Chapter 4

The Processor – Part 2

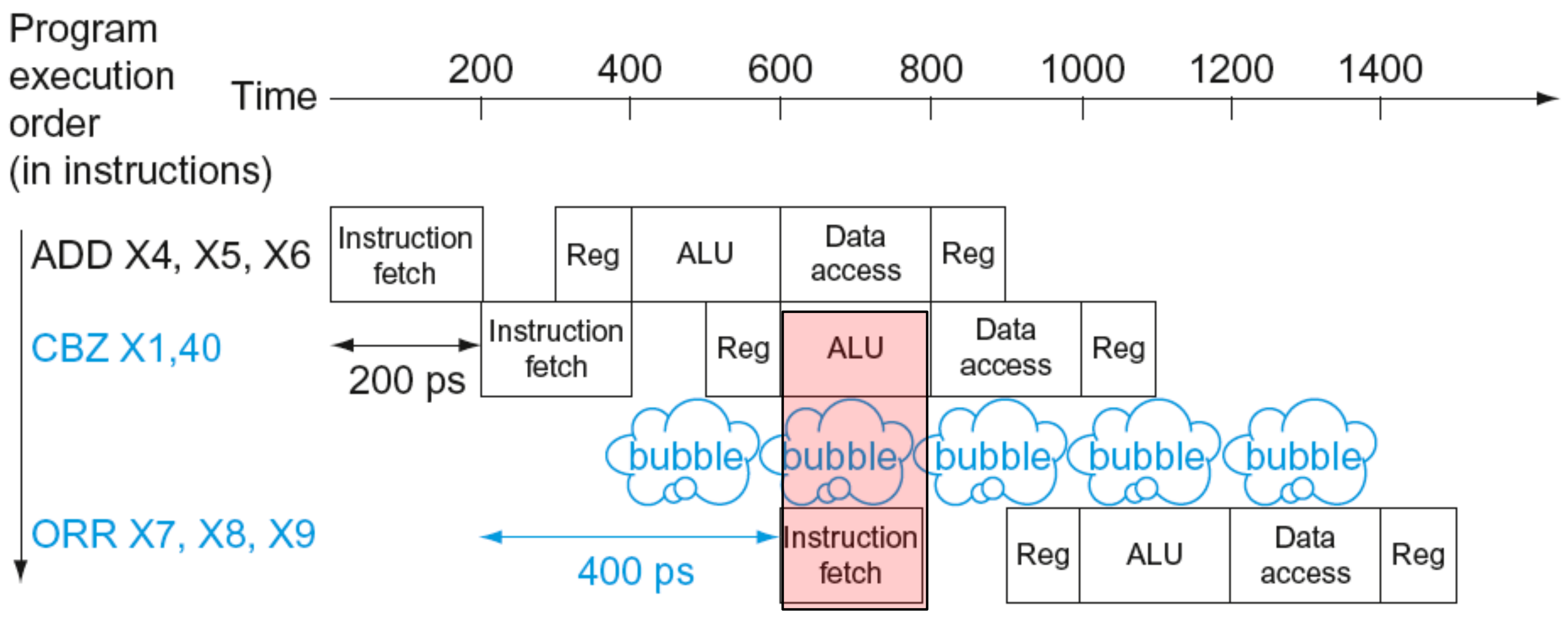
Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
- In LEGv8 pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to find the outcome in earlier stages or use prediction

```
            .
            .
            SUB X1, X2, X3
            CBZ X1, 40
PC+4 :      LDUR X3, [X0, #0]
            .
            .
            .
PC+40 :     ORR X7, X8, X9
```

Stall on Branch

- Wait until branch outcome determined before fetching next instruction



Stall on Branch

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong

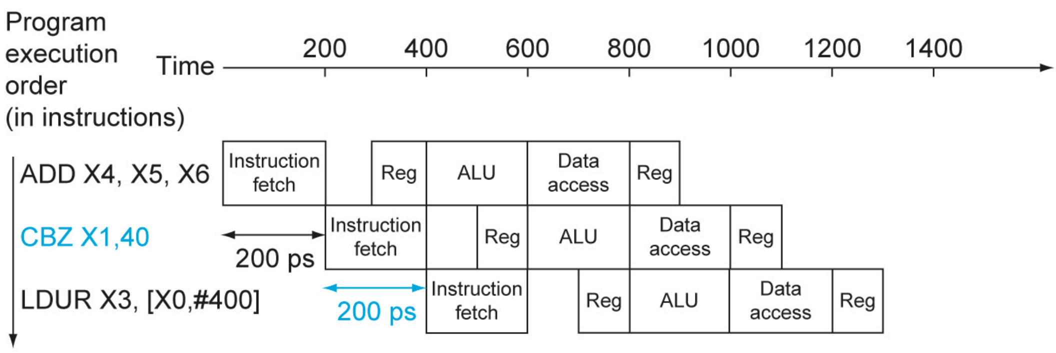
Branch Prediction

- Predict outcome of branch
  - Only stall if prediction is wrong
- In LEGv8 pipeline
  - Predict branches are not taken
    - Fetch instruction after branch instruction , with no delay

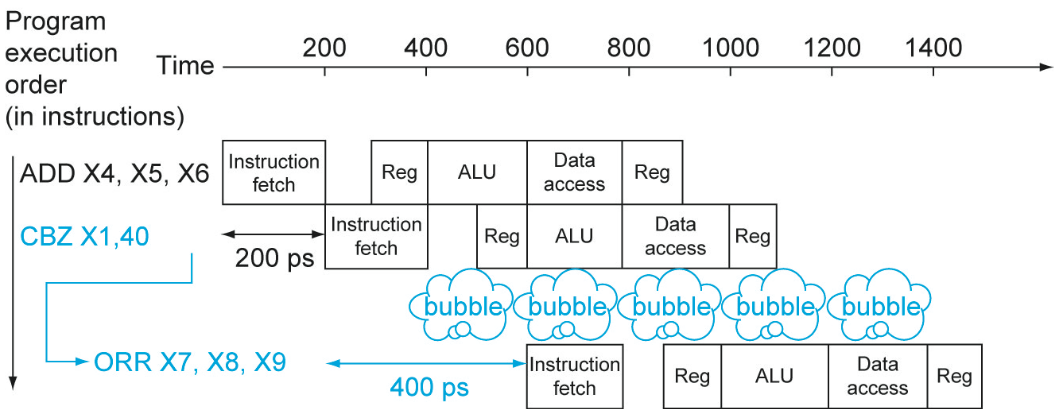
```
                .
                .
          CBZ X1, 40
PC+4 :    LDUR X3, [X0, #400]
                .
                .
PC+40 :   ORR X7, X8, X9
```

Branch Prediction

- Predict branches not taken



- What if the prediction is wrong?



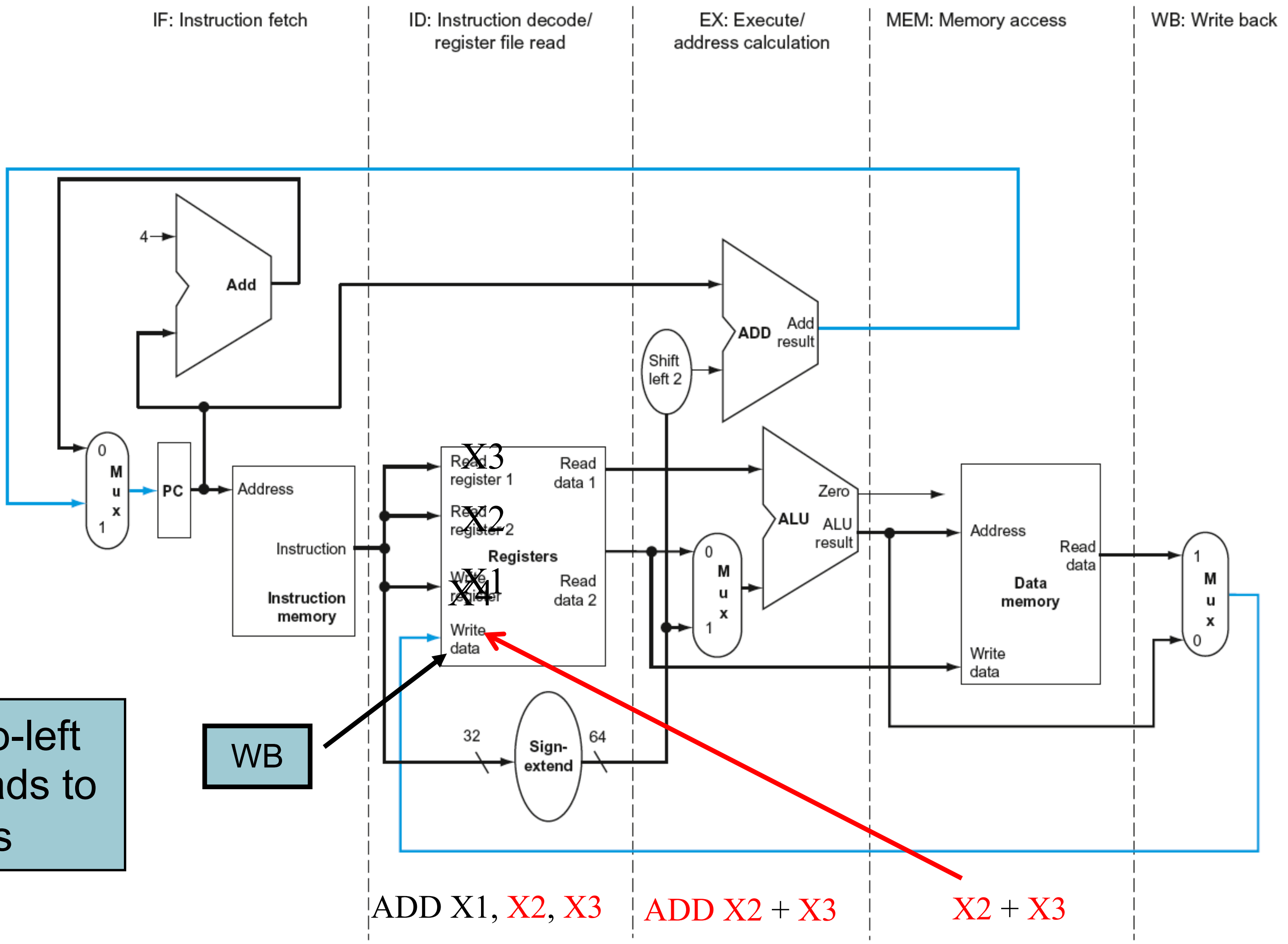
More-Realistic Branch Prediction

- Dynamic branch prediction
  - Hardware measures actual branch behavior
  - Assume future behavior will continue the trend

