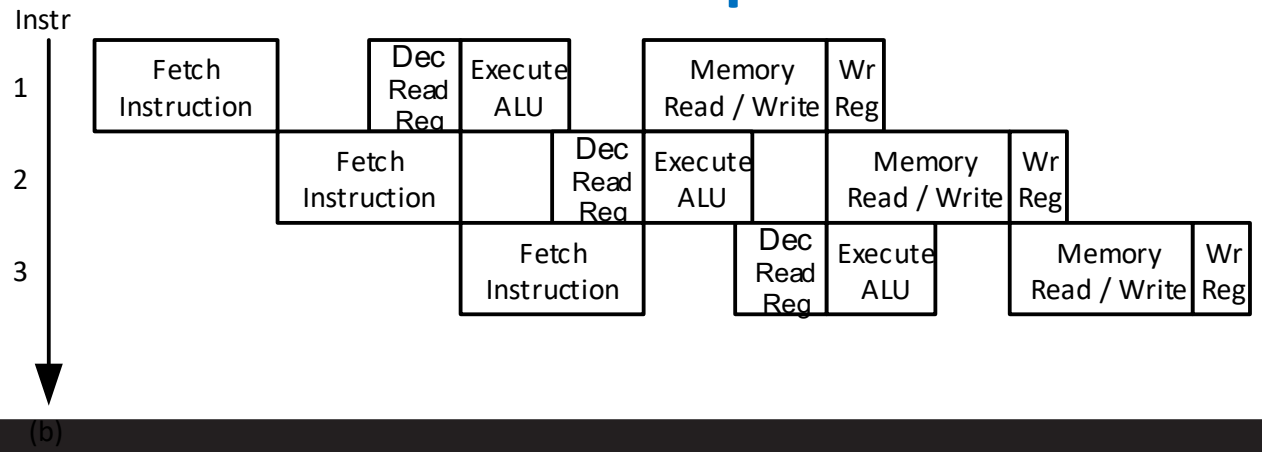


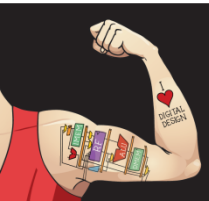
Single-Cycle vs. Pipelined

Single-Cycle

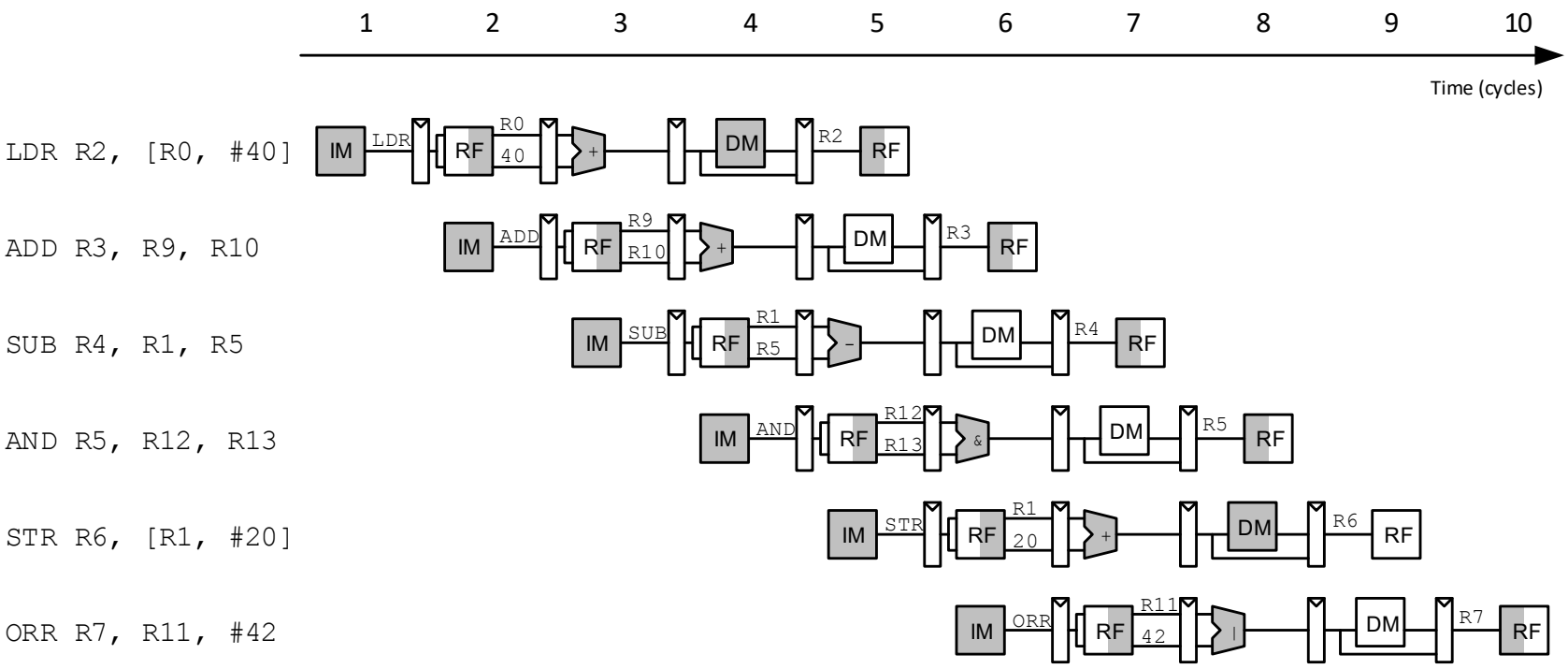


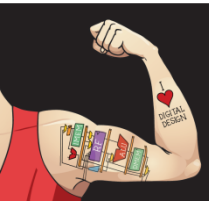
Pipelined



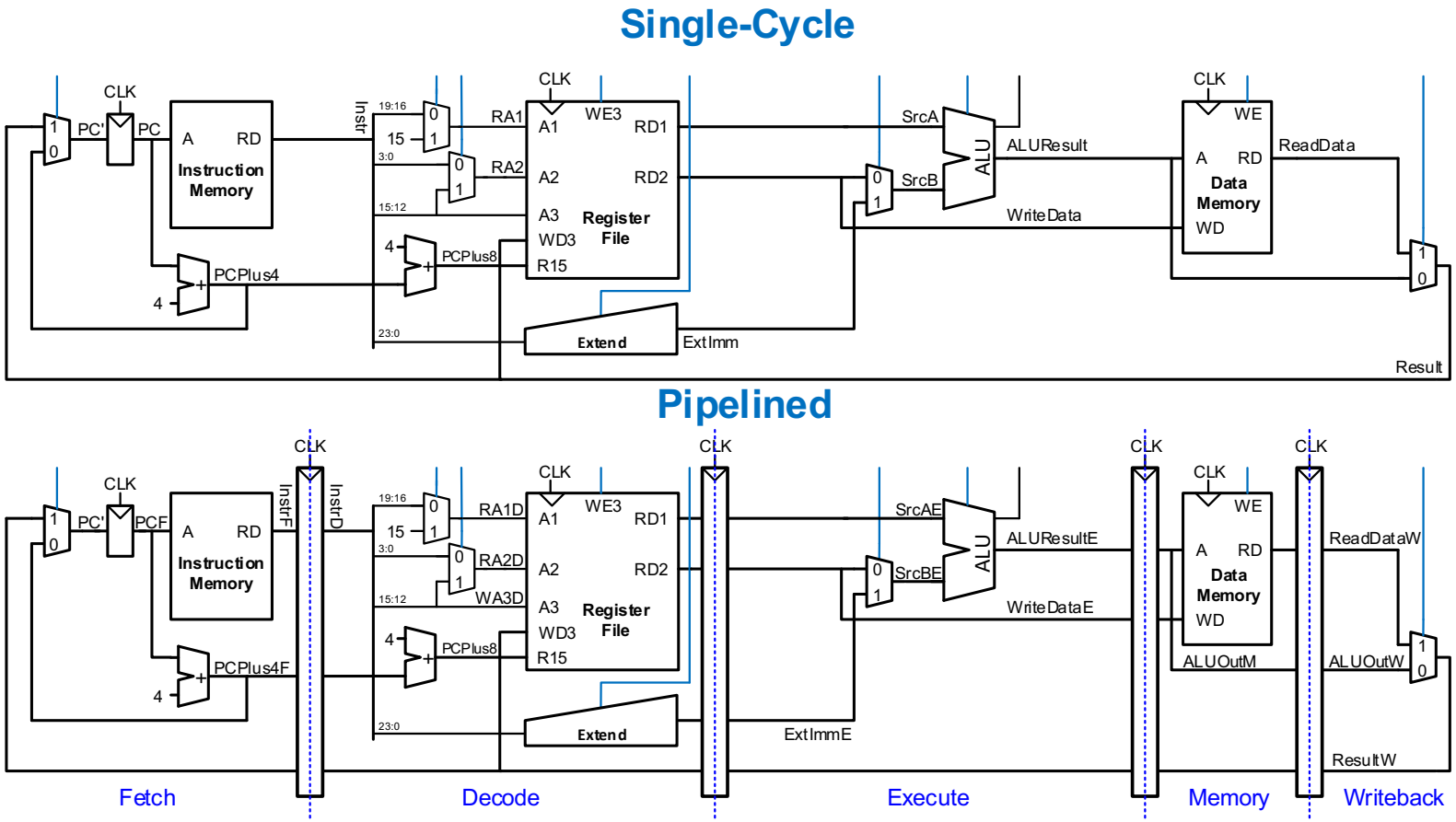


Pipelined Processor Abstraction

