

CSE 141 – Computer Architecture Spring 2005

Lectures 11 Exceptions and Introduction to Pipelining

Pramod V. Argade
May 4, 2005

Announcements

- **Reading Assignment**
 - Sections 5.6, 5.9 The Processor Datapath and Control
 - Section 6.1, Enhancing Performance with Pipelining
- **Homework 2:**
 - None!
- **Quiz**

When: Mon., May 16th, First 10 minutes of the class
Topic: Pipeline Hazards, Chapter 6 **Need:** Paper, pen

Course Schedule

Lecture #	Date	Day	Lecture Topic	Quiz Topic	Homework Due
1	3/28	Monday	Introduction, Ch. 1	-	-
2	3/30	Wednesday	Performance, Ch. 4	-	-
3	4/4	Monday	ISA, Ch. 2	Performance	#1
4	4/6	Wednesday	Arithmetic, Ch. 3	-	-
5	4/11	Monday	Arithmetic, Ch. 3	ISA	#2
6	4/13	Wednesday	Single cycle CPU, Ch. 5	-	-
7	4/18	Monday	Single cycle CPU, Ch. 5	Arithmetic	#3
8	4/20	Wednesday	Multi-cycle CPU, Ch. 5	-	-
9	4/25	Monday	Multi-cycle CPU, Ch. 5	Single Cycle CPU	#4
10	4/27	Wednesday	Review for the Midterm	-	-
	5/2	Monday	Mid-term Exam Center 101	-	-
11	5/4	Wednesday	Exceptions, Ch. 5 and Pipelining, Ch. 6	-	-
12	5/9	Monday	Pipelining, Ch. 6	-	-
13	5/11	Wednesday	Data and control hazards, Ch. 6	-	-
14	5/16	Monday	Data and control hazards, Ch. 6	Pipeline Hazards	#5
15	5/18	Wednesday	Memory & cache design, Ch. 7	-	-
16	5/23	Monday	Memory & cache design, Ch. 7	Cache	#6
17	5/25	Wednesday	Virtual Memory & cache design, Ch. 7	-	-
No Class	5/30	Monday	Memorial Day Holiday	-	-
18	6/1	Wednesday	Course Review	-	-
	6/10	Friday	Final Exam	-	-

Exceptions

- There are two sources of non-sequential control flow in a processor
 - Explicit branch and jump instructions
 - Exceptions
- *Branches* are synchronous and deterministic
- *Exceptions* are typically asynchronous and non-deterministic
- Guess which is more difficult to handle?

(*Control flow* refers to the movement of the program counter through memory)

Exceptions and Interrupts

- The terminology is not consistent, but we'll refer to
 - *Exceptions* as any unexpected change in control flow
 - *Interrupts* as any externally-caused exception

So then, what is:

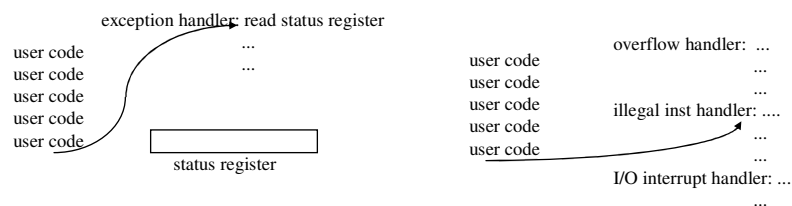
- Arithmetic overflow
- Divide by zero
- I/O device signals completion to CPU
- User program invokes the OS
- Memory parity error
- Illegal instruction
- Timer signal

For now...

