CMPE 200

COMPUTER ARCHITECTURE

Lecture 4 – The Processor – Pipelining

Bo Yu
Computer Engineering Department
SJSU

Adapted from Computer Organization and Design, 5th Edition, 4th edition, Patterson and Hennessy, MK
and Computer Architecture – A Quantitative Approach, 4th edition, Patterson and Hennessy, MK

Lecture 4B Key Concepts Review

- Last Lecture:
  - Pipelined Datapath and Control
  - Data Hazards: Forwarding vs. Stalling

Lecture 4B Key Concepts Review

- Five Stages Pipeline (instruction and datapath)
  - IF, ID, EX, MEM, WB
  - 4 Pipeline registers: IF/ID, ID/EX, EX/MEM, MEM/WB
  - At the end of each clock cycle, all instructions in the pipeline are moved to the next stage

- Two Pipeline Representations
  - Multi-Clock-Cycle pipeline diagram
    - Form showing physical resources uage
    - Form showing name of each stage
    - Time axis (horizontal): CC1, CC2, CC3, … and Instruction axis (vertical): inst1, inst2, inst3, …
    - A pipeline stage is placed along instruction axis, occupying the proper clock cycles
    - Overview of pipelining situation
  - Single-Clock-Cycle pipeline diagram
    - Shows state of the entire datapath during a single clock cycle
    - Represents a vertical slice through a set of multi-clock-cycle diagrams, showing the usage of datapath by each of instructions in the pipeline at the designated clock cycle

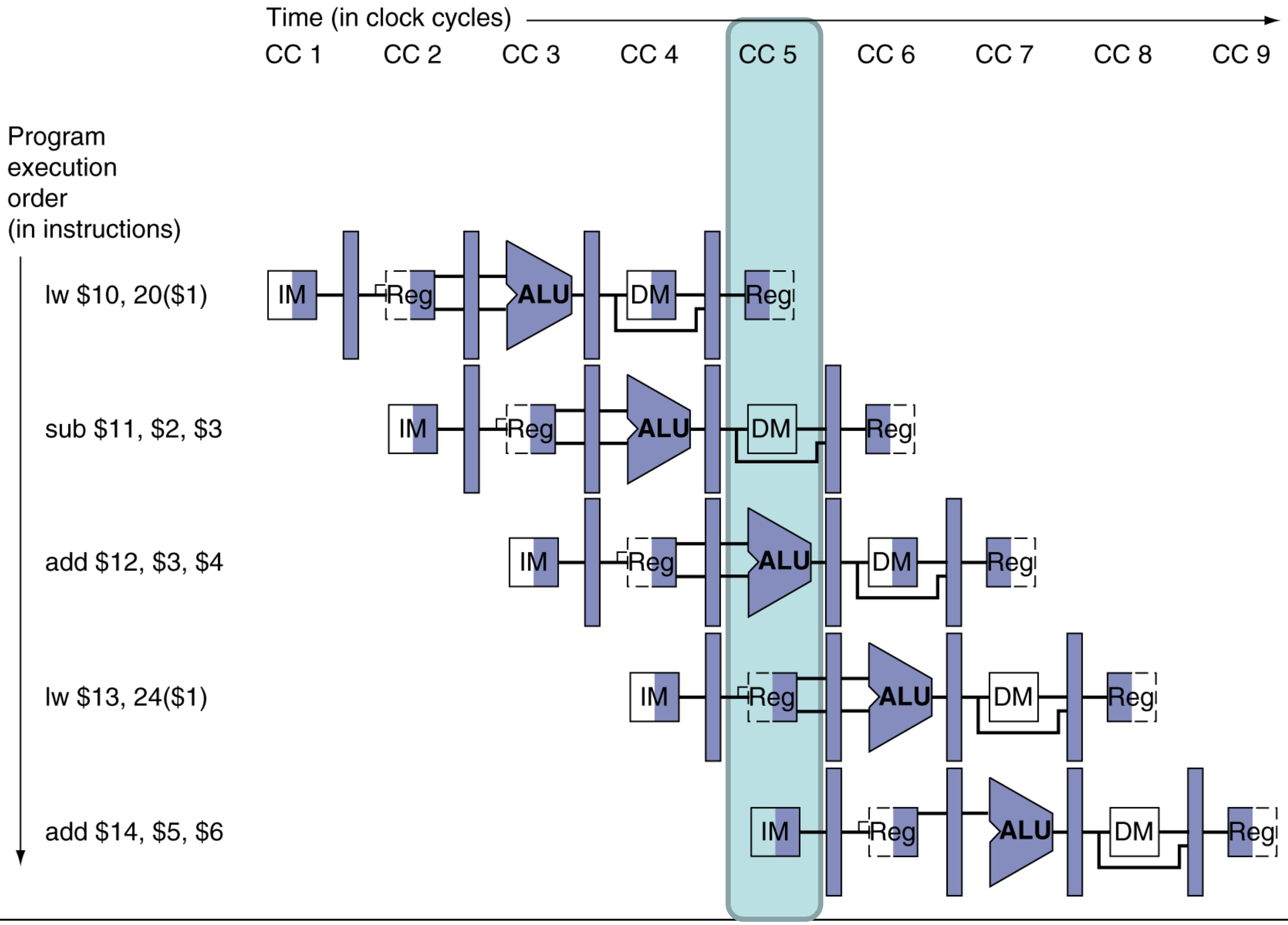
- Pipeline Control
  - 9 control lines grouped by three pipeline stages
    - Execution/address calculation stage control lines (4)
    - Memory access stage control lines (3)
    - Write-back stage control lines (2)
  - Control signals derived from instruction
    - 9 (4 EX, 3 MEM, 2 WB) pass to ID/EX pipeline register
    - 4 are used in EX stage, the rest 5 pass to EX/MEM pipeline register
    - 3 are used in MEM stage, the rest 2 pass to MEM/WB pipeline register
    - 2 are used in WB stage

- Data Hazards Detection
  - Forwarding Condition
    - EX hazard
    - MEM hazard
  - Load-Use Hazard Detection
    - Force control values in pipeline register to 0, EX, MEM, WB do nop
    - Stalls reduce performance but are required to get correct results

