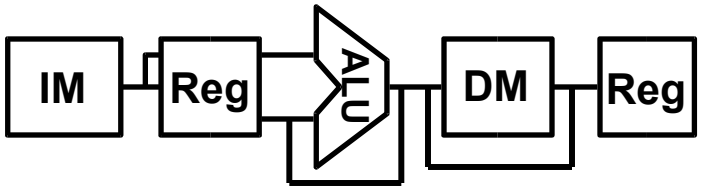
ECS 154B
Computer Architecture II
Spring 2009

Pipelining Datapath and Control
§6.2-6.3

Partially adapted from slides by Mary Jane Irwin, Penn State
And Kurtis Kredo, UCD

Pipelined CPU

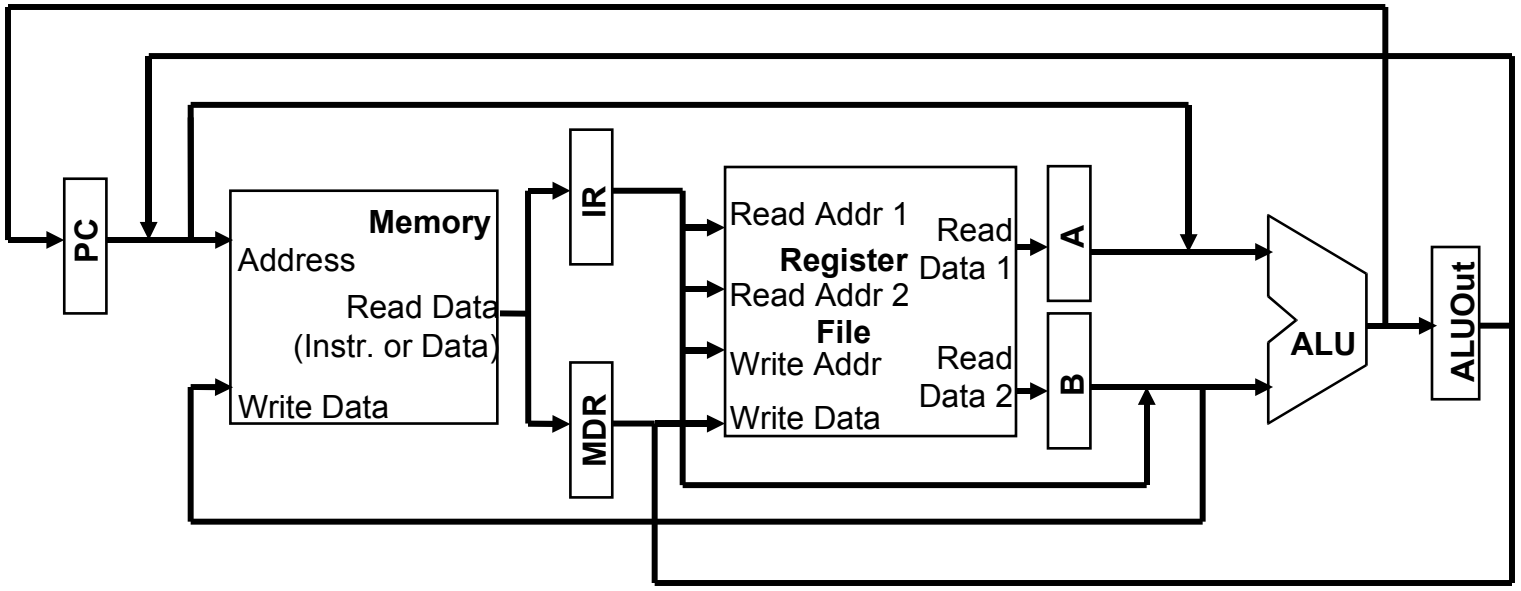
- Break execution into five stages



- Corresponds to the five instruction cycles
  - Fetch from instruction memory (IM)
  - Decode and fetch registers (Reg)
  - Execute the operation in the ALU
  - Access data memory (DM)
  - Write result back to register (Reg)

State Registers

- How do we store values across pipeline stages?
  - Multi-cycle MIPS introduced state registers
  - IR, MDR, B, A introduced
  - Sufficient for a pipelined CPU?



Pipelined State Registers

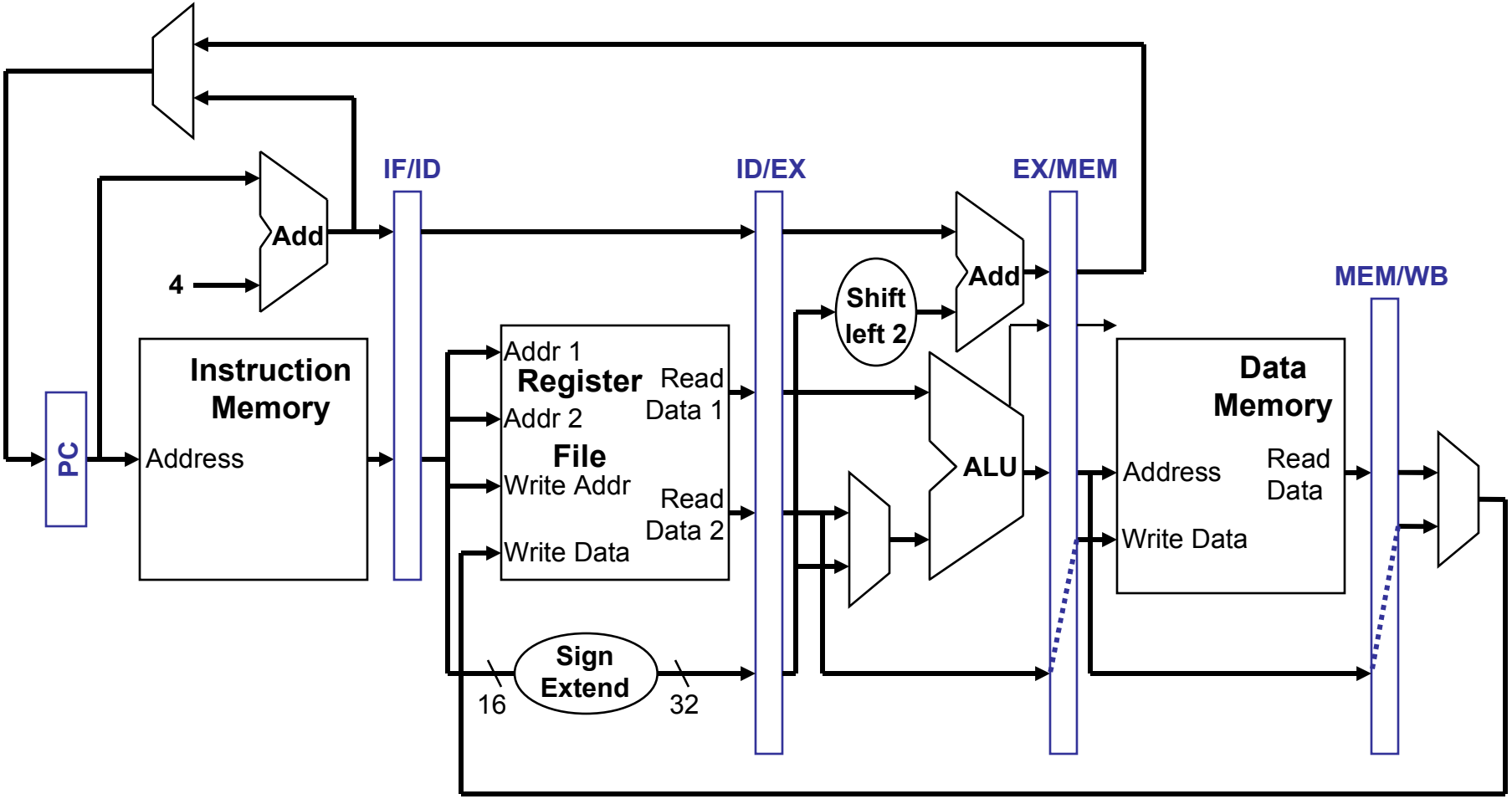
- Each must maintain its own state
- Any information needed by a later stage must be passed along
- Consider PC + 4
  - –
  - –

Pipelined State Registers

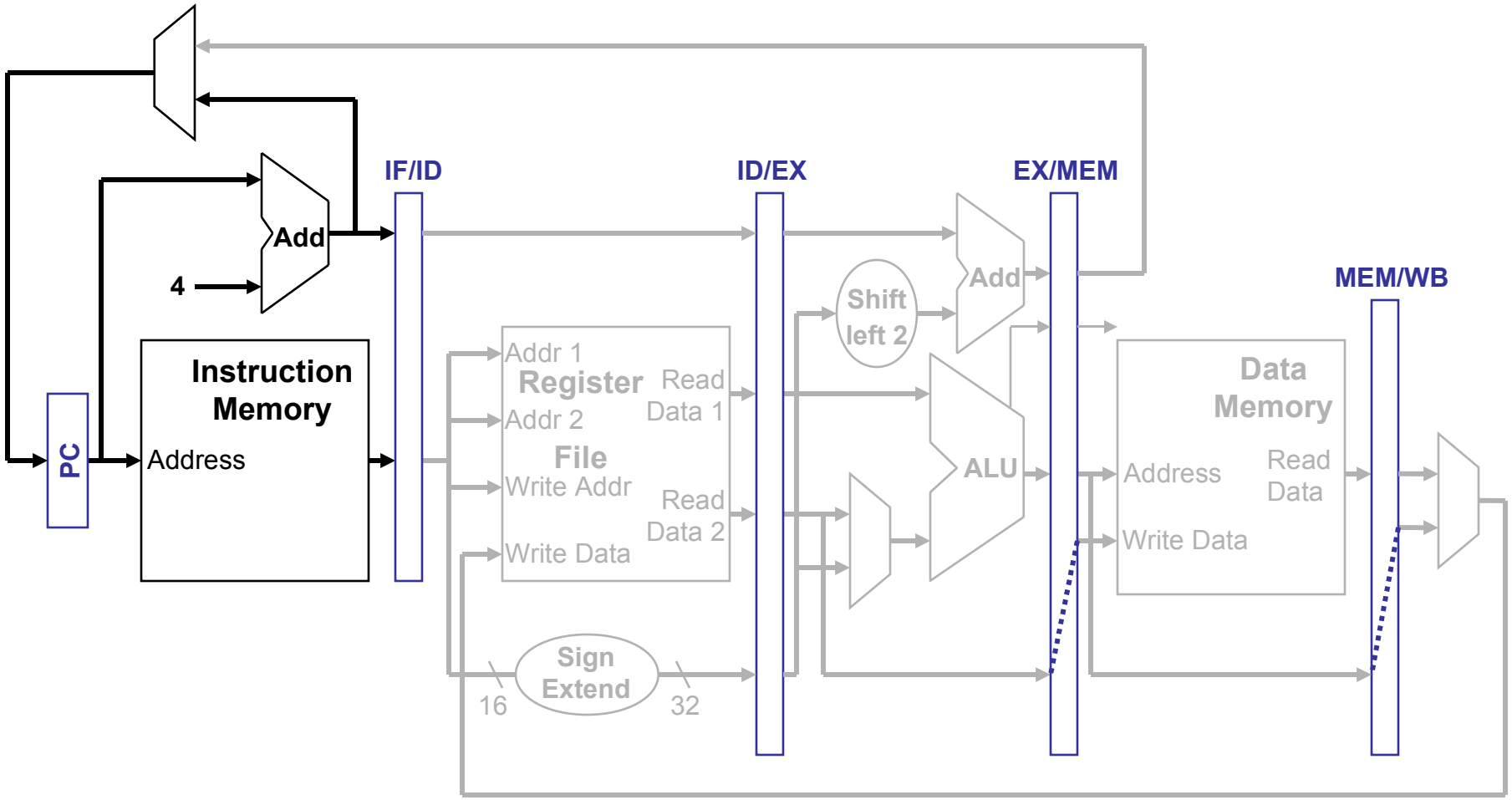
- Each instruction must maintain its own state
- Any information needed by a later stage must be passed along
- Consider PC + 4
  - Computed during Fetch stage (why?)
  - Needed by Execute stage (why?)