- The machine we've been designing in class can generate two types of exceptions.
 - Arithmetic overflow
 - Illegal instruction
- On an exception, we need to
 - Save the PC (invisible to user code)
 - Record the nature of the exception/interrupt
 - Transfer control to OS

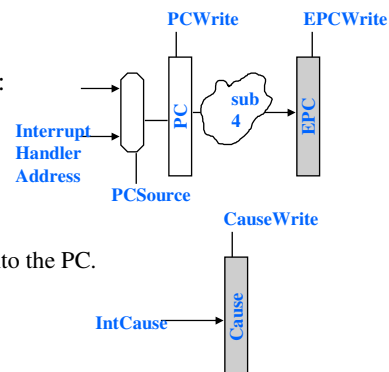
Handling exceptions

- PC saved in EPC (exception program counter), which the OS may read and store in kernel memory
- Two ways of signaling
 - A status *cause register*, and a single exception handler may be used to record the exception and transfer control, or
 - A *vectored interrupt* transfers control to a different location for each possible type of interrupt/exception

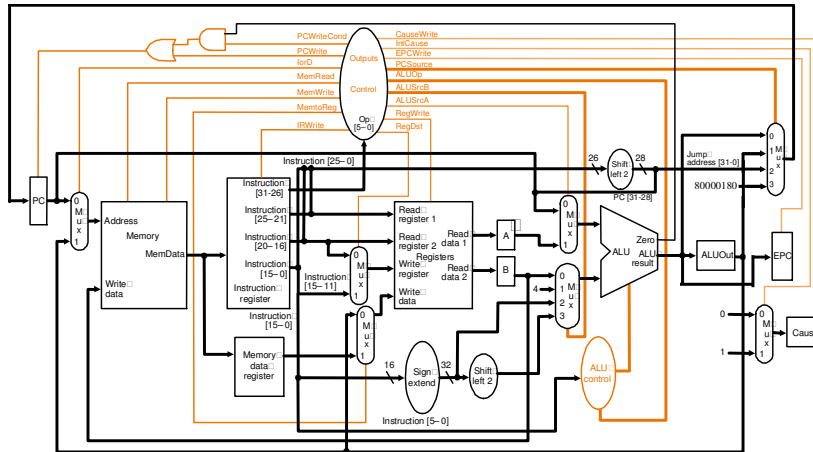


Supporting exceptions

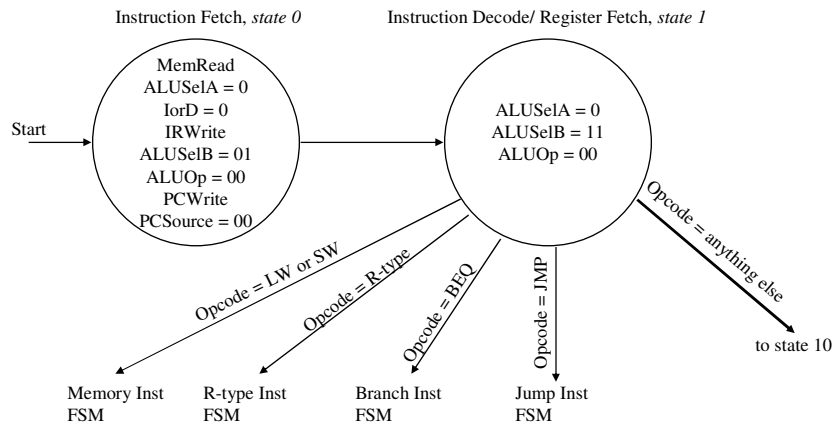
- For our MIPS-subset architecture, we will add two registers:
 - EPC: a 32-bit register to hold the user's PC
 - Cause: A register to record the cause of the exception
 - Undefined inst: Cause = 0
 - Overflow: Cause = 1
- We will also add three control signals:
 - EPCWrite (subtract 4 from PC)
 - CauseWrite
 - IntCause
- Need to force PC
 - Select the interrupt handler address into the PC.



