Lecture 19
Exceptions

School of Computer Science and Engineering
Soongsil University

4. The Processor

4.9 Exceptions

- Who can stop the running program ?

  Who can switch the running program in processor ?
  - Software cannot but hardware control signals can do.
  - Normally, control signals are generated in the processor as the running program specifies.
  - So, we need a control signal which is independent of the running program.
  - It forces processor to execute the instructions at a specified address instead of following the normal program flow.
- **Exceptions and interrupts**
  - Events other than branches or jumps that change the normal flow of instruction execution
  - Unexpected events requiring change in flow of control
  - Hardware-generated function calls

Exceptions and Interrupts

- **Exception**
  - An unscheduled event that disrupts program execution
- **Interrupt**
  - An exception that comes from outside of the processor

Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Causes of Exceptions

- **Asynchronous (external interrupt)**
  - input/output device service request
  - timer expiration
  - power failure
  - hardware malfunction
- **Synchronous (internal exception = trap)**
  - undefined opcode
  - privileged instruction
  - arithmetic overflow
  - misaligned memory access
  - virtual memory exceptions
    - page faults, TLB misses, protection violations
  - software exceptions
    - system calls ... **`SVC`** or **`int`** instruction

How Exceptions Are Handled in the MIPS Architecture

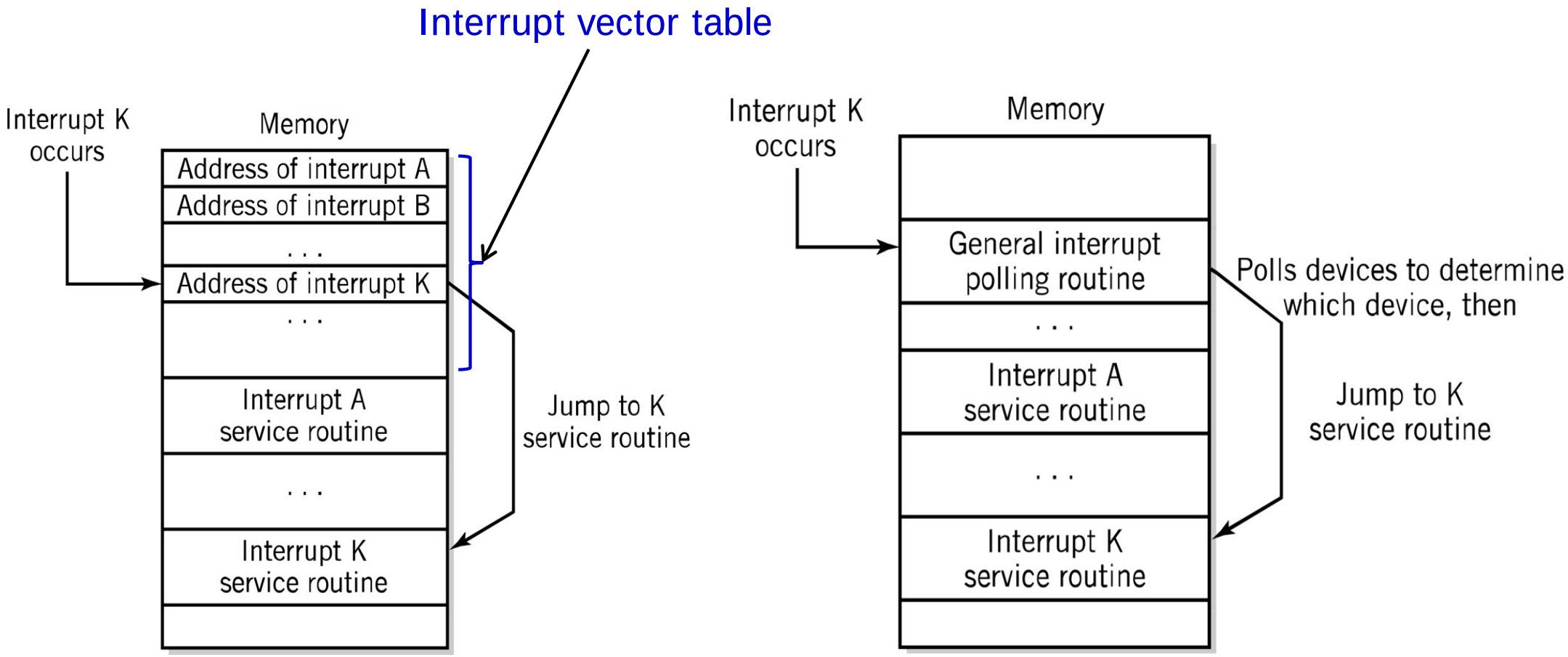
- In MIPS, exceptions managed by a System Control Coprocessor (CP0)

1. Save address of the offending (or interrupted) instruction + 4
   - In MIPS, Exception Program Counter (EPC) ← PC + 4
2. Transfer control to OS at some specified address
   - Read Cause register, and transfer to relevant handler
3. OS can then take the appropriate action
   - If restartable
     - Take corrective action
     - use EPC to return to program
   - Otherwise
     - Terminate program
     - Report error using EPC, cause, ...