Lecture 4B: The Processor - Pipelining

Multi-Cycle Pipeline Diagram

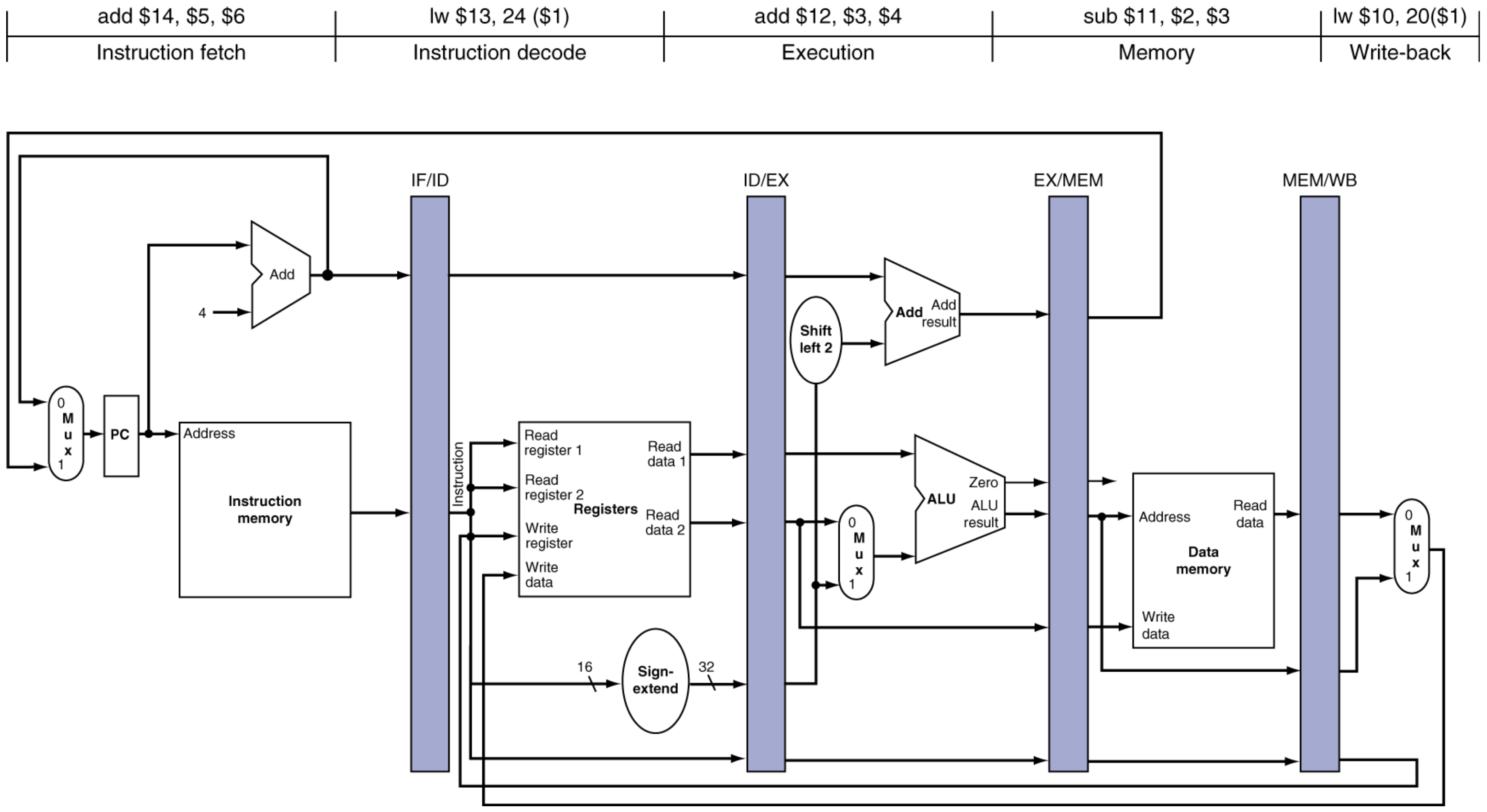
- Form showing physical resources usage



Lecture 4B: The Processor - Pipelining

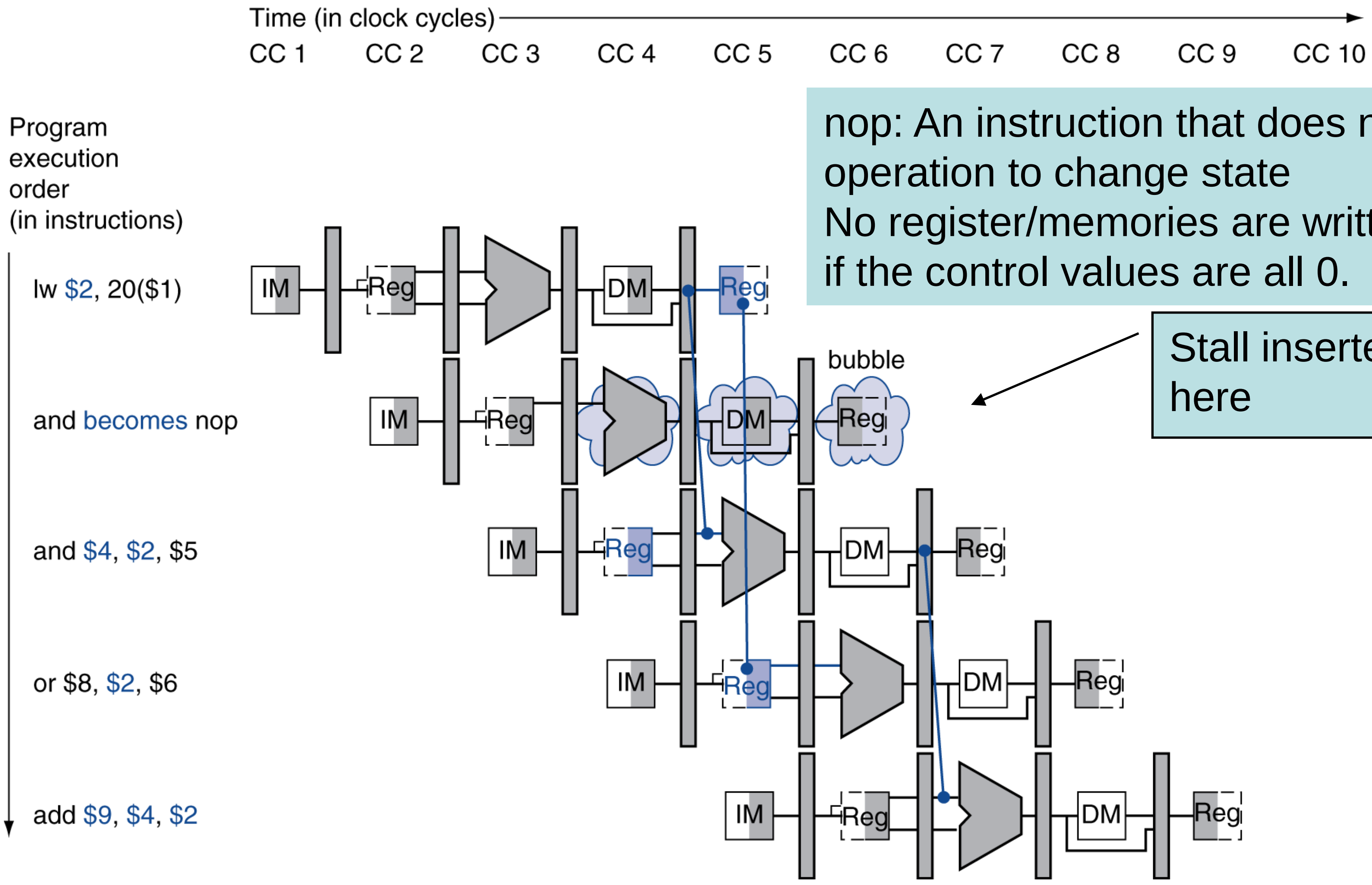
■ Single-Cycle Pipeline Diagram

- State of pipeline in a given cycle



Lecture 4B: The Processor - Pipelining

■ Stall/Bubble in the Pipelined

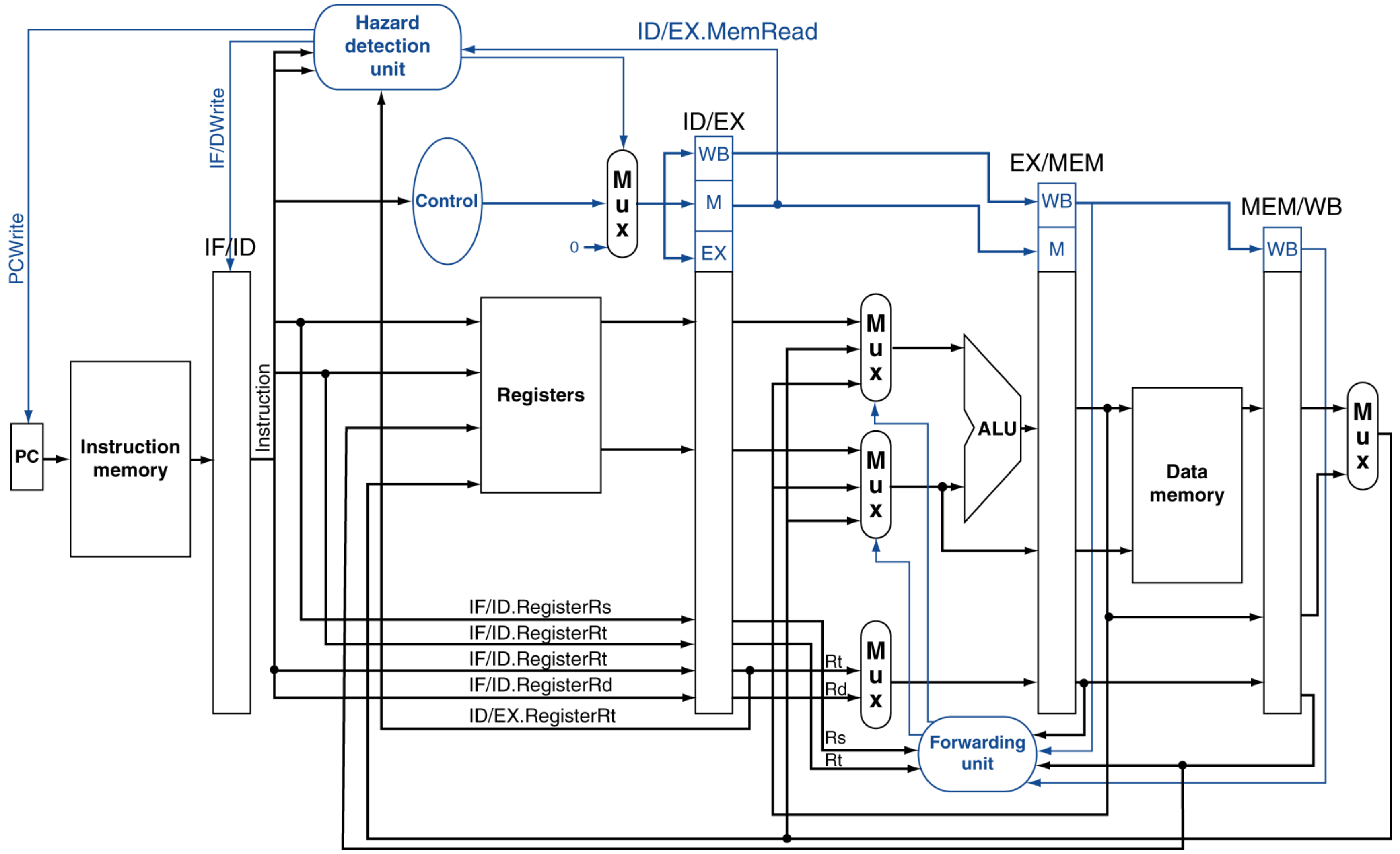


nop: An instruction that does no operation to change state
No register/memories are written if the control values are all 0.

