Chapter 4

The Processor

Dr. Randa Mohamed

Agenda

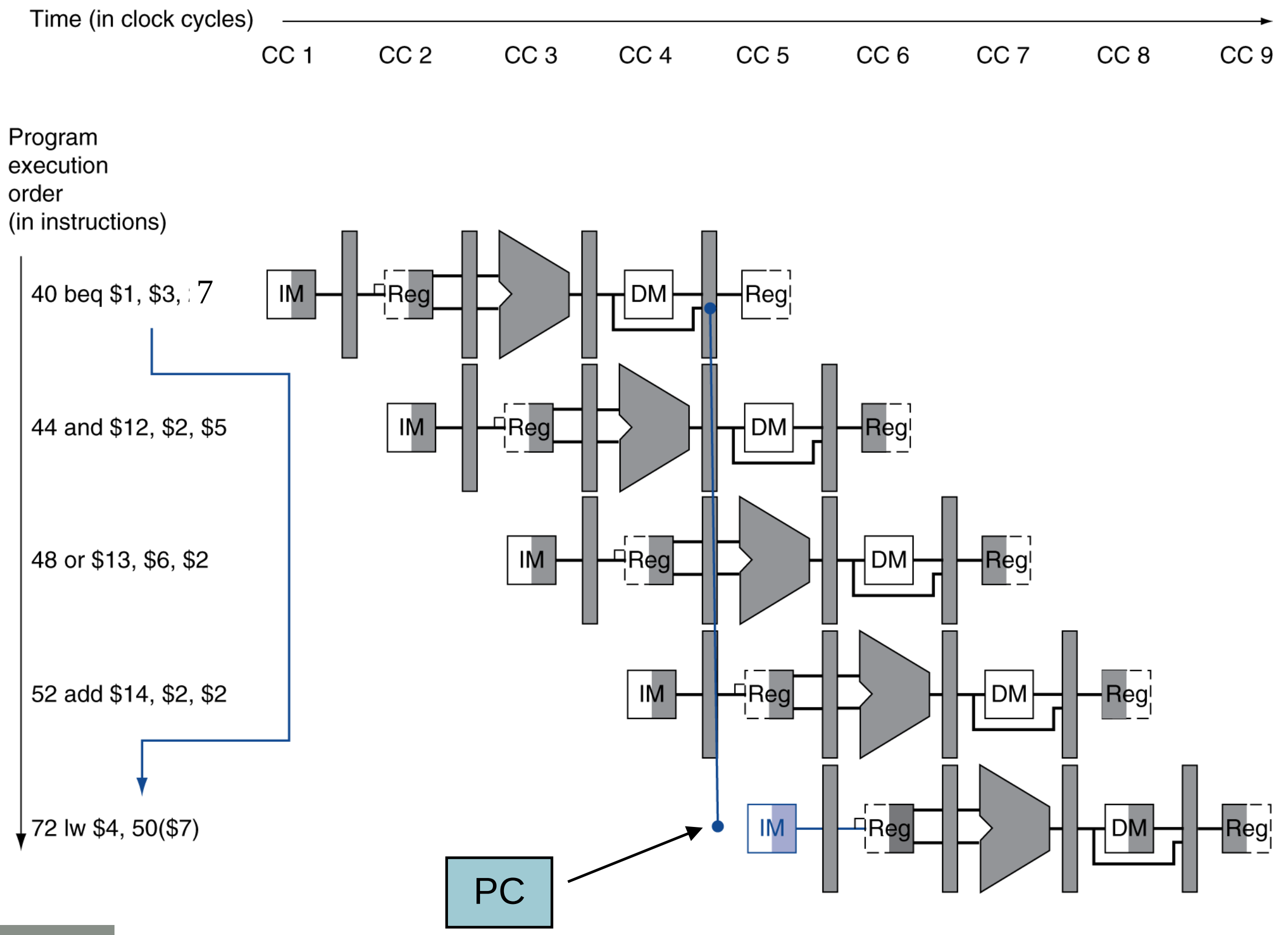
- MIPS Pipeline:
  - Control Hazards
  - Exceptions

Control Hazards

- Pipeline hazards involving branches.
- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch

