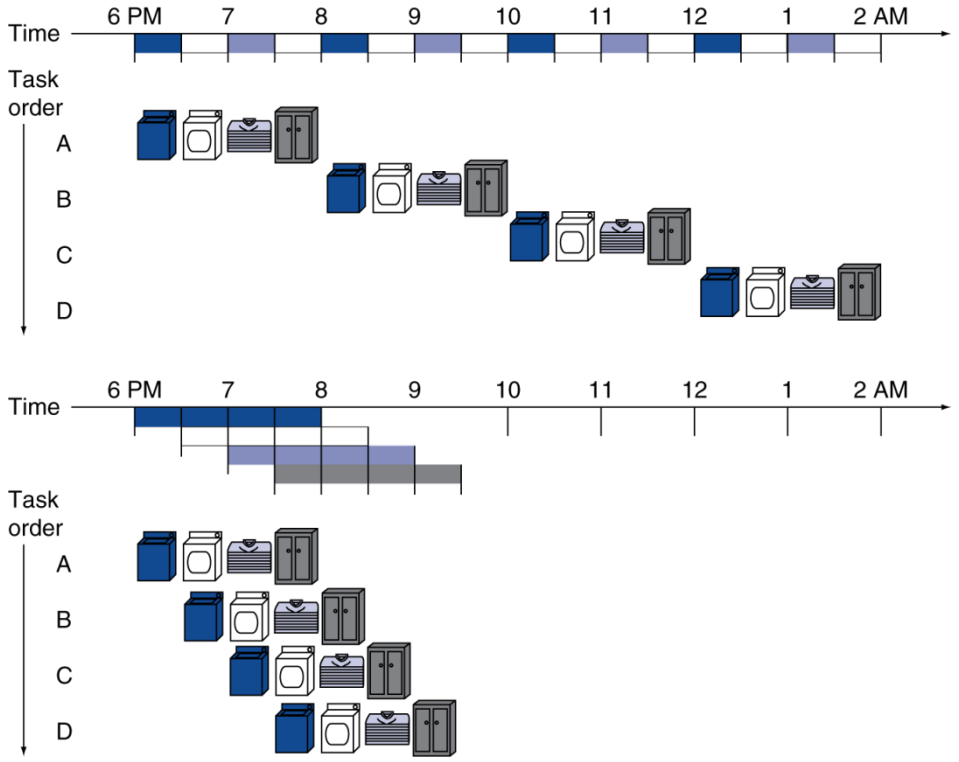
Optimització de processadors (2):
RISC-V pipelined

Remember: **Longest delay**, in a block of the datapath, **determines clock period**

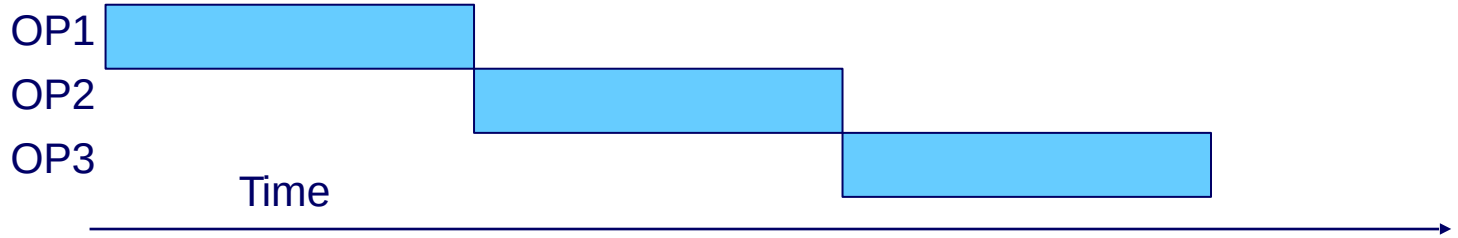
- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:
  - Speedup = 8/3.5 = 2.3
- Non-stop:
  - Speedup = 2n/(0.5n + 1.5) ≈ 4 = number of stages

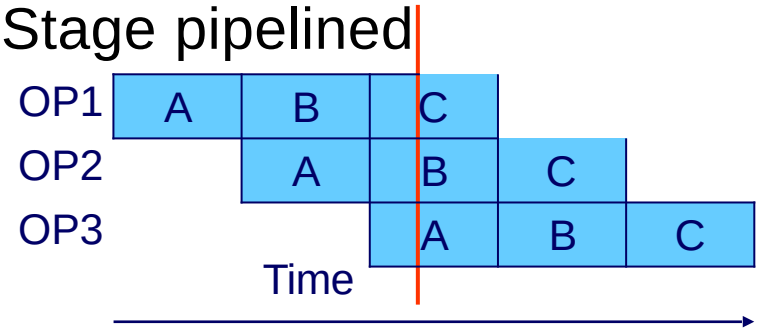
Example: 3-Stage Pipelined Version

Unpipelined



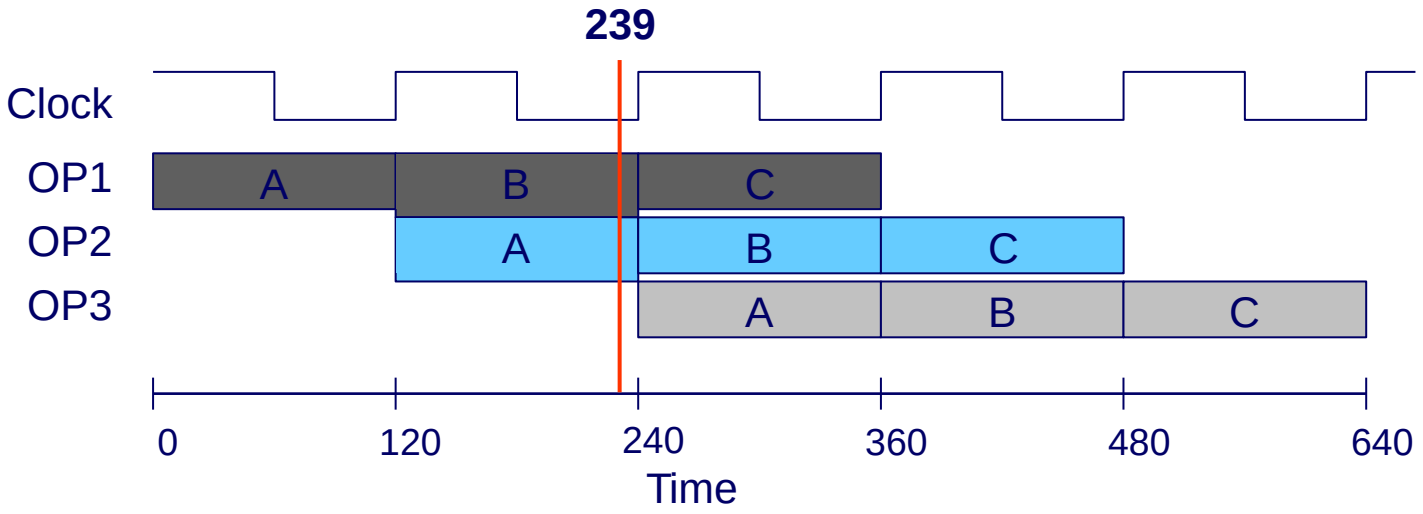
- Cannot start new operation until previous one completes

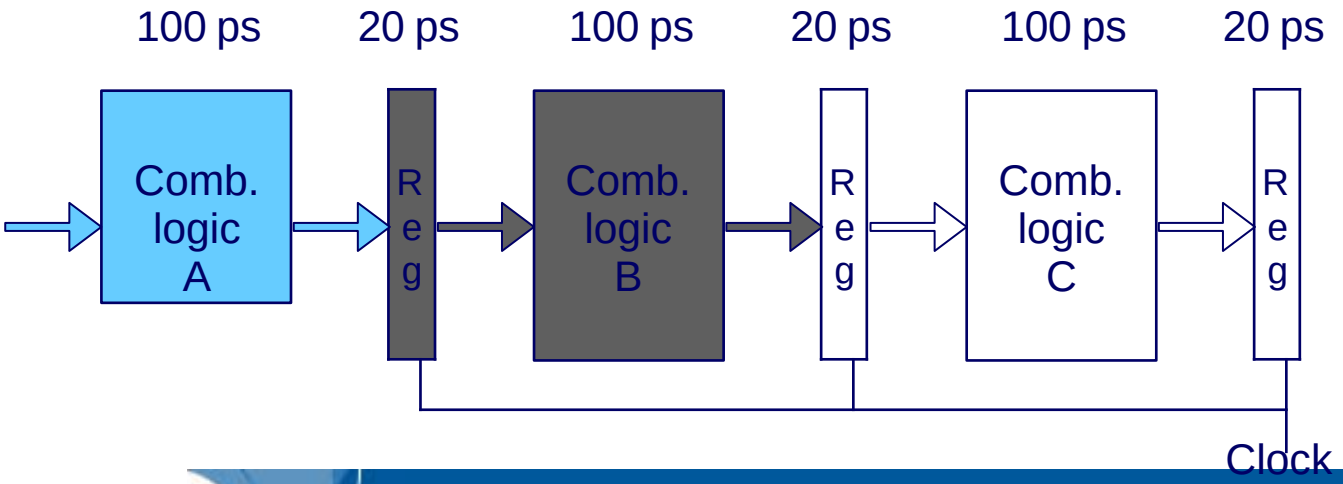
3-Stage pipelined



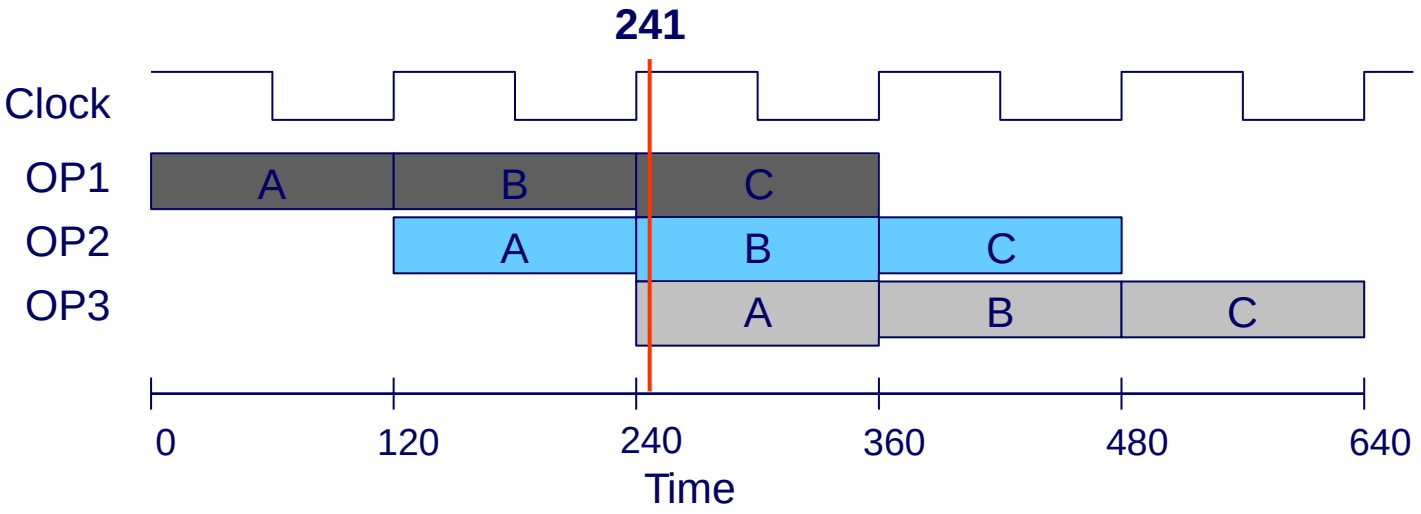
- Up to 3 operations in process simultaneously

