Readings

This week
- Multi-cycle microarchitecture
  - P&P, Appendices A and C
  - H&H, Chapter 7.4
- Microprogramming
  - P&P, Appendices A and C

Next week
- Pipelining
  - H&H, Chapter 7.5
- Pipelining Issues
  - H&H, Chapter 7.8.1-7.8.3
Agenda for Today & Next Few Lectures

- Instruction Set Architectures (ISA): LC-3 and MIPS
- Assembly programming: LC-3 and MIPS
- Microarchitecture (principles & single-cycle uarch)
- Multi-cycle microarchitecture
- Microprogramming
- Pipelining
- Issues in Pipelining: Control & Data Dependence Handling, State Maintenance and Recovery, ...
- Out-of-Order Execution
Recall: Performance Analysis Basics

- Execution time of an instruction
  - \(\text{CPI} \times \text{clock cycle time}\)
    - CPI: Number of cycles it takes to execute an instruction

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

- **Critical path design**
  - Find and **decrease the maximum combinational logic delay**
  - Break a path into multiple cycles if it takes too long

- **Bread and butter (common case) design**
  - Spend time and resources on where it matters most
    - 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
  - **Design to eliminate bottlenecks**: balance the hardware for the work
Recall: Multi-Cycle Microarchitecture

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

\[ \text{Step 1: Process part of instruction in one clock cycle} \]

\[ \text{Step 2: Process part of instruction in the next clock cycle} \]

\[ \ldots \]

\[ \text{AS}' = \text{Architectural (programmer visible) state at the end of a clock cycle} \]
Recall: 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 two things:
  - How the datapath should process the data
  - How to generate the control signals for the (next) clock cycle
Recall: Multi-Cycle MIPS FSM

S0: Fetch

- IorD = 0
- AluSrcA = 0
- AluSrcB = 01
- AluOp = 00
- PcsSrc = 00
- IrWrite
- PcWrite

S1: Decode

- AluSrcA = 0
- AluSrcB = 11
- AluOp = 00
- PcsSrc = 00
- Branch

- Op = Lw
- Op = Sw

S2: MemAdr

- AluSrcA = 1
- AluSrcB = 10
- AluOp = 00
- S3: MemRead

- Op = Lw
- Op = Sw

S3: MemRead

- IorD = 1
- MemWrite

S4: MemWriteback

- RegDst = 0
- MemtoReg = 1
- RegWrite

S5: MemWrite

- RegDst = 1
- MemtoReg = 0
- RegWrite

S6: Execute

- AluSrcA = 1
- AluSrcB = 00
- AluOp = 10
- S7: ALU Writeback

- AluSrcA = 1
- AluSrcB = 00
- AluOp = 01
- S8: Branch

- AluSrcA = 1
- AluSrcB = 10
- AluOp = 00
- S9: ADDI

- Op = Addi
- Op = J

S7: ALU Writeback

- RegDst = 1
- MemtoReg = 0
- RegWrite

S10: ADDI Writeback

- RegDst = 0
- MemtoReg = 0
- RegWrite

S8: Branch

- Op = Beq

S9: ADDI

- Op = ADDI

S11: Jump

- PcsSrc = 10
- PcsWrite

S10: ADDI Writeback

- RegDst = 0
- MemtoReg = 0
- RegWrite
Single-Cycle Performance

- $T_C$ is limited by the critical path (1w)
Single-Cycle Performance

- Single-cycle critical path:
  \[ T_c = t_{pcq\_PC} + t_{mem} + \max(t_{RF\_read}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RF\_setup} \]

- In most implementations, limiting paths are:
  - memory, ALU, register file.
  \[ T_c = t_{pcq\_PC} + 2t_{mem} + t_{RF\_read} + t_{mux} + t_{ALU} + t_{RF\_setup} \]
## Single-Cycle Performance Example