Pipeline Summary (So Far)

- Pipelining improves performance by increasing instruction throughput
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

LEGv8 Pipelined Datapath



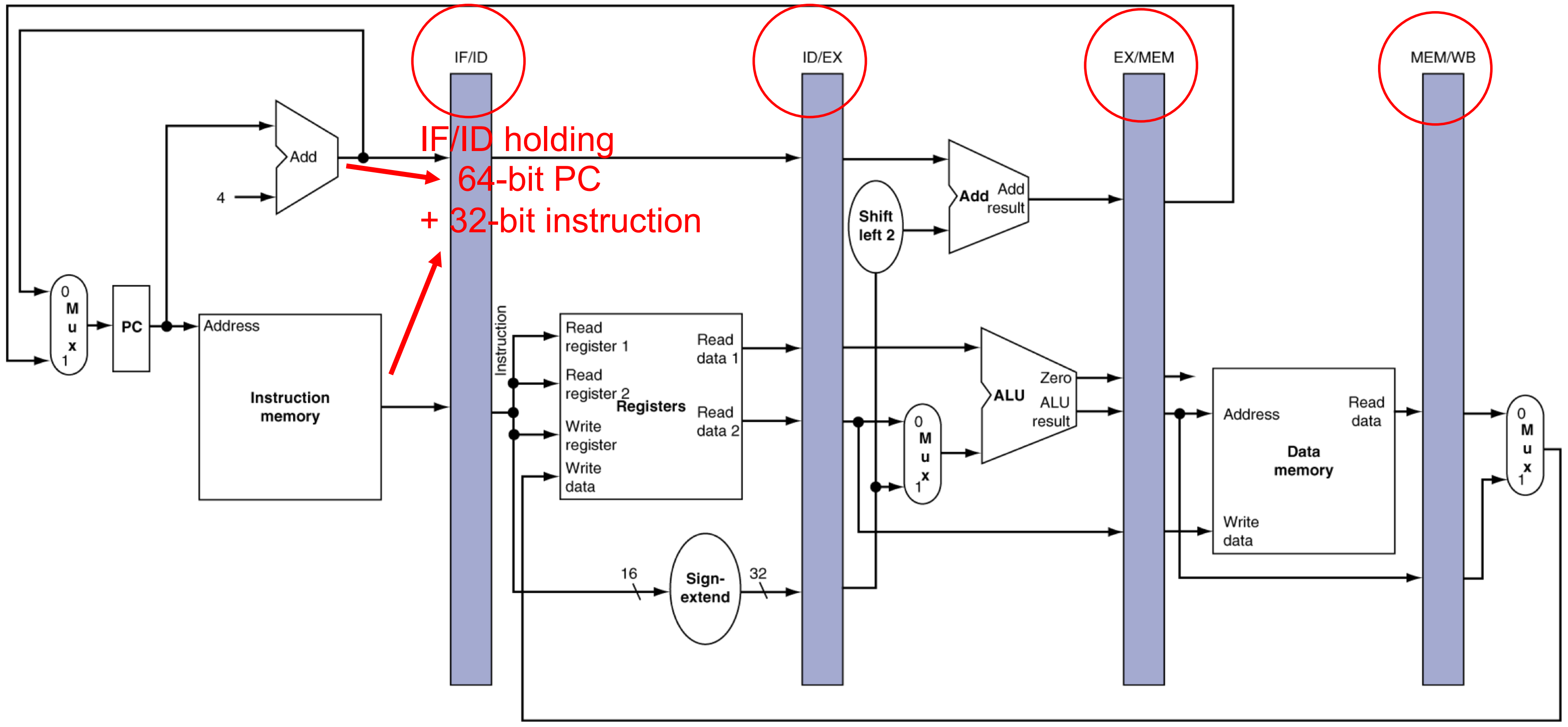
Right-to-left flow leads to hazards

WB

ADD X4, X5, X6

Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle



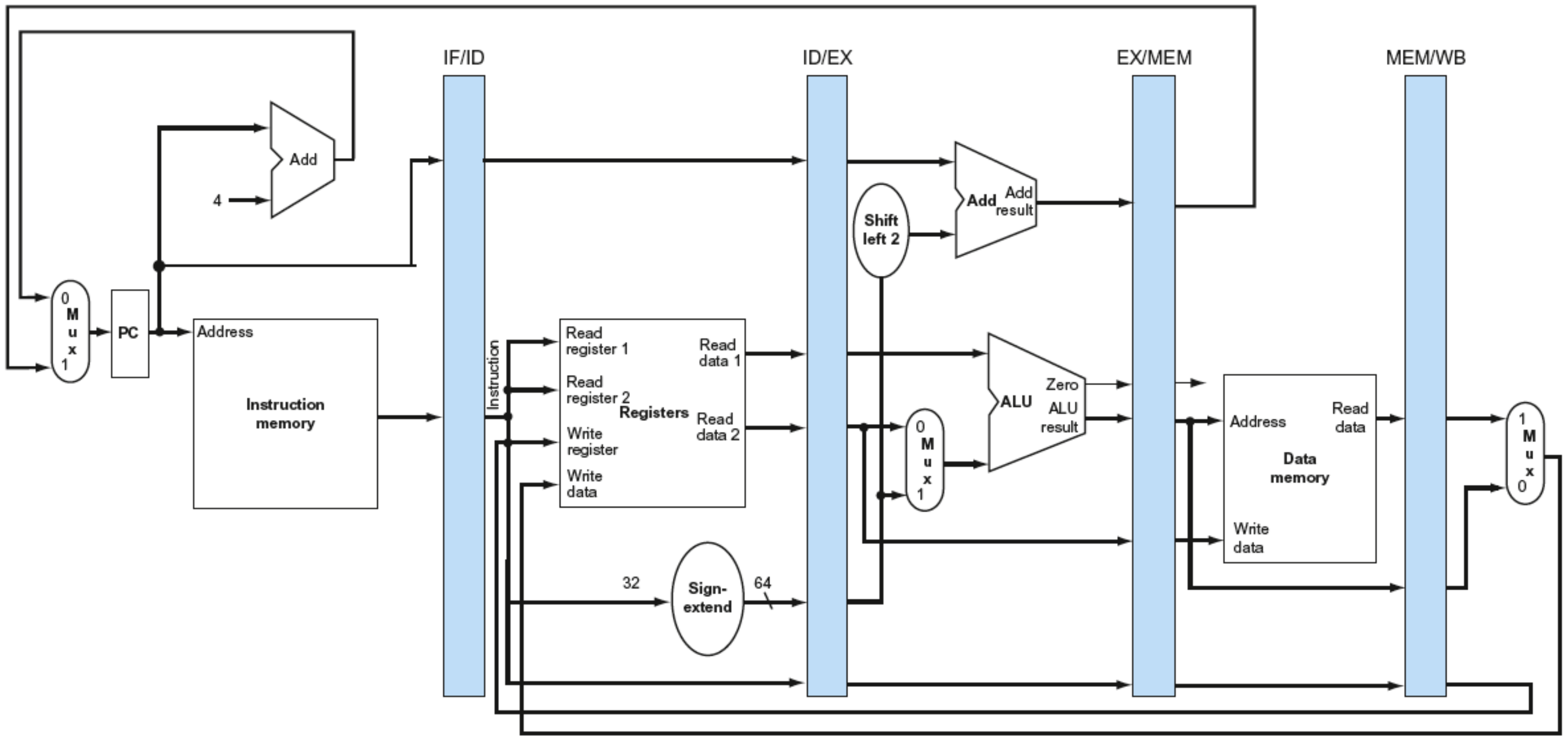
Pipeline Operation

- Cycle-by-cycle flow of instructions through the pipelined datapath
  - “Single-clock-cycle” pipeline diagram
    - Shows pipeline usage (snapshot) in a single cycle
    - Highlight resources used
  - c.f. “multi-clock-cycle” diagram
    - Graph of operation/instructions over time

