CS152
Computer Architecture and Engineering
Lecture 10: Designing Single Cycle Control

September 24, 1997
Dave Patterson (http.cs.berkeley.edu/~patterson)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/
Recap: Summary from last time

° 5 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

° MIPS makes it easier
  • Instructions same size
  • Source registers always in same place
  • Immediates same size, location
  • Operations always on registers/immediates

° Single cycle datapath => CPI=1, CCT => long

° Next time: implementing control
Recap: The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:
  - **R-type**
    - op: operation of the instruction
    - rs, rt, rd: the source and destination registers specifier
    - shamt: shift amount
    - funct: selects the variant of the operation in the “op” field
  - **I-type**
    - op: operation of the instruction
    - rs, rt: the source and destination registers specifier
    - immediate: address offset or immediate value
  - **J-type**
    - op: operation of the instruction
    - target address: target address of the jump instruction

- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination registers specifier
  - shamt: shift amount
  - funct: selects the variant of the operation in the “op” field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction
Recap: The MIPS Subset

- **ADD and subtract**
  - \text{add} \ rd, \ rs, \ rt
  - \text{sub} \ rd, \ rs, \ rt

- **OR Imm:**
  - \text{ori} \ rt, \ rs, \ imm_{16}

- **LOAD and STORE**
  - \text{lw} \ rt, \ rs, \ imm_{16}
  - \text{sw} \ rt, \ rs, \ imm_{16}

- **BRANCH:**
  - \text{beq} \ rs, \ rt, \ imm_{16}

- **JUMP:**
  - \text{j} \ target

\begin{array}{cccccccc}
\text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{funct} \\
6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 6 \text{ bits} \\
\end{array}

\begin{array}{cccc}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 16 \text{ bits} \\
\end{array}

\begin{array}{cc}
\text{op} & \text{target address} \\
6 \text{ bits} & 26 \text{ bits} \\
\end{array}
Recap: A Single Cycle Datapath

- We have everything except control signals (underline)
  - Today’s lecture will show you how to generate the control signals
The Big Picture: Where are We Now?

° The Five Classic Components of a Computer

° Today’s Topic: Designing the Control for the Single Cycle Datapath
Outline of Today’s Lecture

° Recap and Introduction (10 minutes)
° Control for Register-Register & Or Immediate instructions (10 minutes)
° Questions and Administrative Matters (5 minutes)
° Control signals for Load, Store, Branch, & Jump (15 minutes)
° Building a local controller: ALU Control (10 minutes)
° Break (5 minutes)
° The main controller (20 minutes)
° Summary (5 minutes)
RTL: The **Add** Instruction

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</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>

° **add rd, rs, rt**

- **mem[PC]**  
  Fetch the instruction from memory

- **R[rd] <- R[rs] + R[rt]**  
  The actual operation

- **PC <- PC + 4**  
  Calculate the next instruction’s address

©UCB, DAP’ 97
Instruction Fetch Unit at the Beginning of \texttt{Add}

- Fetch the instruction from Instruction memory: \texttt{Instruction $\leftarrow$ mem[PC]}
  - This is the same for all instructions

```
  Inst Memory
    
    Adr

    Instruction<31:0>

    nPCSEL

    4

    Adder

    Adder

    Mux

    00

    PC

    Clk

    imm16

    PCExt
```
The Single Cycle Datapath during \texttt{Add}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
31 & 26 & 21 & 16 & 11 & 6 & 0 \\
\hline
\texttt{op} & \texttt{rs} & \texttt{rt} & \texttt{rd} & \texttt{shamt} & \texttt{funct} \\
\hline
\end{tabular}
\end{center}

\[ R[rd] \leftarrow R[rs] + R[rt] \]

\[ R[d] \leftarrow R[r] + R[t] \]
Instruction Fetch Unit at the End of Add

- \( \text{PC} \leftarrow \text{PC} + 4 \)
  - This is the same for all instructions except: Branch and Jump
The Single Cycle Datapath during Or Immediate

\[ R[rt] \leftarrow R[rs] \text{ or } \text{ZeroExt}[\text{Imm16}] \]
The Single Cycle Datapath during Or Immediate