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameter</th>
<th>Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register clock-to-Q</td>
<td>$t_{pcq_PC}$</td>
<td>30</td>
</tr>
<tr>
<td>Register setup</td>
<td>$t_{\text{setup}}$</td>
<td>20</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>$t_{\text{mux}}$</td>
<td>25</td>
</tr>
<tr>
<td>ALU</td>
<td>$t_{\text{ALU}}$</td>
<td>200</td>
</tr>
<tr>
<td>Memory read</td>
<td>$t_{\text{mem}}$</td>
<td>250</td>
</tr>
<tr>
<td>Register file read</td>
<td>$t_{\text{RFread}}$</td>
<td>150</td>
</tr>
<tr>
<td>Register file setup</td>
<td>$t_{\text{RFsetup}}$</td>
<td>20</td>
</tr>
</tbody>
</table>

$$T_c =$$
Single-Cycle Performance Example

\[
T_c = t_{pcq\_PC} + 2t_{mem} + t_{RF\text{read}} + t_{mux} + t_{ALU} + t_{RF\text{setup}} \\
= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ ps} \\
= 925 \text{ ps}
\]

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameter</th>
<th>Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register clock-to-Q</td>
<td>(t_{pcq_PC})</td>
<td>30</td>
</tr>
<tr>
<td>Register setup</td>
<td>(t_{\text{setup}})</td>
<td>20</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>(t_{\text{mux}})</td>
<td>25</td>
</tr>
<tr>
<td>ALU</td>
<td>(t_{\text{ALU}})</td>
<td>200</td>
</tr>
<tr>
<td>Memory read</td>
<td>(t_{\text{mem}})</td>
<td>250</td>
</tr>
<tr>
<td>Register file read</td>
<td>(t_{RF\text{read}})</td>
<td>150</td>
</tr>
<tr>
<td>Register file setup</td>
<td>(t_{RF\text{setup}})</td>
<td>20</td>
</tr>
</tbody>
</table>

12
Example:

For a program with 100 billion instructions executing on a single-cycle MIPS processor:
Example:  
For a program with 100 billion instructions executing on a single-cycle MIPS processor:

\[
\text{Execution Time} = \# \text{ instructions} \times \text{CPI} \times T_c
\]
\[
= (100 \times 10^9)(1)(925 \times 10^{-12} \text{ s})
\]
\[
= 92.5 \text{ seconds}
\]
Multi-Cycle Performance: CPI

- Instructions take different number of cycles:
  - 3 cycles: beq, j
  - 4 cycles: R-Type, sw, addi
  - 5 cycles: lw

- CPI is weighted average, e.g. SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

- **Average CPI** = (0.11 + 0.02) 3 + (0.52 + 0.10) 4 + (0.25) 5
  = 4.12
Multi-cycle Performance: Cycle Time

- Multi-cycle critical path:

\[ T_c = \]
Multi-cycle Performance: Cycle Time

- Multi-cycle critical path:

\[ T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup} \]
## Multi-Cycle Performance Example

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameter</th>
<th>Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register clock-to-Q</td>
<td>$t_{pcq_PC}$</td>
<td>30</td>
</tr>
<tr>
<td>Register setup</td>
<td>$t_{\text{setup}}$</td>
<td>20</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>$t_{\text{mux}}$</td>
<td>25</td>
</tr>
<tr>
<td>ALU</td>
<td>$t_{\text{ALU}}$</td>
<td>200</td>
</tr>
<tr>
<td>Memory read</td>
<td>$t_{\text{mem}}$</td>
<td>250</td>
</tr>
<tr>
<td>Register file read</td>
<td>$t_{\text{RFread}}$</td>
<td>150</td>
</tr>
<tr>
<td>Register file setup</td>
<td>$t_{\text{RFsetup}}$</td>
<td>20</td>
</tr>
</tbody>
</table>

$$T_c =$$
# Multi-Cycle Performance Example

<table>
<thead>
<tr>
<th>Element</th>
<th>Parameter</th>
<th>Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register clock-to-Q</td>
<td>( t_{pcq_PC} )</td>
<td>30</td>
</tr>
<tr>
<td>Register setup</td>
<td>( t_{\text{setup}} )</td>
<td>20</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>( t_{\text{mux}} )</td>
<td>25</td>
</tr>
<tr>
<td>ALU</td>
<td>( t_{\text{ALU}} )</td>
<td>200</td>
</tr>
<tr>
<td>Memory read</td>
<td>( t_{\text{mem}} )</td>
<td>250</td>
</tr>
<tr>
<td>Register file read</td>
<td>( t_{\text{RFread}} )</td>
<td>150</td>
</tr>
<tr>
<td>Register file setup</td>
<td>( t_{\text{RFsetup}} )</td>
<td>20</td>
</tr>
</tbody>
</table>

