

CSSE232

# Computer Architecture I

Exceptions in Pipelines

# Outline

- Review: exceptions and interrupts
- Exceptions in multicycle datapaths
- Exceptions in pipelined

# What are 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

# How do we Handle Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CPO)
- Save PC of offending (or interrupted) instruction
  - In MIPS: Exception Program Counter (EPC)
- Save indication of the problem
  - In MIPS: Cause register
- Jump to handler at 8000 00180
- Status[1] = 1

# What is the difference between exception level and interrupt enable?

- Interrupt enable – turns on and off interrupts
- Exception level – when its 1 exception is being handled. Further interrupts are disabled (exception level overrides interrupt enable)

# Types of exceptions

- We will look at 2 exceptions
  - Undefined instruction
  - Overflow
- Multi-cycle first
- Where do these happen?
- What can cause them?

# Multicycle RTL

Step	R-Type	lw/sw	beq/bne	j
IF	$IR = Mem[PC]$ $PC = PC + 4$			
ID	$A = Reg[IR[25-21]]$ $B = Reg[IR[20-16]]$ $ALUOut = PC + (SE(IR[15-0]) \ll 2)$			
EX	$ALUOut = A \text{ op } B$	$ALUOut =$ $A + SE(IR[15-0])$	If $(A == B)$ then $PC = ALUOut$	$PC = PC[31-28]$ $  $ $(IR[25-0] \ll 2)$
MEM	$Reg[IR[15-11]] =$ $ALUOut$	$MDR = Mem[ALUOut]$ $Mem[ALUOut] = B$		
WB		$Reg[IR[20-16]] = MDR$		

# Types of exceptions

- Undefined instruction
  - Happens in Decode
  - Can be caused by any invalid instruction
- Overflow
  - Happens in Execute
  - Not caused by jumps, branches, lw/sw
  - Only R-type add/sub



# Types of exceptions

- Decode stage
  - Undefined instruction
- Execution stage
  - Overflow
- Cause = 0 – undefined exception
- Cause = 1 - overflow

# Undefined instruction

- What happens?

# Undefined instruction

- Cause gets 0
- $EPC = PC - 4$
- Exc. Level = 1 (in status register)
- PC = Address of exception handler

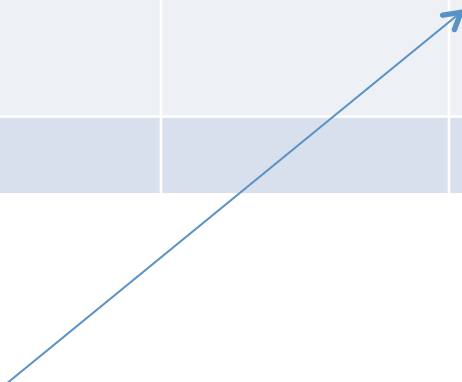
# Overflow exception

- Almost same thing
  - Cause gets 1
  - EPC = PC - 4
  - Exc. Level = 1 (in status register)
  - PC = Address of exception handler