° R[rt] <- R[rs] or ZeroExt[Imm16]
Questions and Administrative Matters

° Midterm next Wednesday 10/8/97:
  • 5:30pm to 8:30pm, 306 Soda
  • No class on that day

° Midterm reminders:
  • Pencil, calculator, two 8.5” x 11” pages of handwritten notes
  • Sit in every other chair, every other row (odd row & odd seat)

° Meet at LaVal’s pizza after the midterm
  - Need a headcount. How many are definitely coming?
The Single Cycle Datapath during Load

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

\[
\begin{array}{c|c|c|c|c}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
\hline
31 & 26 & 21 & 16 & 0
\end{array}
\]
The Single Cycle Datapath during Store

Data Memory \{R[rs] + \text{SignExt}[imm16]\} \leftarrow R[rt]
The Single Cycle Datapath during Store

- Data Memory \( \{R[rs] + \text{SignExt}[\text{imm16}]\} \leftarrow R[rt] \)
The Single Cycle Datapath during Branch

\[
\begin{array}{cccc}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
31 & 26 & 21 & 16 & 0
\end{array}
\]

° if \((R[rs] - R[rt] == 0)\) then \(\text{Zero} <- 1\); else \(\text{Zero} <- 0\)
Instruction Fetch Unit at the End of Branch

if (Zero == 1) then \( \text{PC} = \text{PC} + 4 + \text{SignExt[imm16]} \times 4 \) ; else \( \text{PC} = \text{PC} + 4 \)
Step 4: Given Datapath: RTL -> Control

Instruction<31:0>

Inst Memory

Adr

Op Fun Rt Rs Rd Imm16

Control

nPC_sel RegWr RegDst ExtOp ALUSrc ALUctr MemWr MemtoReg Equal

DATA PATH
A Summary of Control Signals

<table>
<thead>
<tr>
<th>inst</th>
<th>Register Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R[rd] ← R[rs] + R[rt]; PC ← PC + 4</td>
</tr>
<tr>
<td></td>
<td>ALUsrc = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
</tr>
<tr>
<td>SUB</td>
<td>R[rd] ← R[rs] − R[rt]; PC ← PC + 4</td>
</tr>
<tr>
<td></td>
<td>ALUsrc = RegB, ALUctr = “sub”, RegDst = rd, RegWr, nPC_sel = “+4”</td>
</tr>
<tr>
<td>ORi</td>
<td>R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4</td>
</tr>
<tr>
<td></td>
<td>ALUsrc = Im, Extop = “Z”, ALUctr = “or”, RegDst = rt, RegWr, nPC_sel = “+4”</td>
</tr>
<tr>
<td>LOAD</td>
<td>R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4</td>
</tr>
<tr>
<td></td>
<td>ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemtoReg, RegDst = rt, RegWr, nPC_sel = “+4”</td>
</tr>
<tr>
<td>STORE</td>
<td>MEM[ R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4</td>
</tr>
<tr>
<td></td>
<td>ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemWr, nPC_sel = “+4”</td>
</tr>
<tr>
<td>BEQ</td>
<td>if ( R[rs] == R[rt] ) then PC ← PC + sign_ext(Imm16) ]</td>
</tr>
<tr>
<td></td>
<td>nPC_sel = “Br”, ALUctr = “sub”</td>
</tr>
</tbody>
</table>
# A Summary of the Control Signals

<table>
<thead>
<tr>
<th>See Appendix A</th>
<th>func</th>
<th>op</th>
<th>10 0000</th>
<th>10 0010</th>
<th>We Don’t Care :-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegDst</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nPCsel</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jump</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ExtOp</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>ALUctrl&lt;2:0&gt;</td>
<td>Add</td>
<td>Subtract</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
<th>add, sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td></td>
<td>add, sub</td>
</tr>
<tr>
<td>I-type</td>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
<td>ori, lw, sw, beq</td>
</tr>
<tr>
<td>J-type</td>
<td>op</td>
<td></td>
<td></td>
<td>target address</td>
<td></td>
<td>jump</td>
</tr>
</tbody>
</table>

