Chapter 4

Exceptions

Exceptions

- Control is the most challenging aspect of processor design
  - One of the hardest part of control is implementing exceptions
- Exceptions: events other than branches or jumps that change the normal flow of instruction execution

- Were Initially created to handle unexpected events from with in the processor

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

Exception / Interrupt

- Exception : An unscheduled event that disrupts program execution
- Interrupt: An execution that comes form outside of the processor

Type of event	From where	MIPS technology
I/O device request	External	Interrupt
Invoke the OS from user pgm	Internal	Exception
Arithmetic overflow	Internal	Exception
Undefined instruction	Internal	Exception
h/w malfunctions	Either	Interrupt or exception

- Detecting exceptional conditions and taking the appropriate actions is often on the critical timing path of the processor, which determines the clock cycle time and thus performance

Handling Exceptions

- eg: undefined instructions and overflow
  - Add $s1, $s2, $s1

- The basic action that the processor must perform when an exception occur is to
  - Save PC address 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
  - Transfer control to OS(Jump to handler at 8000 00180)

- The OS then take appropriate action, which may involve
  - Providing service to user pgm
  - Taking some predefined action in response to overflow
  - Stopping execution of the pgm and reporting the error

- After performing what ever action is required because of exception, the OS can terminate/continue the pgm execution(using EPC to determine where to start)

Handler Actions

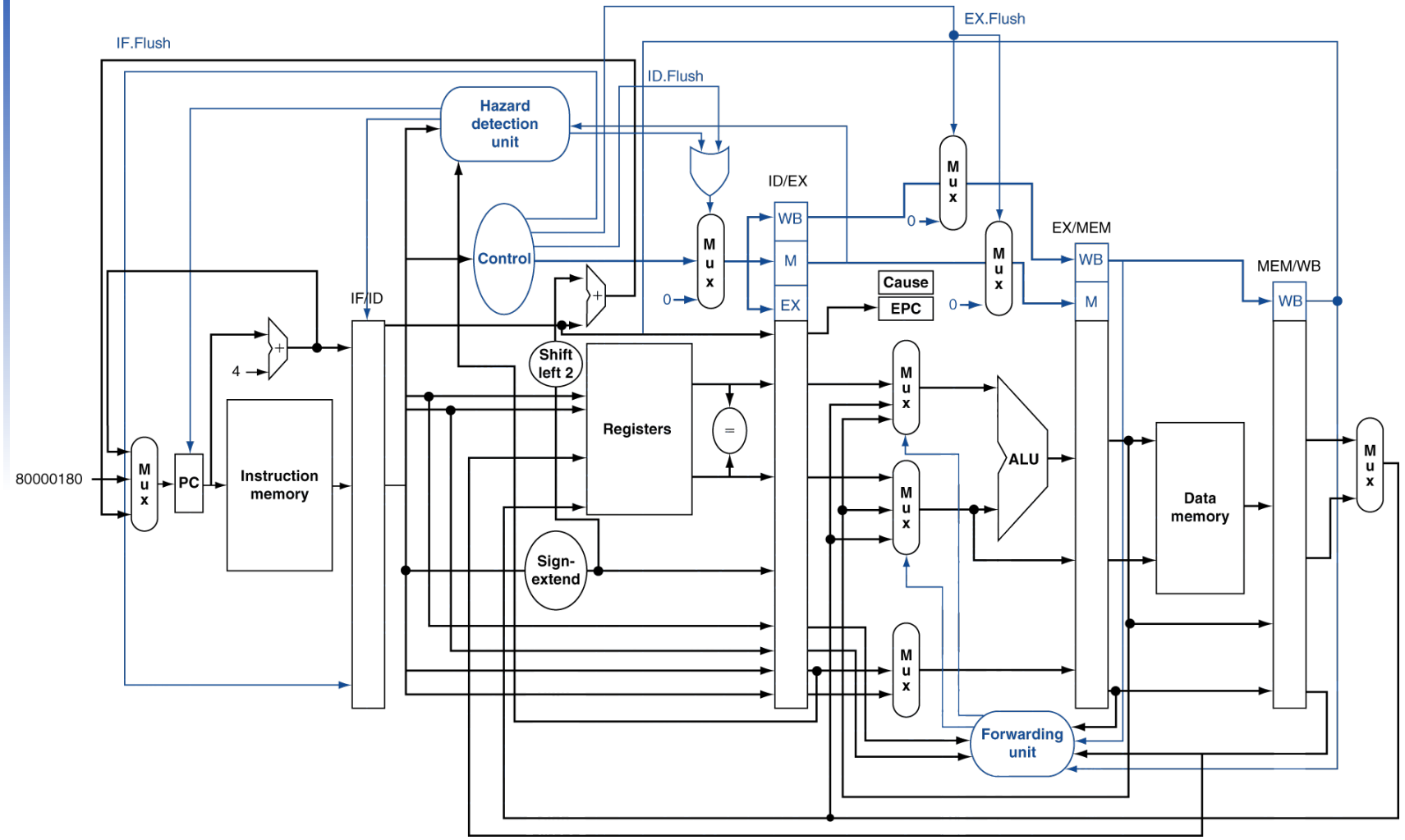
- Read cause, and transfer to relevant handler
- Determine action required
- If restart able
  - Take corrective action
  - use EPC to return to program
- Otherwise
  - Terminate program
  - Report error using EPC, cause, …