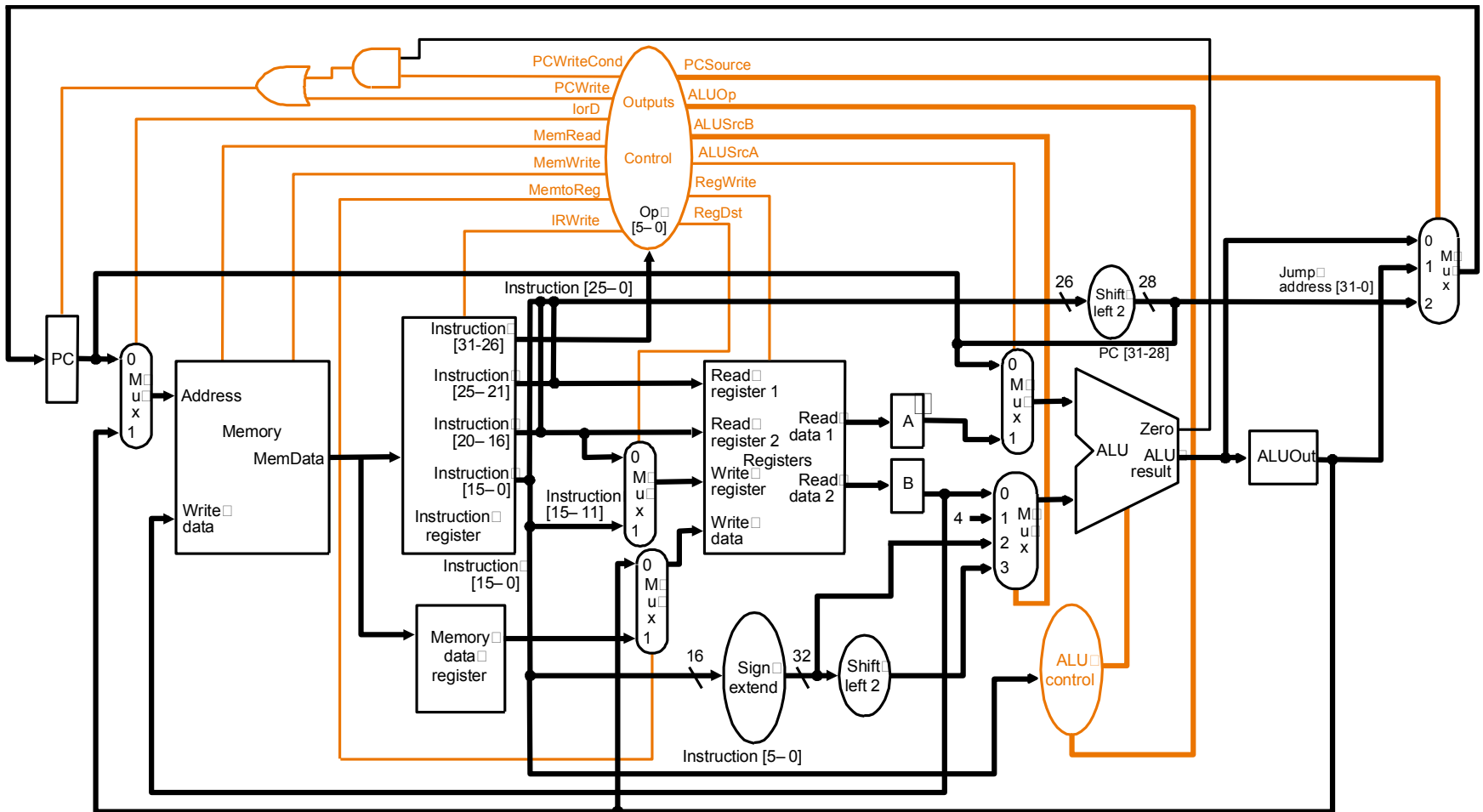
# RTL with Exceptions (overflow)

Step	R-Type	lw/sw	beq/bne	j	Exception
IF	$IR = Mem[PC]$ $PC = PC + 4$				
ID	$A = Reg[IR[25-21]]$ $B = Reg[IR[20-16]]$ $ALUOut = PC + (SE(IR[15-0]) \ll 2)$				
EX	$ALUOut = A \text{ op } B$	$ALUOut = A + SE(IR[15-0])$	If (A==B) then $PC = ALUOut$	$PC = PC[31-28]$    $(IR[25-0] \ll 2)$	<b>PC = 0x8000 0180 EPC = PC - 4 Cause = 0 or 1 Status = ?</b>
MEM	$Reg[IR[15-11]] = ALUOut$	$MDR = Mem[ALUOut]$ $Mem[ALUOut] = B$			
WB		$Reg[IR[20-16]] = MDR$			

The exception steps are done in one clock cycle.



