18-447 Lecture 8: Single-Cycle Implementations

James C. Hoe
Dept. of ECE, CMU
February 11, 2009

Announcements: First midterm in class on Monday 2/16
HW 2 due Monday

Handouts: MIPS R4000 ISA Manual (on Blackboard)
HW2 solution (on Blackboard this Friday)
Practice midterm solutions

Instruction Processing FSM

- An ISA describes an abstract FSM
  - state = program visible state
  - next-state logic = instruction execution
- Nice ISAs have atomic instruction semantics
  - one state transition per instruction in abstract FSM
- Implementation FSMs can vary
Program Visible State

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

“Magic” Memory and Register File

- **Combinational Read**
  - output of the read data port is a combinational function of the register file contents and the corresponding read select port

- **Synchronous write**
  - the selected register is updated on the pos-edge clock transition when write enable is asserted
    - Cannot affect read output in between clock edges
    - Can affect read output at clock edges (but who cares?)
Instruction Processing

- 5 generic steps
  - instruction fetch
  - instruction decode and operand fetch
  - ALU/execute
  - memory access (not required by non-mem instructions)
  - write-back

Single-Cycle Datapath for Arithmetic and Logical Instructions
R-Type ALU Instructions

- Assembly (e.g., register-register signed addition)
  \[ \text{ADD } rd_{reg} \, rs_{reg} \, rt_{reg} \]

- Machine encoding

<table>
<thead>
<tr>
<th>0</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>0</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-bit</td>
<td>5-bit</td>
<td>5-bit</td>
<td>5-bit</td>
<td>6-bit</td>
<td>R-type</td>
</tr>
</tbody>
</table>

- Semantics

if \( \text{MEM}[\text{PC}] == \text{ADD } rd \, rs \, rt \)

\[
\text{GPR}[rd] \leftarrow \text{GPR}[rs] + \text{GPR}[rt]
\]

\[
\text{PC} \leftarrow \text{PC} + 4
\]

ADD rd rs rt

if \( \text{MEM}[\text{PC}] == \text{ADD } rd \, rs \, rt \)

\[
\text{GPR}[rd] \leftarrow \text{GPR}[rs] + \text{GPR}[rt]
\]

\[
\text{PC} \leftarrow \text{PC} + 4
\]
### I-Type ALU Instructions

- **Assembly (e.g., register-immediate signed additions)**
  \[
  \text{ADDI} \; r_t \; r_s \; \text{immediate}_{16}
  \]

- **Machine encoding**

  \[
  \begin{array}{cccc}
    \text{ADDI} & \text{rs} & \text{rt} & \text{immediate} \\
    6\text{-bit} & 5\text{-bit} & 5\text{-bit} & 16\text{-bit} \\
  \end{array}
  \]

- **Semantics**
  
  if \( \text{MEM}[PC] = \text{ADDI} \; r_t \; r_s \; \text{immediate} \)
  
  \[
  \begin{align*}
  \text{GPR}[rt] & \leftarrow \text{GPR}[rs] + \text{sign-extend (immediate)} \\
  \text{PC} & \leftarrow \text{PC} + 4
  \end{align*}
  \]
ADDI rt rs immediate\textsubscript{16}

\textbf{Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]}
Single-Cycle Datapath for Data Movement Instructions

Load Instructions

- Assembly (e.g., load 4-byte word)
  \[\text{LW } rt_{\text{reg}} \text{ offset}_{16} \text{ (base}_{\text{reg}})\]
- Machine encoding

<table>
<thead>
<tr>
<th>LW</th>
<th>base</th>
<th>rt</th>
<th>offset</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
  If \(\text{MEM[PC]} = \text{LW } rt \text{ offset}_{16} \text{ (base)}\)
  \[\text{EA} = \text{sign-extend(}\text{offset}\text{)} + \text{GPR[base]}\]
  \(\text{GPR[rt]} \leftarrow \text{MEM[ translate(EA) ]}\)
  \(\text{PC} \leftarrow \text{PC} + 4\)

Let's not worry about delay slots here
**LW Datapath**

```
if MEM[PC]==LW rt offset_{16} (base)
EA = sign-extend(offset) + GPR[base]
GPR[rt] ← MEM[ translate(EA) ]
PC ← PC + 4
```

**Store Instructions**

- **Assembly (e.g., store 4-byte word)**
  
  \[
  \text{SW } rt_{\text{reg}} \text{ offset}_{16} (base_{\text{reg}})
  \]

- **Machine encoding**

<table>
<thead>
<tr>
<th>SW</th>
<th>base</th>
<th>rt</th>
<th>offset</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**

  if MEM[PC]==SW rt offset_{16} (base)

  \[
  EA = \text{sign-extend}(\text{offset}) + \text{GPR}[	ext{base}]
  \]

  \[
  \text{MEM[ translate(EA) ]} ← \text{GPR[rt]}
  \]

  \[
  PC ← PC + 4
  \]
**SW Datapath**

if \( \text{MEM}[\text{PC}] == \text{SW} \ rt \ offset_{16} (\text{base}) \)

\[
\text{EA} = \text{sign-extend}(\text{offset}) + \text{GPR}[\text{base}] \\
\text{MEM}[\text{translate}(\text{EA})] \leftarrow \text{GPR}[\text{rt}] \\
\text{PC} \leftarrow \text{PC} + 4
\]

