18-447
Computer Architecture
Lecture 6: Multi-cycle Microarchitectures

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2012, 2/6/2012
Reminder: Homeworks

- Homework 1 solutions
  - Check and study the solutions!
  - Learning now is better than rushing later

- Homework 2
  - Already out
  - Due February 13
  - ISA concepts, ISA vs. microarchitecture, microcoded machines
Reminder: Lab Assignment 2

- Lab Assignment 1.5
  - Verilog practice
  - Not to be turned in

- Lab Assignment 2
  - Due Friday, Feb 17, at the end of the lab
  - Individual assignment
    - No collaboration; please respect the honor code
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 are discussing in class.

- We are not specifying any particular details of the microcode format or the microarchitecture; you should 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.
Feedback on Lab Assignment 1

- Chris, Lavanya, and Abeer are working hard on grading
  - We will have very comprehensive tests for all labs
  - Lab 1 tests exercise every case of each instruction as well as long programs (e.g., REP MOVS)
  - We will release test cases and register dumps

- Be thorough and test all possible cases
- Follow directions – they are there for a reason
  - No modifications to shell code!
  - No unaligned accesses to memory
- Remove all your debugging printf’s before handing in code
- Do the extra credit work if the lab is too easy!
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
Readings for 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
Today’s Agenda

- Finish single-cycle microarchitectures
  - Critical path

- Microarchitecture design principles

- Performance evaluation primer

- Multi-cycle microarchitectures
  - Microprogrammed control
Do the readings
- P&P Appendixes A and C
- Wilkes 1951 paper
- Today’s lectures 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
Review: The Full Single-Cycle Datapath

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

JAL, JR, JALR omitted
Single-Cycle Datapath for
Arithmetic and Logical Instructions
if MEM[PC] == ADDI rt rs immediate
GPR[rt] ← GPR[rs] + sign-extend (immediate)
PC ← PC + 4

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
Single-Cycle Datapath for Data Movement Instructions
Review: Datapath for Non-Control-Flow Insts.

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

**
Single-Cycle Datapath for Control Flow Instructions
Review: Unconditional Jump Instructions

- **Assembly**
  
  \[ J \text{ immediate}_{26} \]

- **Machine encoding**

  
<table>
<thead>
<tr>
<th>J</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-bit</td>
<td>26-bit</td>
</tr>
</tbody>
</table>

  J-type

- **Semantics**

  \[
  \text{if} \ \text{MEM}[PC] == J \text{ immediate}_{26} \\
  \text{target} = \{ \text{PC}[31:28], \text{immediate}_{26}, 2'b00 \} \\
  \text{PC} \leftarrow \text{target}
  \]
Review: Unconditional Jump Datapath

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

if MEM[PC]==J immediate26
PC = { PC[31:28], immediate26, 2’b00 }
Conditional Branch Instructions

- Assembly (e.g., branch if equal)
  \[ \text{BEQ } rs_{\text{reg}} \ rt_{\text{reg}} \immediate_{16} \]

- Machine encoding

<table>
<thead>
<tr>
<th>BEQ</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-bit</td>
<td>5-bit</td>
<td>5-bit</td>
<td>16-bit</td>
</tr>
</tbody>
</table>

- Semantics (assuming no branch delay slot)
  \[
  \text{if } \text{MEM}[\text{PC}] = = \text{BEQ } rs \ rt \immediate_{16} \\
  \text{target} = \text{PC} + 4 + \text{sign-extend(}\immediate\text{)} \times 4 \\
  \text{if } \text{GPR}[rs] = = \text{GPR}[rt] \text{ then } \text{PC} \leftarrow \text{target} \\
  \text{else} \quad \text{PC} \leftarrow \text{PC} + 4
  \]
How to uphold the delayed branch semantics?
Putting It All Together