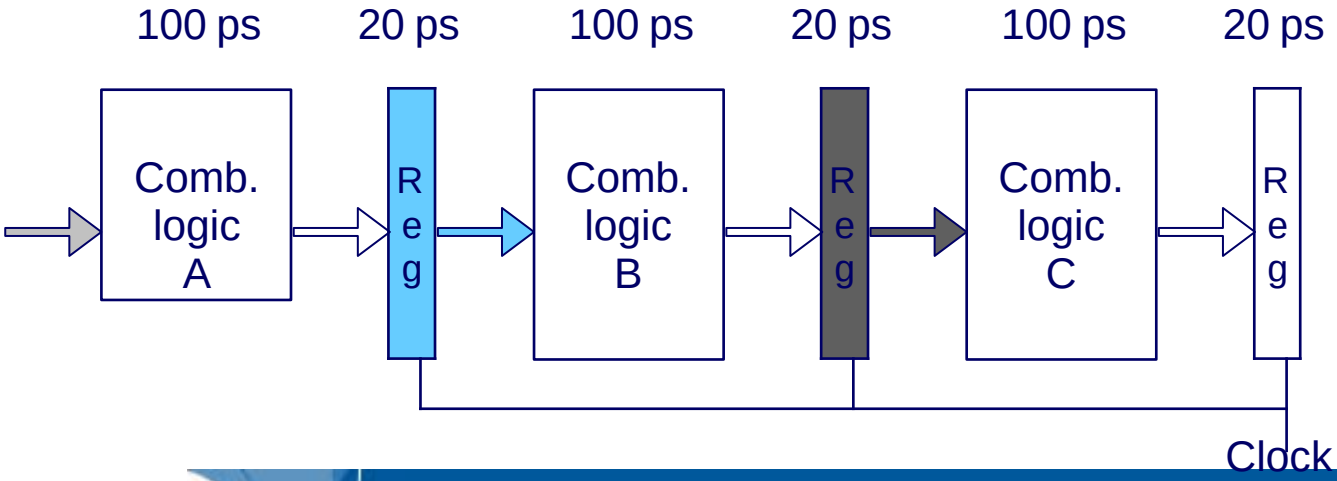
What happens in a clock cycle ?



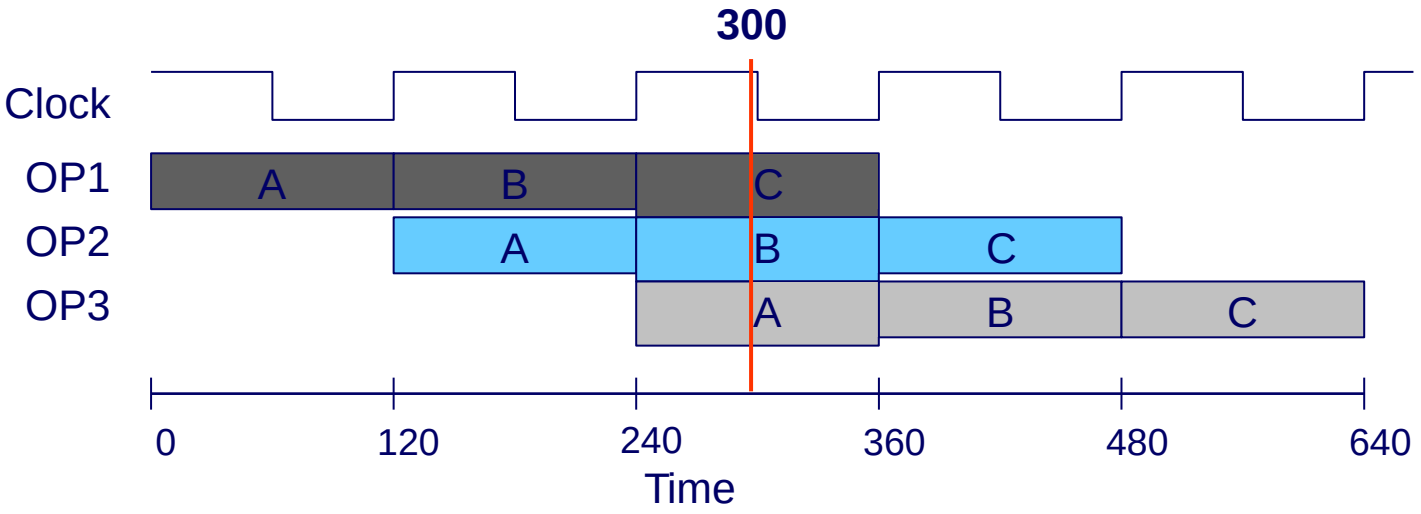


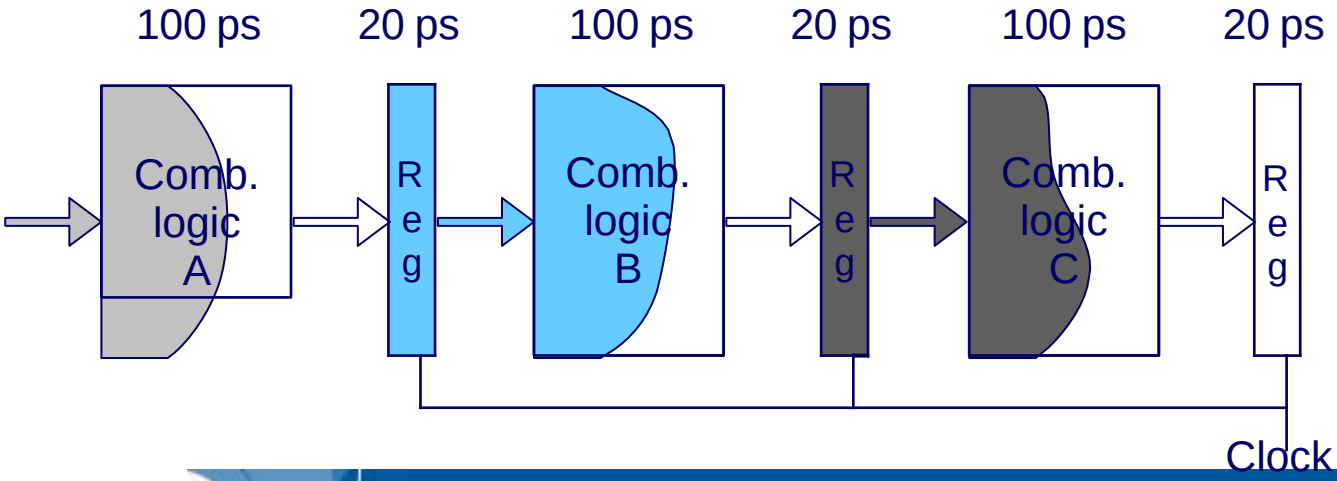
What happens in a clock cycle ?



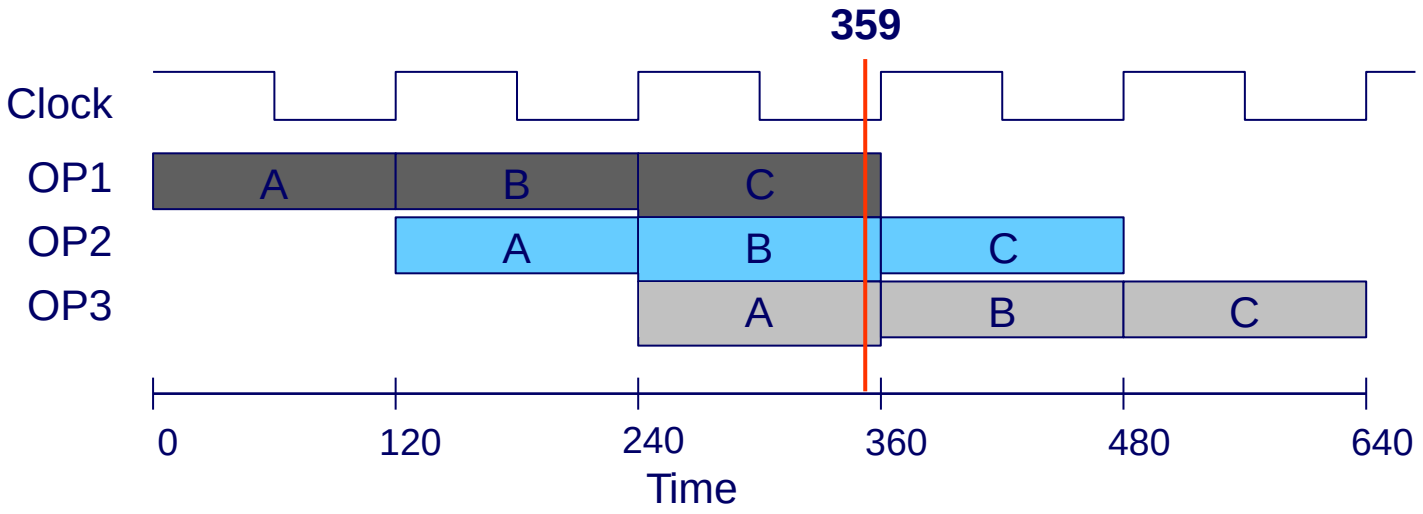


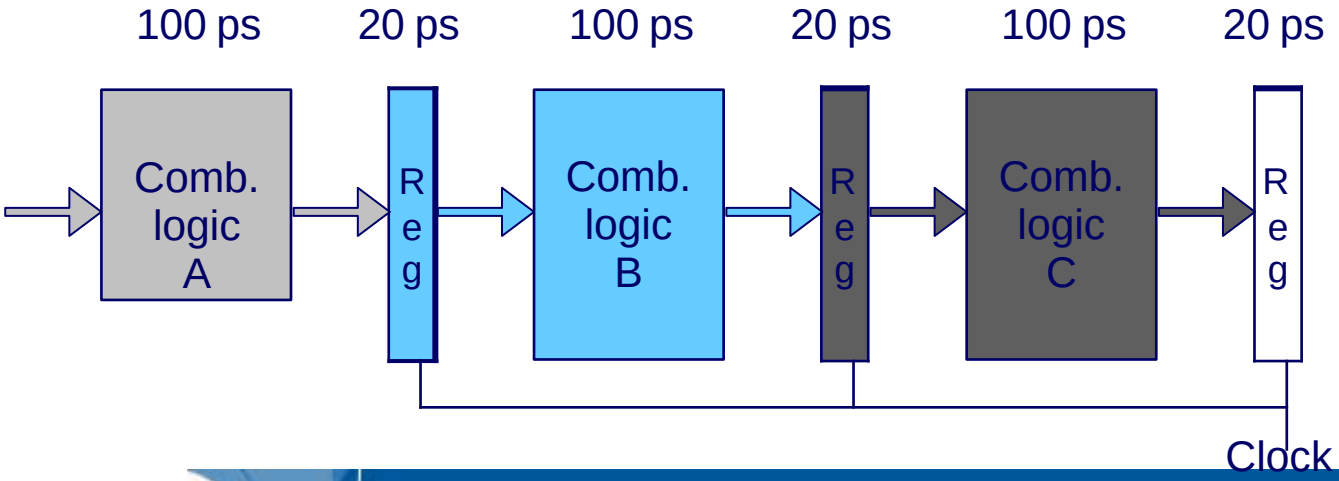
What happens in a clock cycle ?