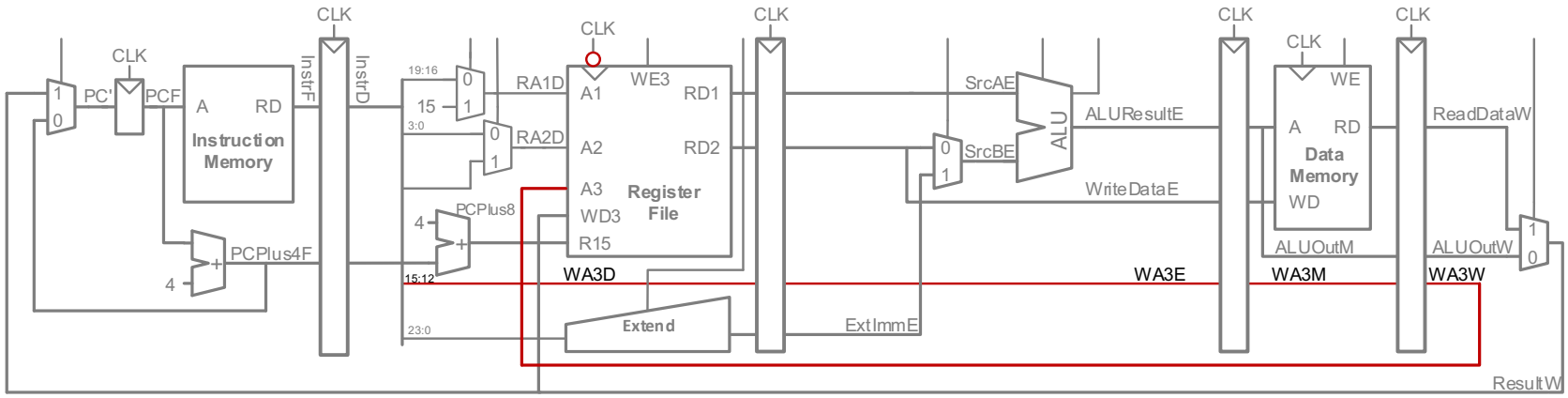




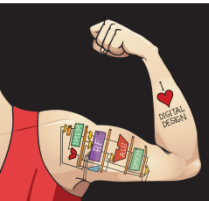
Single-Cycle & Pipelined Datapath



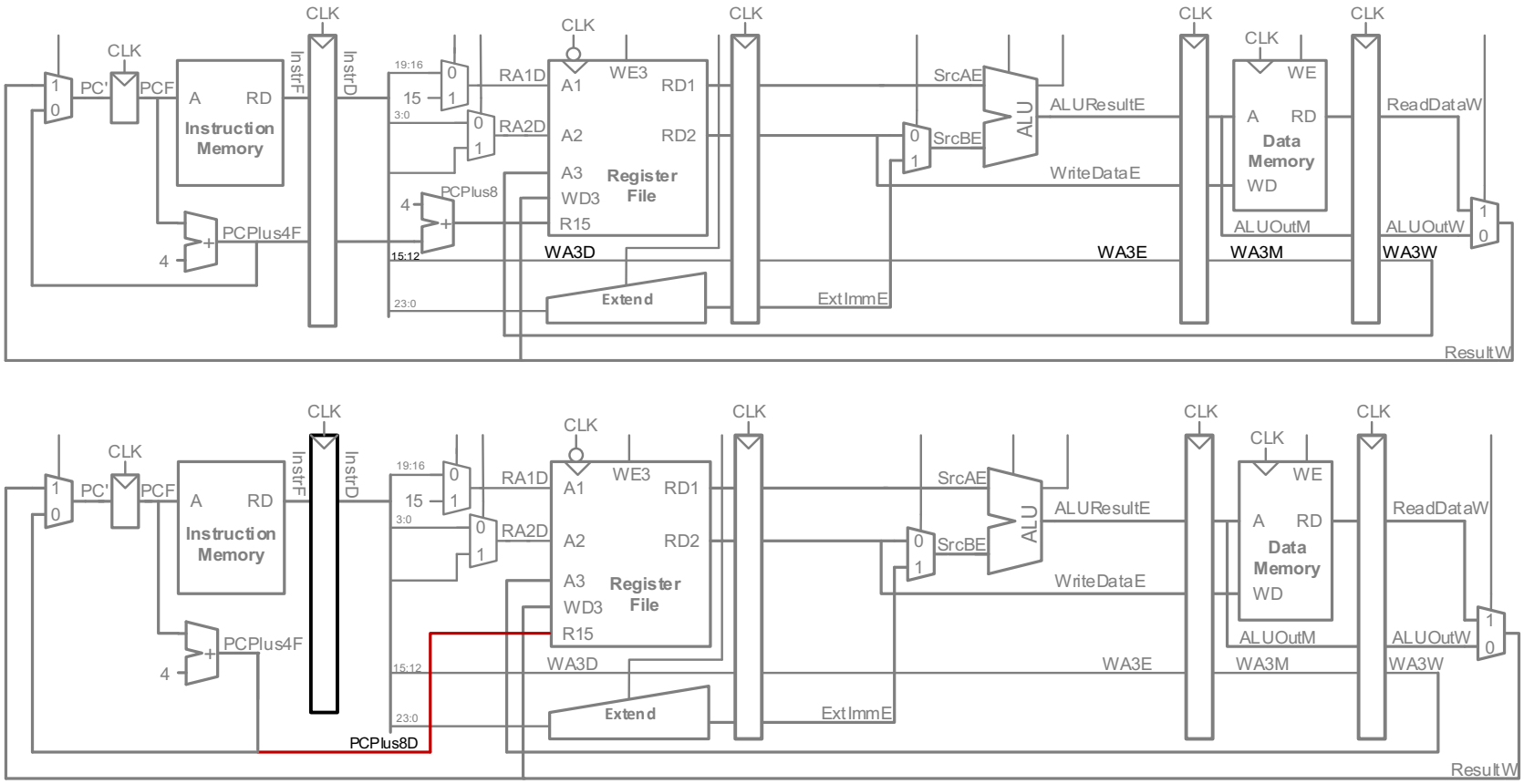
Corrected Pipelined Datapath



- ***WA3* must arrive at same time as *Result***
- **Register file written on falling edge of *CLK***

