CMPE110 Lecture 14
Exceptions

Heiner Litz

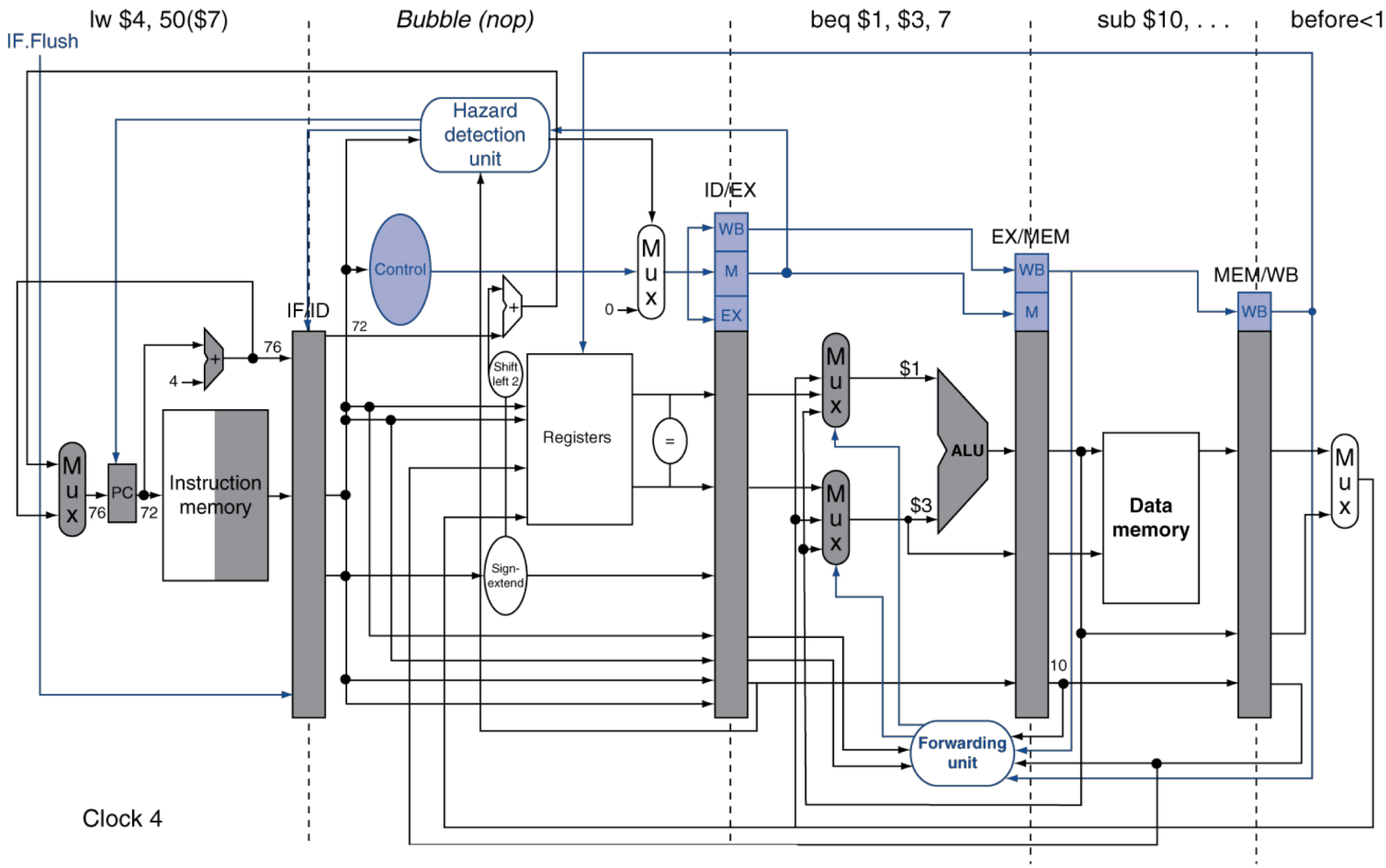
https://canvas.ucsc.edu/courses/19290

Announcements

- No Quiz on Friday
- Midterm: Wed 02/13
  - 1 double sided page of notes
  - RISC-V Green card/cheat sheet part of the exam