Stall inserted here

Lecture 4B: The Processor - Pipelining

■ Datapath with Hazard Detection

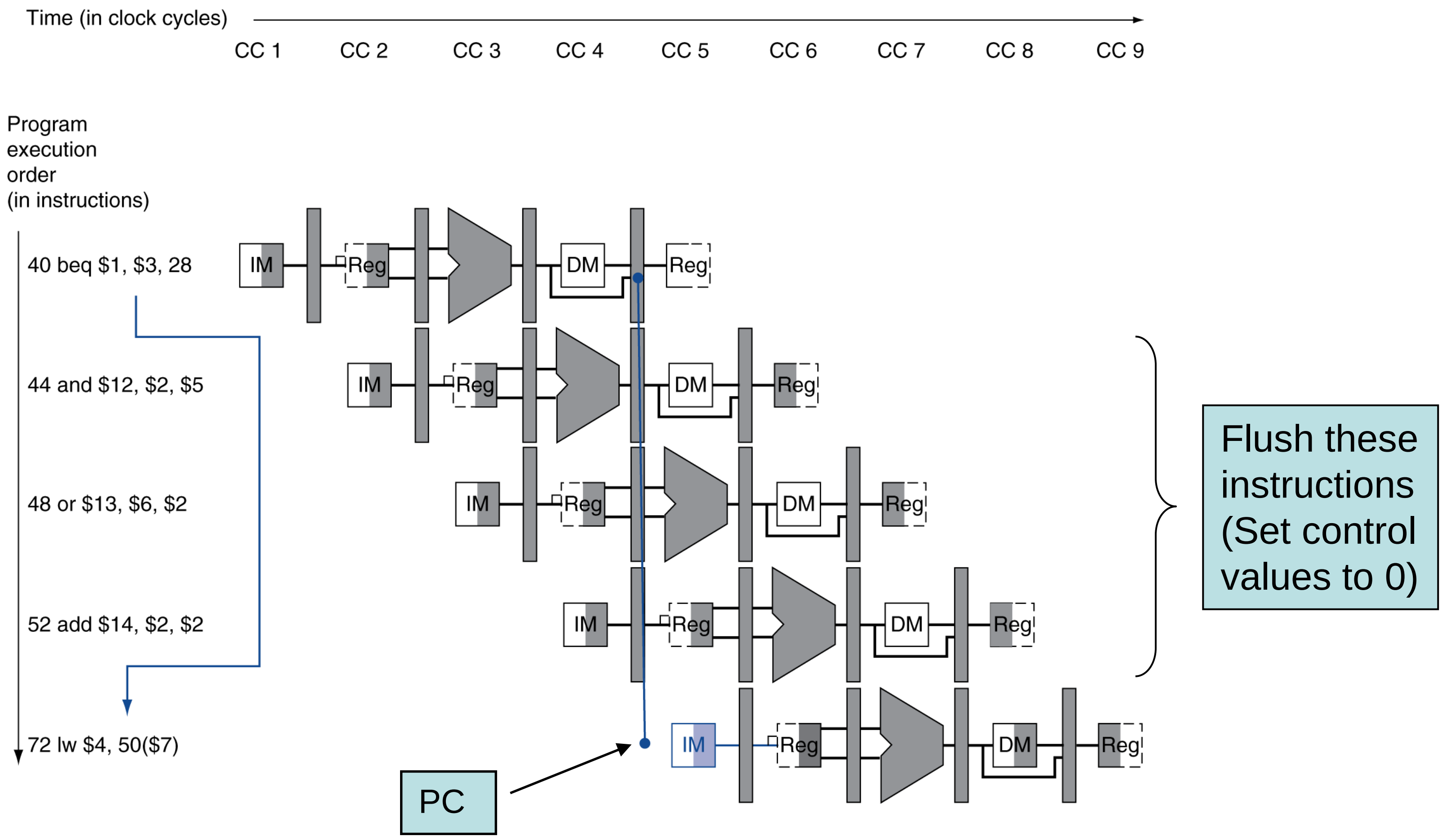


Lecture 4C: The Processor - Pipelining

- Today's Lecture (CH4.8-4.9):
  - Control Hazards
  - Exceptions

Branch Hazards

- If branch outcome determined in MEM

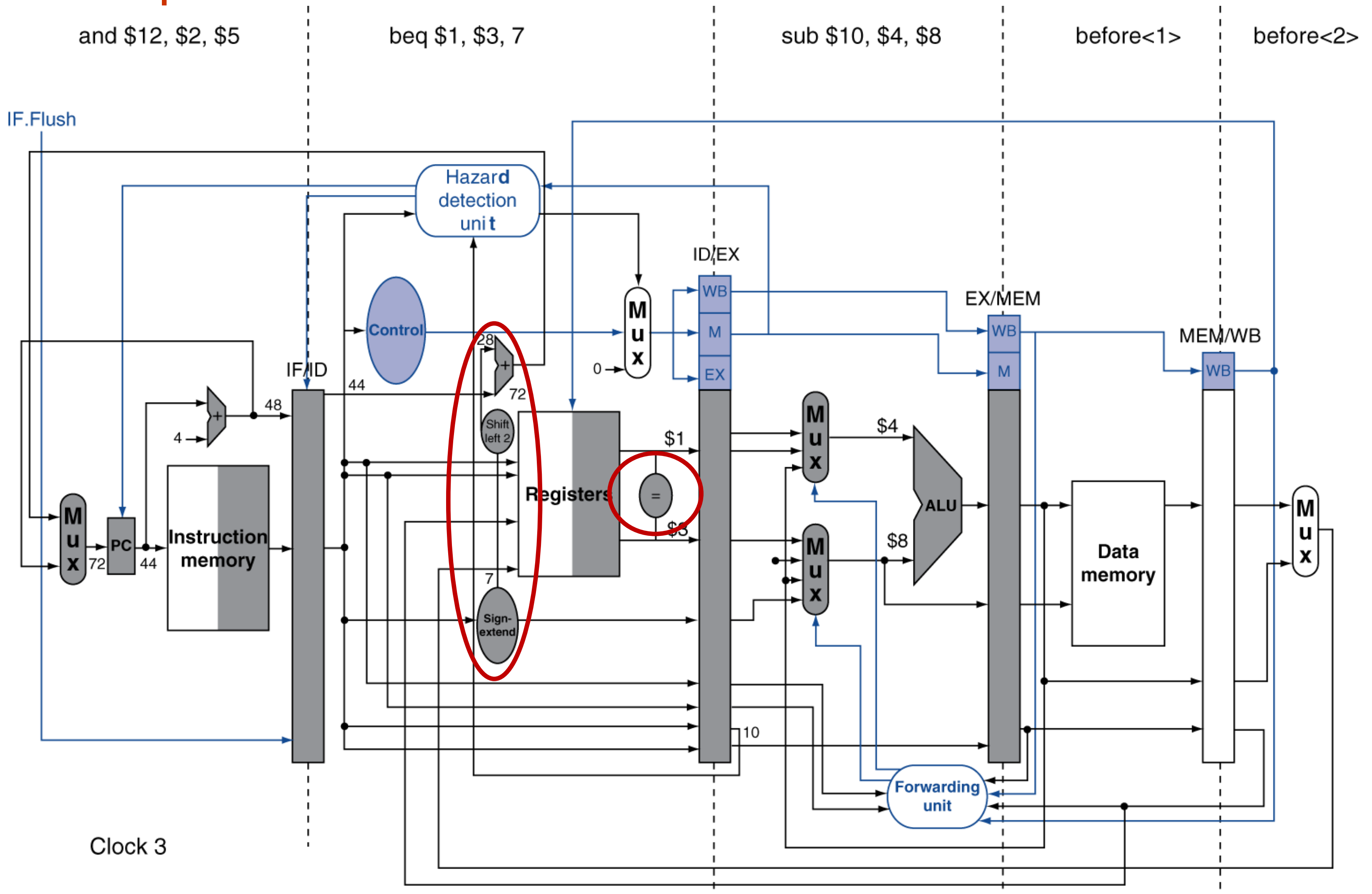


Lecture 4: The Processor - Pipelining

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