Pipelined State Registers

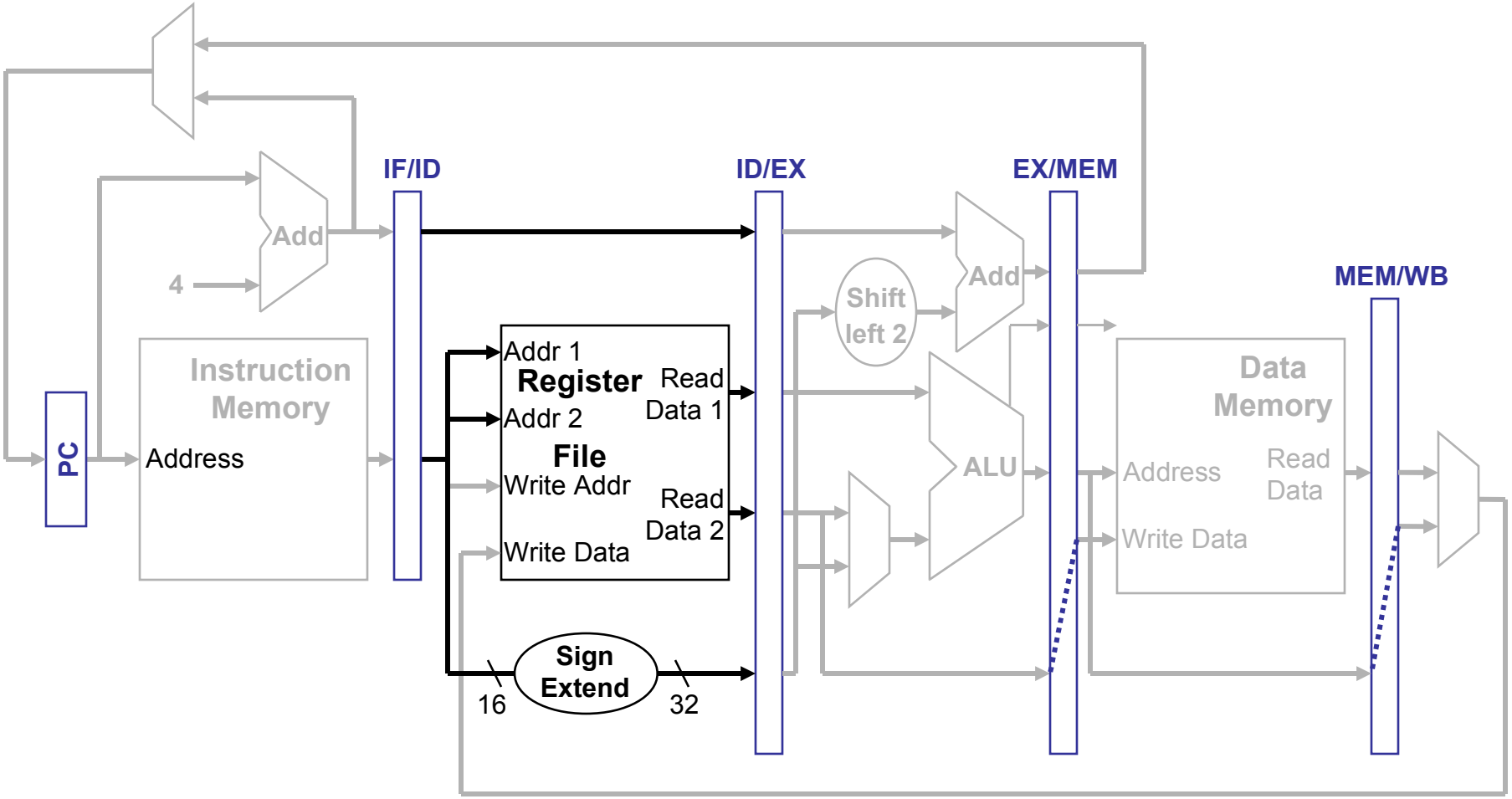
- More registers necessary for a pipelined CPU



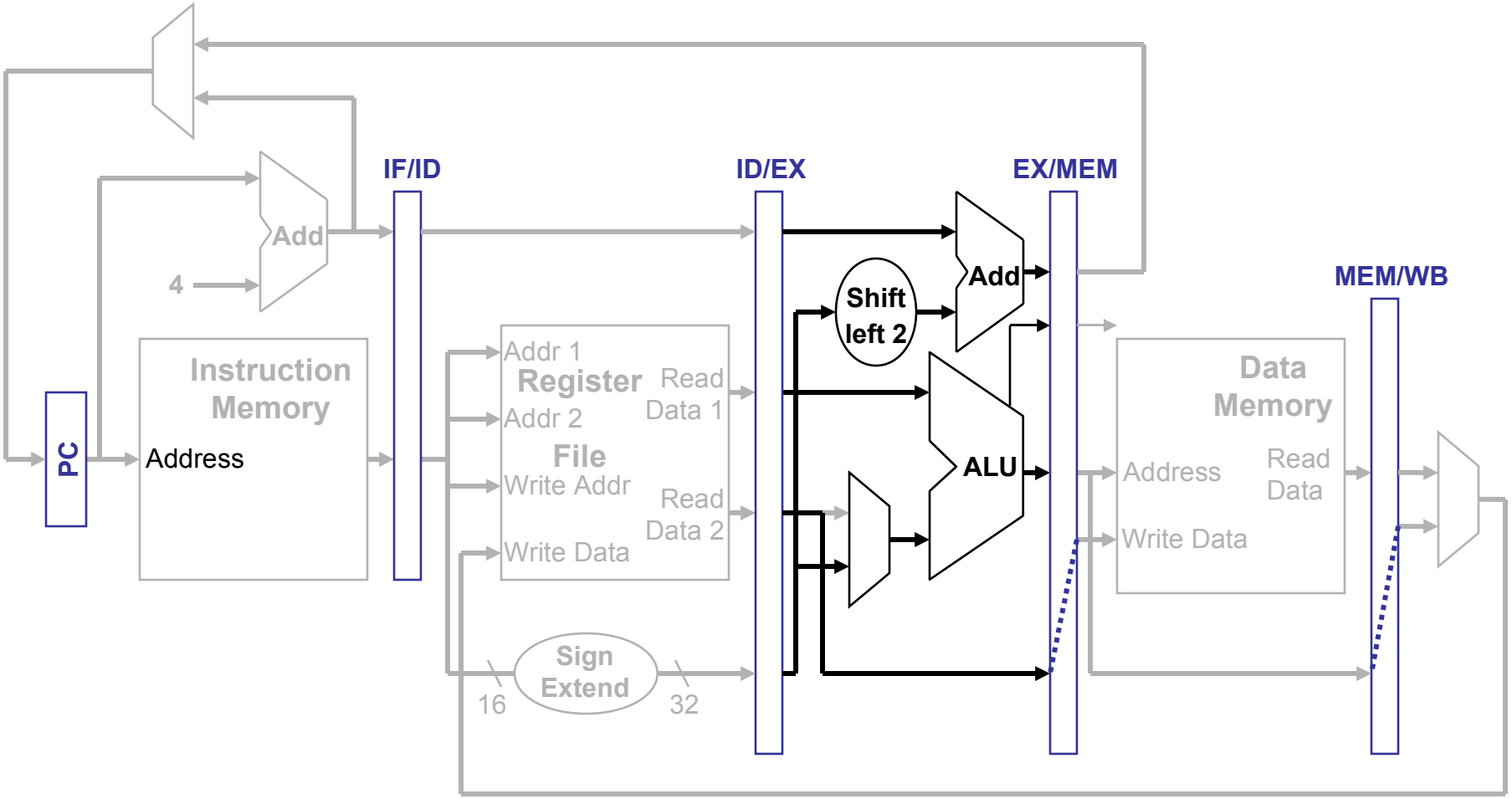
Load Word Example (Fetch)



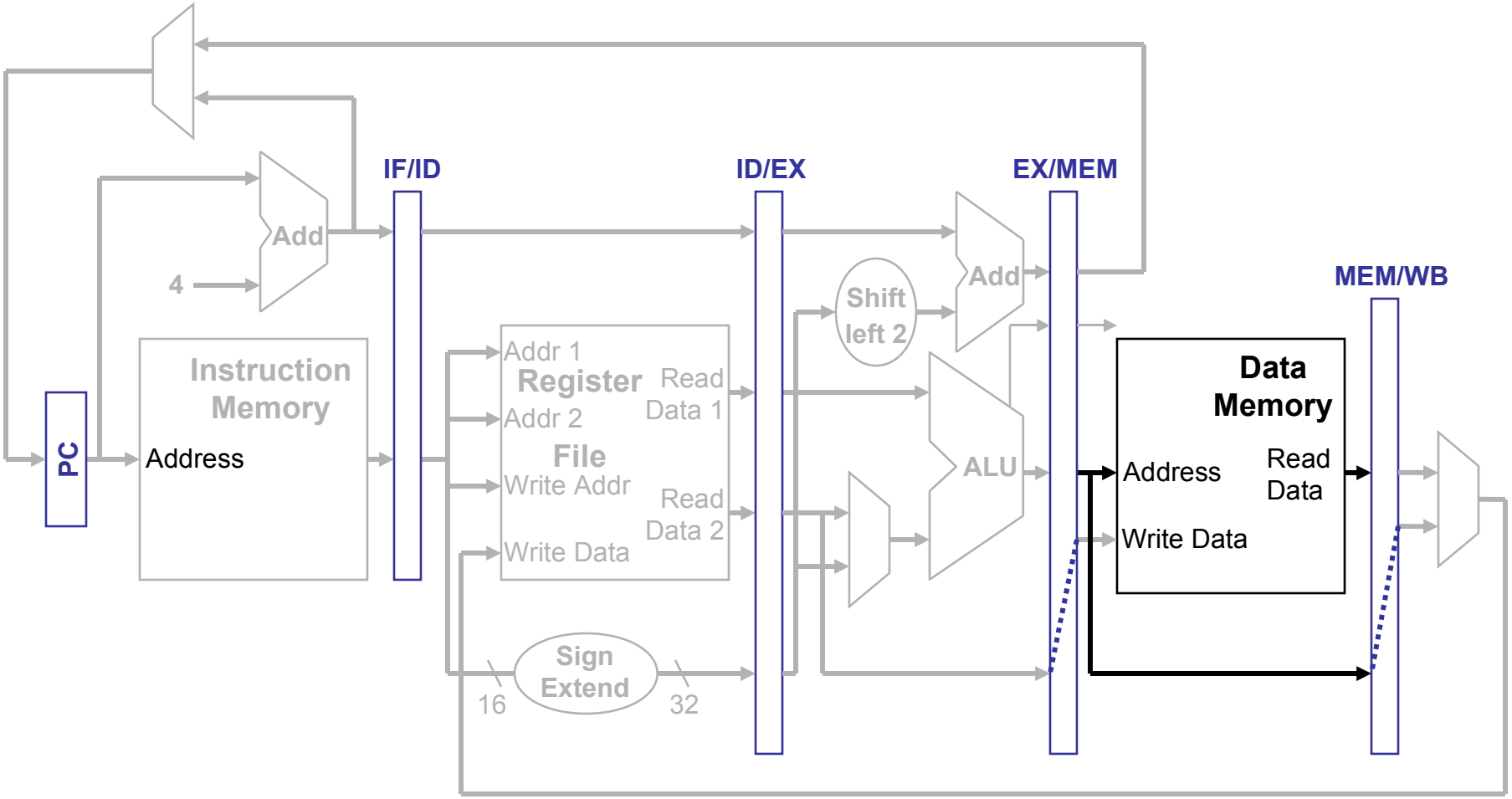
Load Word Example (Decode)