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**

JAL, JR, JALR omitted
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

<table>
<thead>
<tr>
<th></th>
<th>When De-asserted</th>
<th>When asserted</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RegDest</strong></td>
<td>GPR write select according to ( rt ), i.e., ( \text{inst}[20:16] )</td>
<td>GPR write select according to ( rd ), i.e., ( \text{inst}[15:11] )</td>
<td>( \text{opcode} == 0 )</td>
</tr>
<tr>
<td><strong>ALUSrc</strong></td>
<td>( 2^{nd} ) ALU input from ( 2^{nd} ) GPR read port</td>
<td>( 2^{nd} ) ALU input from sign-extended 16-bit immediate</td>
<td>( \text{opcode} != 0 ) &amp;&amp; ( \text{opcode} != \text{BEQ} ) &amp;&amp; ( \text{opcode} != \text{BNE} )</td>
</tr>
<tr>
<td><strong>MemtoReg</strong></td>
<td>Steer ALU result to GPR write port</td>
<td>steer memory load to GPR wr. port</td>
<td>( \text{opcode} == \text{LW} )</td>
</tr>
</tbody>
</table>

JAL and JALR require additional RegDest and MemtoReg options.
# Single-Bit Control Signals

<table>
<thead>
<tr>
<th></th>
<th>When De-asserted</th>
<th>When asserted</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MemRead</strong></td>
<td>Memory read disabled</td>
<td>Memory read port return load value</td>
<td>(\text{opcode}==\text{LW})</td>
</tr>
<tr>
<td><strong>MemWrite</strong></td>
<td>Memory write disabled</td>
<td>Memory write enabled</td>
<td>(\text{opcode}==\text{SW})</td>
</tr>
<tr>
<td><strong>PCSrc(_1)</strong></td>
<td>According to (\text{PCSrc}_2)</td>
<td>next PC is based on 26-bit immediate jump target</td>
<td>((\text{opcode}==\text{J}) \</td>
</tr>
<tr>
<td><strong>PCSrc(_2)</strong></td>
<td>next PC = PC + 4</td>
<td>next PC is based on 16-bit immediate branch target</td>
<td>((\text{opcode}==\text{Bxx}) \ &amp;</td>
</tr>
</tbody>
</table>

JR and JALR require additional PCSrc options.
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

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
I-Type ALU

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.])**
**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
Branch Not Taken

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
Branch Taken

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
Jump

**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
What is in That Control Box?

- Combinational Logic $\rightarrow$ Hardwired Control
  - Idea: Control signals generated combinationally based on instruction

- Sequential Logic $\rightarrow$ Sequential/Microprogrammed Control
  - Control Store
  - Idea: A memory structure contains the control signals associated with an instruction
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

<table>
<thead>
<tr>
<th>steps</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources</td>
<td>mem</td>
<td>RF</td>
<td>ALU</td>
<td>mem</td>
<td>RF</td>
<td></td>
</tr>
<tr>
<td>R-type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>I-type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td>50</td>
<td>400</td>
</tr>
<tr>
<td>LW</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>50</td>
<td>600</td>
</tr>
<tr>
<td>SW</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td></td>
<td>550</td>
</tr>
<tr>
<td>Branch</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td></td>
<td></td>
<td>350</td>
</tr>
<tr>
<td>Jump</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>
Let’s Find the Critical Path
R-Type and I-Type ALU

![Diagram of R-Type and I-Type ALU](image_url)

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Branch Taken

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Jump

Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.
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?  
  - VAX INDEX instruction  
  - 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 $\rightarrow$ AS+MS1 $\rightarrow$ AS+MS2 $\rightarrow$ AS+MS3 $\rightarrow$ AS’ (take multiple clock cycles to transform AS to AS’)
Multi-Cycle Microarchitecture

\[ \text{AS} = \text{Architectural (programmer visible) state at the beginning of an instruction} \]

- Step 1: Process part of instruction in one clock cycle
- Step 2: Process part of instruction in the next clock cycle
- \( \cdots \)

\[ \text{AS'} = \text{Architectural (programmer visible) state at the end of a clock cycle} \]
Benefits of Multi-Cycle Design