\[
T_c = t_{pcq\_PC} + t_{\text{mux}} + \max(t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}}
\]

\[
= [30 + 25 + 250 + 20] \text{ ps}
\]

\[
= 325 \text{ ps}
\]
Multi-Cycle Performance Example

- For a program with 100 billion instructions executing on a multi-cycle MIPS processor
  - CPI = 4.12
  - $T_c = 325$ ps
- **Execution Time**
  \[\text{Execution Time} = (\# \text{ instructions}) \times \text{CPI} \times T_c\]
  \[= (100 \times 10^9)(4.12)(325 \times 10^{-12})\]
  \[= 133.9 \text{ seconds}\]

- This is slower than the single-cycle processor (92.5 seconds). **Why?**

- Did we break the stages in a balanced manner?
- Overhead of register setup/hold paid many times
- How would the results change with different assumptions on memory latency and instruction mix?
Review: Single-Cycle MIPS Processor
Review: Multi-Cycle MIPS Processor
Review: Multi-Cycle MIPS FSM

What is the shortcoming of this design?

What does this design assume about memory?
What If Memory Takes > One Cycle?

- Stay in the same “memory access” state until memory returns the data
- “Memory Ready?” bit is an input to the control logic that determines the next state
Another Example:

Microprogrammed Multi-Cycle Microarchitecture
How Do We Implement This?


An elegant implementation:
- The concept of microcoded/microprogrammed machines
Recall: 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 two things:
  - 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)
Example Control Structure

Simple Design of the Control Structure

Microsequencer

Control Store

$2^6 \times 35$

Microinstruction

(R, IR[15:11], BEN)

(J, COND, IRD)
What Happens In A Clock Cycle?

- The control signals (microinstruction) for the current state control two things:
  - Processing in the data path
  - Generation of control signals (microinstruction) for the next cycle
  - See Supplemental Figure 1 (next-next slide)

- Datapath and microsequencer operate concurrently

- Question: why not generate control signals for the current cycle in the current cycle?
  - This could lengthen the clock cycle
  - Why could it lengthen the clock cycle?
  - See Supplemental Figure 2
Microarchitecture of the LC-3b, major components

1. J[5:0], COND[1:0], and IRD—9 bits of control signals provided by the current clock cycle.
2. inst[15:12], which identifies the opcode, and inst[11:11], which differentiates JSR from JSRR (i.e., the addressing mode for the target of the subroutine call).
3. BEN to indicate whether or not a BR should be taken.

Microarchitecture of the LC-3b, major components

Read P&P Revised Appendix C
On website
A Clock Cycle

1) Processing in Datapath for Cycle N
2) 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!

Step ① is dependent on Step ②

If Step ② takes non-zero time (it does!), clock cycle increases unnecessarily

→ Violates the "Critical Path Design" principle

Fig 2
A Simple LC-3b Control and Datapath

Read P&P Revised Appendix C
On website
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

Microarchitecture of the LC-3b, major components

1. J[5:0], COND[1:0], and IRD—9 bits of control signals provided by the current clock cycle.
2. inst[15:12], which identifies the opcode, and inst[11:11], which differentiates JSR from JSRR (i.e., the addressing mode for the target of the subroutine call).
3. BEN to indicate whether or not a BR should be taken.
4. If a memory operation is in progress, whether it is completing during this cycle.