©UCB, DAP’ 97
The Concept of Local Decoding

<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 1101</th>
<th>10 0011</th>
<th>10 1011</th>
<th>00 0100</th>
<th>00 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R-type</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
<td>jump</td>
</tr>
<tr>
<td>RegDst</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Branch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Jump</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ExtOp</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ALUop&lt;N:0&gt;</td>
<td>“R-type”</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td>x</td>
</tr>
</tbody>
</table>

```
00 0000 00 1101 10 0011 10 1011 00 0100 00 0010
```

![Diagram of control flow and ALU operations](image-url)
The Encoding of ALUop

° In this exercise, ALUop has to be 2 bits wide to represent:
  • (1) “R-type” instructions
  • “I-type” instructions that require the ALU to perform:
    - (2) Or, (3) Add, and (4) Subtract

° To implement the full MIPS ISA, ALUop has to be 3 bits to represent:
  • (1) “R-type” instructions
  • “I-type” instructions that require the ALU to perform:
    - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

<table>
<thead>
<tr>
<th>ALUop (Symbolic)</th>
<th>R-type</th>
<th>ori</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
<th>jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>“R-type”</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td>xxx</td>
<td></td>
</tr>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td>1 00</td>
<td>0 10</td>
<td>0 00</td>
<td>0 00</td>
<td>0 01</td>
<td>xxx</td>
</tr>
</tbody>
</table>
The Decoding of the “func” Field

<table>
<thead>
<tr>
<th>ALUop (Symbolic)</th>
<th>R-type</th>
<th>ori</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
<th>jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 00</td>
<td>0 10</td>
<td>0 00</td>
<td>0 00</td>
<td>0 01</td>
<td>xxx</td>
</tr>
</tbody>
</table>

Recall ALU Homework (also P. 286 text):

<table>
<thead>
<tr>
<th>funct&lt;5:0&gt;</th>
<th>Instruction Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 0000</td>
<td>add</td>
</tr>
<tr>
<td>10 0010</td>
<td>subtract</td>
</tr>
<tr>
<td>10 0100</td>
<td>and</td>
</tr>
<tr>
<td>10 0101</td>
<td>or</td>
</tr>
<tr>
<td>10 1010</td>
<td>set-on-less-than</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALUctr&lt;2:0&gt;</th>
<th>ALU Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Add</td>
</tr>
<tr>
<td>001</td>
<td>Subtract</td>
</tr>
<tr>
<td>010</td>
<td>And</td>
</tr>
<tr>
<td>110</td>
<td>Or</td>
</tr>
<tr>
<td>111</td>
<td>Set-on-less-than</td>
</tr>
</tbody>
</table>
### The Truth Table for ALUctr

<table>
<thead>
<tr>
<th>ALUop (Symbolic)</th>
<th>R-type</th>
<th>ori</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
</tr>
</thead>
<tbody>
<tr>
<td>“R-type”</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td></td>
</tr>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td>1 0 0</td>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALUop bit&lt;2&gt; bit&lt;1&gt; bit&lt;0&gt;</th>
<th>func bit&lt;3&gt; bit&lt;2&gt; bit&lt;1&gt; bit&lt;0&gt;</th>
<th>ALU Operation</th>
<th>ALUctr bit&lt;2&gt; bit&lt;1&gt; bit&lt;0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>x x x x x</td>
<td>Add</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 x 1</td>
<td>x x x x x</td>
<td>Subtract</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 1 x</td>
<td>x x x x x</td>
<td>Or</td>
<td>0 0 1</td>
</tr>
<tr>
<td>1 x x</td>
<td>0 0 0 0 0</td>
<td>Add</td>
<td>0 1 0</td>
</tr>
<tr>
<td>1 x x</td>
<td>0 0 1 0 0</td>
<td>Subtract</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 x x</td>
<td>0 1 0 0 0</td>
<td>And</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 x x</td>
<td>0 1 0 0 1</td>
<td>Or</td>
<td>0 0 1</td>
</tr>
<tr>
<td>1 x x</td>
<td>1 0 1 0 0</td>
<td>Set on &lt;</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>add</td>
</tr>
<tr>
<td>0010</td>
<td>subtract</td>
</tr>
<tr>
<td>0100</td>
<td>and</td>
</tr>
<tr>
<td>0101</td>
<td>or</td>
</tr>
<tr>
<td>1010</td>
<td>set-on-less-than</td>
</tr>
</tbody>
</table>
Break (5 Minutes)
The Logic Equation for ALUctr<2>

