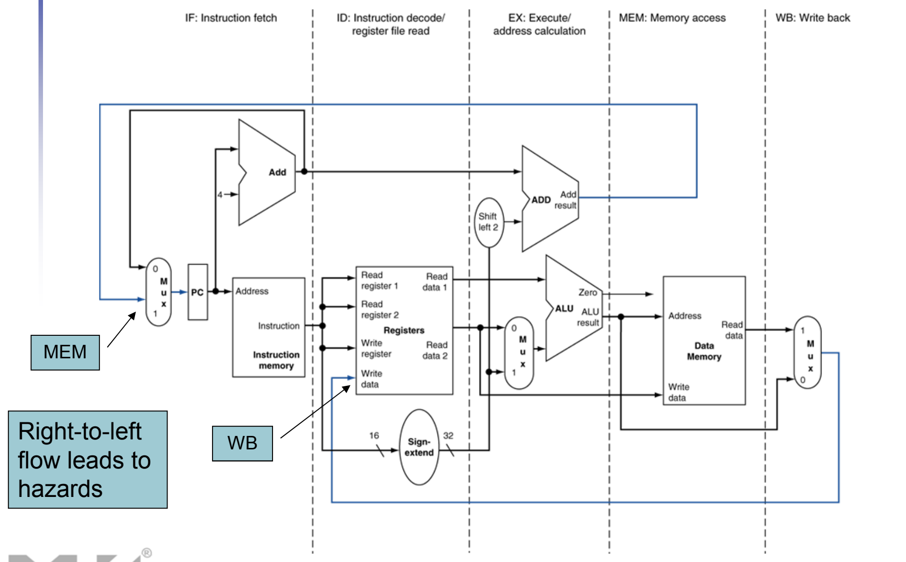
MIPS Pipelined Datapath

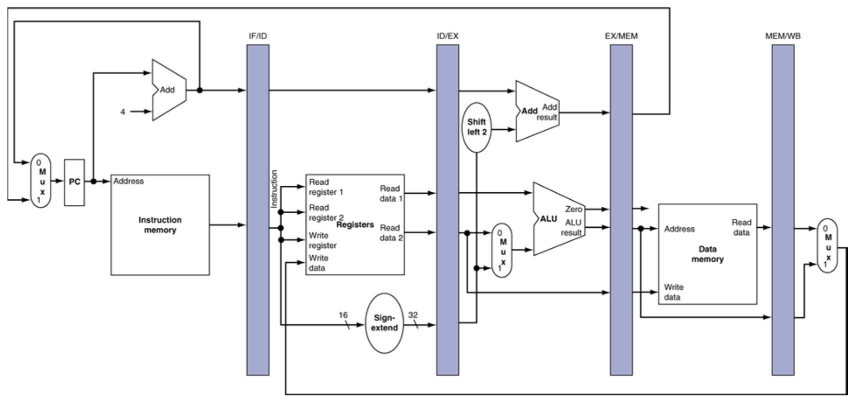
§4.6 Pipelined Datapath and Control



Right-to-left flow leads to hazards

Pipeline registers

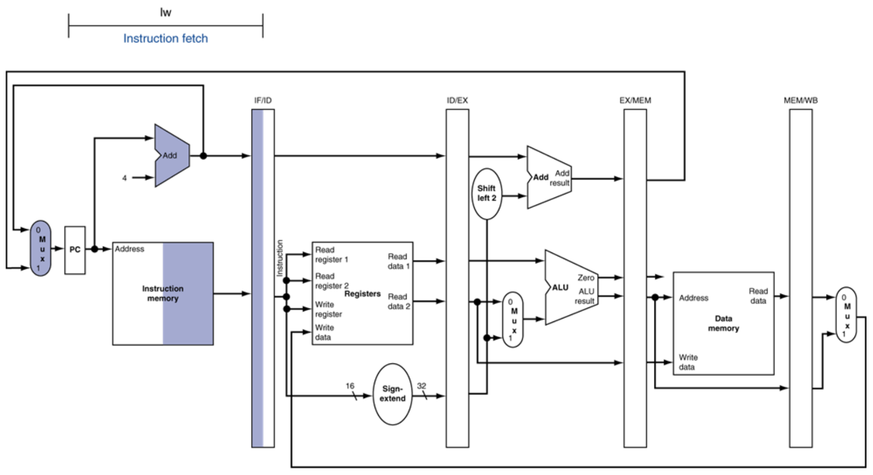
- Need registers between stages
  - To hold information produced in previous cycle