Single-Cycle Pipeline Diagram

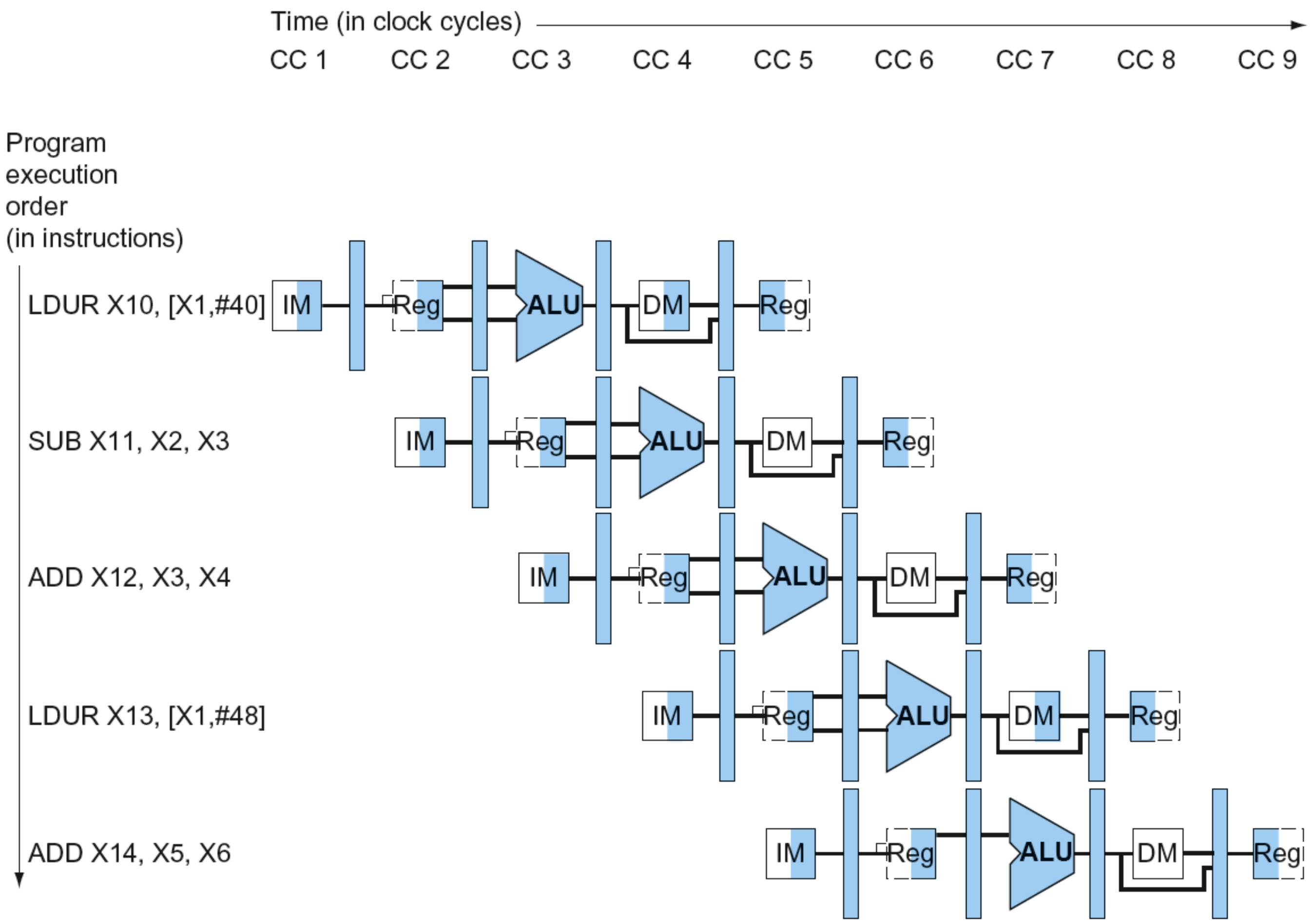
- State of pipeline in a given cycle

ADD X14, X5, X6	LDUR X13, [X1,48]	ADD X12, X3, X4	SUB X11, X2, X3	LDUR X10, [X1,40]
Instruction fetch	Instruction decode	Execution	Memory	Write-back