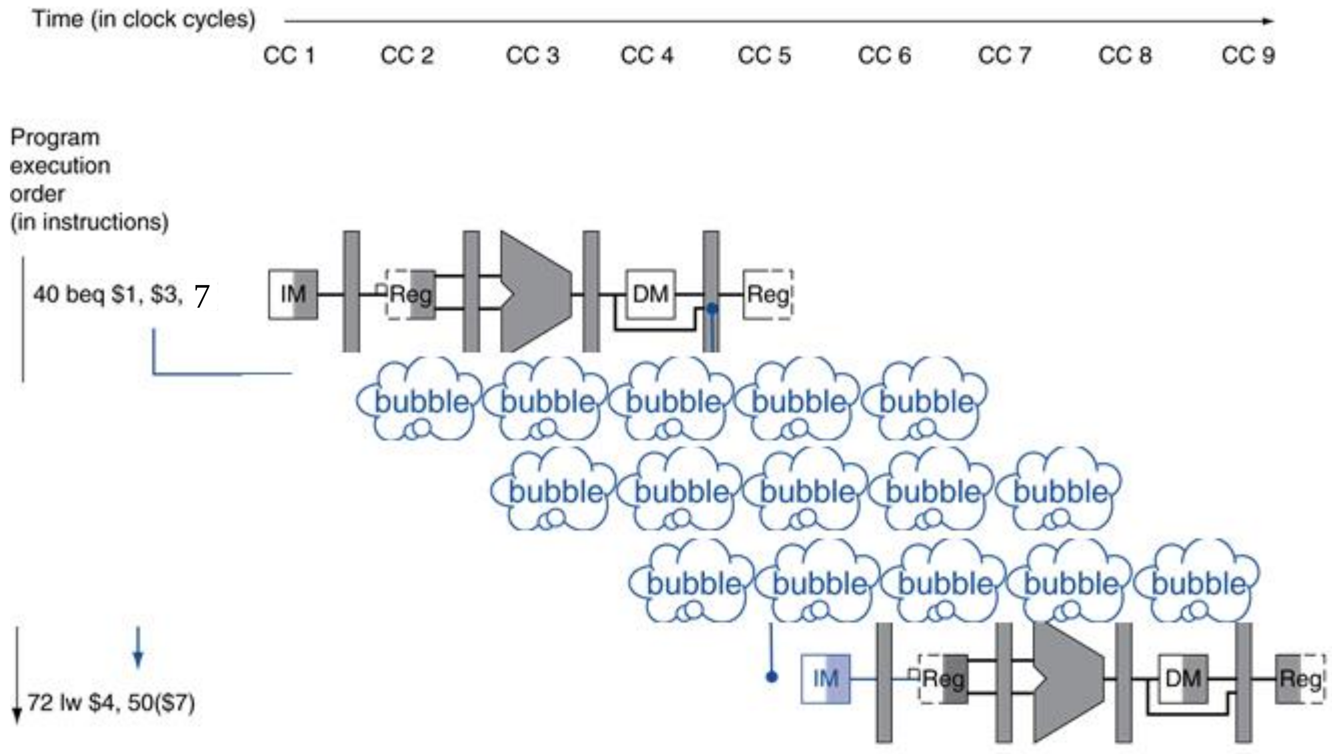
Branch Hazards

- If branch outcome determined in MEM



1) Stall on Branch

- Wait until branch outcome determined before fetching next instruction

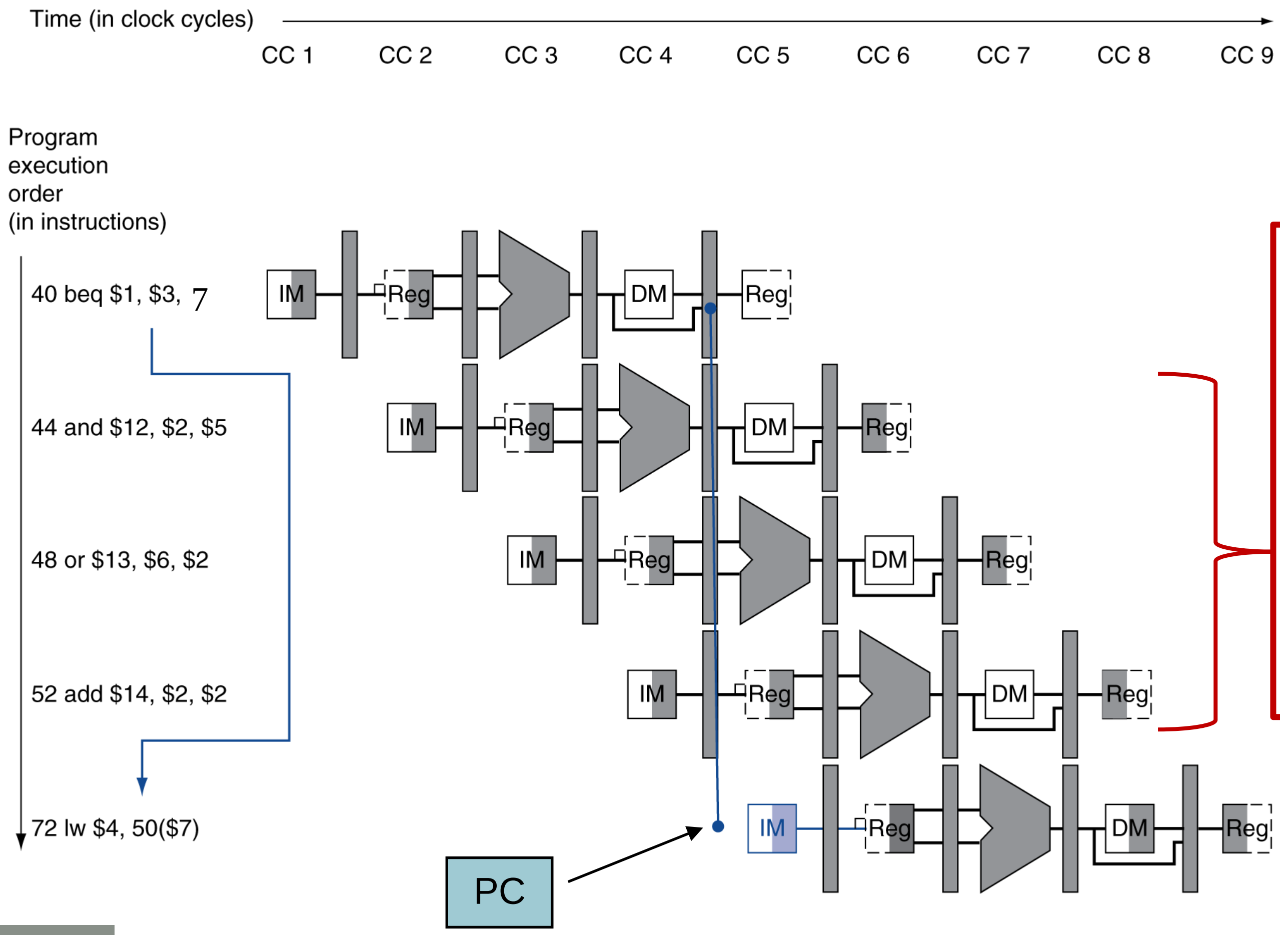


2) Branch Prediction

- Longer pipelines can’t readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

