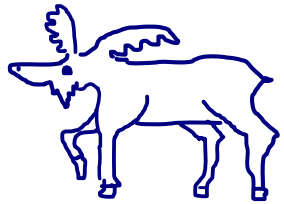
Lecture 19
Exceptions

Byung-gi Kim
School of Computing
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
  - ( ) events requiring change in flow of control
  - ( )-generated function calls

Exceptions and Interrupts

- **Exception**
  - An unscheduled event that disrupts program execution
- **Interrupt**
  - An exception that comes from ( ) 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

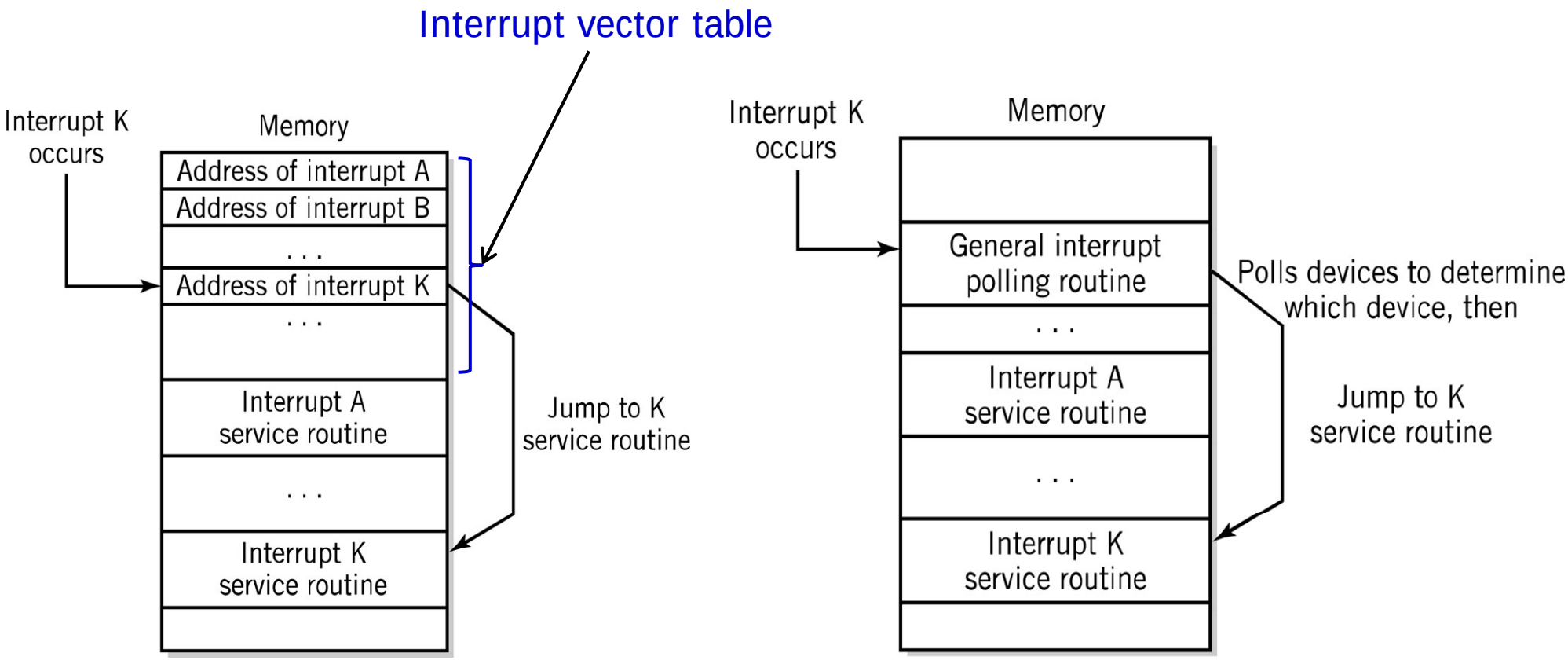
- In MIPS, exceptions managed by a System Control Coprocessor (CP0)

1. Save address of the offending (or interrupted) instruction + 4
   - In MIPS, Exception Program Counter (EPC) ← PC
2. Transfer control to OS at some specified address
   - Read cause, 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
  - ( ) in the interrupt handler
  - Then jump to the device's service routine
- **Vectored interrupt**
  - ( )
    - 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