Figure C.1 identifies the specific information in our implementation of the LC-3b that corresponds to these items.
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, *Revised Appendix 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

NOTES

B+off6 : Base + SEXT[offset6]
P+off9 : PC + SEXT[offset9]
*OP2 may be SR2 or SEXT[imm5]
** [15:8] or [7:0] depending on MAR[0]
The FSM Implements the LC-3b ISA

- P&P Appendix A (revised):
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 time?
LC-3b Datapath

- Patt and Patel, Revised Appendix 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, Revised Appendix C, Table C.1
Figure C.3: The LC-3b data path provides you with the additional flexibility of more states, so we have selected a control store consisting of 26 locations.
Remember the MIPS datapath

IR[11:9] → DR
111 → DRMUX

IR[11:9] → DR
IR[8:6] → SR1MUX

IR[11:9] → SR1

IR[11:9] → Logic
N → Logic
Z → Logic
P → Logic

Logic → BEN

(c)
<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>ALUK/2</td>
<td>ADD, AND, XOR, PASSA</td>
</tr>
<tr>
<td>MIO.EN/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?
  - Gating: Enable/disable an input to be connected to the bus
    - Combinational: during a clock cycle
  - Loading: Enable/disable an input to be written to a register
    - Sequential: e.g., at a clock edge (assume at the end of cycle)

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

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

- Three components:
  - Microinstruction, control store, microsequencer

- **Microinstruction**: control signals that control the datapath (26 of them) and help 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)
Simple Design of the Control Structure

Figure C.4: The control structure of a microprogrammed implementation, overall block diagram on the LC-3b instruction being executed during the current instruction cycle. This state carries out the DECODE phase of the instruction cycle. If the IRD control signal in the microinstruction corresponding to state \(32\) is 1, the output MUX of the microsequencer (Figure C.5) will take its source from the six bits formed by 00 concatenated with the four opcode bits IR\([15:12]\). Since IR\([15:12]\) specifies the opcode of the current LC-3b instruction being processed, the next address of the control store will be one of 16 addresses, corresponding to the 14 opcodes plus the two unused opcodes, IR\([15:12]\) = 1010 and 1011. That is, each of the 16 next states is the first state to be carried out after the instruction has been decoded in state \(32\). For example, if the instruction being processed is ADD, the address of the next state is state \(1\), whose microinstruction is stored at location 000001. Recall that IR\([15:12]\) for ADD is 0001.
Address of Next State