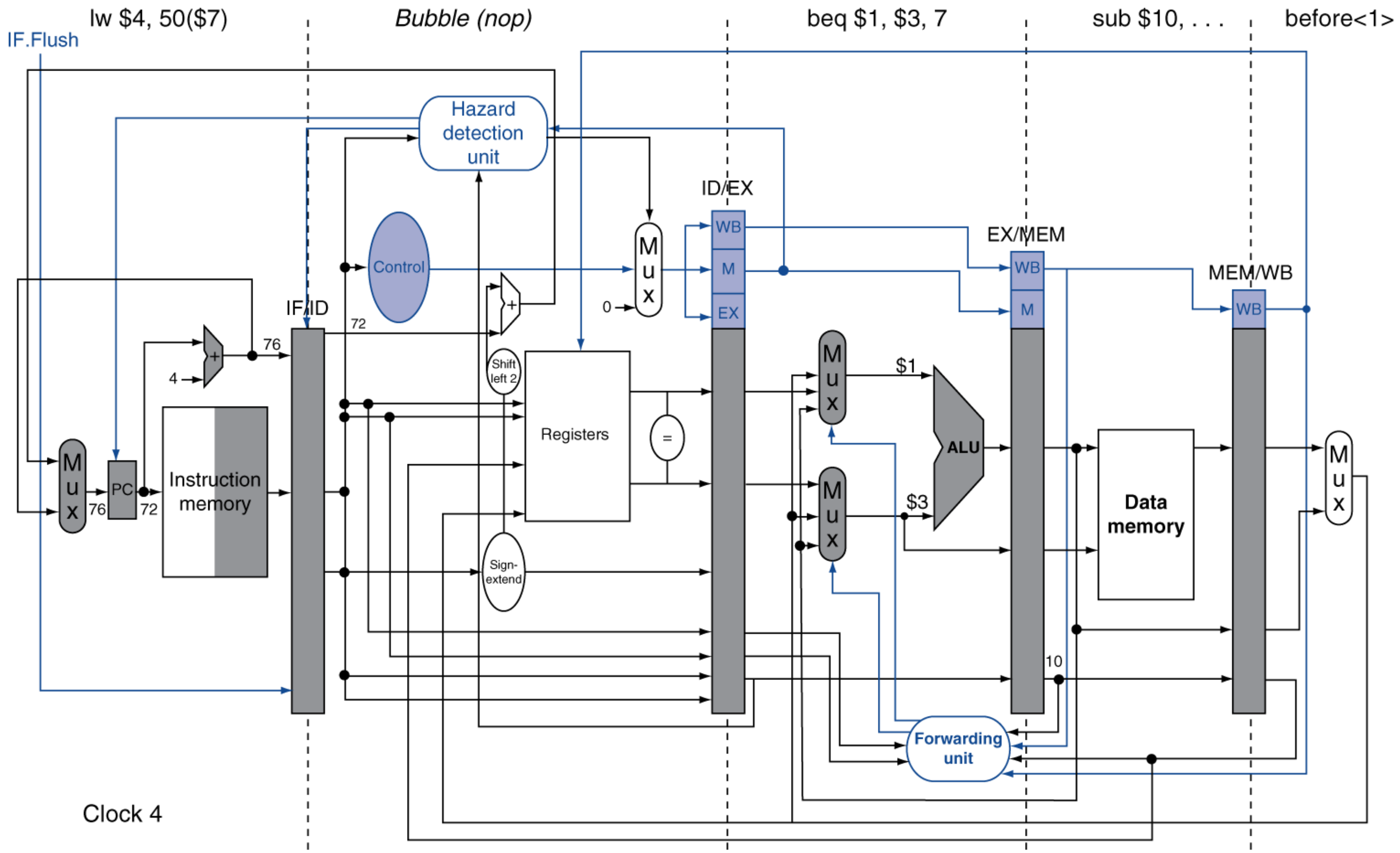
```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

- Example: Branch Taken



Lecture 4: The Processor - Pipelining

- Example: Branch Taken

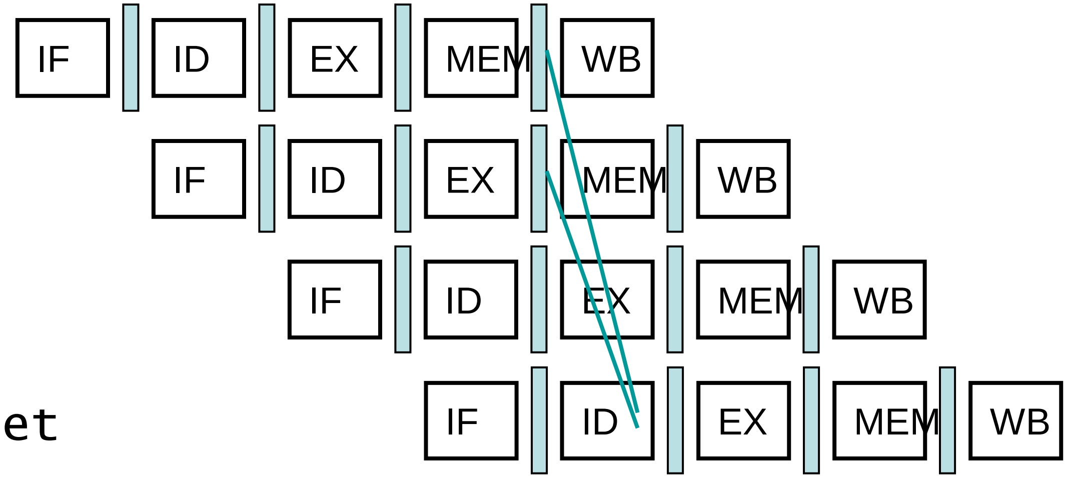


nop – an instruction that has no action and changes no state

Lecture 4: The Processor - Pipelining

- 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
```