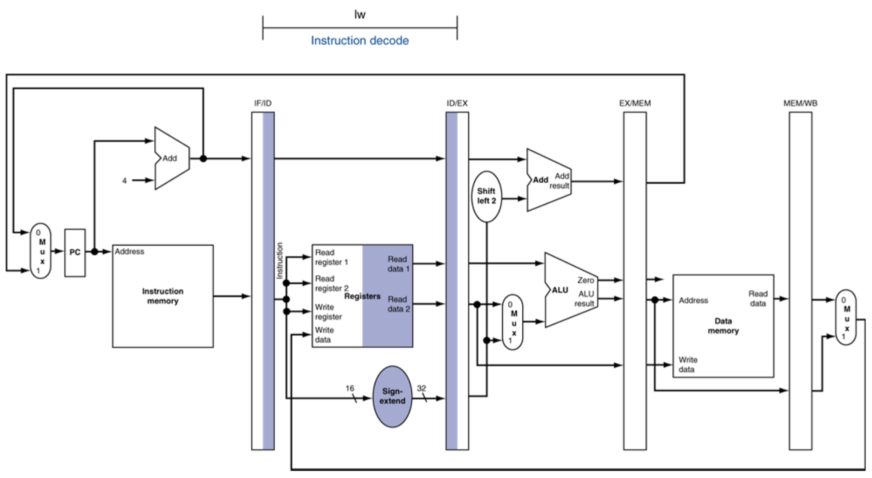
Pipeline Operation

- 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, …

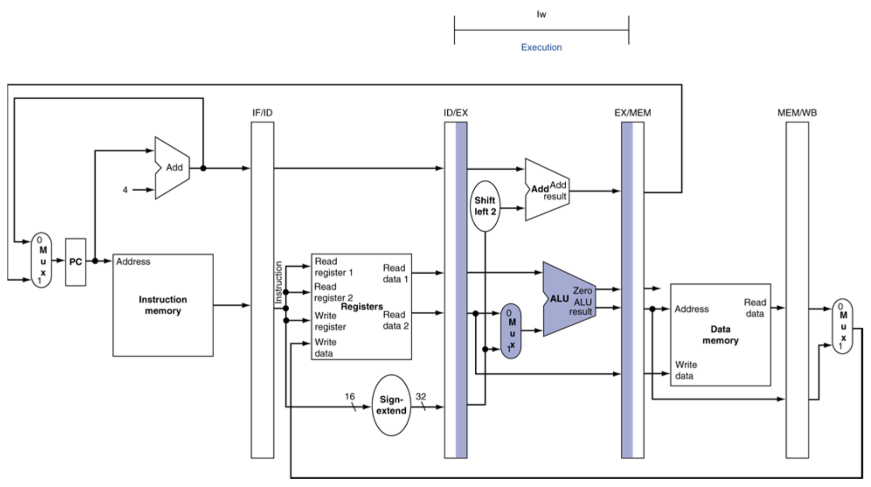


ID for Load, Store, …



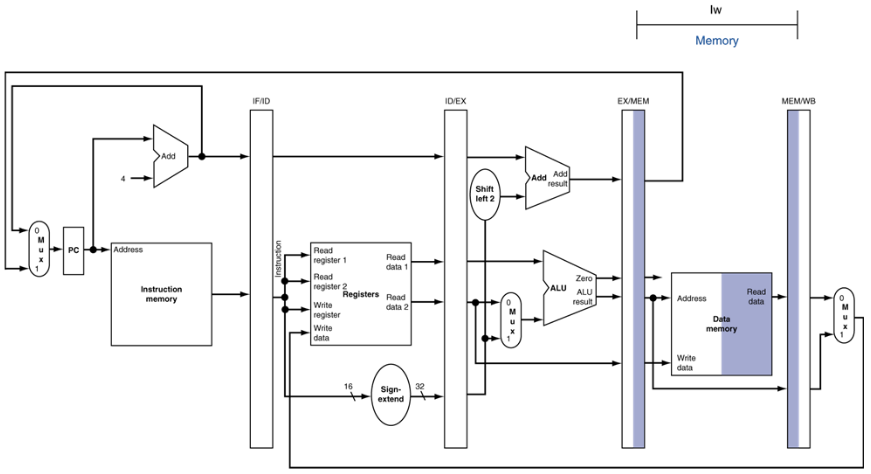
Chapter 4 — The Processor — 79

EX for Load

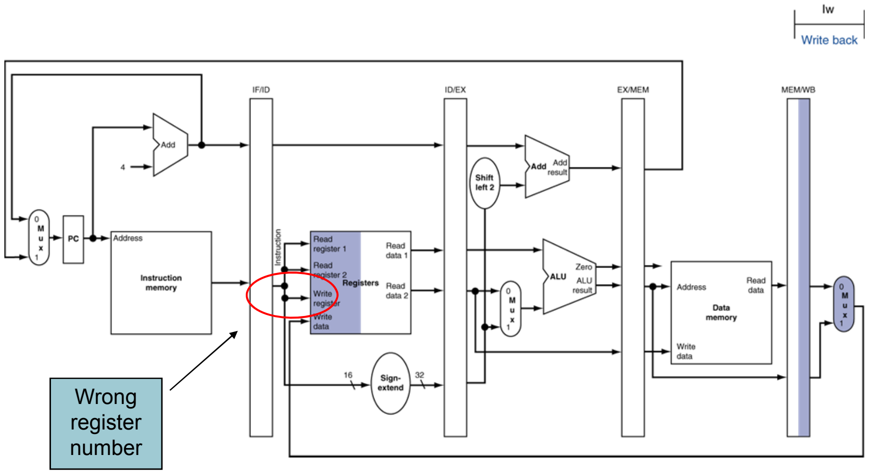


Chapter 4 — The Processor — 80

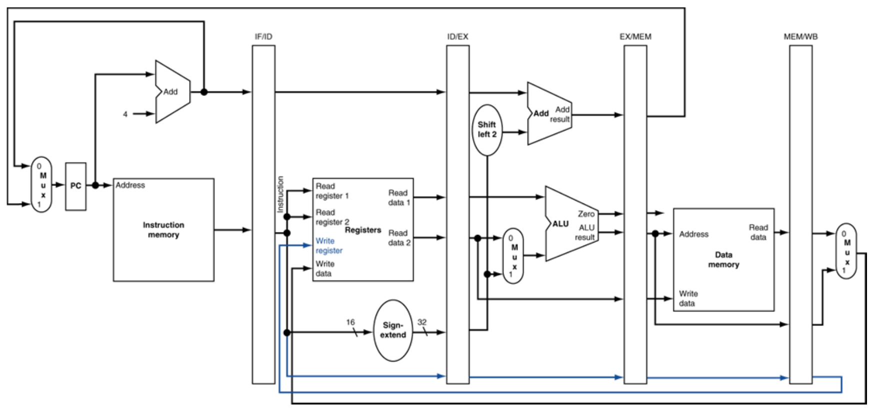
MEM for Load



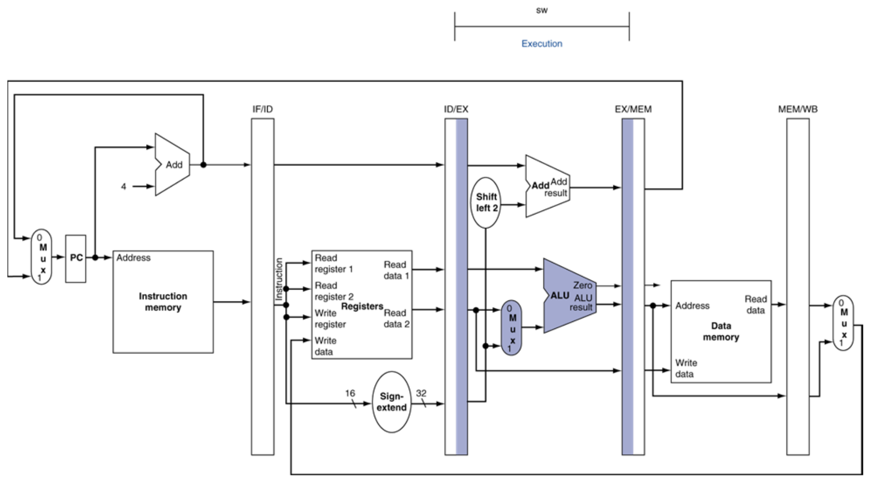
WB for Load

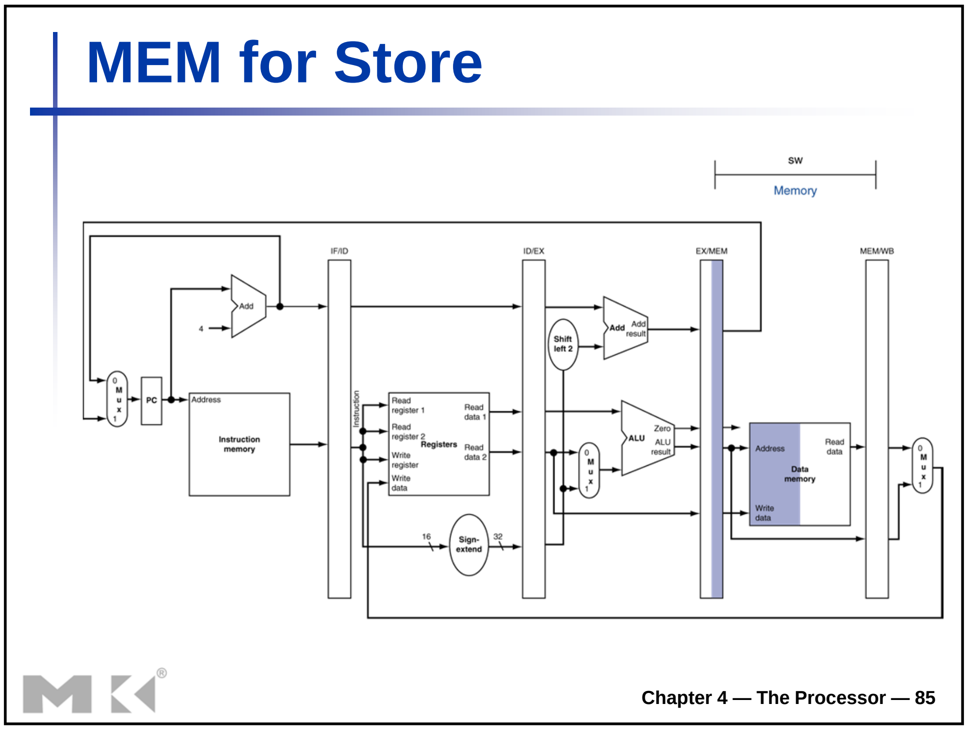