Communicating the Reason for an Exception

- **Non-vectored interrupt**
  - Single interrupt handler shared by all interrupts
  - Polling in the interrupt handler
  - Then jump to the device's service routine
- **Vectored interrupt**
  - Interrupt vector
    - Memory address of an interrupt handler or index of the interrupt vector table (Interrupt Descriptor Table in x86)
  - Interrupting device supplies interrupt vector to CPU
  - Interrupt vector is used to generate the address of the handler routine for the interrupt

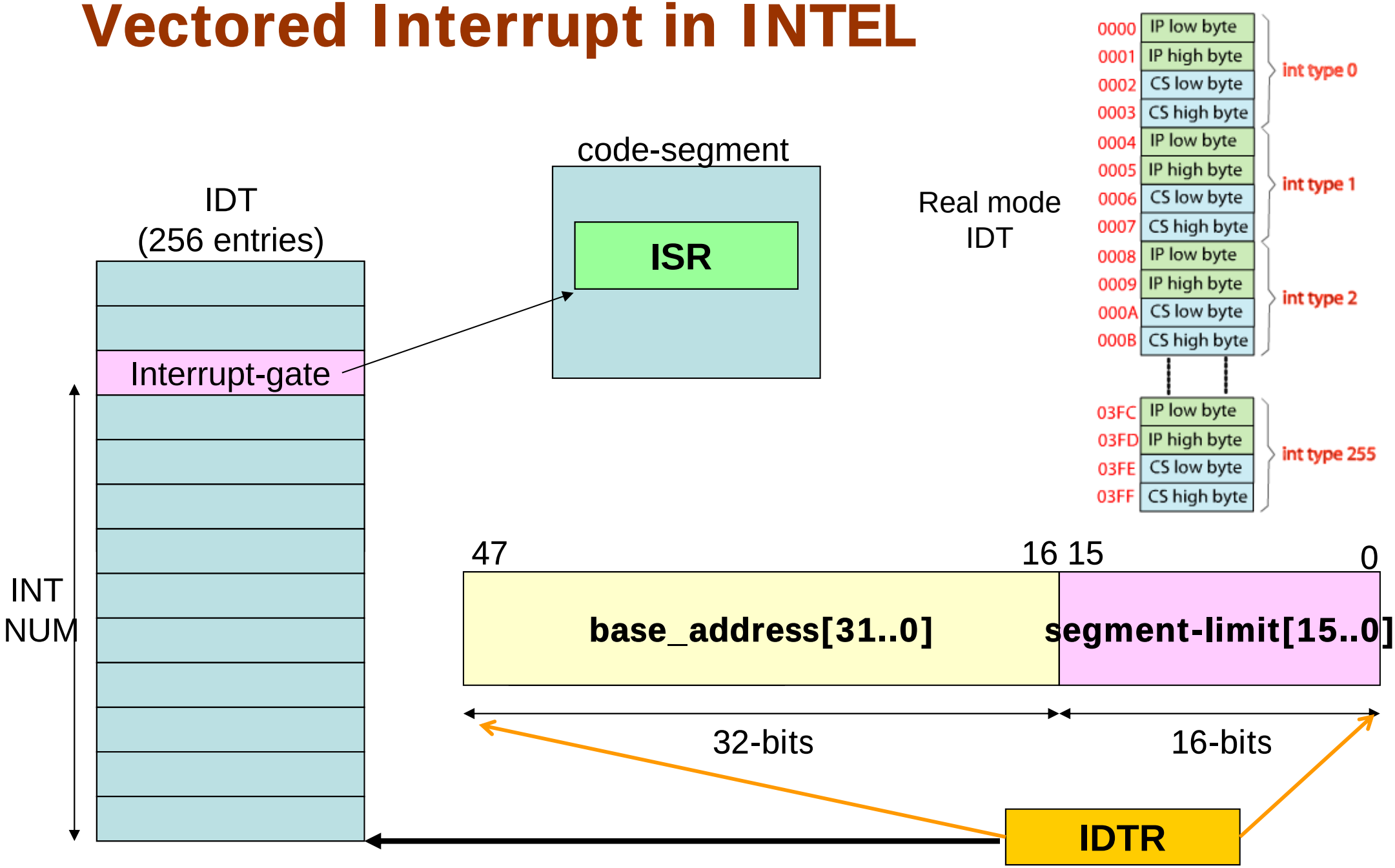
Vectored vs. Polled



Responsibilities of Hardware and OS

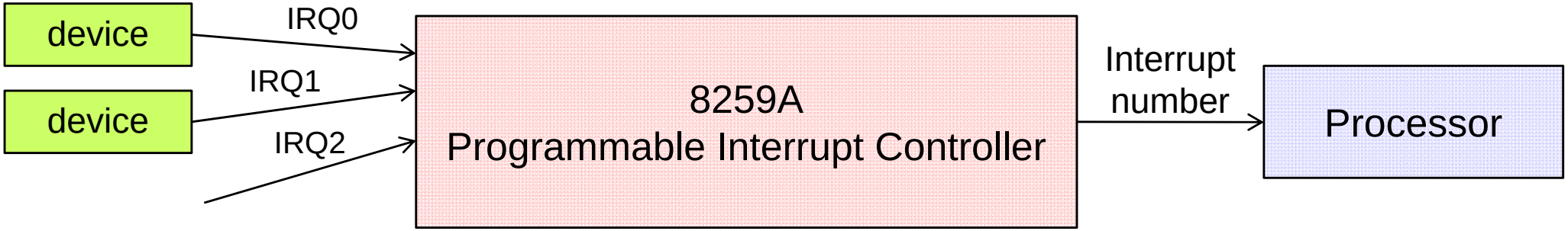
- Hardware saves the current PC and status flags.
- Hardware disables all interrupts.
- Hardware determines cause of the interrupt. (vectored)
- Hardware loads new PC and flags.
