



Topic 10

Exceptions

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 handler is too big



Handler Actions

- Read CAUSE register, and transfer to exception handler (jump to the function)
 - If single vector, have to determine what exception/interrupt in handler
 - If multiple vector, run handler directly
- If restartable exception
 - Take corrective action
 - use EPC ($PC \leq EPC - 4$) to return to program
- Otherwise
 - Terminate program
 - Report error using EPC and CAUSE register

Exception Example

- Exception on `add` in

40	<code>sub</code>	<code>\$11, \$2, \$4</code>
44	<code>and</code>	<code>\$12, \$2, \$5</code>
48	<code>or</code>	<code>\$13, \$2, \$6</code>
4C	<code>add</code>	<code>\$1, \$2, \$1</code>
50	<code>slt</code>	<code>\$15, \$6, \$7</code>
54	<code>lw</code>	<code>\$16, 50(\$7)</code>

...

- Handler

80000180	<code>sw</code>	<code>\$25, 1000(\$0)</code>
80000184	<code>sw</code>	<code>\$26, 1004(\$0)</code>

...

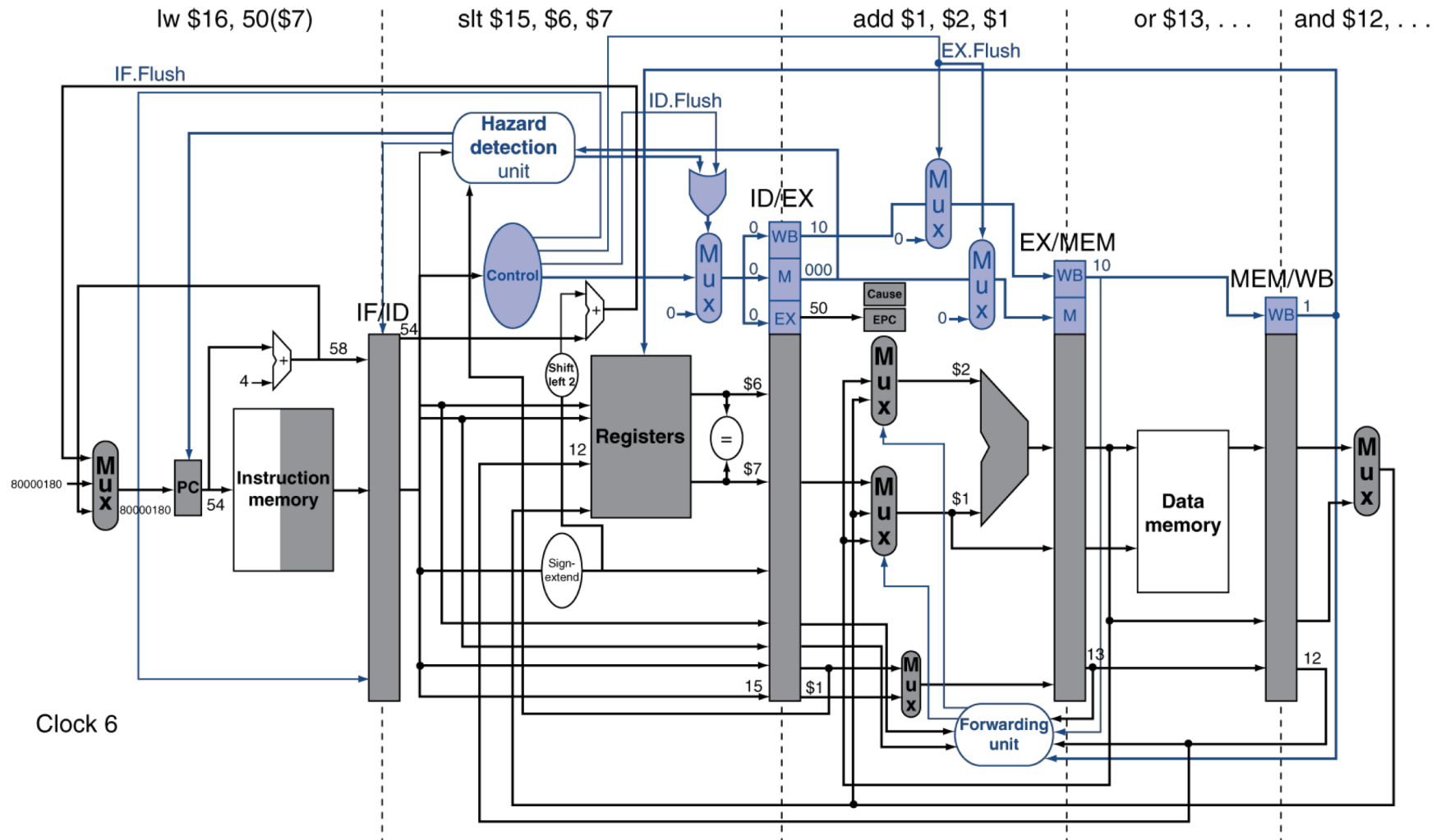
Exceptions in a Pipeline

- 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

After Exception is Serviced

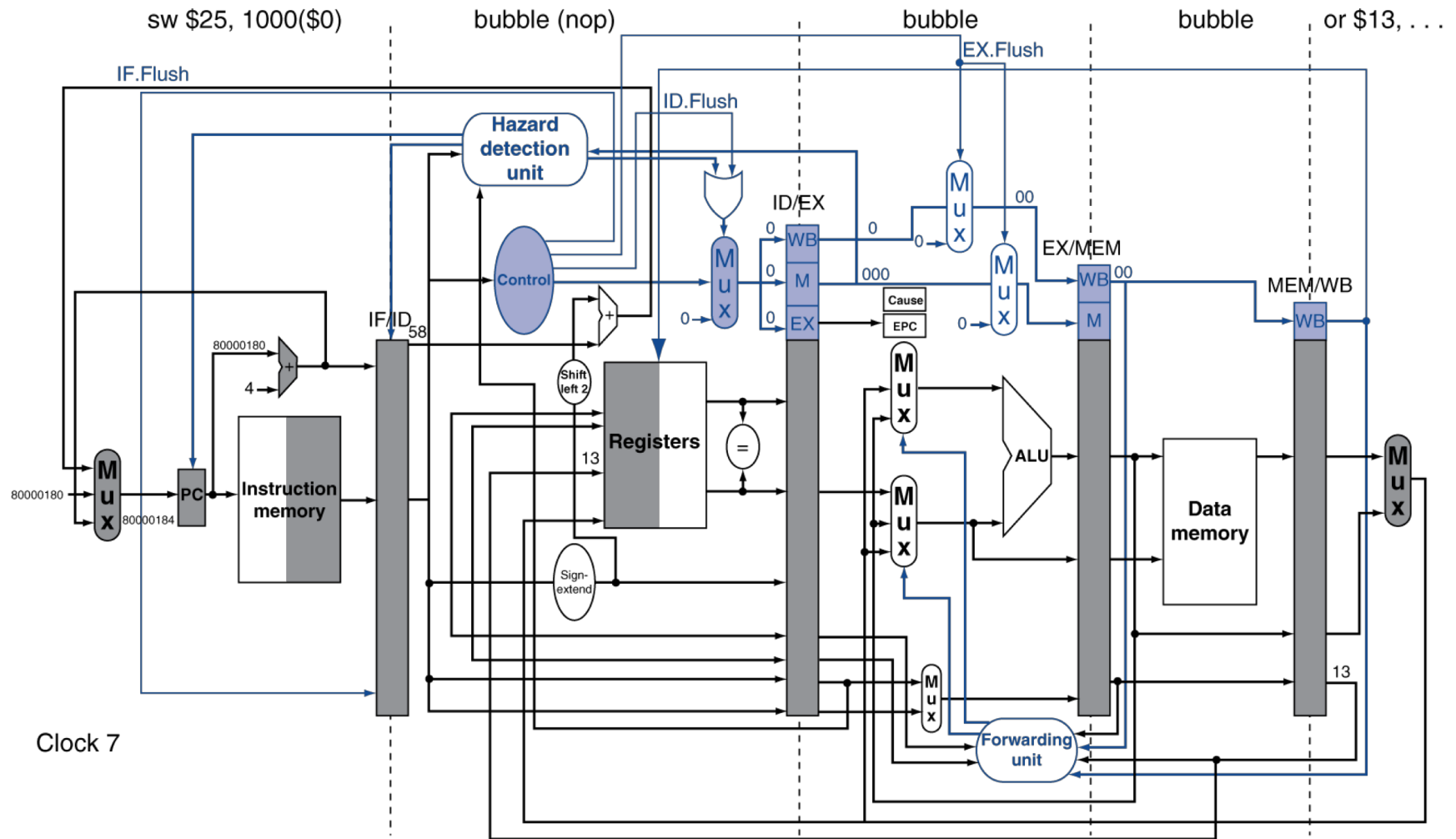
- Pass control back to the interrupted instruction
 - Restart the instruction
 - Start the instruction from the moment/stage it's interrupted – how?
- Or terminate the program & report error and cause

