Topic 10

Exceptions and Interrupts

Exceptions and Interrupts

- "Unexpected" events requiring normal program flow to be altered, these events include
  - Exception
    - Arises within the CPU
    - e.g., undefined opcode, overflow, syscall, …
  - Interrupt
    - From an external peripheral (device) or environment
- Dealing with them without sacrificing performance is hard
  - Because alteration of CPU flow

Handling Exceptions

- Save PC of interrupted instruction
  - In MIPS: Exception Program Counter (EPC), 32 bits
  - Actually, save address of interrupted instruction + 4
- Service the exception or interrupt according to the causes
  - Cause is coded in a 32-bit CAUSE register in MIPS
  - Single entry point – single vector interrupt
  - Multiple entry points, each entry point for one cause – Multiple Vectored Interrupts
- Jump to handler at entry point
  - Handler is a special function that handles a specific exception

Vectored Interrupts

- Vectored Interrupts
  - Handler address determined by the cause
- Example:
  - Undefined opcode: 0xC000 0000
  - Overflow: 0xC000 0020
  - ….: 0xC000 0040
  - MIPS: Entry points separated by limited number of words
    - 32 bytes in above example
- Instructions at interrupt vector
  - Deal with the interrupt if handler is small, or
  - Jump to bigger mem space if hander is too big

Handler Actions

- Read CAUSE register, and transfer to exception handler (jump to the function)
  - If single vector, determine type of exception first
  - If multiple vector, run handler directly
- If restartable exception
  - Take corrective action
  - use EPC (PC<=EPC-4) to return to program
- If time consuming
  - Suspend current program, switch to another
- If fatal
  - Terminate program
  - Report error using EPC and CAUSE register