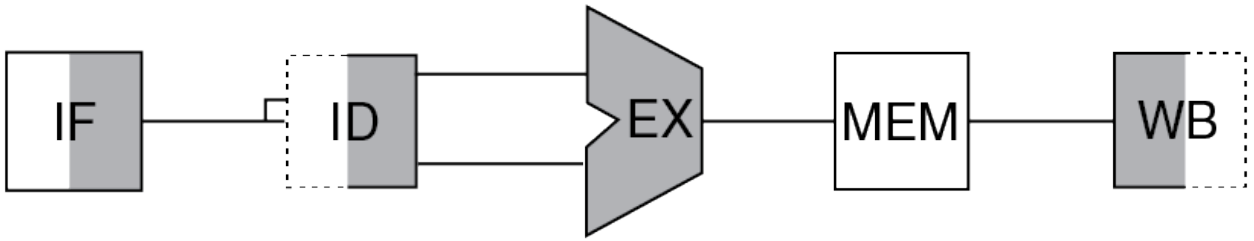


What happens in a clock cycle ?





- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

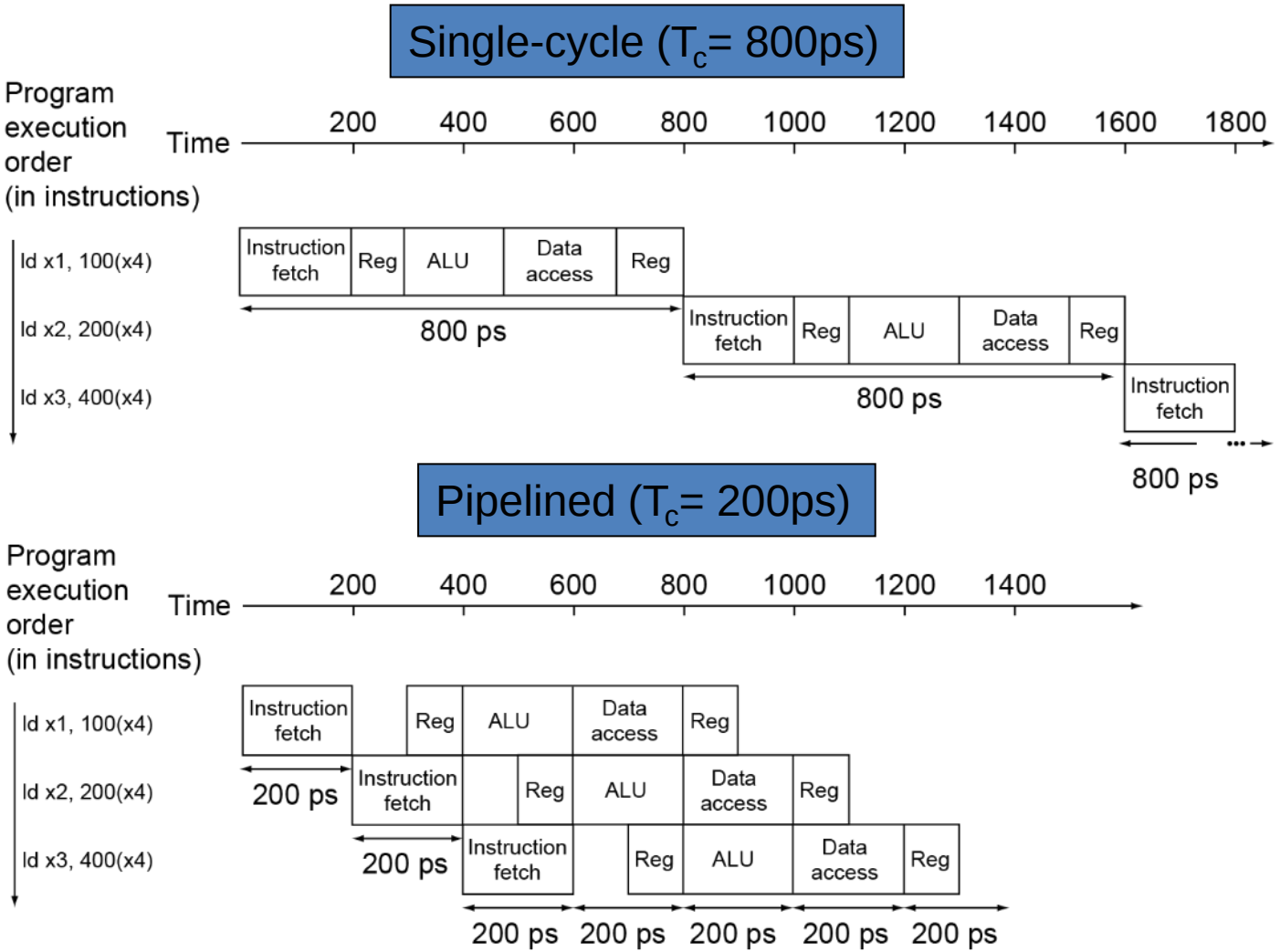


Performance comparison between RISC-V single-cycle and RISC-V 5-stage pipelined

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
ld	200ps	100 ps	200ps	200ps	100 ps	800ps
sd	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

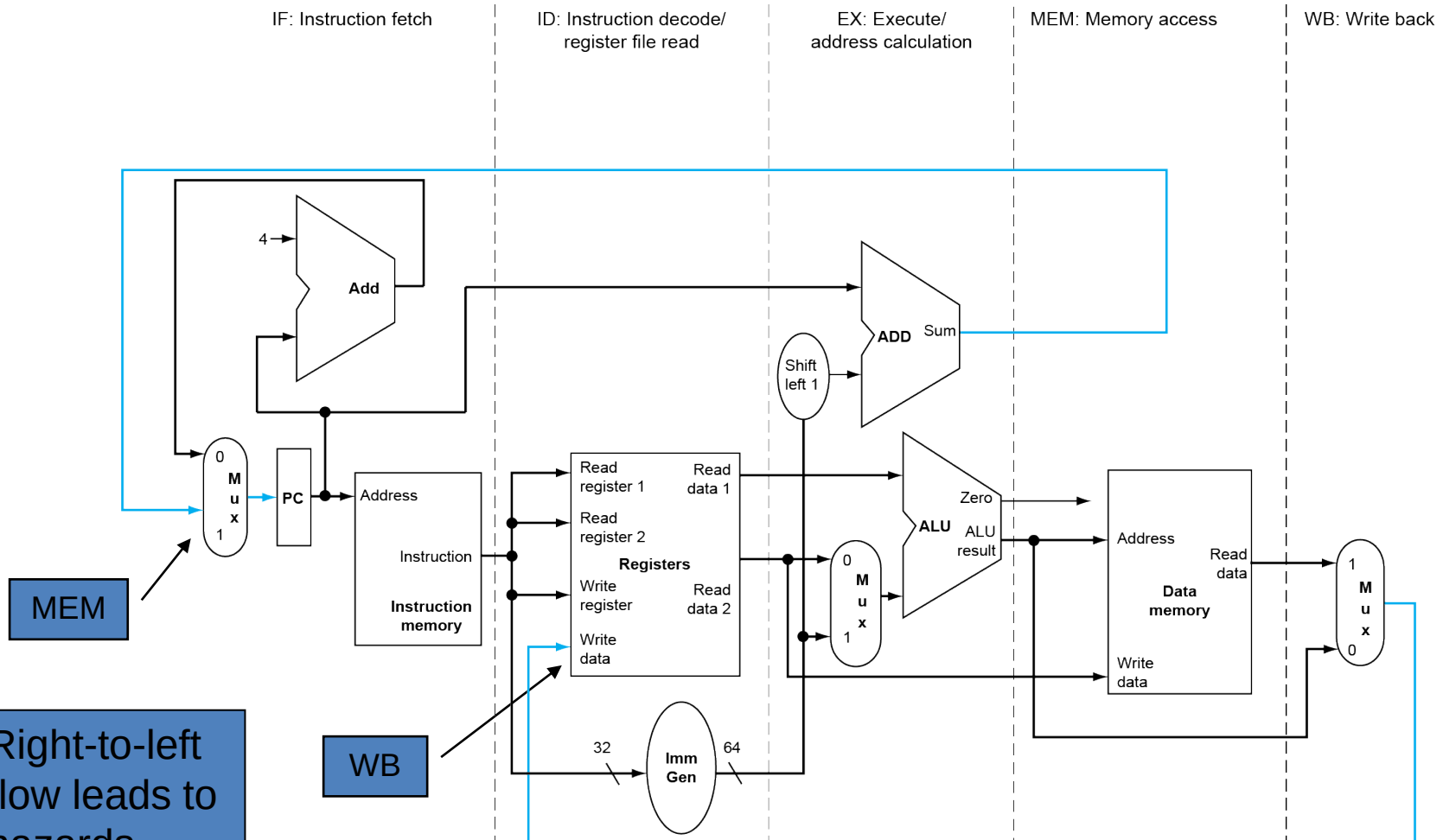
Performance comparison between RISC-V single-cycle and RISC-V 5-stage pipelined



- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions$_{pipelined}$ = $\frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