Figure C.6 shows the additional logic needed to generate DR, SR1, and BEN. The remaining signal, R, is a signal generated by the memory in order to allow...
<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>(State 0)</td>
</tr>
<tr>
<td>000001</td>
<td>(State 1)</td>
</tr>
<tr>
<td>000010</td>
<td>(State 2)</td>
</tr>
<tr>
<td>000011</td>
<td>(State 3)</td>
</tr>
<tr>
<td>000100</td>
<td>(State 4)</td>
</tr>
<tr>
<td>000101</td>
<td>(State 5)</td>
</tr>
<tr>
<td>000110</td>
<td>(State 6)</td>
</tr>
<tr>
<td>000111</td>
<td>(State 7)</td>
</tr>
<tr>
<td>001000</td>
<td>(State 8)</td>
</tr>
<tr>
<td>001001</td>
<td>(State 9)</td>
</tr>
<tr>
<td>001010</td>
<td>(State 10)</td>
</tr>
<tr>
<td>001011</td>
<td>(State 11)</td>
</tr>
<tr>
<td>001100</td>
<td>(State 12)</td>
</tr>
<tr>
<td>001101</td>
<td>(State 13)</td>
</tr>
<tr>
<td>001110</td>
<td>(State 14)</td>
</tr>
<tr>
<td>001111</td>
<td>(State 15)</td>
</tr>
<tr>
<td>010000</td>
<td>(State 16)</td>
</tr>
<tr>
<td>010001</td>
<td>(State 17)</td>
</tr>
<tr>
<td>010010</td>
<td>(State 18)</td>
</tr>
<tr>
<td>010011</td>
<td>(State 19)</td>
</tr>
<tr>
<td>010100</td>
<td>(State 20)</td>
</tr>
<tr>
<td>010101</td>
<td>(State 21)</td>
</tr>
<tr>
<td>010110</td>
<td>(State 22)</td>
</tr>
<tr>
<td>010111</td>
<td>(State 23)</td>
</tr>
<tr>
<td>011000</td>
<td>(State 24)</td>
</tr>
<tr>
<td>011001</td>
<td>(State 25)</td>
</tr>
<tr>
<td>011010</td>
<td>(State 26)</td>
</tr>
<tr>
<td>011011</td>
<td>(State 27)</td>
</tr>
<tr>
<td>011100</td>
<td>(State 28)</td>
</tr>
<tr>
<td>011101</td>
<td>(State 29)</td>
</tr>
<tr>
<td>011110</td>
<td>(State 30)</td>
</tr>
<tr>
<td>011111</td>
<td>(State 31)</td>
</tr>
<tr>
<td>100000</td>
<td>(State 32)</td>
</tr>
<tr>
<td>100001</td>
<td>(State 33)</td>
</tr>
<tr>
<td>100010</td>
<td>(State 34)</td>
</tr>
<tr>
<td>100011</td>
<td>(State 35)</td>
</tr>
<tr>
<td>100100</td>
<td>(State 36)</td>
</tr>
<tr>
<td>100101</td>
<td>(State 37)</td>
</tr>
<tr>
<td>100110</td>
<td>(State 38)</td>
</tr>
<tr>
<td>100111</td>
<td>(State 39)</td>
</tr>
<tr>
<td>101000</td>
<td>(State 40)</td>
</tr>
<tr>
<td>101001</td>
<td>(State 41)</td>
</tr>
<tr>
<td>101010</td>
<td>(State 42)</td>
</tr>
<tr>
<td>101011</td>
<td>(State 43)</td>
</tr>
<tr>
<td>101100</td>
<td>(State 44)</td>
</tr>
<tr>
<td>101101</td>
<td>(State 45)</td>
</tr>
<tr>
<td>101110</td>
<td>(State 46)</td>
</tr>
<tr>
<td>101111</td>
<td>(State 47)</td>
</tr>
<tr>
<td>110000</td>
<td>(State 48)</td>
</tr>
<tr>
<td>110001</td>
<td>(State 49)</td>
</tr>
<tr>
<td>110010</td>
<td>(State 50)</td>
</tr>
<tr>
<td>110011</td>
<td>(State 51)</td>
</tr>
<tr>
<td>110100</td>
<td>(State 52)</td>
</tr>
<tr>
<td>110101</td>
<td>(State 53)</td>
</tr>
<tr>
<td>110110</td>
<td>(State 54)</td>
</tr>
<tr>
<td>110111</td>
<td>(State 55)</td>
</tr>
<tr>
<td>111000</td>
<td>(State 56)</td>
</tr>
<tr>
<td>111001</td>
<td>(State 57)</td>
</tr>
<tr>
<td>111010</td>
<td>(State 58)</td>
</tr>
<tr>
<td>111011</td>
<td>(State 59)</td>
</tr>
<tr>
<td>111100</td>
<td>(State 60)</td>
</tr>
<tr>
<td>111101</td>
<td>(State 61)</td>
</tr>
<tr>
<td>111110</td>
<td>(State 62)</td>
</tr>
<tr>
<td>111111</td>
<td>(State 63)</td>
</tr>
</tbody>
</table>
LC-3b Microsequencer

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

- **The purpose of the microsequencer** is to determine the address of the next microinstruction (i.e., next state)
  - Next state could be conditional or unconditional

- Next state address depends on 9 control signals (plus 7 data 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>COND&lt;sub&gt;0&lt;/sub&gt;; Unconditional</td>
</tr>
<tr>
<td>COND&lt;sub&gt;1&lt;/sub&gt;</td>
<td>Memory Ready</td>
</tr>
<tr>
<td>COND&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Branch</td>
</tr>
<tr>
<td>COND&lt;sub&gt;3&lt;/sub&gt;</td>
<td>Addressing Mode</td>
</tr>
<tr>
<td>IRD/1:</td>
<td>NO, YES</td>
</tr>
</tbody>
</table>

Table C.2: Microsequencer control signals
APPENDIX C. THE MICROARCHITECTURE OF THE LC-3B, B ASIC MACHINE

Address of Next State

0,0,IR[15:12] 6

IRD

6

Figure C.5: The microsequencer of the LC-3b base machine

Unused opcodes, the microarchitecture would execute a sequence of microinstructions, starting at state 10 or state 11, depending on which illegal opcode was being decoded. In both cases, the sequence of microinstructions would respond to the fact that an instruction with an illegal opcode had been fetched.