- Critical path design
  - Can keep reducing the critical path independently of worst-case 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
  - \( \{\text{CPI}\} \times \{\text{clock cycle time}\} \)

- Execution time of a program
  - Sum over all instructions \([\{\text{CPI}\} \times \{\text{clock cycle time}\}]\]
  - \( \{\text{# of instructions}\} \times \{\text{Average CPI}\} \times \{\text{clock cycle time}\} \)

- Single cycle microarchitecture performance
  - CPI = 1
  - Clock cycle time = long

- Multi-cycle microarchitecture performance
  - CPI = different for each instruction
    - Average CPI \( \rightarrow \) 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 multiple instructions
  - Example: Pipelining, superscalar execution
A Multi-Cycle Microarchitecture

A Closer Look
How Do We Implement This?


- 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

- Fetch
- Decode
- Evaluate Address
- Fetch Operands
- Execute
- Store Result
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 Fig 1

- 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 Fig 2
A Clock Cycle

- Processing in Path for Cycle N
- Generation of Control Signals for Cycle N+1

Latch
1) Results of current cycle N
2) Control signals needed for the next cycle N+1
A Bad Clock Cycle!

Alternative - A BAD ONE!

- Generation of Control Signals for Cycle N
- Processing for Datapath for Cycle N

Step (1) is dependent on Step (0)

If Step (0) takes non-zero time (it does!), clock cycle increases unnecessarily

→ Violates the “Critical Path Design” principle
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
  - Fetch phase: state 18, 19 → state 33 → state 35
  - Decode phase: state 32
Figure C.2: A state machine for the LC-3b
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?
LC-3b Datapath

- Patt and Patel, App C, Figure C.3

Single-bus datapath design
- At any point only one value can be “gated” on the bus (i.e., can be driving the bus)
- Advantage: Low hardware cost: one bus
- Disadvantage: Reduced concurrency – if instruction needs the bus twice for two different things, these need to happen in different states

Control signals (26 of them) determine what happens in the datapath in one clock cycle
- Patt and Patel, App C, Table C.1
We did not cover the following slides in lecture. These are for your preparation for the next lecture.
<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD.MAR/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>LD.MDR/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>LD.IR/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>LD.BEN/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>LD.REG/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>LD.CC/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>LD.PC/1</td>
<td>NO, LOAD</td>
</tr>
<tr>
<td>GatePC/1</td>
<td>NO, YES</td>
</tr>
<tr>
<td>GateMDR/1</td>
<td>NO, YES</td>
</tr>
<tr>
<td>GateALU/1</td>
<td>NO, YES</td>
</tr>
<tr>
<td>GateMARMUX/1</td>
<td>NO, YES</td>
</tr>
<tr>
<td>GateSHF/1</td>
<td>NO, YES</td>
</tr>
<tr>
<td>PCMUX/2</td>
<td>PC+2 ;select pc+2</td>
</tr>
<tr>
<td></td>
<td>BUS ;select value from bus</td>
</tr>
<tr>
<td></td>
<td>ADDER ;select output of address adder</td>
</tr>
<tr>
<td>DRMUX/1</td>
<td>11.9 ;destination IR[11:9]</td>
</tr>
<tr>
<td></td>
<td>R7 ;destination R7</td>
</tr>
<tr>
<td>SR1MUX/1</td>
<td>11.9 ;source IR[11:9]</td>
</tr>
<tr>
<td></td>
<td>8.6 ;source IR[8:6]</td>
</tr>
<tr>
<td>ADDR1MUX/1</td>
<td>PC, BaseR</td>
</tr>
<tr>
<td>ADDR2MUX/2</td>
<td>ZERO ;select the value zero</td>
</tr>
<tr>
<td></td>
<td>offset6 ;select SEXT[IR[5:0]]</td>
</tr>
<tr>
<td></td>
<td>PCoffset9 ;select SEXT[IR[8:0]]</td>
</tr>
<tr>
<td></td>
<td>PCoffset11 ;select SEXT[IR[10:0]]</td>
</tr>
<tr>
<td>MARMUX/1</td>
<td>7.0 ;select LSHF(ZEXT[IR[7:0]],1)</td>
</tr>
<tr>
<td></td>
<td>ADDER ;select output of address adder</td>
</tr>
<tr>
<td>ALU2/2</td>
<td>ADD, AND, XOR, PASSA</td>
</tr>
<tr>
<td>MIO.ENABLE/1</td>
<td>NO, YES</td>
</tr>
<tr>
<td>R.W/1</td>
<td>RD, WR</td>
</tr>
<tr>
<td>DATA.SIZE/1</td>
<td>BYTE, WORD</td>
</tr>
<tr>
<td>LSHF1/1</td>
<td>NO, YES</td>
</tr>
</tbody>
</table>