Review

Example: Branch Taken

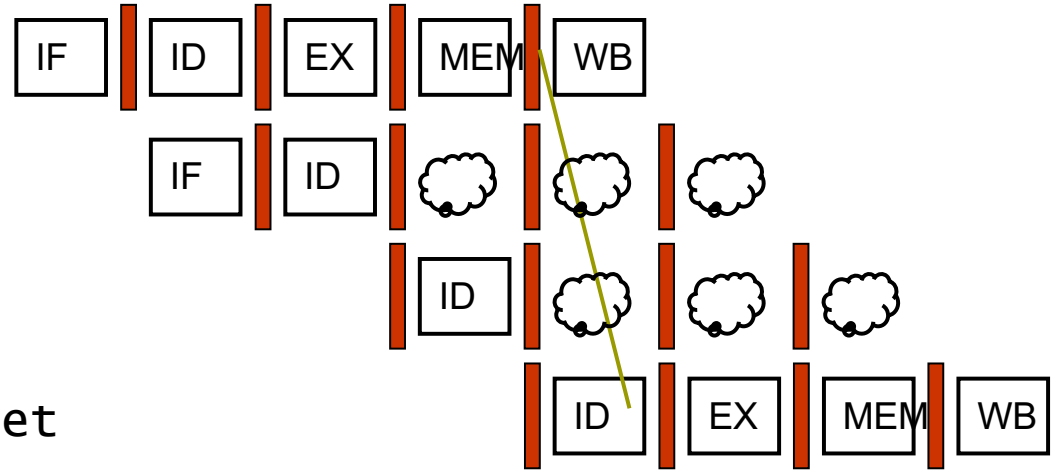


nop inserted using IF.Flush

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
```



Exceptions and Interrupts

Exceptions and Interrupts

- "Unexpected" events requiring change in flow of control
  - Switch from user to privileged (kernel) mode
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
  - E.g., undefined opcode, overflow, div by 0, syscall, …
- Interrupt
  - E.g., from an external I/O controller (network card)
- Dealing with them without sacrificing performance is hard

Handler Actions

- Read cause, and transfer to relevant handler
  - A software switch statement
  - May be necessary even if vectored interrupts are available
- Determine action required
- If user process is restartable
  - Take corrective action
  - Use EPC to return to program
- Otherwise
  - Terminate program (e.g., segfault, …)
  - Report error using EPC, cause, …

Precise Exceptions

- Definition: precise exceptions
  - All previous instructions had completed
  - The faulting instruction was not started
  - None of the following instructions were started