Corrected Datapath for Load

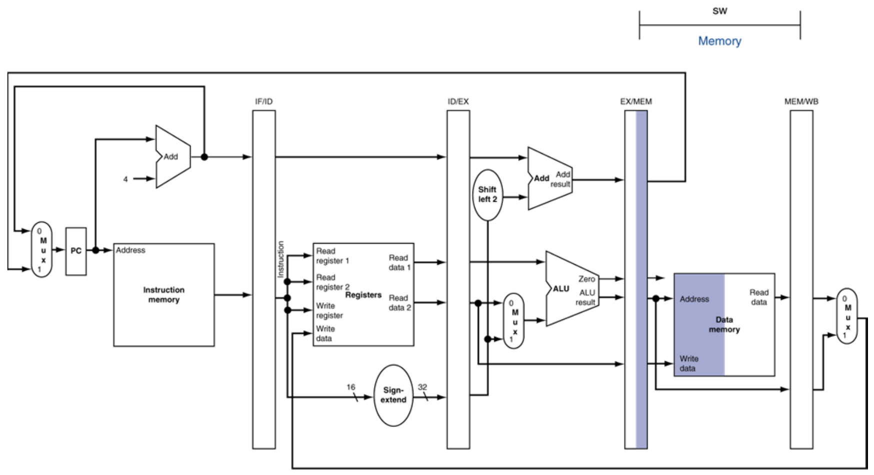


EX for Store

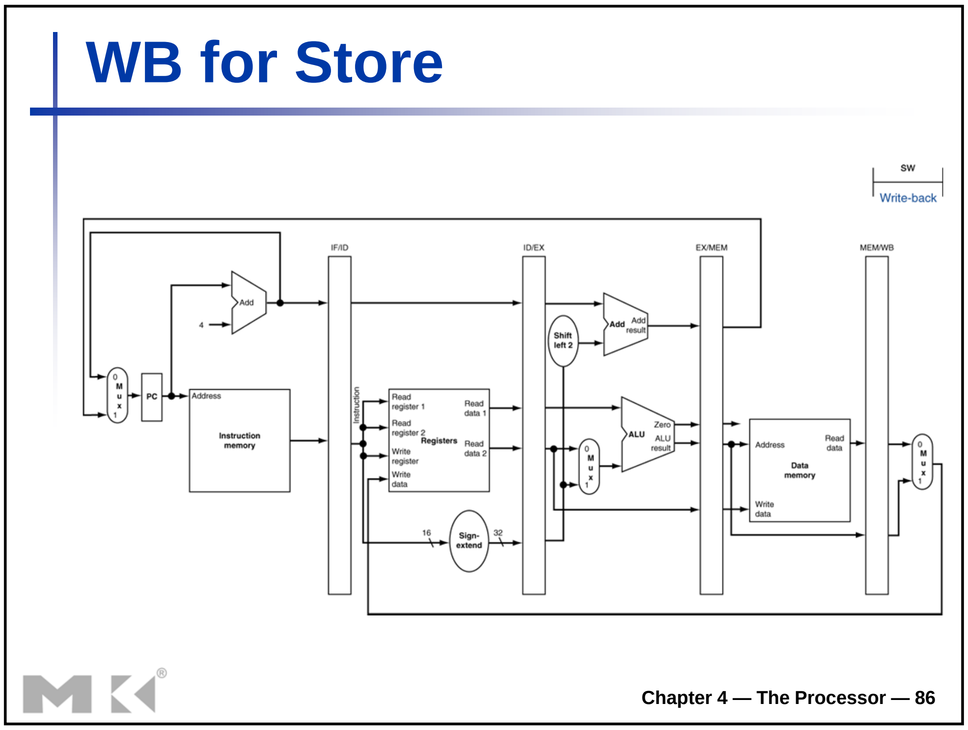




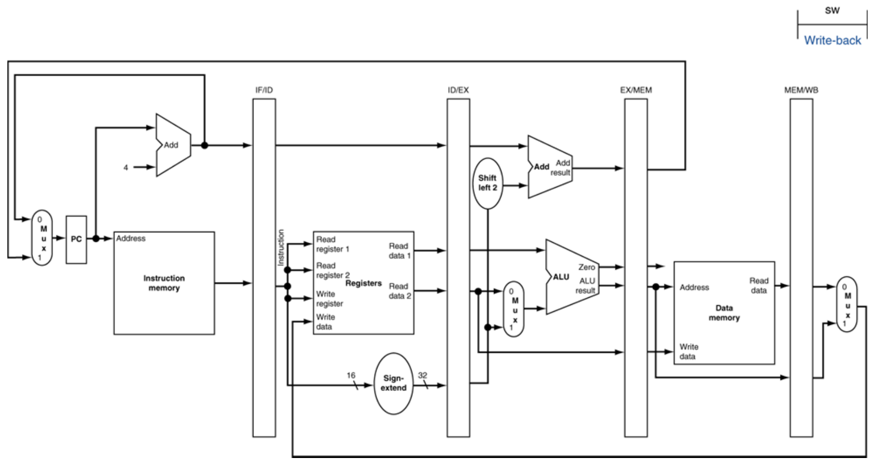
MEM for Store



Chapter 4 — The Processor — 85



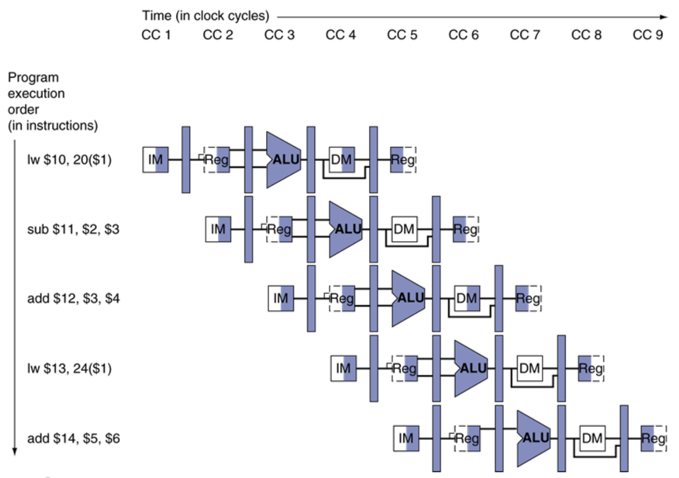
WB for Store



Chapter 4 — The Processor — 86

Multi-Cycle Pipeline Diagram

- Form showing resource usage



Chapter 4 — The Processor — 87

Multi-Cycle Pipeline Diagram

- Traditional form

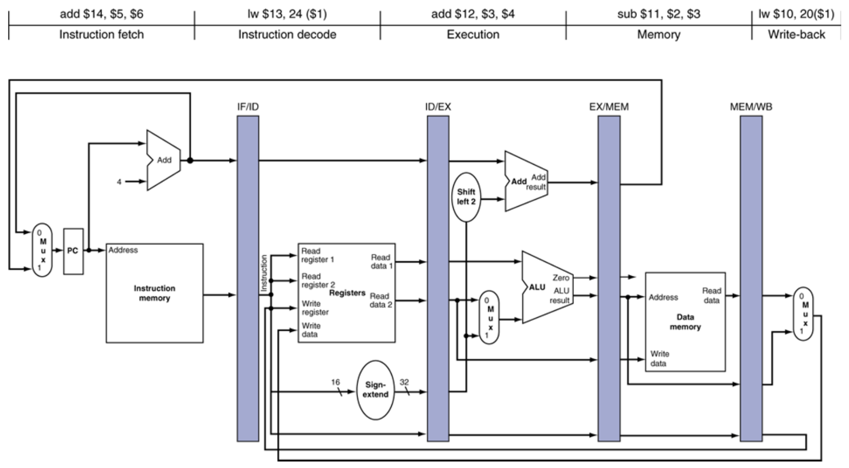
Time (in clock cycles)