# Datapath changes



# Datapath changes

- Add special cause register
- Add special status register
- Add logic to set cause register
  
- Add exception handler address as input PC

# Control changes

- Add exception control stage
  - Set ALU to subtract PC values
  - Update exception registers
  - Update PC
- Can arrive at exception stage from
  - Decode – undefined instruction
  - Execute – overflow
- Transition back to fetch stage
  - Run exception handler



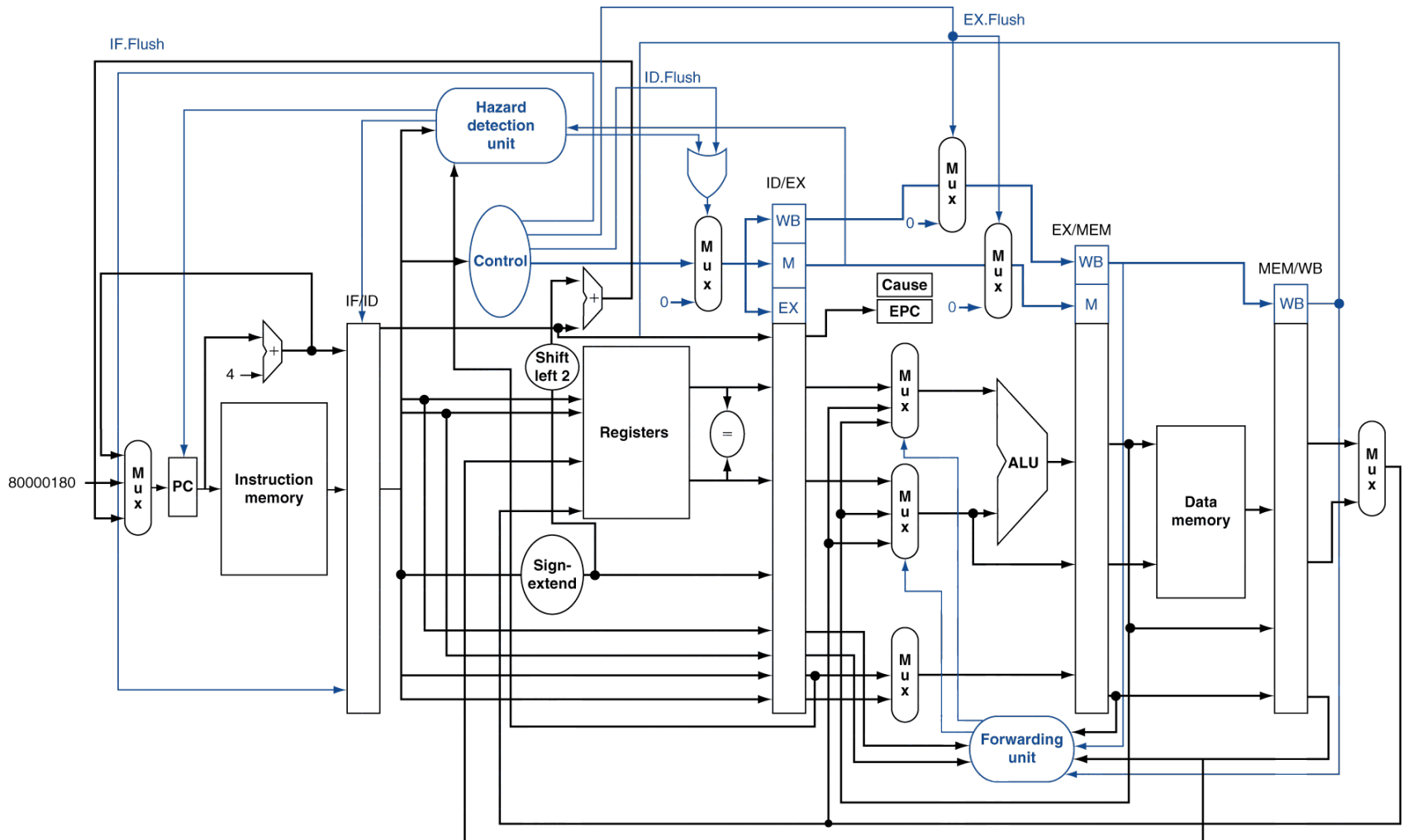
# What if we ditched the cause register?

- Vectored Interrupts
  - Jump to different address based on the cause
- Example:
  - Undefined opcode: 0x8000 0180
  - Overflow: 0x8000 0280
- MIPS
  - Cause register (with a few exceptions)
- X86
  - Uses vectored interrupts

# Exceptions in a Pipeline

- Possible exception
  - Invalid instruction
  - Overflow
  - Invalid memory address
  - Invalid instruction address

# Exceptions in a Pipeline



# Exceptions in a Pipeline

- Possible exception
  - Invalid instruction (ID)
  - Overflow (EX)
  - Invalid memory address (MEM)
  - Invalid instruction address (IF)
- Could generate 4 exception in one cycle!