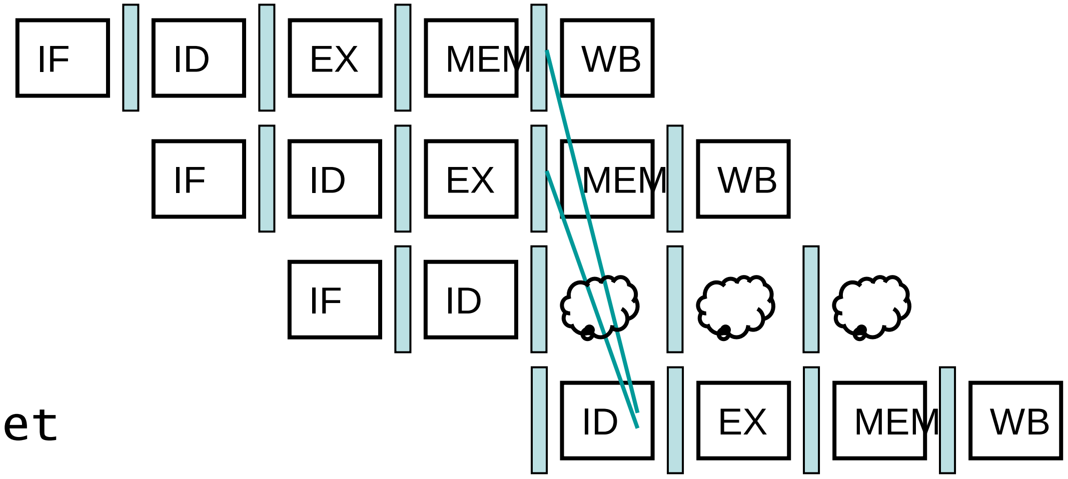


  - Can resolve using forwarding

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

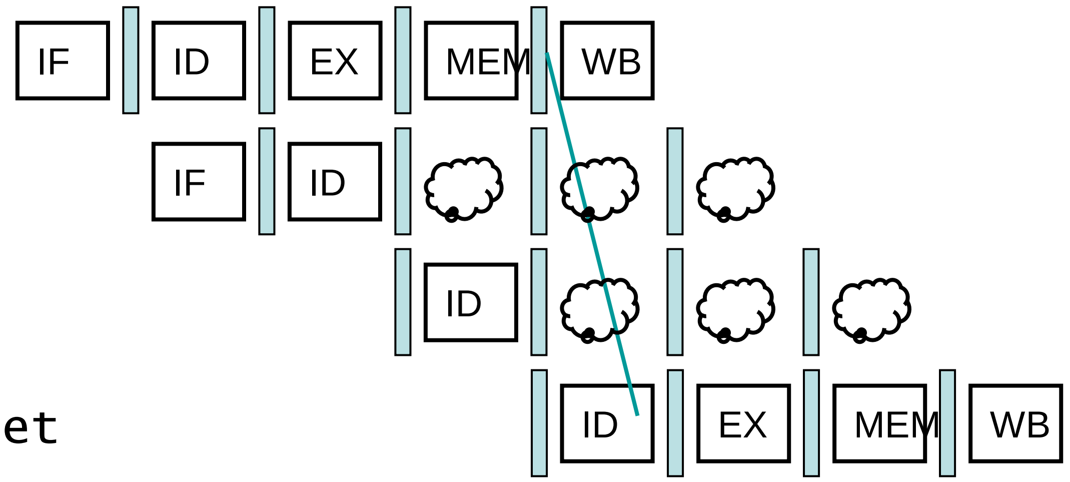
- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw  $1, addr

beq stalled

beq stalled

beq $1, $0, target
```

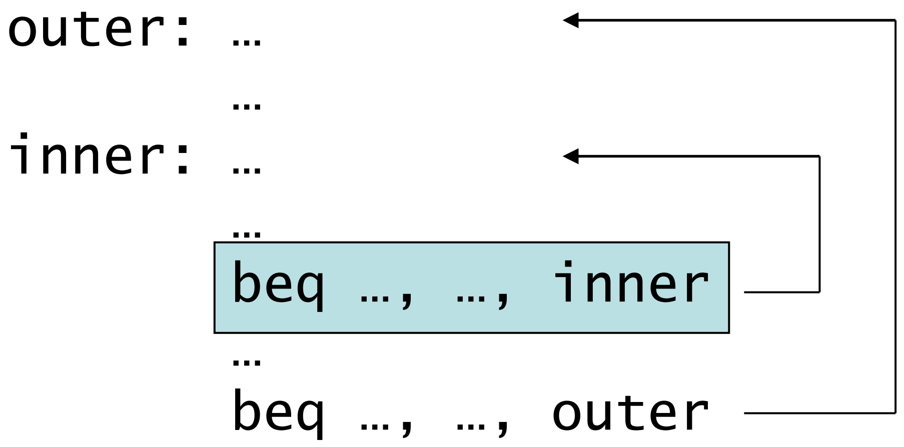


Dynamic Branch Prediction

- 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
    - 1-bit memory stores outcome (recently 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

Lecture 4: The Processor - Pipelining

- 1-Bit Predictor: Shortcoming
  - Inner loop branches mispredicted twice!