Program execution order (in instructions)	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
lw $10, 20($1)	Instruction fetch	Instruction decode	Execution	Data access	Write back				
sub $11, $2, $3		Instruction fetch	Instruction decode	Execution	Data access	Write back			
add $12, $3, $4			Instruction fetch	Instruction decode	Execution	Data access	Write back		
lw $13, 24($1)				Instruction fetch	Instruction decode	Execution	Data access	Write back	
add $14, $5, $6					Instruction fetch	Instruction decode	Execution	Data access	Write back

Chapter 4 — The Processor — 88

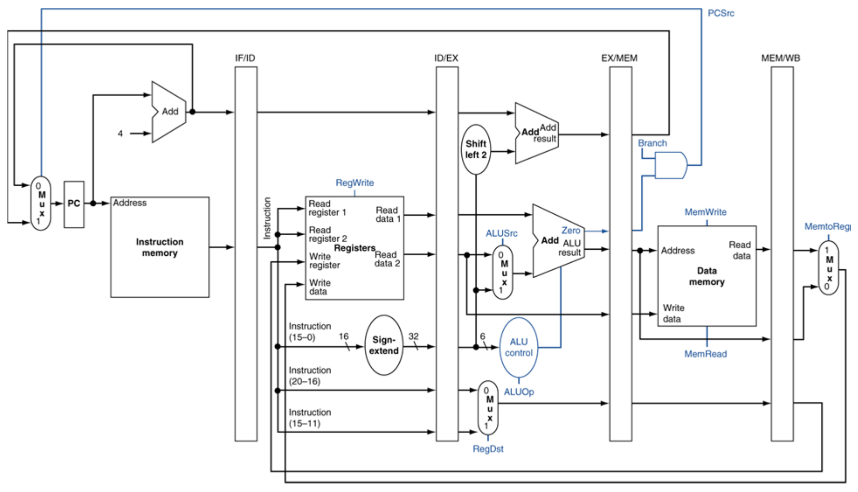
Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle



Chapter 4 — The Processor — 89

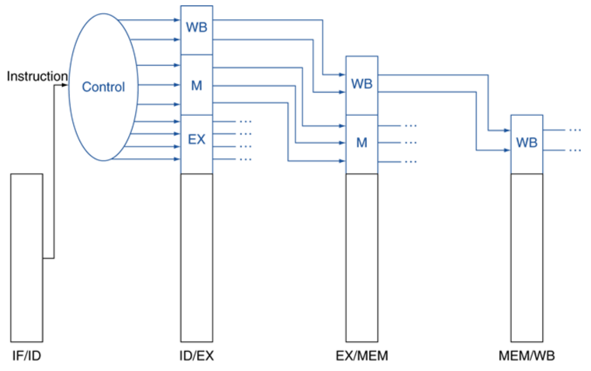
Pipelined Control (Simplified)



Chapter 4 — The Processor — 90

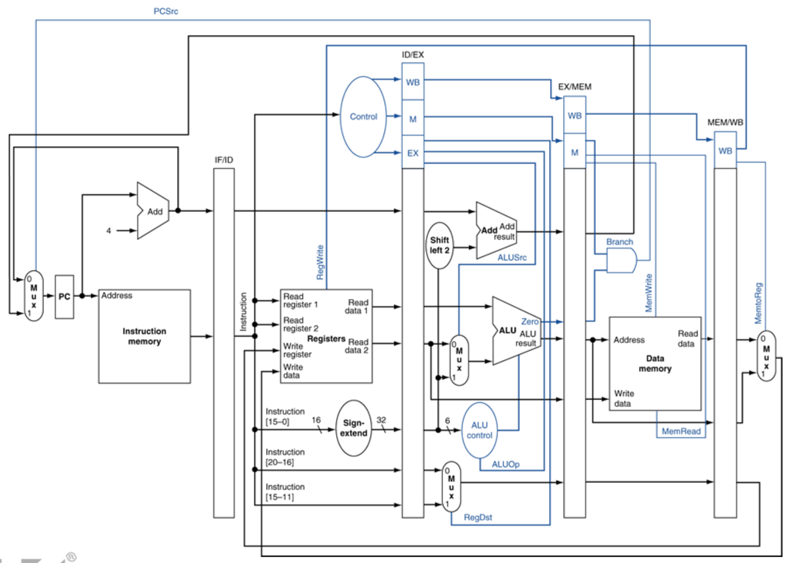
Pipelined Control

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



Chapter 4 — The Processor — 91

Pipelined Control



Chapter 4 — The Processor — 92

Data Hazards in ALU Instructions

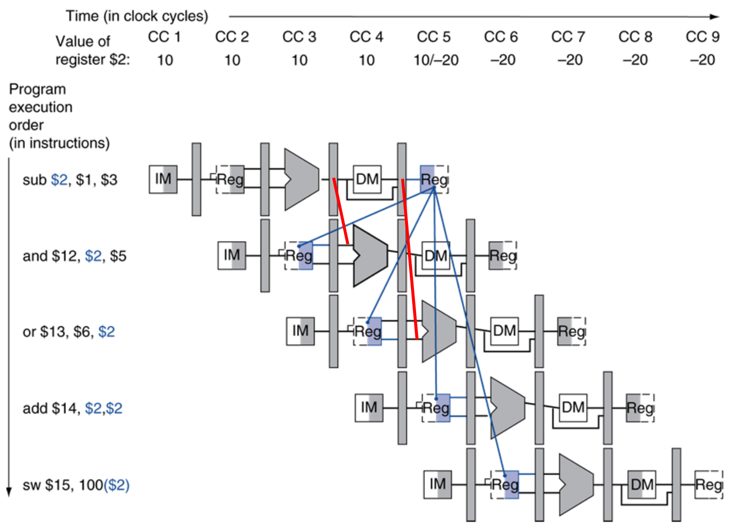
§4.7 Data Hazards: Forwarding vs. Stalling