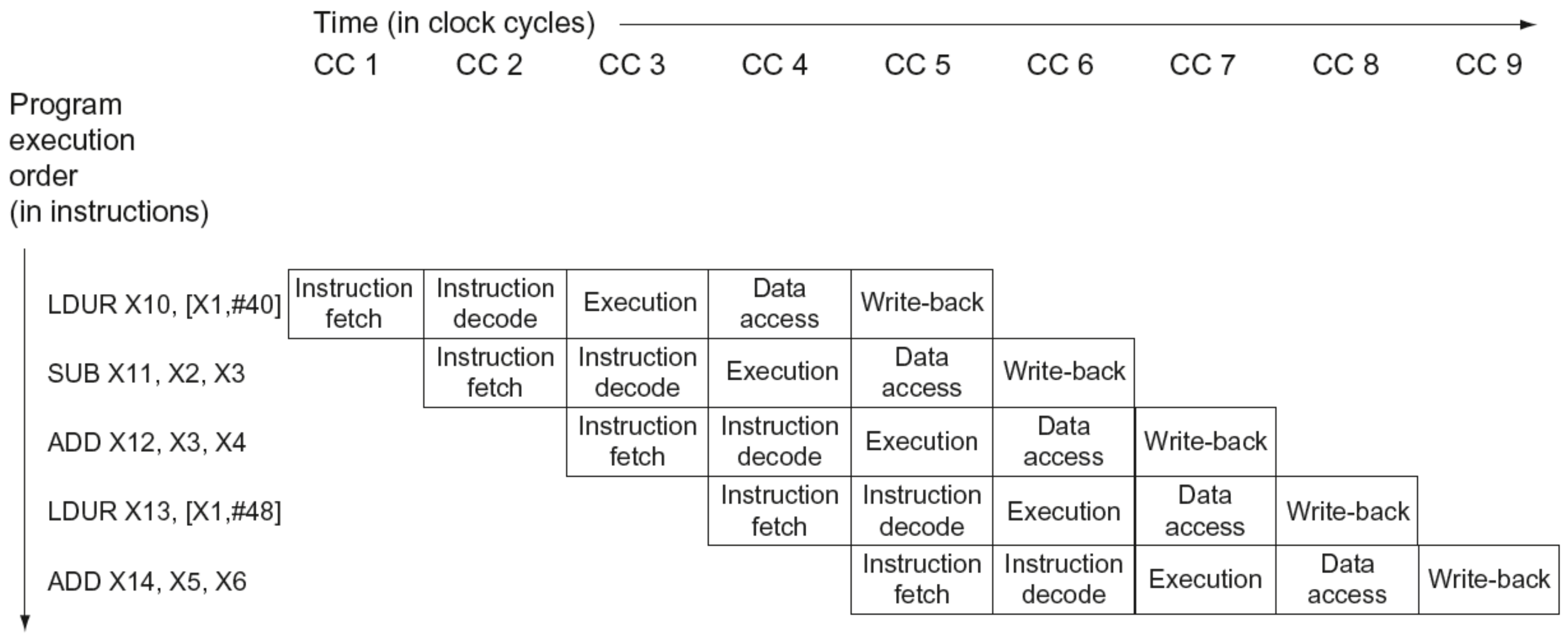
Multi-Cycle Pipeline Diagram

- Form showing resource usage



Multi-Cycle Pipeline Diagram

- Traditional form

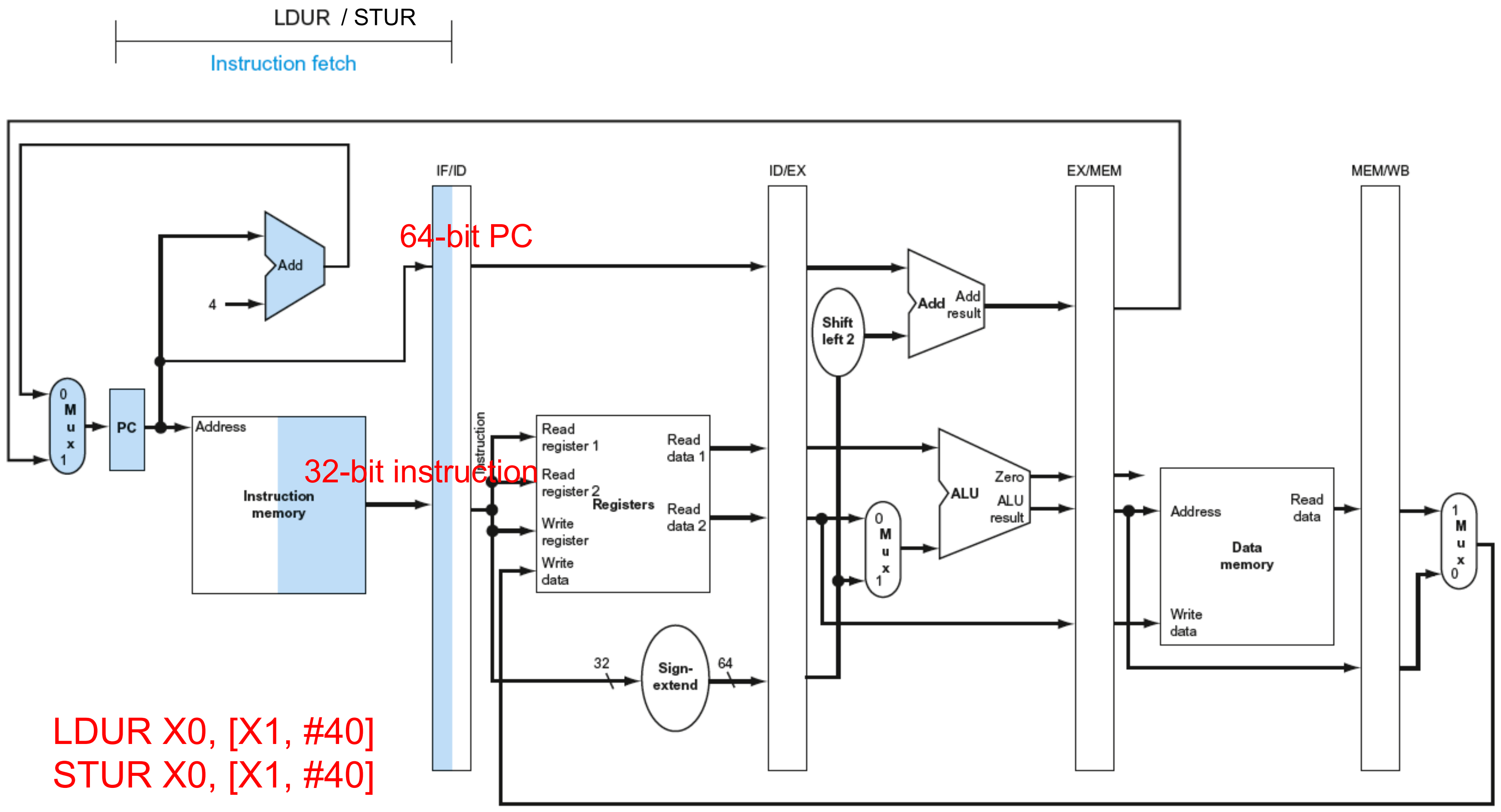


Time (in clock cycles) →

Program execution order (in instructions) ↓

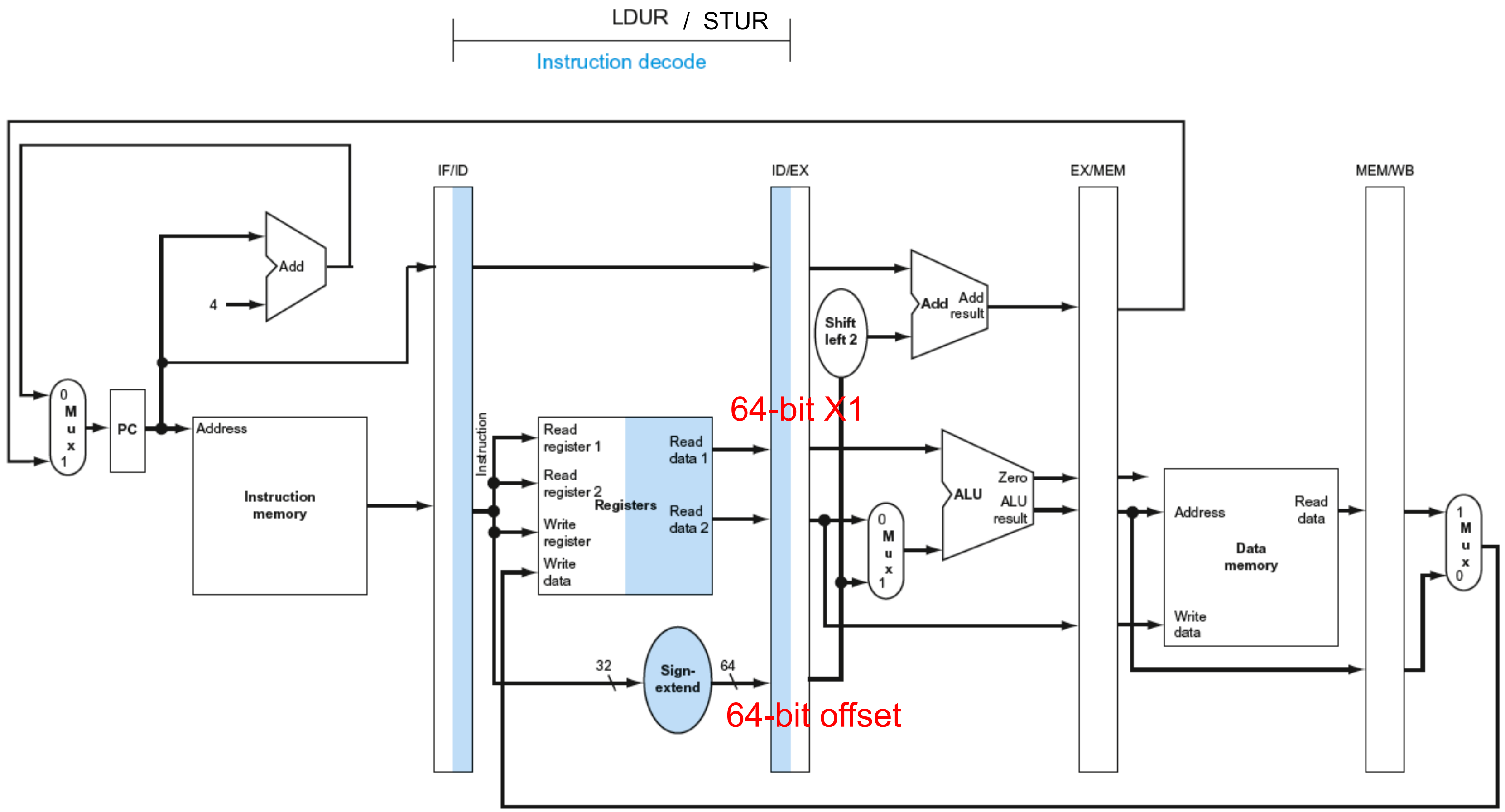
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
LDUR X10, [X1,#40]	Instruction fetch	Instruction decode	Execution	Data access	Write-back				
SUB X11, X2, X3		Instruction fetch	Instruction decode	Execution	Data access	Write-back			
ADD X12, X3, X4			Instruction fetch	Instruction decode	Execution	Data access	Write-back		
LDUR X13, [X1,#48]				Instruction fetch	Instruction decode	Execution	Data access	Write-back	
ADD X14, X5, X6					Instruction fetch	Instruction decode	Execution	Data access	Write-back

IF for Load, Store, …



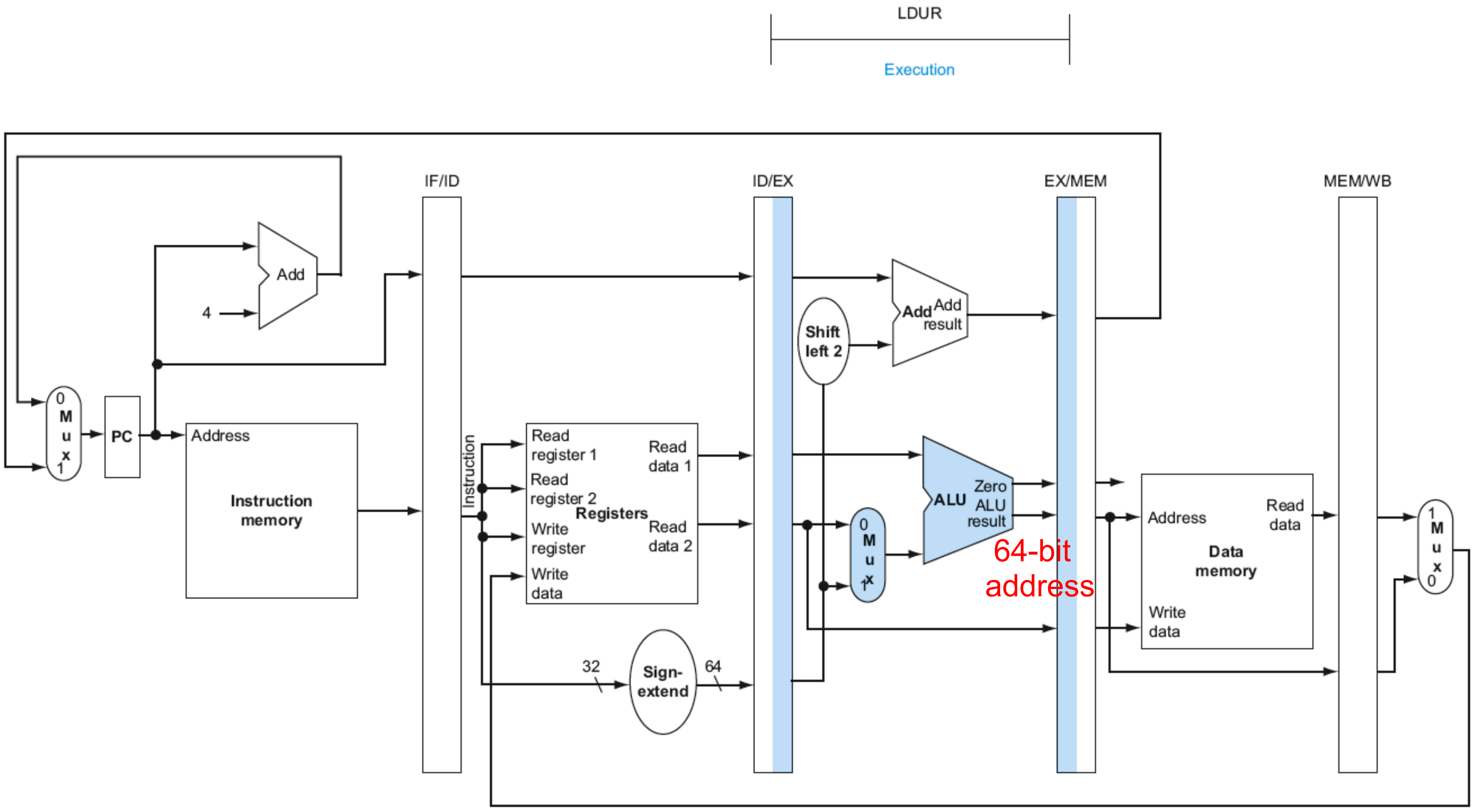
LDUR X0, [X1, #40]
STUR X0, [X1, #40]

