Systems Architecture I

Topics
A Simple Implementation of MIPS*
A Multicycle Implementation of MIPS**

*This lecture was derived from material in the text (sec. 5.1-5.3).
**This lecture was derived from material in the text (sec. 5.4-5.5).


Notes Courtesy of Jeremy R. Johnson
Systems Architecture I

Topic 1: A Simple Implementation of MIPS
Introduction

- **Objective**: To understand how to implement the MIPS instruction set.
- **Combine components** (registers, memory, ALU) and add control
- **Fetch-Execute cycle**
- **Topics**
  - Sequential logic (elements with state) and timing (edge triggered)
    - Memory
    - Registers
  - Datapath components: Instruction memory, PC, Add, Register File, ALU, Data Memory
  - Implement a subset of MIPS in a single cycle computer
  - Shortcomings of a single cycle computer
The Processor: Datapath & Control

• Implementation of MIPS
• Simplified to contain only:
  – memory-reference instructions: lw, sw
  – arithmetic-logical instructions: add, sub, and, or, slt
  – control flow instructions: beq, j
• Generic Implementation:
  – use the program counter (PC) to supply instruction address
  – get the instruction from memory
  – read registers
  – use the instruction to decide exactly what to do
Abstract View

- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain state (sequential)
Timing

- Clocks used in synchronous logic
  - when should an element that contains state be updated?
- Edge-triggered timing

\[ \text{cycle time} \]

\[ \text{falling edge} \]

\[ \text{rising edge} \]
Edge Triggered Timing

- State updated at clock edge
- read contents of some state elements,
- send values through some combinational logic
- write results to one or more state elements
Components for Simple Implementation

- Functional Units needed for each instruction

- a. Instruction memory
- b. Program counter
- c. Adder

- a. Data memory unit
- b. Sign-extension unit

- a. Registers
- b. ALU
- c. ALU control
- d. Data
- e. Address
- f. Read data
- g. Write data
- h. Registers
- i. Read register 1
- j. Read register 2
- k. Write register
- l. Read data 1
- m. Read data 2
- n. RegWrite
- o. MemRead
- p. MemWrite
- q. ALU result
- r. Zero
- s. 16
- t. Sign extend
- u. 32
Building the Datapath

- Use Multiplexors to put components together
Adding Control

- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
MIPS Instructions

- **add $t0,$s1,$s2**

  ![Add Instruction Binary](image)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>10001</td>
<td>10010</td>
<td>01000</td>
<td>00000</td>
<td>100000</td>
</tr>
</tbody>
</table>

- **lw $t0,256($t1)**

  ![Load Word Instruction Binary](image)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>100011</td>
<td>01001</td>
<td>01000</td>
<td>0000 0001 0000 0000</td>
</tr>
</tbody>
</table>

MIPS Instructions Continued

- `beq $s1,$s2,100`

  ```
  000100 10001 10010 0000 0000 0001 1001
  op  rs  rt  offset
  ```

- `j 4096`

  ```
  000010 00 0000 0000 0000 0100 0000 0000
  op  address
  ```
- Control Lines
  000 and
  001 or
  010 add
  110 sub
  111 slt
Determining ALU Control Bits

- ALUOp determined by instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>ALUOp</th>
<th>Instruction funct</th>
<th>ALU</th>
<th>ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>operation</td>
<td>action</td>
<td>control</td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>00</td>
<td>load word</td>
<td>xxxx</td>
<td>add</td>
</tr>
<tr>
<td>SW</td>
<td>00</td>
<td>store word</td>
<td>xxxx</td>
<td>add</td>
</tr>
<tr>
<td>BEQ</td>
<td>01</td>
<td>branch eq</td>
<td>xxxx</td>
<td>sub</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>add</td>
<td>100000</td>
<td>add</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>sub</td>
<td>100100</td>
<td>sub</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>and</td>
<td>100100</td>
<td>and</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>or</td>
<td>100101</td>
<td>or</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>slt</td>
<td>101010</td>
<td>slt</td>
</tr>
</tbody>
</table>
ALU Control