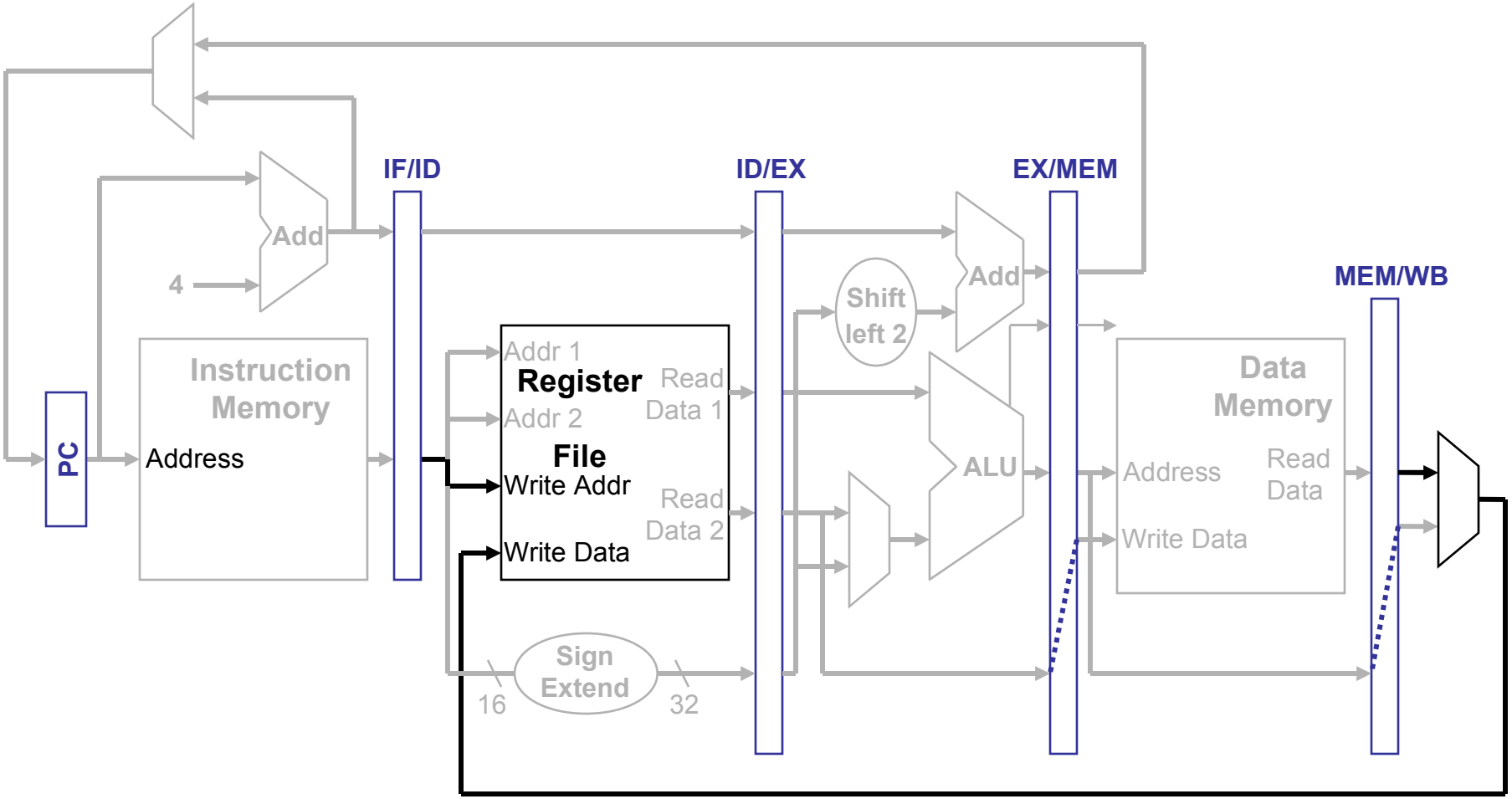
Load Word Example (Execute)



Load Word Example (Memory)

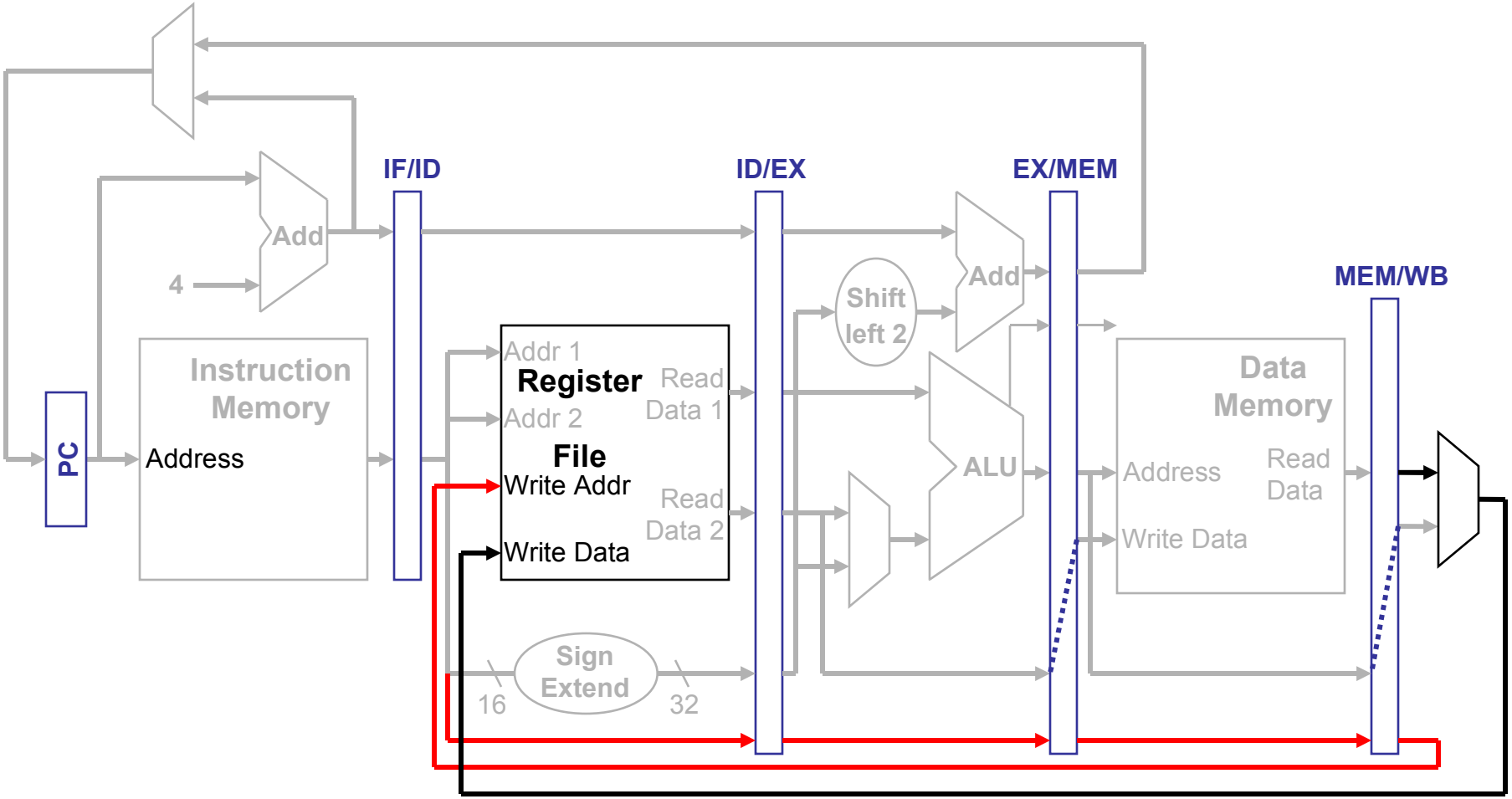


Load Word Example (Write Back)



Load Word Example (Write Back)

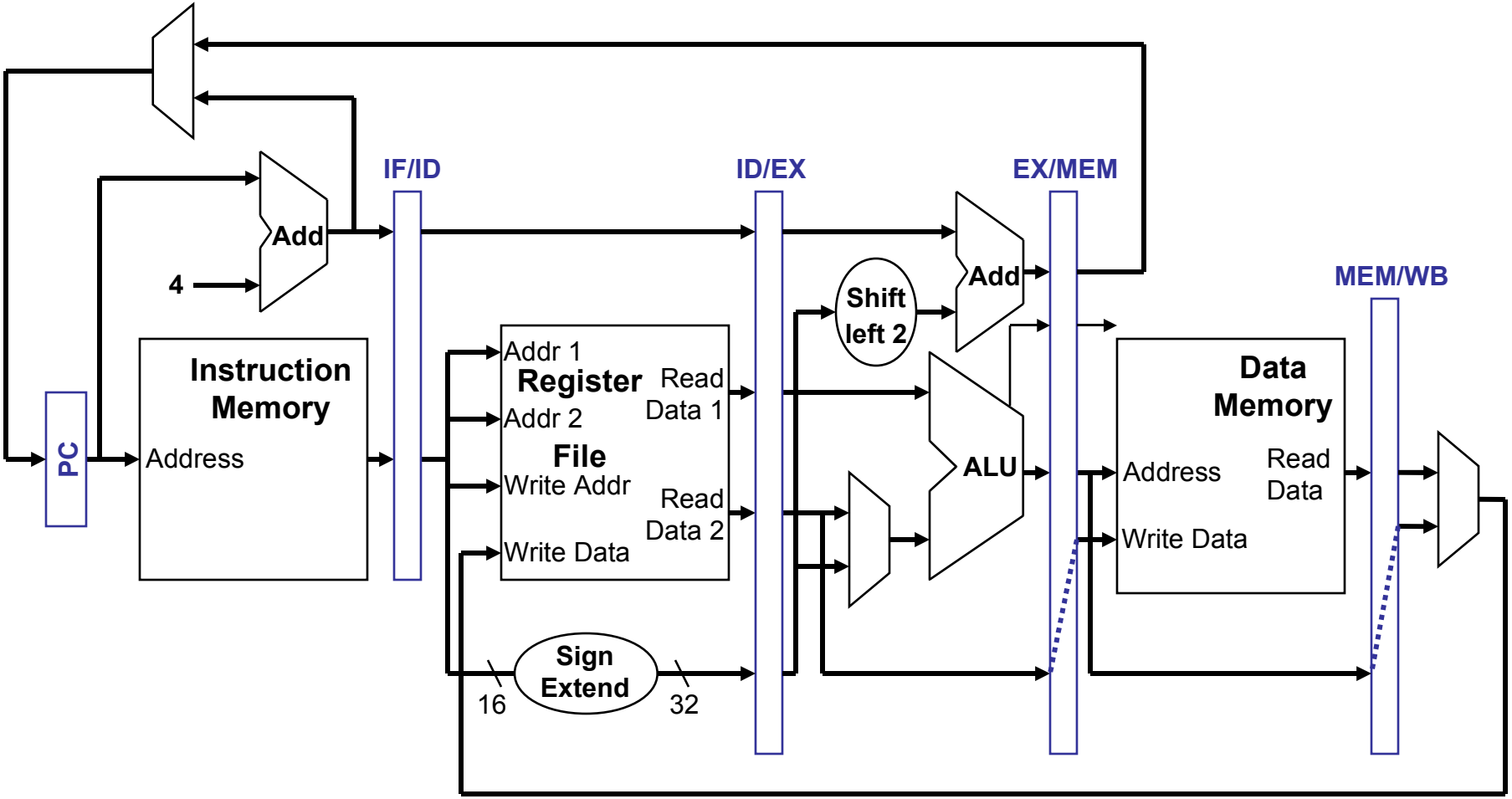
- All required values must pass through registers