    - No changes to the architecture state (registers, memory)
- Why are they desirable by OS developers?
- With single cycle design, precise exceptions are easy
  - Why?
- With pipelined design, they can be tricky
  - Why?

Exceptions in a Pipeline

- Another form of control hazard
- Consider division by zero on add in EX stage

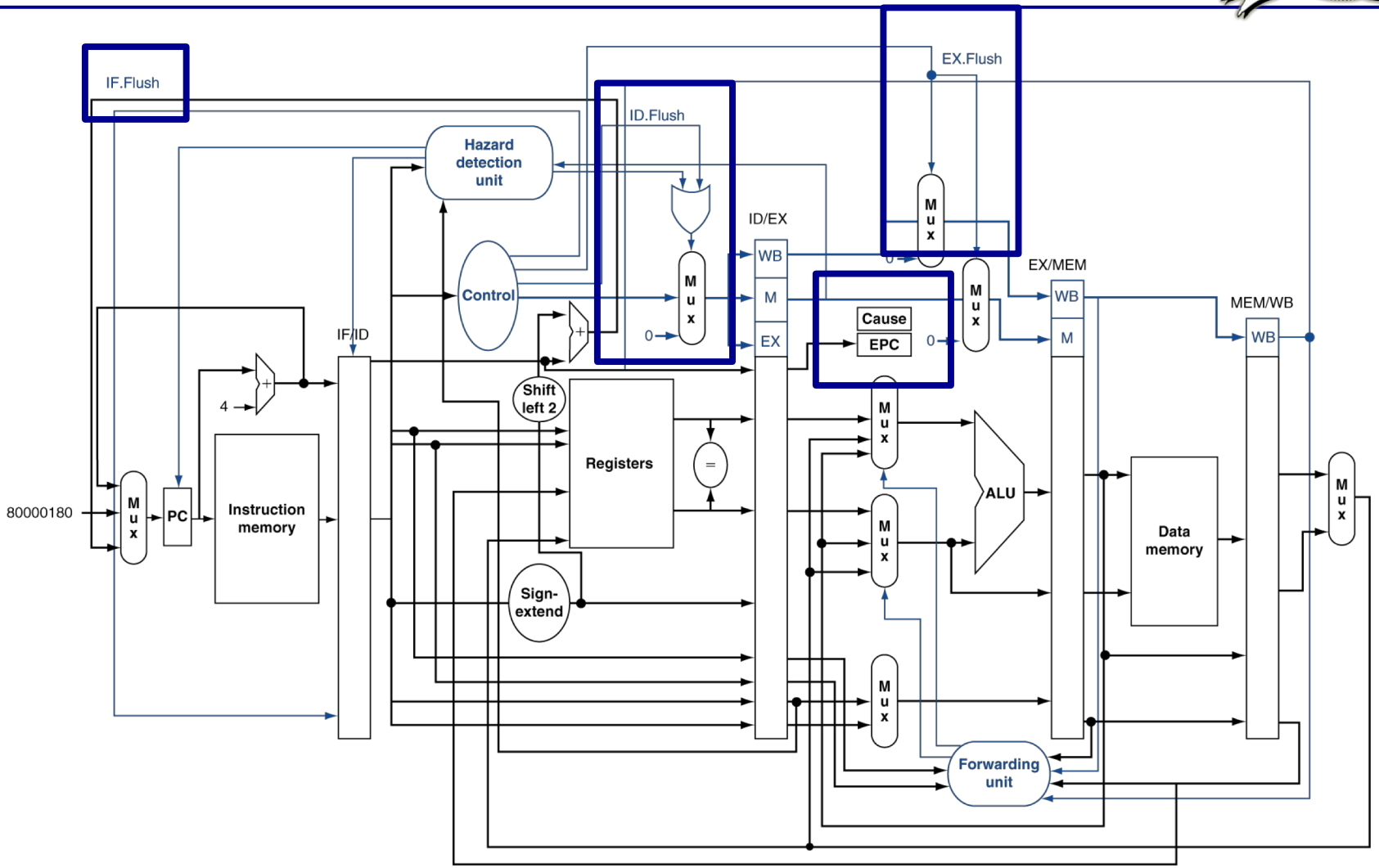
```
divi $1, $1, 0
```

  - Prevent $1 from being overwritten with – what?
  - Complete previous instructions
  - Nullify subsequent instructions
  - Set Cause and EPC register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Uses much of the same hardware

Nullifying Instructions

- Nullify = turn an instruction into a nop (or bubble)
  - Reset its RegWrite and MemWrite signals
    - Does not affect the state of the system
  - Resets its branch and jump signals
    - Does not cause unexpected flow control
  - Mark that it should not raise any exceptions of its own
    - Does not cause unexpected flow control
  - Let it flow down the pipeline

5-stage Pipeline with Exceptions



Exception Example

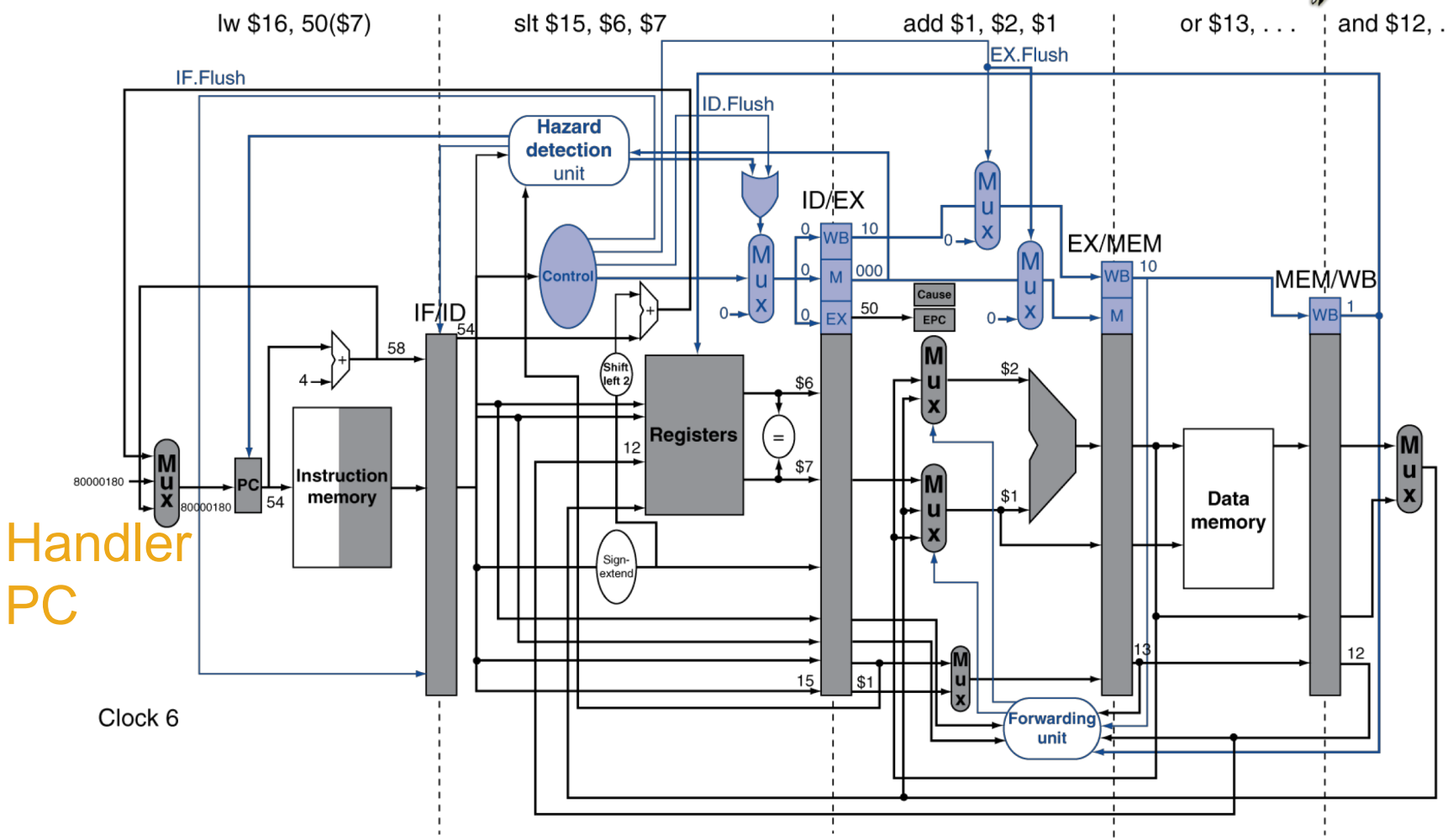
- Exception on `add` in