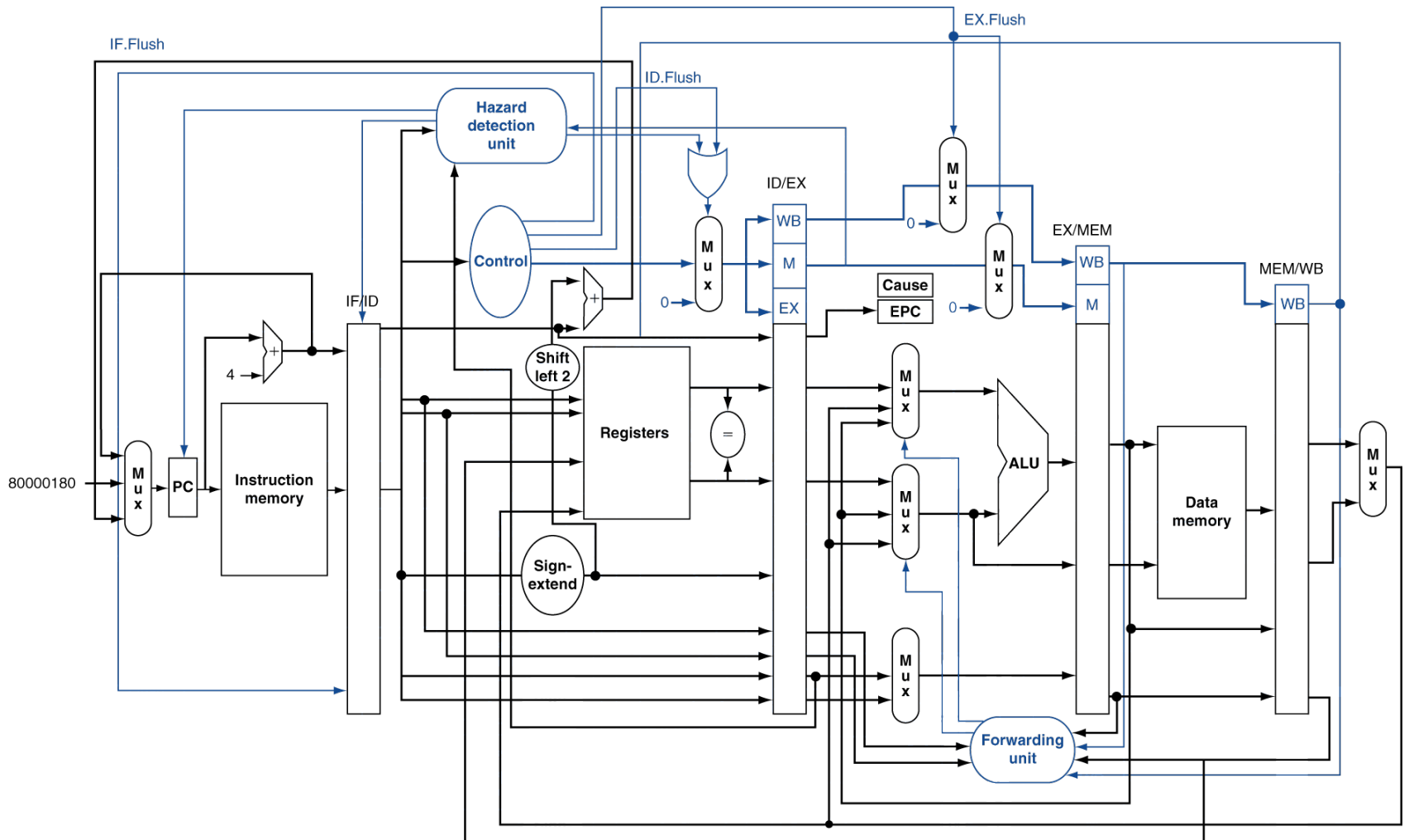
# Multiple Exceptions

- 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!

