

CSE 141 – Computer Architecture Summer Session 1 2004

Lecture 8 Multi-cycle CPU Part 1

Pramod V. Argade
April 20, 2005

CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hour:
Mon. 5:00 - 6:00 PM (AP&M 5218)

TAs:

Baris Arslan: barslan@cs.ucsd.edu
Chris Roedel: croedel@cs.ucsd.edu
Raid Ayoub: rayoub@cs.ucsd.edu
Leo Porter: leporter@cs.ucsd.edu

Textbook: Computer Organization & Design
The Hardware Software Interface, 3rd Edition.
Authors: Patterson and Hennessy

Web-page: <http://www.cse.ucsd.edu/classes/sp05/cse141>

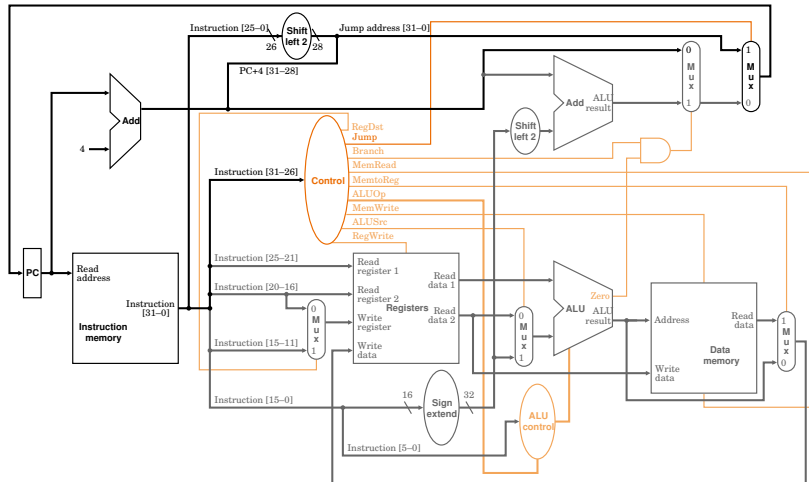
Announcements

- **Discussions Sections for 141:**
 - Fridays, 10:00 - 10:50 am, Peterson Hall 104 (Chris)
 - Fridays, 2:00 - 2:50 pm, Peterson Hall 104 (Leo)
- **Reading Assignment**
 - Chapter 5. Processor: Datapath and Control
 - Sec. 5.1 - 5.4
- **Homework 4: Due Mon., April 25th in class**
 - 5.2, 5.8, 5.9, 5.10, 5.13, 5.20, 5.22, 5.28
- **Quiz**
 - When:** Mon., April 25th, First 10 minutes of the class
 - Topic:** Single Cycle CPU, Chapter 5 **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	-	-
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	-	-

Single Cycle CPU



Single-cycle CPU Observations

- All instructions take the same amount of time
 - Make common case fast!
- Assume following delays
 - Memory units: 200 ps, ALU & Adders: 100 ps, Reg. File: 50 ps

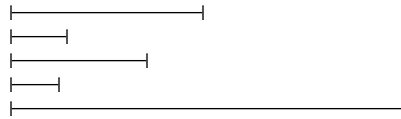
Time specified in ps

Instruction Class	Instruction Memory	Register Read	ALU Operation	Data Memory	Register Write	Total (ps)
ALU Type	200	50	100	0	50	400
Load Word	200	50	100	200	50	600
Store word	200	50	100	200		550
Branch	200	50	100			350
Jump	200					200

- Longest instruction (LW) determines the clock cycle for the machine!
- Single-cycle implementation is inefficient in performance and HW cost

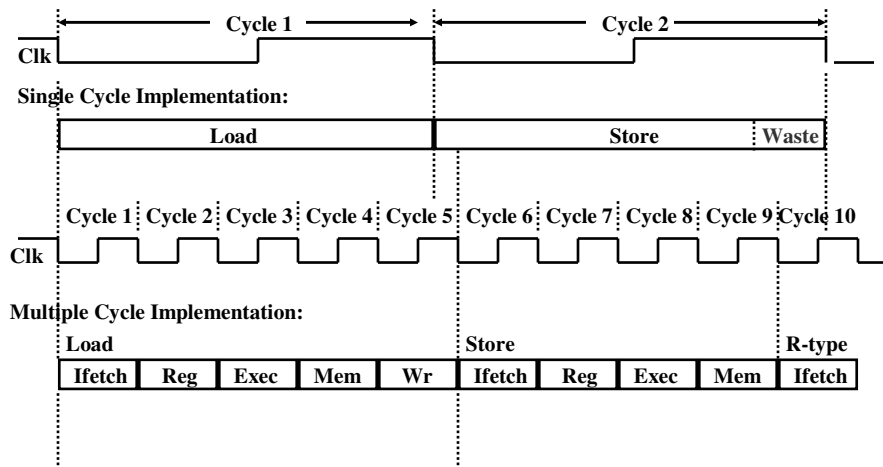
Single Cycle CPU: How to improve the Performance?