- Must describe hardware to compute 3-bit ALU control input
  - given instruction type
    00 = lw, sw
    01 = beq,
    10 = arithmetic
  - function code for arithmetic
- Describe it using a truth table (can turn into gates):

<table>
<thead>
<tr>
<th>ALUOp1</th>
<th>ALUOp0</th>
<th>F5</th>
<th>F4</th>
<th>F3</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>010</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>111</td>
</tr>
</tbody>
</table>
Datapath with Control
Control Line Settings

- 8 control lines (control read/write and multiplexors)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>Memto-Reg</th>
<th>Reg Write</th>
<th>Mem Read</th>
<th>Mem Write</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Shortcomings of a Single Cycle Implementation

- **Limits reuse of hardware components**
  - each functional unit can be used only once per cycle
  - e.g. instruction and data memory required
- **Inefficient**
  - clock cycle determined by longest possible path in the machine
  - E.G. Assume time for:
    - Memory units = 2 ns
    - ALU and adders = 2 ns
    - Register file (read or write) = 1 ns
  - R-format = Inst Mem + reg read + ALU + reg write = 6 ns
  - Load = Inst Mem + reg read + ALU + Data Mem + reg write = 8 ns
  - Store = Inst Mem + reg read + ALU + Data Mem = 7 ns
  - Branch = Inst Mem + reg read + ALU = 5 ns
Loss of Efficiency Due to Single Cycle

• Assume 24% loads, 12% stores, 44% R-format, 18% branches, and 2% jumps (2 ns)

\[
\frac{\text{CPU perf [var]}}{\text{CPU perf [single]}} = \frac{\text{CPU Time [single]}}{\text{CPU Time [var]}} = \frac{\text{IC} \times \text{CPU clock [single]}}{\text{IC} \times \text{CPU clock [var]}} = \frac{\text{CPU clock [single]}}{\text{CPU clock [var]}} = \frac{8}{6.34} = 1.26
\]

• Next class, we examine a multi-cycle implementation (5.4)
Systems Architecture I

Topic 2:
A Multicycle Implementation of MIPS
Introduction

• Objective: To re-implement the MIPS instruction set using a multicycle implementation. The benefits are shared hardware and that instructions can take a different number of cycles (reduced computing time).
• How: break instructions into steps, each step taking a clock cycle.
• What’s New:
  – control depends on step
  – Need to store temporary values in internal registers
• Topics
  – Steps in multicycle implementation
  – Control using finite state machine
  – Control using microprogramming
Multicycle Approach

- Break up the instructions into steps, each step takes a cycle
  - balance the amount of work to be done
  - restrict each cycle to use only one major functional unit
  - Functional units: memory, register file, and ALU

- At the end of a cycle
  - Use internal registers to store results between steps
Execution Steps

1. **Instruction fetch**
   - IR = Memory[PC];
   - PC = PC + 4;

2. **Instruction decode and register fetch**
   - A = Reg[IR[25-21]];
   - B = Reg[IR[20-16]];
   - ALUOut = PC + (sign-extend (IR[15-0]) << 2);
Execution Steps

3 Execution, memory address computation, or branch completion

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

- Arithmetic-logical instruction (R-type)
  - \( \text{ALUOut} = A \text{ op } B \);

- Branch
  - if \( A == B \) \( \text{PC} = \text{ALUOut} \);

- Jump
  - \( \text{PC} = \text{PC [31-28]} || \text{IR[25-0] <<=2} \)
Execution Steps

4 Memory access or R-type instruction completion
   – memory reference
     • MDR = Memory [ALUOut];
   – or
     • Memory [ALUOut] = B;
   – R-type completion
     • Reg [IR[15-11]] = ALUOut;