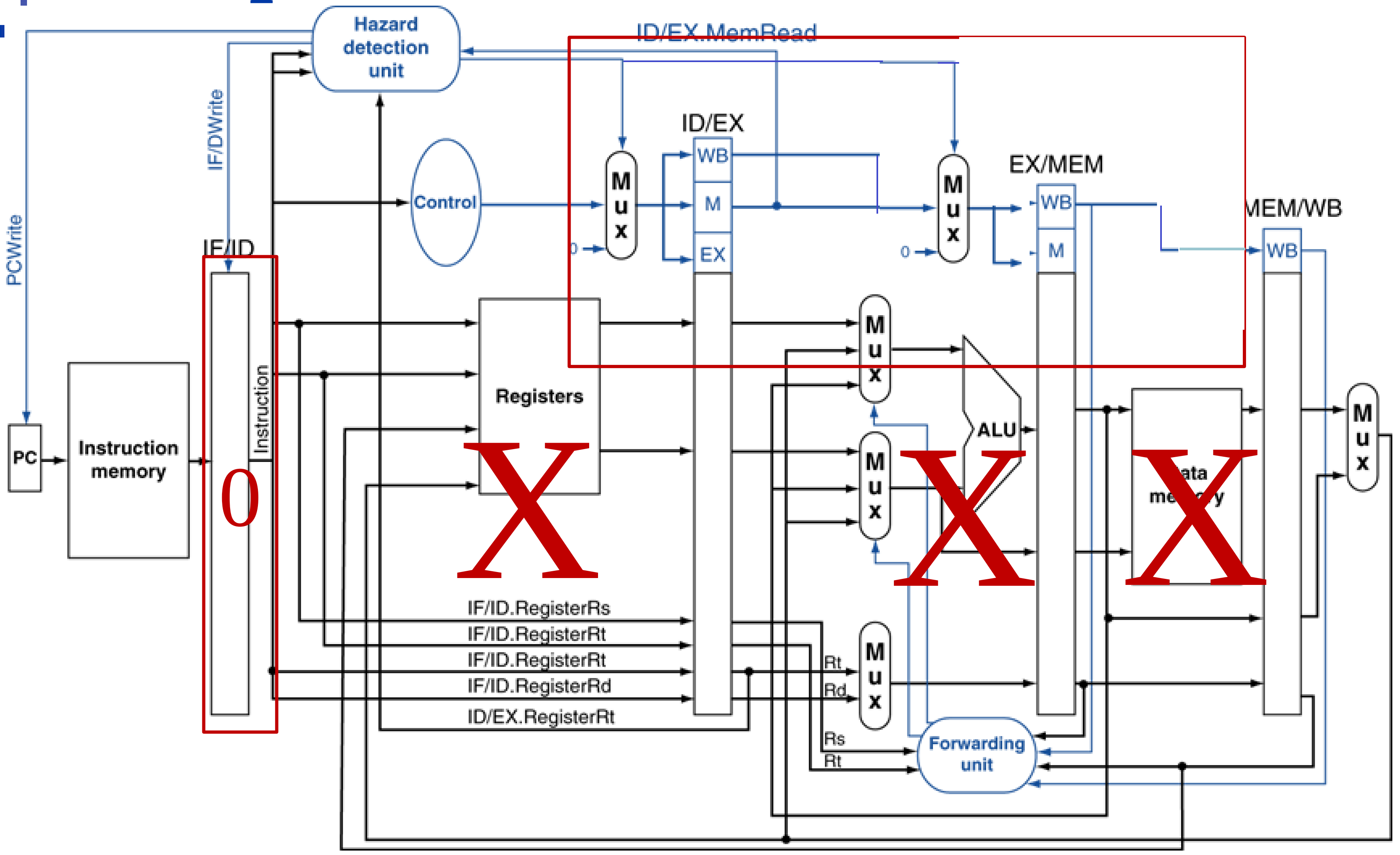
2) Branch Prediction: not Taken

- If branch outcome determined in MEM



Flush:
1) change the original control values to 0s (EX,MEM)
2) Set instruction register to 0 (ID-nop)

Datapath with Hazard Detection



3) Reducing Branch Delay

- We have assumed the next PC for a branch is selected in the MEM stage
- But if we move the branch execution earlier in the pipeline, then fewer instructions need be flushed
- In MIPS pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