```
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)
…
```

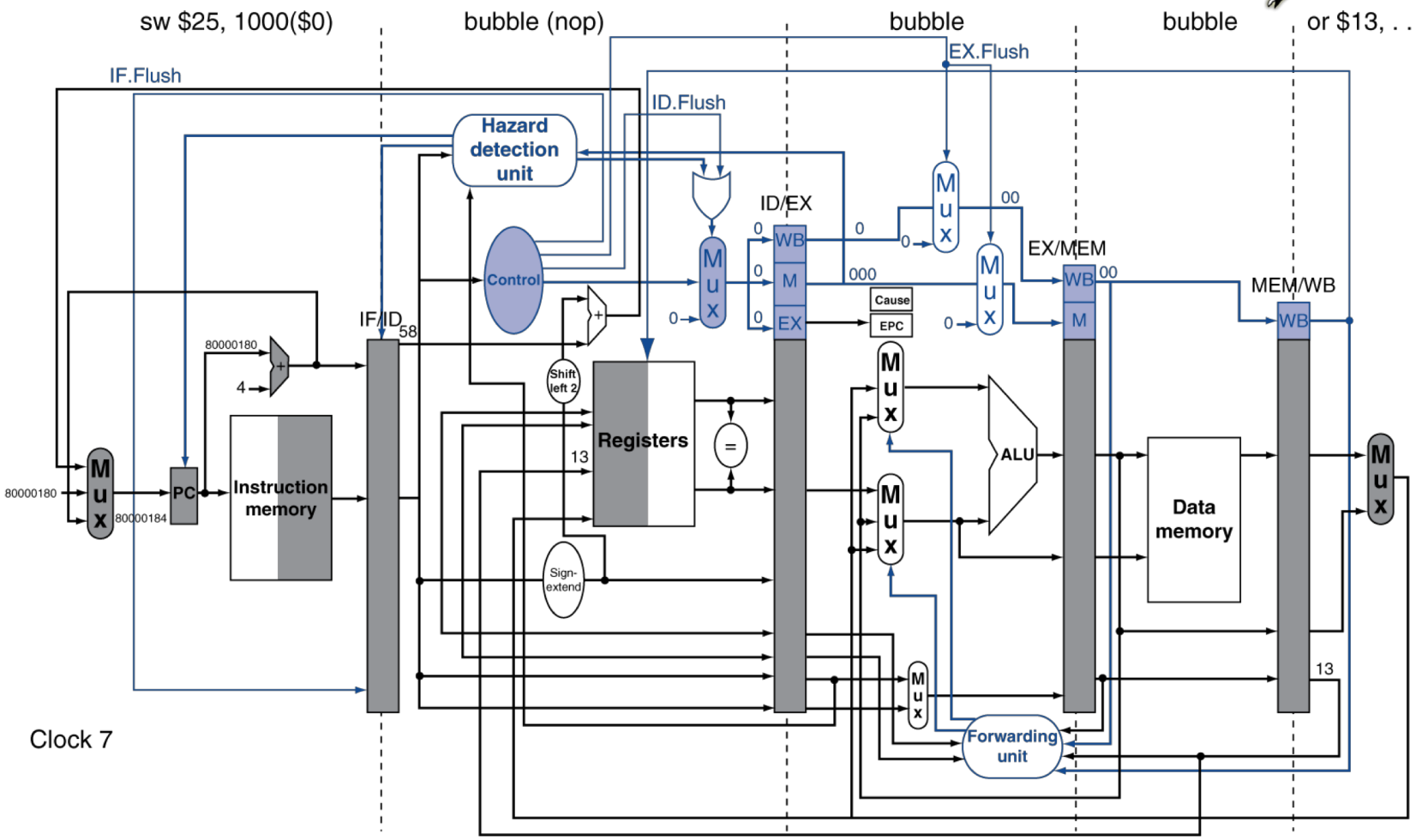
- Handler

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

Exception Example



Exception Example



Dealing with Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once
- Simple approach: deal with exception in program order
  - Single commit point (MEM or WB stage)
  - Flush subsequent instructions
  - Necessary for “precise” exceptions

Beyond the 5-stage Pipeline

Desirable properties

- Higher clock frequency
- $CPI_{base} < 1$
- Avoid in-order stalls
  - When instruction stalls, independent instructions behind it stall
- Avoid stalls due to branches

How do we go about it?

Modern Processors

High clock frequency → deep pipelines

10 – 20 stages

$CPI_{base} < 1$ → superscalar pipelines

Launch 2, 3, or 4 instructions every cycle

Avoid in-order stalls → out-of-order execution

Re-order instructions based on dependencies

Avoid stalls due to branches → branch prediction

Keep history about direction and target of branches

Why Does it Work: Instruction-level Parallelism

Independence among instructions

Example with ILP