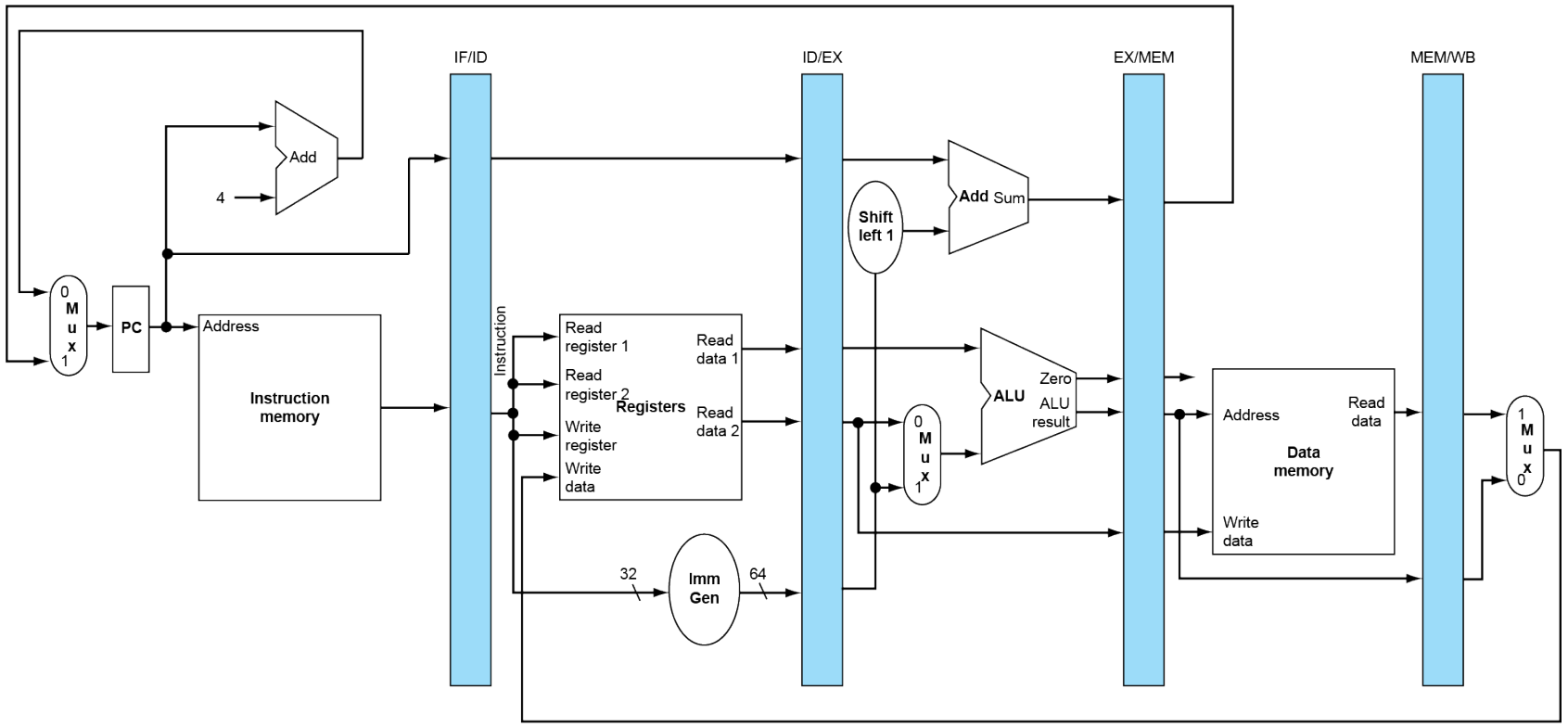
- No dependency between stages => Adding registers between stages
- Memory: **IF** only reads Instruction Mem; **MEM** reads/writes data Mem.
- RISC-V ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage

RISC-V Pipelined Datapath



Pipeline registers

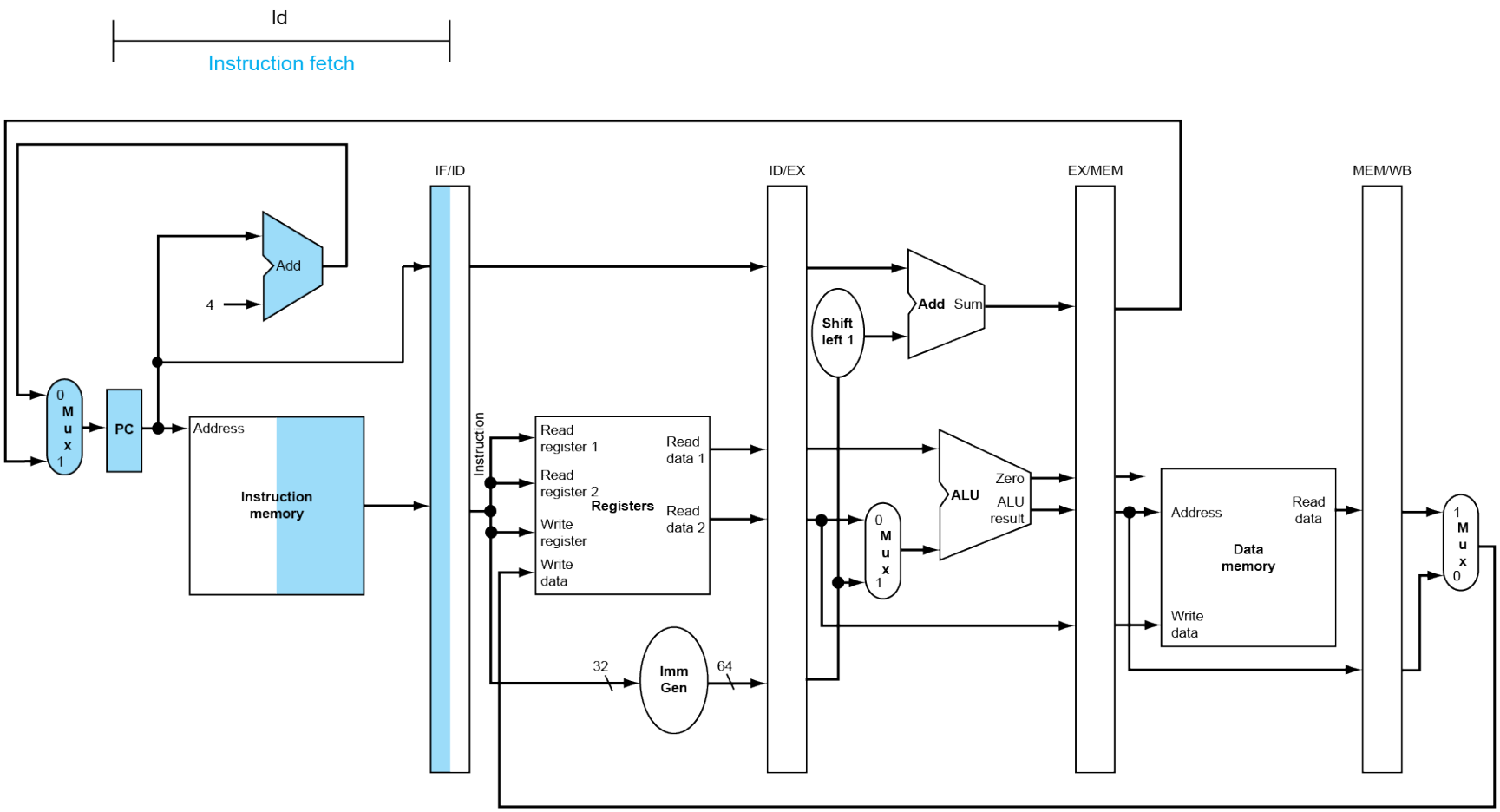
- Need registers between stages
  - To hold information produced in previous cycle



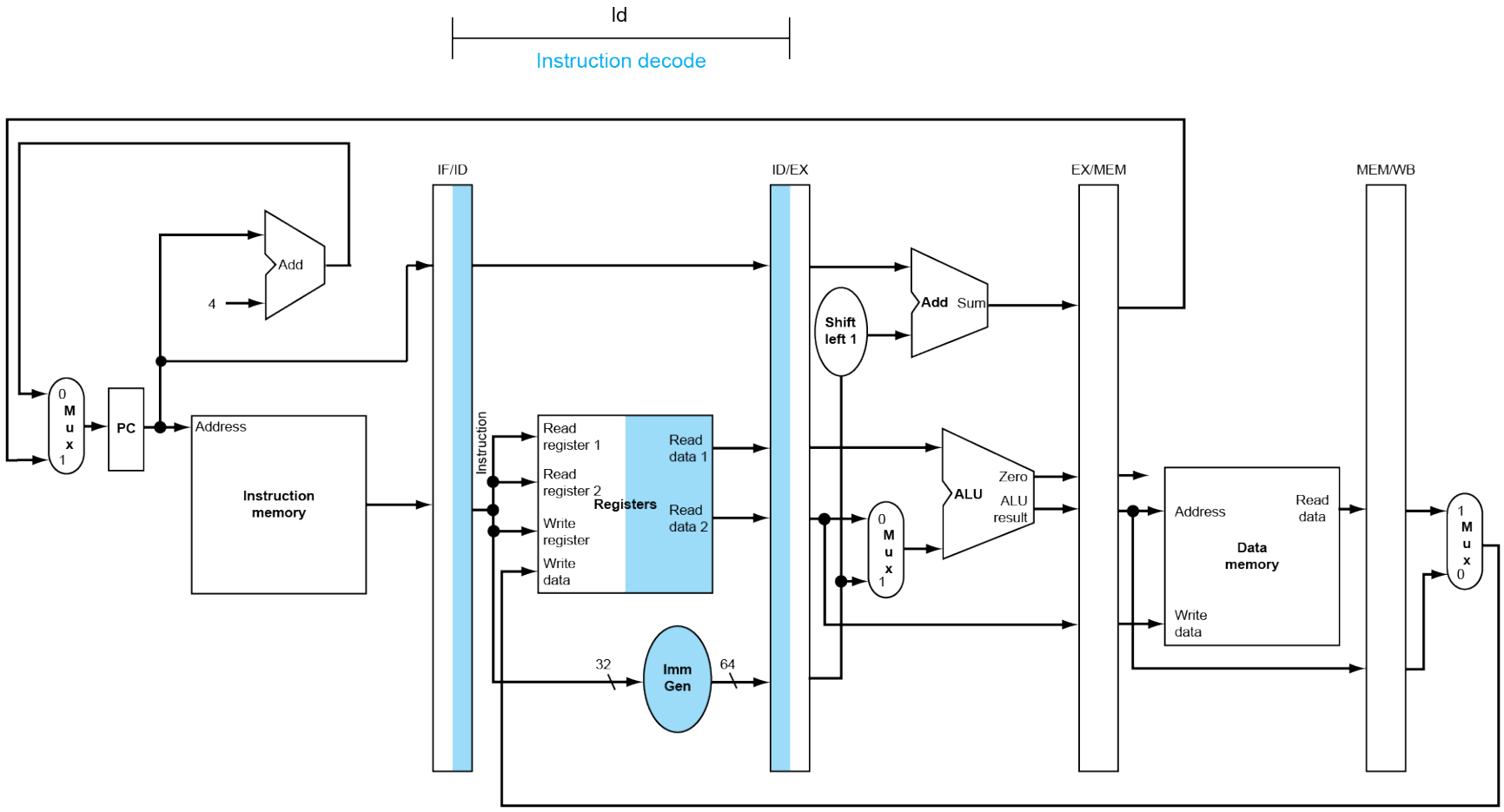
Pipeline Operation

- Cycle-by-cycle flow of instructions through the pipelined datapath
  - "Single-clock-cycle" pipeline diagram
    - Shows pipeline usage in a single cycle
    - Highlight resources used
  - c.f. "multi-clock-cycle" diagram
    - Graph of operation over time
- We'll look at "single-clock-cycle" diagrams for load & store