**Load-Store Datapath**

How to uphold the delayed load semantics?
Datapath for Non-Control Flow Instructions

Single-Cycle Datapath for Control Flow Instructions
Unconditional Jump Instructions

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

- **Machine encoding**
  \[
  \text{J} \quad \text{immediate}_{26} \quad \text{J-type}
  \]

- **Semantics**
  \[
  \text{if MEM[PC]} = \text{J immediate}_{26} \\
  \text{target} = \{ \text{PC}[31:28], \text{immediate}_{26}, 2'b00 \} \\
  \text{PC} \leftarrow \text{target}
  \]

Let's not worry about delay slots here.

Unconditional Jump Datapath

What about JR, JAL, JALR?
Conditional Branch Instructions

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQ</td>
<td>6-bit</td>
<td>5-bit</td>
<td>5-bit</td>
</tr>
</tbody>
</table>
- Semantics
  if \( \text{MEM}[PC] == \text{BEQ } rs \ rt \text{ immediate}_{16} \)
  \[
  \text{target} = PC + \text{sign-extend}(\text{immediate}) \times 4 + 4
  \]
  if \( \text{GPR[rs]} == \text{GPR[rt]} \) then \( PC \leftarrow \text{target} \)
  else \( PC \leftarrow PC + \text{4} \)

Let’s not worry about delay slots here

Conditional Branch Datapath

How to uphold the delayed branch semantics?

**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]**
Putting It All together

Fig 5.24 P&H

**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.] JAL, JR, JALR omitted**
Single-Cycle Control

- As combinational function of Inst=MEM[PC]

\[
\begin{array}{cccccc}
31 & 26 & 21 & 16 & 11 & 6 \\
0 & rs & rt & rd & shamt & funct \\
6-bit & 5-bit & 5-bit & 5-bit & 5-bit & 6-bit \\
\end{array}
\]

R-type

\[
\begin{array}{cccc}
31 & 26 & 21 & 16 \\
opcode & rs & rt & immediate \\
6-bit & 5-bit & 5-bit & 16-bit \\
\end{array}
\]

I-type

\[
\begin{array}{cccc}
31 & 26 \\
opcode & immediate \\
6-bit & 26-bit \\
\end{array}
\]

J-type

- 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>Signal</th>
<th>When De-asserted</th>
<th>When asserted</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegDest</td>
<td>GPR write select according to rt, i.e., inst[20:16]</td>
<td>GPR write select according to rd, i.e., inst[15:11]</td>
<td>opcode==0</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>2nd ALU input from 2nd GPR read port</td>
<td>2nd ALU input from sign-extended 16-bit immediate</td>
<td>(opcode==0) &amp;&amp; (opcode==BEQ) &amp;&amp; (opcode==BNE)</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>Steer ALU result to GPR write port</td>
<td>steer memory load to GPR wr. port</td>
<td>opcode==LW</td>
</tr>
<tr>
<td>RegWrite</td>
<td>GPR write disabled</td>
<td>GPR write enabled</td>
<td>(opcode==SW) &amp;&amp; (opcode==Bxx) &amp;&amp; (opcode==J) &amp;&amp; (opcode==JR)</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>MemRead</td>
<td>Memory read disabled</td>
<td>Memory read port return load value</td>
<td>opcode==LW</td>
</tr>
<tr>
<td>MemWrite</td>
<td>Memory write disabled</td>
<td>Memory write enabled</td>
<td>opcode==SW</td>
</tr>
<tr>
<td>PCSrc₁</td>
<td>According to PCSrc₂</td>
<td>next PC is based on 26-bit immediate jump target</td>
<td>(opcode==J)</td>
</tr>
</tbody>
</table>
| PCSrc₂           | next PC = PC + 4                  | next PC is based on 16-bit immediate branch target | (opcode==Bxx) && "bcond is satisfied"

JR and JALR require additional PCSrc options

ALU Control Table Lookup

- 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.
**Format of the Quiz**

- **Coverage**
  - lectures (L1~L7), HWs, projects, assigned readings (textbooks and papers)

- **Types of questions**
  - freebies: can you remember the materials
  - probing: did you understand the materials
  - applied: can you apply the materials in original thoughts

- **100 minutes, 100 points**
  - if a question is worth 5 points, don't spend 20 minutes on it
  - skip questions you can't do and come back to them later
  - closed-book, one 2-sided 8½x11 crib sheet
  - no calculators

*** Use pencil or black/blue ink only
*** Be on time, 2:30 sharp!!!