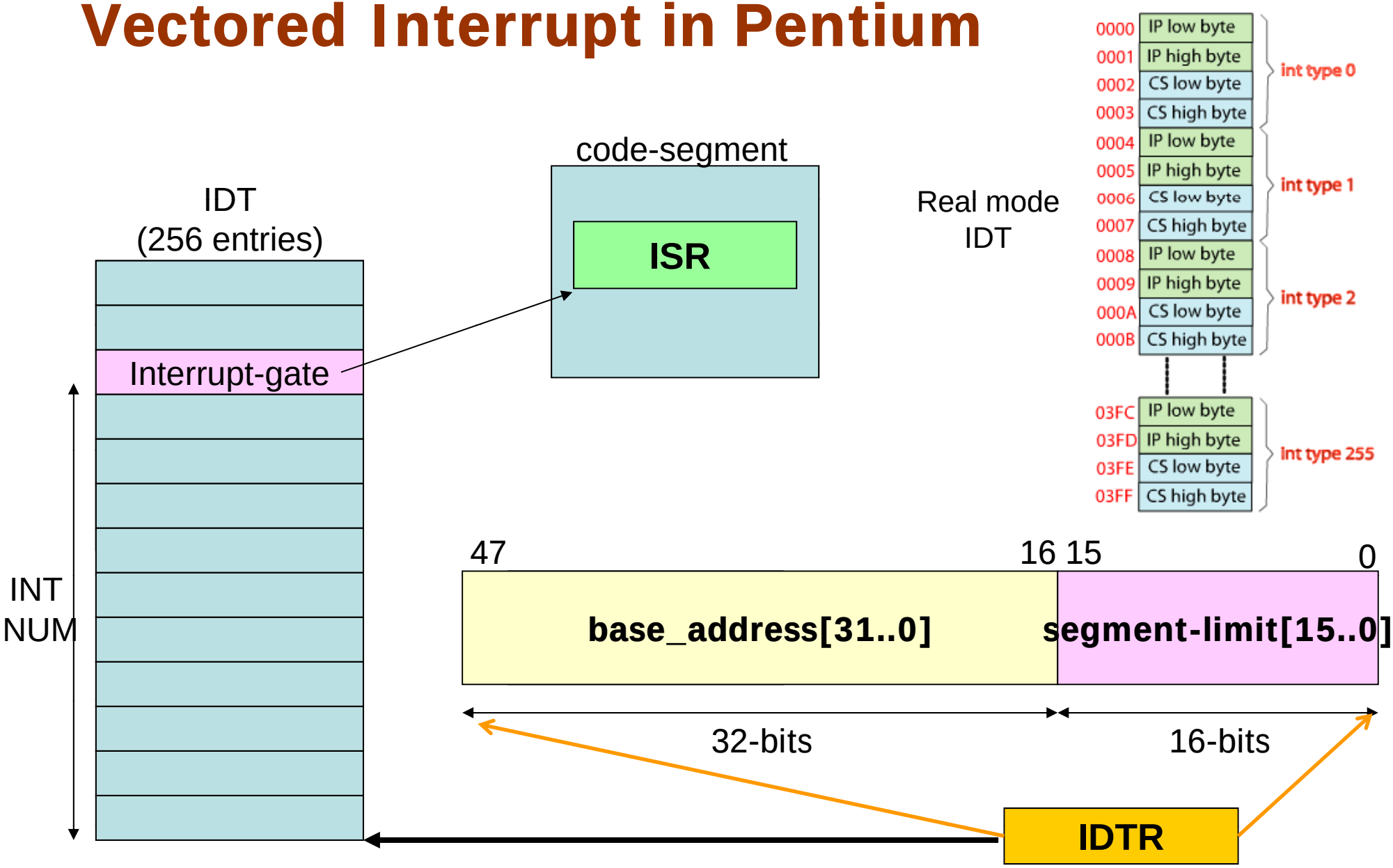
Vectored vs. Polled



Responsibilities of Hardware and OS

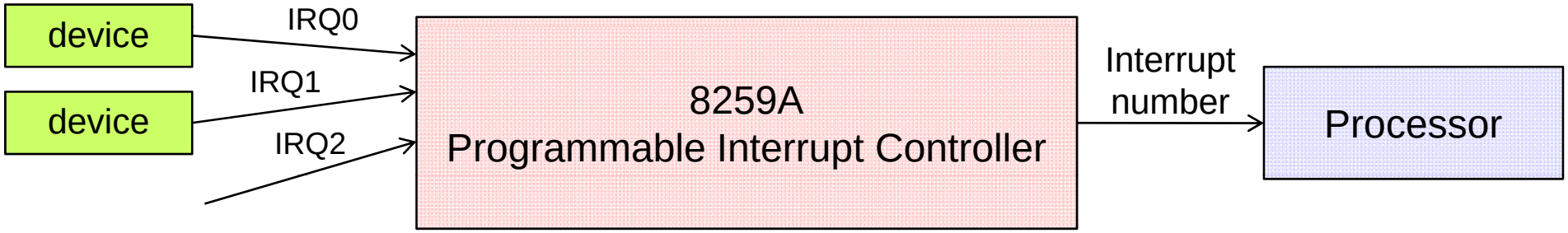
- Hardware saves the current PC and status flags.
- Hardware disables all interrupts.
- Hardware determines cause of the interrupt. (vectored)
- Hardware loads new PC and flags.
- OS saves the registers and interrupt masks.
- OS determines cause of the interrupt. (nonvectored)
- OS sets new interrupt 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 Pentium



