Processor Design — Single Cycle Processor

Hung-Wei Tseng
Recap: the stored-program computer

- Store instructions in memory
- The program counter (PC) controls the execution
Recap: MIPS ISA

- **R-type**: add, sub, and etc...

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

- **I-type**: addi, lw, sw, beq, and etc...

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

- **J-type**: j, jal, and etc...

  
<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>6</td>
</tr>
<tr>
<td>target</td>
<td>26</td>
</tr>
</tbody>
</table>
Outline

• Implementing a MIPS processor
  • Single-cycle processor
  • Pipelined 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: fetch an instruction from memory
- Decode:
  - What's the instruction?
  - Where are the operands?
- Execute
- Memory access
  - Where is my data? (The data memory address)
- Write back
  - Where to put the result
- Determine the next PC
Recap: MIPS ISA

- **R-type**: add, sub, and etc...
  
  \[
  \begin{array}{cccccc}
  \text{Opcode} & \text{rs} & \text{rt} & \text{rd} & \text{Shift amount} & \text{funct} \\
  6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 6 \text{ bits}
  \end{array}
  \]

- **I-type**: addi, lw, sw, beq, and etc...
  
  \[
  \begin{array}{cccc}
  \text{Opcode} & \text{rs} & \text{rt} & \text{Immediate / offset} \\
  6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 16 \text{ bits}
  \end{array}
  \]

- **J-type**: j, jal, and etc...
  
  \[
  \begin{array}{c}
  \text{Opcode} & \text{Target} \\
  6 \text{ bits} & 26 \text{ bits}
  \end{array}
  \]
Implementing an R-type instruction

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

Tell the ALU what ALU function to perform

Tell the Processor when to start an instruction
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
Implementing a store 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]
MEM[signext(immediate) + REG[rs]] = REG[rt]
PC = PC + 4

Set to 0 if it's a store
Set to 1 if it's a store
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