Exceptions in a Pipeline

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

  `add $s1, $s2, $s1`
  - Prevent $s1 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 Properties

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

Exception Example

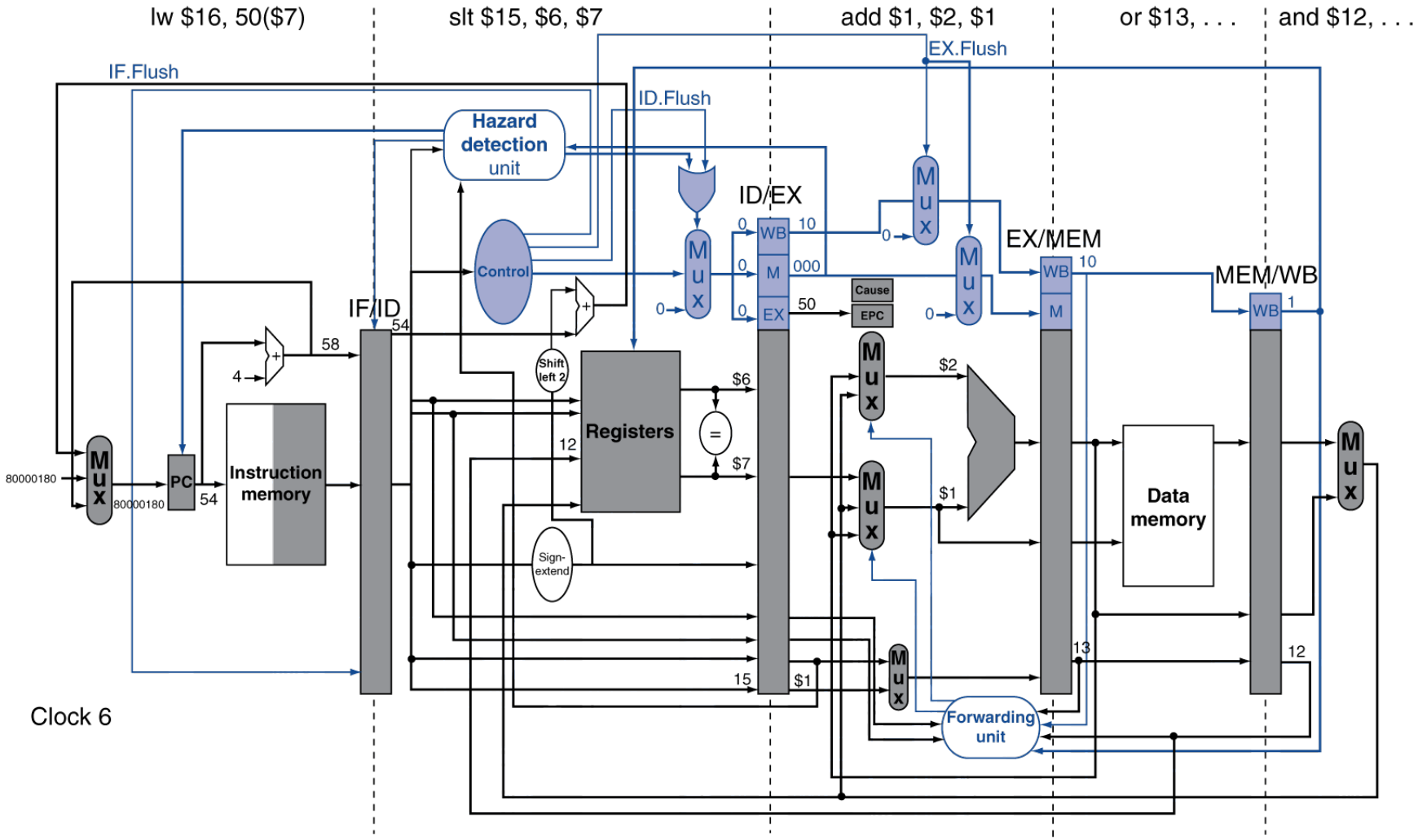
- Exception on add in

```
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)
…
```