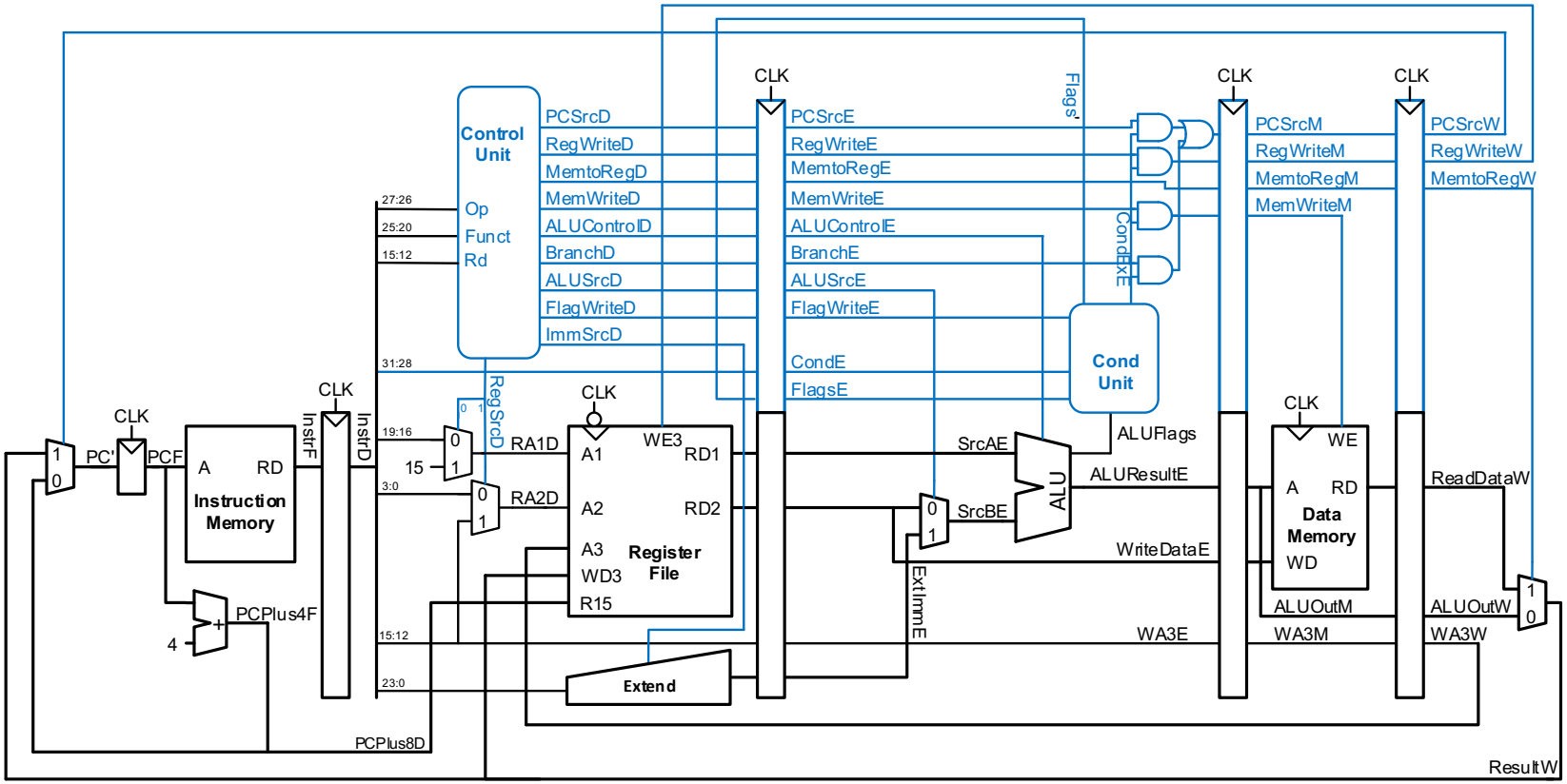


Optimized Pipelined Datapath

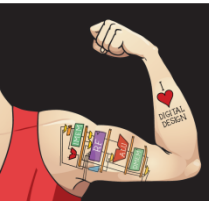


Remove adder by using *PCPlus4F* after *PC* has been updated to *PC*+4

Pipelined Processor Control

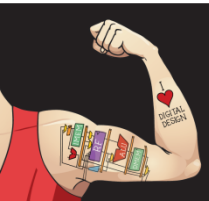


- **Same control unit as single-cycle processor**
- **Control delayed to proper pipeline stage**

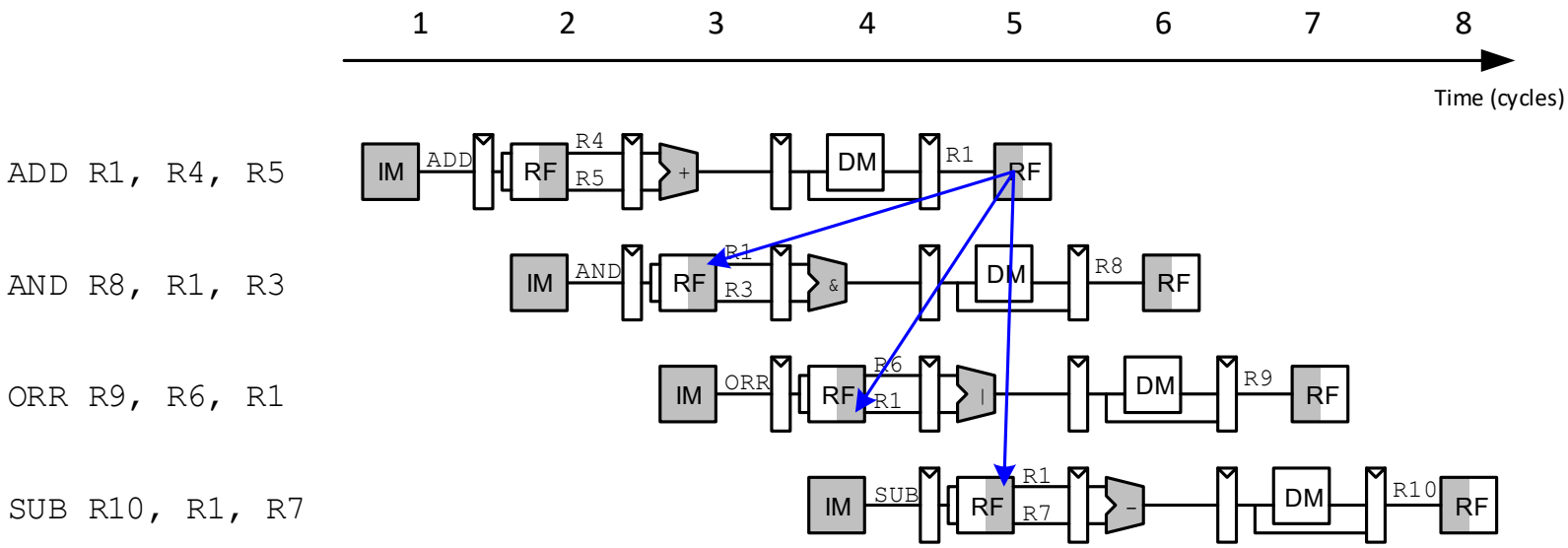


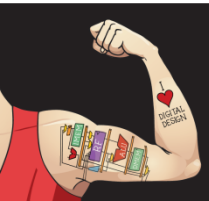
Pipeline Hazards

- When an instruction depends on result from instruction that hasn't completed
- Types:
  - **Data hazard:** register value not yet written back to register file
  - **Control hazard:** next instruction not decided yet (caused by branch)