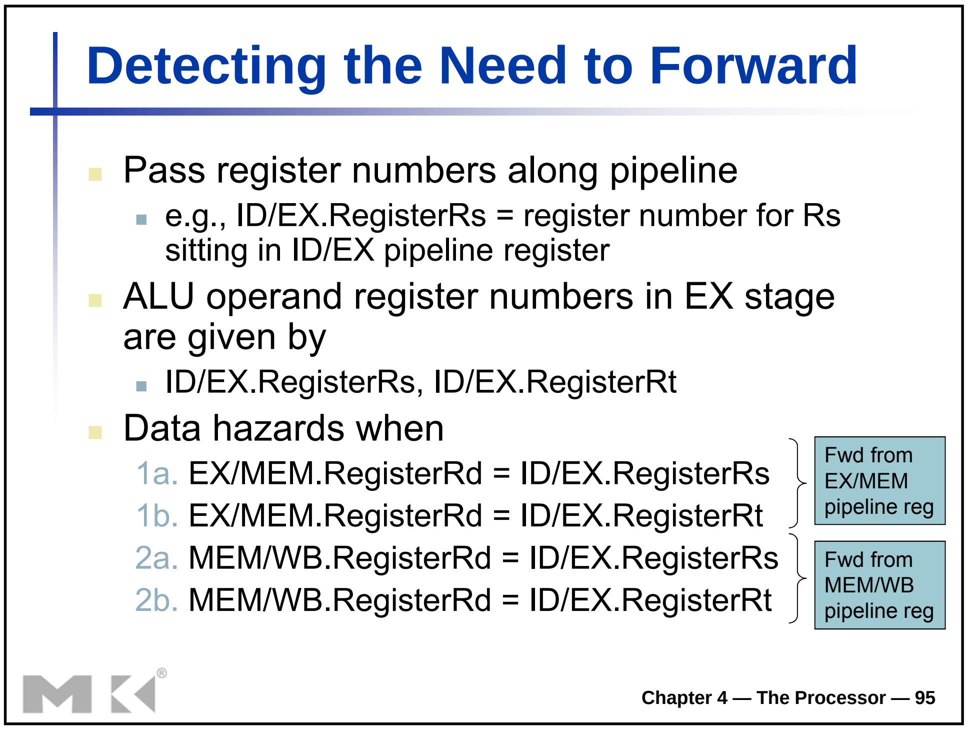
- Consider this sequence:

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- We can resolve hazards with forwarding
  - How do we detect when to forward?

Dependencies & Forwarding


Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Value of register $2: 10 10 10 10 10/–20 –20 –20 –20 –20
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2,$2
sw $15, 100($2)
IM
Reg
DM
Reg



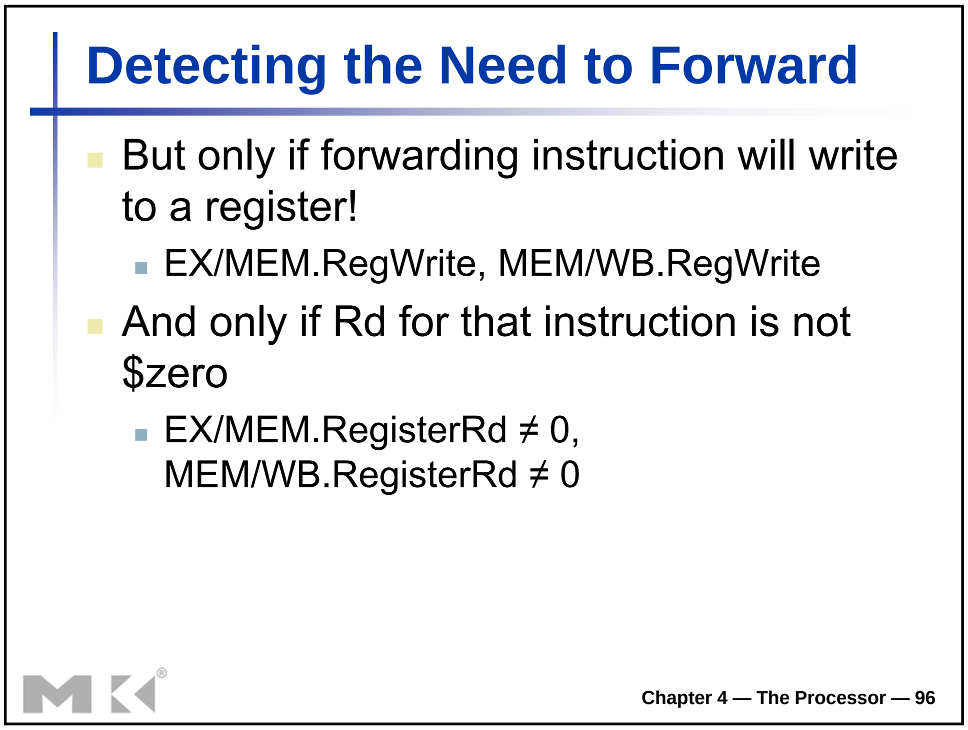
Detecting the Need to Forward

- Pass register numbers along pipeline
  - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs, ID/EX.RegisterRt
- Data hazards when
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

    Fwd from EX/MEM pipeline reg
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

    Fwd from MEM/WB pipeline reg

Chapter 4 — The Processor — 95

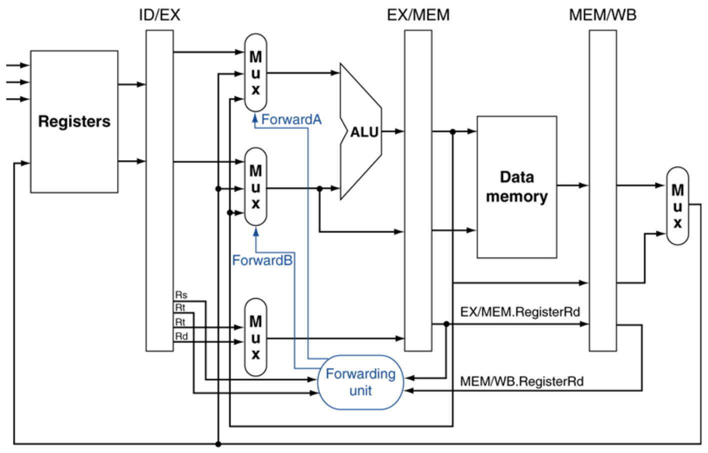


Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not $zero
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