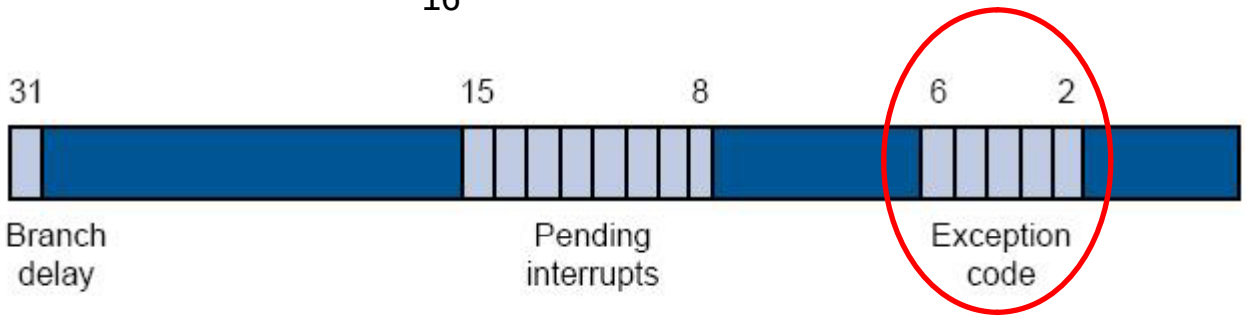
Interrupt Descriptor

- **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

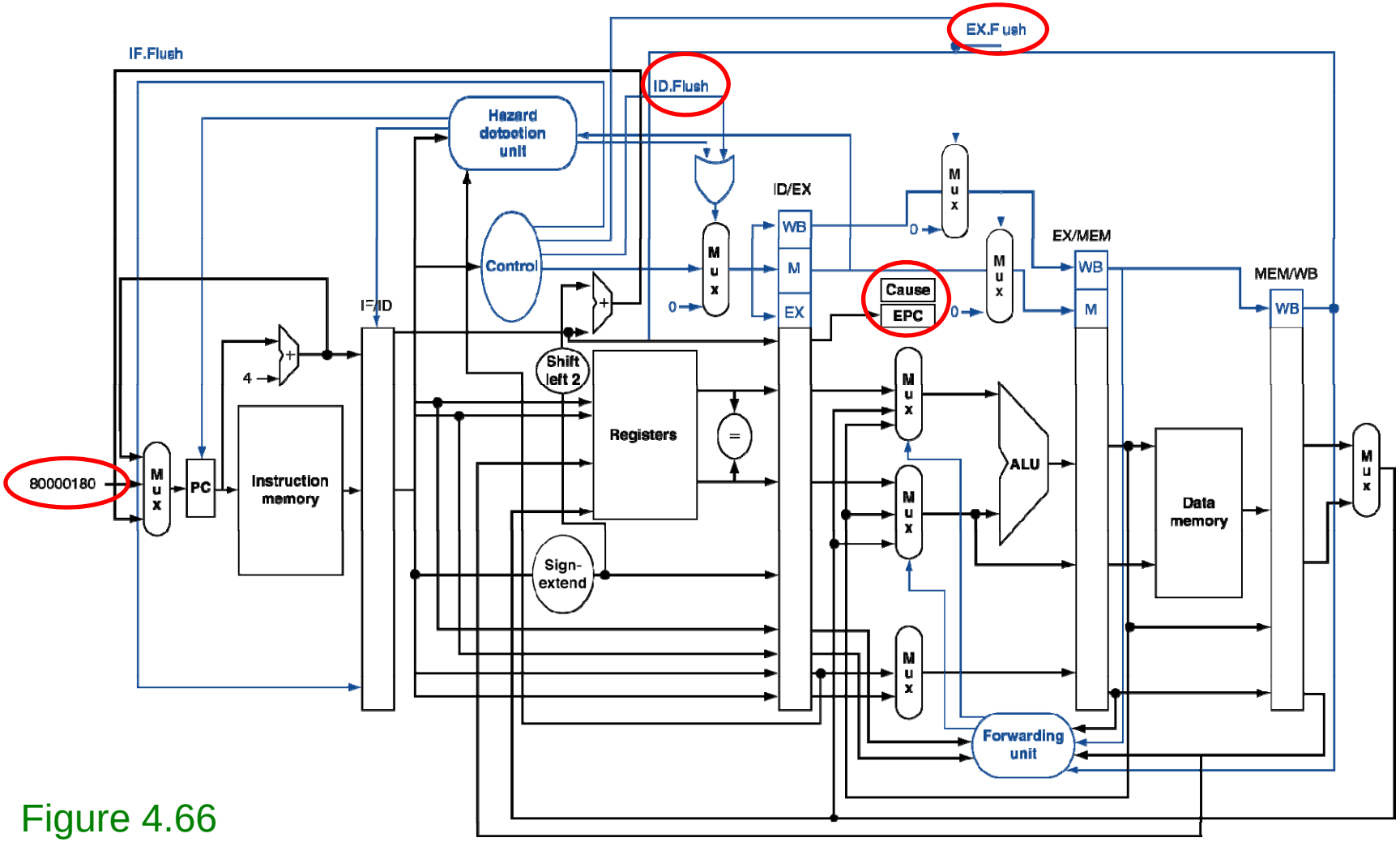


Figure 4.66

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

- **Overflow exception in the add instruction**

