18-447

Computer Architecture Lecture 6: Multi-cycle Microarchitectures

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2013, 1/28/2013

Reminder: Homeworks

- Homework 1
 - Due today, midnight
 - Turn in via AFS (hand-in directories)
- Homework 2
 - Will be assigned later today. Stay tuned...
 - □ ISA concepts, ISA vs. microarchitecture, microcoded machines

Reminder: Lab Assignment 1

- Due this Friday (Feb 1), at the end of Friday lab
- A functional C-level simulator for a subset of the MIPS ISA
- Study the MIPS ISA Tutorial
 - □TAs will continue to cover this in Lab Sessions this week

Lookahead: Lab Assignment 2

- Lab Assignment 1.5
 - Verilog practice
 - Not to be turned in
- Lab Assignment 2
 - Due Feb 15
 - Single-cycle MIPS implementation in Verilog
 - All labs are individual assignments
 - No collaboration; please respect the honor code

Lookahead: Extra Credit for Lab Assignment 2

- Complete your normal (single-cycle) implementation first, and get it checked off in lab.
- Then, implement the MIPS core using a microcoded approach similar to what we will discuss in class.
- We are not specifying any particular details of the microcode format or the microarchitecture; you can be creative.
- For the extra credit, the microcoded implementation should execute the same programs that your ordinary implementation does, and you should demo it by the normal lab deadline.

Readings for Today

- P&P, Revised Appendix C
 - Microarchitecture of the LC-3b
 - Appendix A (LC-3b ISA) will be useful in following this
- P&H, Appendix D
 - Mapping Control to Hardware
- Optional
 - Maurice Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester Univ. Computer Inaugural Conf., 1951.

Lookahead: Readings for A Next Lecture

- Pipelining
 - □ P&H Chapter 4.5-4.8

Review of Last Lecture: Single-Cycle Uarch

- What phases of the instruction processing cycle does the MIPS JAL instruction exercise?
- How many cycles does it take to process an instruction in the single-cycle microarchitecture?
 - What determines the clock cycle time?
- What is the difference between datapath and control logic?
 - What about combinational vs. sequential control?
- What is the semantics of a delayed branch?
 - Why this is so will become clear when we cover pipelining

Review: Instruction Processing "Cycle"

- Instructions are processed under the direction of a "control unit" step by step.
- Instruction cycle: Sequence of steps to process an instruction
- Fundamentally, there are six phases:
- Fetch
- Decode
- Evaluate Address
- Fetch Operands
- Execute
- Store Result
- Not all instructions require all six stages (see P&P Ch. 4)

Review: Datapath vs. Control Logic

- Instructions transform Data (AS) to Data' (AS')
- This transformation is done by functional units
 - Units that "operate" on data
- These units need to be told what to do to the data
- An instruction processing engine consists of two components
 - Datapath: Consists of hardware elements that deal with and transform data signals
 - functional units that operate on data
 - hardware structures (e.g. wires and muxes) that enable the flow of data into the functional units and registers
 - storage units that store data (e.g., registers)
 - Control logic: Consists of hardware elements that determine control signals, i.e., signals that specify what the datapath elements should do to the data

A Note: How to Make the Best Out of 447?

Do the readings

- P&P Appendixes A and C
- Wilkes 1951 paper
- Today's lecture will be easy to understand if you read these
- And, you can ask more in-depth questions and learn more

Do the assignments early

- You can do things for extra credit if you finish early
- We will describe what to do for extra credit

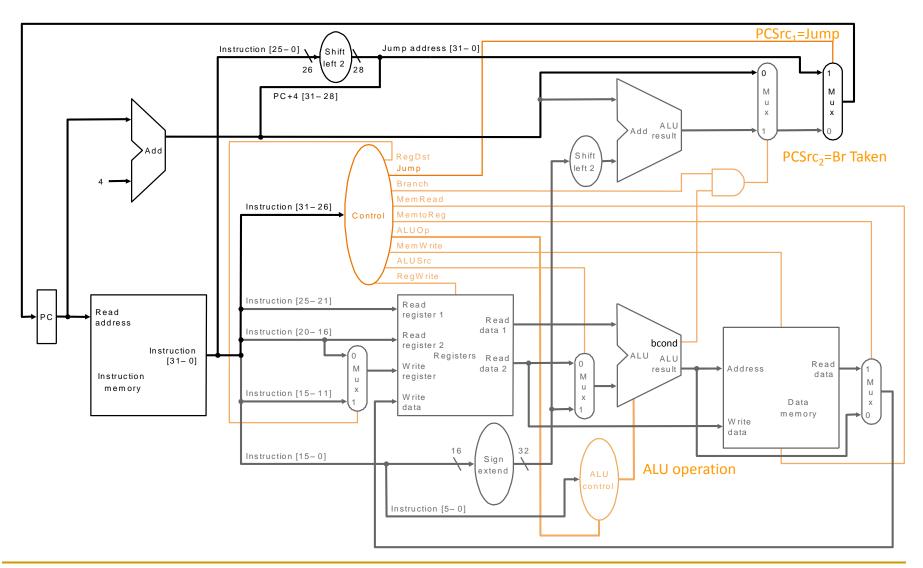
Study the material and buzzwords daily

- Lecture notes, videos
- □ Buzzwords → take notes during class

Today's Agenda

- Finish single-cycle microarchitectures
 - Critical path
- Microarchitecture design principles
- Performance evaluation primer
- Multi-cycle microarchitectures
 - Microprogrammed control

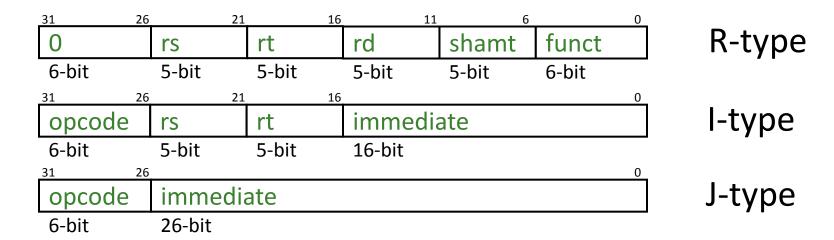
Review: The Full Single-Cycle Datapath



Single-Cycle Control Logic

Single-Cycle Hardwired Control

As combinational function of Inst=MEM[PC]