Several signals necessary to control the data path and the microsequencer are not among those listed in Tables C.1 and C.2. They are DR, SR1, BEN, and R. Figure C.6 shows the additional logic needed to generate DR, SR1, and BEN. The remaining signal, R, is a signal generated by the memory in order to allo...
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 (~ control store size)
  - Start with the 16-way branch
  - Then determine constraint tables and states dependent on COND
An Exercise in Microprogramming
Handouts

- 7 pages of Microprogrammed LC-3b design
A Simple LC-3b Control and Datapath

Microarchitecture of the LC-3b, major components

1. J[5:0], COND[1:0], and IRD—9 bits of control signals provided by the current clock cycle.
2. Inst[15:12], which identifies the opcode, and Inst[11:11], which differentiates JSR from JSRR (i.e., the addressing mode for the target of the subroutine call).
3. BEN to indicate whether or not a BR should be taken.

Figure C.1 identifies the specific information in our implementation of the LC-3b that corresponds to these five items.
A Simple Datapath Can Become Very Powerful
Fill in the microinstructions for the 7 states for LDW:  

State Machine for LDW

Microsequencer

<table>
<thead>
<tr>
<th>IRD</th>
<th>Cond</th>
<th>J1</th>
<th>J2</th>
<th>J3</th>
<th>J4</th>
<th>J5</th>
<th>J6</th>
<th>Address of Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>18</td>
<td>19</td>
<td>32</td>
<td>33</td>
<td>35</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

State 18 (010010)
State 33 (100001)
State 35 (100011)
State 32 (100000)
State 6 (000110)
State 25 (011001)
State 27 (011011)
Figure C.6: Additional logic required to provide control signals to operate correctly with a memory that takes multiple clock cycles to read or store a value.

Suppose it takes memory firmware cycles to read a value. That is, once MAR contains the address to be read and the microinstruction asserts READ, it will take firmware cycles before the contents of the specified location in memory are available to be loaded into MDR. (Note that the microinstruction asserts READ by means of three control signals: MIO.EN/YES, R.W/RD, and DA.TA.SIZE/W; see Figure C.3.)

Recall our discussion in Section C.2 of the function of state 33, which accesses an instruction from memory during the fetch phase of each instruction cycle. For the LC-3b to operate correctly, state 33 must execute firmware before moving on to state 35. That is, until MDR contains valid data from the memory location specified by the contents of MAR, we want state 33 to continue to re-execute. After firmware clock cycles, the memory has completed the "read," resulting in valid data in MDR, so the processor can move on to state 35.

What if the microarchitecture did not wait for the memory to complete the read operation before moving on to state 35? Since the contents of MDR would still be garbage, the microarchitecture would put garbage into IR in state 35. The ready signal (R) enables the memory read to execute correctly. Since the memory knows it needs firmware cycles to complete the read, it asserts a ready signal (R) throughout the fifth clock cycle. Figure C.2 shows that the next state is 33 (i.e., 100001) if the memory read will not complete in the current clock cycle and state 35 (i.e., 100011) if it will. As we have seen, it is the job of the microsequencer (Figure C.5) to produce the next state address.
<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>ALUK/2</td>
<td>ADD, AND, XOR, PASSA</td>
</tr>
<tr>
<td>MIO.EN/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
Simple Design of the Control Structure

Microsequencer

Control Store

$2^6 \times 35$

Microinstruction

R

IR[15:11]

BEN

6

35

9

26

(J, COND, IRD)
APPENDIX C. THE MICROARCHITECTURE OF THE LC-3B, B ASIC MACHINE

Figure C.5: The microsequencer of the LC-3b base machine

unused opcodes, the microarchitecture would execute a sequence of microinstructions, starting at state 10 or state 11, depending on which illegal opcode was being decoded. In both cases, the sequence of microinstructions would respond to the fact that an instruction with an illegal opcode had been fetched.

Several signals necessary to control the data path and the microsequencer are not among those listed in Tables C.1 and C.2. They are DR, SR1, BEN, and R.