ID for Load, Store, …



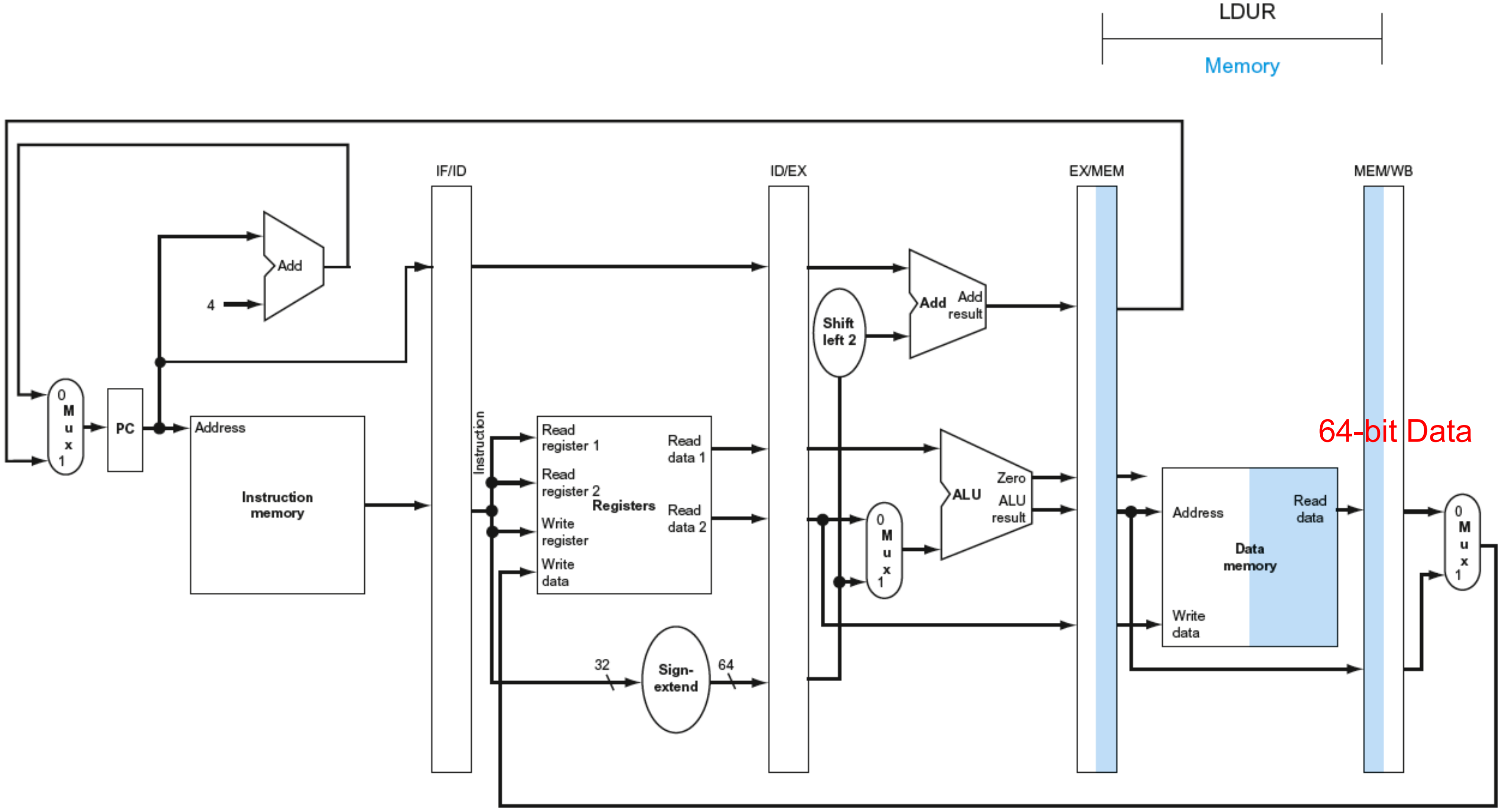
LDUR X0, **[X1, #40]**
STUR X0, **[X1, #40]**

EX for Load



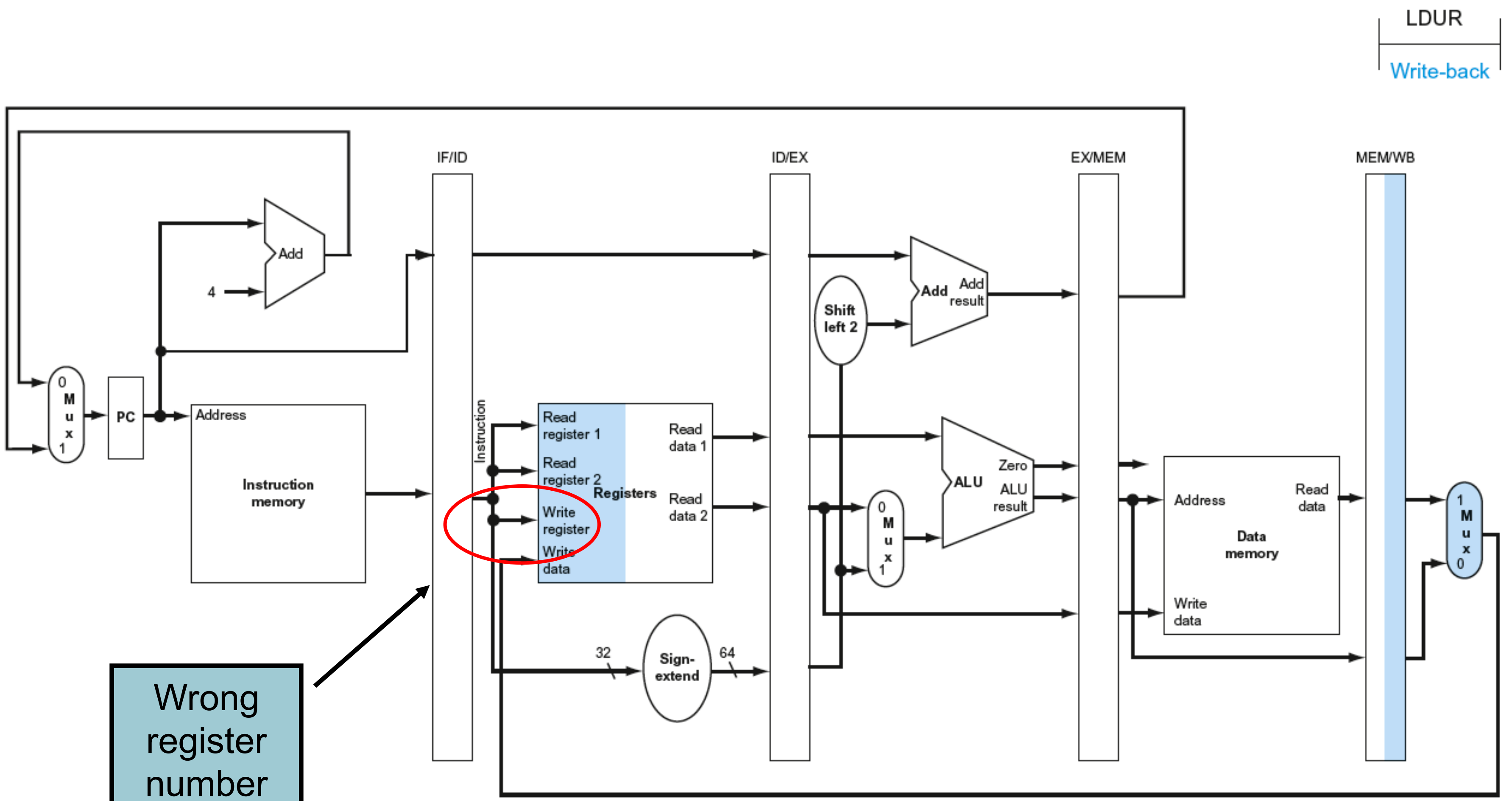
LDUR X0, **[X1, #40]**

MEM for Load



LDUR X0, [X1, #40]