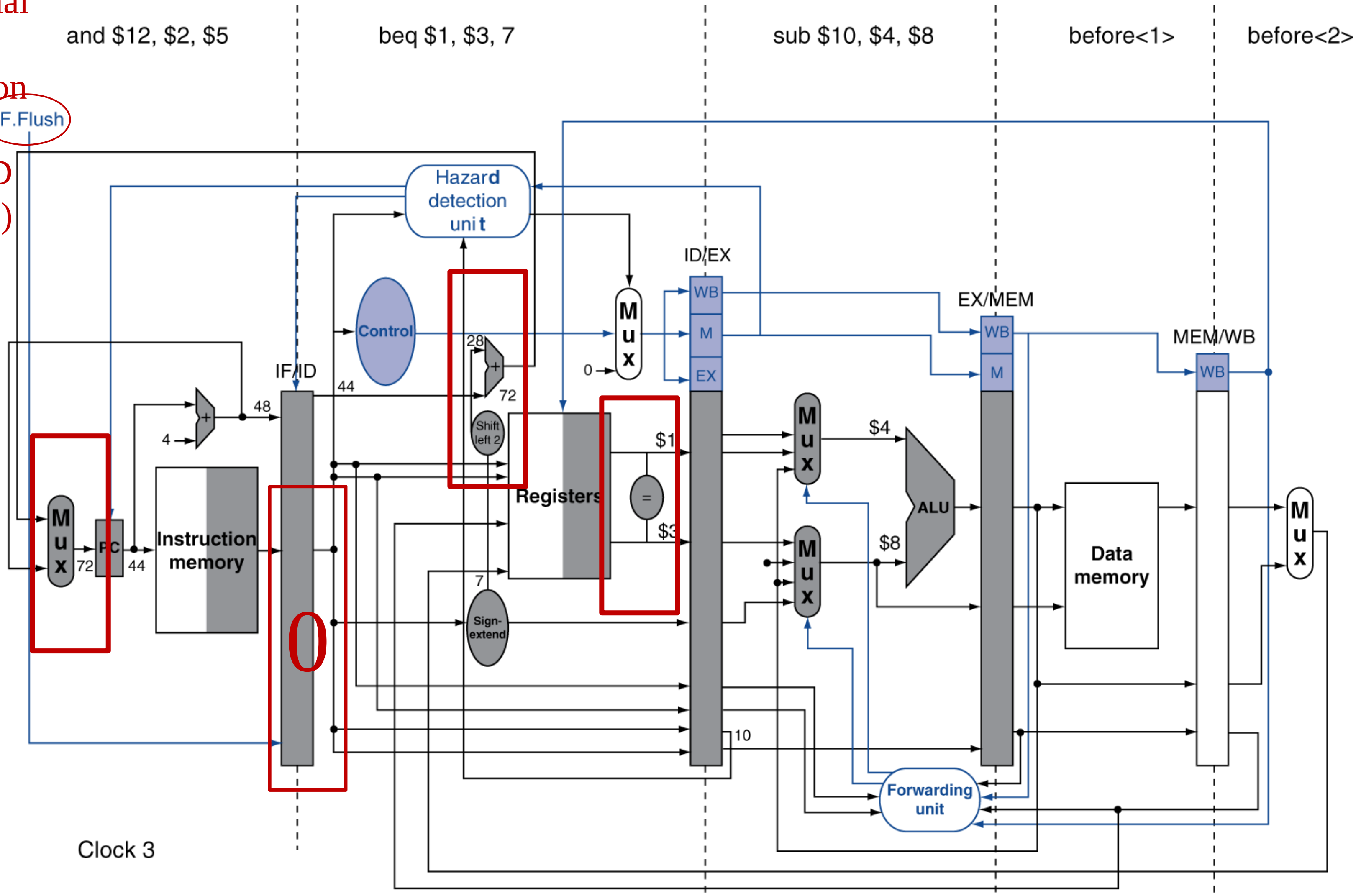
3) Reducing Branch Delay

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

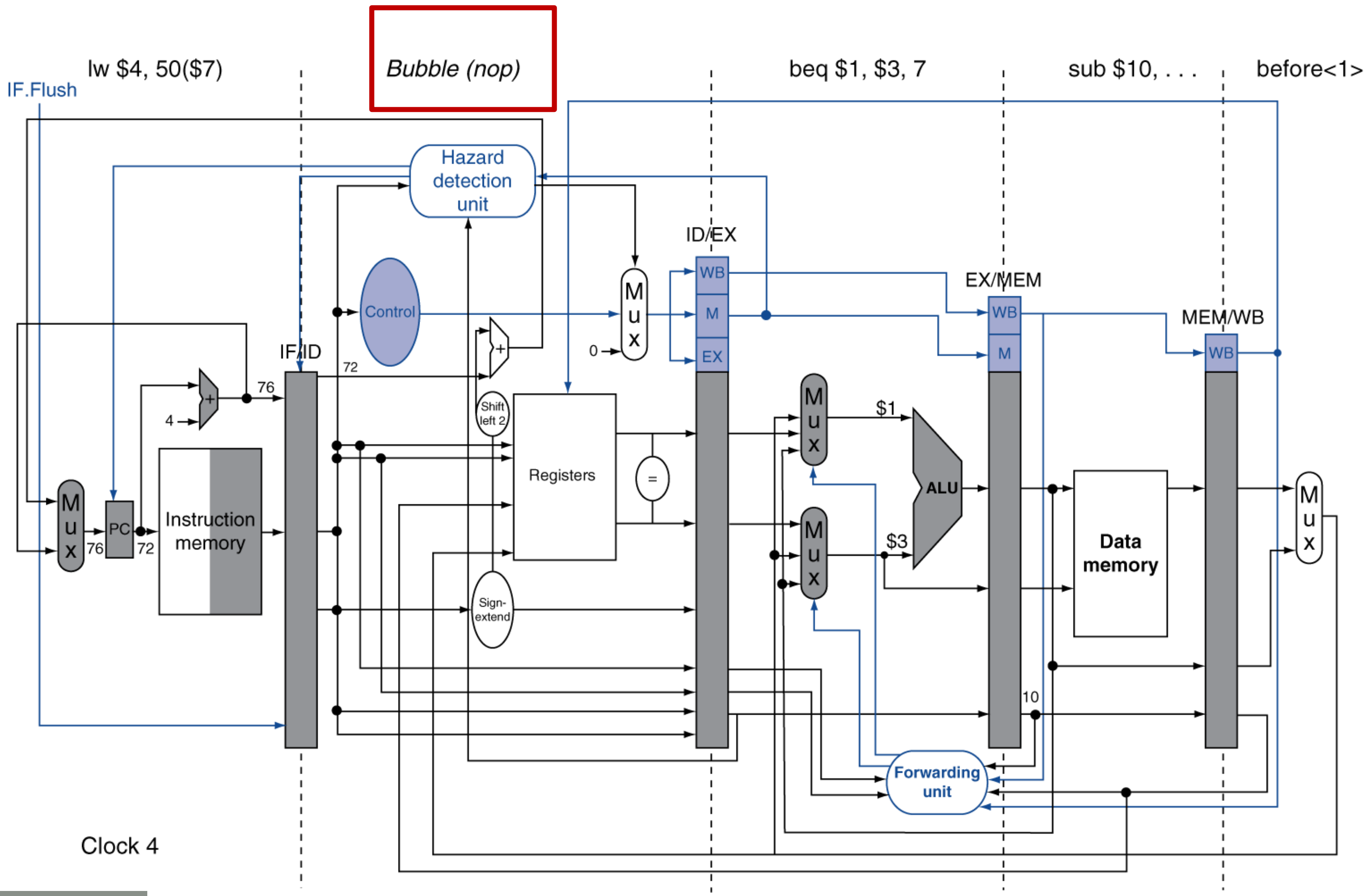
```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

Example: Branch Taken

This signal zeros Instruction field in IF/ID (flushes)



Example: Branch Taken



More-Realistic Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

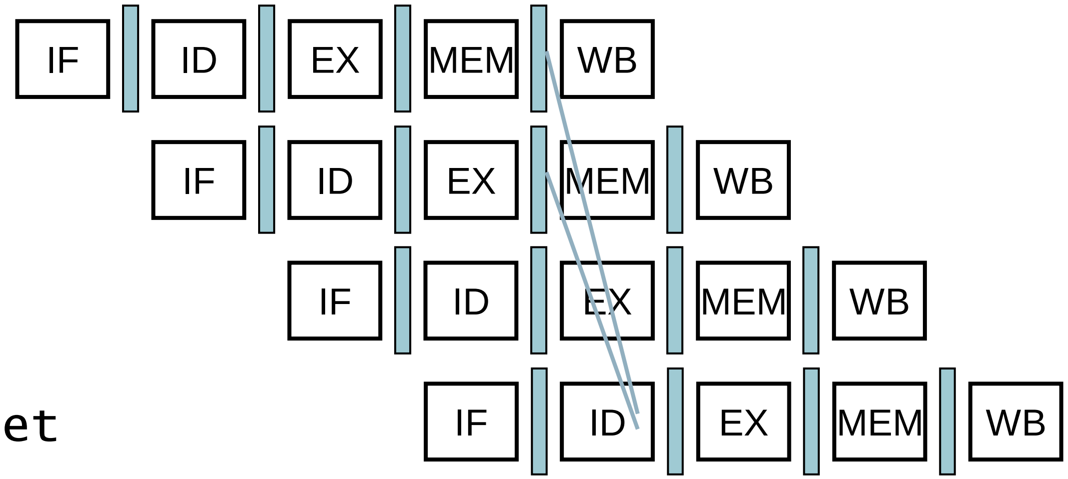
Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction:
    - 1-bit predictor vs 2-bit predictor

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
```