[Answer] - Clock 6: Overflow Detection

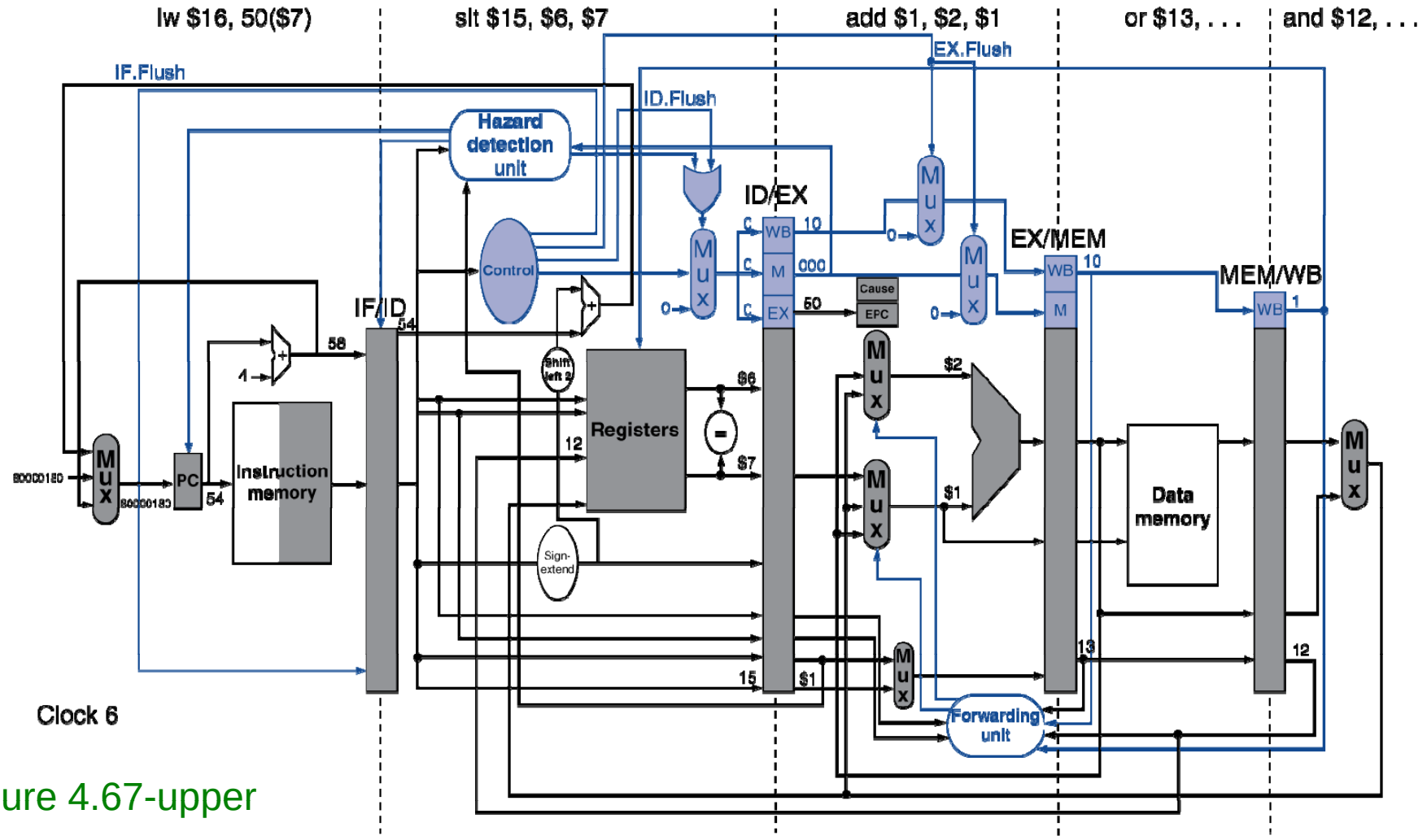


Figure 4.67-upper

[Answer] - Clock 7: Flushing Instructions

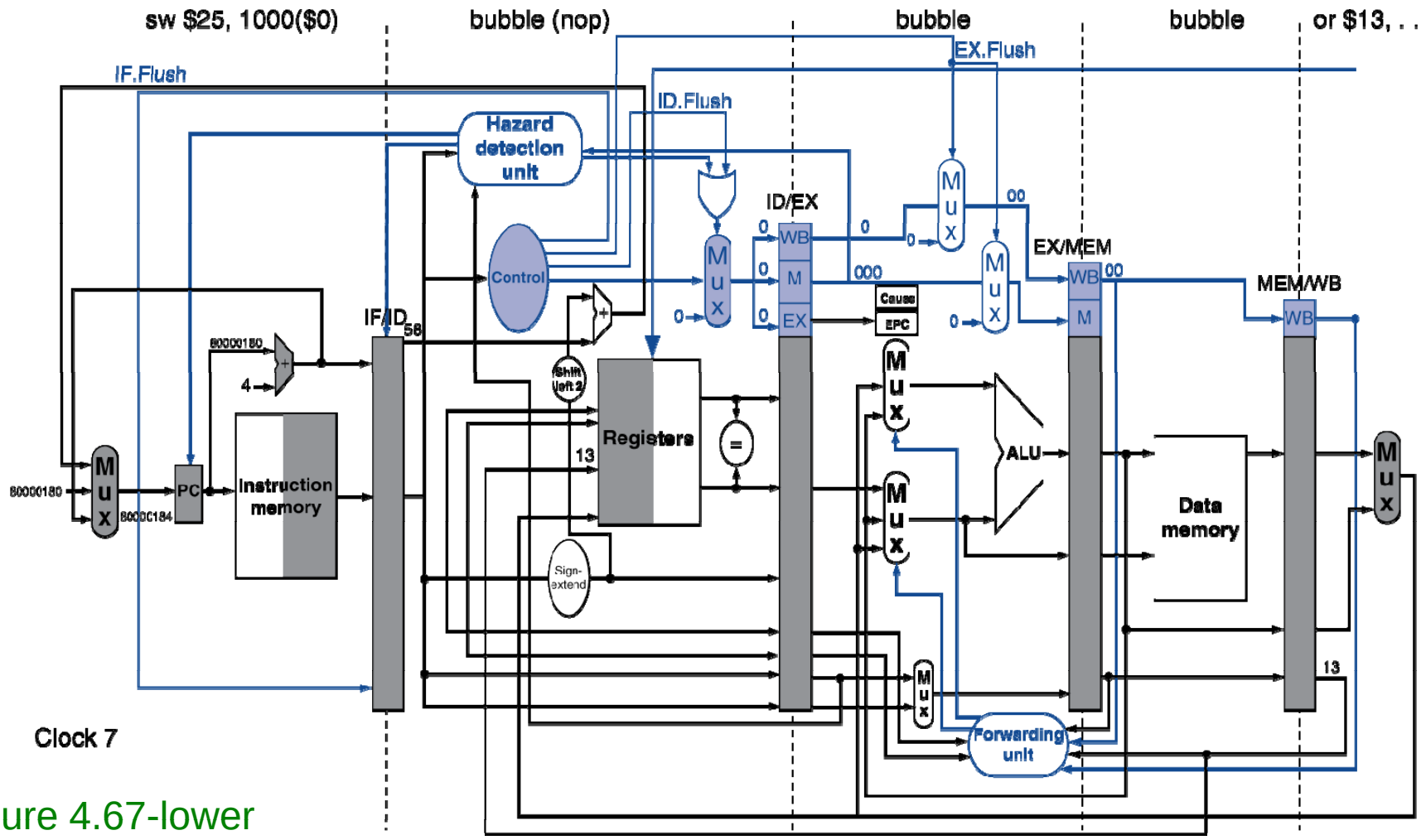


Figure 4.67-lower

4.10 Parallelism and Advanced ILP

- **Instruction level parallelism (ILP)**
  - Executing multiple instructions in parallel
- **Static multiple issue**
  - Very Long Instruction Word (VLIW)
  - Explicitly Parallel Instruction Computer (EPIC)
- **Dynamic multiple issue**
  - Superscalar processors
  - Dynamic pipeline scheduling

Real Products

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
AMD Phenom II	2008	3700MHz	13(I) 16(FP)	3	Yes	6	125W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W
MIPS R10000	1996	200MHz	7	4	Yes	1	
MIPS32 1074K	2010	1500MHz	15	4	Yes	4	
ARM7			3				
ARM11			8			1	
ARM Cortex-A15 MPCore	2011	2500MHz	15 (I) 17~25(FP)	3	Yes	1~8	