- Consider
 - All R-type and I-type ALU instructions
 - LW and SW
 - BEQ, BNE, BLEZ, BGTZ
 - J, JR, JAL, JALR

Single-Bit Control Signals

	When De-asserted	When asserted	Equation
RegDest	GPR write select according to rt, i.e., inst[20:16]	GPR write select according to rd, i.e., inst[15:11]	opcode==0
ALUSrc	2 nd ALU input from 2 nd GPR read port	2 nd ALU input from sign- extended 16-bit immediate	(opcode!=0) && (opcode!=BEQ) && (opcode!=BNE)
MemtoReg	Steer ALU result to GPR write port	steer memory load to GPR wr. port	opcode==LW
RegWrite	GPR write disabled	GPR write enabled	(opcode!=SW) && (opcode!=Bxx) && (opcode!=J) && (opcode!=JR))

Single-Bit Control Signals

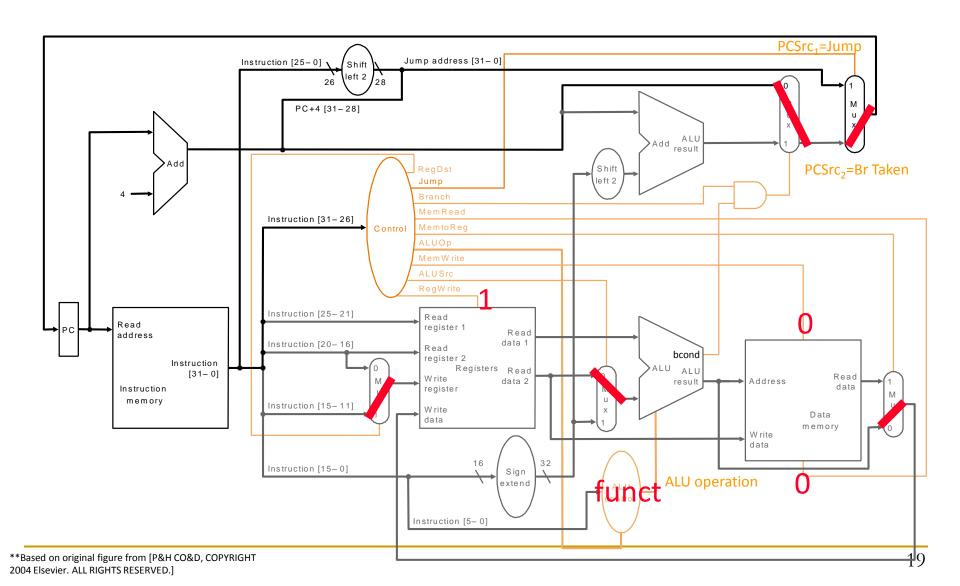
	When De-asserted	When asserted	Equation	
MemRead	Memory read disabled	Memory read port return load value	opcode==LW	
MemWrite	Memory write disabled	Memory write enabled	opcode==SW	
PCSrc ₁	According to PCSrc ₂	next PC is based on 26- bit immediate jump target	(opcode==J) (opcode==JAL)	
PCSrc ₂	next PC = PC + 4	next PC is based on 16- bit immediate branch target	(opcode==Bxx) && "bcond is satisfied"	

ALU Control

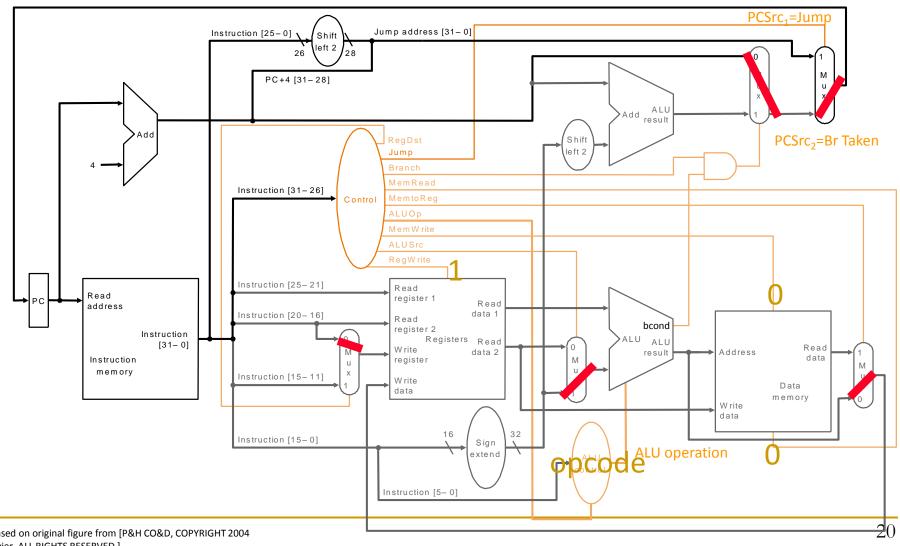
case opcode

- '0' ⇒ select operation according to funct
 'ALUi' ⇒ selection operation according to opcode
 'LW' ⇒ select addition
 'SW' ⇒ select addition
 'Bxx' ⇒ select bcond generation function
 ⇒ don't care
- Example ALU operations
 - ADD, SUB, AND, OR, XOR, NOR, etc.
 - bcond on equal, not equal, LE zero, GT zero, etc.

