Recap: A Single Cycle Datapath

- Rs, Rt, Rd and Immed16 hardwired into datapath from Fetch Unit
- We have everything except control signals (underline)

Recap: Flexible Instruction Fetch

- Branch (nPC_sel = "Br"): If (Equal == 1) then PC = PC + 4 + SignExt[imm16]*4; else PC = PC + 4
- Other (nPC_sel = "+4"): PC = PC + 4

Recap: The Single Cycle Datapath during Add

- R[rd] <- R[rs] + R[rt]

Let’s choose second option
Recap: The Single Cycle Datapath during $or$ Immediate

- $R[rt] \leftarrow R[rs]$ or $\text{ZeroExt}[\text{imm16}]$

```
\begin{array}{cccc}
  op & rs & rt & immediate \\
  \hline
  31 & 26 & 21 & 16 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  Rd & Rt & Rs & Clk \\
  \hline
  32 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  ALUctr & Rs & Rd & Clk \\
  \hline
  \text{Add} & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  ALUsrc & WrEn & Addr & Clk \\
  \hline
  1 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  Equal & WrEn & Addr & Clk \\
  \hline
  1 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  ExtOp & Data Memory & WrEn & Clk \\
  \hline
  0 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

Recap: The Single Cycle Datapath during Store

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

```
\begin{array}{cccc}
  op & rs & rt & immediate \\
  \hline
  31 & 26 & 21 & 16 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  Rd & Rt & Rs & Clk \\
  \hline
  32 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  ALUctr & Rs & Rd & Clk \\
  \hline
  \text{Add} & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  ALUsrc & WrEn & Addr & Clk \\
  \hline
  1 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  Equal & WrEn & Addr & Clk \\
  \hline
  1 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  ExtOp & Data Memory & WrEn & Clk \\
  \hline
  0 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

Recap: The Single Cycle Datapath during Branch

- if ($R[rs] - R[rt] == 0$) then $\text{Zero} \leftarrow 1$; else $\text{Zero} \leftarrow 0$

```
\begin{array}{cccc}
  op & rs & rt & immediate \\
  \hline
  31 & 26 & 21 & 16 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  Rd & Rt & Rs & Clk \\
  \hline
  32 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  ALUctr & Rs & Rd & Clk \\
  \hline
  \text{Sub} & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  ALUsrc & WrEn & Addr & Clk \\
  \hline
  1 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  Equal & WrEn & Addr & Clk \\
  \hline
  1 & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  ExtOp & Data Memory & WrEn & Clk \\
  \hline
  x & 32 & 32 & 32 \\
\end{array}
```

```
\begin{array}{cccc}
  Data Memory & MemWr & MemtoReg & Clk \\
  \hline
  32 & 0 & 0 & 0 \\
\end{array}
```

```
\begin{array}{cccc}
  ExtOp & Data Memory & WrEn & Clk \\
  \hline
  x & 32 & 32 & 32 \\
\end{array}
```
Recap: A Summary of Control Signals

<table>
<thead>
<tr>
<th>inst</th>
<th>Register Transfer</th>
<th>ALUsrc</th>
<th>ALUctr</th>
<th>RegDst</th>
<th>RegWr</th>
<th>nPC_{sel}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>R[rd] ← R[rs] + R[rt]; PC ← PC + 4</td>
<td>RegB</td>
<td>“add”</td>
<td>rd</td>
<td>Wr</td>
<td>“+4”</td>
</tr>
<tr>
<td>SUB</td>
<td>R[rd] ← R[rs] - R[rt]; PC ← PC + 4</td>
<td>RegB</td>
<td>“sub”</td>
<td>rd</td>
<td>Wr</td>
<td>“-4”</td>
</tr>
<tr>
<td>ORi</td>
<td>R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4</td>
<td>Im</td>
<td>“Z”</td>
<td>rt</td>
<td>Wr</td>
<td>“+4”</td>
</tr>
<tr>
<td>LOAD</td>
<td>R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4</td>
<td>Im</td>
<td>“add”</td>
<td>rt</td>
<td>Wr</td>
<td>“+4”</td>
</tr>
<tr>
<td>STORE</td>
<td>MEM[ R[rs] + sign_ext(Imm16)] ← R[rt]; PC ← PC + 4</td>
<td>Im</td>
<td>“Su”</td>
<td>rs</td>
<td>Wr</td>
<td>“+4”</td>
</tr>
<tr>
<td>BEQ</td>
<td>if ( R[rs] == R[rt] ) then PC ← PC + sign_ext(Imm16)</td>
<td></td>
<td>00 else PC ← PC + 4</td>
<td>“Br”</td>
<td>“sub”</td>
<td>rs</td>
</tr>
</tbody>
</table>