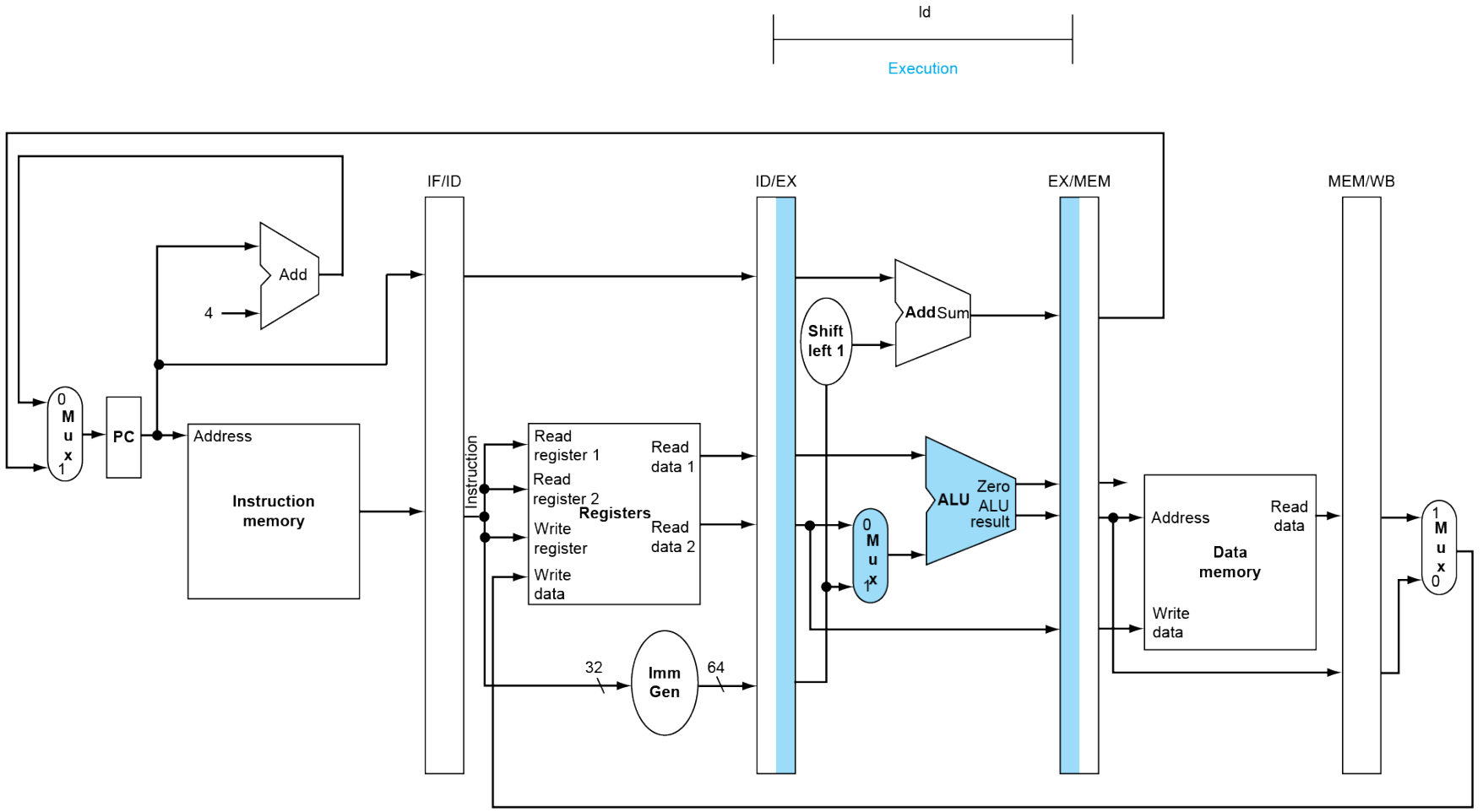
IF for Load, Store, …



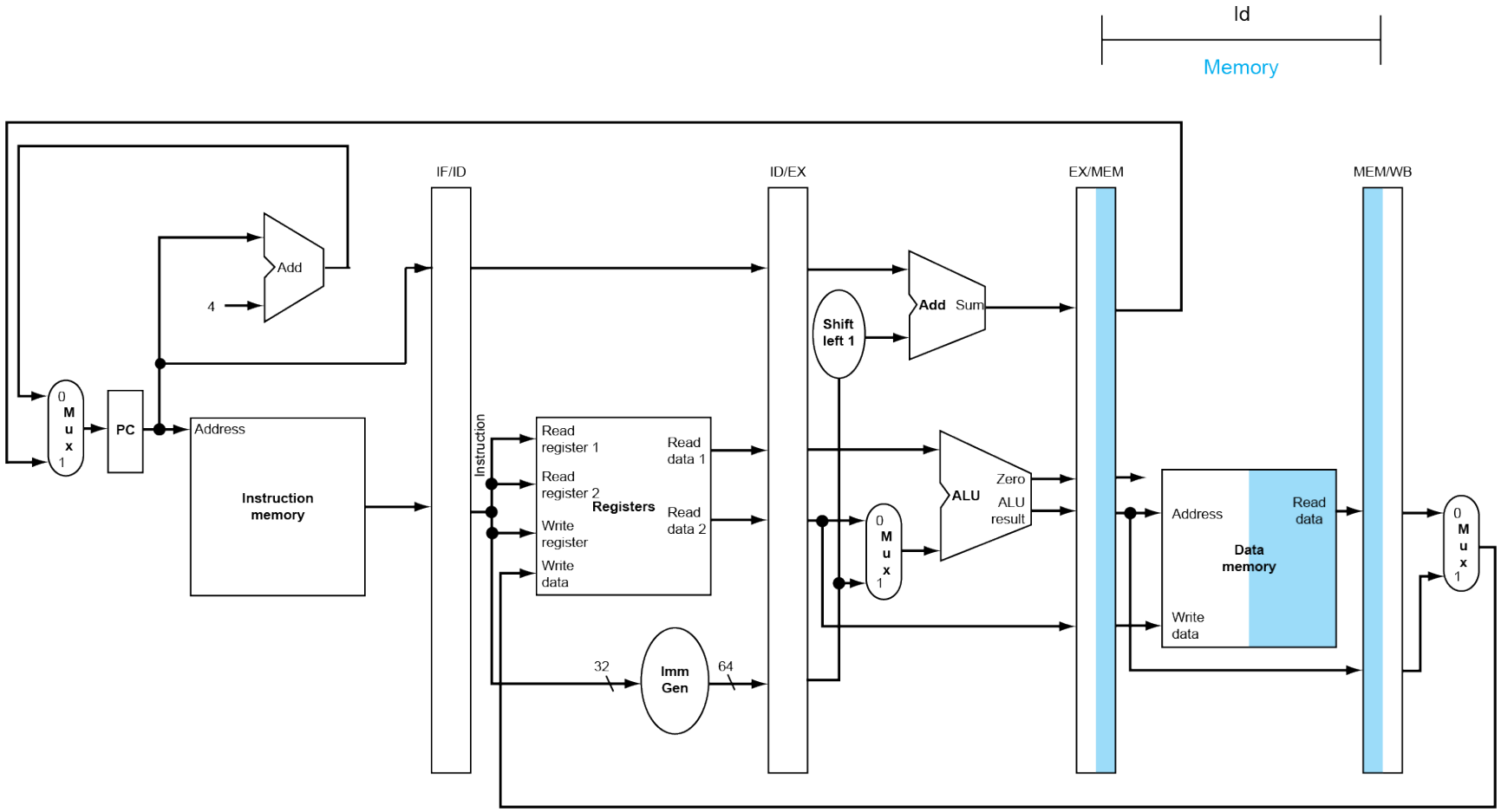
ID for Load, Store, …



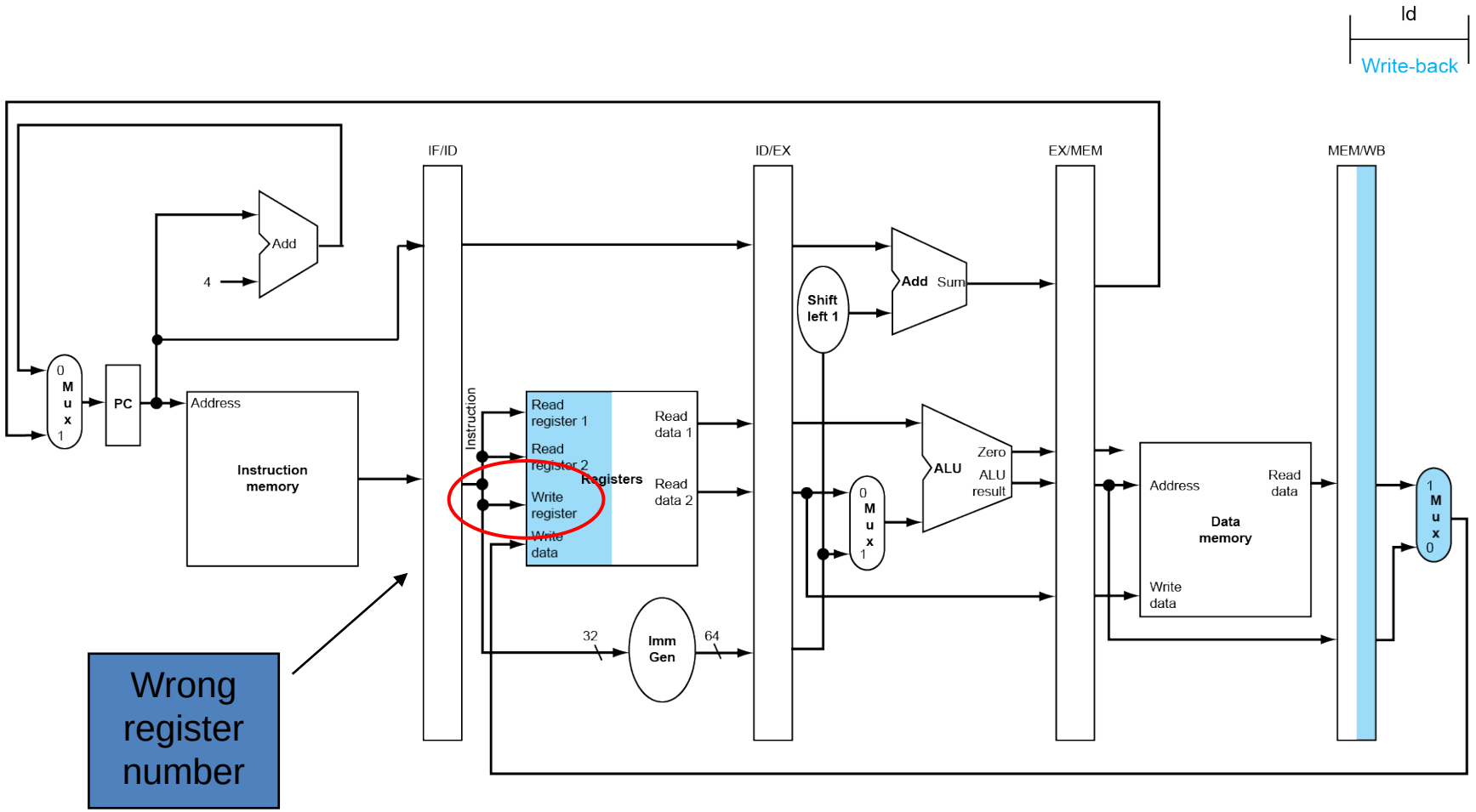
EX for Load


ld
Execution
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
Imm Gen
64
Shift left 1
Add
Sum
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0