# 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



# Pipeline with Exceptions

- Diagram doesn't show  $EPC=PC-4$
- Possible solutions
  - Add an ALU
  - Pass original PC along



# 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)
```

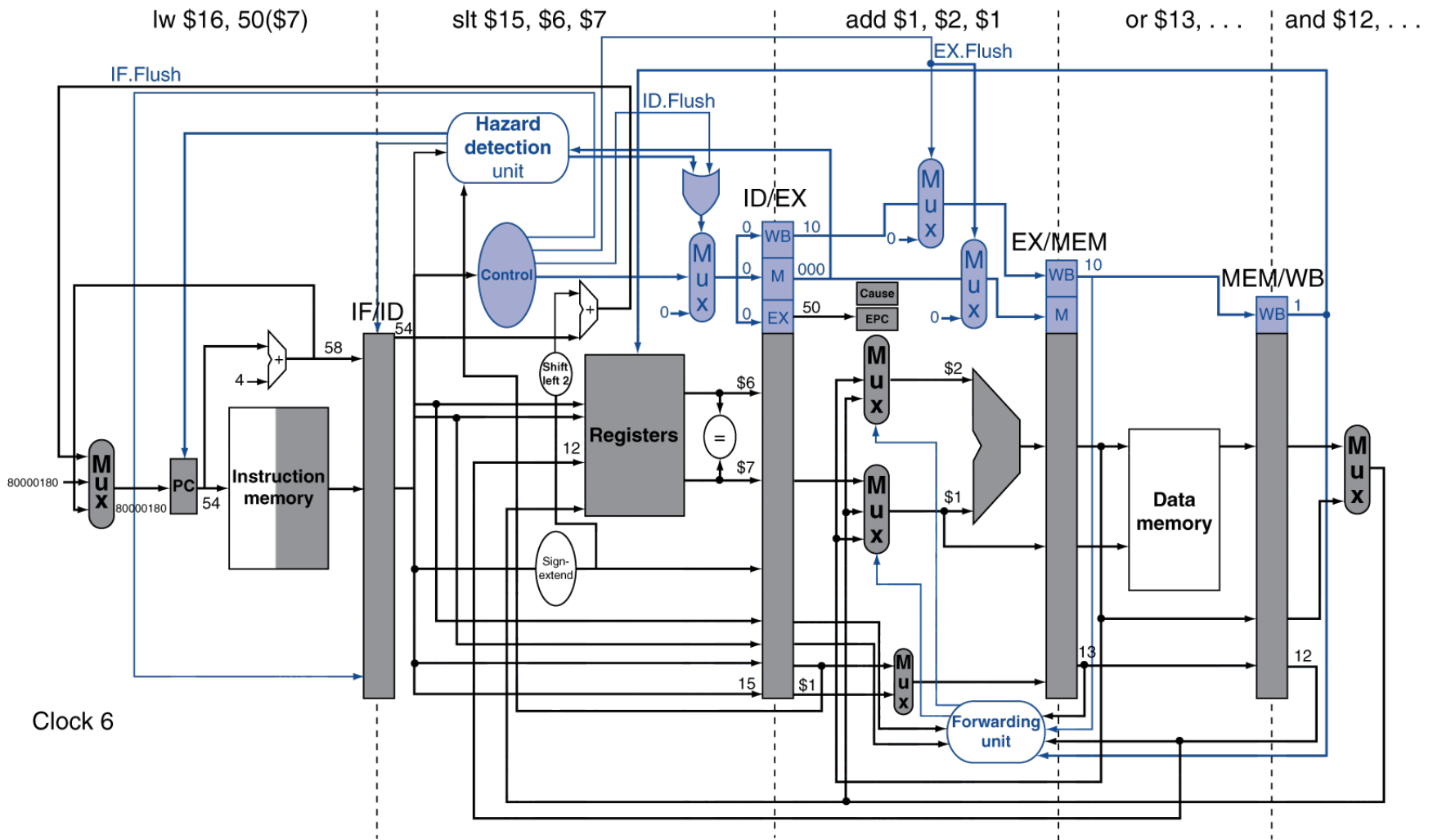
...

- Handler

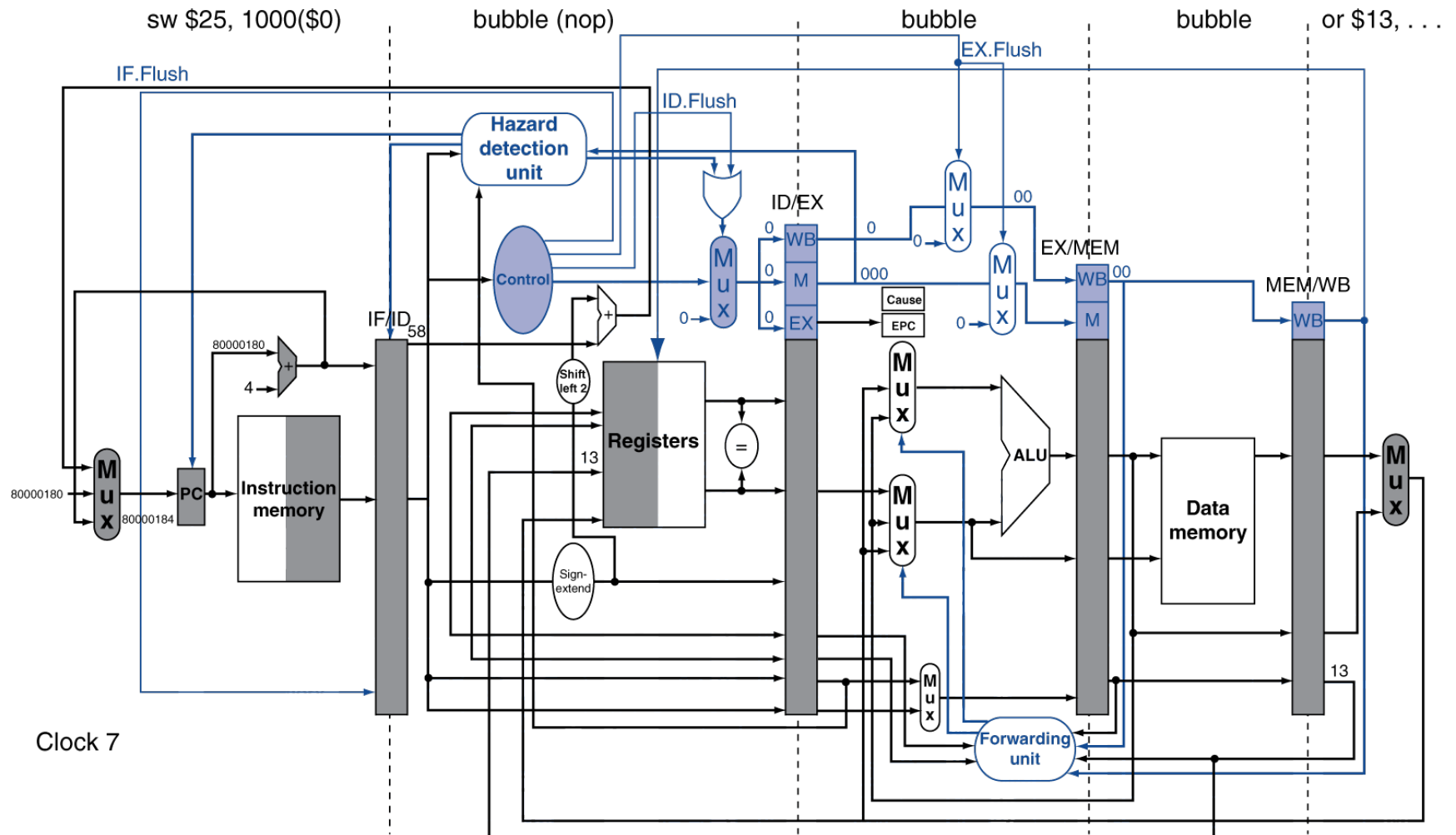
```
80000180 sw    $25, 1000($0)
80000184 sw    $26, 1004($0)
```