<table>
<thead>
<tr>
<th>ALUop</th>
<th>func</th>
<th>ALUctr&lt;2&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit&lt;2&gt;</td>
<td>bit&lt;1&gt;</td>
<td>bit&lt;0&gt;</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

This makes func<3> a don’t care

\[ ALUctr<2> = \neg ALUop<2> \land ALUop<0> + ALUop<2> \land \neg func<2> \land func<1> \land \neg func<0> \]
The Logic Equation for ALUctr<1>

\[ ALUctr<1> = \overline{ALUop<2>} \land \overline{ALUop<0>} + ALUop<2> \land \overline{func<2>} \land \overline{func<0>} \]
The Logic Equation for ALUctr<0>

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
\text{ALUop} & \text{bit<2>} & \text{bit<1>} & \text{bit<0>} & \text{func} & \text{bit<3>} & \text{bit<2>} & \text{bit<1>} & \text{bit<0>} & \text{ALUctr<0>} \\
\hline
0 & 1 & x & x & x & x & 1 & x & x & x & 1 \\
1 & x & x & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & x & x & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
\hline
\end{array}
\]

\[
\text{ALUctr<0>} = \neg\text{ALUop<2>} \& \text{ALUop<0>} \\
+ \text{ALUop<2>} \& \neg\text{func<3>} \& \text{func<2>} \& \neg\text{func<1>} \& \text{func<0>} \\
+ \text{ALUop<2>} \& \text{func<3>} \& \neg\text{func<2>} \& \text{func<1>} \& \neg\text{func<0>}
\]
The ALU Control Block

\[ ALU_{\text{ctr}}^{<2>} = \overline{ALU_{\text{op}}^{<2>}} \land ALU_{\text{op}}^{<0>} + \]
\[ ALU_{\text{op}}^{<2>} \land \overline{\text{func}^{<2>}} \land \text{func}^{<1>} \land \overline{\text{func}^{<0>}} \]

\[ ALU_{\text{ctr}}^{<1>} = \overline{ALU_{\text{op}}^{<2>}} \land \overline{ALU_{\text{op}}^{<0>}} + \]
\[ ALU_{\text{op}}^{<2>} \land \overline{\text{func}^{<2>}} \land \overline{\text{func}^{<0>}} \]

\[ ALU_{\text{ctr}}^{<0>} = \overline{ALU_{\text{op}}^{<2>}} \land ALU_{\text{op}}^{<0>} + \]
\[ ALU_{\text{op}}^{<2>} \land \overline{\text{func}^{<3>}} \land \text{func}^{<2>} \land \overline{\text{func}^{<1>}} \land \text{func}^{<0>} + \]
\[ ALU_{\text{op}}^{<2>} \land \text{func}^{<3>} \land \overline{\text{func}^{<2>}} \land \text{func}^{<1>} \land \overline{\text{func}^{<0>}} \]
Step 5: Logic for each control signal

° nPC_sel <= if (OP == BEQ) then EQUAL else 0

° ALUsrc <= if (OP == "Rtype") then "regB" else "immed"

° ALUctr <= if (OP == "Rtype") then funct
     elseif (OP == ORi) then "OR"
     elseif (OP == BEQ) then "sub"
     else "add"