5 Memory read completion
   – Reg [IR[20-16]] = MDR;
Summary

- **3 - 5 clock cycles**
- **Uses “optimistic” actions**

<table>
<thead>
<tr>
<th>Step name</th>
<th>Action for R-type instructions</th>
<th>Action for memory-reference instructions</th>
<th>Action for branches</th>
<th>Action for jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction fetch</td>
<td></td>
<td>IR = Memory[PC]</td>
<td>PC = PC + 4</td>
<td></td>
</tr>
<tr>
<td>Instruction decode/ register fetch</td>
<td></td>
<td>A = Reg [IR[25:21]]</td>
<td>B = Reg [IR[20:16]]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALUOut = PC + (sign-extend (IR[15:0]) &lt;&lt; 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution, address computation, branch/jump completion</td>
<td>ALUOut = A op B</td>
<td>ALUOut = A + sign-extend (IR[15:0])</td>
<td>if (A == B) then PC = ALUOut</td>
<td>PC = PC [31:28] II (IR[25:0] &lt;&lt; 2)</td>
</tr>
<tr>
<td>Memory access or R-type completion</td>
<td>Reg [IR[15:11]] = ALUOut</td>
<td>Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory read completion</td>
<td>Load: Reg[IR[20:16]] = MDR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CPI in a Multicycle CPU

• Using the previous multicycle implementation determine the average cycles per instruction (from gcc)
  – 22% loads
  – 11% stores
  – 49% R-format
  – 16% branches
  – 2% jumps

\[
\text{CPI} = 0.22 \times 5 + 0.11 \times 4 + 0.49 \times 4 + 0.16 \times 3 + 0.02 \times 3 \\
= 4.04
\]
## Control Signals

**1-bit control signals**
- RegDst
- RegWrite
- ALUSrcA
- MemRead
- MemWrite
- MemtoReg
- IorD
- IRWrite
- PCWrite
- PCWriteCond

**2-bit control signals**
- **ALUOp**
  - 00 (add)
  - 01 (sub)
  - 10 (funct field)
- **ALUSrcB**
  - 00 (B register)
  - 01 (constant 4)
  - 10 (lower 16 bits from IR)
  - 11 (lower 16 bits from IR shifted left 2 bits)
- **PCSource**
  - 00 (output of ALU = PC+4)
  - 01 (ALUOut = branch addr)
  - 10 (jump target)
Complete Datapath
Finite State Machine

- Set of states
- Next function determined by input and current state
- Output determined by current state and possibly input

- Moore machine (output determined only by current state)
Control Using FSM

Instruction fetch:
- MemRead: ALUSrcA = 0
- IorD = 0
- IRWrite
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

Instruction decode/ register fetch:
- ALUSrcA = 0:
- ALUSrcB = 11:
- ALUOp = 00

Branch completion:
- MemWrite
- IorD = 1

Jump completion:
- MemWrite
- IorD = 1

Execution:
- MemRead
- IorD = 1

ALUSrcA = 1:
ALUSrcB = 10:
ALUOp = 00

Memory address computation:
- (Op = 'LW') or (Op = 'SW')

Write-back step:
- RegWrite
- MemWrite
- MemToReg = 0

R-type completion:
- RegWrite
- MemToReg = 1

Memory access:
- (Op = 'LW') or (Op = 'SW')