Data Hazard

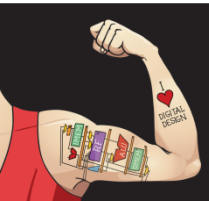




ELSEVIER

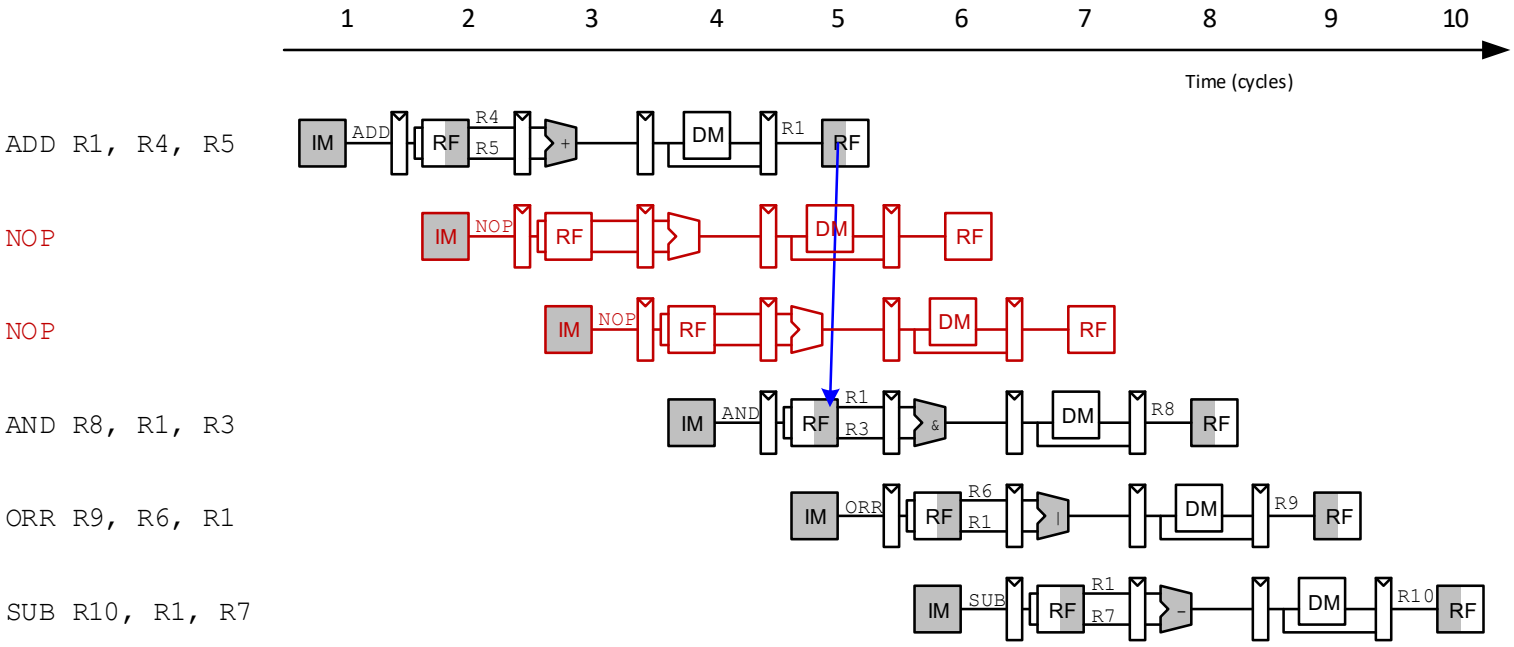
Handling Data Hazards

- Insert `NOP`s in code at compile time
- Rearrange code at compile time
- Forward data at run time
- Stall the processor at run time

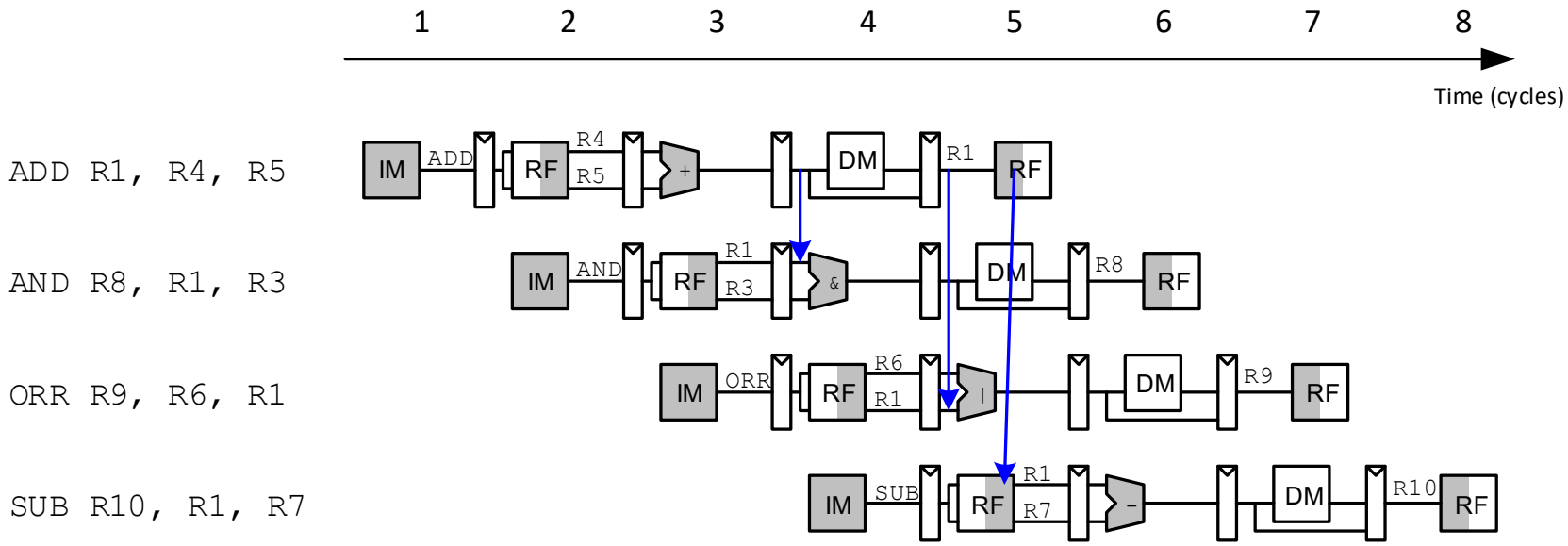


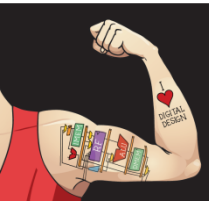
Compile-Time Hazard Elimination

- Insert enough NOPs for result to be ready
- Or move independent useful instructions forward