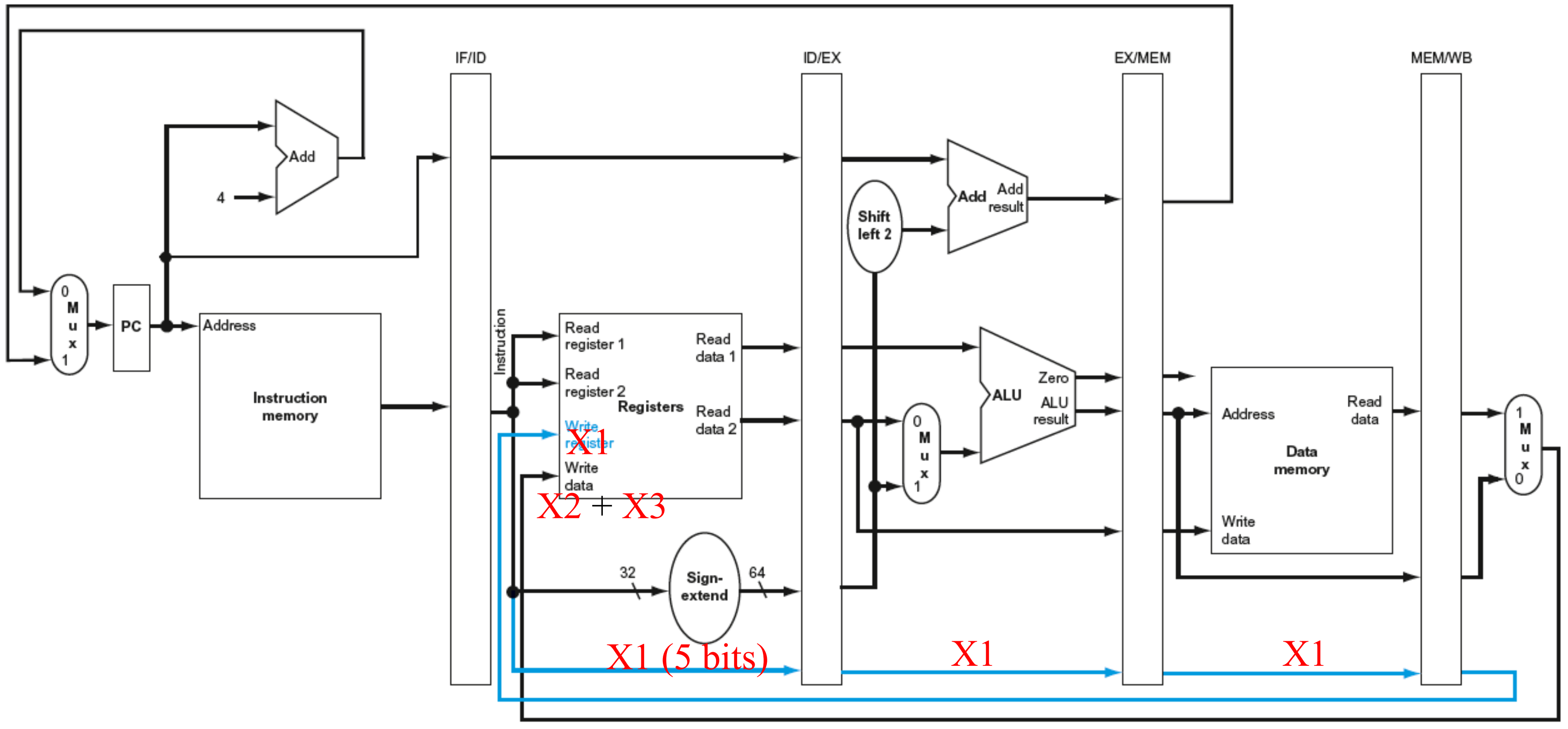
WB for Load



Wrong register number

LDUR X0, [X1, #40]