Start
Implementation of FSM
Control using Microprogramming

- Represent asserted values on control lines symbolically (microinstructions)
  - Fields: Label, ALU control, SRC1, SRC2, Register control, Memory, PCWrite control, Sequencing
  - specifies non-overlapping set of control signals
- Placed in ROM or PLA (provides address for microinstructions)
- Sequencing mechanism
  - next microinstruction
  - branch to microinstruction (FETCH) for next MIPS instruction
  - choose next microinstruction based on control unit input (dispatch).
  Implemented using a table of addresses (dispatch table).
## Microinstruction Fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Value</th>
<th>Signals active</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALU control</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>ALUOp = 00</td>
<td>Cause the ALU to add.</td>
<td></td>
</tr>
<tr>
<td>Subt</td>
<td>ALUOp = 01</td>
<td>Cause the ALU to subtract; this implements the compare for branches.</td>
<td></td>
</tr>
<tr>
<td>Func code</td>
<td>ALUOp = 10</td>
<td>Use the instruction’s function code to determine ALU control.</td>
<td></td>
</tr>
<tr>
<td><strong>SRC1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>ALUSrcA = 0</td>
<td>Use the PC as the first ALU input.</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>ALUSrcA = 1</td>
<td>Register A is the first ALU input.</td>
<td></td>
</tr>
<tr>
<td><strong>SRC2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>ALUSrcB = 00</td>
<td>Register B is the second ALU input.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ALUSrcB = 01</td>
<td>Use 4 as the second ALU input.</td>
<td></td>
</tr>
<tr>
<td>Extend</td>
<td>ALUSrcB = 10</td>
<td>Use output of the sign extension unit as the second ALU input.</td>
<td></td>
</tr>
<tr>
<td>Extshift</td>
<td>ALUSrcB = 11</td>
<td>Use the output of the shift-by-two unit as the second ALU input.</td>
<td></td>
</tr>
<tr>
<td><strong>Register control</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td></td>
<td>Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.</td>
<td></td>
</tr>
<tr>
<td>Write ALU</td>
<td>RegWrite,</td>
<td>Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RegDst = 1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MemtoReg = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write MDR</td>
<td>RegWrite,</td>
<td>Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RegDst = 0,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MemtoReg = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read PC</td>
<td>MemRead,</td>
<td>Read memory using the PC as address; write result into IR (and the MDR).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lorD = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read ALU</td>
<td>MemRead,</td>
<td>Read memory using the ALUOut as address; write result into MDR.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lorD = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write ALU</td>
<td>MemWrite,</td>
<td>Write memory using the ALUOut as address, contents of B as the data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lorD = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PC write control</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALU</td>
<td>PCSource = 00, PCWrite</td>
<td>Write the output of the ALU into the PC.</td>
<td></td>
</tr>
<tr>
<td>ALUOut-cond</td>
<td>PCSource = 01, PCWriteCond</td>
<td>If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.</td>
<td></td>
</tr>
<tr>
<td>jump address</td>
<td>PCSource = 10, PCWrite</td>
<td>Write the PC with the jump address from the instruction.</td>
<td></td>
</tr>
<tr>
<td><strong>Sequencing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq</td>
<td>AddrCtl = 11</td>
<td>Choose the next microinstruction sequentially.</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>AddrCtl = 00</td>
<td>Go to the first microinstruction to begin a new instruction.</td>
<td></td>
</tr>
<tr>
<td>Dispatch 1</td>
<td>AddrCtl = 01</td>
<td>Dispatch using the ROM 1.</td>
<td></td>
</tr>
<tr>
<td>Dispatch 2</td>
<td>AddrCtl = 10</td>
<td>Dispatch using the ROM 2.</td>
<td></td>
</tr>
</tbody>
</table>
# Microprogram

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register control</th>
<th>Memory</th>
<th>PCWrite control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>PC</td>
<td>Extshft</td>
<td>Read</td>
<td></td>
<td>Dispatch 1</td>
<td></td>
</tr>
<tr>
<td>Mem1</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td></td>
<td></td>
<td>Dispatch 2</td>
<td></td>
</tr>
<tr>
<td>LW2</td>
<td></td>
<td></td>
<td></td>
<td>Read ALU</td>
<td></td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write MDR</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>SW2</td>
<td></td>
<td></td>
<td></td>
<td>Write ALU</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>Rformat1</td>
<td>Func code</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write ALU</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>BEQ1</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td></td>
<td>ALUOut-cond</td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>JUMP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Jump address</td>
<td>Fetch</td>
<td></td>
</tr>
</tbody>
</table>
Implementation