- OS saves the registers and interrupt masks.
- OS determines cause of the interrupt. (nonvectored)
- OS sets new interrupt masks.
- OS enables interrupts.
- OS services the interrupt. ➡ ISR (interrupt service routine)
- OS restores the registers and interrupt masks.
- OS executes an interrupt return instruction to load saved PC and flag values.

Vectored Interrupt in INTEL



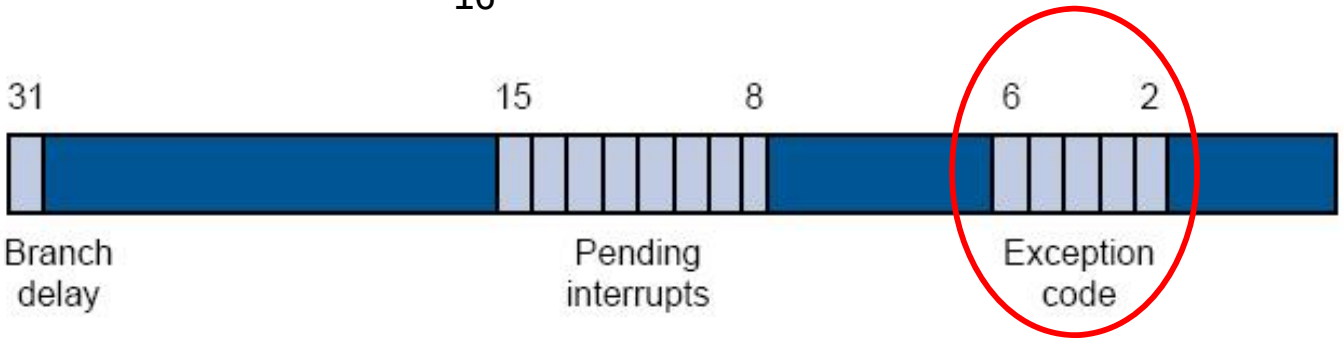
Interrupt Descriptor

- **Interrupt Descriptor Table (IDT)**
  - Interrupt vector table
  - Up to 256 interrupt vectors, 8 bytes/vector → 2KB
  - INT_NUM between 0x0 and 0x1F are reserved for exceptions.
- **IDT base Register (IDTR)**
  - Store the physical base address and the length of the IDT
- **`int x` instruction**
  - Generating a software interrupt
  - x = interrupt number (0 ~ 255)