Corrected Datapath for Load



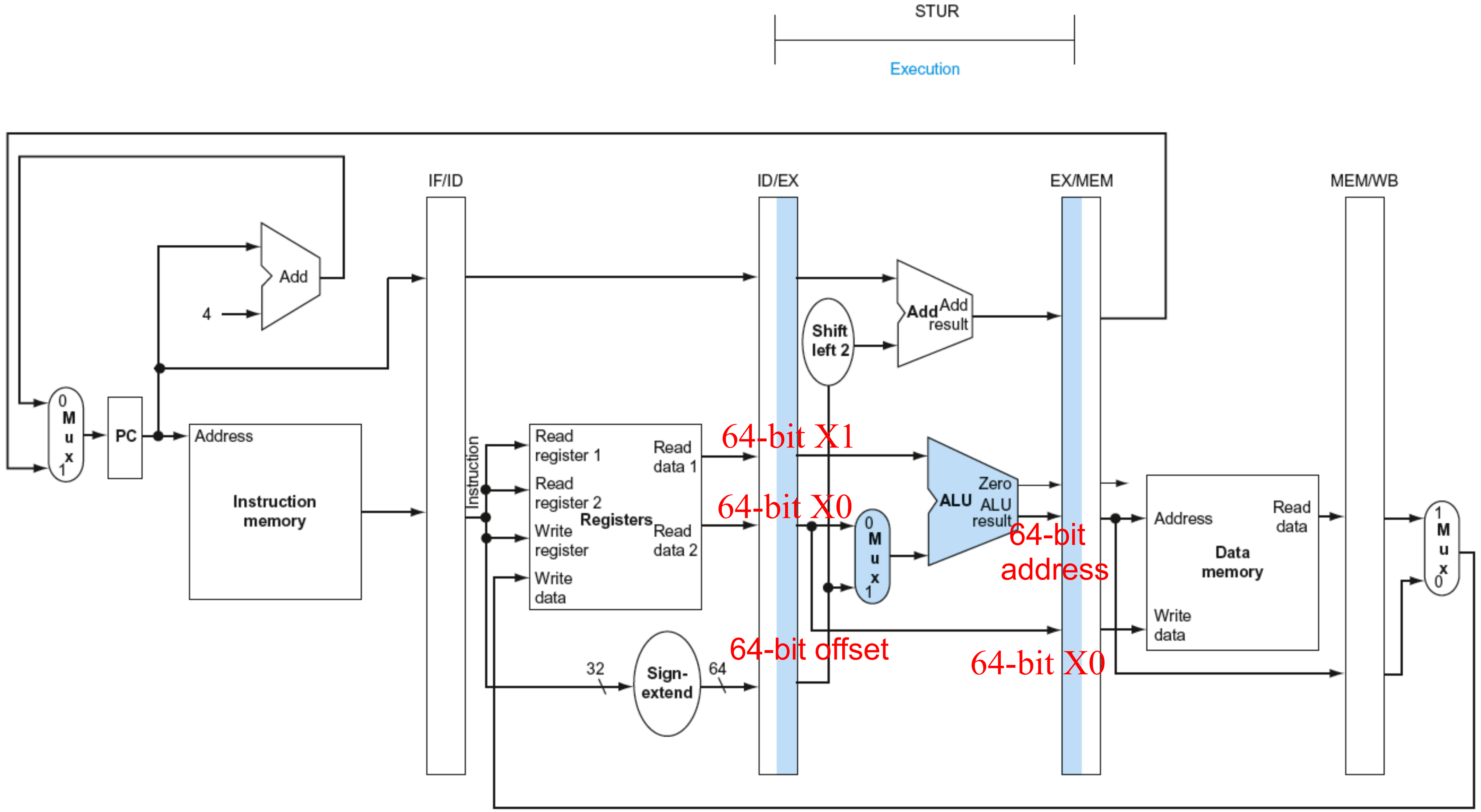
ADD X1, X2, X3

ADD X2 + X3

X2 + X3

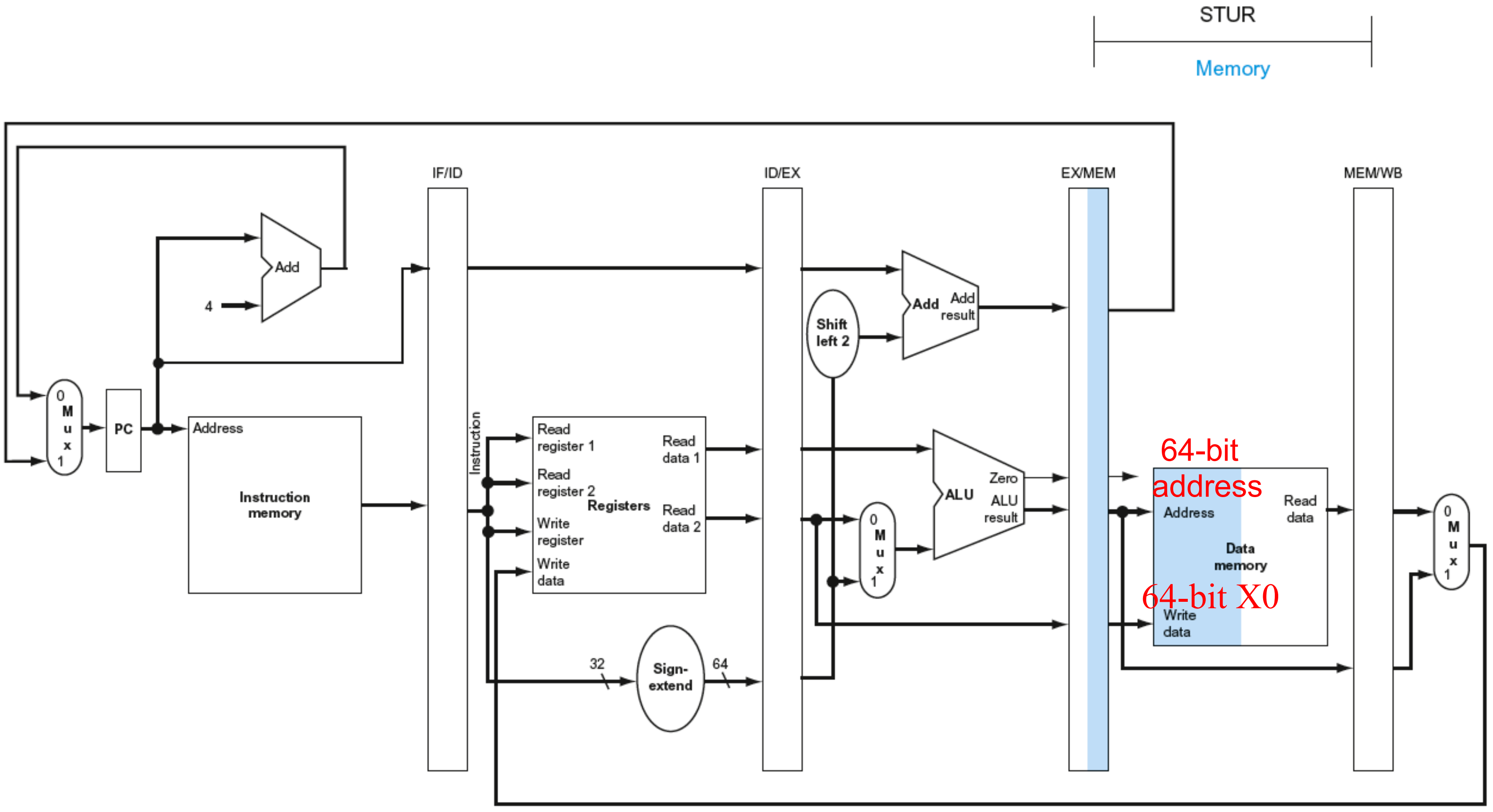
ADD X4, X5, X6

EX for Store



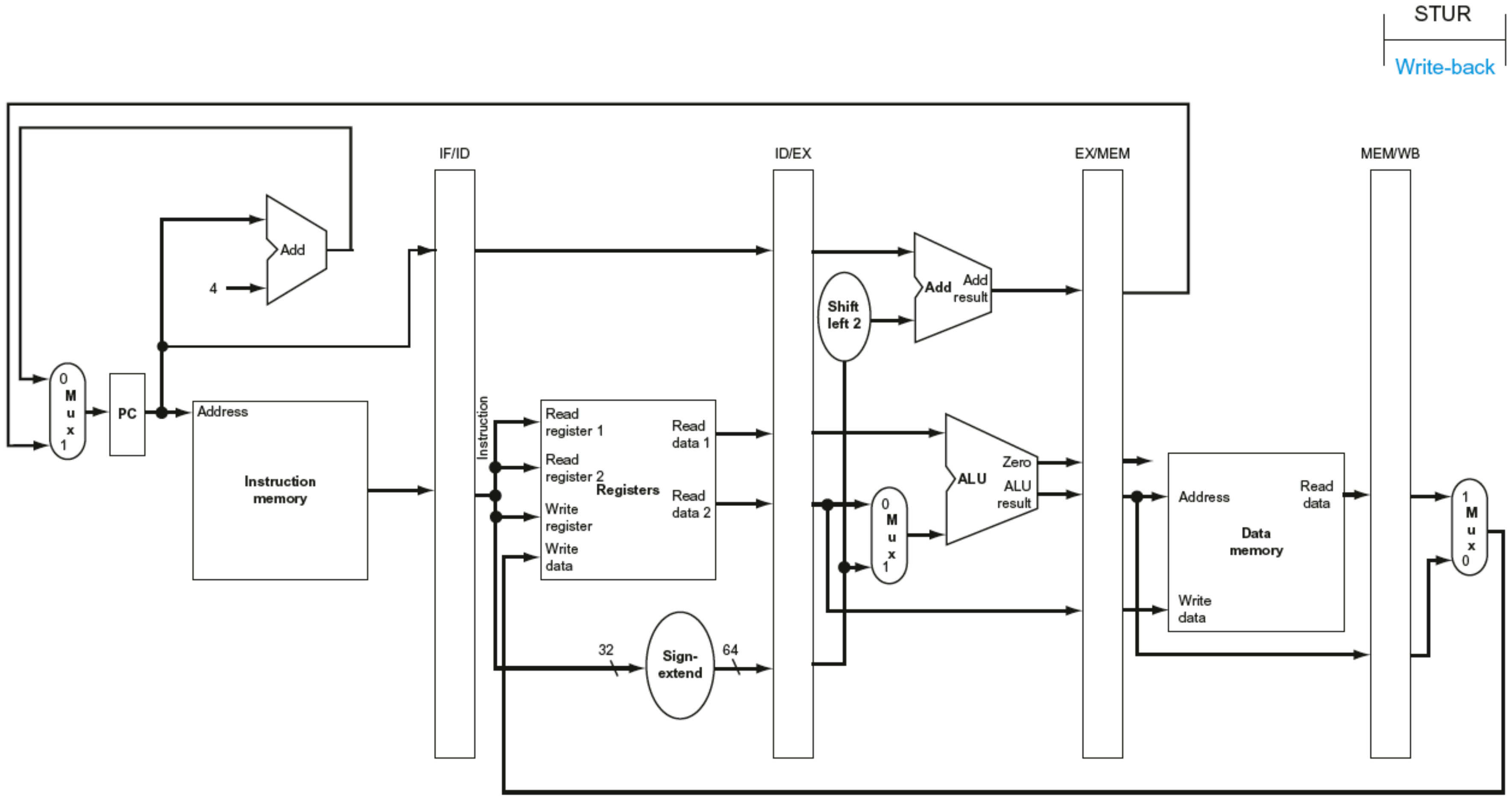
STUR X0, **[X1, #40]**

MEM for Store

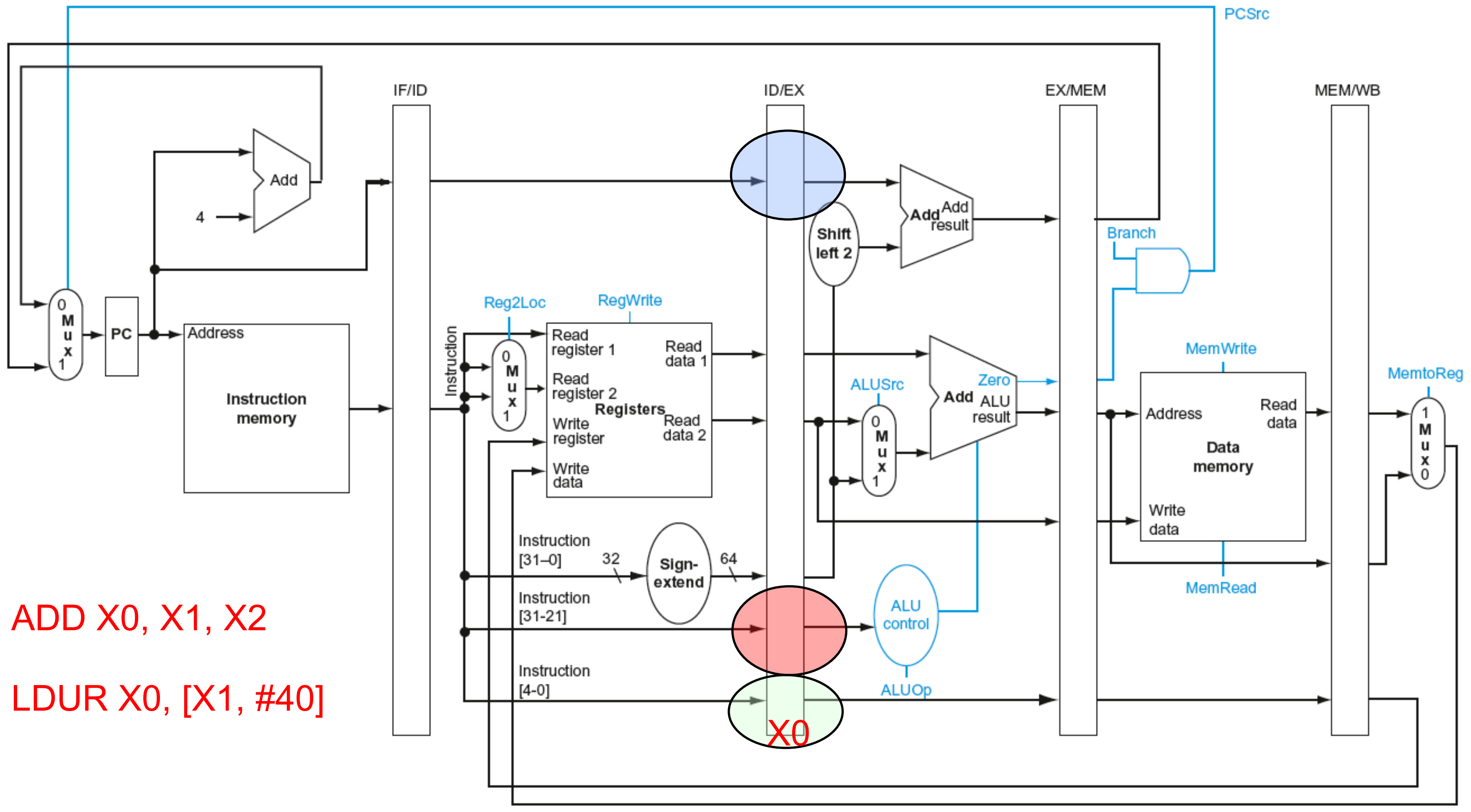


STUR **X0**, [**X1**, **#40**]

WB for Store


STUR
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
32
Sign-extend
64
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Write data
Read data
1
M
u
x
0

Pipelined Control (Simplified)