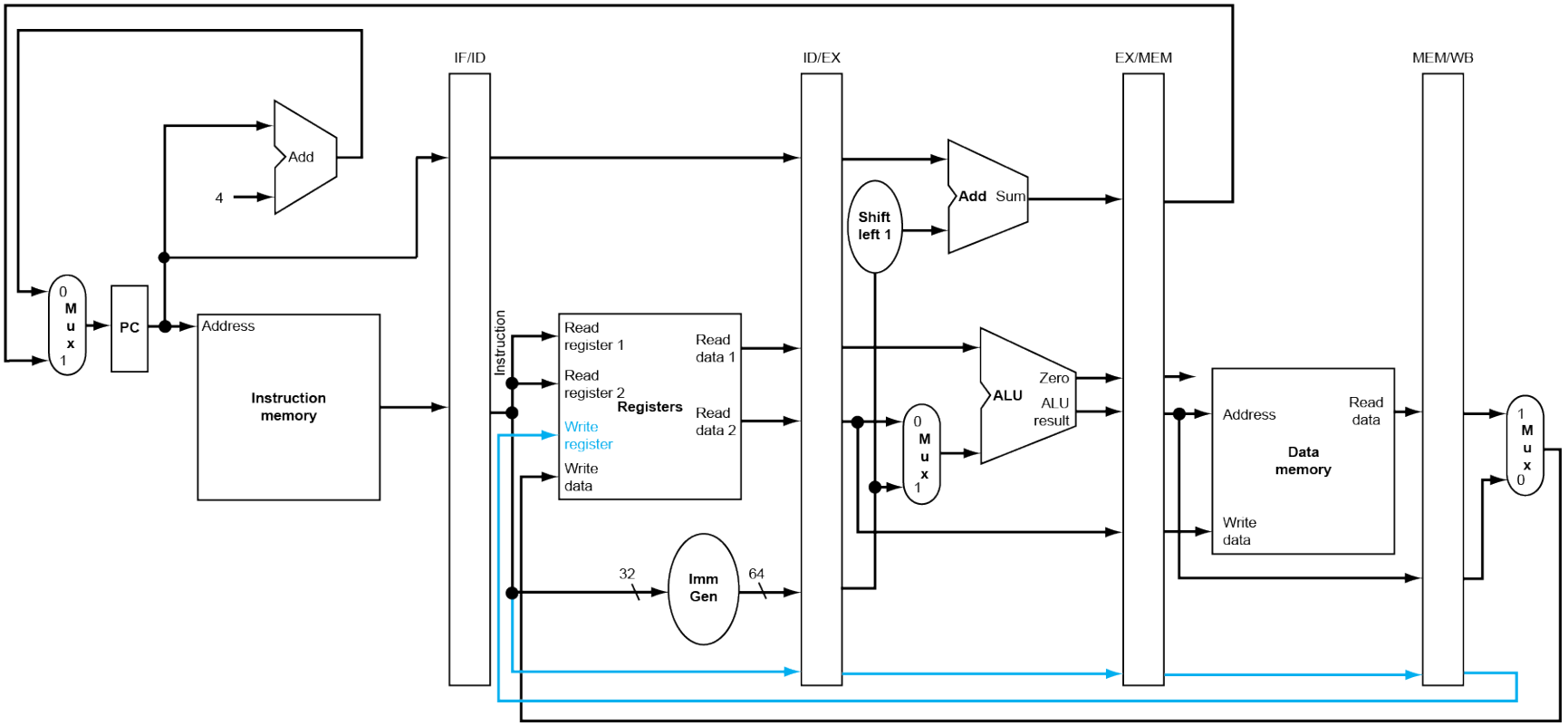
MEM for Load


ld
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
Add Sum
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
32
Imm Gen
64

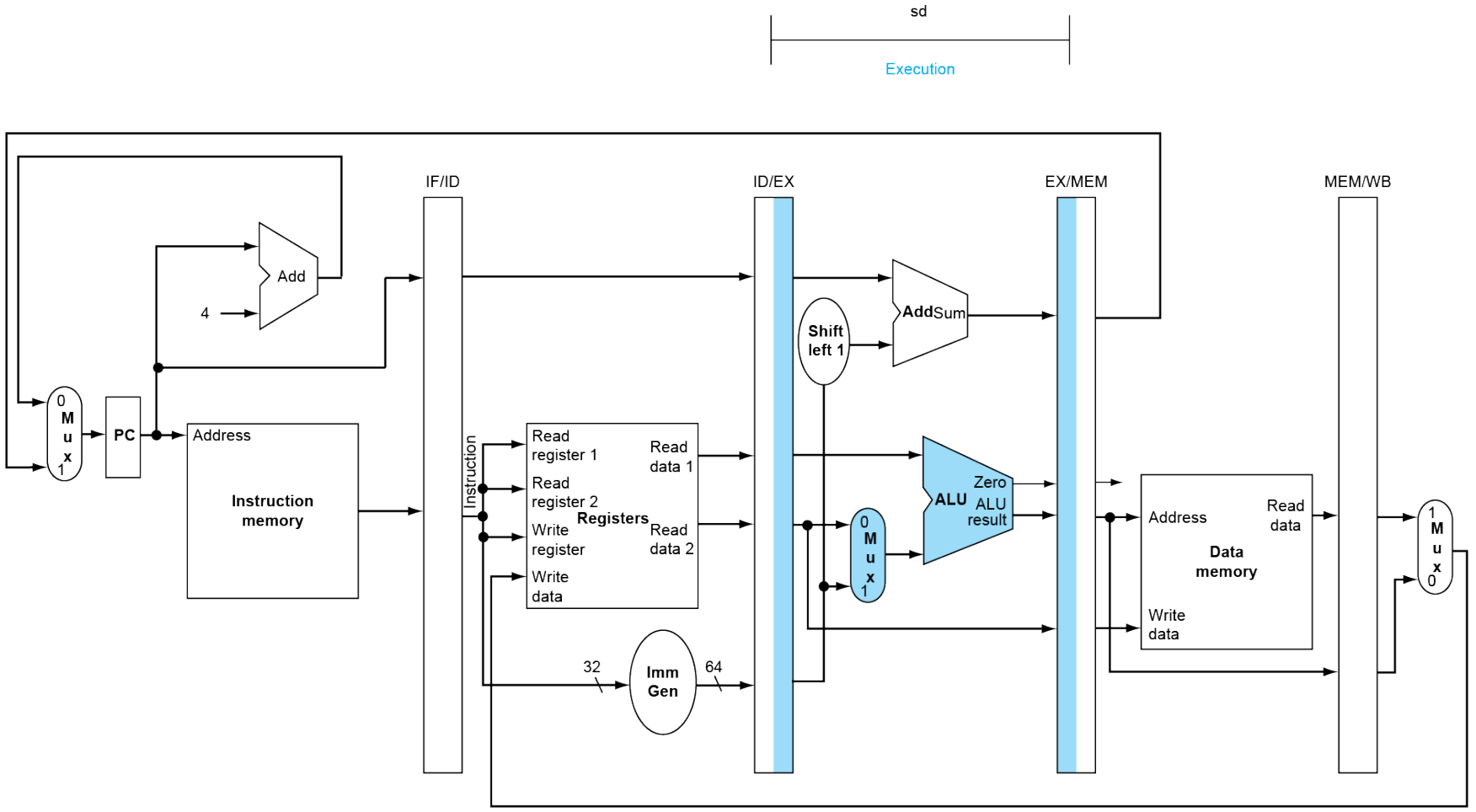
WB for Load


ld
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
Add
Sum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
32
Imm Gen
64
Wrong register number