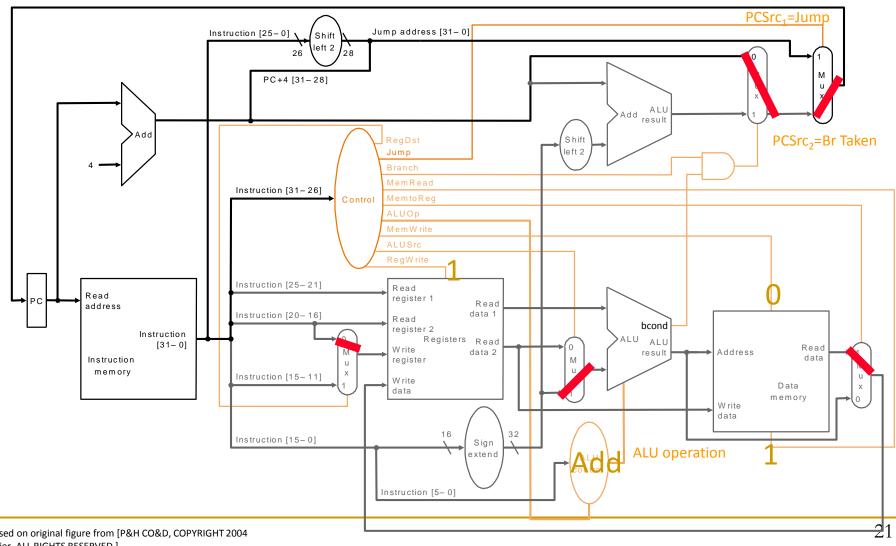
R-Type ALU

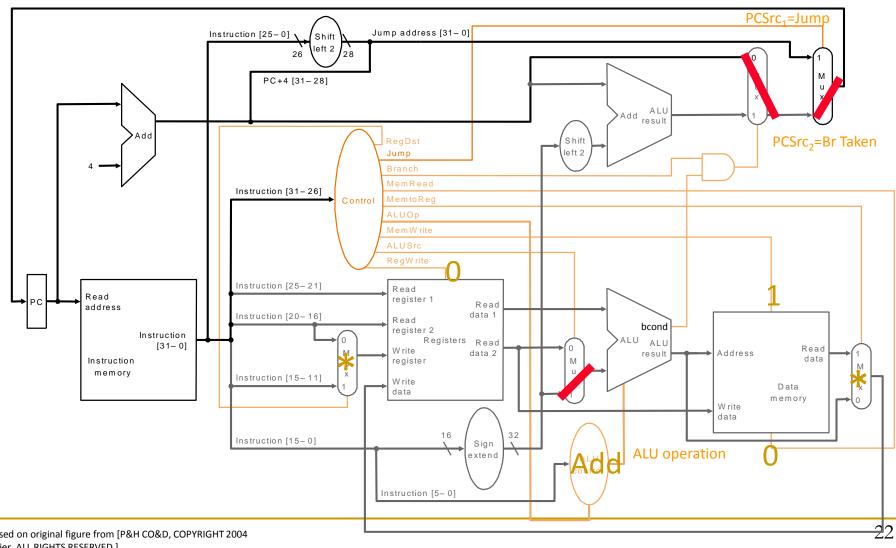


I-Type ALU

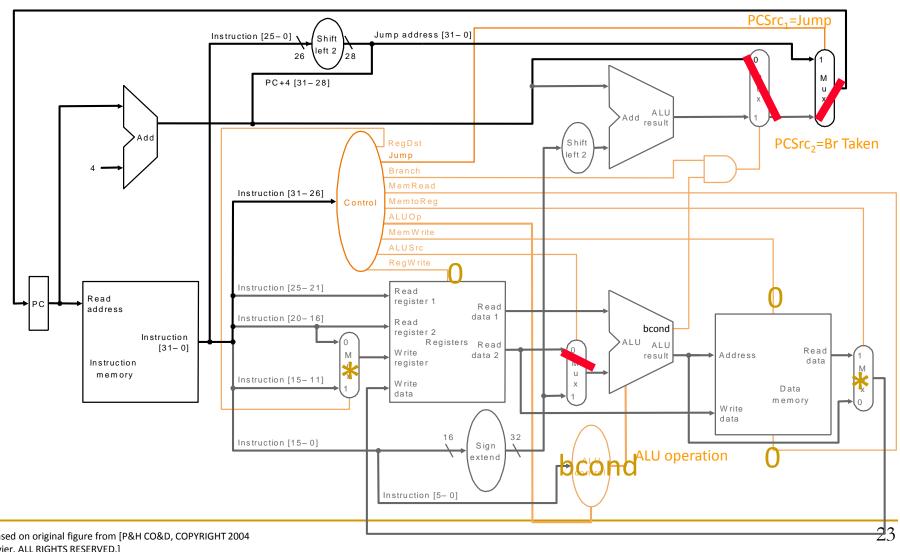




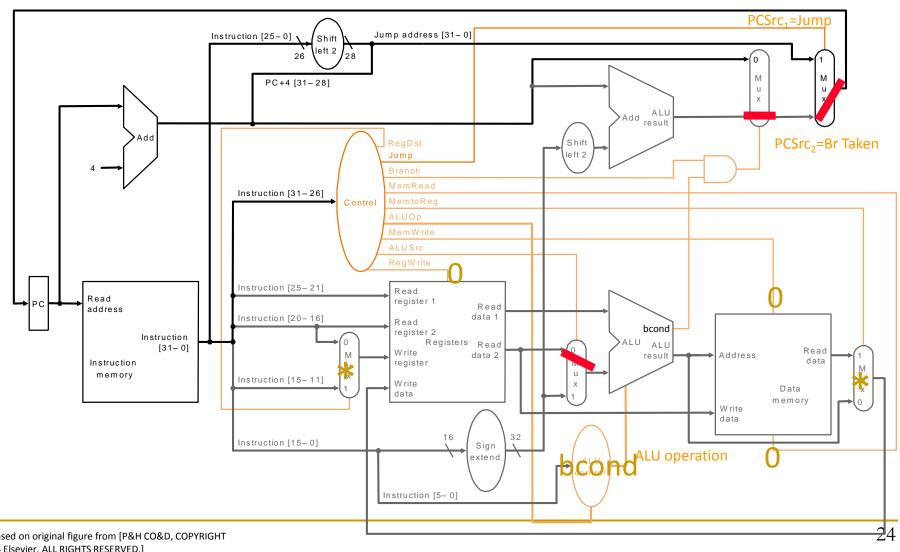


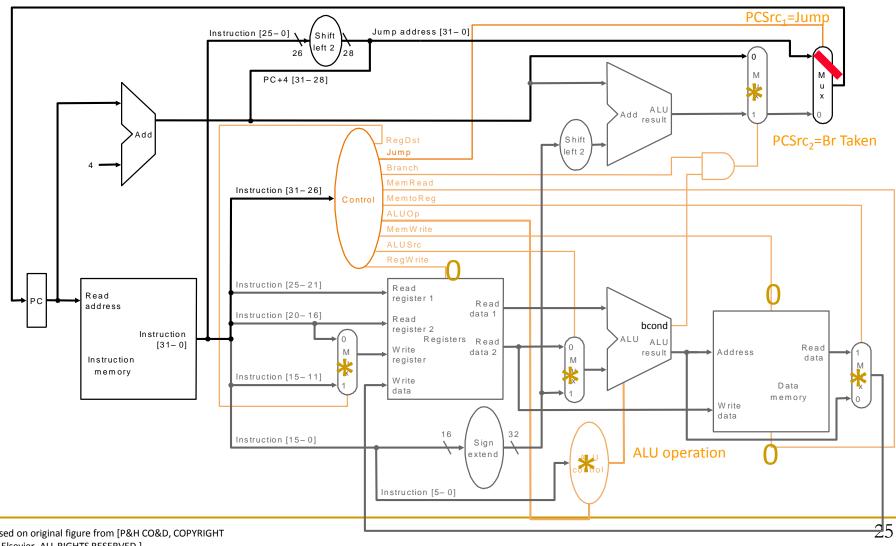


Branch Not Taken



Branch Taken





What is in That Control Box?

- Combinational Logic → Hardwired Control
 - Idea: Control signals generated combinationally based on instruction
 - Necessary in a single-cycle microarchitecture...
- Sequential Logic → Sequential/Microprogrammed Control
 - Idea: A memory structure contains the control signals associated with an instruction
 - Control Store

Evaluating the Single-Cycle Microarchitecture

A Single-Cycle Microarchitecture

- Is this a good idea/design?
- When is this a good design?
- When is this a bad design?
- How can we design a better microarchitecture?

A Single-Cycle Microarchitecture: Analysis

- Every instruction takes 1 cycle to execute
 - CPI (Cycles per instruction) is strictly 1
- How long each instruction takes is determined by how long the slowest instruction takes to execute
 - Even though many instructions do not need that long to execute