MIPS Interrupts

- Non-vectored interrupt
  - Using status register (e.g. Cause register)
  - PC ← 8000 0180_{16}



- Vectored interrupt
  - Jump address is determined by the cause of the exception

Exception type	Exception vector address (in hex)
Undefined instruction	$8000\ 0000_{hex}$
Arithmetic overflow	$8000\ 0180_{hex}$

Exceptions in a Pipelined Implementation

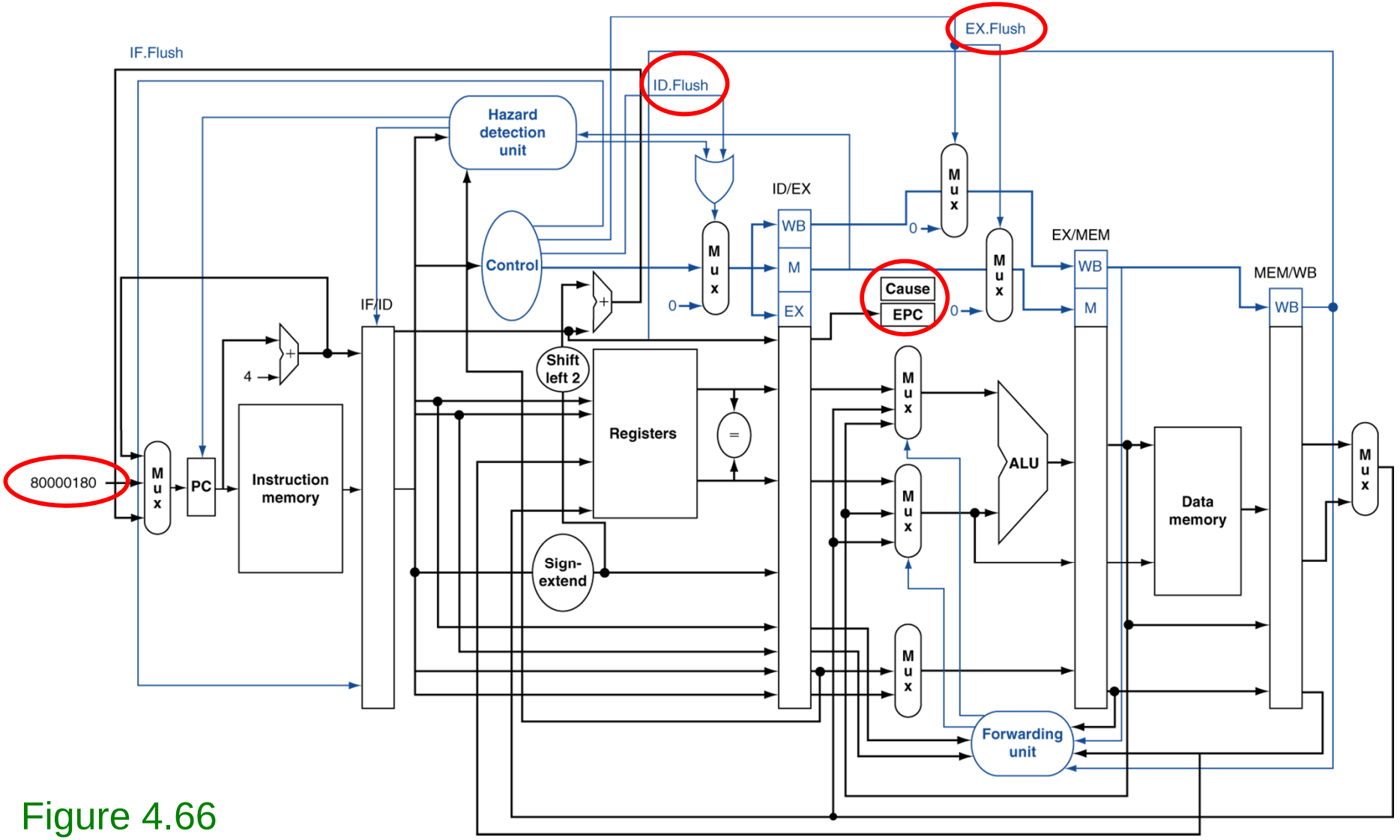


Figure 4.66

Example: Exception in a Pipelined Computer

- **Program :**

```
40hex   sub   $11, $2, $4
44hex   and   $12, $2, $5
48hex   or    $13, $2, $6
4chex   add   $1,  $2, $1
50hex   slt   $15, $6, $7
54hex   lw    $16, 50($7)
```