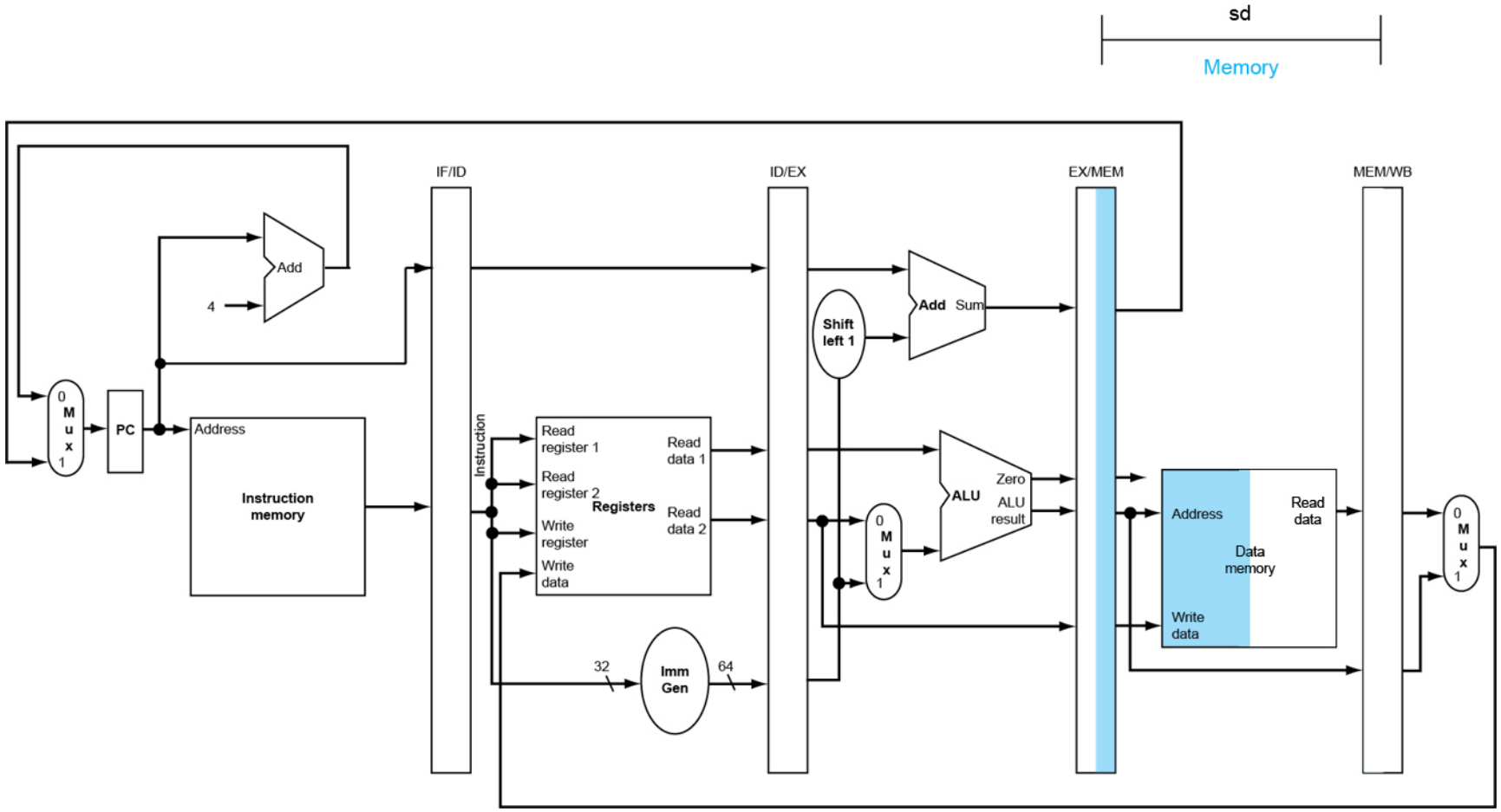
Corrected Datapath for Load



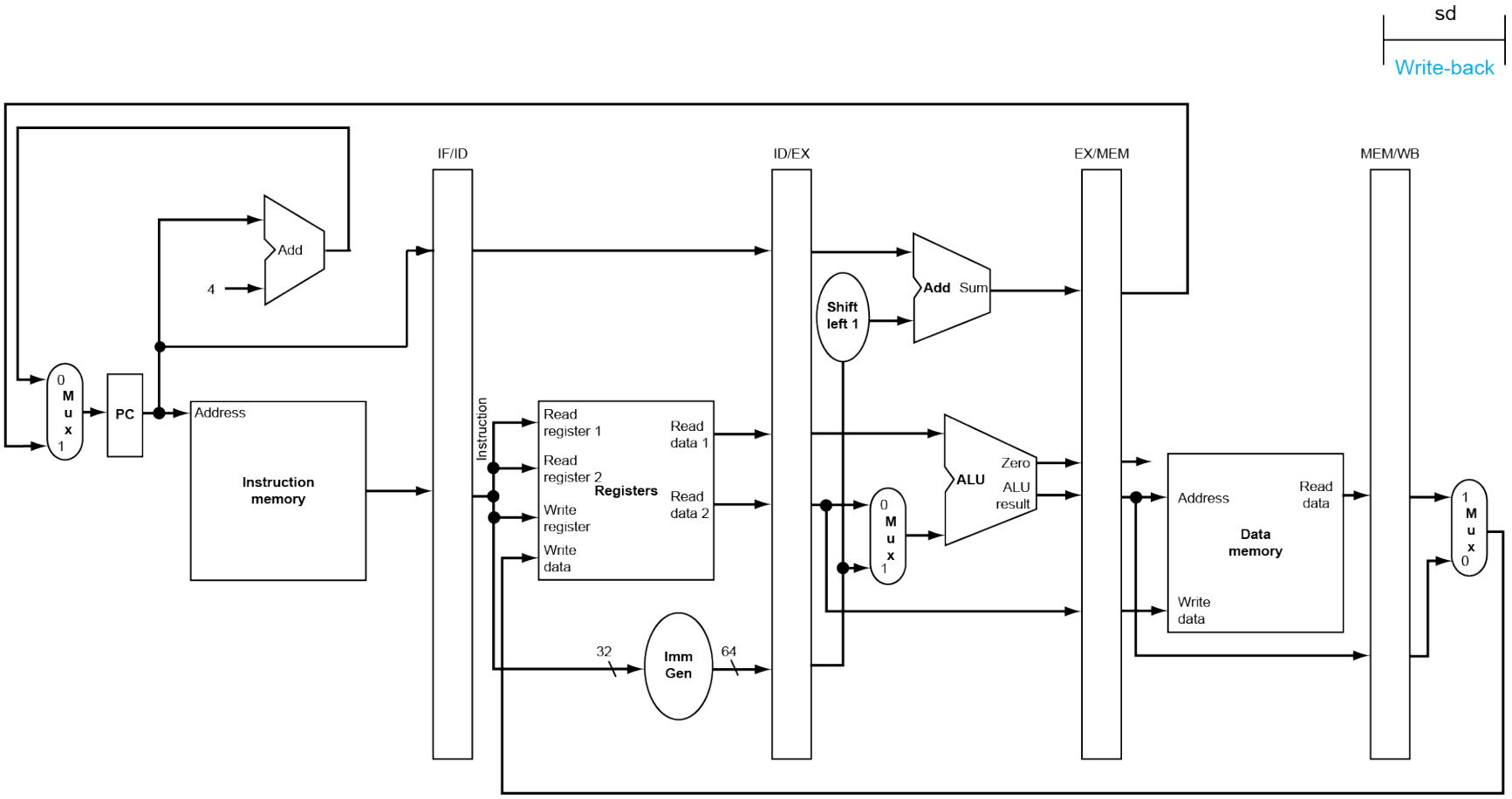
EX for Store


sd
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
AddSum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
32
Imm Gen
64