- Clock cycle time of the microarchitecture is determined by how long it takes to complete the slowest instruction
 - Critical path of the design is determined by the processing time of the slowest instruction

What is the Slowest Instruction to Process?

- Let's go back to the basics
- All six phases of the instruction processing cycle take a single machine clock cycle to complete
- Fetch
- Decode
- Evaluate Address
- Fetch Operands
- Execute
- Store Result

- 1. Instruction fetch (IF)
- 2. Instruction decode and register operand fetch (ID/RF)
- 3. Execute/Evaluate memory address (EX/AG)
- 4. Memory operand fetch (MEM)
- 5. Store/writeback result (WB)

Do each of the above phases take the same time (latency) for all instructions?

Single-Cycle Datapath Analysis

Assume

memory units (read or write): 200 ps

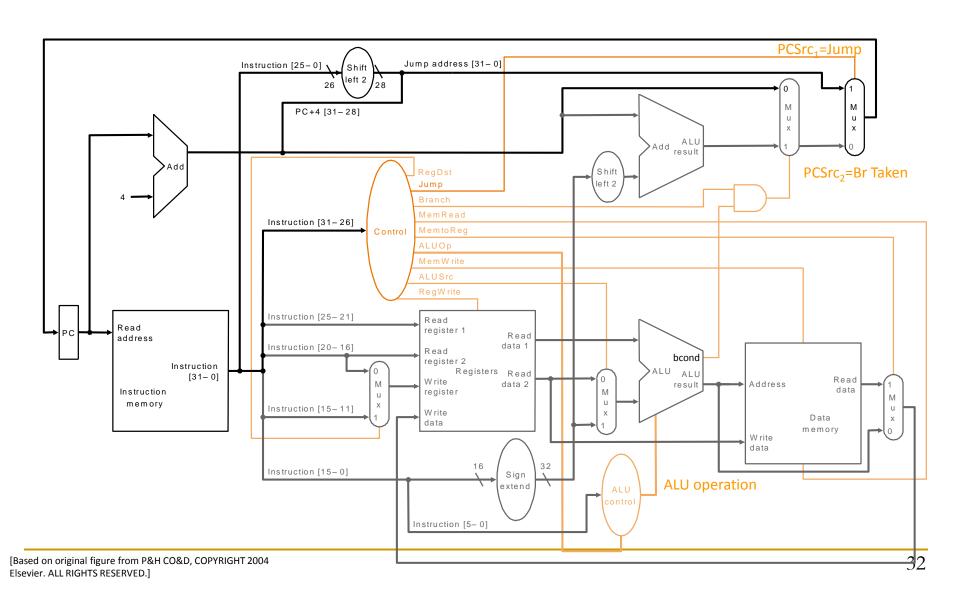
ALU and adders: 100 ps

register file (read or write): 50 ps

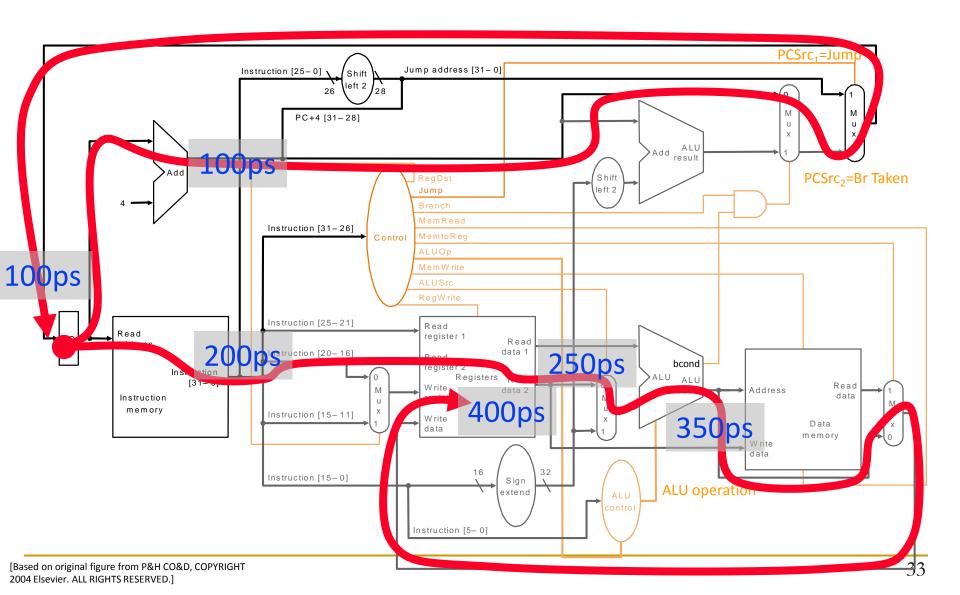
other combinational logic: 0 ps

steps	IF	ID	EX	MEM	WB	
resources	mem	RF	ALU	mem	RF	Delay
R-type	200	50	100		50	400
I-type	200	50	100		50	400
LW	200	50	100	200	50	600
SW	200	50	100	200		550
Branch	200	50	100			350
Jump	200					200 3 ⁻¹