...

# Exception Example



# Exception Example



# Imprecise Exceptions

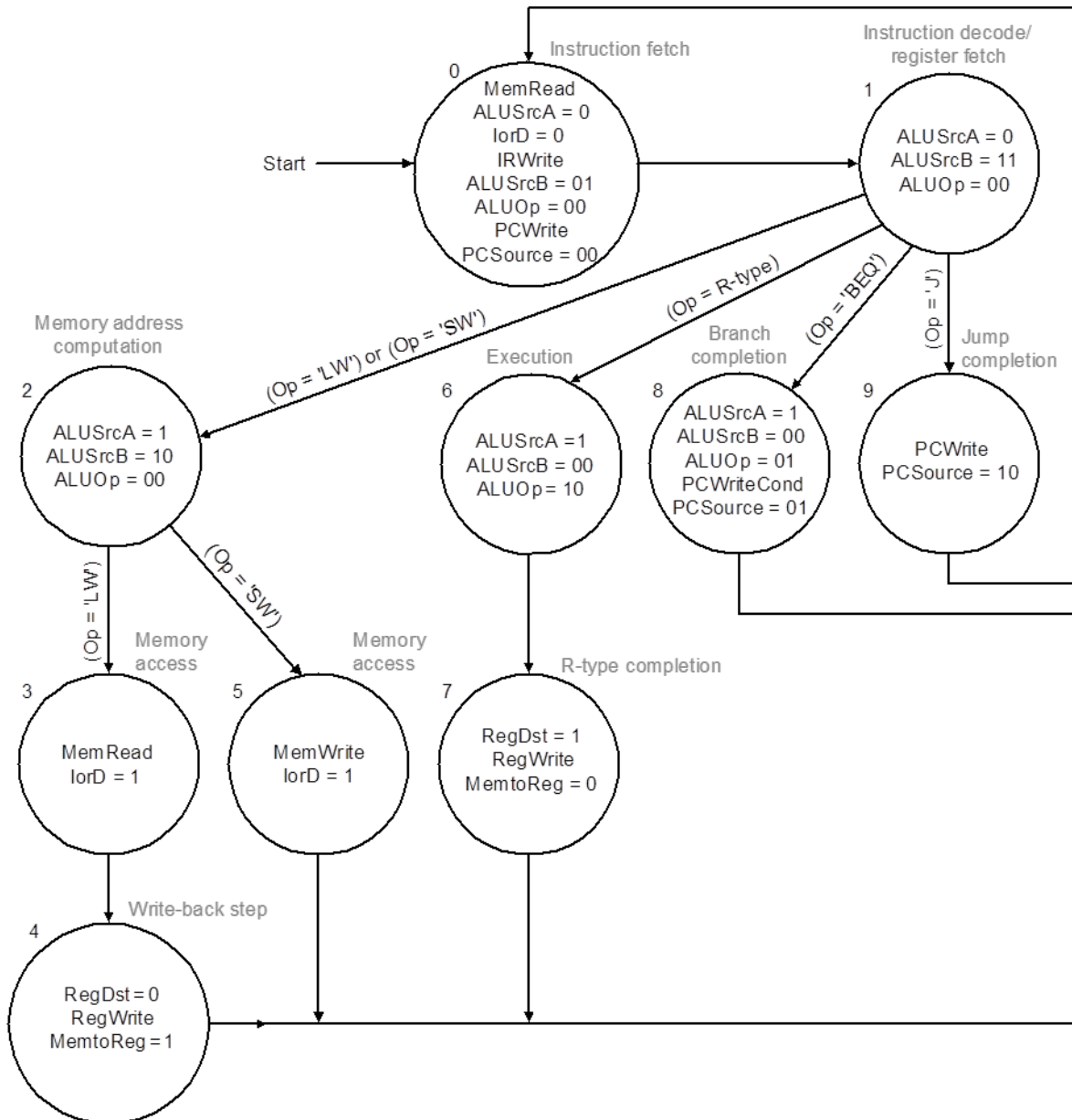
- Just stop pipeline and save state
  - Including exception cause(s)
- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require “manual” completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue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

# Multicycle Interrupts

- Are interrupts very different from exceptions?