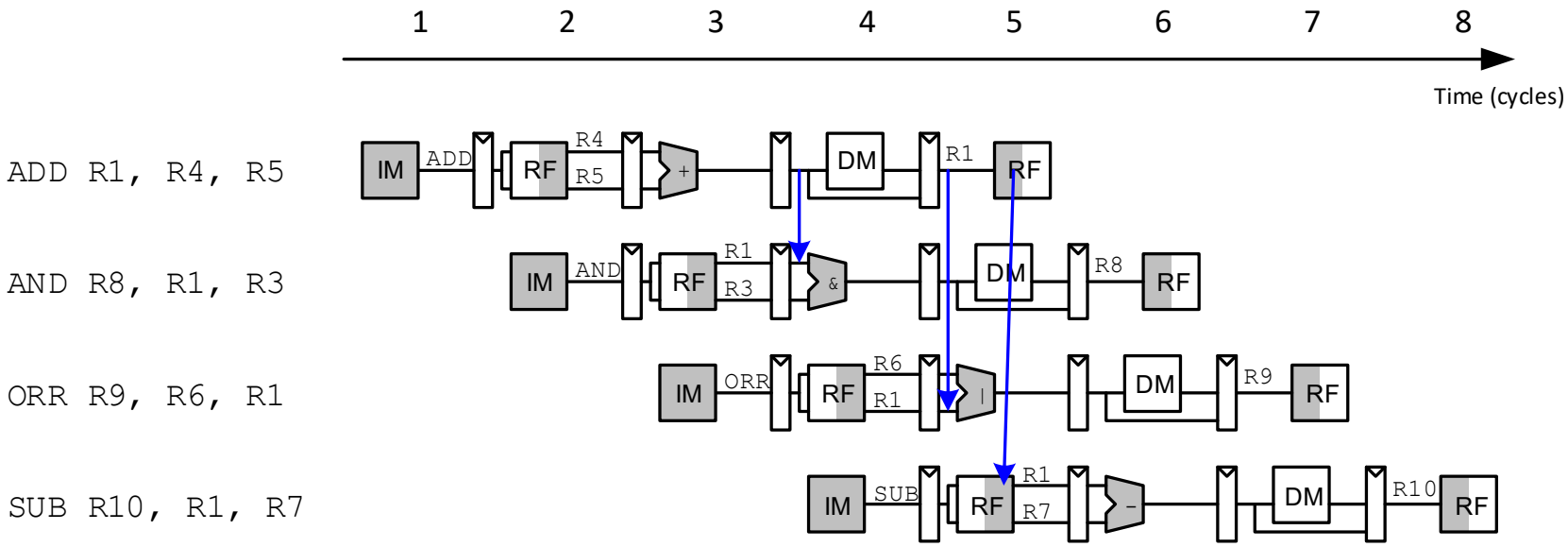
Data Forwarding



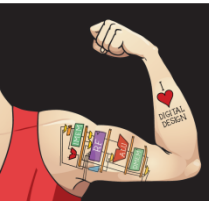


ELSEVIER

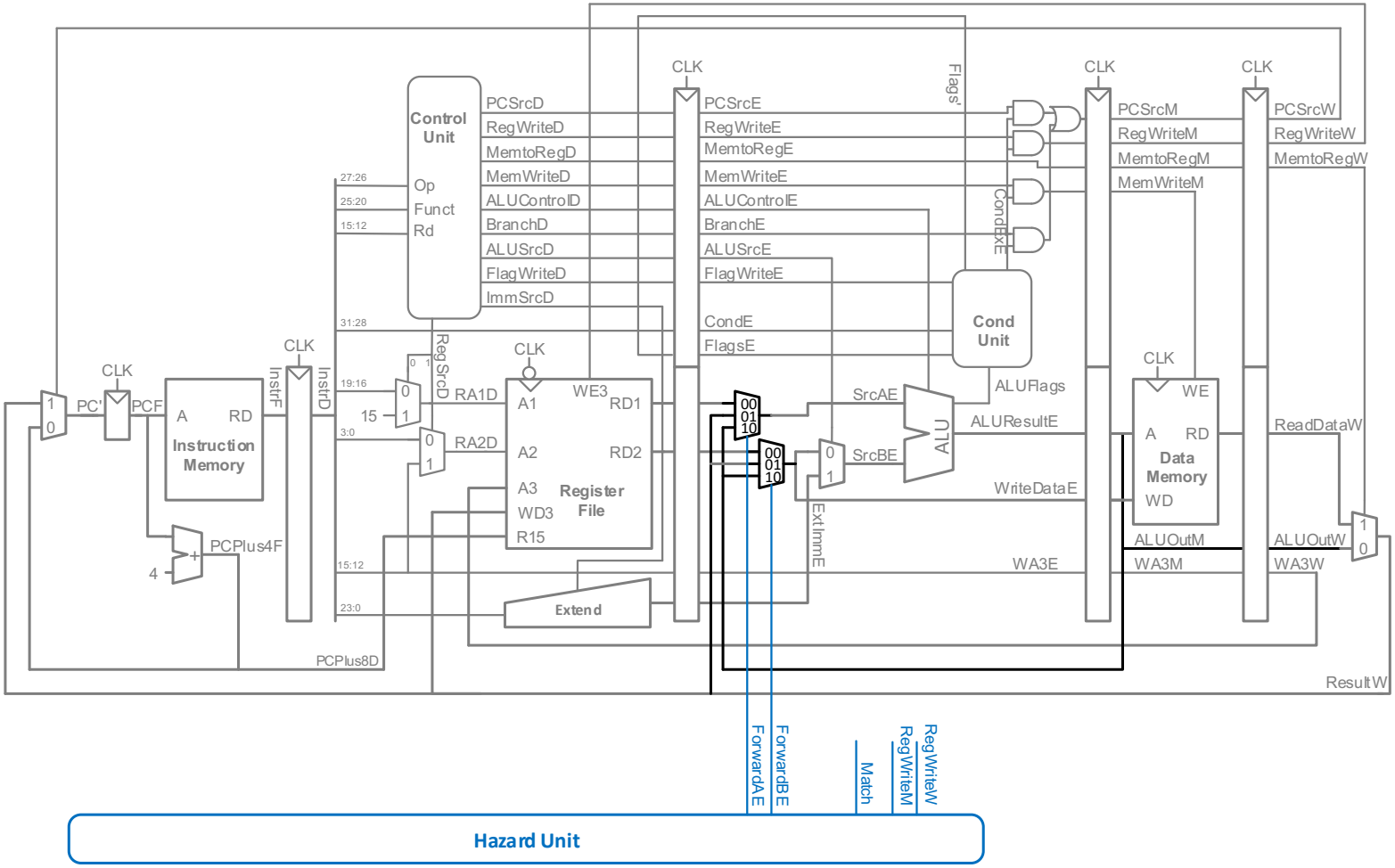
Data Forwarding



- Check if register read in Execute stage matches register written in Memory or Writeback stage
- If so, forward result



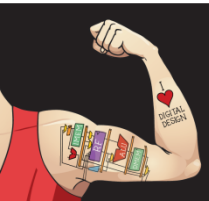
Data Forwarding



Data Forwarding

- **Execute** stage register matches **Memory** stage register?
  Match_1E_M = (RA1E == WA3M)
  Match_2E_M = (RA2E == WA3M)
- **Execute** stage register matches **Writeback** stage register?
  Match_1E_W = (RA1E == WA3W)
  Match_2E_W = (RA2E == WA3W)
- If it matches, forward result:

```
if      (Match_1E_M • RegWriteM)    ForwardAE = 10;
else if (Match_1E_W • RegWriteW)    ForwardAE = 01;
else                                ForwardAE = 00;
```

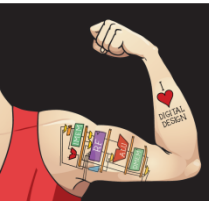


Data Forwarding

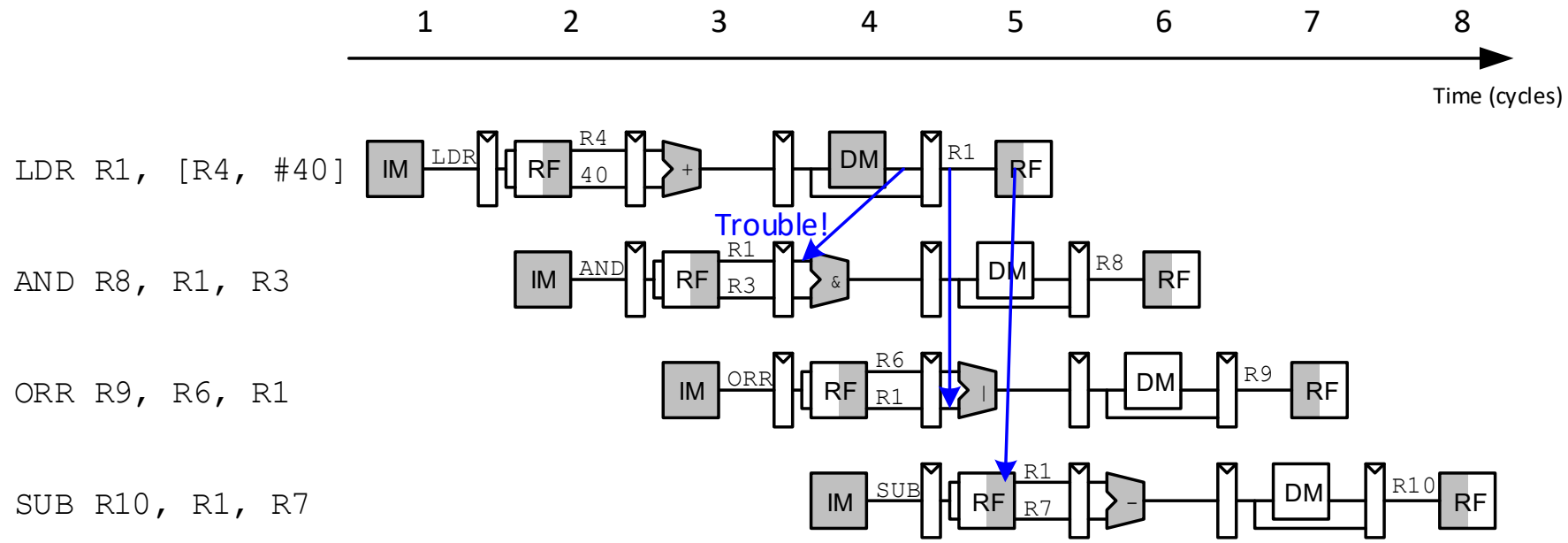
- **Execute** stage register matches **Memory** stage register?
  Match_1E_M = (RA1E == WA3M)
  Match_2E_M = (RA2E == WA3M)
- **Execute** stage register matches **Writeback** stage register?
  Match_1E_W = (RA1E == WA3W)
  Match_2E_W = (RA2E == WA3W)
- If it matches, forward result:

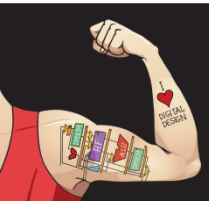
```
if      (Match_1E_M • RegWriteM)    ForwardAE = 10;
else if (Match_1E_W • RegWriteW)    ForwardAE = 01;
else                                ForwardAE = 00;
```

