Great Ideas in Computer Architecture

Single Cycle MIPS CPU—Part II

Instructors:
Krste Asanovic, Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa12

10/26/12

Levels of Representation/Interpretation

- High Level Language Program (e.g., C)
  - Compiler
  - Assembly Language Program (e.g., MIPS)
  - Assembler

Machine Language Program (MIPS)

- Direct Machine Instruction
- Machine Architecture Description
- Architecture Implementation
- Logic Circuit Description (Circuit Schematic Diagrams)

Processor Design Process

1. Five steps to design a processor:
   1. Analyze instruction set → datapath requirements
   2. Select set of datapath components & establish clock methodology
   3. Assemble datapath meeting the requirements
   4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
   5. Assemble the control logic
      - Formulate Logic Equations
      - Design Circuits

Agenda

- Datapath Control
- Administrivia
- Control Implementation
The MIPS-lite Subset

- **ADDU and SUBU**
  - `addu rd, rs, rt`
  - `subu rd, rs, rt`
- **OR Immediate**
  - `ori rt, rs, imm16`
- **LOAD and STORE Word**
  - `lw rt, rs, imm16`
  - `sw rt, rs, imm16`
- **BRANCH**
  - `beq rs, rt, imm16`

RTL gives the meaning of the instructions

Let's analyze the instructions in detail:

**ADDU**

- `addu rd, rs, rt`
- From memory: `MEM[PC]`
- Actual operation: `R[rd] ← R[rs] + R[rt]`
- Next instruction's address: `PC ← PC + 4`

**SUBU**

- `subu rd, rs, rt`
- From memory: `MEM[PC]`
- Actual operation: `R[rd] ← R[rs] - R[rt]`
- Next instruction's address: `PC ← PC + 4`

**ORI**

- `ori rt, rs, imm16`
- From memory: `MEM[PC]`
- Actual operation: `R[rt] ← R[rs] | zero_ext(imm16)`
- Next instruction's address: `PC ← PC + 4`

**LOAD**

- `lw rt, rs, imm16`
- From memory: `MEM[R[rs] + sign_ext(imm16)]`
- Next instruction's address: `PC ← PC + 4`

**STORE**

- `sw rt, rs, imm16`
- To memory: `MEM[R[rs] + sign_ext(imm16)] ← R[rt]`
- Next instruction's address: `PC ← PC + 4`

**BEQ**

- `beq rs, rt, imm16`
- If `R[rs] == R[rt]` then `PC ← PC + 4 + (sign_ext(imm16) || 00)`
  - else `PC ← PC + 4`

**Register Transfer Language (RTL)**

- **RTL** gives the meaning of the instructions

  
  ```
  (op, rs, rt, rd, shamt, funct) ← MEM[PC]
  (op, rs, rt,   Imm16) ← MEM[PC]
  ```

  
- **All start by fetching the instruction**

  ```
  ADDU R[rd] ← R[rs] + R[rt]; PC ← PC + 4
  SUBU R[rd] ← R[rs] - R[rt]; PC ← PC + 4
  ORI  R[rt] ← R[rs] | zero_ext(imm16); PC ← PC + 4
  LOAD R[rt] ← MEM[R[rs] + sign_ext(imm16)]; PC ← PC + 4
  STORE MEM[R[rs] + sign_ext(imm16)] ← R[rt]; PC ← PC + 4
  BEQ if ( R[rs] == R[rt] ) then PC ← PC + 4 + (sign_ext(imm16) || 00) else PC ← PC + 4
  ```

**Single Cycle Datapath during Add**

- `R[rd] = R[rs] + R[rt]`
- `nPC_sel=+4`
- `ALUSrc=0`
- `ExtOp=x`
- `RegDst=1`
- `MemtoReg=0`
- `Rd = rt`
- `Rs = rs`
- `Imm16`
- `aluOut=ADD`

**Instruction Fetch Unit at the Beginning of Add**

- Fetch the instruction from Instruction memory:
  - Instruction = `MEM[PC]`
  - `Inst Address`
  - `Inst Memory`
  - `nPC_sel`