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>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>RegDst</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
</tr>
<tr>
<td>ALUctr&lt;2:0&gt;</td>
<td>Add</td>
<td>Subtract</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
</tr>
</tbody>
</table>

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>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>RegDst</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Branch</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>
</tr>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td>“R-type” Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td></td>
</tr>
</tbody>
</table>

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>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>RegDst</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Branch</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>
</tr>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td>“R-type” Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td></td>
</tr>
</tbody>
</table>

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>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>RegDst</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Branch</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>
</tr>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td>“R-type” Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td></td>
</tr>
</tbody>
</table>

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>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>RegDst</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Branch</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>
</tr>
<tr>
<td>ALUop&lt;2:0&gt;</td>
<td>“R-type” Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
<td></td>
</tr>
</tbody>
</table>
The Encoding of ALUop

- 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

<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 10</td>
<td>0 00</td>
<td>0 00</td>
<td>0 01</td>
</tr>
</tbody>
</table>

The Decoding of the “func” Field

- Instruction Operation:
- ALU Operation:
- ExtOp:
- ExtOp <= (OP == `ORi) : `ZEROextend : `SIGNextend;
- MemWrite <= (OP == `Store) ? 1 : 0;
- MemtoReg <= (OP == `Load) ? 1 : 0;
- RegWrite <= ((OP == `Store) || (OP == `BEQ)) ? 0 : 1;
- RegDest <= ((OP == `Load) || (OP == `ORi)) ? 0 : 1;

Step 5: Logic for each control signal

- `define Rtype 6'b000000;
- `define BEQ 6'b000100;
- `define Ori 6'b001101;
- `define Load 6'b100011;
- `define Store 6'b101011;
- etc

nPC_sel <= (OP == `BEQ) ? `Br : `plus4;
ALUsrc <= (OP == `Rtype) ? `regB : `immed;
ALUctr <= (OP == `Rtype) ? `funct :
          (OP == `ORi) ? `ORfunction :
          (OP == `BEQ) ? `SUBfunction : `ADDfunction;
ExtOp <= (OP == `ORi) : `ZEROextend : `SIGNextend;
MemWrite <= (OP == `Store) ? 1 : 0;
MemtoReg <= (OP == `Load) ? 1 : 0;
RegWrite <= ((OP == `Store) || (OP == `BEQ)) ? 0 : 1;
RegDest <= ((OP == `Load) || (OP == `ORi)) ? 0 : 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>
</tr>
</thead>
<tbody>
<tr>
<td>RegDst</td>
<td>1</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>or</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ExtOp</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>ALUop&lt;6&gt;</td>
<td>“R-type”</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
</tr>
<tr>
<td>ALUop&lt;5&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALUop&lt;4&gt;</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALUop&lt;3&gt;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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 0100</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\* RegWrite = R-type + ori + lw

\* Design Document for lab 3 due in section this Thursday!

\* Midterm on Wednesday 3/10

\* Midterm on Wednesday 3/10

\* Meet at LaVal’s pizza after the midterm

Administrative Issues

- Read Chapter 5
- This lecture and next one slightly different from the book
- Design Document for lab 3 due in section this Thursday!
  - Describe your division of labor
  - Your testing methodology (how will you test each step of the way?)
  - Top-level block diagrams
- Midterm on Wednesday 3/10
  - 5:30pm to 8:30pm, location TBA
  - No class on that day:
    - Pencil, calculator, one 8.5” x 11” (both sides) of handwritten notes
- Meet at LaVal’s pizza after the midterm
The Big Picture: Where are We Now?

- The Five Classic Components of a Computer
  - Processor
  - Control
  - Memory
  - Datapath
  - Output

Today’s Topic: Designing the Datapath for the Multiple Clock Cycle Datapath

Abstract View of our single cycle processor

- looks like a FSM with PC as state

What’s wrong with our CPI=1 processor?