- **Exception handling program :**

```
80000180hex   sw   $26, 1000($0)
80000184hex   sw   $27, 1004($0)
```

- **Overflow exception in the `add` instruction**

[Answer] - Clock 6: Overflow Detection

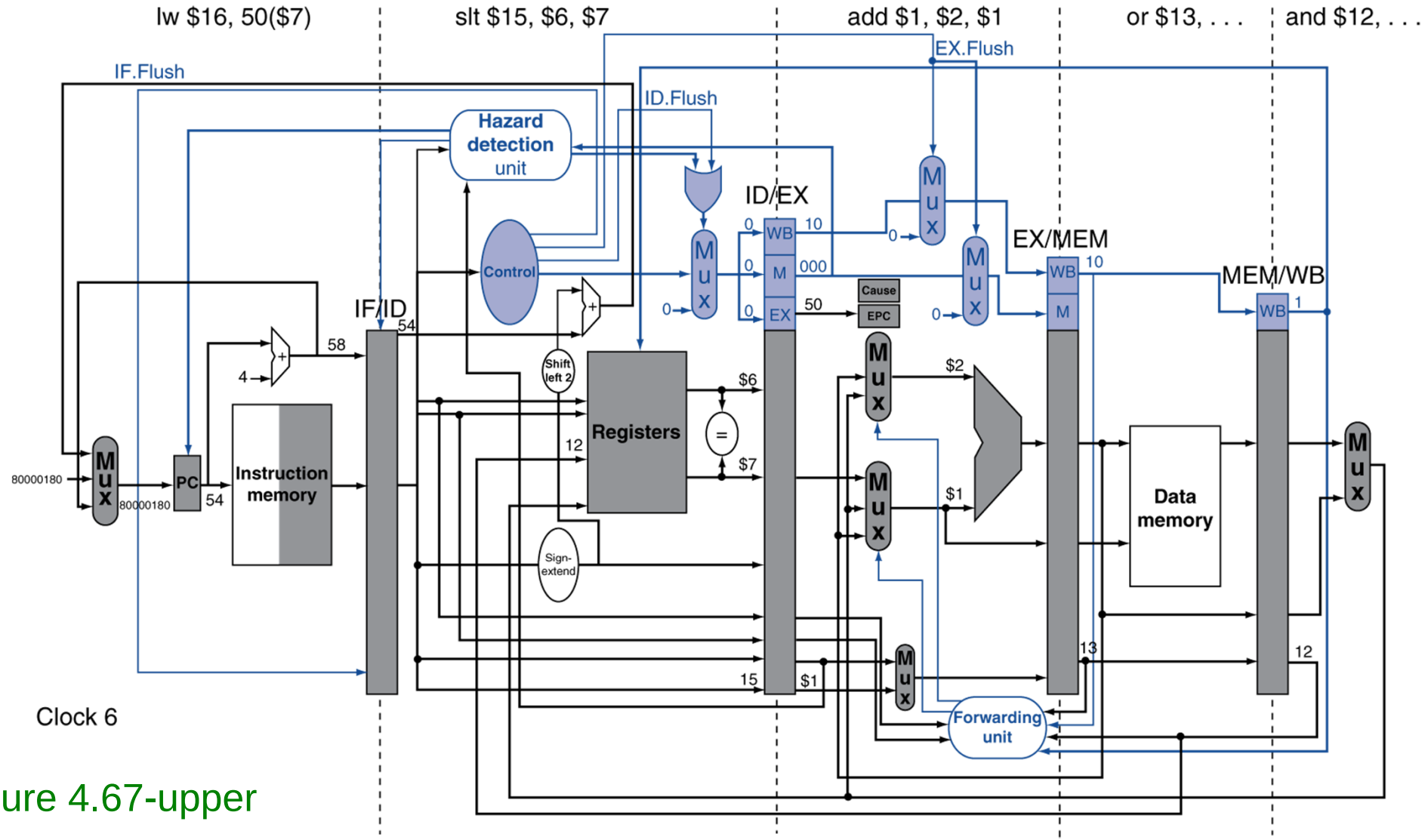


Figure 4.67-upper

[Answer] - Clock 7: Flushing Instructions

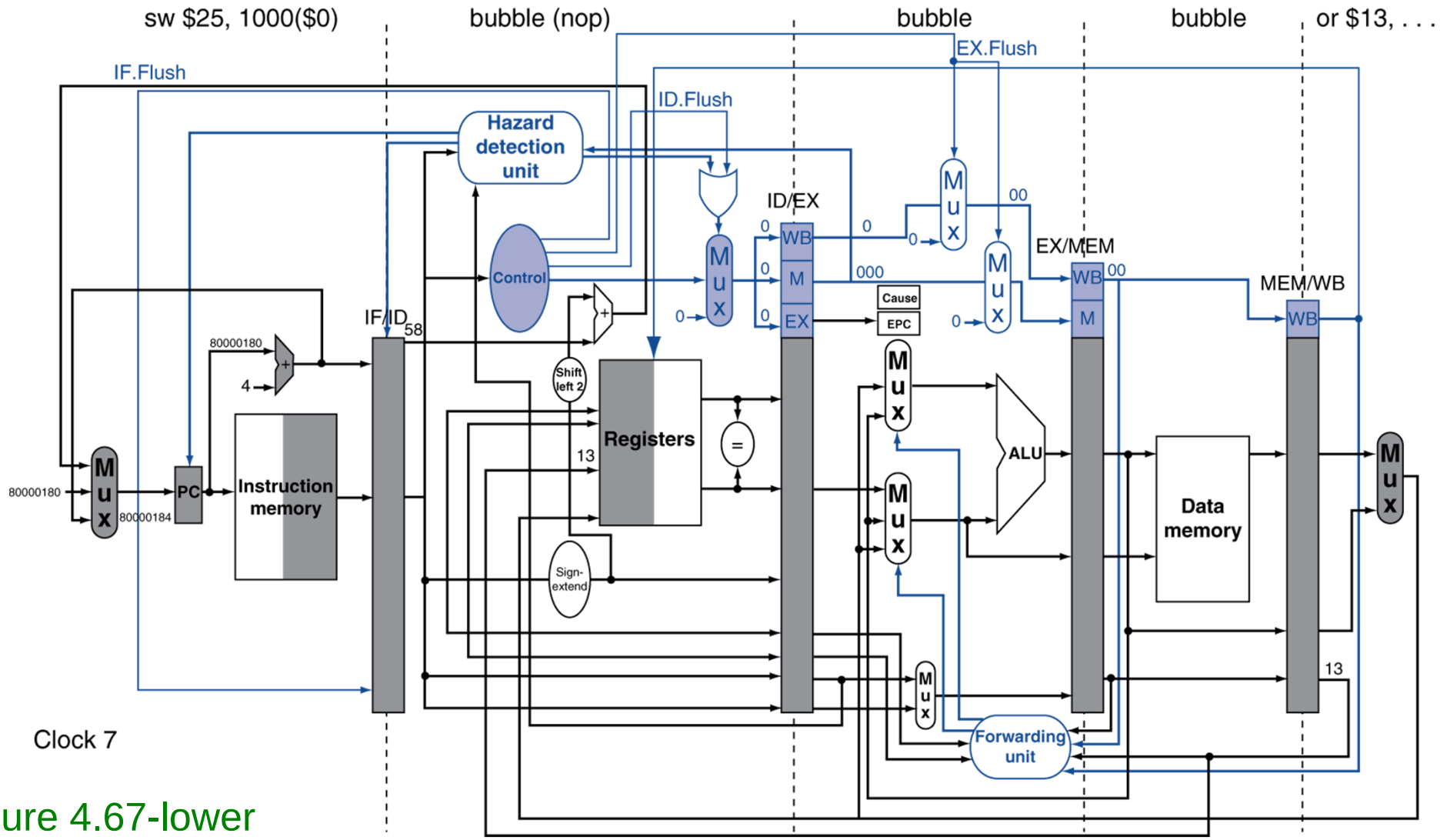


Figure 4.67-lower

4.10 Parallelism via Instruction

- **Instruction level parallelism (ILP)**
  - Executing multiple instructions in parallel
  - A measure of how many of the operations in a computer program can be performed simultaneously
- **Static multiple issue**
  - Very Long Instruction Word (VLIW)
  - Explicitly Parallel Instruction Computer (EPIC)
- **Dynamic multiple issue**
  - Superscalar processors
  - Dynamic pipeline scheduling

Intel Processors

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	2-4	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

Figure 4.73

4.11 Real Stuff: The ARM Cortex-A8 and Intel Core i7 Pipelines

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	14	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	2-level	2-level
1st level caches / core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level cache / core	128–1024 KiB	256 KiB
3rd level cache (shared)	–	2–8 MiB

Figure 4.74