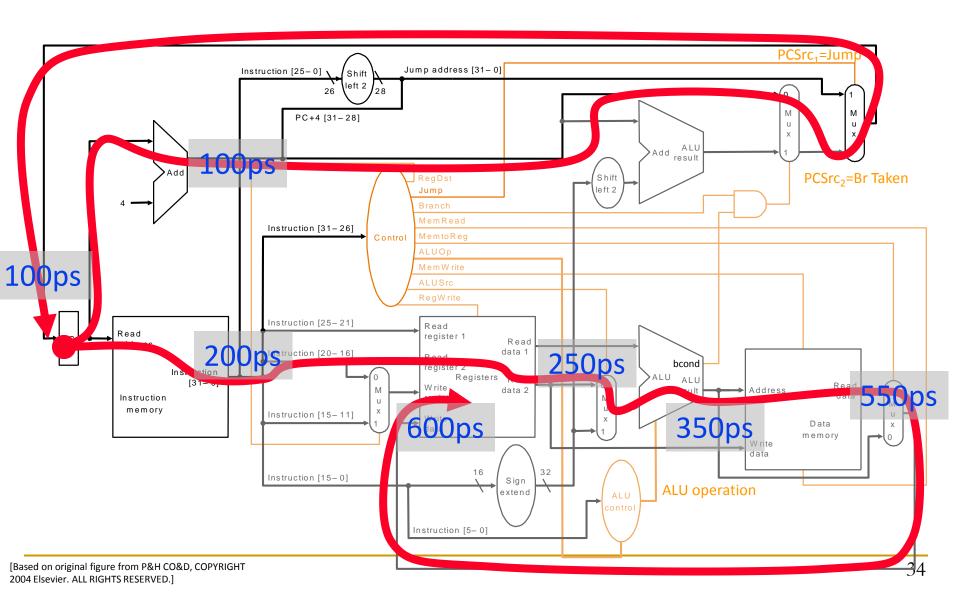
Let's Find the Critical Path

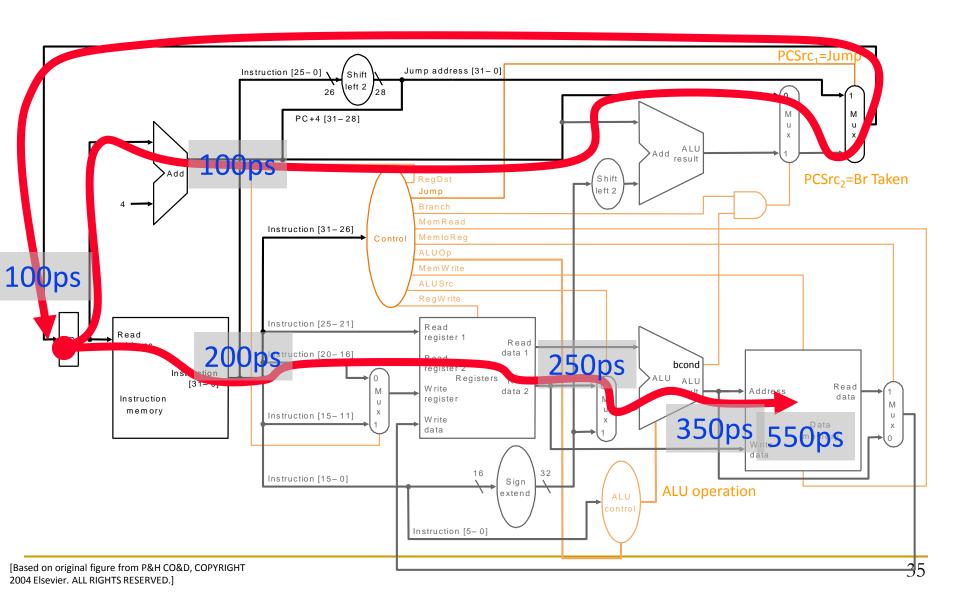


R-Type and I-Type ALU

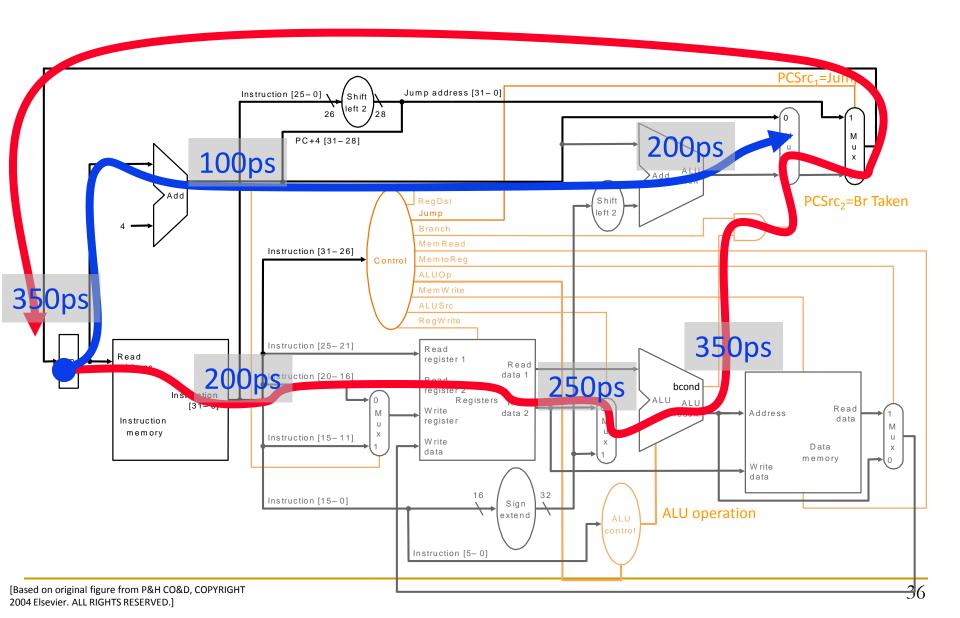




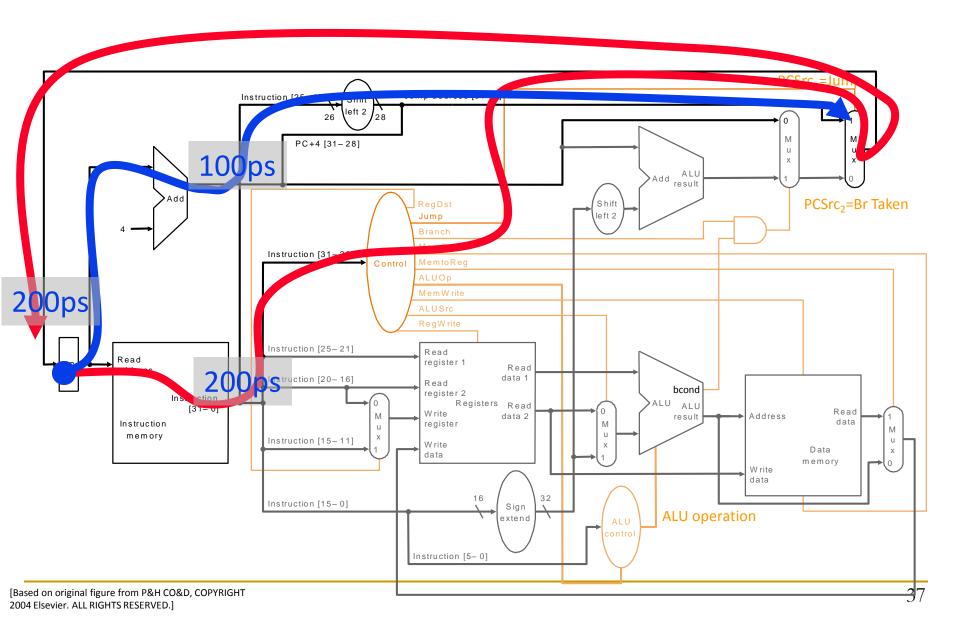




Branch Taken



Jump



What About Control Logic?

- How does that affect the critical path?
- Food for thought for you:
 - Can control logic be on the critical path?
 - A note on CDC 5600: control store access too long...

What is the Slowest Instruction to Process?

- Memory is not magic
- What if memory sometimes takes 100ms to access?
- Does it make sense to have a simple register to register add or jump to take {100ms+all else to do a memory operation}?
- And, what if you need to access memory more than once to process an instruction?
 - Which instructions need this?
 - Do you provide multiple ports to memory?

Single Cycle uArch: Complexity

Contrived

All instructions run as slow as the slowest instruction

Inefficient

- All instructions run as slow as the slowest instruction
- Must provide worst-case combinational resources in parallel as required by any instruction
- Need to replicate a resource if it is needed more than once by an instruction during different parts of the instruction processing cycle
- Not necessarily the simplest way to implement an ISA
 - Single-cycle implementation of REP MOVS, INDEX, POLY?
- Not easy to optimize/improve performance
 - Optimizing the common case does not work (e.g. common instructions)
 - Need to optimize the worst case all the time

Microarchitecture Design Principles

- Critical path design
 - Find the maximum combinational logic delay and decrease it
- Bread and butter (common case) design
 - Spend time and resources on where it matters
 - i.e., improve what the machine is really designed to do
 - Common case vs. uncommon case
- Balanced design
 - Balance instruction/data flow through hardware components
 - Balance the hardware needed to accomplish the work
- How does a single-cycle microarchitecture fare in light of these principles?

Multi-Cycle Microarchitectures

Multi-Cycle Microarchitectures

 Goal: Let each instruction take (close to) only as much time it really needs

Idea

- Determine clock cycle time independently of instruction processing time
- Each instruction takes as many clock cycles as it needs to take
 - Multiple state transitions per instruction
 - The states followed by each instruction is different

Remember: The "Process instruction" Step

- ISA specifies abstractly what A' should be, given an instruction and A