- What modifications would you make for an interrupt?

# Multicycle Interrupts

- Are interrupts very different from exceptions?
  - Basically the same thing
  - Show that interrupt happened (cause register)
- What modifications would you make for an interrupt?
  - Check before each new instruction



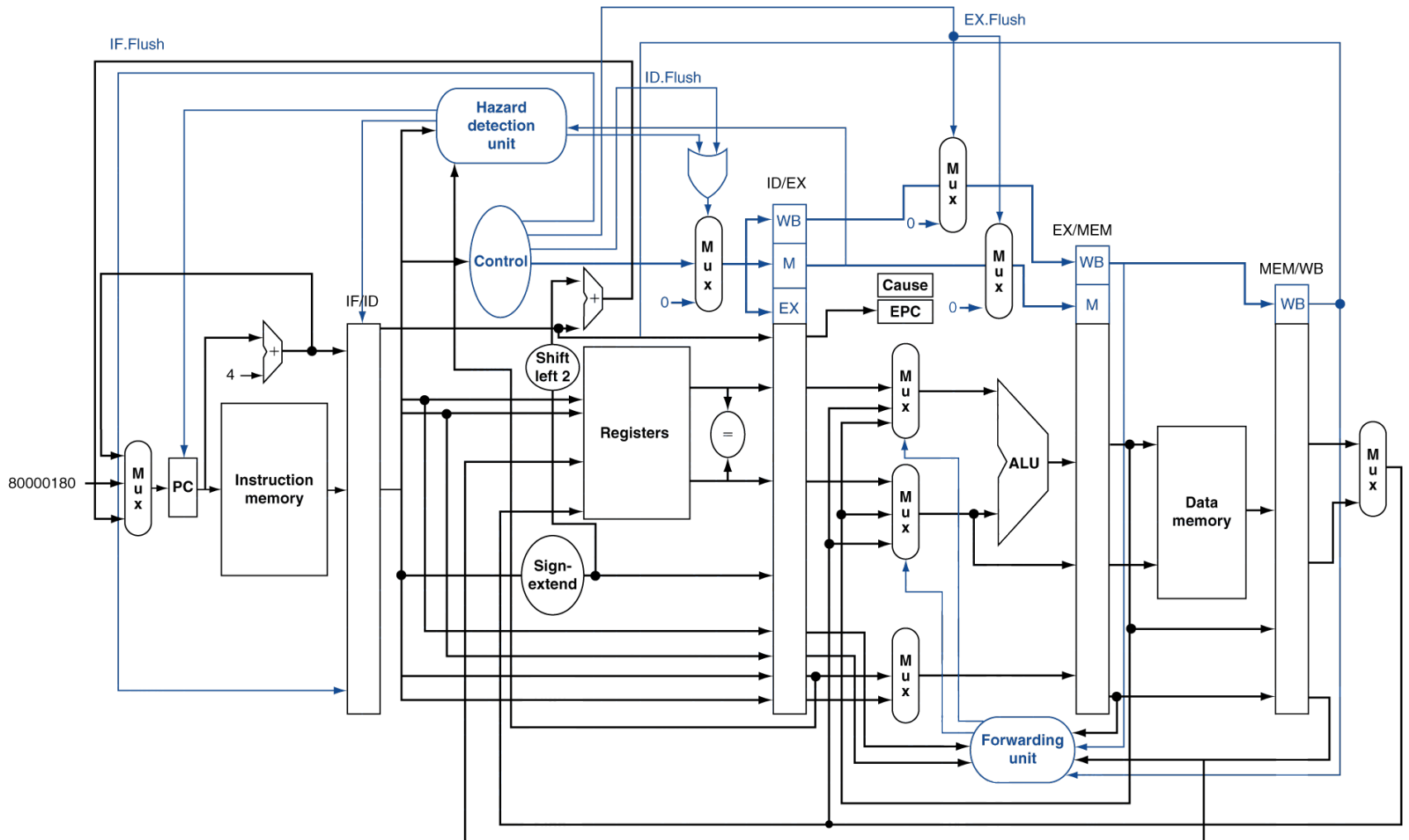
Check for  
 interrupts  
 before  
 fetching  
 next  
 instruction

# Pipeline interrupt

- Any changes from pipeline exceptions?



# Pipeline with Exceptions



# Pipeline interrupt

- Any changes from pipeline exceptions?
  - Don't flush
  - Finish current instructions
  - Go to handler
  - Return to PC+4

# Review and Questions

- Review: exceptions and interrupts
- Exceptions in multicycle datapaths
- Exceptions in pipelined
- Interrupts