Arithmetic & Logical

<table>
<thead>
<tr>
<th>PC</th>
<th>Inst Memory</th>
<th>Reg File</th>
<th>mux</th>
<th>ALU</th>
<th>mux</th>
<th>setup</th>
</tr>
</thead>
</table>

Load

| PC | Inst Memory | Reg File | mux | ALU | Data Mem | mux | setup |

Store

| PC | Inst Memory | Reg File | mux | ALU | Data Mem |

Branch

| PC | Inst Memory | Reg File | cmp | mux |

- Long Cycle Time
  - All instructions take as much time as the slowest
  - Real memory is not as nice as our idealized memory
    - cannot always get the job done in one (short) cycle

Memory Access Time

- Physics => fast memories are small (large memories are slow)

Storage Array

- selected word line
- storage cell
- bit line
- sense amps

- question: register file vs. memory

- => Use a hierarchy of memories
  - 1 time-period
  - 2-3 time-periods
  - 20 - 50 time-periods
Reducing Cycle Time

° Cut combinational dependency graph and insert register / latch
° Do same work in two fast cycles, rather than one slow one
° May be able to short-circuit path and remove some components for some instructions!

Worst Case Timing (Load)

Basic Limits on Cycle Time

° Next address logic
  • PC <= branch ? PC + offset : PC + 4
° Instruction Fetch
  • InstructionReg <= Mem[PC]
° Register Access
  • A <= R[rs]
° ALU operation
  • R <= A + B

Partitioning the CPI=1 Datapath

° Add registers between smallest steps
° Place enables on all registers
Example Multicycle Datapath

Recall: Step-by-step Processor Design

Step 1: ISA => Logical Register Transfers
Step 2: Components of the Datapath
Step 3: RTL + Components => Datapath
Step 4: Datapath + Logical RTs => Physical RTs
Step 5: Physical RTs => Control

Step 4: R-type (add, sub, ...)

Logical Register Transfer
ADDU R[rd] ← R[rs] + R[rt]; PC ← PC + 4

Physical Register Transfers
ADDU A ← R[rs]; B ← R[rt]
S ← A + B
R[rd] ← S; PC ← PC + 4

Step 4: Logical immed

Logical Register Transfer
ORI R[rt] ← R[rs] OR ZExt(Im16); PC ← PC + 4

Physical Register Transfers
ORI A ← R[rs]; B ← R[rt]
S ← A or ZExt(Im16)
R[rt] ← S; PC ← PC + 4
### Step 4: Load

<table>
<thead>
<tr>
<th>Logical Register Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>LW ( R[rt] \leftarrow \text{MEM}[R[rs] + \text{SExt(Im16)}] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Register Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR ( \leftarrow \text{MEM}[pc] )</td>
</tr>
<tr>
<td>LW ( A \leftarrow R[rs]; B \leftarrow R[rt] )</td>
</tr>
<tr>
<td>S ( \leftarrow A + \text{SExt(Im16)} )</td>
</tr>
<tr>
<td>M ( \leftarrow \text{MEM}[S] )</td>
</tr>
<tr>
<td>( R[rd] \leftarrow M; ): PC ( \leftarrow \text{PC} + 4 )</td>
</tr>
</tbody>
</table>

### Step 4: Store

<table>
<thead>
<tr>
<th>Logical Register Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW ( \text{MEM}[R[rs] + \text{SExt(Im16)}] \leftarrow R[rt]; )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Register Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR ( \leftarrow \text{MEM}[pc] )</td>
</tr>
<tr>
<td>SW ( A \leftarrow R[rs]; B \leftarrow R[rt] )</td>
</tr>
<tr>
<td>S ( \leftarrow A + \text{SExt(Im16)} )</td>
</tr>
<tr>
<td>( \text{MEM}[S] \leftarrow B )</td>
</tr>
</tbody>
</table>

### Step 4: Branch

<table>
<thead>
<tr>
<th>Logical Register Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQ if ( R[rs] ) == ( R[rt] ) then PC ( \leftarrow \text{PC} + 4 + \text{SExt(Im16)}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Register Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR ( \leftarrow \text{MEM}[pc] )</td>
</tr>
<tr>
<td>BEQ E ( \leftarrow (R[rs] = R[rt]) )</td>
</tr>
<tr>
<td>if ( !E ) then PC ( \leftarrow \text{PC} + 4 ) else PC ( \leftarrow \text{PC} + 4 + \text{SExt(Im16)}</td>
</tr>
</tbody>
</table>