4.11 Real Stuff: AMD Opteron X4(Barcelona)

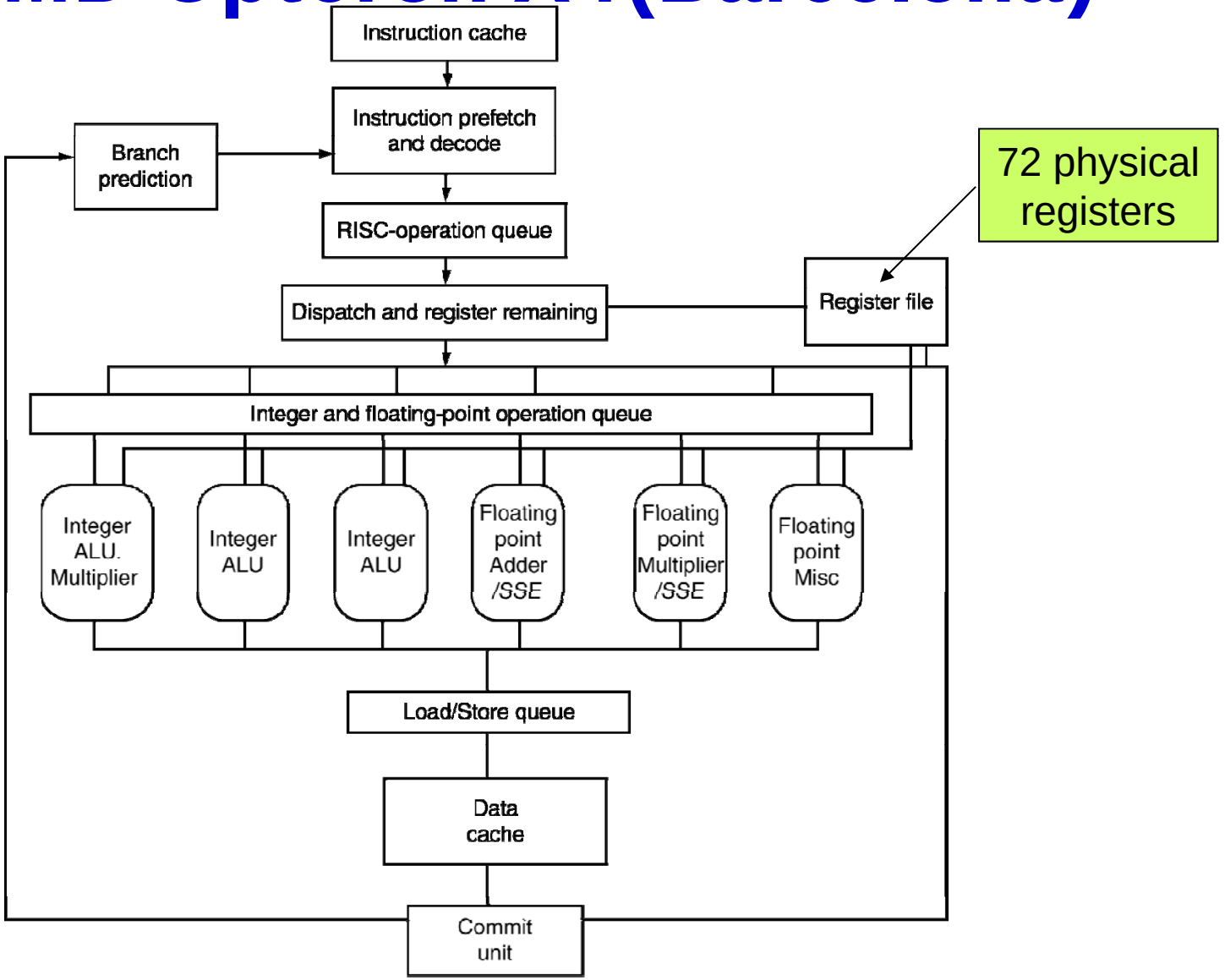
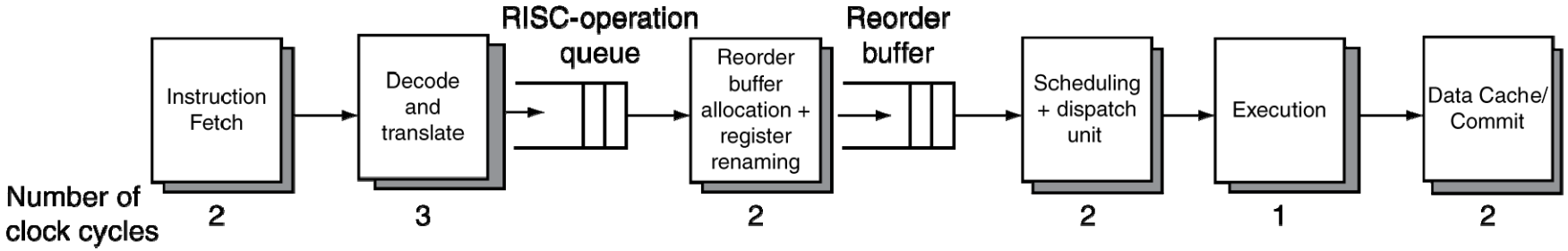


Figure 4.74

The Opteron X4 Pipelines

- **Integer pipeline**
  - 12 stages



- **Floating-point pipeline**
  - 17 stages