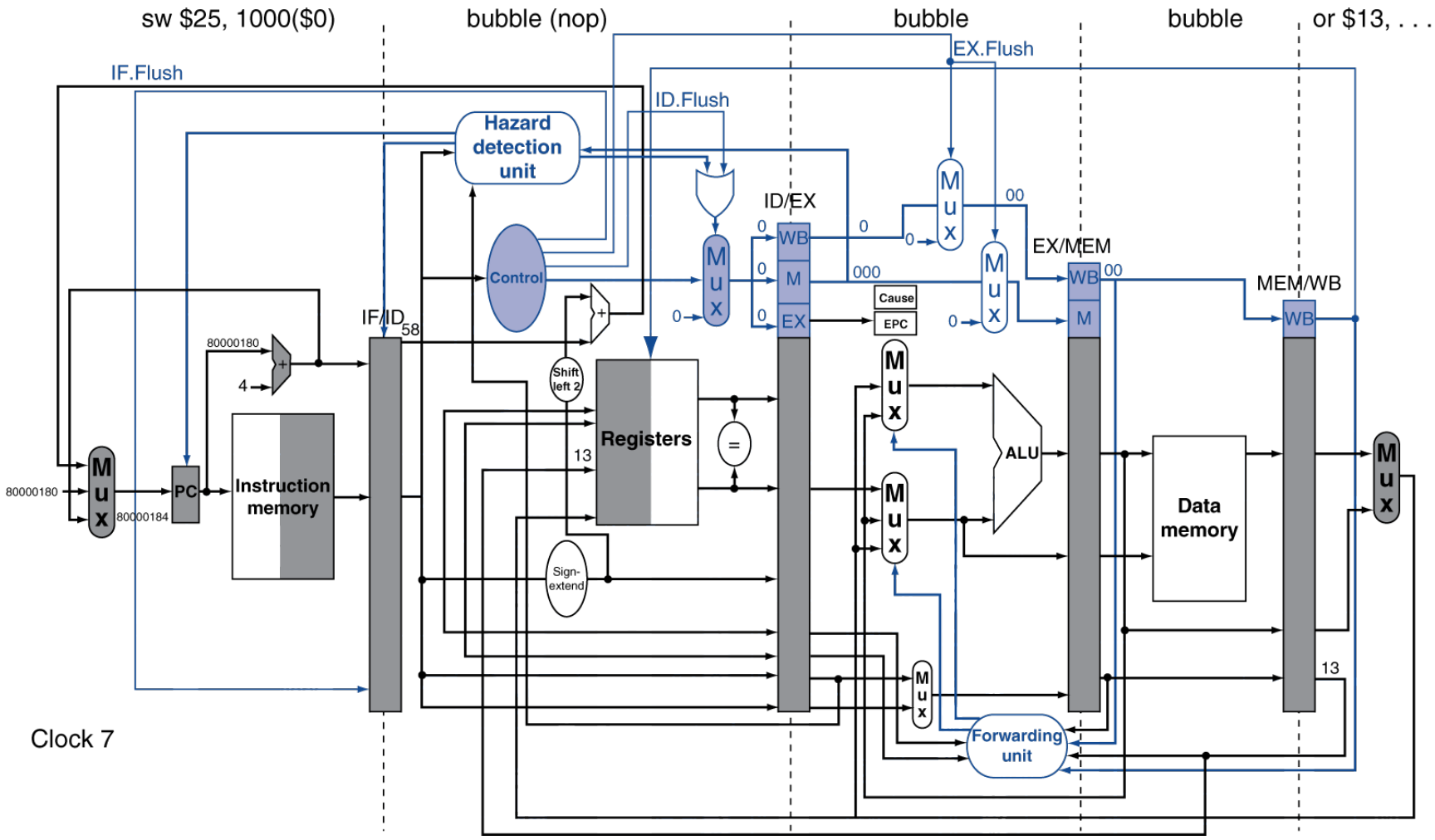
- Handler

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

Exception Example



Exception Example



Multiple Exceptions

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