- Problem with single-cycle CPU
 - Cycle time long enough to complete the longest instruction
- A solution: Multi-cycle CPU
 - Break up execution into smaller tasks, each task taking a cycle, different instructions requiring different numbers of cycles or tasks



- Other advantages of Multi-cycle CPU
 - Reuse of functional units (e.g., alu, memory)
- Performance = Instructions * CPI * Cycle time

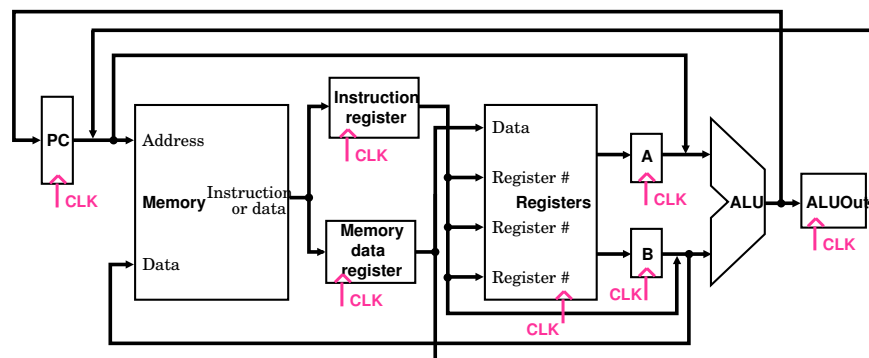
Multicycle CPU



Breaking Execution Into Clock Cycles

- Introduces extra registers when:
 - Signal is computed in one clock cycle and used in another, AND
 - The inputs to the functional block that outputs this signal can change before the signal is written into a state element.
- Significantly complicates control. Why?
- The goal is to balance the amount of work done each cycle.

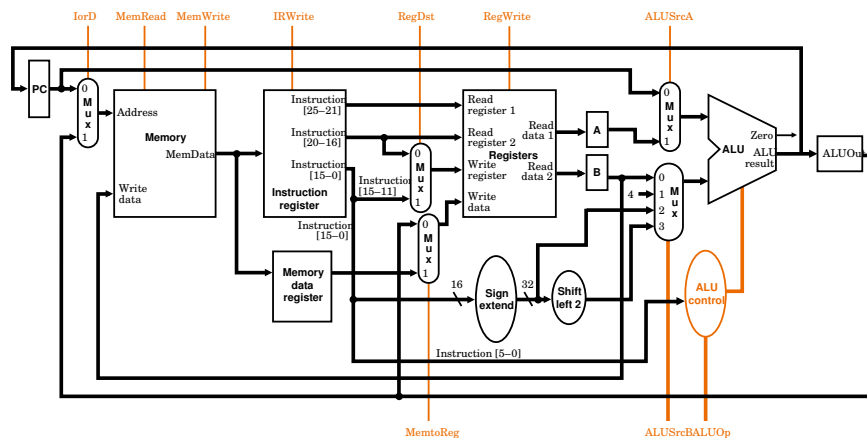
Multi-cycle Datapath: High-level View



Breaking Execution Into Clock Cycles

- Each stage has at most one of following
 - ALU Operation
 - Register file access
 - One memory access
- Save output from above into a temporary data register
 - Written every clock cycle, no write control needed
- We will have five execution steps (not all instructions use all five)
 - fetch
 - decode & register fetch
 - execute
 - memory access
 - write-back
- Perform tasks early if they are not harmful