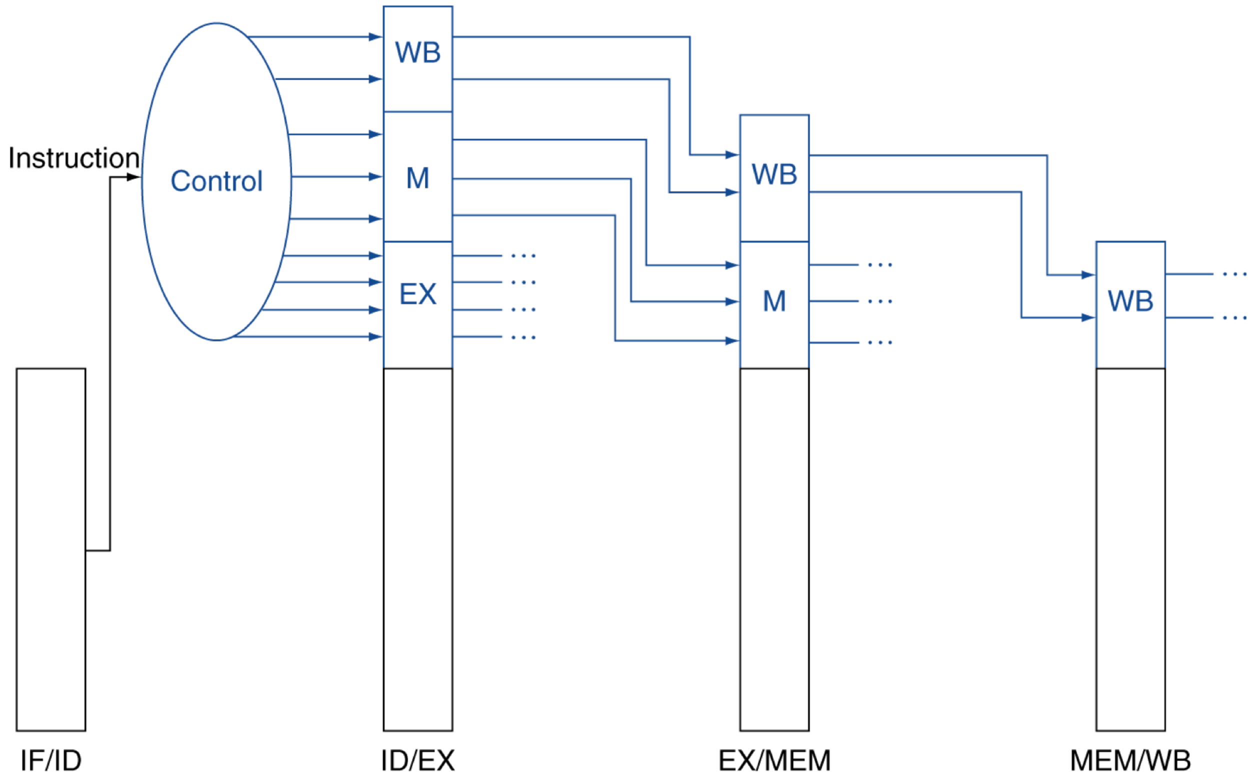


ADD X0, X1, X2

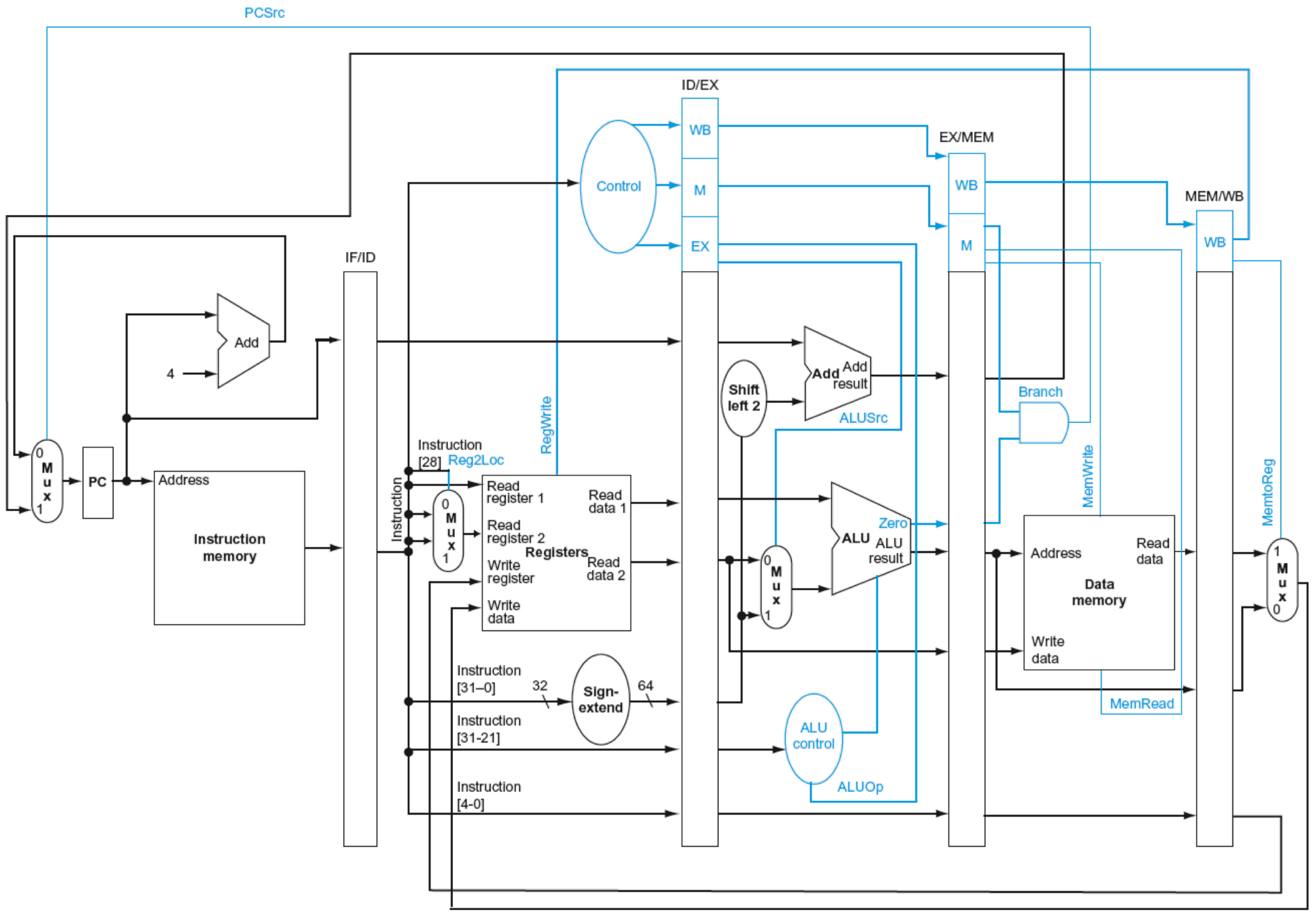
LDUR X0, [X1, #40]

Pipelined Control

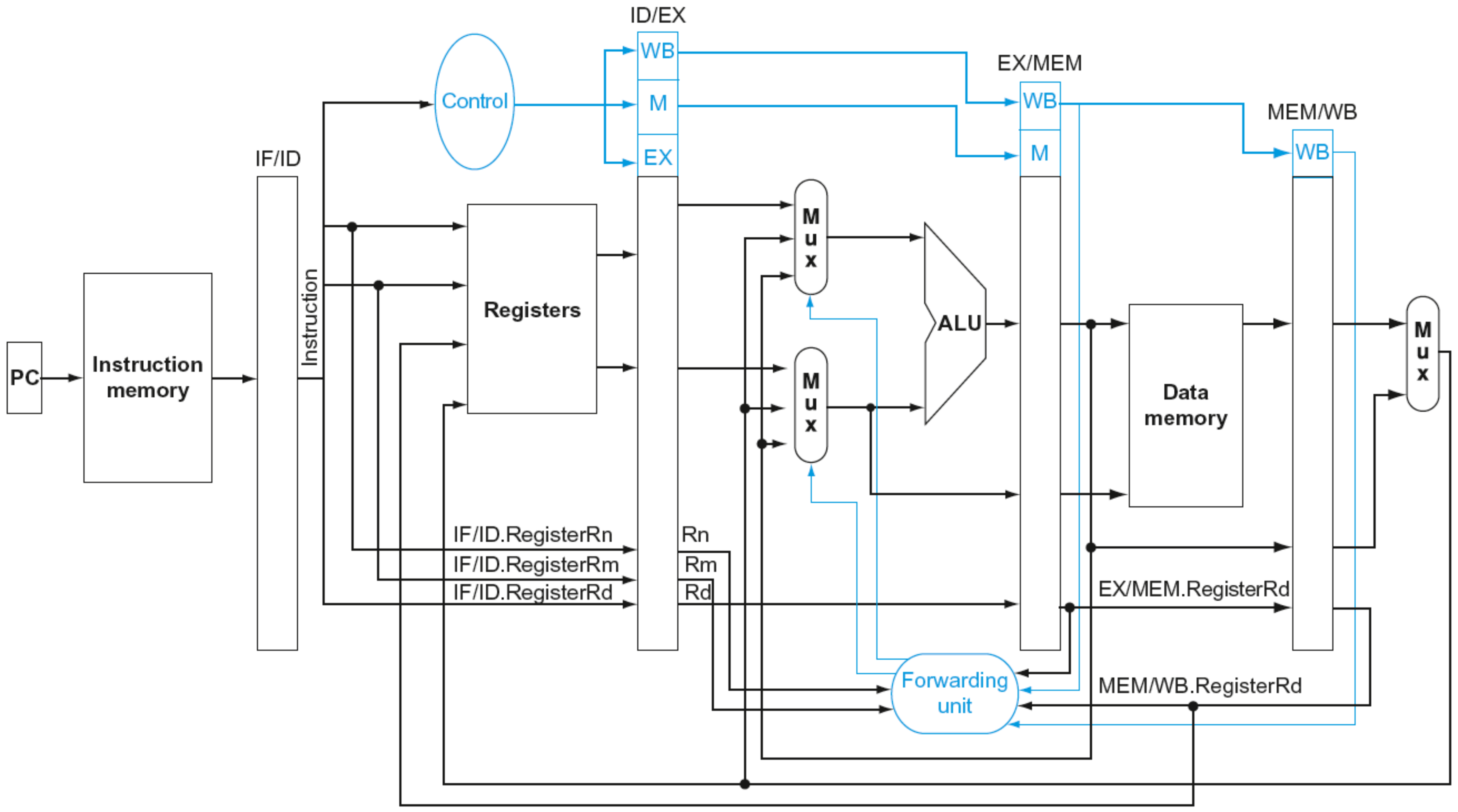
- Control signals derived from instruction
  - As in single-cycle implementation



Pipelined Control



Datapath with Forwarding



Fallacies

- Pipelining is easy (!)
  - The basic idea is easy
  - The devil is in the details
    - e.g., detecting data hazards
- Pipelining is independent of technology
  - So why haven't we always done pipelining?
    - More transistors make more advanced techniques feasible
  - Pipeline-related ISA design needs to take account of technology trends
    - e.g., predicated instructions

Pitfalls

- Poor ISA design can make pipelining harder
  - e.g., complex instruction sets (VAX, IA-32)
    - Significant overhead to make pipelining work
    - IA-32 micro-op approach
  - e.g., complex addressing modes
    - Register update side effects, memory indirection
  - e.g., delayed branches
    - Advanced pipelines have long delay slots

Concluding Remarks

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control