ForwardBE same but with Match2E

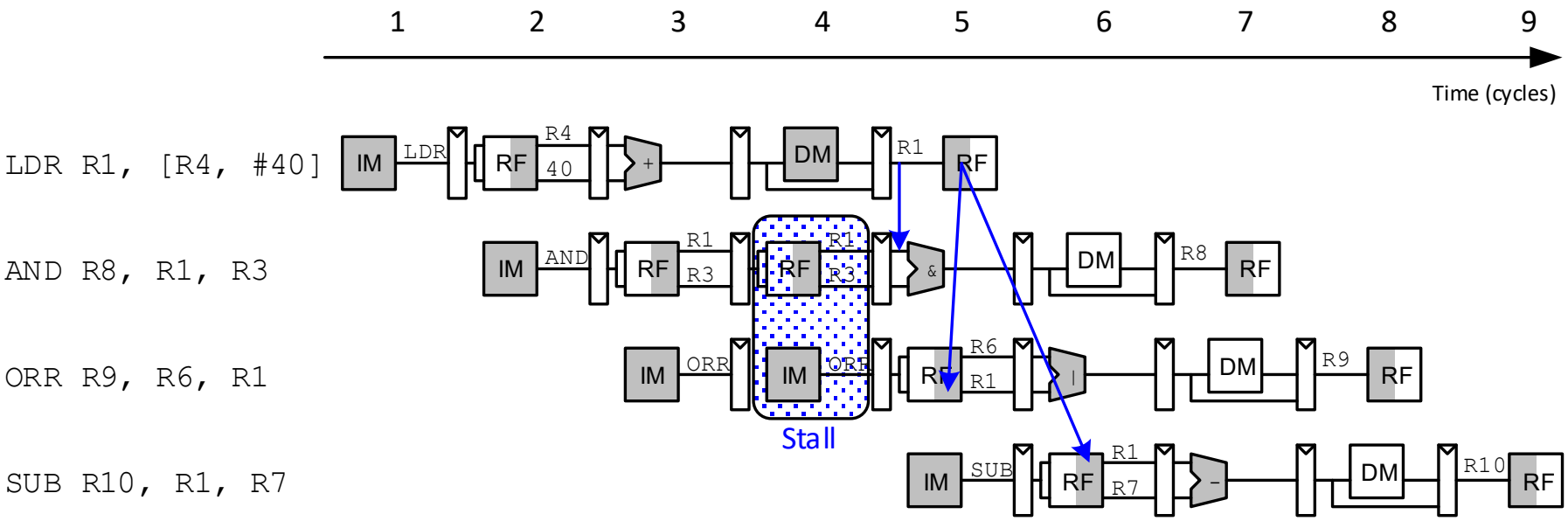


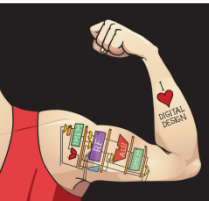
Stalling



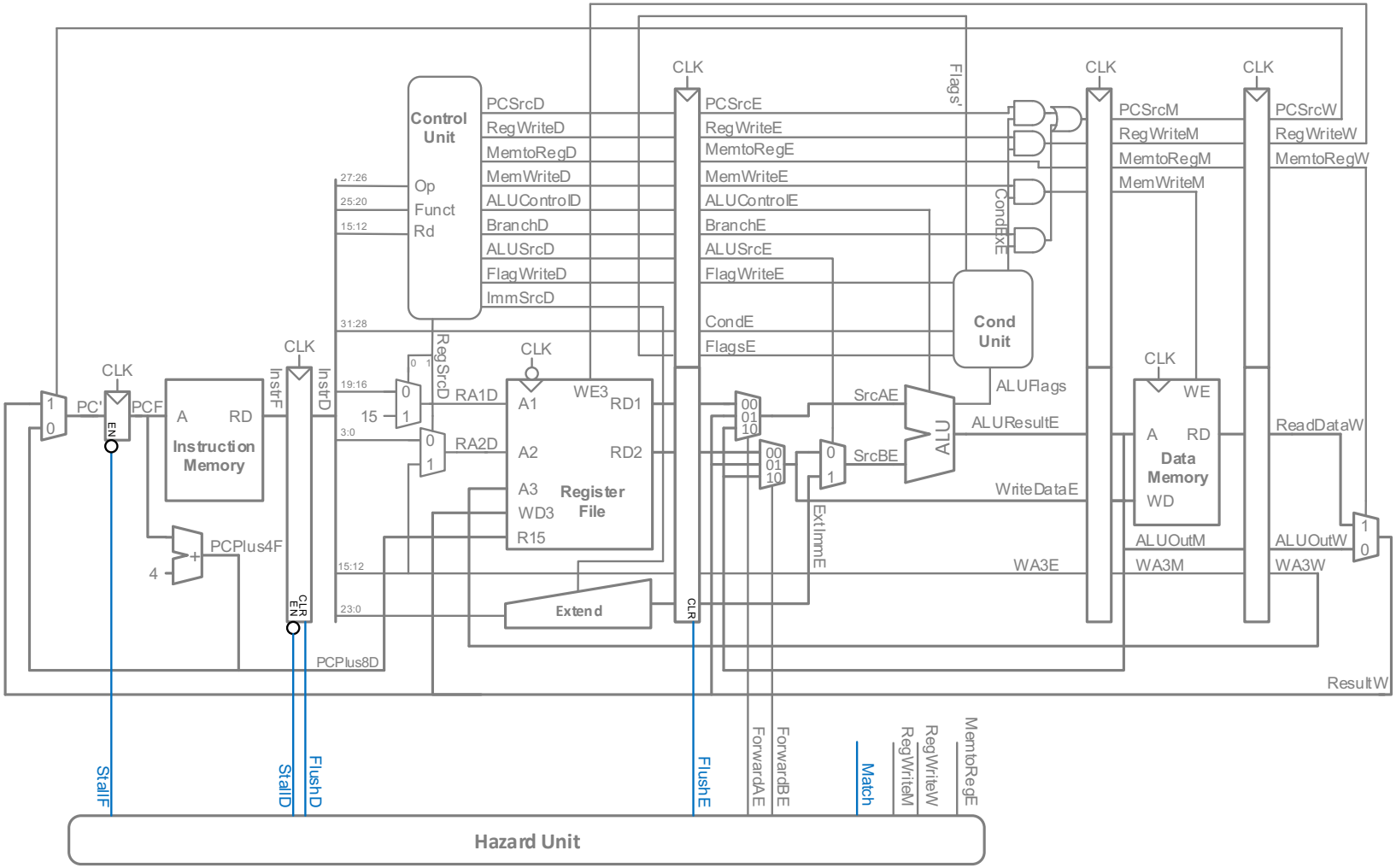


Stalling





Stalling Hardware



Stalling Logic

- Is either source register in the Decode stage the same as the one being written in the Execute stage?

Match_12D_E = (*RA1D* == *WA3E*) + (*RA2D* == *WA3E*)

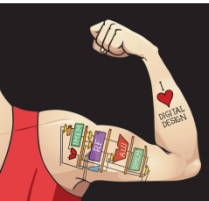
- Is a `LDR` in the Execute stage AND *Match_12D_E*?

ldrstall = *Match_12D_E* • *MemtoRegE*

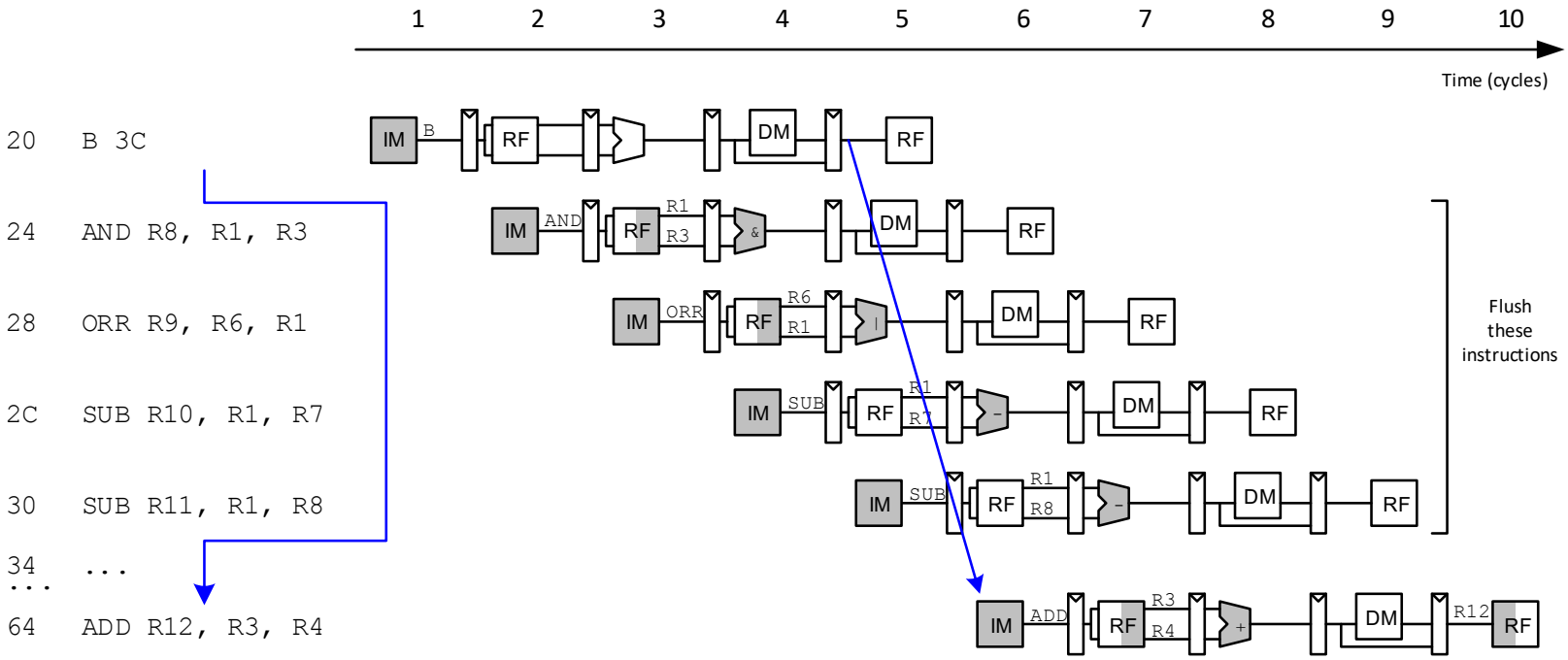
StallF = *StallD* = *FlushE* = *ldrstall*