```
add $t0, $t1, $t2
or  $t3, $t1, $t2
sub $t4, $t1, $t2
and $t5, $t1, $t2
```

Most programs have ILP

But it is not infinite

Scope also matters a lot

Example with no ILP

```
add   $t0, $t1, $t2
or  $t3, $t0, $t2
sub $t4, $t3, $t2
and $t5, $t4, $t2
```

Question

How much ILP is there in common programs?

Multiple Instruction Issue (Superscalar)

Fetch and execute multiple instructions per cycle => CPI < 1

Example: fetch 2 instructions/clock cycle

Dynamically decide if they can go down the pipe together or not (hazard)

Pipe Stages

IF	ID	EX	MEM	WB		
IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB	
Ideal CPI = 0.5	IF	ID	EX	MEM	WB	
		IF	ID	EX	MEM	WB
		IF	ID	EX	MEM	WB

Resources: double amount of hardware (FUs, Register file ports)

Issues: hazards, branch delay, load delay

Modern processors: Intel x86: 4-way, ARM: 6-way/8-way

Deeper Pipelining

Increase number of pipeline stages

- Fewer levels of logic per pipe stage
- Higher clock frequency

Example 9-stage pipeline

Pipe Stages

IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB	
	IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB

Issues: branch delay, load delay: CPI = 1.4 - 2.0, cost of pipeline registers

Modern pipelines

- Pentium: 5 stages
- Original Pentium Pro: 12 stages
- Pentium 4: 31 stages, 3+ GHz
- Tejas: 40-50 stages, 10 GHz, cancelled!
- Core 2: 14 stages

Dynamic Scheduling

Execute instructions out-of-order

Fetch multiple instructions per cycle using branch prediction

Figure out which are independent and execute them in parallel

Example

```
add   $t0, $t1, $t2
or  $t3, $t0, $t2
sub $t0, $t1, $t2
and $t5, $t0, $t2
```

Superscalar + Dynamic scheduling

```
add   $t0, $t1, $t2        sub $t0, $t1, $t2
or  $t3, $t0, $t2          and $t5, $t0, $t2
```

What's wrong with this?

Announcements

- **Evaluation of the Tensor Processing Unit: A Deep Neural Network Accelerator for the Datacenter**
- David Patterson, UC Berkeley & Google
- Friday, February 8, 2:40 PM, E2-180