 - It defines an abstract finite state machine where
 - State = programmer-visible state
 - Next-state logic = instruction execution specification
 - From ISA point of view, there are no "intermediate states" between A and A' during instruction execution
 - One state transition per instruction
- Microarchitecture implements how A is transformed to A'
 - There are many choices in implementation
 - We can have programmer-invisible state to optimize the speed of instruction execution: multiple state transitions per instruction
 - Choice 1: $AS \rightarrow AS'$ (transform A to A' in a single clock cycle)
 - Choice 2: AS → AS+MS1 → AS+MS2 → AS+MS3 → AS' (take multiple clock cycles to transform AS to AS')

Multi-Cycle Microarchitecture

AS = Architectural (programmer visible) state at the beginning of an instruction



Step 1: Process part of instruction in one clock cycle



Step 2: Process part of instruction in the next clock cycle





AS' = Architectural (programmer visible) state at the end of a clock cycle

Benefits of Multi-Cycle Design

Critical path design

 Can keep reducing the critical path independently of the worstcase processing time of any instruction

Bread and butter (common case) design

 Can optimize the number of states it takes to execute "important" instructions that make up much of the execution time

Balanced design

- No need to provide more capability or resources than really needed
 - An instruction that needs resource X multiple times does not require multiple X's to be implemented
 - Leads to more efficient hardware: Can reuse hardware components needed multiple times for an instruction

Performance Analysis

- Execution time of an instruction
 - □ {CPI} x {clock cycle time}
- Execution time of a program
 - Sum over all instructions [{CPI} x {clock cycle time}]
 - {# of instructions} x {Average CPI} x {clock cycle time}
- Single cycle microarchitecture performance
 - \Box CPI = 1
 - Clock cycle time = long
- Multi-cycle microarchitecture performance
 - CPI = different for each instruction
 - Average CPI → hopefully small
 - Clock cycle time = short

Now, we have two degrees of freedom to optimize independently

An Aside: CPI vs. Frequency

- CPI vs. Clock cycle time
- At odds with each other
 - Reducing one increases the other for a single instruction
 - Why?
- Average CPI can be amortized/reduced via concurrent processing of multiple instructions
 - The same cycle is devoted to processing multiple instructions
 - Example: Pipelining, superscalar execution

A Multi-Cycle Microarchitecture A Closer Look

How Do We Implement This?