° ExtOp <= _____________

° MemWr <= _____________

° MemtoReg <= _____________

° RegWr: <= _____________

° RegDst: <= _____________
Step 5: Logic for each control signal

° nPC_sel = if (OP == BEQ) then EQUAL else 0

° ALUsrc = if (OP == “Rtype”) then “regB” else “immed”

° ALUctr = if (OP == “Rtype”) then \text{funct}
  elseif (OP == ORi) then “OR”
  elseif (OP == BEQ) then “sub”
  else “add”

° ExtOp = if (OP == ORi) then “zero” else “sign”

° MemWr = (OP == Store)

° MemtoReg = (OP == Load)

° RegWr: = if ((OP == Store) || (OP == BEQ)) then 0 else 1

° RegDst: = if ((OP == Load) || (OP == ORi)) then 0 else 1
The “Truth Table” for the Main Control

```
<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 1101</th>
<th>10 0011</th>
<th>10 1011</th>
<th>00 0100</th>
<th>00 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R-type</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
<td>jump</td>
</tr>
<tr>
<td>RegDst</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Branch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Jump</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ExtOp</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ALUop (Symbolic)</td>
<td>“R-type”</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td>xxx</td>
</tr>
<tr>
<td>ALUop &lt;2&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>ALUop &lt;1&gt;</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>ALUop &lt;0&gt;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>
```
The “Truth Table” for RegWrite

<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 1101</th>
<th>10 0011</th>
<th>10 1011</th>
<th>00 0100</th>
<th>00 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ori</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>jump</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \text{RegWrite} = \text{R-type} + \text{ori} + \text{lw} \]

\[ = \neg \text{op}_5 \land \neg \text{op}_4 \land \neg \text{op}_3 \land \neg \text{op}_2 \land \neg \text{op}_1 \land \neg \text{op}_0 \quad \text{(R-type)} \]
\[ + \text{op}_5 \land \neg \text{op}_4 \land \text{op}_3 \land \text{op}_2 \land \neg \text{op}_1 \land \text{op}_0 \quad \text{(ori)} \]
\[ + \text{op}_5 \land \text{op}_4 \land \neg \text{op}_3 \land \neg \text{op}_2 \land \text{op}_1 \land \text{op}_0 \quad \text{(lw)} \]
PLA Implementation of the Main Control

R-type  ori  lw  sw  beq  jump

op<5>  op<5>  op<5>  op<5>  op<5>  op<5>
<0>    <0>    <0>    <0>    <0>    <0>

RegWrite
ALUSrc
RegDst
MemtoReg
MemWrite
Branch
Jump
ExtOp
ALUop<2>
ALUop<1>
ALUop<0>
A Real MIPS Datapath (CNS T0)
Putting it All Together: A Single Cycle Processor
Worst Case Timing (Load)

- **Clk**: Clk-to-Q
- **PC**: Instruction Memory Access Time
- **Rs, Rt, Rd, Op, Func**: Delay through Control Logic
- **ALUctr**: Old Value
- **ExtOp**: Old Value
- **ALUSrc**: Old Value
- **MemtoReg**: Old Value
- **RegWr**: Old Value
- **busA**: Delay through Extender & Mux
- **busB**: ALU Delay
- **Address**: Data Memory Access Time
- **busW**: Register File Access Time

**Register Write Occurs**
Drawback of this Single Cycle Processor

° Long cycle time:
  • Cycle time must be long enough for the load instruction:
    PC’s Clock -to-Q +
    Instruction Memory Access Time +
    Register File Access Time +
    ALU Delay (address calculation) +
    Data Memory Access Time +
    Register File Setup Time +
    Clock Skew

° Cycle time for load is much longer than needed for all other instructions
Summary
° Single cycle datapath => CPI=1, CCT => long

° 5 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

° Control is the hard part

° MIPS makes control easier
  • Instructions same size
  • Source registers always in same place
  • Immediates same size, location
  • Operations always on registers/immediates
Where to get more information?

° Chapter 5.1 to 5.3 of your text book:

° One of the best PhD thesis on processor design:

° For a reference on the MIPS architecture:
  • Gerry Kane, “MIPS RISC Architecture,” Prentice Hall.