Processor Design – Single Cycle Processor

Hung-Wei Tseng
Announcement

- Reading quizzes for 4.5-4.9 due next Tuesday
- No new reading quizzes until midterm (oh! yeah~~~)
- Homework 2 due 4/26
Recap: the stored-program computer

- Store instructions in memory
- The program counter (PC) controls the execution

### Processor

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>120007a30</td>
<td>0f00bb27</td>
<td>ldah gp,15(t12)</td>
</tr>
<tr>
<td>120007a34</td>
<td>509cbd23</td>
<td>lda gp,-25520(gp)</td>
</tr>
<tr>
<td>120007a38</td>
<td>00005d24</td>
<td>ldah t1,0(gp)</td>
</tr>
<tr>
<td>120007a3c</td>
<td>0000bd24</td>
<td>ldah t4,0(gp)</td>
</tr>
<tr>
<td>120007a40</td>
<td>2ca422a0</td>
<td>ldlt0,-23508(t1)</td>
</tr>
<tr>
<td>120007a44</td>
<td>130020e4</td>
<td>beq t0,120007a94</td>
</tr>
<tr>
<td>120007a48</td>
<td>00003d24</td>
<td>ldah t0,0(gp)</td>
</tr>
<tr>
<td>120007a4c</td>
<td>2ca4e2b3</td>
<td>stl zero,-23508(t1)</td>
</tr>
<tr>
<td>800bf9000</td>
<td>00c2e800</td>
<td>12773376</td>
</tr>
<tr>
<td>800bf9004</td>
<td>00000008</td>
<td>8</td>
</tr>
<tr>
<td>800bf9008</td>
<td>00c2f000</td>
<td>12775424</td>
</tr>
<tr>
<td>800bf900c</td>
<td>00000008</td>
<td>8</td>
</tr>
<tr>
<td>800bf9010</td>
<td>00c2f800</td>
<td>12777472</td>
</tr>
<tr>
<td>800bf9014</td>
<td>00000008</td>
<td>8</td>
</tr>
<tr>
<td>800bf9018</td>
<td>00c30000</td>
<td>12779520</td>
</tr>
<tr>
<td>800bf901c</td>
<td>00000008</td>
<td>8</td>
</tr>
</tbody>
</table>
Recap: Clock

• A hardware signal defines when data is valid and stable
  • Think about the clock in real life!
• We use edge-triggered clocking
  • Values stored in the sequential logic is updated only on a clock edge
Recap: MIPS ISA

- **R-type**: add, sub, and etc...
  
  - 6 bits: opcode
  - 5 bits: rs
  - 5 bits: rt
  - 5 bits: rd
  - 5 bits: shift amount
  - 6 bits: funct

- **I-type**: addi, lw, sw, beq, and etc...
  
  - 6 bits: opcode
  - 5 bits: rs
  - 5 bits: rt
  - 16 bits: immediate / offset

- **J-type**: j, jal, and etc...
  
  - 6 bits: opcode
  - 26 bits: target
Outline

• Implementing a Single-cycle MIPS processor
Designing a simple MIPS processor

• Support MIPS ISA in hardware
  • Design the datapath: add and connect all the required elements in the right order
  • Design the control path: control each datapath element to function correctly.

• Starts from designing a single cycle processor
  • Each instruction takes exactly one cycle to execute
Basic steps of execution

- Instruction fetch: where? instruction memory
- Decode:
  - What’s the instruction?
  - Where are the operands? registers
- Execute ALUs
- Memory access
  - Where is my data? data memory
- Write back registers
  - Where to put the result
- Determine the next PC
Recap: MIPS ISA

- **R-type**: add, sub, and etc...
  
  ![R-type diagram]

- **I-type**: addi, lw, sw, beq, and etc...
  
  ![I-type diagram]

- **J-type**: j, jal, and etc...
  