Figure C.6 shows the additional logic needed to generate DR, SR1, and BEN. The remaining signal, R, is a signal generated by the memory in order to allo...
### APPENDIX C

#### THE MICROARCHITECTURE OF THE LC-3B, B ASIC MACHINE

End of the Exercise in Microprogramming
The ready signal (R) enables memory read/write to execute correctly

- Example: transition from state 33 to state 35 is controlled by the R bit asserted by memory when memory data is available

- Could we have done this in a single-cycle microarchitecture?

- What did we assume about memory and registers 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 instruction in x86
    - string copy of N elements starting from address A to address B
The Power of Abstraction

- The concept of a control store of microinstructions enables the hardware designer with a new abstraction: microprogramming

- The designer can translate any desired operation to a sequence of microinstructions

- All the designer needs to provide is
  - The sequence of microinstructions needed to implement the desired operation
  - The ability for the control logic to correctly sequence through the microinstructions
  - Any additional datapath elements and control signals needed (no need if the operation can be “translated” into existing control signals)
Let’s Do Some More Microprogramming

- Implement REP MOV$ in the LC-3b microarchitecture

- What changes, if any, do you make to the
  - state machine?
  - datapath?
  - control store?
  - microsequencer?

- Show all changes and microinstructions
- Optional HW Assignment
x86 REP MOVVS (String Copy) Instruction

**REP MOVVS (DEST SRC)**

IF AddressSize = 16
   THEN
      Use CX for CountReg;
   ELSE IF AddressSize = 64 and REX.W used
      THEN Use RCX for CountReg; Fl;
   ELSE
      Use ECX for CountReg;
   Fl;
WHILE CountReg ≠ 0
   DO
      Service pending interrupts (if any);
      Execute associated string instruction;
      CountReg ← (CountReg - 1);
      IF CountReg = 0
         THEN exit WHILE loop; Fl;
      IF (Repeat prefix is REPZ or REPE) and (ZF = 0)
         or (Repeat prefix is REPNZ or REPNE) and (ZF = 1)
         THEN exit WHILE loop; Fl;
   OD;

---

**How many instructions does this take in MIPS ISA?**

**How many microinstructions does this take to add to the LC-3b microarchitecture?**
Aside: Alignment Correction in Memory

- **Unaligned accesses**

- LC-3b has byte load and byte store instructions that move data not aligned at the word-address boundary
  - Convenience to the programmer/compiler

- How does the hardware ensure this works correctly?
  - Take a look at state 29 for LDB
  - States 24 and 17 for STB
  - Additional logic to handle unaligned accesses

- P&P, Revised Appendix C.5
Aside: Memory Mapped I/O

- Address control logic determines whether the specified address of LDW and STW are to memory or I/O devices.

- Correspondingly enables memory or I/O devices and sets up muxes.

- An instance where the final control signals of some datapath elements (e.g., MEM.EN or INMUX/2) cannot be stored in the control store.
  - These signals are dependent on memory address.

- P&P, Revised Appendix C.6
Advantages of Microprogrammed Control

- Allows a very simple design to do powerful computation by controlling the datapath (using a sequencer)
  - High-level ISA translated into microcode (sequence of u-instructions)
  - Microcode (u-code) enables a minimal datapath to emulate an ISA
  - Microinstructions can be thought of as a user-invisible ISA (u-ISA)

- Enables easy extensibility of the ISA
  - Can support a new instruction by changing the microcode
  - Can support complex instructions as a sequence of simple microinstructions (e.g., REP MOVS, INC [MEM])

- Enables update of machine behavior
  - A buggy implementation of an instruction can be fixed by changing the microcode in the field
    - Easier if datapath provides ability to do the same thing in different ways
Update of Machine Behavior

- **The ability to update/patch microcode in the field** (after a processor is shipped) enables
  - Ability to add new instructions without changing the processor!
  - Ability to “fix” buggy hardware implementations

- **Examples**
  - IBM 370 Model 145: microcode stored in main memory, can be updated after a reboot
  - IBM System z: Similar to 370/145.
  - B1700 microcode can be updated while the processor is running
    - User-microprogrammable machine!
Multi-Cycle vs. Single-Cycle uArch

- Advantages
- Disadvantages
- For you to fill in
Can We Do Better?