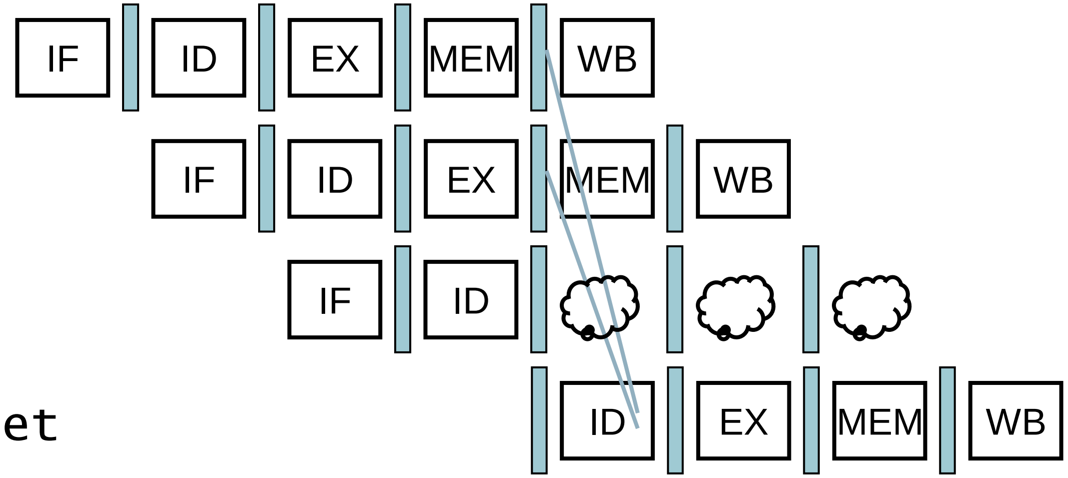


- Can resolve using forwarding

Data Hazards for Branches

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

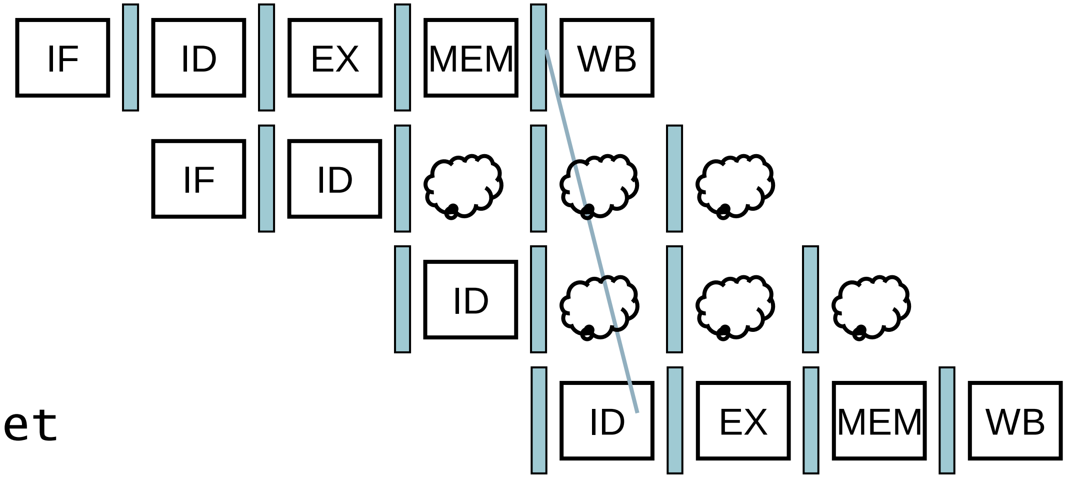
```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```



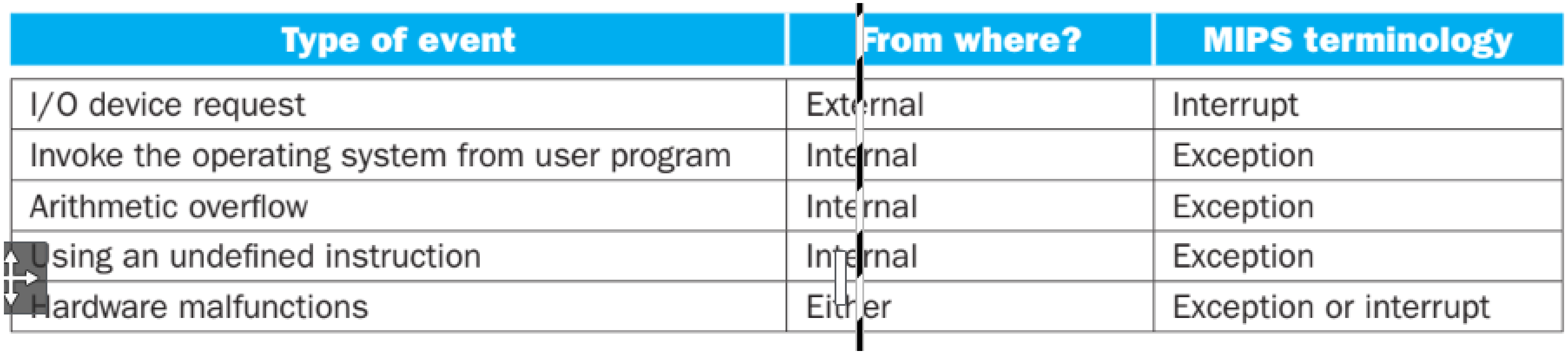
Pipeline Summary

The BIG Picture

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structural, data, control
- Instruction set design affects complexity of pipeline implementation

Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
- Exception: Arises within the CPU (undefined opcode, overflow)
- Interrupt: From an external I/O controller
- Dealing with them without sacrificing performance is hard



Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Handling Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)

1. Save PC of offending (or interrupted) instruction
   - In MIPS: Exception Program Counter (EPC)
2. Save indication of the problem
   - In MIPS: Cause register
   - We'll assume 1-bit: 0 for undefined opcode, 1 for overflow
3. Jump to handler at 8000 00180
4. If restartable
   - Take corrective action
   - use EPC to return to program

   Otherwise
   - Terminate program
   - Report error using EPC, cause, …

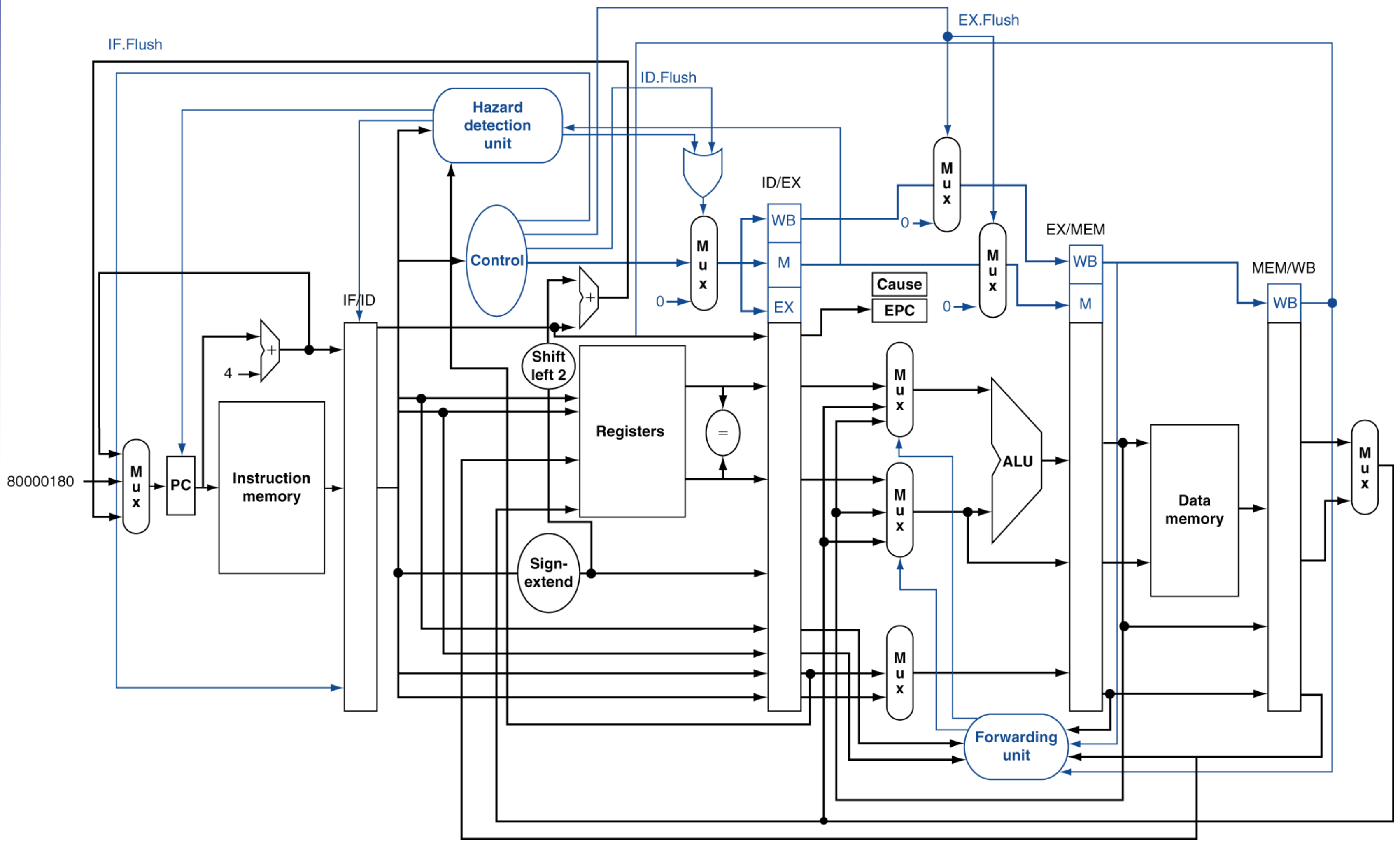
Exceptions in a Pipeline

- Another form of control hazard
- Consider overflow on add in EX stage