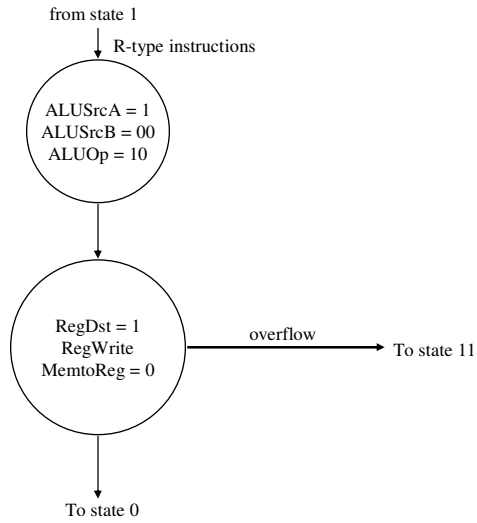
Exception Datapath



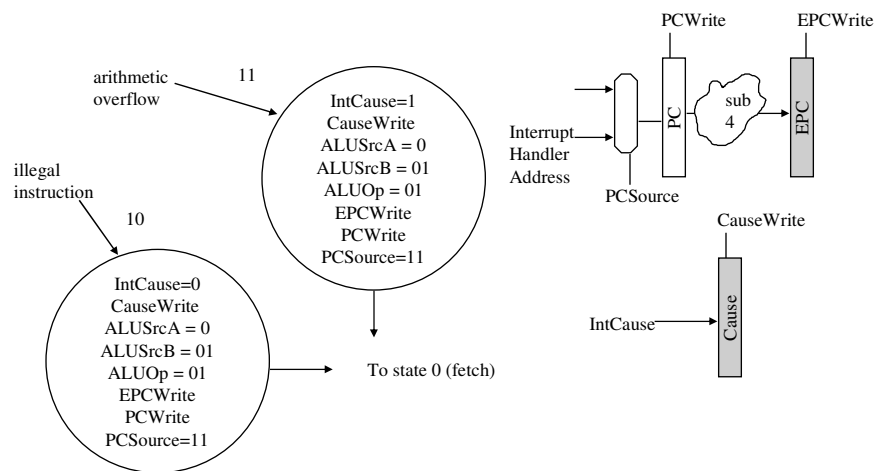
Supporting exceptions in our FSM



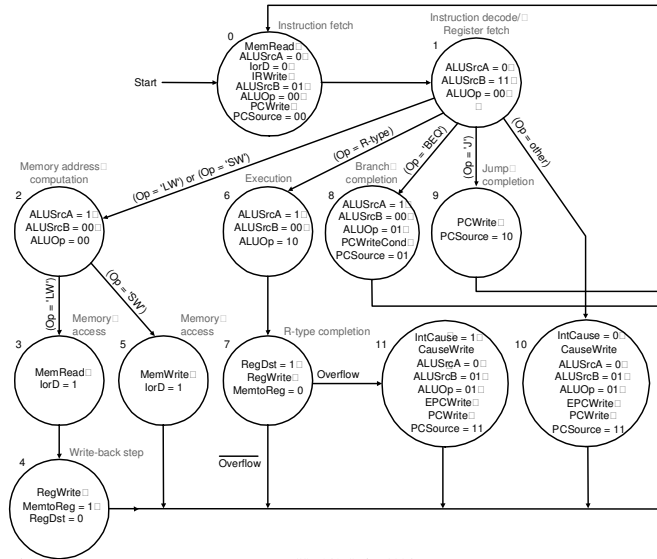
Supporting exceptions in our FSM



State to Support Exceptions



FSM with Exceptions



Pramod V. Argade

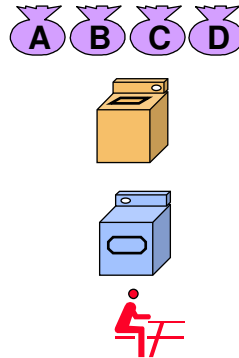
CSE 141, Spring 2005

11-13

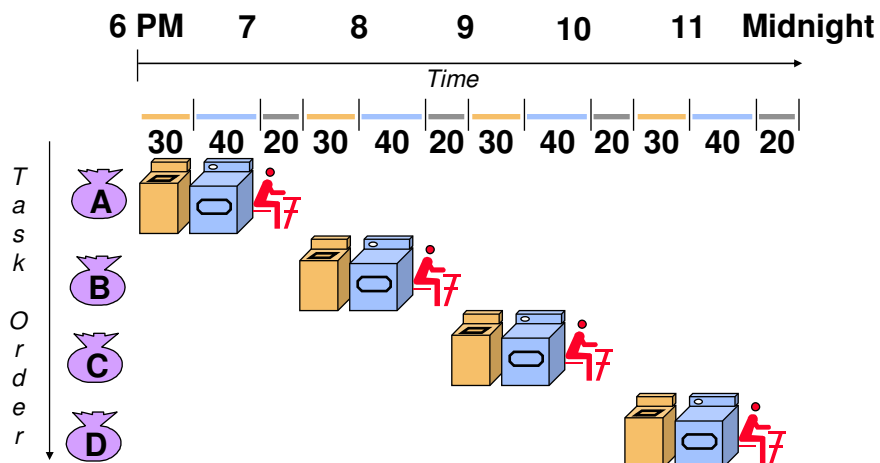
Overview of Pipelining

Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- “Folder” takes 20 minutes

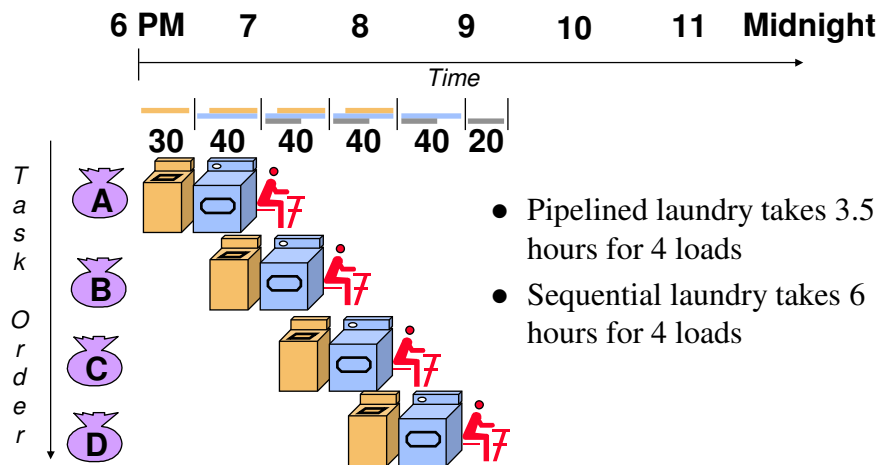


Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



Pipelining Overview

- What is pipelining?
 - Multiple instructions are overlapped in execution
- Notes:
 - Time for completion of a single instruction is not shorter
 - Multiple tasks operate simultaneously
 - Pipelining does not change latency
 - Pipelining increases the throughput
 - Pipelining rate is limited by the slowest stage
 - Potential speedup = number of pipeline stages
 - Time to “fill” pipeline and time to “drain” it reduces speedup

Pipelining

- Requires separable jobs per stage
- Requires separate resources
- Achieves parallelism with replication
- Pipeline efficiency (keeping the pipeline full) critical to performance
- Time between instructions_{pipelined}
= (Time between instructions_{non-pipelined})/(# Pipe Stages)
- Fundamentally invisible to the programmer

Pipelining: Instructions Covered

- Only following instructions will be implemented
 - Memory: LW, SW
 - Arithmetic: ADD, SUB, AND, OR, SLT
 - Branch: BEQ

Instruction Class	Instruction Fetch	Register Read	ALU Operation	Data Access	Register Write	Total Time
Load Word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store Word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-Format	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Pipeline Hazards

- What are hazards?
 - Next instruction cannot execute in the following cycle
- Types of hazards
 - Structural hazards
 - Hardware does not support combination of instructions
 - Data hazards
 - One instruction must wait for another to complete
 - Control hazards
 - Need to make a decision based on the result of one instruction while others are executing

Announcements

- **Reading Assignment**
 - Sections 5.6, 5.9 The Processor Datapath and Control
Section 6.1, Enhancing Performance with Pipelining
- **Homework 2:**
 - **None!**
- **Quiz**

When: Mon., May 16th, First 10 minutes of the class
Topic: Pipeline Hazards, Chapter 6 **Need:** Paper, pen