**Instruction Fetch Unit at End of Add**

- `PC = PC + 4`
- `Inst Address`
  - `Inst Memory`
  - `aluOut=ADD`
  - `nPC_sel=+4`

**Register File**

- `RegWr=1`
  - `BusW`
  - `BusA`
  - `RegFile`

**Data Memory**

- `MemWr=0`
  - `Data In`
  - `Data Out`
  - `Data Memory`

**Bus**

- `aluOut=ADD`
  - `aluOut=0`
  - `aluOut=ADD`
  - `aluOut=ADD`
  - `aluOut=ADD`
  - `aluOut=ADD`
Single Cycle Datapath during Or Immediate

\[ R[rt] = R[rs] \text{ OR ZeroExt}[imm16] \]

Single Cycle Datapath during Load

\[ R[rt] = \text{Data Memory} \times (R[rs] + \text{SignExt}[imm16]) \]

Single Cycle Datapath during Store

\[ \text{Data Memory} \times (R[rs] + \text{SignExt}[imm16]) = R[rt] \]
### Single Cycle Datapath during Branch

- If \( R_{rs} - R_{rt} = 0 \) then Zero = 1; else Zero = 0

\[
\text{if } (R_{rs} = R_{rt}) \text{ then Zero = 1; else Zero = 0}
\]

### Instruction Fetch Unit at the End of Branch

- If (Zero = 1) then PC = PC + 4 + 4(Ext[imm16]) else PC = PC + 4

\[
\text{if } (\text{Zero} = 1) \text{ then } \text{PC = PC} + 4 + 4(\text{Ext[imm16]}) \text{ else } \text{PC = PC} + 4
\]

- What is encoding of nPC_sel?
  - Direct MUX select?
  - Branch inst./ not branch
- Let’s pick 2nd option

### Summary: Datapath’s Control Signals

- ExtOp: “zero”, “sign”
- ALUsrc: 0 → regB, 1 → imm16
- ALUSrc: “ADD”, “SUB”, “OR”
- MemWr: 0 → write memory, 1 → write register
- ALUctr: ALU;
- MemtoReg: 0 → regB, 1 → imm16

### Agenda

- Datapath Control
- Administrivia
- Control Implementation

---

**Sleek Tablet, but Clumsy Software**

**Move to the Cloud in the Least Expensive iPad Mini**

The iPad Mini was announced Tuesday, with a sell price starting at $399. How does it compare with other tablet models currently available on the market?

---
Given Datapath: RTL → Control

Summary of the Control Signals (1/2)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ori</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of the Control Signals (2/2)

Boolean Expressions for Controller

<table>
<thead>
<tr>
<th>R-type</th>
<th>add</th>
<th>sub</th>
<th>ori</th>
<th>lw</th>
<th>beq</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L-type</td>
<td>add</td>
<td>sub</td>
<td>ori</td>
<td>lw</td>
<td>beq</td>
<td>rs</td>
<td>rt</td>
<td>Imm31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Agenda

• Datapath Control
• Administrivia
• Control Implementation

Administrivia

• Control Implementation
Controller Implementation

Controller Implementation

`AND` logic

`OR` logic

RegOut
ALUSrc
MemToReg
MemWrite
ExtOp
ALUsrc[0]
ALUsrc[1]

AND Control in Logisim

```
AND Control Logic in Logisim
```

Single Cycle Performance

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr</th>
<th>Register</th>
<th>ALU</th>
<th>Memory</th>
<th>Register</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200 ps</td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200 ps</td>
<td>100 ps</td>
<td>700ps</td>
</tr>
<tr>
<td>branch</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200 ps</td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>500 ps</td>
<td>600ps</td>
</tr>
</tbody>
</table>

- What can we do to improve clock rate?
- Will this improve performance as well?
- Want increased clock rate to mean faster programs

And in Conclusion, ...

Single-Cycle Processor

- Five steps to design a processor:
  1. Analyze instruction set \( \rightarrow \)
datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
     - Formulate logic Equations
     - Design Circuits