Multi-cycle Datapath with Control



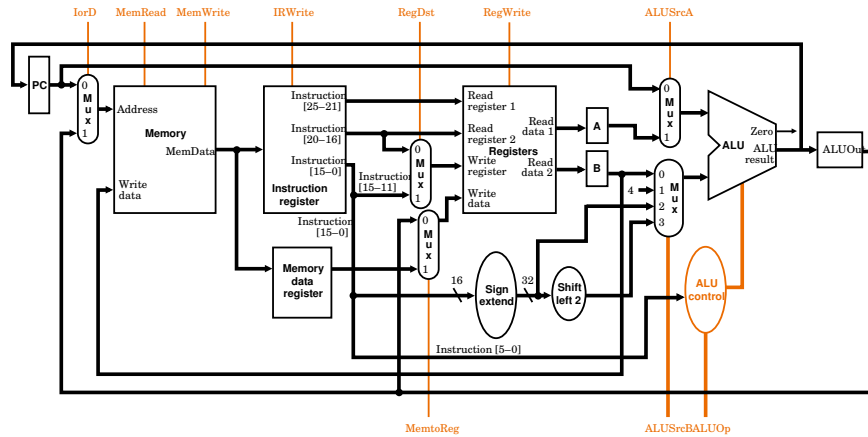
Note:

- I-Mem and D-Mem combined into one memory
- PC adder and PC offset adder have been eliminated

1. Instruction Fetch Cycle (Instruction Independent)

$IR = \text{Memory}[PC]$

$PC = PC + 4$ [may not be the final value of PC]



Pramod Argade

UCSD CSE 141, Spring 2005

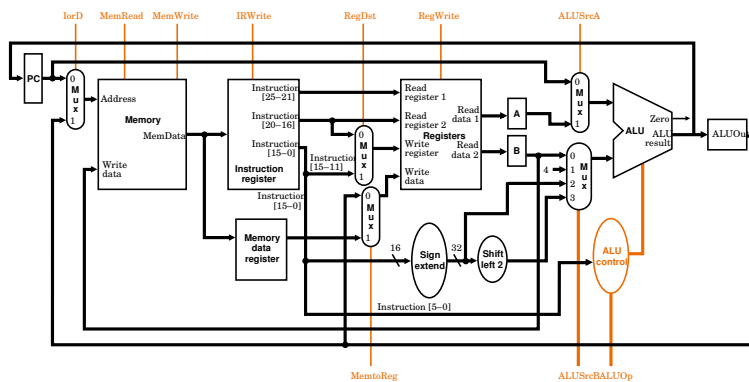
Slide 7-13

2. Instruction Decode and Register Fetch Cycle (Instruction Independent)

$A = \text{Register}[IR[25-21]]$

$B = \text{Register}[IR[20-16]]$

$ALUOut = PC + (\text{sign-extend}(IR[15-0]) \ll 2)$ [May not get used.]



Pramod Argade

UCSD CSE 141, Spring 2005

Slide 7-14

3. Execution, memory address computation, or branch completion

Memory reference (load or store)

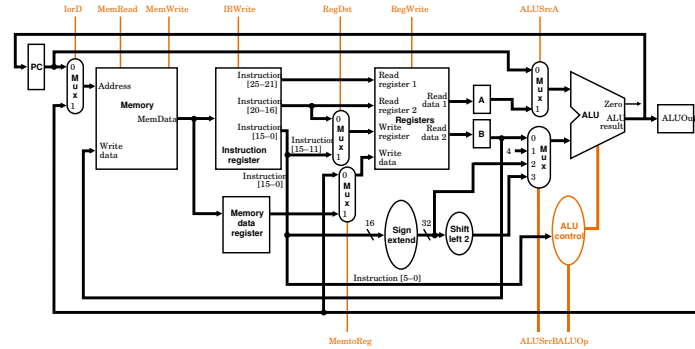
$$\text{ALUOut} = A + \text{sign-extend}(\text{IR}[15-0])$$

R-type

$$\text{ALUOut} = A \text{ op } B$$

Branch (completes in this cycle)

$$\text{if}(A == B) \text{ PC} = \text{ALUOut} \text{ (Add bus from ALUOut to PC)}$$