```
add $1, $2, $1
```

  - Prevent $1 from being clobbered
  - Complete previous instructions
  - Flush `add` and subsequent instructions
  - Set Cause and EPC register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

Pipeline with Exceptions



Exception Example

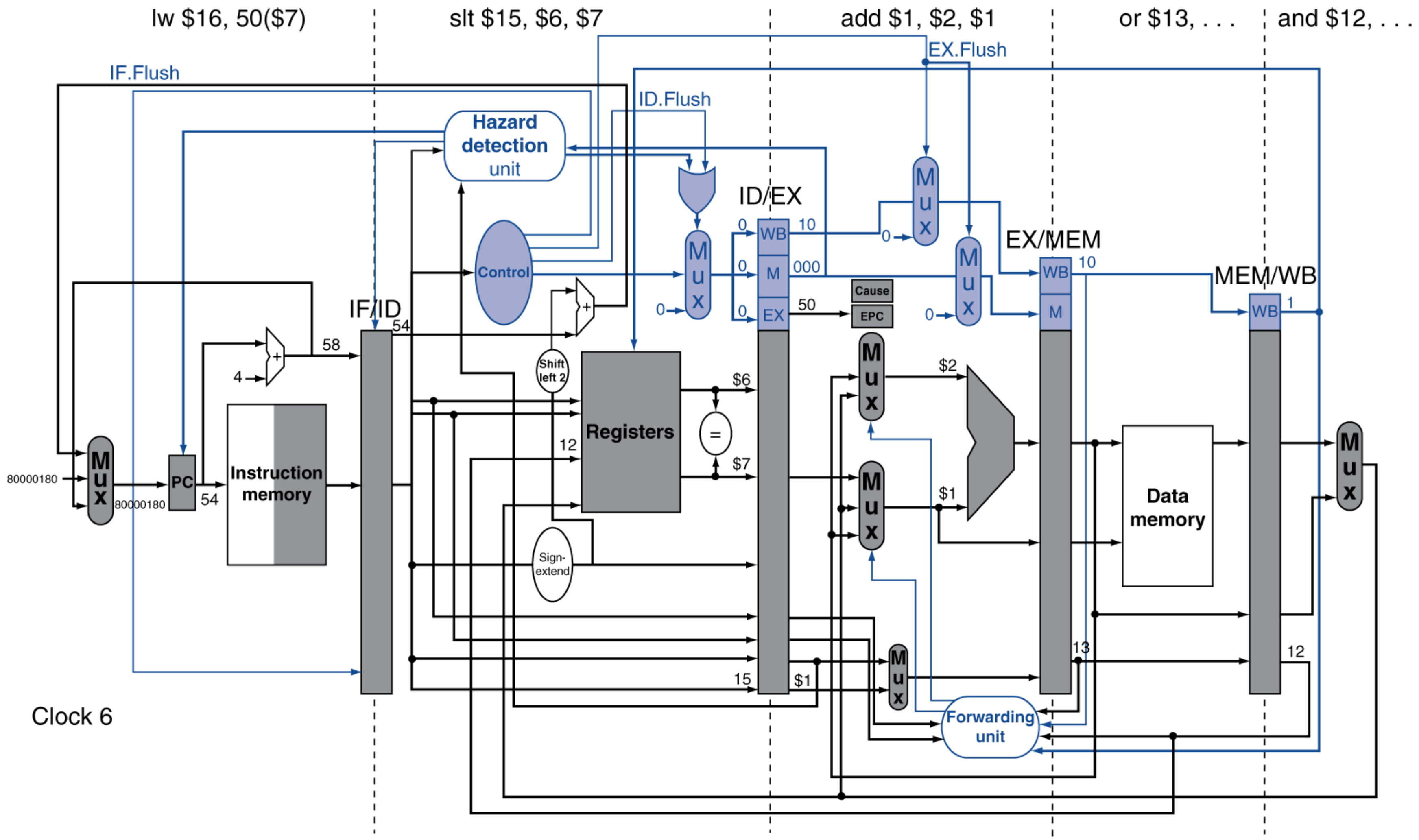
- Exception on add in