Control Hazards

- **B:**
  - branch not determined until the Writeback stage of pipeline
  - Instructions after branch fetched before branch occurs
  - These 4 instructions must be flushed if branch happens
- **Writes to PC (R15) similar**

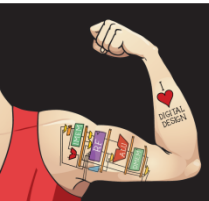


Control Hazards



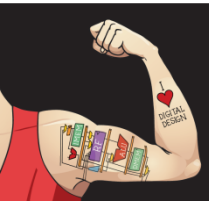
Branch misprediction penalty

- number of instruction flushed when branch is taken (4)
- May be reduced by determining BTA earlier

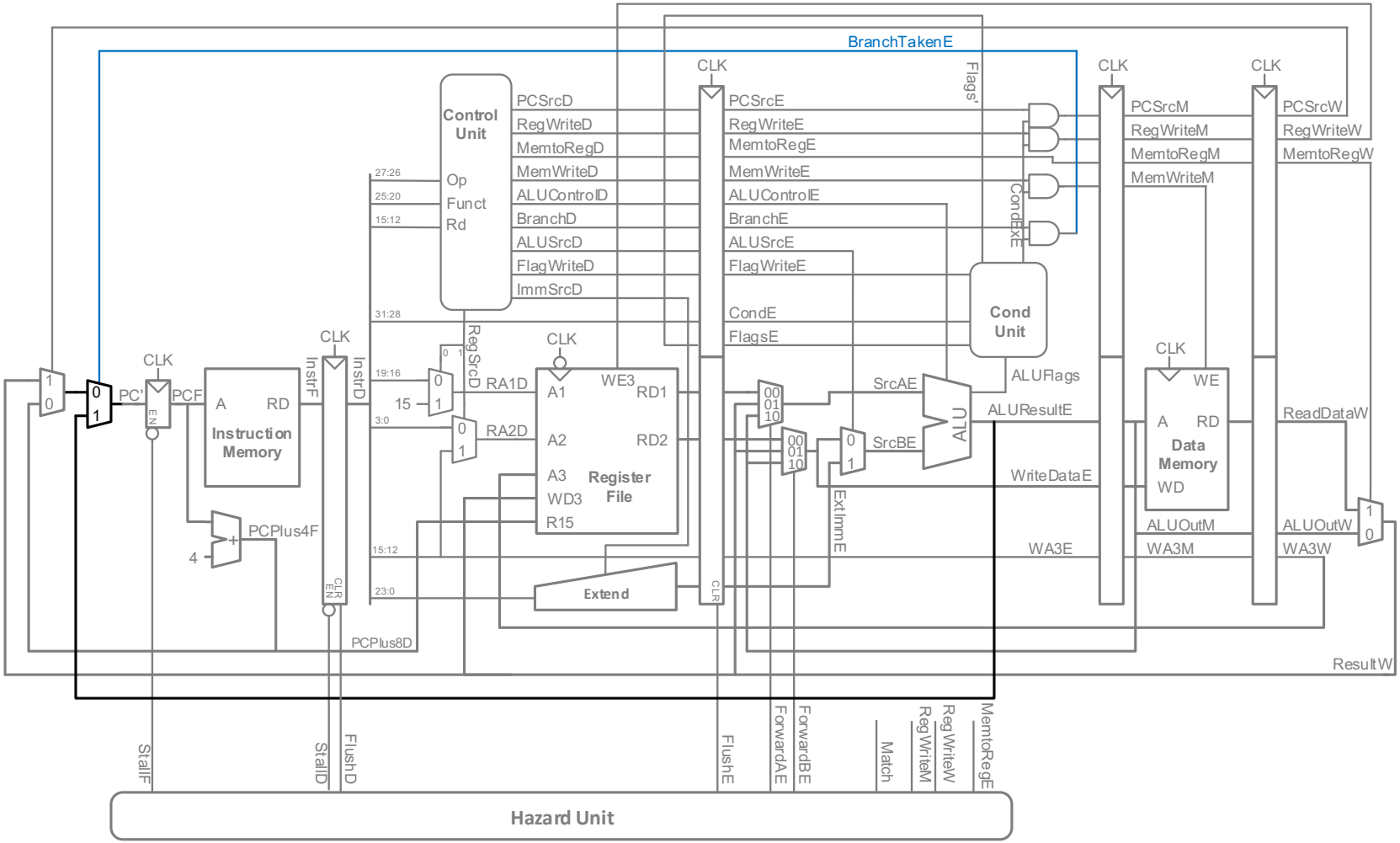


Early Branch Resolution

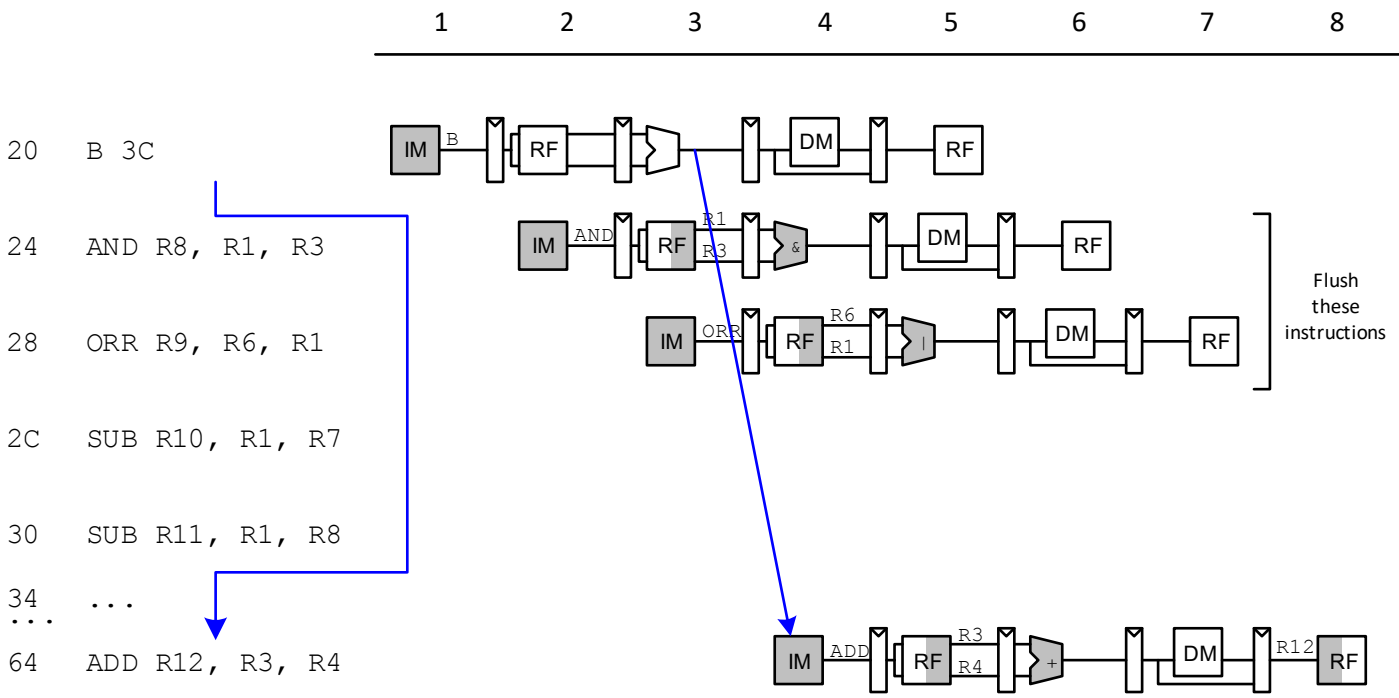
- **Determine BTA in Execute stage**
  - Branch misprediction penalty = 2 cycles
- **Hardware changes**
  - Add a branch multiplexer before *PC* register to select BTA from *ALUResultE*
  - Add *BranchTakenE* select signal for this multiplexer (only asserted if branch condition satisfied)
  - *PCSrcW* now only asserted for writes to *PC*