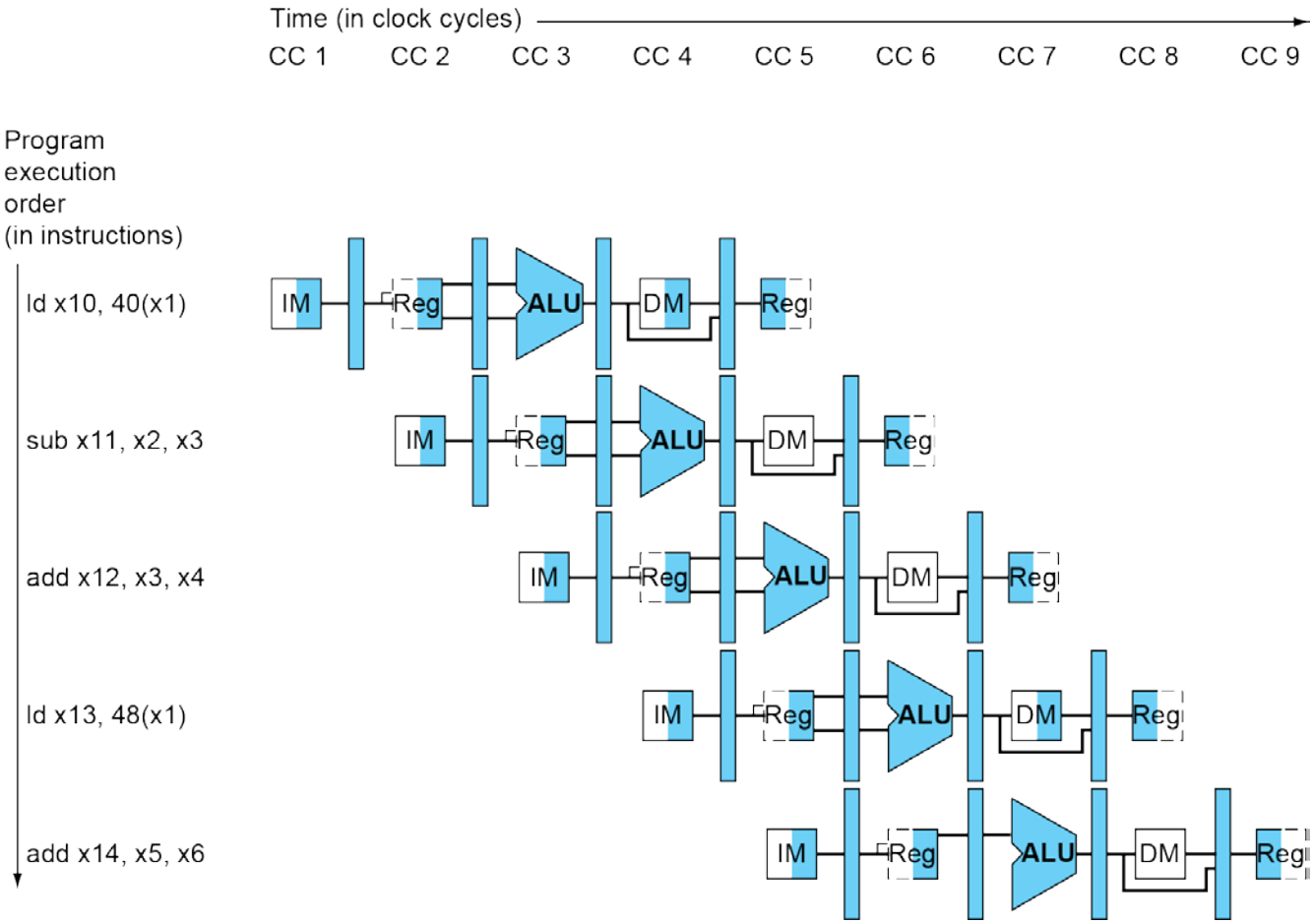
MEM for Store



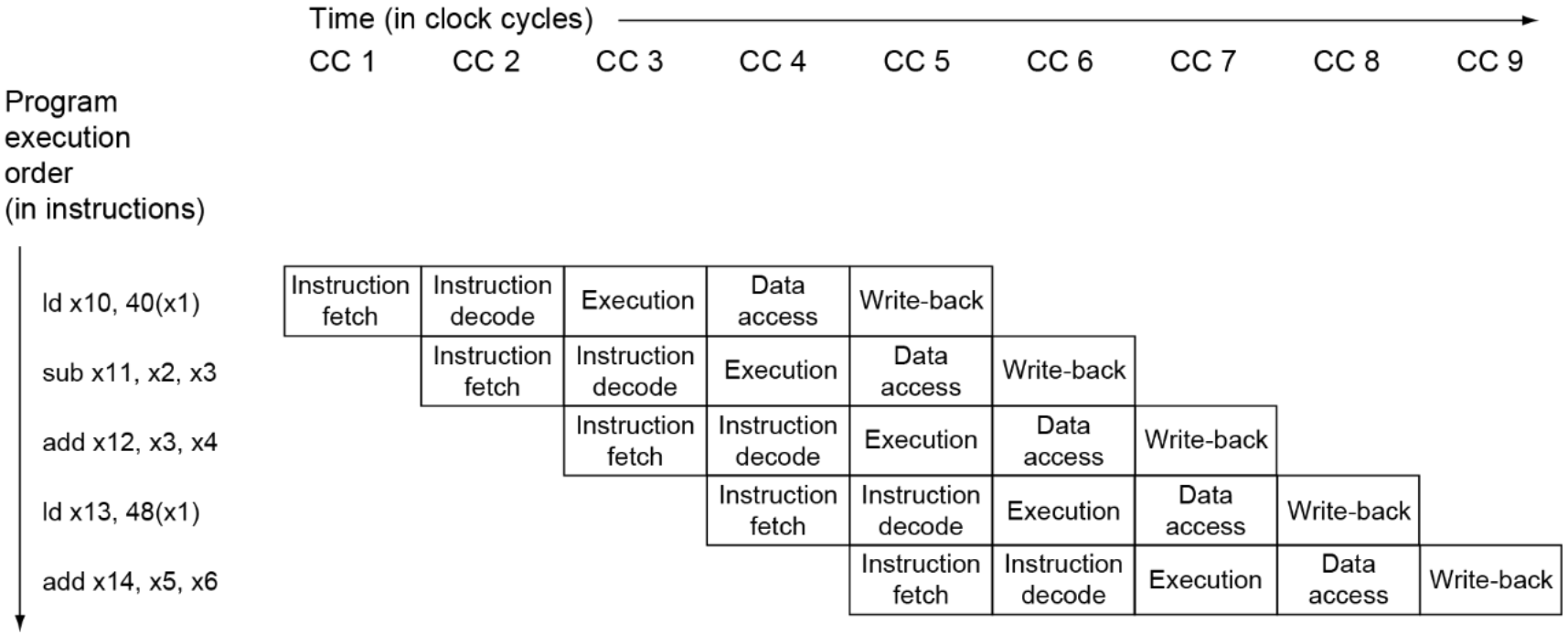
WB for Store


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

- Form showing resource usage

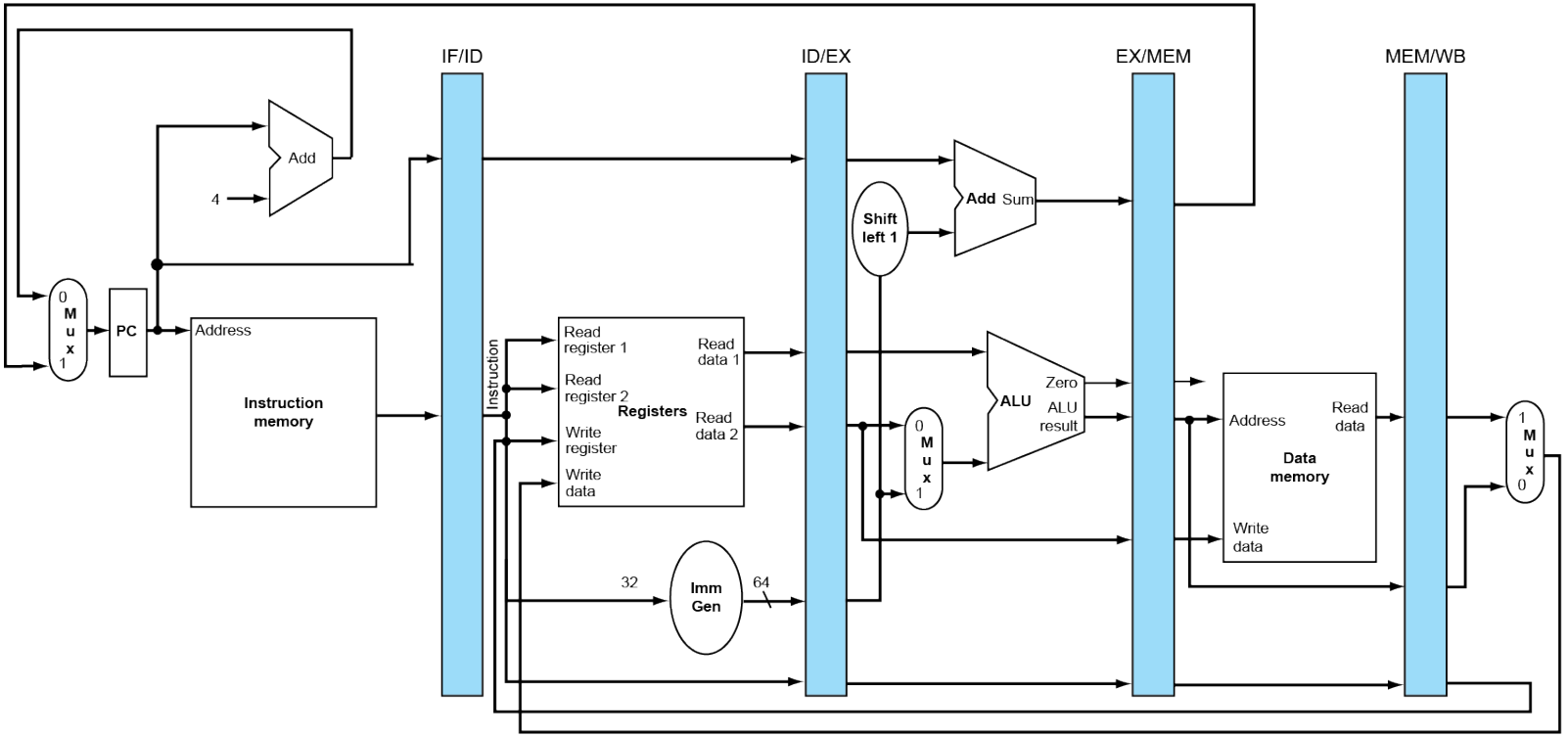


- Traditional form

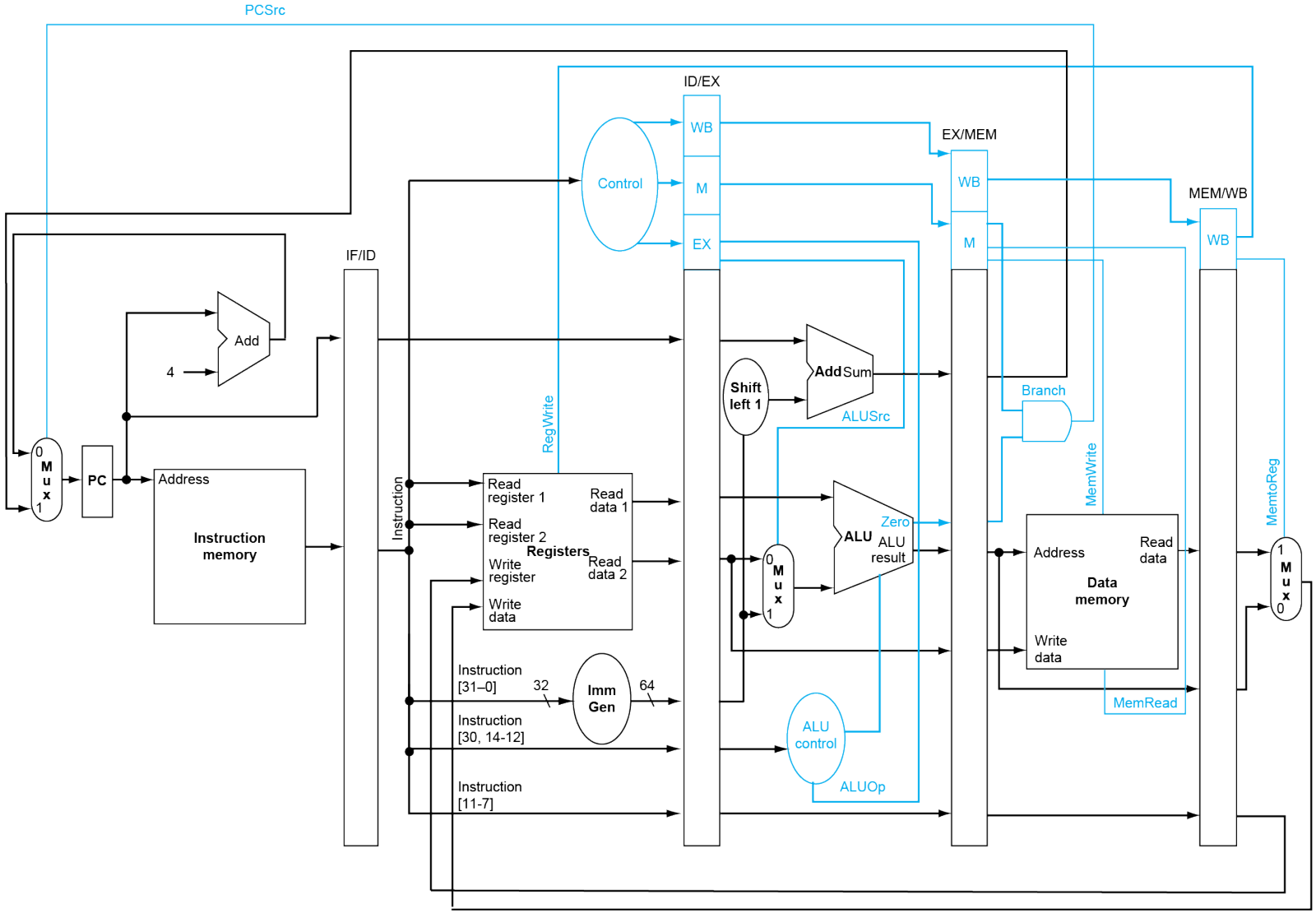


- State of pipeline in a given cycle

add x14, x5, x6	ld x13, 48(x1)	add x12, x3, x4	sub x11, x2, x3	ld x10, 40(x1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back

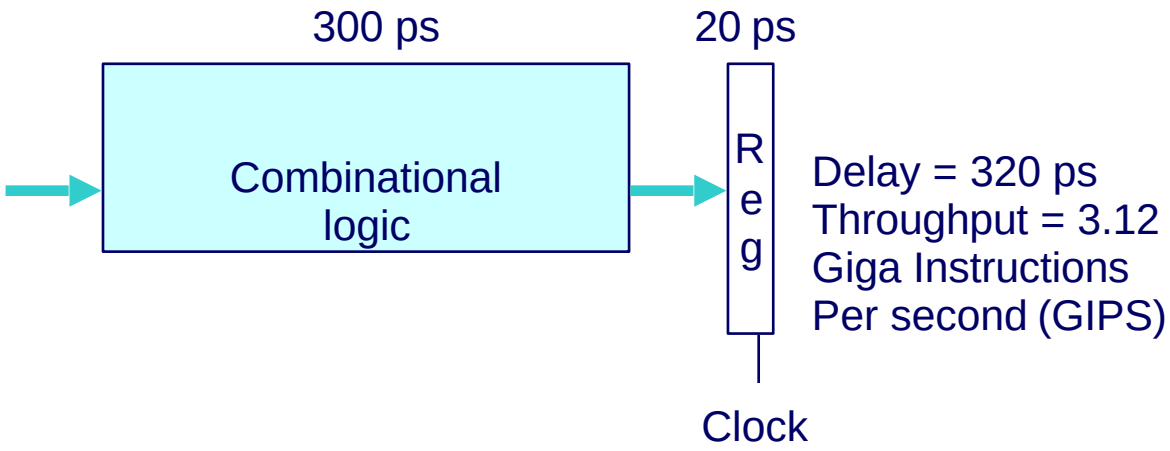


- Control signals derived from instruction (as in single-cycle implementation)



Informació Extra:
Comparació Processador pipelined vs unpipelined

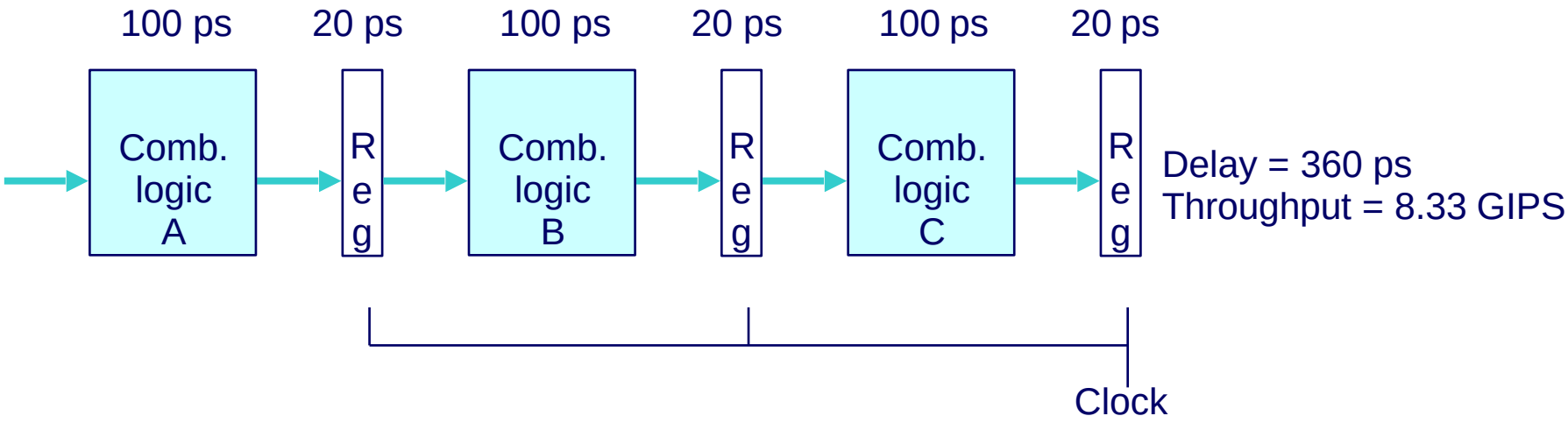
Performance in a “single-cycle” processor



System

- Computation requires total of 300 picoseconds
- Additional 20 picoseconds to save result in register
- Must have clock cycle time of at least 320 ps

Performance in a "3-Stage Pipelined" processor



System

- Divide combinational logic into 3 blocks of 100 ps each
- Can begin new operation as soon as previous one passes through stage A.
  - Begin new operation every 120 ps
- Overall latency increases
  - 360 ps from start to finish