```
40      sub  $11, $2, $4
44      and  $12, $2, $5
48      or   $13, $2, $6
4c      add  $1,  $2, $1
50      slt  $15, $6, $7
54      lw   $16, 50($7)
…
```

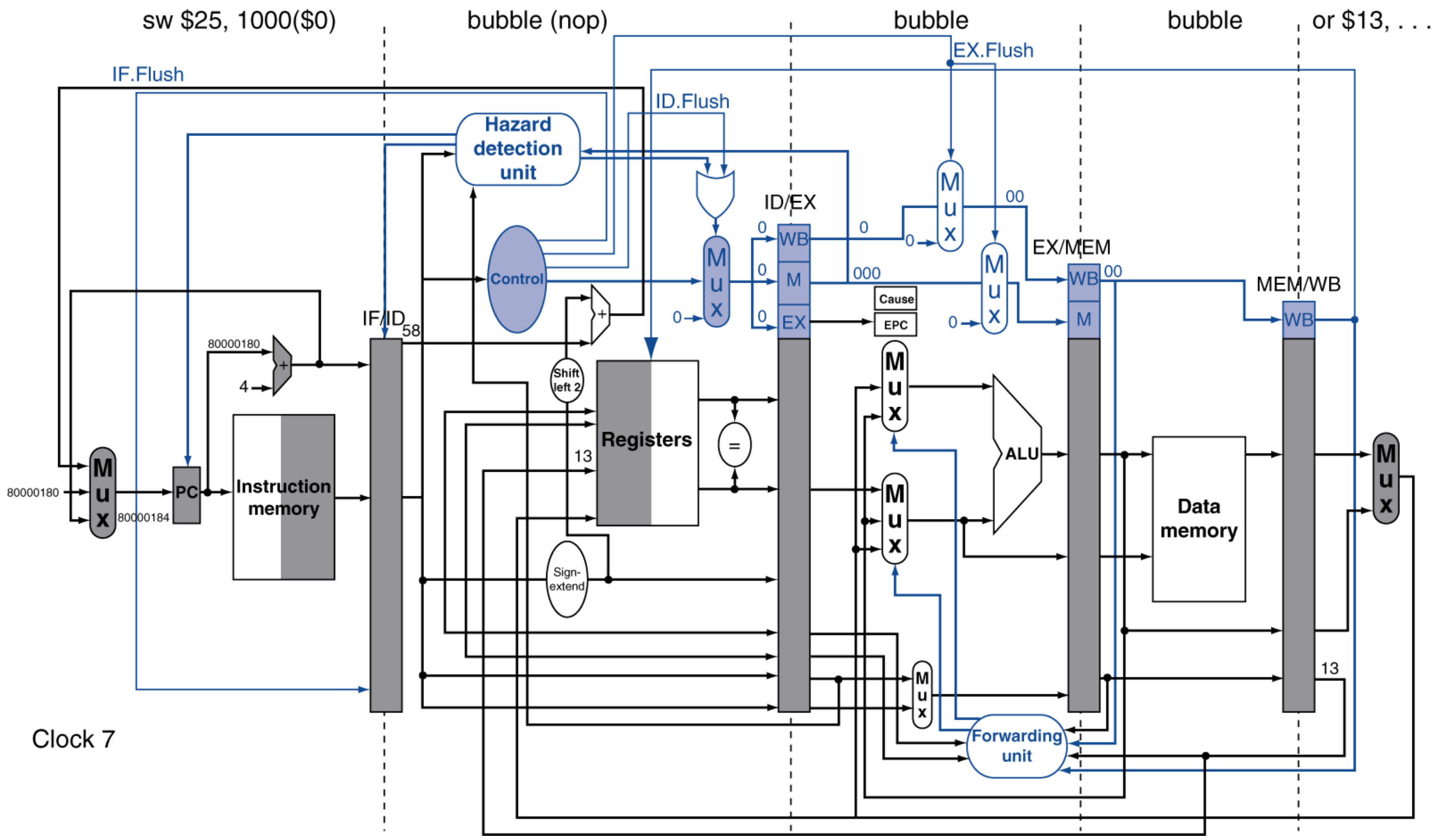
- Handler

```
80000180   sw   $25, 1000($0)
80000184   sw   $26, 1004($0)
…
```

Exception Example



Exception Example



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
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall

Problems to Solve

- 4.14, 4.15, 4.17