```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

Lecture 4: The Processor - Pipelining

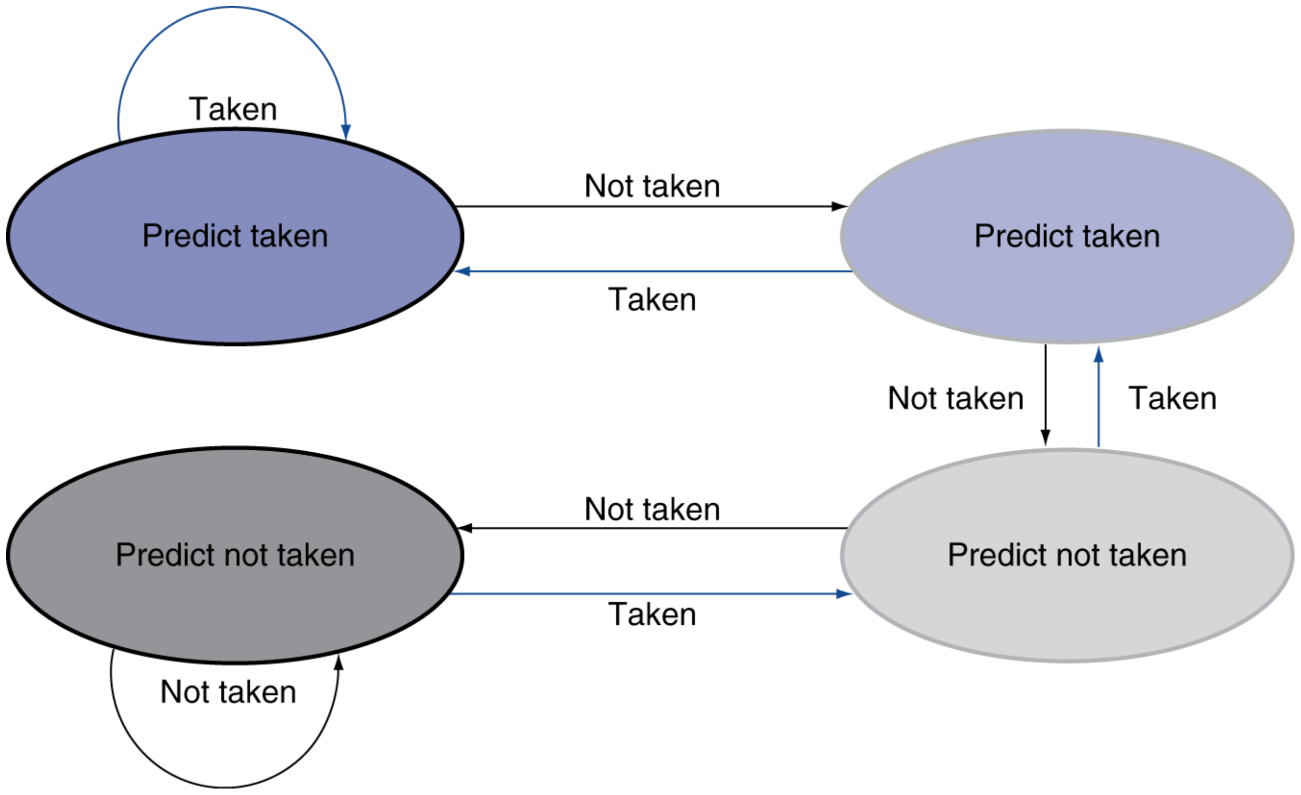
1-Bit Predictor: Example

- A loop branch that branches 9 times in a row, then is not taken once. What is the prediction accuracy for this branch, assuming the prediction bit for this branch remains in the prediction buffer?

  - Mispredicting as taken on last iteration of inner loop is inevitable as the branch has been taken 9 times in a row.
  - Then misprediction as not taken on first iteration happens because the bit is flipped on prior execution of the last iteration of the loop since the branch was not taken on that exiting iteration.
  - So the prediction accuracy for this branch: taken 90% of time, only 80% accuracy (2 incorrect predictions and 8 correct ones).

Lecture 4: The Processor - Pipelining

- 2-Bit Predictor
  - Only change prediction on two successive mispredictions – Will only mispredict once



Lecture 4: The Processor - Pipelining

- Calculating the Branch Target
  - Even with predictor, still need to calculate the target address
    - 1-cycle penalty for a taken branch
  - Branch target buffer
    - Cache of target addresses
    - Indexed by PC when instruction fetched
      - If hit and instruction is branch predicted taken, can fetch target immediately

Lecture 4C: The Processor - Pipelining

- Polling

  Consider three branch prediction schemes: predict not taken, predict taken, and dynamic prediction. Assume that they all have zero penalty when they predict correctly and two cycles penalty when they are wrong. Assume average predict accuracy of the dynamic predictor is 90%. Which predictor is the best choices for the following branches?

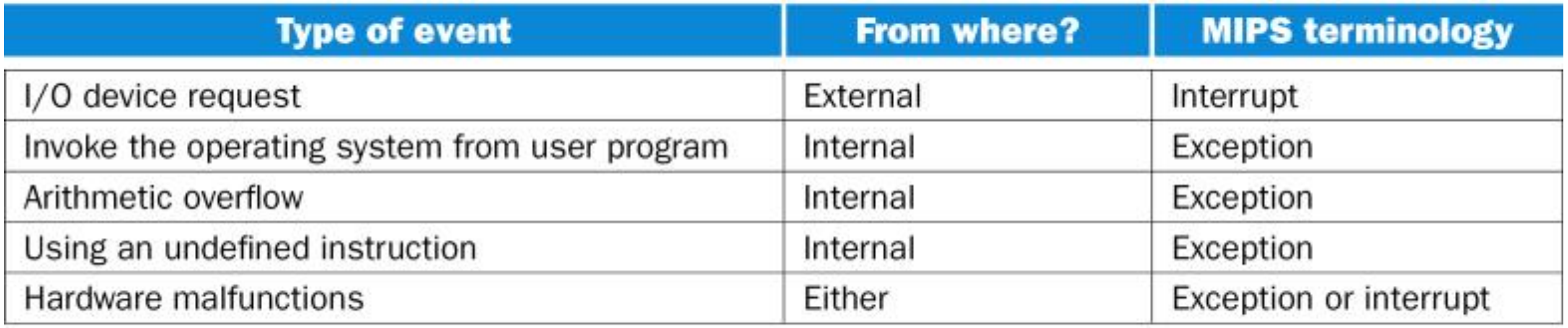
  1. A branch that is taken with 5% frequency
  2. A branch that is taken with 95% frequency
  3. A branch that is taken with 70% frequency

Exceptions and Interrupts

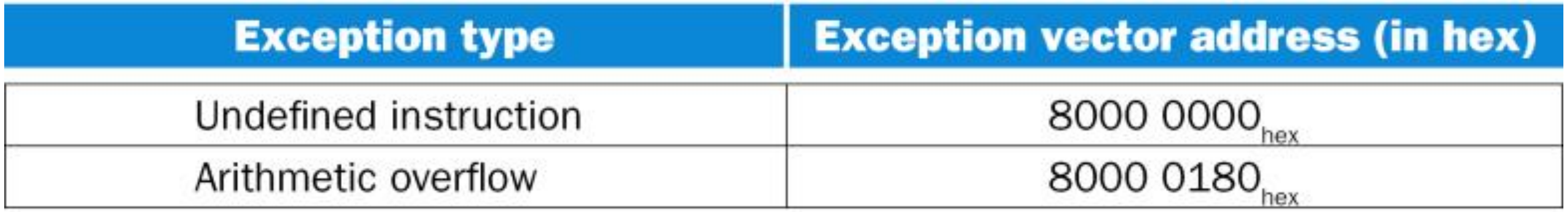
- "Unexpected" events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, overflow, syscall, …
- Interrupt
  - From an external I/O controller
- Dealing with them without sacrificing performance is hard