Pramod Argade

UCSD CSE 141, Spring 2005

Slide 7-15

4. Memory access or R-type completion

Memory reference:

load

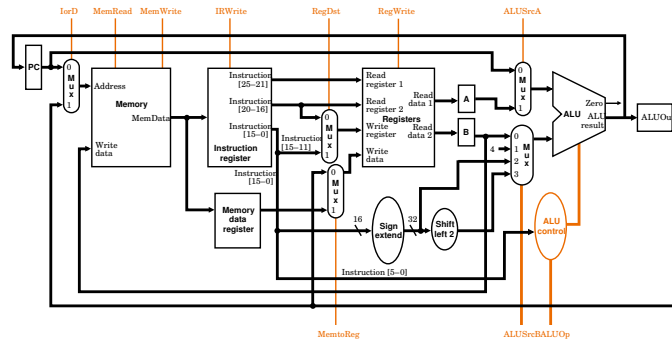
$$\text{MDR} = \text{Memory}[\text{ALUOut}]$$

store

$$\text{Memory}[\text{ALUOut}] = B$$

R-type (Completes in this cycle):

$$\text{Reg}[\text{IR}[15-11]] = \text{ALUOut}$$



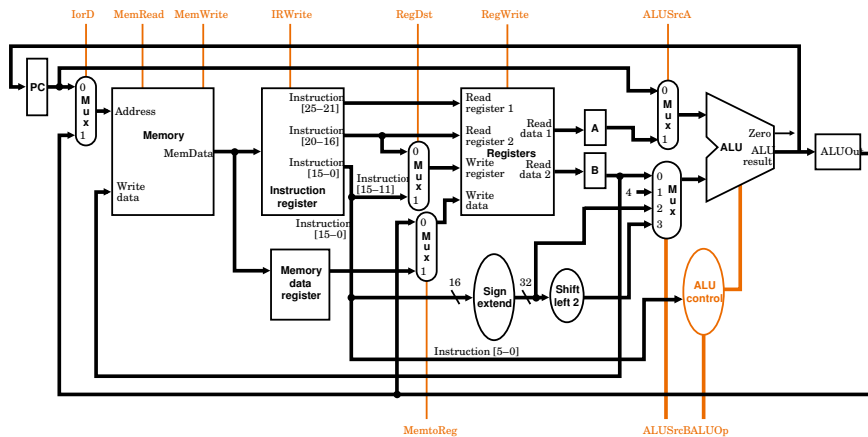
Pramod Argade

UCSD CSE 141, Spring 2005

Slide 7-16

5. Memory Write-Back

$\text{Reg}[\text{IR}[20-16]] = \text{MDR}$ [Completes in this cycle]



Pramod Argade

UCSD CSE 141, Spring 2005

Slide 7-17

Summary of execution steps

Step	R-type	Memory	Branch
Instruction Fetch	IR = Memory[PC] PC = PC + 4		
Instruction Decode/ register fetch	A = Registers[IR[25-21]] B = Registers[IR[20-16]] Target = PC + (sign-extend(IR[15-0]) << 2)		
Execution, address computation, branch completion	ALUoutput = A op B	ALUoutput = A + sign- extend(IR[15-0])	if (A==B) then PC = Target
Memory access or R- type completion	Reg[IR[15-11]] = ALUoutput	memory-data = Mem[ALUoutput] or Mem[ALUoutput] = B	
Write-back		Reg[IR[20-16]] = memory-data	

Pramod Argade

UCSD CSE 141, Spring 2005

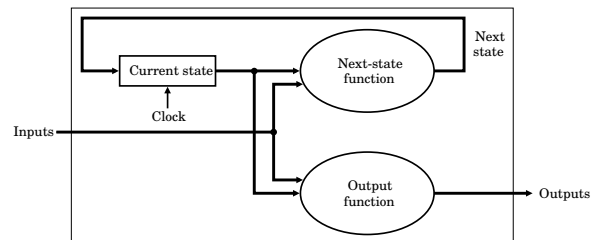
Slide 7-18

Multicycle Control

- Single-cycle control used combinational logic
- Multi-cycle control
 - Need to specify a sequence of controls for each cycle
- FSM defines a succession of states, transitions between states (based on inputs), and outputs (based on state and inputs)
 - Outputs not explicitly asserted are deasserted
 - Multiplexor control is always specified
- First two states same for every instruction, next state depends on opcode.