Pipelined Instructions

- With the multi-cycle CPU, instructions could take a different number of cycles to complete
- Can we do this with a pipelined CPU?
  - R-type
  - sw
  - Jump

Pipelined Instructions



Pipeline Control

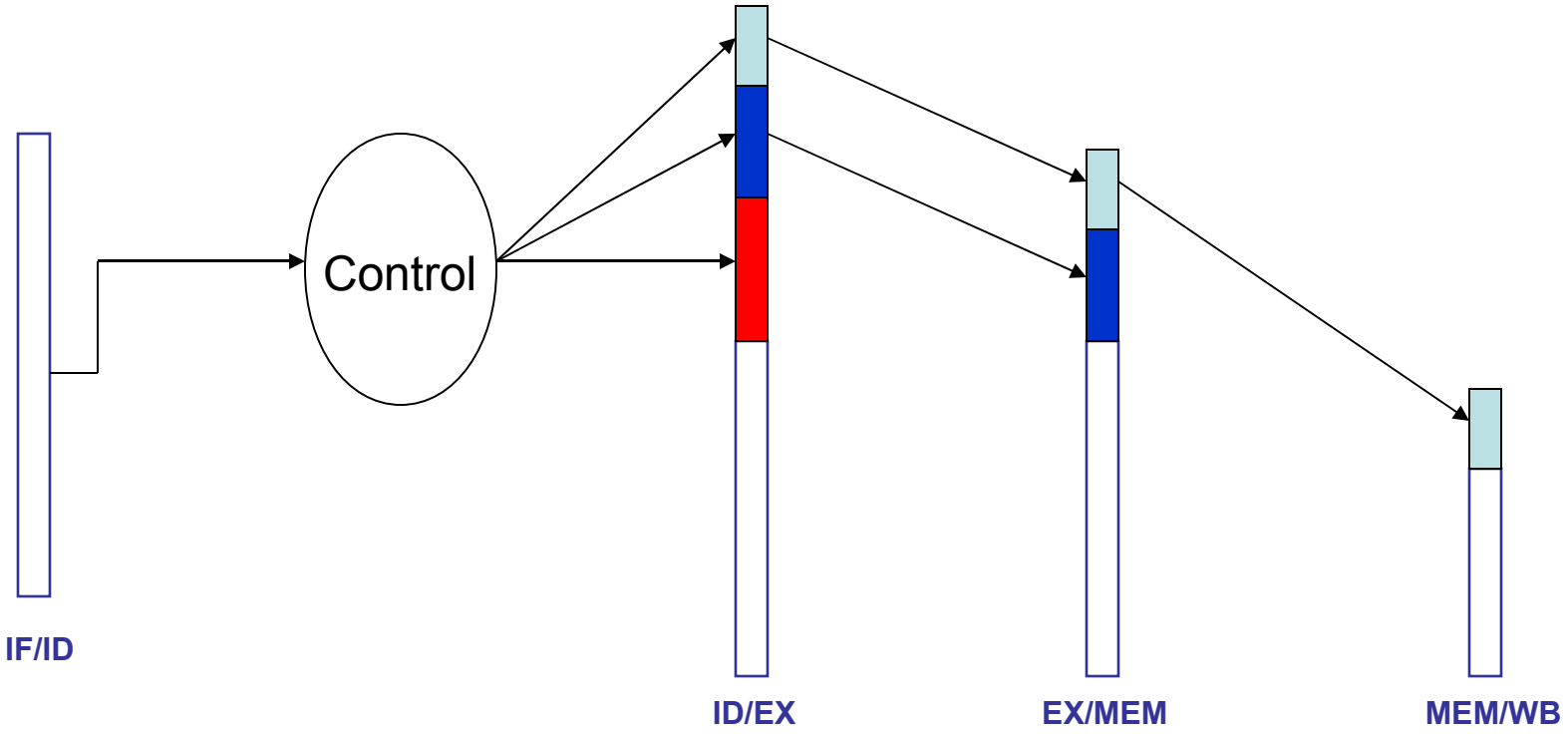
- What about control signals?
  - Different for each
  - Required at
- Where does the control unit go?

Pipeline Control

- What about control signals?
  - Different for each instruction
  - Required at different stages
- Where does the control unit go?

Pipeline Control

- Pass control signals using the state registers
  - No need to pass after they are used



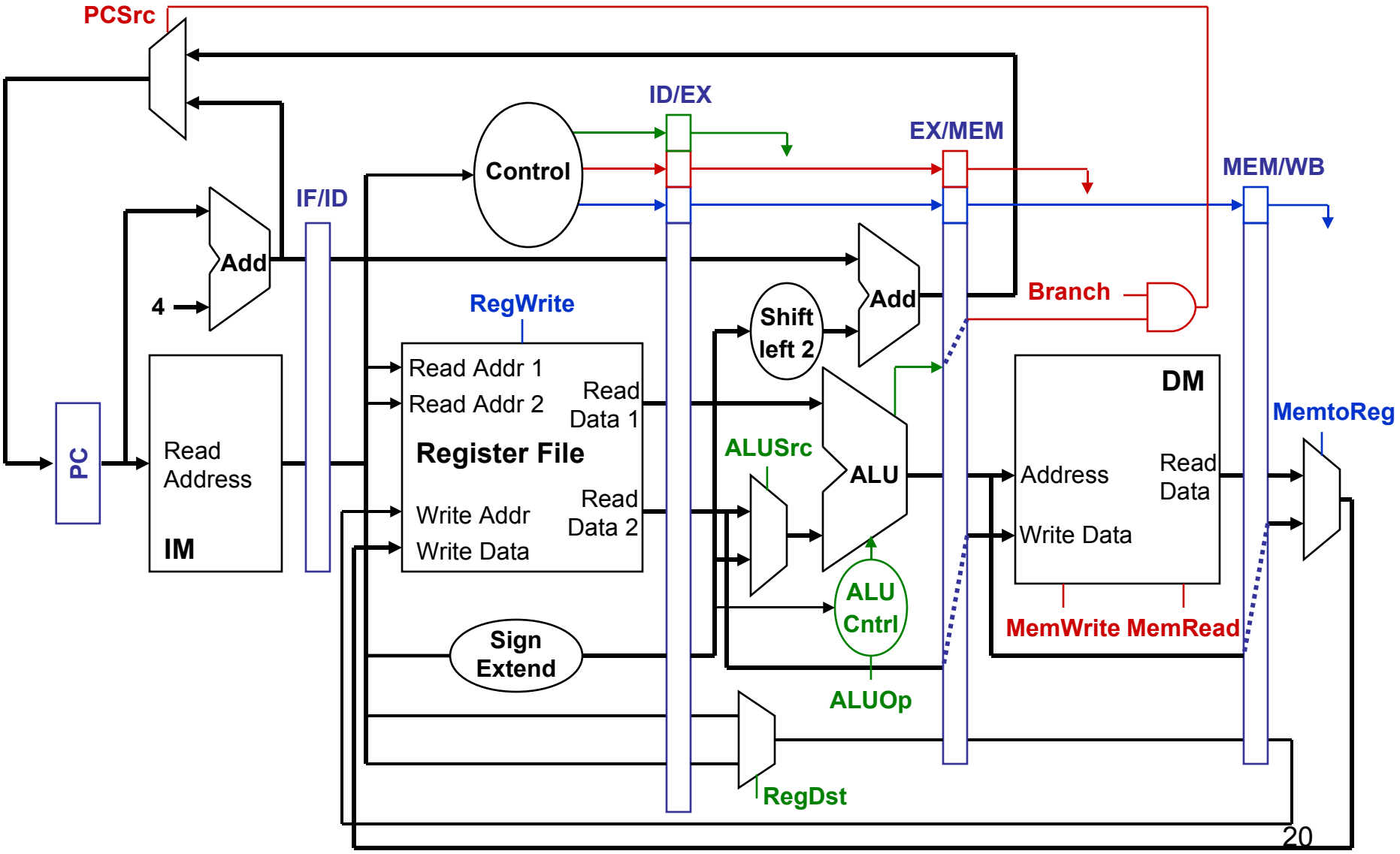
Pipeline Control Signals

- Execute signals
  - –
  - –
  - –
  - –
- Memory access signals
  - –
  - –
- Write back signals
  - –
  - –

Pipeline Control Signals

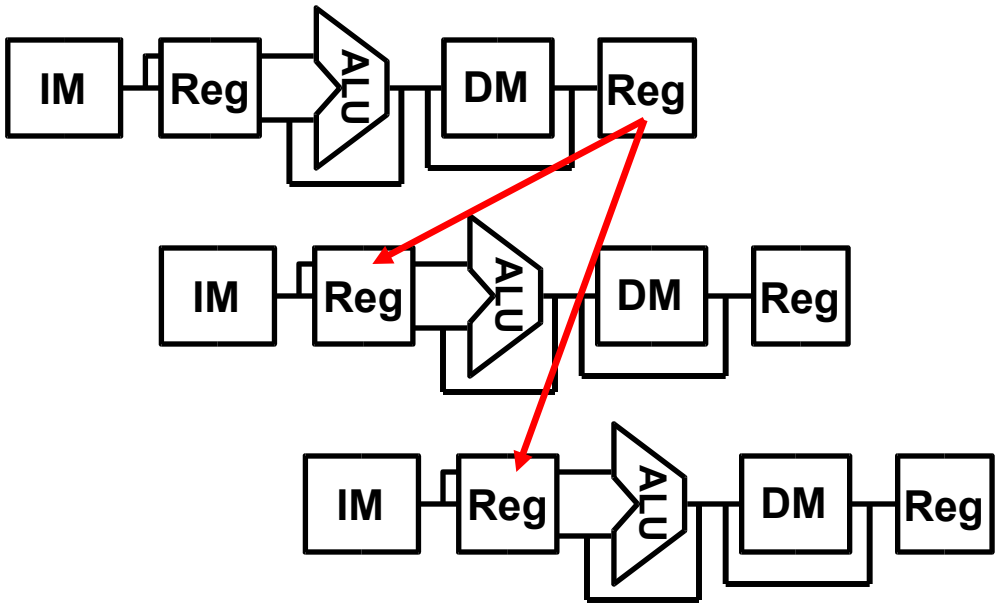
- Execute signals
  - RegDst
  - ALUOp[1..0]
  - ALUSrc
  - Branch
- Memory access signals
  - MemRead
  - MemWrite
- Write back signals
  - RegWrite
  - MemtoReg

The Pipelined CPU So Far



Data Hazard Review

- Caused when data is needed before it is ready
  - Read before write: Result of previous instruction needed by later instruction
  - Load use: Value in data memory needed by later instruction