Lecture 4: The Processor - Pipelining

- Examples



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



Exception type	Exception vector address (in hex)
Undefined instruction	$8000\ 0000_{hex}$
Arithmetic overflow	$8000\ 0180_{hex}$

Lecture 4: The Processor - Pipelining

- Handling Exceptions
  - In MIPS, exceptions managed by a System Control Coprocessor (CP0)
  - Save PC of offending (or interrupted) instruction
    - In MIPS: Exception Program Counter (EPC)
  - Save indication of the problem
    - In MIPS: Cause register
    - We'll assume 1-bit
      - 0 for undefined opcode, 1 for overflow
  - Jump to handler at 8000 00180

An Alternate Mechanism

- Vectored Interrupts
  - Handler address determined by the cause
- Example:

– Undefined opcode:	C000 0000
– Overflow:	C000 0020
– ….:	C000 0040

- Instructions either
  - Deal with the interrupt, or
  - Jump to real handler

Lecture 4: The Processor - Pipelining

- Handler Actions
  - Read cause, and transfer to relevant handler
  - Determine action required
  - If restartable
    - Take corrective action
    - use EPC to return to program
  - Otherwise
    - Terminate program
    - Report error using EPC, cause, …

Exceptions in a Pipeline

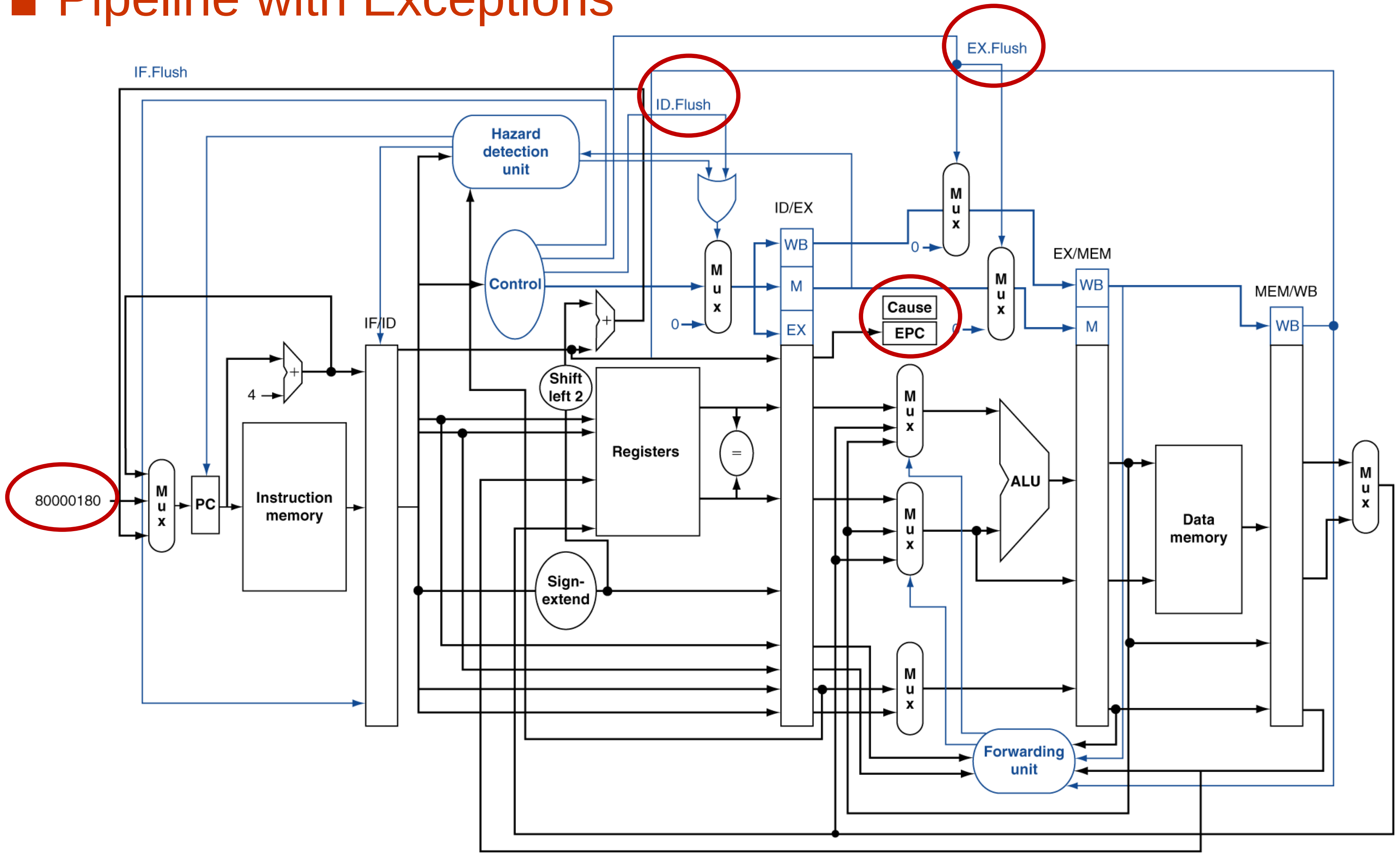
- Another form of control hazard
- Consider overflow on add in EX stage

  ```
  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

Lecture 4: The Processor - Pipelining

- Pipeline with Exceptions



Exception Properties

- Restartable exceptions
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Refetched and executed from scratch