Table C.1: Data path control signals
LC-3b Datapath: Some Questions

- How does instruction fetch happen in this datapath according to the state machine?

- What is the difference between gating and loading?

- Is this the smallest hardware you can design?
LC-3b Microprogrammed Control Structure

- Patt and Patel, App C, Figure C.4

- Three components:
  - Microinstruction, control store, microsequencer

- **Microinstruction**: control signals that control the datapath (26 of them) and determine the next state (9 of them)

- Each microinstruction is stored in a unique location in the **control store** (a special memory structure)

- Unique location: address of the state corresponding to the microinstruction
  - Remember each state corresponds to one microinstruction

- **Microsequencer** determines the address of the next microinstruction (i.e., next state)
Microsequencer

Control Store

Microinstruction

(J, COND, IRD)
The microarchitecture of the LC-3B, basic machine

Address of Next State

0,0,IR[15:12]

IRD

COND1
COND0

BEN
R
IR[11]

Branch
Ready
Addr. Mode


6

6
<table>
<thead>
<tr>
<th>RD</th>
<th>Code</th>
<th>DATA SIZE</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>63</td>
</tr>
</tbody>
</table>

Figure C.7: Specification of the control store
LC-3b Microsequencer

- Patt and Patel, App C, Figure C.5

- **The purpose of the microsequencer** is to determine the address of the next microinstruction (i.e., next state)

- Next address depends on 9 control signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>J/6:</td>
<td></td>
</tr>
<tr>
<td>COND/2:</td>
<td>COND0 ;Unconditional</td>
</tr>
<tr>
<td></td>
<td>COND1 ;Memory Ready</td>
</tr>
<tr>
<td></td>
<td>COND2 ;Branch</td>
</tr>
<tr>
<td></td>
<td>COND3 ;Addressing Mode</td>
</tr>
<tr>
<td>IRD/1:</td>
<td>NO, YES</td>
</tr>
</tbody>
</table>

Table C.2: Microsequencer control signals
Address of Next State

0,0,IR[15:12]

IRD

BEN

COND0

COND1

Branch

Ready

Addr. Mode

IR[11]


In both cases, the use of microinstructions is needed to generate DR, SR1, and BEN. The remaining signal, R, is a signal generated by the memory in order to allow the...
The Microsequencer: Some Questions

- When is the IRD signal asserted?
- What happens if an illegal instruction is decoded?
- What are condition (COND) bits for?
- How is variable latency memory handled?
- How do you do the state encoding?
  - Minimize number of state variables
  - Start with the 16-way branch
  - Then determine constraint tables and states dependent on COND
Variable-Latency Memory

- The ready signal (R) enables memory read/write to execute correctly
  - Example: transition from state 18 to state 33 is controlled by the R bit asserted by memory when memory data is available

- Could we have done this in a single-cycle microarchitecture?
The Microsequencer: Advanced Questions

- What happens if the machine is interrupted?
- What if an instruction generates an exception?
- How can you implement a complex instruction using this control structure?
  - Think REP MOVS