- Maurice Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester Univ. Computer Inaugural Conf., 1951.
- The concept of microcoded/microprogrammed machines

Realization

- One can implement the "process instruction" step as a finite state machine that sequences between states and eventually returns back to the "fetch instruction" state
- A state is defined by the control signals asserted in it
- Control signals for the next state determined in current state

The Instruction Processing Cycle



A Basic Multi-Cycle Microarchitecture

- Instruction processing cycle divided into "states"
 - A stage in the instruction processing cycle can take multiple states
- A multi-cycle microarchitecture sequences from state to state to process an instruction
 - The behavior of the machine in a state is completely determined by control signals in that state
- The behavior of the entire processor is specified fully by a finite state machine
- In a state (clock cycle), control signals control
 - How the datapath should process the data
 - How to generate the control signals for the next clock cycle

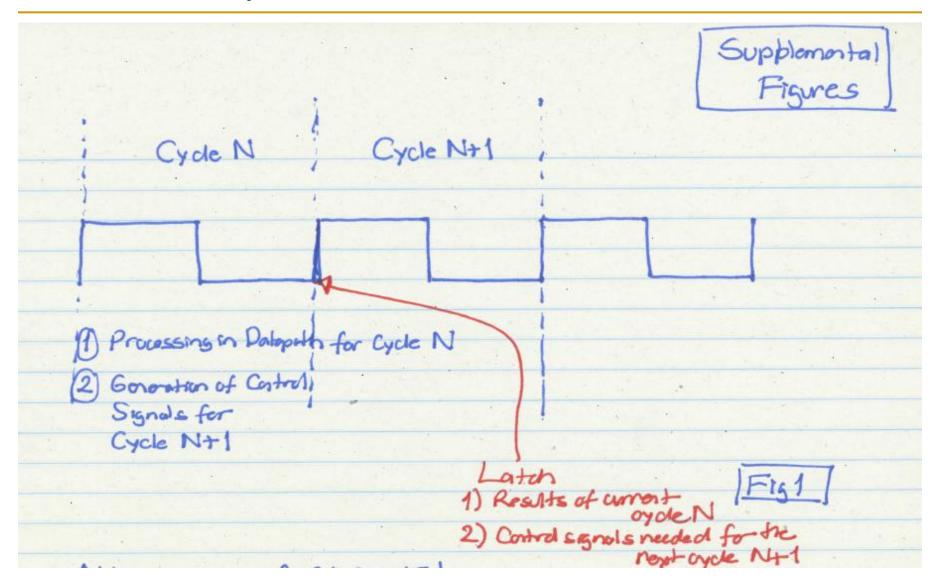
Microprogrammed Control Terminology

- Control signals associated with the current state
 - Microinstruction
- Act of transitioning from one state to another
 - Determining the next state and the microinstruction for the next state
 - Microsequencing
- Control store stores control signals for every possible state
 - Store for microinstructions for the entire FSM
- Microsequencer determines which set of control signals will be used in the next clock cycle (i.e., next state)

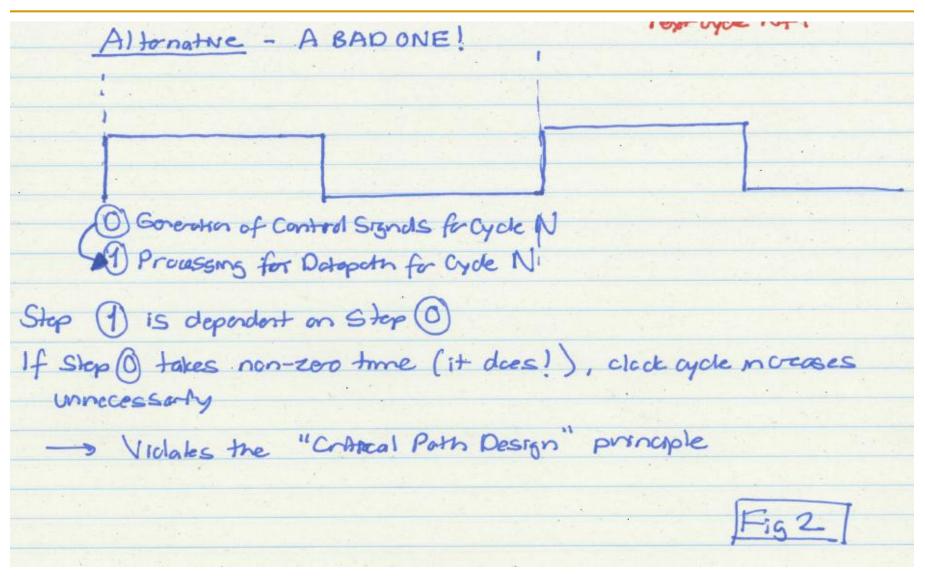
What Happens In A Clock Cycle?

- The control signals (microinstruction) for the current state control
 - Processing in the data path
 - Generation of control signals (microinstruction) for the next cycle
 - See Supplemental Figure 1 (next slide)
- Datapath and microsequencer operate concurrently
- Question: why not generate control signals for the current cycle in the current cycle?
 - This will lengthen the clock cycle
 - Why would it lengthen the clock cycle?
 - See Supplemental Figure 2

A Clock Cycle



A Bad Clock Cycle!