- PC saved in EPC register
  - Identifies causing instruction
  - Actually PC + 4 is saved
    - Handler must adjust

Exception Example

- Exception on add in

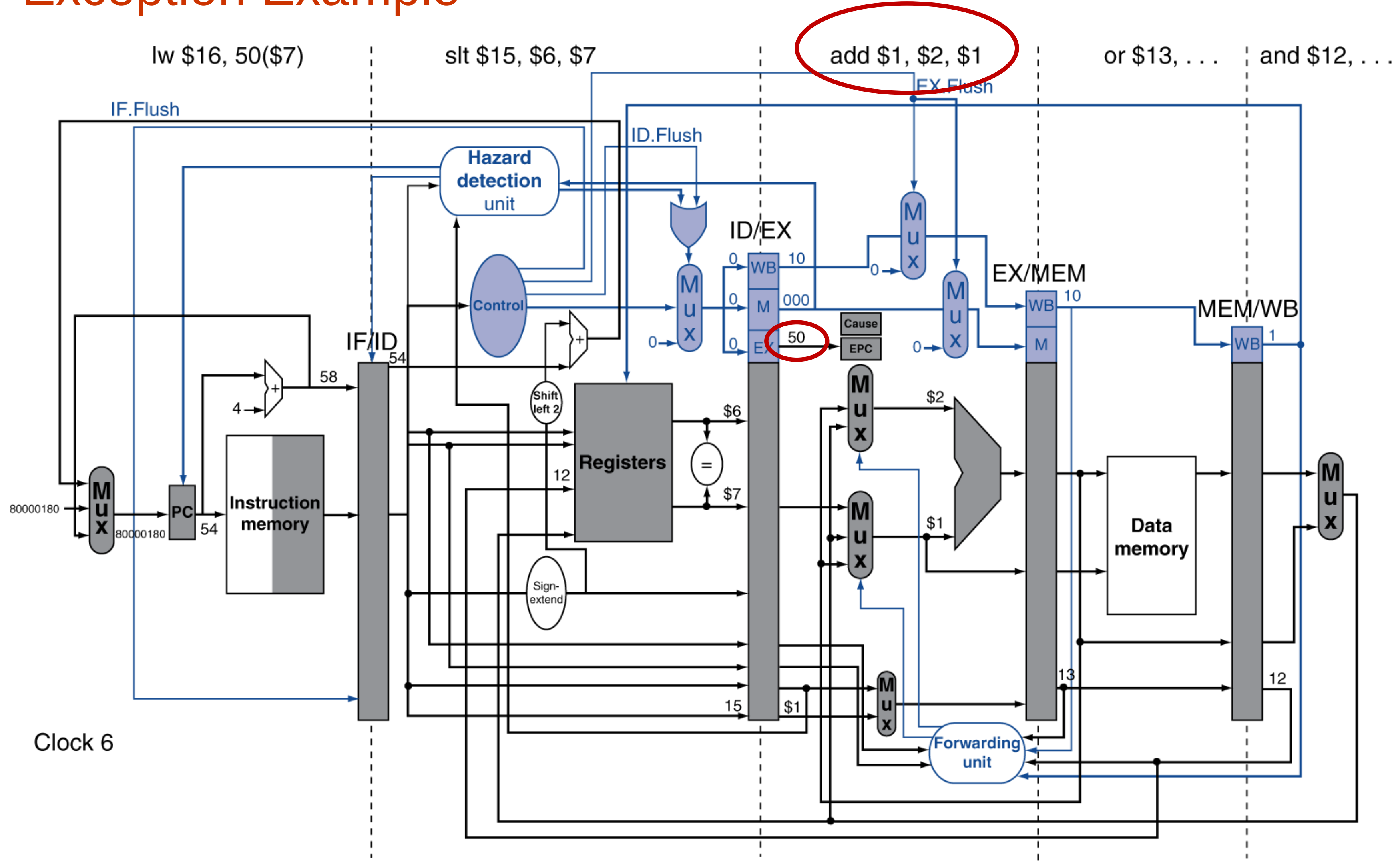
```
40      sub   $11, $2, $4
44      and   $12, $2, $5
48      or    $13, $2, $6
4C      add   $1,  $2, $1
50      slt   $15, $6, $7
54      lw    $16, 50($7)
…
```

- Handler

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

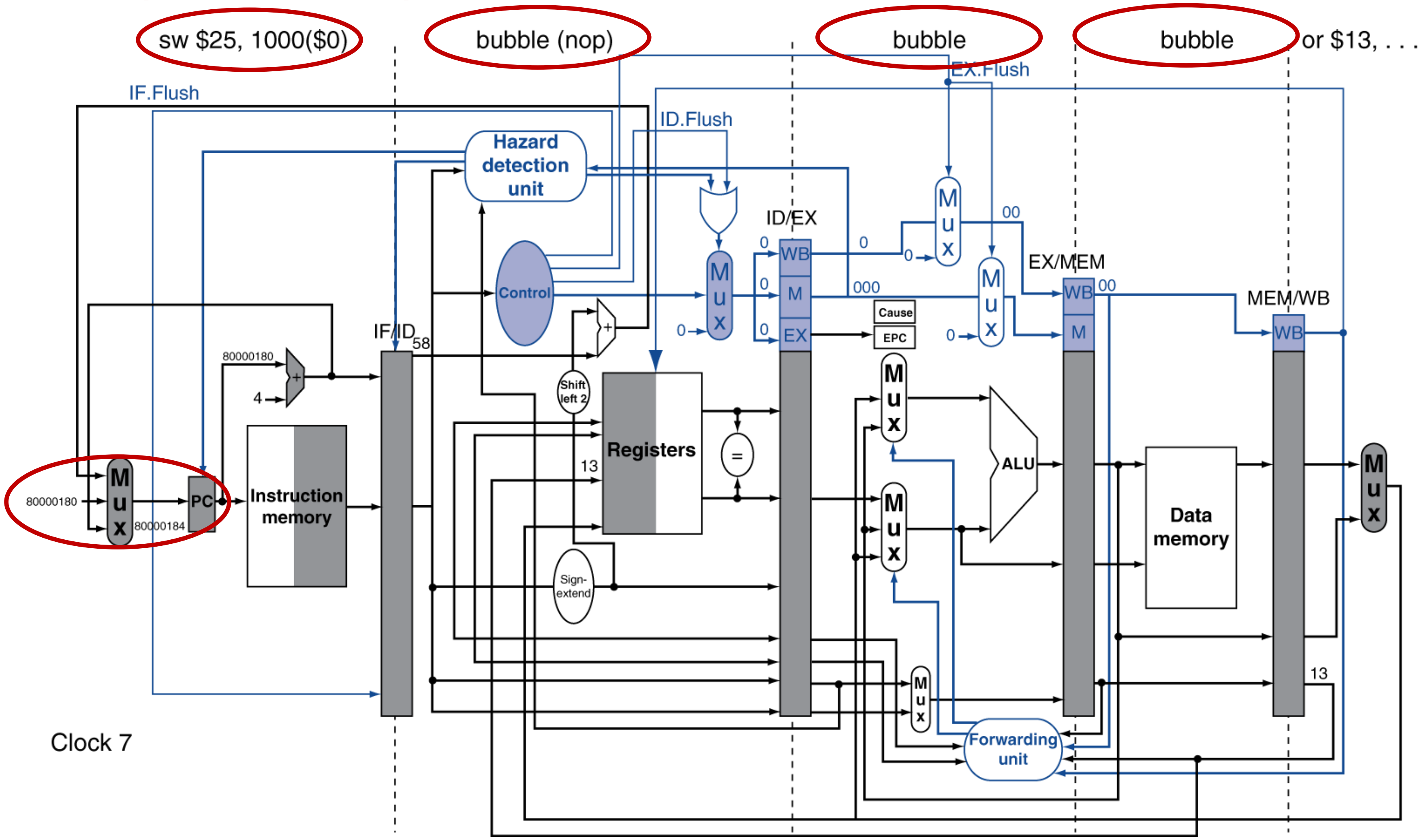
Lecture 4: The Processor - Pipelining

- Exception Example



Lecture 4: The Processor - Pipelining

- Exception Example



Lecture 4: The Processor - Pipelining

- Multiple Exceptions
  - Pipelining overlaps multiple instructions
    - Could have multiple exceptions at once
  - Simple approach: deal with exception from earliest instruction
    - Flush subsequent instructions
    - “Precise” exceptions
  - In complex pipelines
    - Multiple instructions issued per cycle
    - Out-of-order completion
    - Maintaining precise exceptions is difficult!

Lecture 4: The Processor - Pipelining

- Imprecise Exceptions
  - Just stop pipeline and save state
    - Including exception cause(s)
  - Let the handler work out
    - Which instruction(s) had exceptions
    - Which to complete or flush
      - May require “manual” completion
  - Simplifies hardware, but more complex handler software
  - Not feasible for complex multiple-issue out-of-order pipelines

Lecture 4: The Processor - Pipelining

- Next lecture
  - ILP (Instruction Level Parallelism)