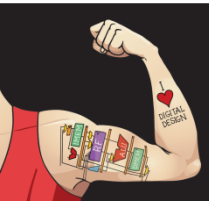


Pipelined processor with Early BTA



Control Hazards with Early BTA





Control Stalling Logic

- ***PCWrPendingF* = 1** if write to *PC* in Decode, Execute or Memory

  PCWrPendingF = PCSrcD + PCSrcE + PCSrcM

- **Stall Fetch** if *PCWrPendingF*

  StallF = ldrStallD + PCWrPendingF

- **Flush Decode** if *PCWrPendingF* OR *PC* is written in Writeback OR branch is taken

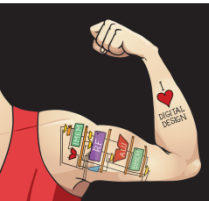
  FlushD = PCWrPendingF + PCSrcW + BranchTakenE

- **Flush Execute** if branch is taken

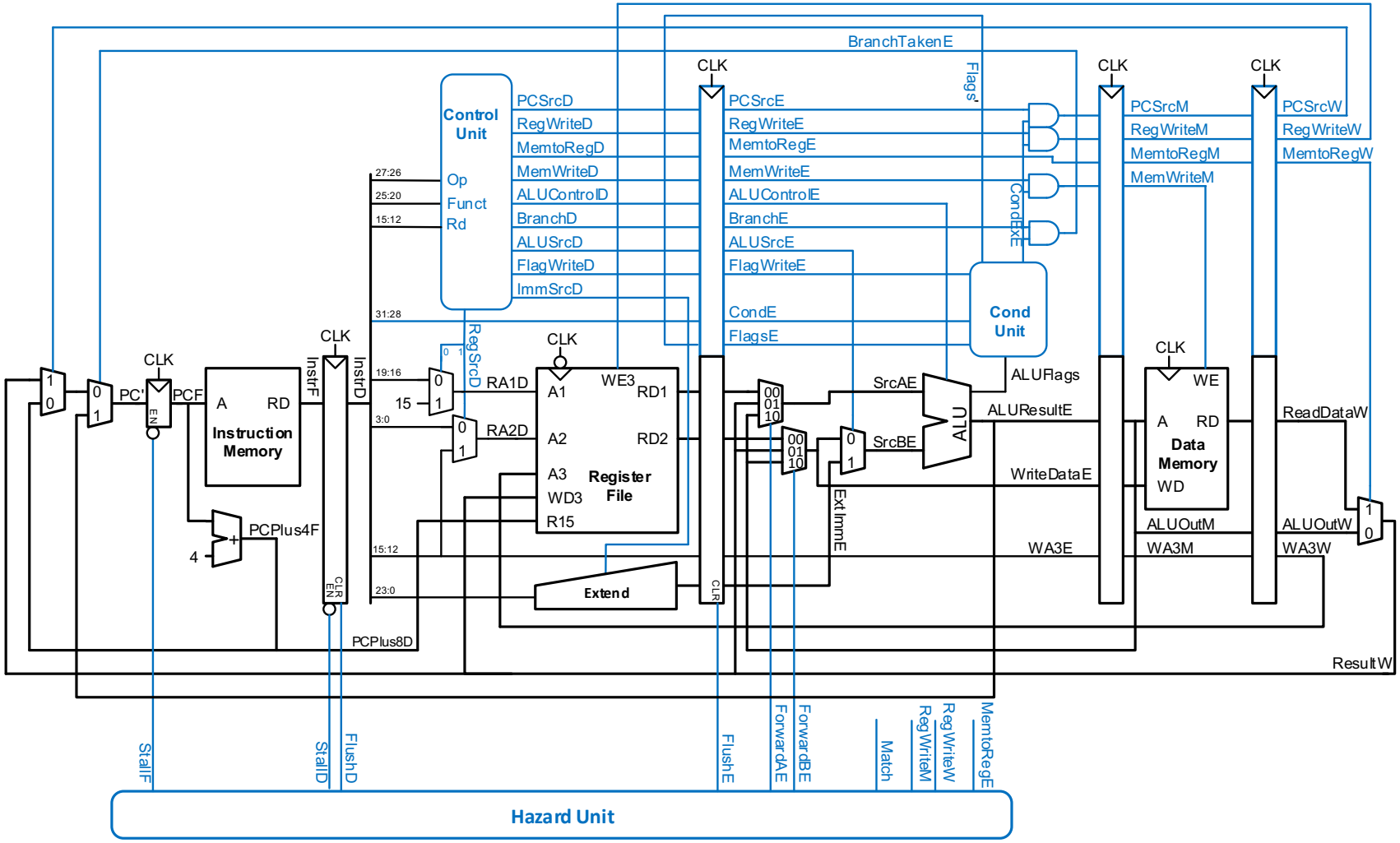
  FlushE = ldrStallD + BranchTakenE

- **Stall Decode** if *ldrStallD* (as before)

  StallD = ldrStallD

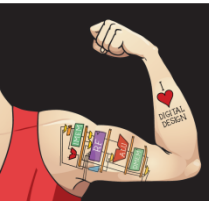


ARM Pipelined Processor with Hazard Unit



Pipelined Performance Example

- **SPECINT2000 benchmark:**
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% data processing
- **Suppose:**
  - 40% of loads used by next instruction
  - 50% of branches mispredicted
- **What is the average CPI?**



Pipelined Performance Example

- **SPECINT2000 benchmark:**
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% data processing
- **Suppose:**
  - 40% of loads used by next instruction
  - 50% of branches mispredicted
- **What is the average CPI?**
  - Load CPI = 1 when not stalling, 2 when stalling

    So, $\mathbf{CPI_{lw}} = 1(0.6) + 2(0.4) = 1.4$
  - Branch CPI = 1 when not stalling, 3 when stalling

    So, $\mathbf{CPI_{beq}} = 1(0.5) + 3(0.5) = 2$

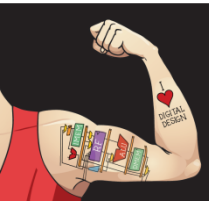
Average CPI = $(0.25)(1.4) + (0.1)(1) + (0.13)(2) + (0.52)(1)$ **= 1.23**

Pipelined Performance

- Pipelined processor critical path:

$$T_{c3} = \max [$$

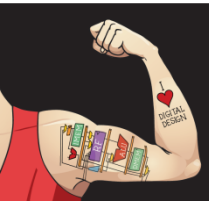
$t_{pcq} + t_{mem} + t_{setup}$	**Fetch**
$2(t_{RFread} + t_{setup})$	**Decode**
$t_{pcq} + 2t_{mux} + t_{ALU} + t_{setup}$	**Execute**
$t_{pcq} + t_{mem} + t_{setup}$	**Memory**
$2(t_{pcq} + t_{mux} + t_{RFwrite})$]	**Writeback**



Pipelined Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60
Register file write	$t_{RFwrite}$	70

Cycle time: $T_{c3} = ?$

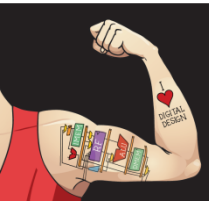


Pipelined Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60
Register file write	$t_{RFwrite}$	70

Cycle time: $T_{c3} = 2(t_{RFread} + t_{setup})$

$= 2[100 + 50]$ ps $=$ **300 ps**



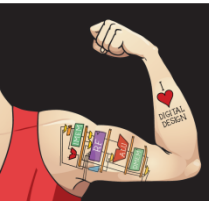
Pipelined Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) $\times$ CPI $\times$ T_c

= $(100 \times 10^9)(1.23)(300 \times 10^{-12})$

= 36.9 seconds



Processor Performance Comparison

Processor	Execution Time (seconds)	Speedup (single-cycle as baseline)
Single-cycle	84	1
Multicycle	140	0.6
Pipelined	36.9	2.28