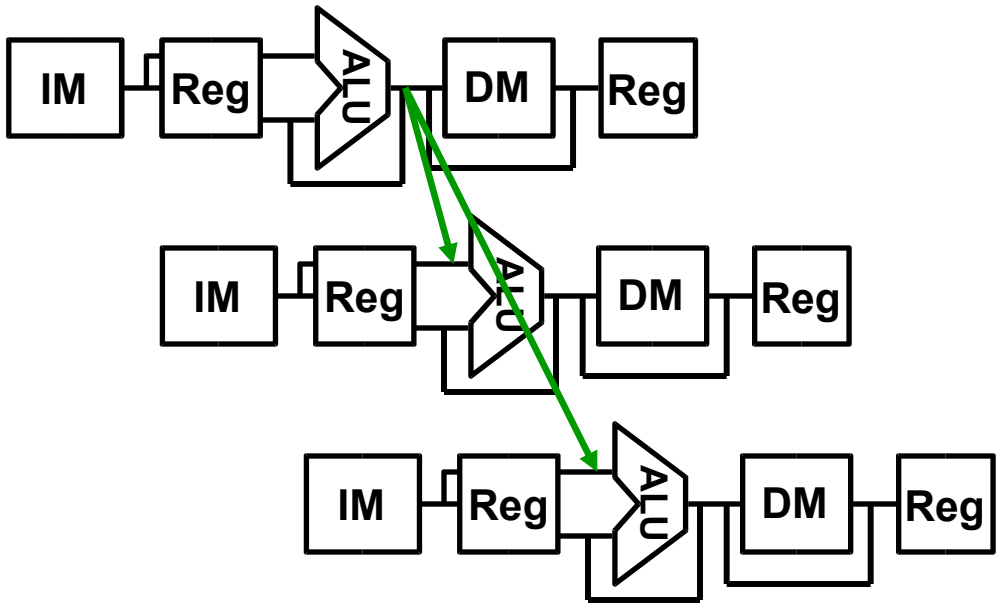
Read After Write Hazard Solution

- Stalling always an option
- Forwarding data improves CPI over stalling

add $4, $5, $6

add $8, $4, $6

add $10, $9, $4



Data Forwarding

- Take the result from the point that it exists in any of the pipeline state registers and forward it to the that need it that cycle
- For ALU functional unit: the inputs can come from any pipeline register rather than just from ID/EX by
  - adding the inputs of the ALU
  - connecting the in or to either (or both) of the EX's stage inputs
  - adding the proper to control the new muxes
- Other functional units may need similar forwarding logic
- With forwarding, the CPU can achieve a CPI of 1 even in the presence of data dependencies

Data Forwarding

- Take the result from the earliest point that it exists in any of the pipeline state registers and forward it to the functional units (e.g., the ALU) that need it that cycle
- For ALU functional unit: the inputs can come from any pipeline register rather than just from ID/EX by
  - adding multiplexors to the inputs of the ALU
  - connecting the Rd write data in EX/MEM or MEM/WB to either (or both) of the EX's stage Rs and Rt ALU mux inputs
  - adding the proper control hardware to control the new muxes
- Other functional units may need similar forwarding logic
- With forwarding, the CPU can achieve a CPI of 1 even in the presence of data dependencies

Data Forwarding Conditions

- Only forward when state changes
  -
  -
  -
- Forwarding unnecessary in other cases
- Forward if either source register needs it

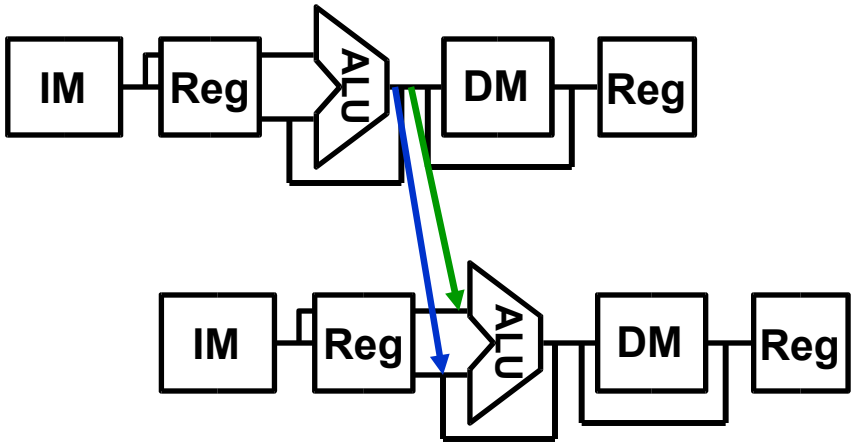
Data Forwarding Conditions

- Only forward when state changes
  - Use RegWrite control signal
  - Don’t forward if destination is $0
  - Forward if previous destination current source
- Forwarding unnecessary in other cases
- Forward if either source register needs it

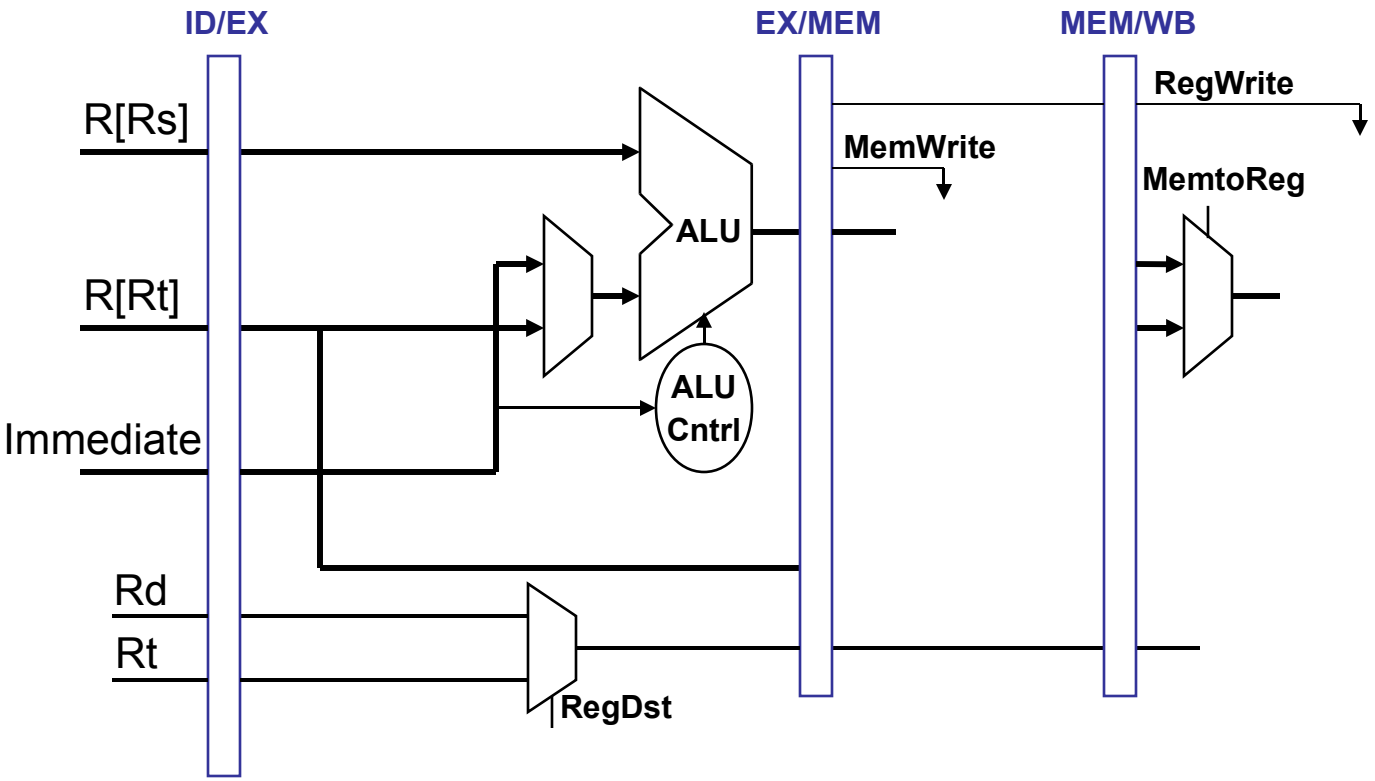
EX/MEM Forwarding

- Register value needed by next instruction
  - Calculated by ALU this clock cycle
  - Needed as input to ALU on next clock cycle

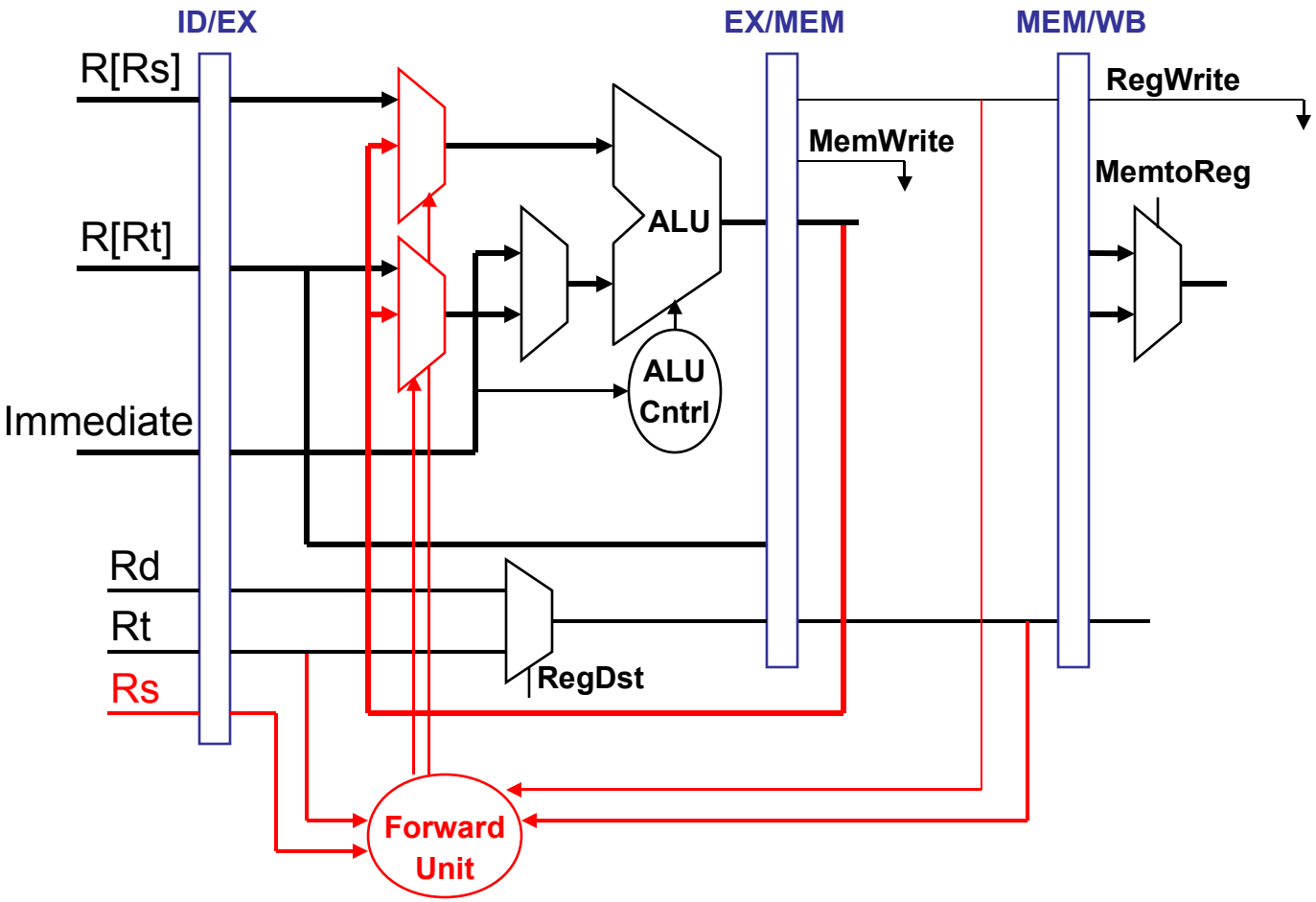
```
add $4, $5, $6

add $8, $4, $7
      or
add $8, $7, $4
```