Chapter 4 — The Processor — 96

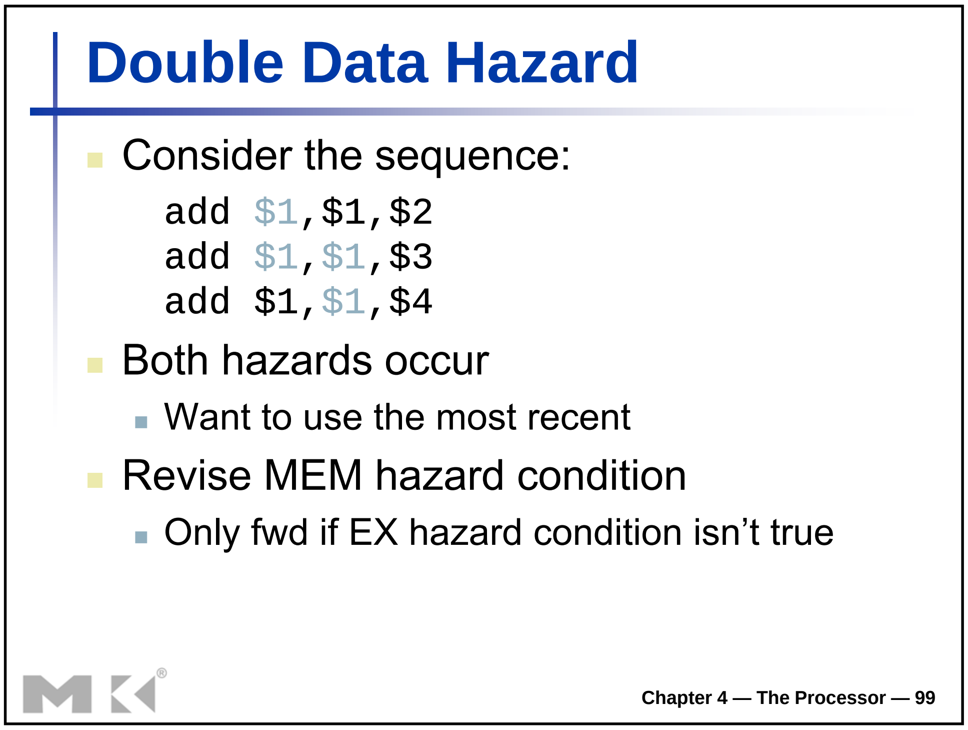
Forwarding Paths



b. With forwarding

Forwarding Conditions

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01



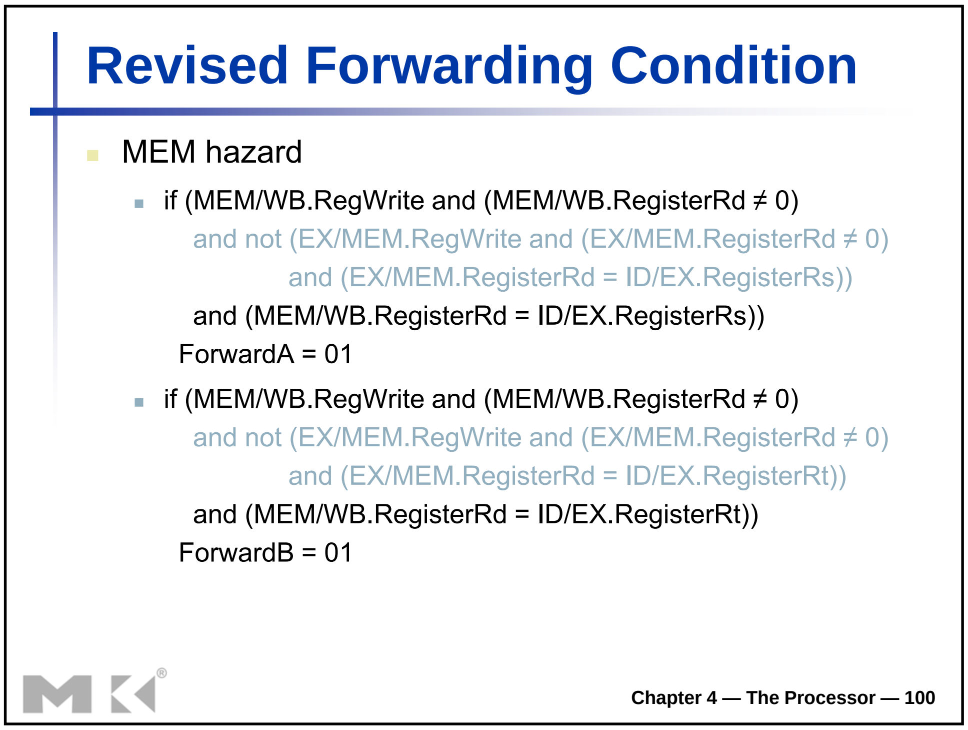
Double Data Hazard

- Consider the sequence:

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn’t true

Chapter 4 — The Processor — 99

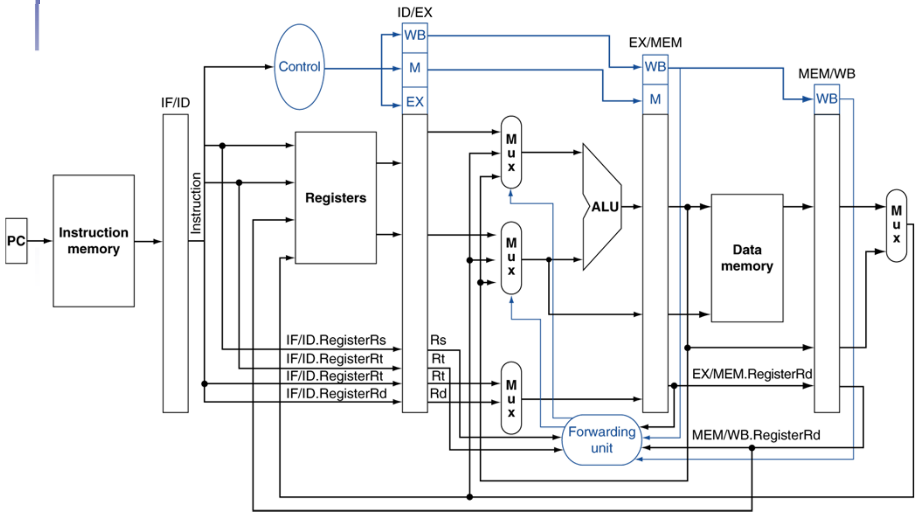


Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
              and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
              and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
      and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