### Alternative Datapath (book): Multiple Cycle Datapath

- Minimizes Hardware: 1 memory, 1 adder

[Diagram of Alternative Datapath]
Our Control Model

- State specifies control points for Register Transfer
- Transfer occurs upon exiting state (same falling edge)

Step 4 ⇒ Control Specification for multicycle proc

- IR <= MEM[PC] “instruction fetch”
- A <= R[rs] B <= R[rt] “decode / operand fetch”

Traditional FSM Controller

<table>
<thead>
<tr>
<th>state</th>
<th>op</th>
<th>cond</th>
<th>next state</th>
<th>control points</th>
</tr>
</thead>
</table>

Truth Table

Step 5 ⇒ (datapath + state diagram ⇒ control)

- Translate RTs into control points
- Assign states
- Then go build the controller
Mapping RTs to Control Points

IR <= MEM[PC]
“A instruction fetch”

R-type
A <= R[rs]
B <= R[rt]
S <= A and B
R[rd] <= S
PC <= PC + 4

ORi
S <= A or B
R[rt] <= S
PC <= PC + 4

LW
S <= A + B
MEM[PC] <= B
PC <= PC + 4

BEQ
PC <= Next(PC)

SW
“instruction fetch”

Assigning States

IR <= MEM[PC]
“A instruction fetch”

R-type
A <= R[rs]
B <= R[rt]
S <= A and B
R[rd] <= S
PC <= PC + 4

ORi
S <= A or B
R[rt] <= S
PC <= PC + 4

LW
S <= A + B
MEM[PC] <= B
PC <= PC + 4

BEQ
PC <= Next(PC)

Performance Evaluation

° What is the average CPI?
• state diagram gives CPI for each instruction type
• workload gives frequency of each type

<table>
<thead>
<tr>
<th>Type</th>
<th>CPI, for type</th>
<th>Frequency</th>
<th>CPI_i x freq_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arith/Logic</td>
<td>4</td>
<td>40%</td>
<td>1.6</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
<td>30%</td>
<td>1.5</td>
</tr>
<tr>
<td>Store</td>
<td>4</td>
<td>10%</td>
<td>0.4</td>
</tr>
<tr>
<td>branch</td>
<td>3</td>
<td>20%</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Average CPI: 4.1
Controller Design

- The state diagrams that arise define the controller for an instruction set processor are highly structured.
- Use this structure to construct a simple “microsequencer”.
- Control reduces to programming this very simple device.
  \[ \rightarrow \text{microprogramming} \]

Example: Jump-Counter

Using a Jump Counter

Our Microsequencer
### Microprogram Control Specification

<table>
<thead>
<tr>
<th>lPC</th>
<th>Taken</th>
<th>Next</th>
<th>IR</th>
<th>PC</th>
<th>en sel</th>
<th>Ops</th>
<th>A</th>
<th>B</th>
<th>Exec</th>
<th>Sr</th>
<th>ALU</th>
<th>S</th>
<th>M</th>
<th>W</th>
<th>M-R</th>
<th>Wr</th>
<th>Dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>?</td>
<td>inc</td>
<td>1</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>zero</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>x</td>
<td>inc</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>x</td>
<td>zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>x</td>
<td>inc</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>x</td>
<td>zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>x</td>
<td>inc</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>x</td>
<td>inc</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>x</td>
<td>zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>x</td>
<td>inc</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>x</td>
<td>zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Overview of Control

- Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.

### Initial Representation

- Finite State Diagram
- Microprogram

### Sequencing Control

- Explicit Next State Function
- Microprogram counter + Dispatch ROMs

### Logic Representation

- Logic Equations
- Truth Tables

### Implementation Technique

- PLA
  - “hardwired control”
- ROM
  - “microprogrammed control”

### Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load

- Multiple Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle

- Partition datapath into equal size chunks to minimize cycle time
  - ~10 levels of logic between latches

- Follow same 5-step method for designing “real” processor

### Summary (cont’d)

- Control is specified by finite state diagram
- Specialize state-diagrams easily captured by microsequencer
  - simple increment & “branch” fields
  - datapath control fields
- Control design reduces to Microprogramming
- Control is more complicated with:
  - complex instruction sets
  - restricted datapaths (see the book)
- Simple Instruction set and powerful datapath ⇒ simple control
  - could try to reduce hardware (see the book)
  - rather go for speed ⇒ many instructions at once!