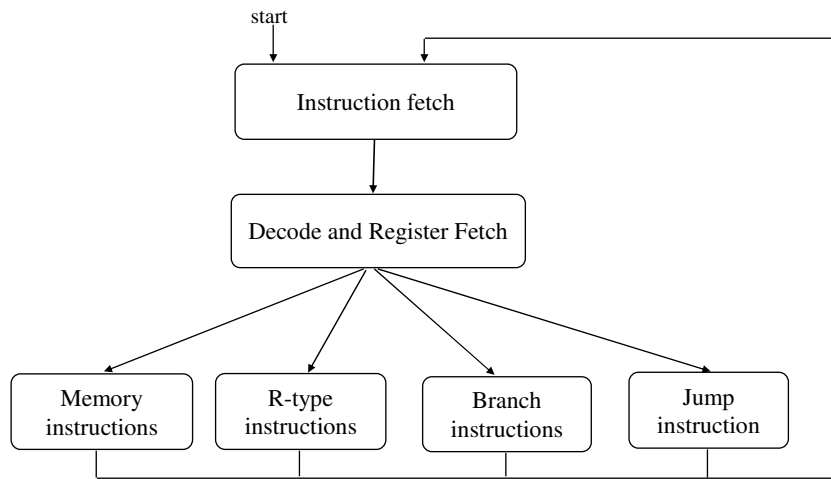
Review: Finite State Machines

- Finite state machines:
 - a set of states and
 - next state function (determined by current state and the input)
 - output function (determined by current state and possibly input)

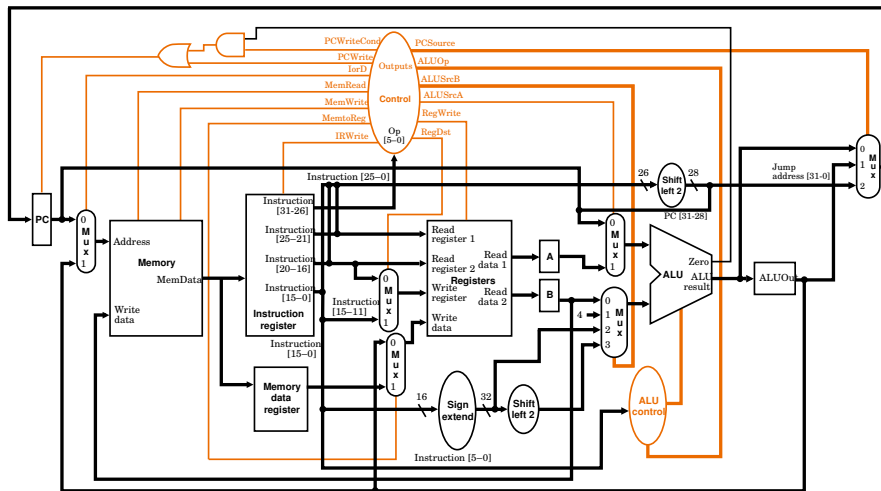


- We'll use a Moore machine (output based only on current state)

Multi-cycle CPU: Control FSM



Complete Datapath and Control



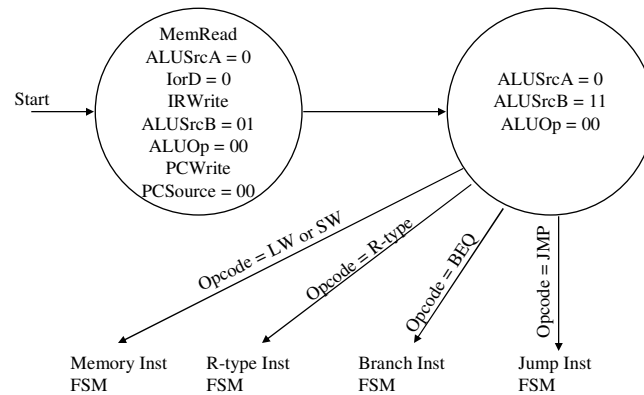
First two states of the FSM

Instruction Fetch, state 0

IR = Memory[PC]
PC = PC + 4

Instruction Decode/ Register Fetch, state 1

A = Register[IR[25-21]]
B = Register[IR[20-16]]
ALUOut = PC + (sign-extend (IR[15-0]) << 2)



Announcements

- **Discussions Sections for 141:**
 - Fridays, 10:00 - 10:50 am, Peterson Hall 104 (Chris)
 - Fridays, 2:00 - 2:50 pm, Peterson Hall 104 (Leo)
- **Reading Assignment**
 - Chapter 5. Processor: Datapath and Control
 - Sec. 5.1 - 5.4
- **Homework 4: Due Mon., April 25th in class**
 - 5.2, 5.8, 5.9, 5.10, 5.13, 5.20, 5.22, 5.28
- **Quiz**
 - When:** Mon., April 25th, First 10 minutes of the class
 - Topic:** Single Cycle CPU, Chapter 5 **Need:** Paper, pen