EX/MEM Forwarding

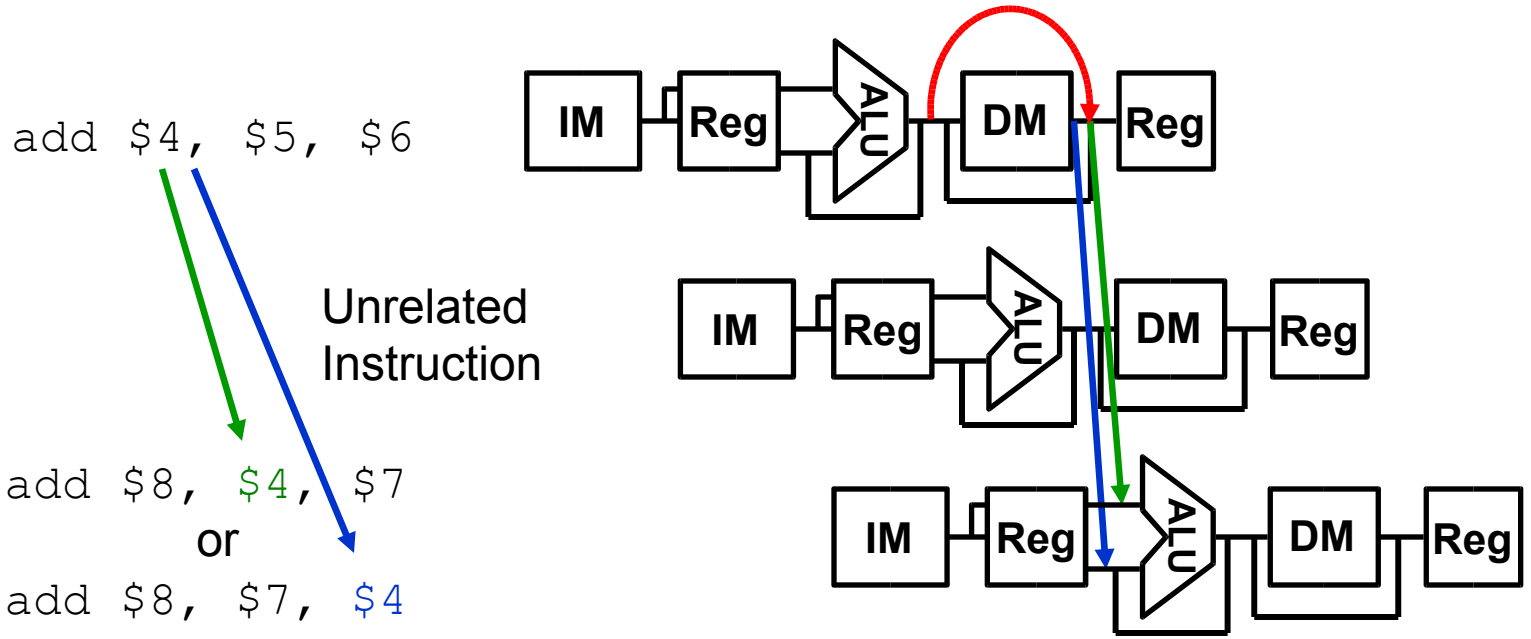


EX/MEM Forwarding

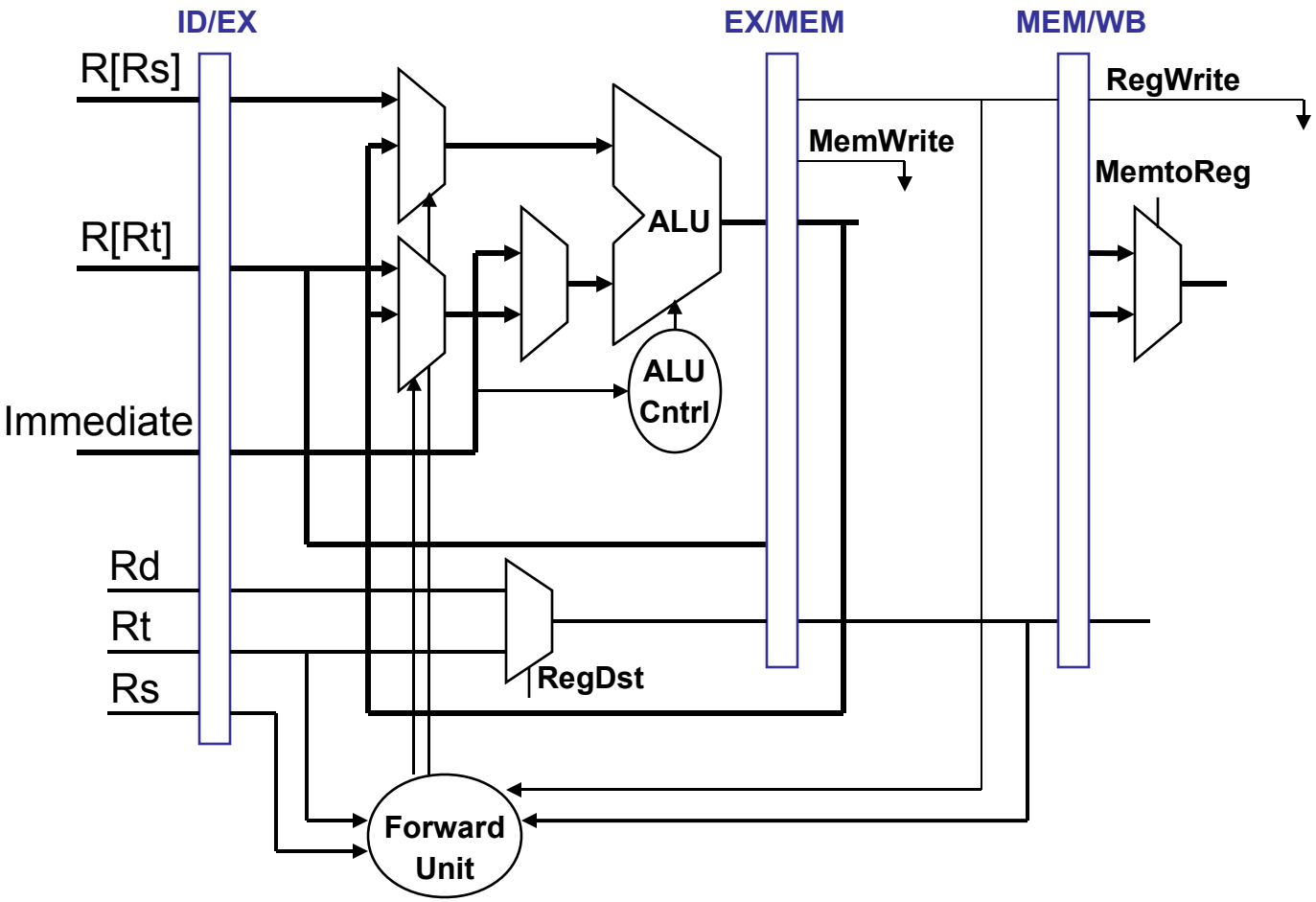


MEM/WB Forwarding

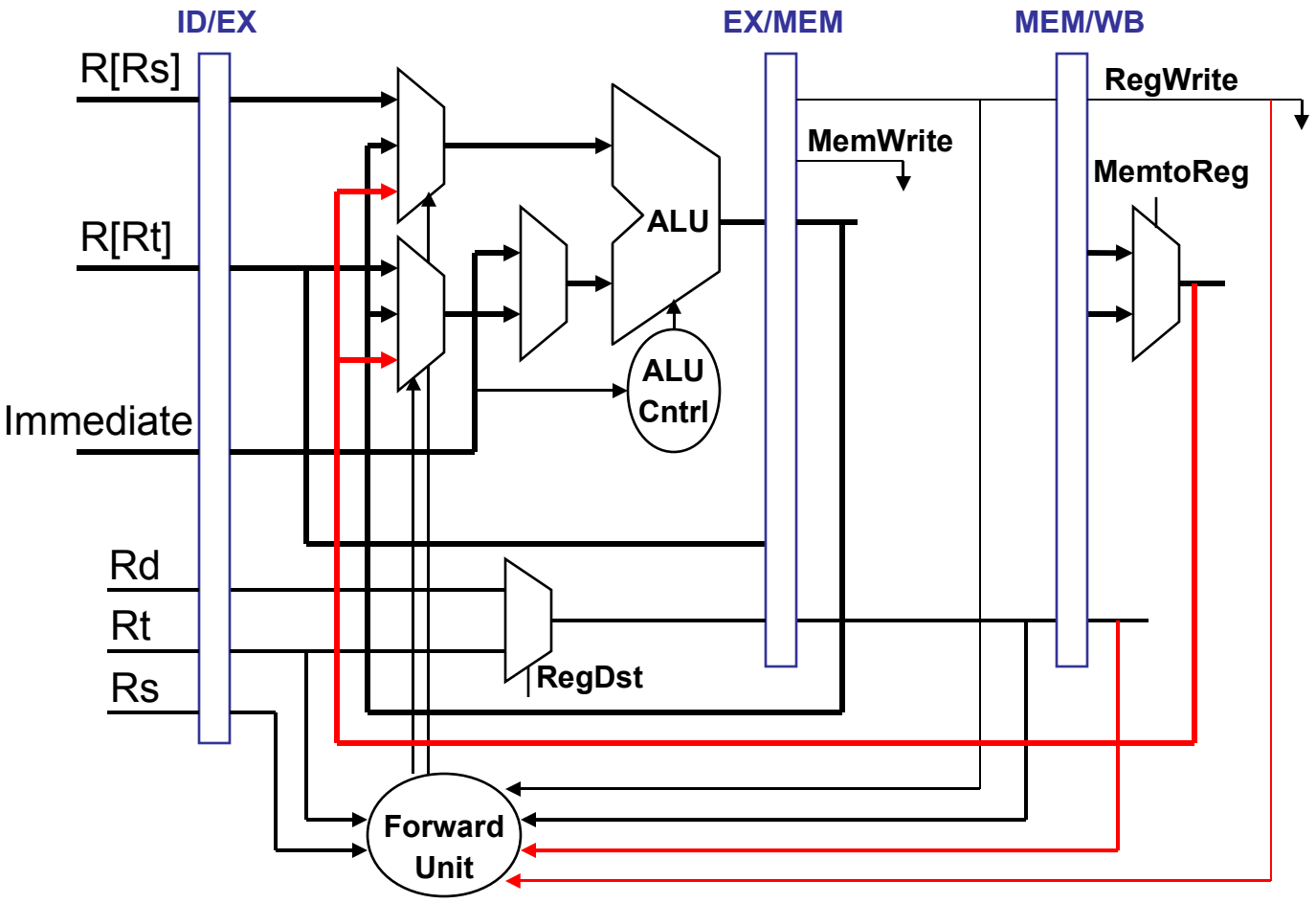
- Register value needed two instructions later
  - Calculated by ALU this clock cycle
  - Needed as input to ALU in two clock cycles