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)
…
```

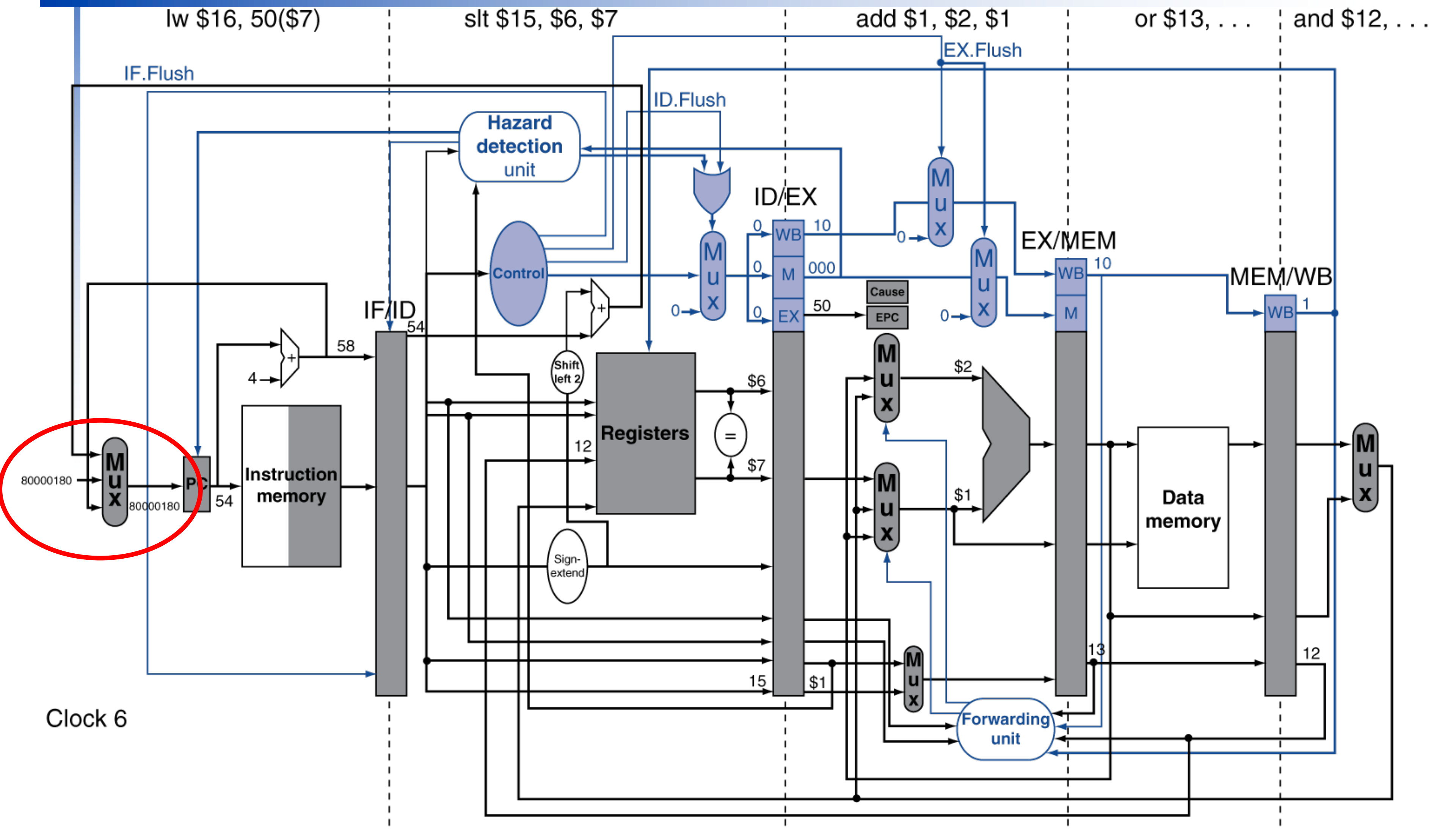
Exceptions in a Pipeline

- Example: overflow on add in EX stage

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

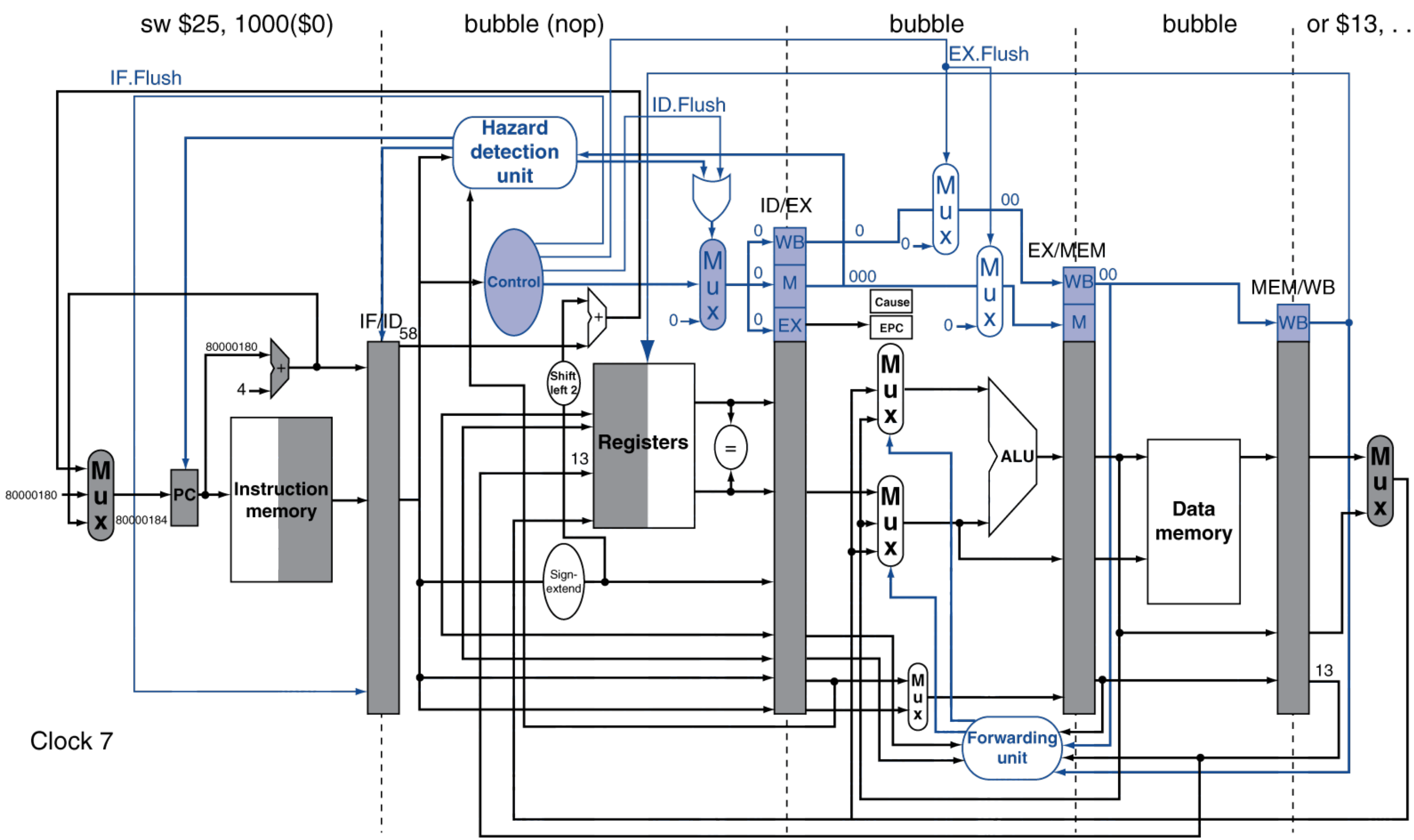
  - Complete previous instructions
  - Flush add and subsequent instructions
    - Or put the interrupted instruction on hold
  - Set CAUSE and EPC register values
  - Load address of handler into PC
  - Transfer control to handler
- Like another form of control hazard
  - treats like branch hazard – flush instructions
  - Use similar hardware

Exception Example



Cause and EPC not necessarily in EX stage

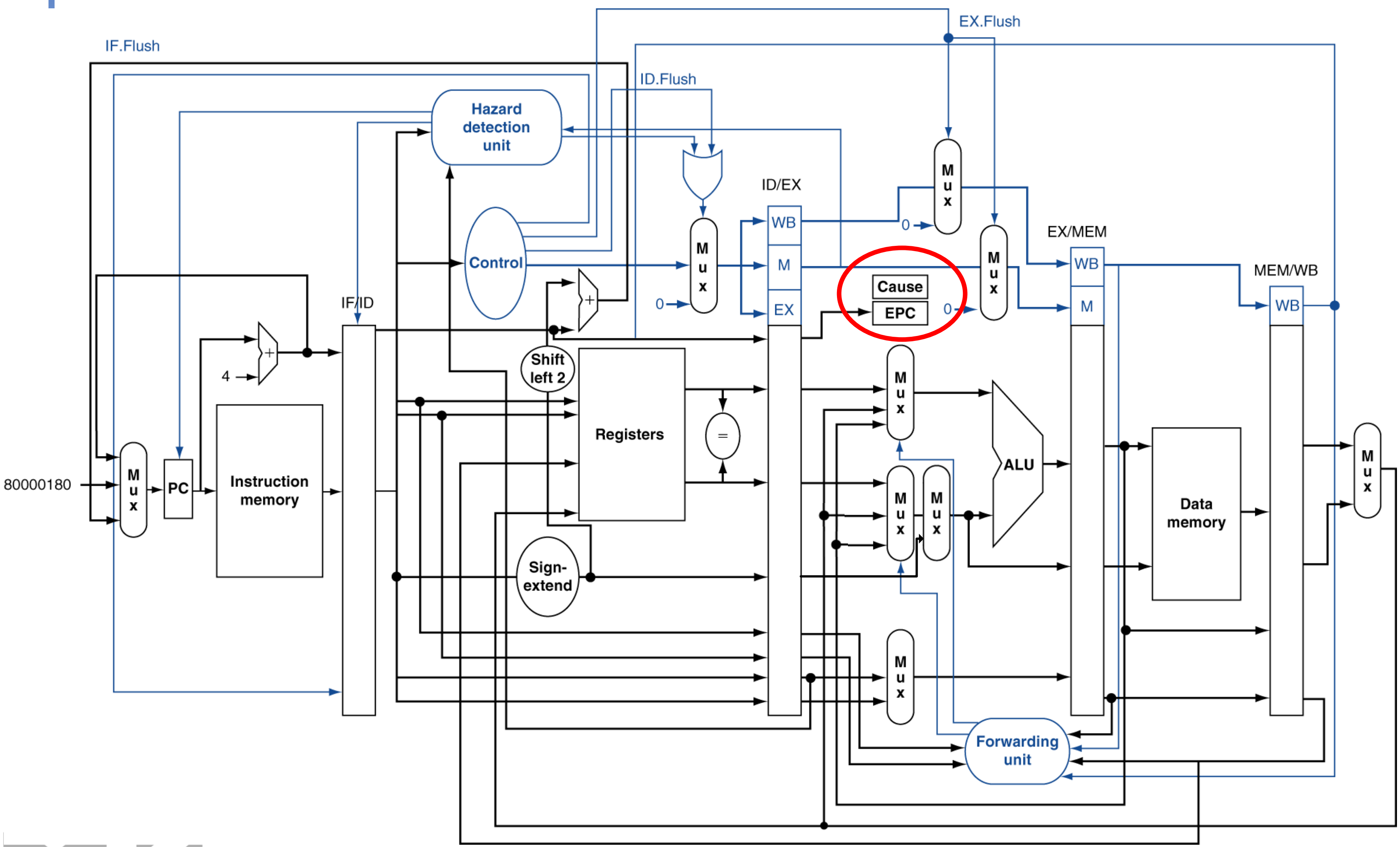
Exception Example



After Exception is Serviced

- Pass control back to the interrupted instruction
  - Restart the interrupted instruction from IF, or
  - Continue the instructions from the moment/stages they were interrupted – how?
- Or terminate the program & report error and cause

Pipeline with Exceptions



Multiple Exceptions

- Could have one exception followed by another
  - EPC could be overwritten
- Could have multiple exceptions simultaneously
  - Pipeline holds multiple instructions causing exception
- Simple approach: deal with exception from earliest instruction (aka. Precise exception)
  - Flush subsequent instructions
- In complex pipelines
  - Multiple instructions issued per cycle
  - Maintaining precise exceptions is difficult!

Imprecise Exceptions

- Just stop pipeline and save state
  - Including exception cause(s)
- Let the operating system 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

Summary Notes

- Pipelining is easy
  - The devil is in the details
    - e.g., detecting data hazards
- Pipelining is independent of technology