  ![J-type diagram]
Implementing an R-type instruction

- How many of the following datapath elements is necessary for an R-type instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU

A. 1
B. 2
C. 3
D. 4
E. 5

\[
\text{instruction} = \text{MEM}[PC] \\
\text{REG}[rd] = \text{REG}[rs] \text{ op } \text{REG}[rt] \\
\text{PC} = \text{PC} + 4
\]
Implementing an R-type instruction

• What’s right order of accessing the datapath elements for an R-type instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU
  A. I, III, V, IV
  B. IV, I, III, V
  C. I, V, III, IV
  D. IV, V, I, III
  E. none of the above
Implementing an R-type instruction

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shift amt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

```
instruction = MEM[PC]
REG[rd] = REG[rs] op REG[rt]
PC = PC + 4
```

Tell the processor when to start an instruction

Tell the ALU what ALU function to perform
Implementing a load instruction

- How many of the following datapath elements is necessary for a load instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU

A. 1
B. 2
C. 3
D. 4
E. 5

opcode  rs  rt  immediate / offset

instruction = MEM[PC]
REG[rt] = MEM[signext(immediate) + REG[rs]]
PC = PC + 4
Implementing a load instruction

• What’s right order of accessing the datapath elements for a load instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU
  A. IV, I, III, V, II
  B. IV, I, III, II, V
  C. IV, I, V, II, III
  D. IV, I, II, V, III
  E. none of the above
Implementing a load instruction

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate / offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

instruction = MEM[PC]
REG[rt] = MEM[signext(immediate) + REG[rs]]
PC = PC + 4

Set different control signals for different types of instructions

Set to 1 if it’s a load
Set to 0 if it’s a load
Implementing a store instruction

- How many of the following datapath elements is necessary for a store instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU

| A. 1 | MEM[PC] | instruction = MEM[PC] |
| B. 2 | MEM[signext(immediate) + REG[rs]] = REG[rt] |
| C. 3 | PC = PC + 4 |
| D. 4 | 6 bits | 5 bits | 5 bits | 16 bits | opcode rs rt immediate / offset |
| E. 5 |
Implementing a store instruction

- What’s right order of accessing the datapath elements for a store instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU

A. IV, I, III, V, II
B. IV, I, III, II, V
C. IV, I, V, II, III
D. IV, I, II, V, III
E. none of the above
Implementing a store instruction

\[
\text{instruction} = \text{MEM}[\text{PC}]
\]

\[
\text{MEM}[\text{signext(immediate)} + \text{REG}[rs]] = \text{REG}[rt]
\]

\[
\text{PC} = \text{PC} + 4
\]
Implementing a branch instruction

• How many of the following datapath elements is necessary for a branch instruction?
  I. Instruction Memory
  II. Data memory
  III. Register file
  IV. Program counter
  V. ALU

A. 1
B. 2
C. 3
D. 4
E. 5

instruction = MEM[PC]
PC = (REG[rs] == REG[rt]) ? PC + 4 + SignExtImmediate *4 : PC + 4
Implementing a branch instruction

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate / offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

instruction = MEM[PC]
PC = (REG[rs] == REG[rt]) ? PC + 4 + SignExtImmediate *4 : PC + 4

Calculate the target address
Performance of a single-cycle processor

• How many of the following statements about a single-cycle processor is correct?
  • The CPI of a single-cycle processor is always 1
  • If the single-cycle implements lw, sw, beq, and add instructions, the sw instruction determines the cycle time
  • Hardware elements are mostly idle during a cycle
  • We can always reduce the cycle time of a single-cycle processor by supporting fewer instructions

A. 0
B. 1
C. 2
D. 3
E. 4
Do we need to modify the current processor to support “jr”? 

A. Yes. We need both new control and datapath  
B. Yes. We only need to modify datapath  
C. No. But we should modify for better performance  
D. No. Just changing the control signals is fine  
E. Single cycle cannot do jump register
ISA and cycle time

x86 supports add instruction with memory content as a source operand. e.g. add %eax, (%r12) means %eax = %eax + memory[%r12]

If MIPS wants to support this instruction, what modification is necessary? What’s the performance impact?
Q & A