MEM/WB Forwarding



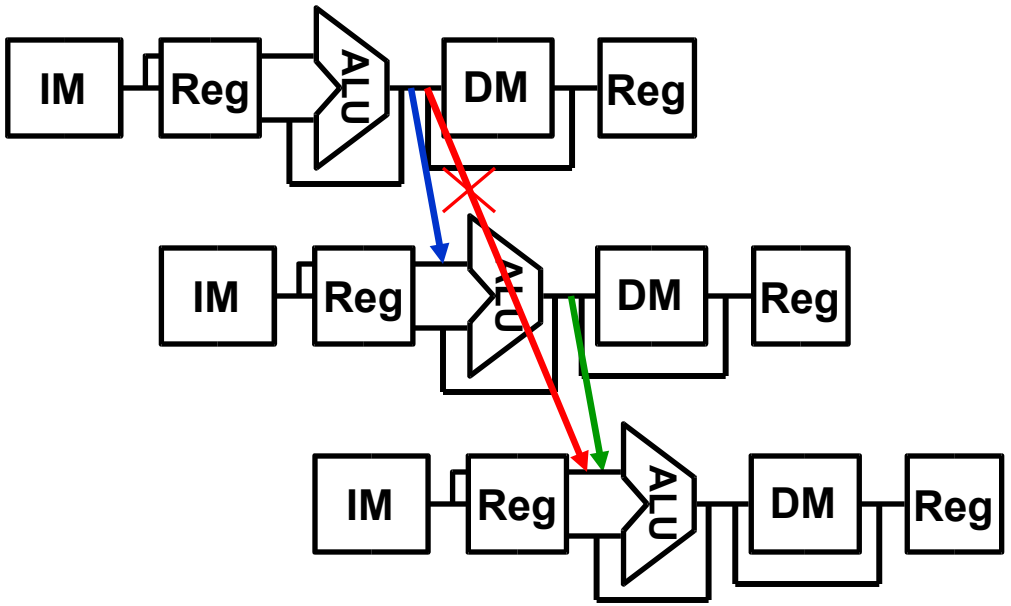
MEM/WB Forwarding



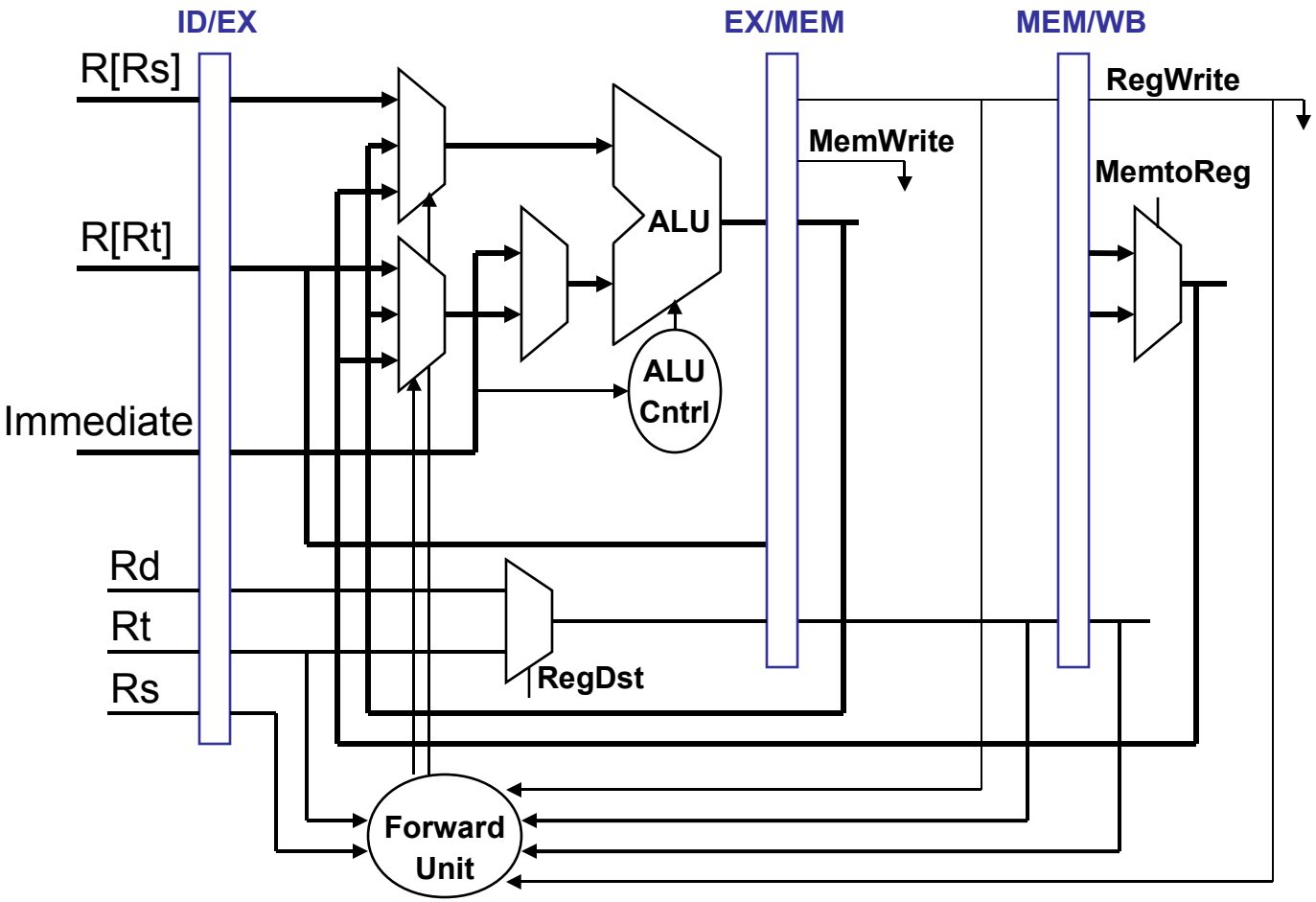
Forwarding Complication

- Forward unit must forward most recent value
  - It may appear necessary to do MEM/WB and EX/MEM forwarding simultaneously
  - Only do EX/MEM forwarding this cycle
  - Do EX/MEM forwarding again next cycle

```
add $4, $5, $6
add $4, $4, $13
add $8, $4, $7
```



Complete ALU Input Forwarding



Register Definition

- How can we specify a particular signal?
  - Each state register has a copy
  - May vary across stages
- Reference the register that contains the value
  - RegWrite value in EX/MEM state register

    `EX/MEM.RegWrite`
  - RegWrite value in MEM/WB state register

    `MEM/WB.RegWrite`

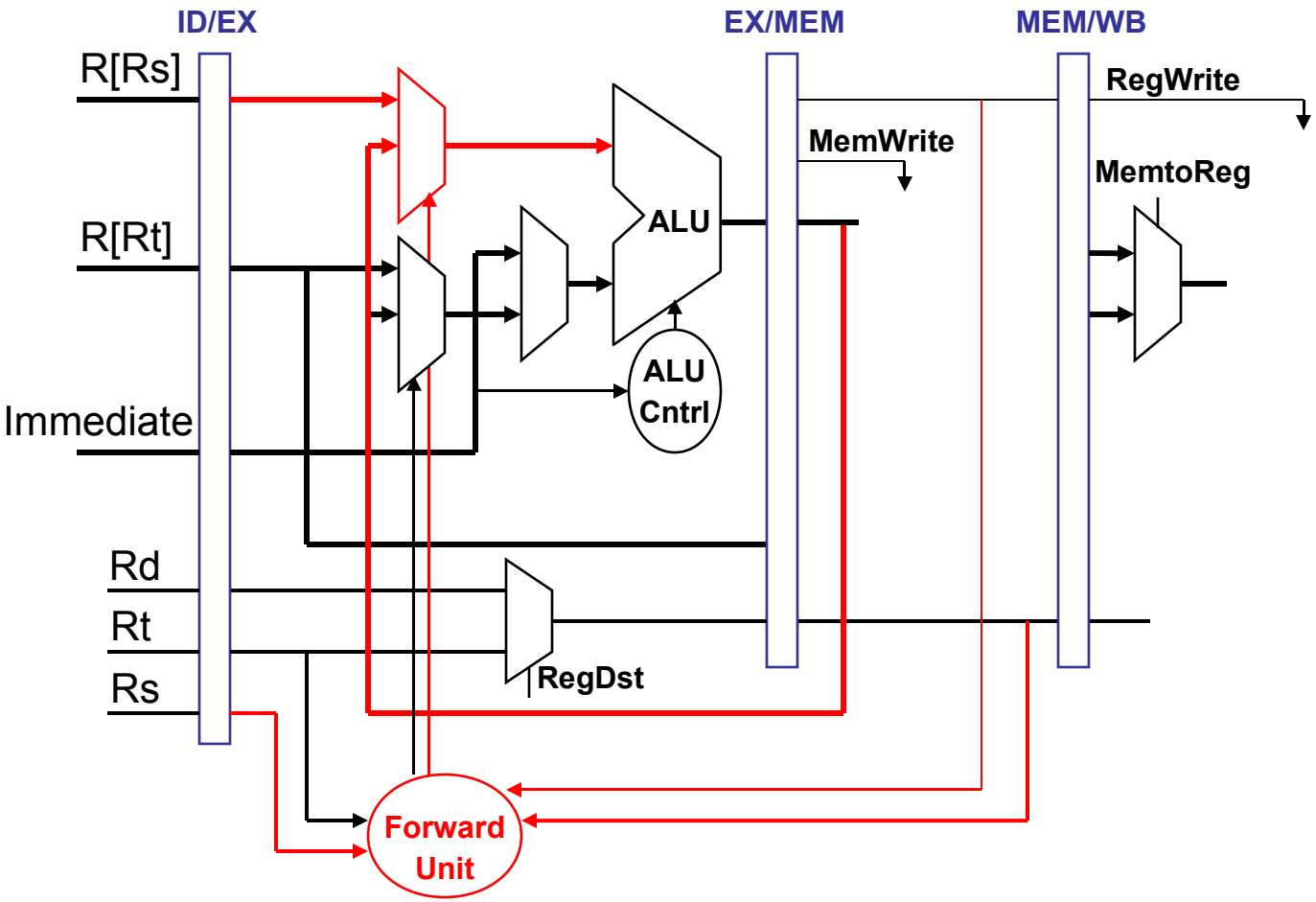
Forwarding Conditions

- We want to forward when
  - Previous instruction updates state
  - Previous destination used as current source
  - Previous destination not $0
- Data Hazard code

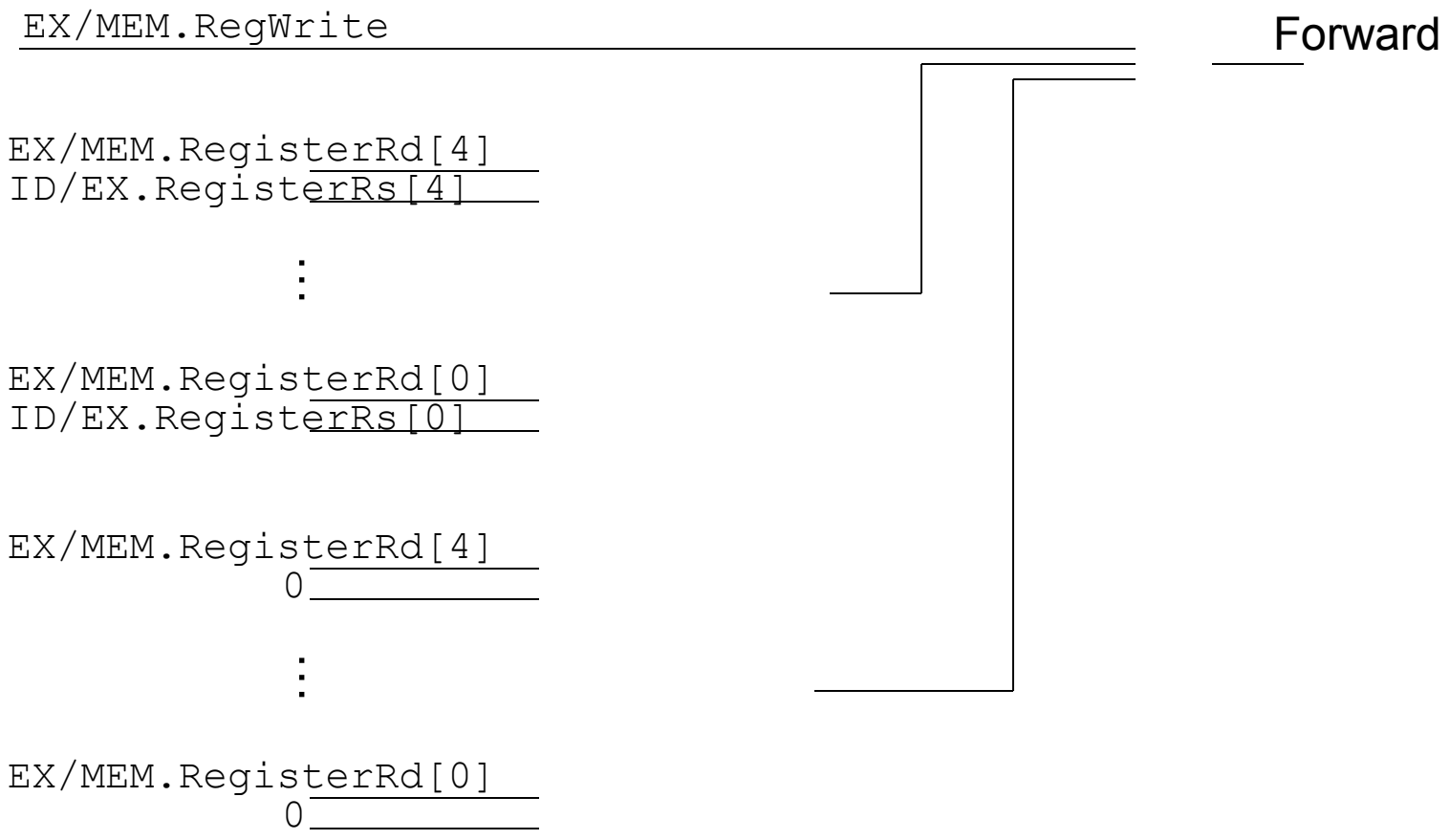
```
add $4, $5, $6
sub $8, $4, $9
```

- How do we do this in hardware?