A Simple LC-3b Control and Datapath

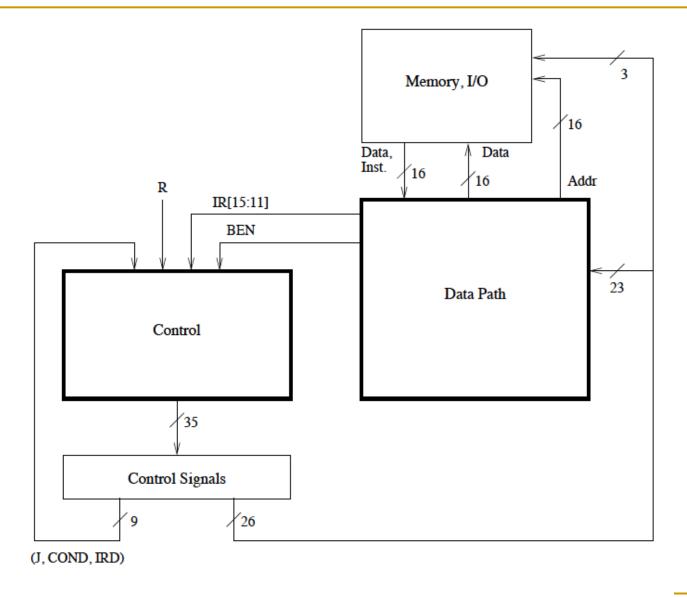


Figure C.1: Microarchitecture of the LC-3b, major components

What Determines Next-State Control Signals?

- What is happening in the current clock cycle
 - See the 9 control signals coming from "Control" block
 - What are these for?
- The instruction that is being executed
 - IR[15:11] coming from the Data Path
- Whether the condition of a branch is met, if the instruction being processed is a branch
 - BEN bit coming from the datapath
- Whether the memory operation is completing in the current cycle, if one is in progress
 - R bit coming from memory

A Simple LC-3b Control and Datapath

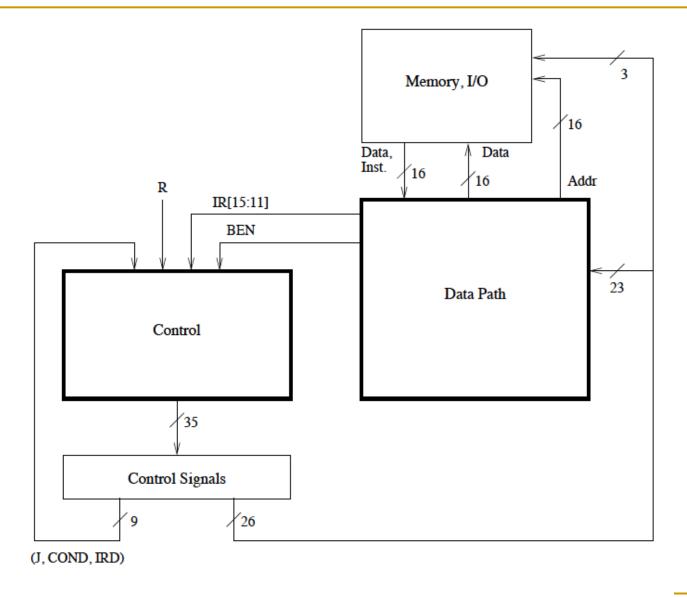


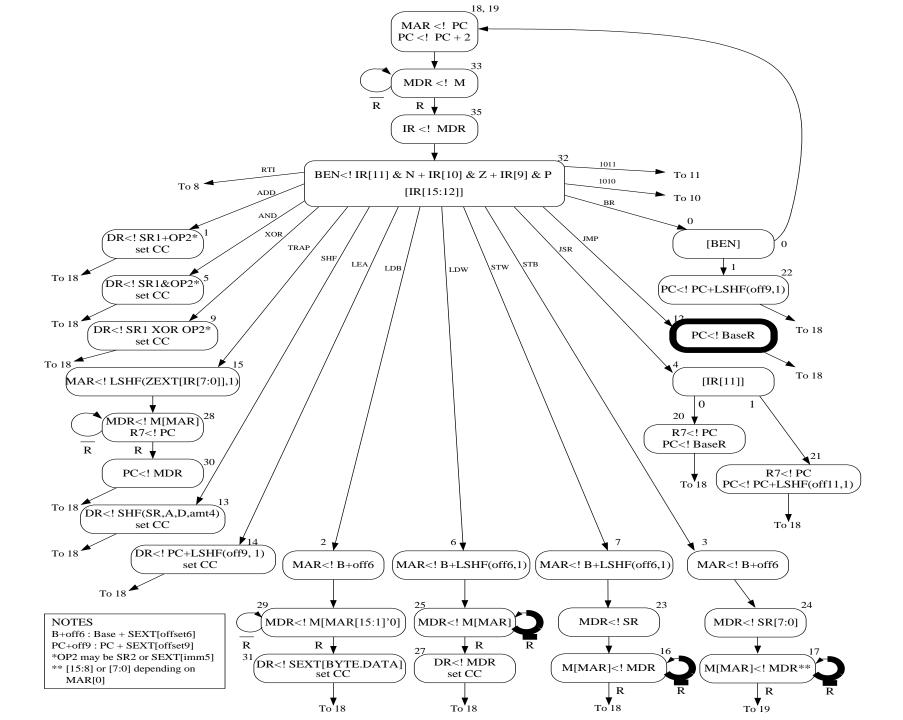
Figure C.1: Microarchitecture of the LC-3b, major components

The State Machine for Multi-Cycle Processing

- The behavior of the LC-3b uarch is completely determined by
 - the 35 control signals and
 - additional 7 bits that go into the control logic from the datapath
- 35 control signals completely describe the state of the control structure
- We can completely describe the behavior of the LC-3b as a state machine, i.e. a directed graph of
 - Nodes (one corresponding to each state)
 - Arcs (showing flow from each state to the next state(s))

An LC-3b State Machine

- Patt and Patel, App C, Figure C.2
- Each state must be uniquely specified
 - Done by means of state variables
- 31 distinct states in this LC-3b state machine
 - Encoded with 6 state variables
- Examples
 - State 18,19 correspond to the beginning of the instruction processing cycle
 - \square Fetch phase: state 18, 19 \rightarrow state 33 \rightarrow state 35
 - Decode phase: state 32



LC-3b State Machine: Some Questions

- How many cycles does the fastest instruction take?
- How many cycles does the slowest instruction take?
- Why does the BR take as long as it takes in the FSM?
- What determines the clock cycle?
- Is this a Mealy machine or a Moore machine?