Exception Example

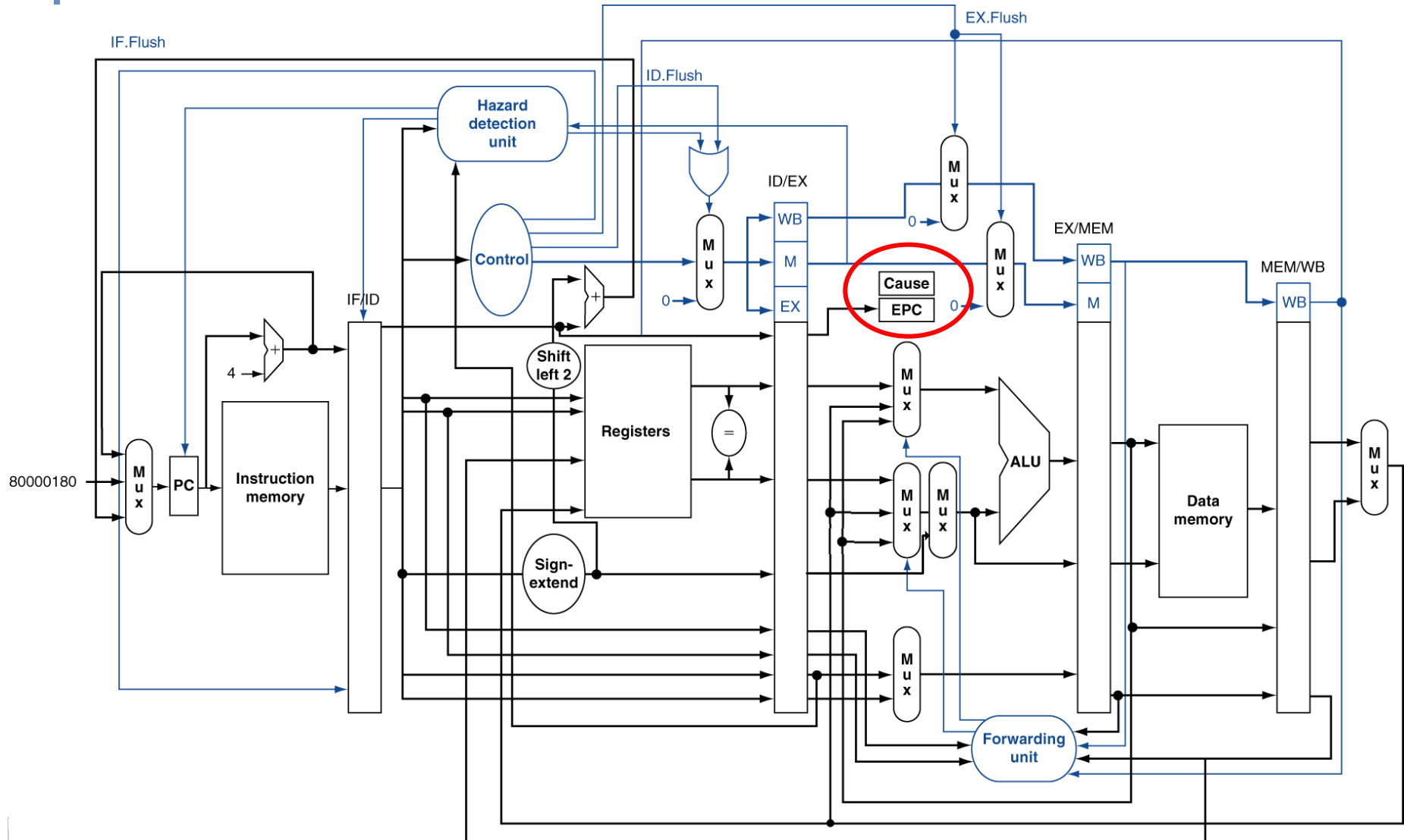


Cause and EPC not necessarily in EX stage

Exception Example



Pipeline with Exceptions



Multiple Exceptions

- Could have multiple exceptions simultaneously
 - Pipeline holds multiple instructions causing exception
 - Multiple unexpected events happen simultaneously
- Simple approach: deal with exception from earliest instruction
 - Flush subsequent instructions
 - Aka “Precise” exceptions
- 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