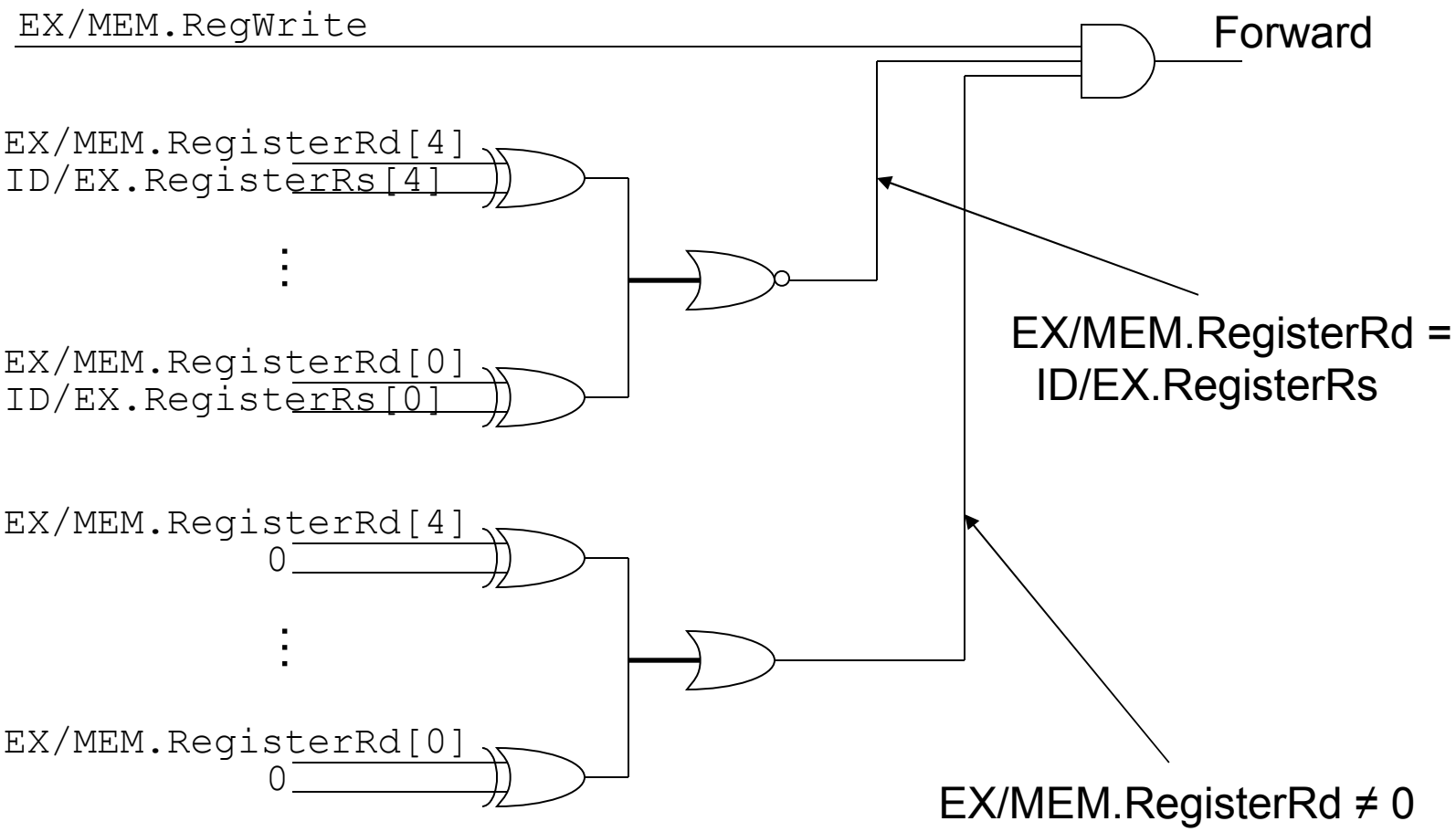
Forwarding Unit



Forwarding Unit Details



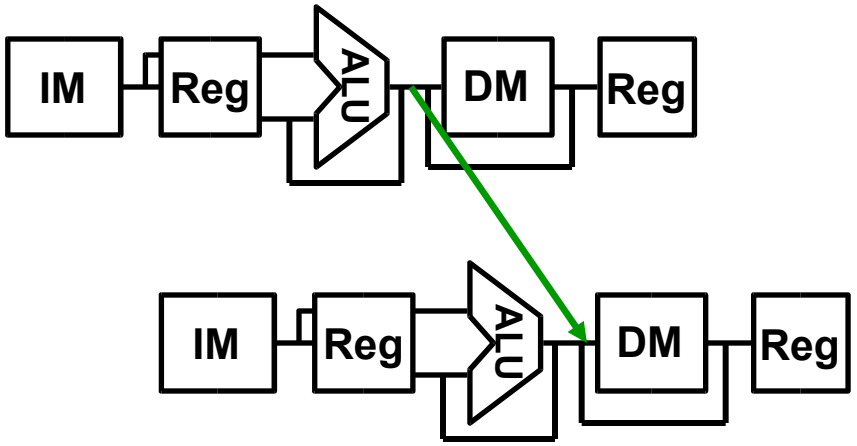
Forwarding Unit Details



Other Forwarding Possible

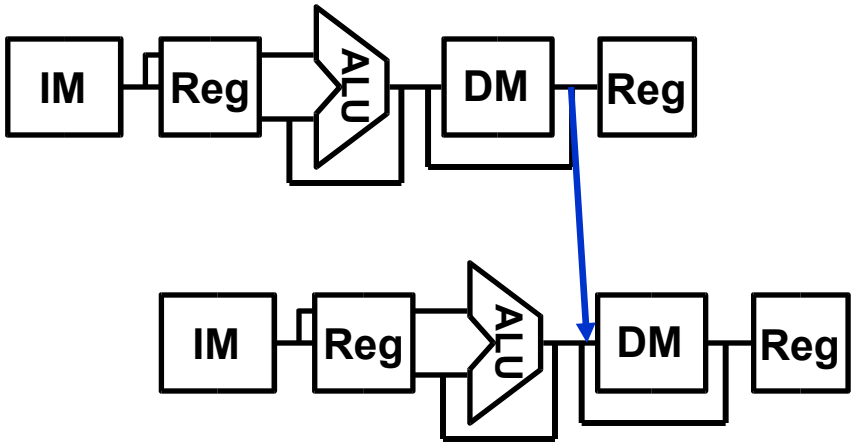
- Forwarding to Data Memory

```
add $4, $5, $6
sw $4, 40($7)
```



- Data memory to data memory copy

```
lw $4, 16($7)
sw $4, 40($7)
```



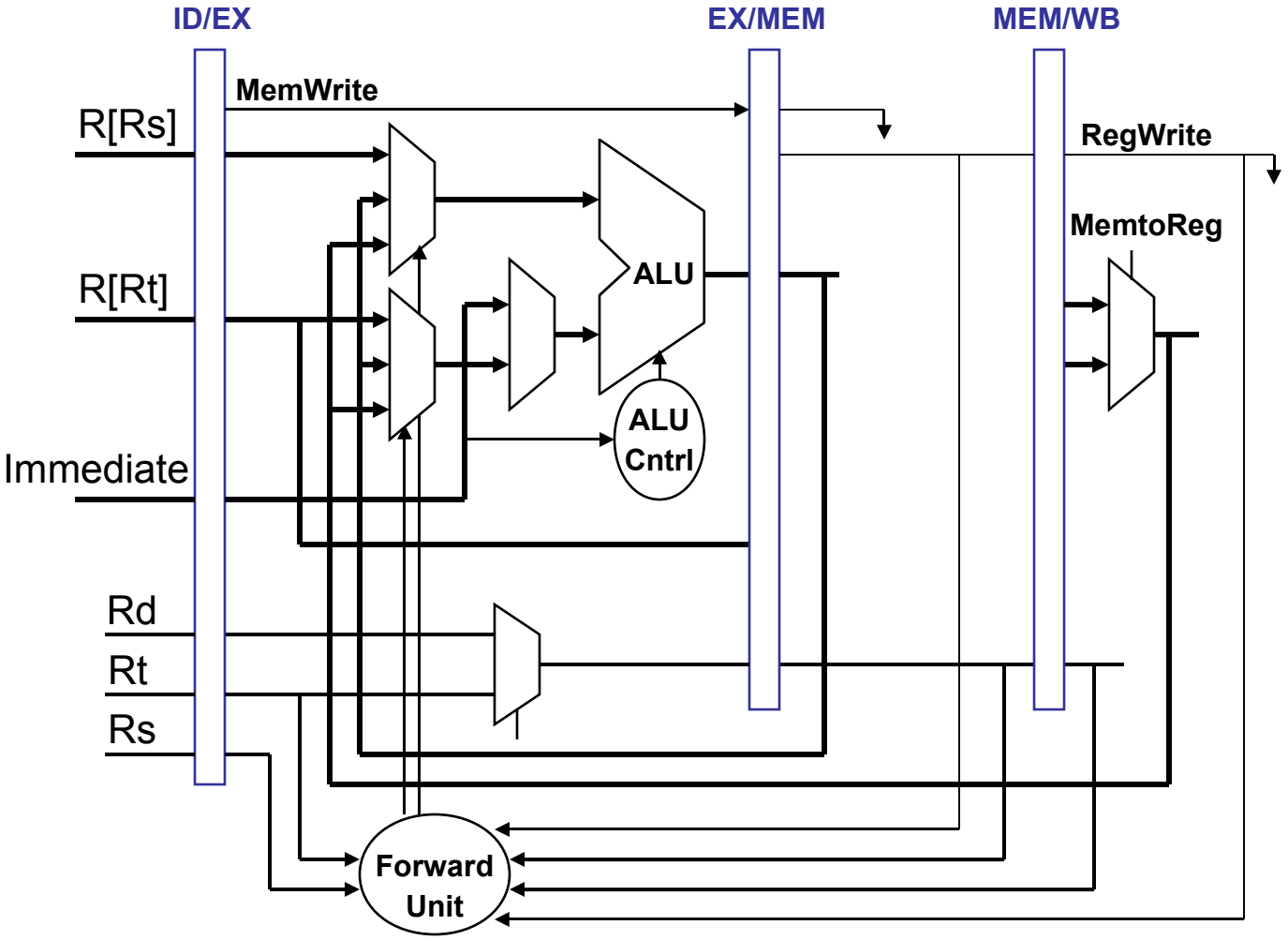
Forwarding to Memory

- What happens here?

```
add $5, $6, $7
sw $5, 8($10)
```

- Forwarding must occur, but not through ALU

Forwarding to Memory



Forwarding to Memory