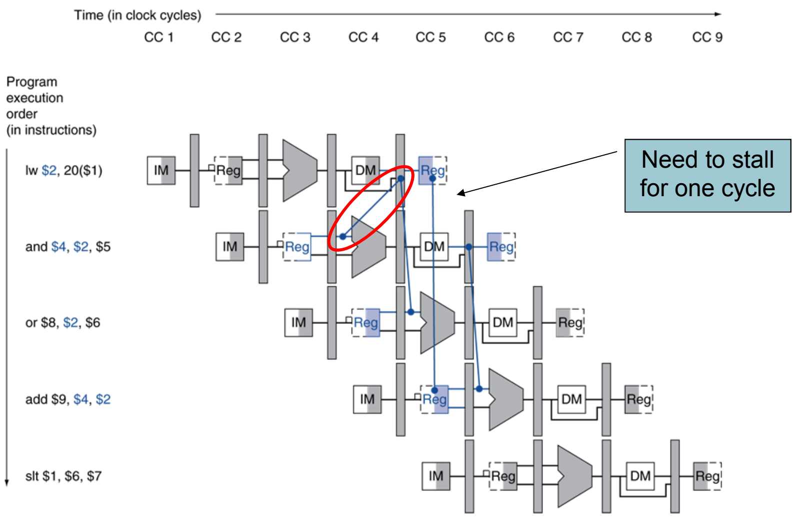
Chapter 4 — The Processor — 100

Datapath with Forwarding

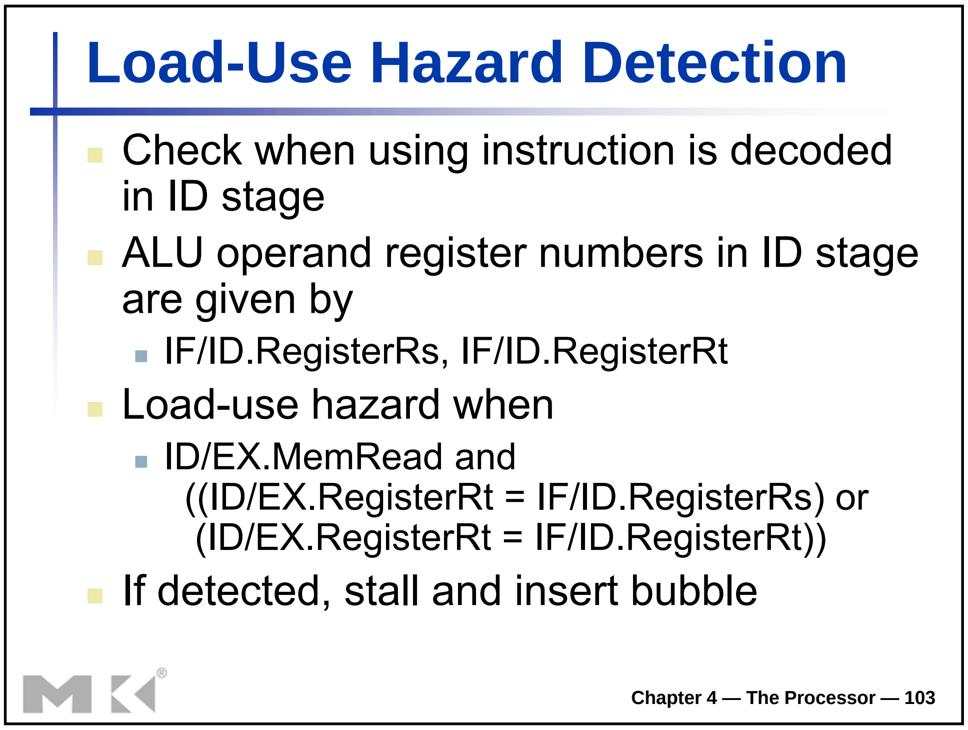


Chapter 4 — The Processor — 101

Load-Use Data Hazard



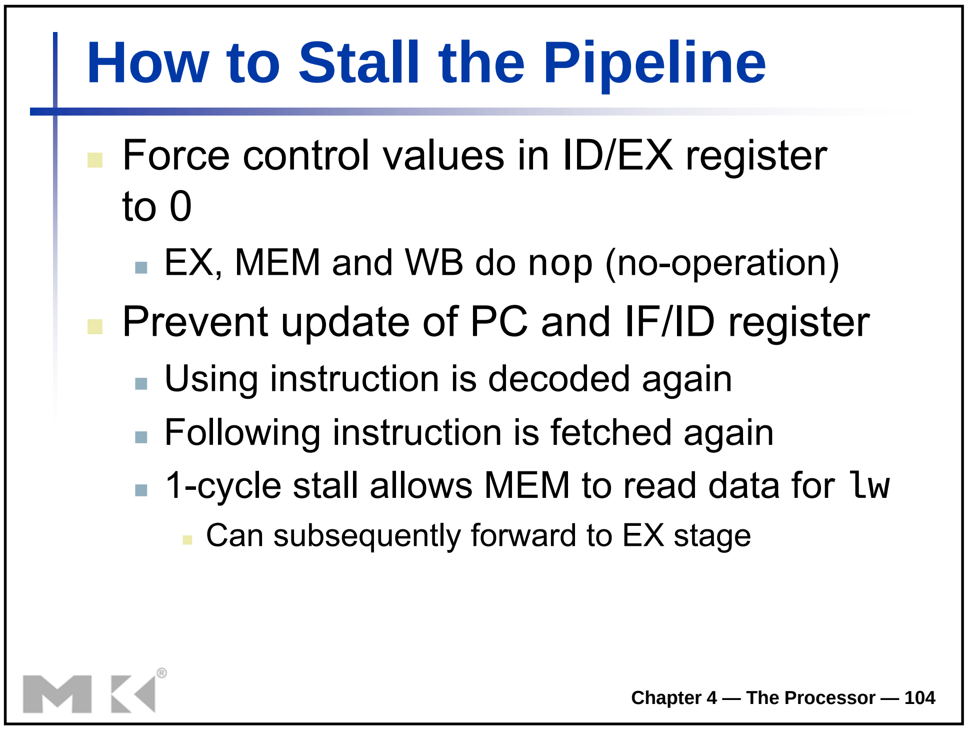
Chapter 4 — The Processor — 102



Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

Chapter 4 — The Processor — 103

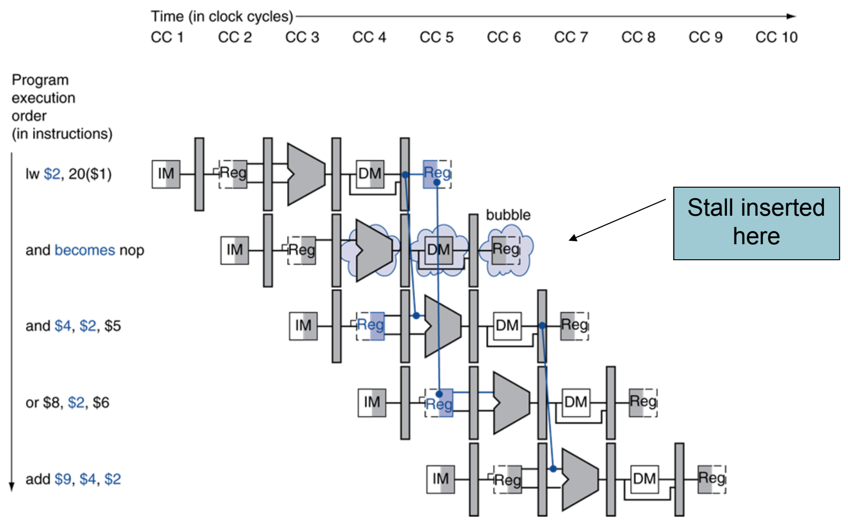


How to Stall the Pipeline

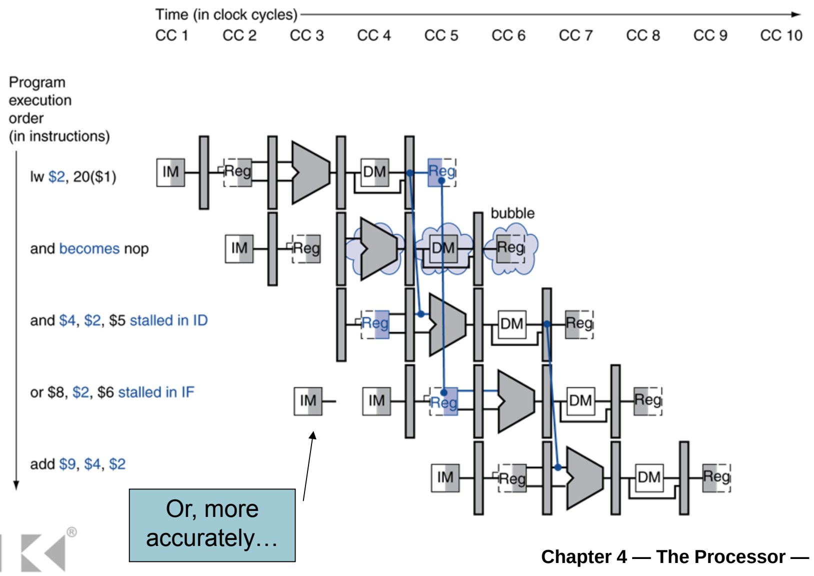
- Force control values in ID/EX register to 0
  - EX, MEM and WB do `nop` (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `lw`
    - Can subsequently forward to EX stage

