Single Cycle Datapath Design

Datapath design steps

• Analyze ISA, Obtain datapath requirements
• Select modules for datapath
• Connect the modules and build and datapath
• Generate control signals for each instruction
• Combine control signals, create control logic
MIPS Instructions (1)

- Unsigned addition and Subtraction

  addu rd, rs, rt
  subu rd, rs, rt

- Format

  \[
  \begin{array}{cccccccc}
  & 6\text{-bit} & 5\text{-bit} & 5\text{-bit} & 5\text{-bit} & 6\text{-bit} \\
  R & \text{opcode} & rs & rt & rd & shamt & funct \\
  \hline
  31 & 26 & 25 & 21 & 16 & 10 & 6 & 5 & 0
  \end{array}
  \]

- Operation

  \[
  \begin{align*}
  \text{ADDU} & \quad R[rd] \leftarrow R[rs] + R[rt]; \quad PC \leftarrow PC + 4 \\
  \text{SUBU} & \quad R[rd] \leftarrow R[rs] - R[rt]; \quad PC \leftarrow PC + 4
  \end{align*}
  \]

MIPS Instructions (2)

- Immediate type instruction

  ori rt, rs, imm16

- Format

  \[
  \begin{array}{cccccccc}
  & 6\text{-bit} & 5\text{-bit} & 5\text{-bit} & 16\text{-bit} \\
  I & \text{opcode} & rs & rt & \text{immediate} \\
  \hline
  31 & 26 & 25 & 21 & 16 & 15 & 0
  \end{array}
  \]

- Operation

  \[
  \begin{align*}
  \text{ORI} & \quad R[rt] \leftarrow R[rs] \mid \text{zero_ext(Imm16)}; \quad PC \leftarrow PC + 4
  \end{align*}
  \]
Hardware components

• ALU
  – Operations: add, sub, compare etc
  – Operands: Two 32-bit numbers from registers or extended immediate number

• Immediate number extender

• Program Counter
  – One 32-bit register
  – Add 4 or an immediate number

Hardware components

• Register File
  – 32 32-bit register
  – Read rs, rt
  – Write rt, rd

• Memory
  – Read only Instructions memory, 32-bit address
  – Data memory, read&write, 32-bit address

• Program Counter
  – One 32-bit register
  – Add 4 or an immediate number
Register File

- **Organization**
  - 32 32-bit registers

- **Data Ports**
  - busA, busB: Two 32-bit data output
  - busW: one 32-bit data input

- **Read/Write Control**
  - Ra(5-bit): select the register, copy content to busA
  - Rb(5-bit): select the register, copy content to busb
  - Rw(5-bit): select the register, at clock (clk) rising edge, if writeEnable==1, write the content of busW to selected register

Memory

- **Data Port**
  - Data In: 32-bit data input
  - Data-out: 32-bit data output

- **Read/Write Control**
  - Address: 32-bit address signal. Send the specified memory content to data-out
  - Write-Enable: at clock rising edge, if write-enable is set, writes data-in content to specified address
Instruction Requirements

- **Fetch Instruction**
  - PC: instruction address
  - Fetch instruction using PC specified address

- Update PC
  - Sequential
    - PC = PC + 4
  - Branch
    - PC = Branch Target address
Instruction Requirements

- **Fetch Instruction**
  - PC: instruction address
  - Fetch instruction using PC specified address
- **Update PC**
  - Sequential
    - PC = PC + 4
  - Branch
    - PC = Branch Target address

---

Instruction Requirements

- **Fetch Instruction**
  - PC: instruction address
  - Fetch instruction using PC specified address
- **Update PC**
  - Sequential
    - PC = PC + 4
  - Branch
    - PC = Branch Target address
Add, Sub Requirements

\[ R[rd] = R[rs] \text{ op } R[rt] \]

- \text{addu rd,rs,rt} \quad \text{subu rd,rs,rt}

<table>
<thead>
<tr>
<th>R</th>
<th>6-bit</th>
<th>5-bit</th>
<th>5-bit</th>
<th>5-bit</th>
<th>5-bit</th>
<th>6-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>16</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

Decoder generate RegWr, ALUCtr control signals

ori Requirements

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

- \text{ori rt,rs,imm16}

<table>
<thead>
<tr>
<th>R</th>
<th>6-bit</th>
<th>5-bit</th>
<th>5-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

- Destination is rt, not rd
- ALU input is immediate number
- Immediate number is 16-bit
ori Requirements

Solution: Add two multiplexer

MIPS Instructions (3)

- Load and Store a word (32-bit)
  
  ```
  lw  rt,imm16(rs)  
  sw  rt,imm16(rs)  
  ```

- Format

- Operation

LOAD   R[rt] ← MEM[R[rs]+sign_ext(Imm16)]; PC ← PC+4
STORE   MEM[R[rs]+sign_ext(Imm16)] ← R[rt]; PC ← PC+4
Load Instruction Requirements

\[ R[rt] = \text{Mem}[R[rs]] + \text{SignExt}[\text{imm16}] \]

- \( \text{lw } rt, \text{imm16}(rs) \)

- How to sign extend?
- Load data from where?

ZeroExtender
- \( \rightarrow \) Extender
- Add memory
## Store Instruction Requirements

\[ \text{Mem}[R[rs] + \text{SignExt}[\text{imm16}]] = R[rt] \]

*sw rt, imm16(rs)*

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
</tbody>
</table>

### So far we have

![Diagram of a processor unit](image-url)
Datapath design steps

- Analyze ISA, Obtain datapath requirements
- Select modules for datapath
- Connect the modules and build and datapath
- Generate control signals for each instruction
- Combine control signals, create control logic