Chapter 4 — The Processor — 104

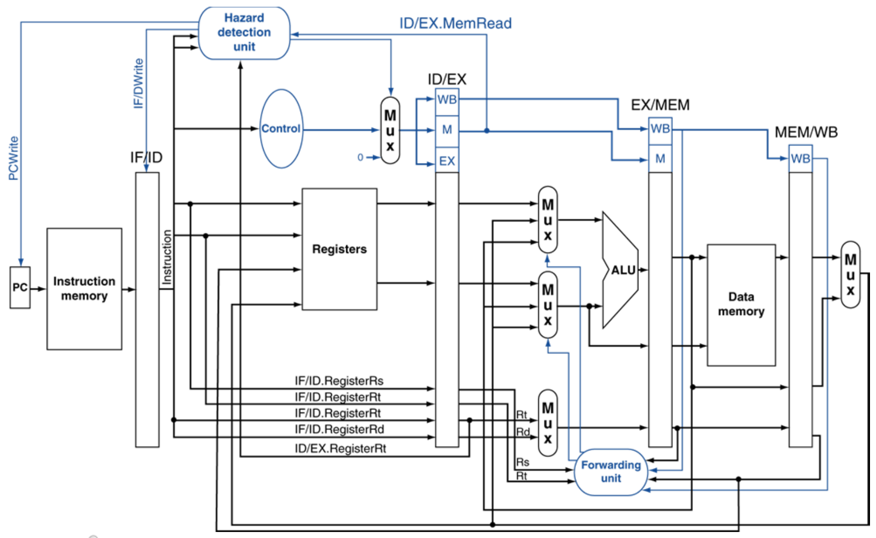
Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline


Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9 CC 10
Program execution order (in instructions)
lw $2, 20($1)
and becomes nop
and $4, $2, $5 stalled in ID
or $8, $2, $6 stalled in IF
add $9, $4, $2
IM
Reg
DM
bubble
Or, more accurately…

Datapath with Hazard Detection



Chapter 4 — The Processor — 107

Stalls and Performance

The BIG Picture

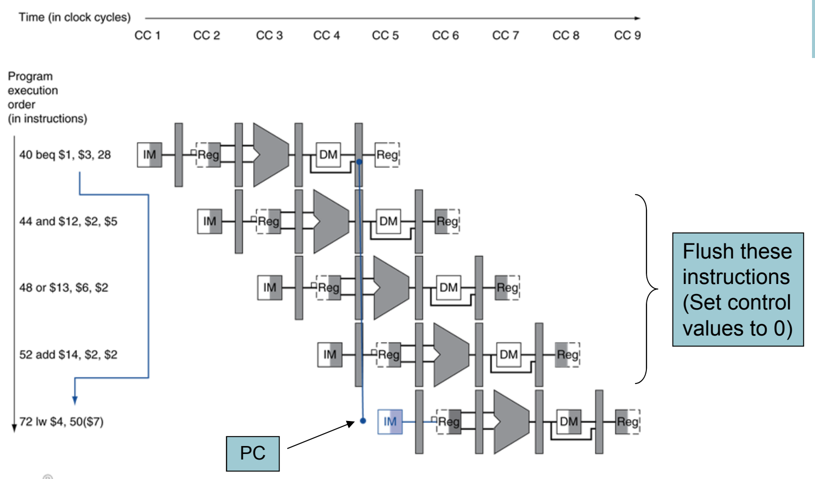
- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

Chapter 4 — The Processor — 108

Branch Hazards

§4.8 Control Hazards

- If branch outcome determined in MEM

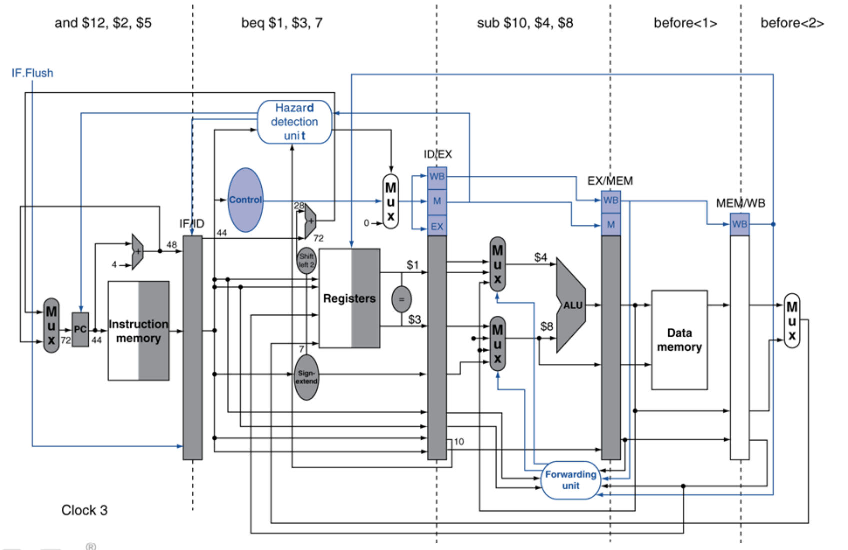


Reducing Branch Delay

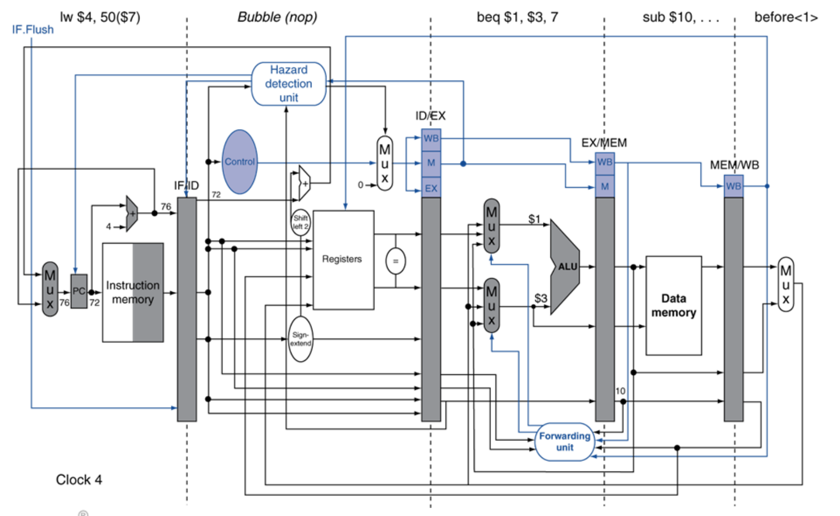
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

Example: Branch Taken



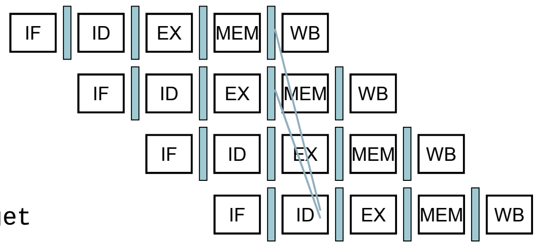
Example: Branch Taken



Data Hazards for Branches

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```



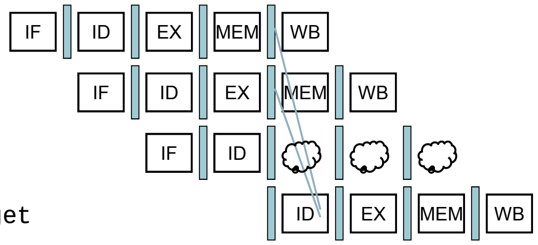
- Can resolve using forwarding

Chapter 4 — The Processor — 113

Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle

```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



Chapter 4 — The Processor — 114