Reconfigurable Computing

Multicycle RISC Processor

*It’s got multiple personalities.*

- Woody Harrelson

Philip Leong (philip.leong@sydney.edu.au)
School of Electrical and Information Engineering
http://www.ee.usyd.edu.au/~phwl

lecture slides adapted from Patterson and Hennesy,
inst.eecs.berkeley.edu/~cs152/
Step 5: Assemble Control logic

Instruction<31:0>

Inst Memory

Op Fun Rt Rs Rd Imm16

Decoder

nPC_sel RegWr RegDst ExtOp ALUSrc ALUctr MemWr MemtoReg Equal

DATA PATH
### A Summary of the Control Signals

See Appendix A

<table>
<thead>
<tr>
<th>func</th>
<th>op</th>
<th>add</th>
<th>sub</th>
<th>ori</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0000</td>
<td>10 0000</td>
<td>10 0010</td>
<td>We Don’t Care :-)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RegDst</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegWrite</th>
<th>MemWrite</th>
<th>nPCsel</th>
<th>ExtOp</th>
<th>ALUctr&lt;2:0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>Add</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Subtract</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>Or</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Add</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>Add</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td></td>
<td>Subtract</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>
</tr>
</thead>
<tbody>
<tr>
<td>add, sub</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ori, lw, sw, beq</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>target address</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump</td>
<td></td>
</tr>
</tbody>
</table>

---

©UCB Spring 2004

2/23/04
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></td>
<td>R-type</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
</tr>
<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>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>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</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;N:0&gt;</td>
<td>“R-type”</td>
<td>Or</td>
<td>Add</td>
<td>Add</td>
<td>Subtract</td>
</tr>
</tbody>
</table>

Diagram:

- op (6)
- Main Control
- ALU Control (Local)
- ALU
- ALUop (N)
- func (6)
- ALUctr (3)
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>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>
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>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;R-type&quot;</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 00</td>
<td>0 10</td>
<td>0 00</td>
<td>0 00</td>
<td>0 01</td>
</tr>
</tbody>
</table>
```

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>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUctr&lt;2:0&gt;</td>
<td>ALU Operation</td>
</tr>
<tr>
<td>000</td>
<td>And</td>
</tr>
<tr>
<td>001</td>
<td>Or</td>
</tr>
<tr>
<td>010</td>
<td>Add</td>
</tr>
<tr>
<td>110</td>
<td>Subtract</td>
</tr>
<tr>
<td>111</td>
<td>Set-on-less-than</td>
</tr>
</tbody>
</table>
```

©UCB Spring 2004

CS152 / Kubiatowicz

Lec8.7
# 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 00</td>
<td>0 10</td>
<td>0 00</td>
<td>0 00</td>
<td>0 01</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>

<table>
<thead>
<tr>
<th>ALUop&lt;2:0&gt;</th>
<th>ALUop&lt;1&gt;</th>
<th>ALUop&lt;0&gt;</th>
<th>func&lt;3&gt;</th>
<th>func&lt;2&gt;</th>
<th>func&lt;1&gt;</th>
<th>func&lt;0&gt;</th>
<th>ALU Operation</th>
<th>ALUctr&lt;2&gt;</th>
<th>ALUctr&lt;1&gt;</th>
<th>ALUctr&lt;0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Add</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Subtract</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Or</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Add</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Subtract</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>And</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Or</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Set on &lt;</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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;
MemWr <= (OP == `Store) ? 1 : 0;
MemtoReg<= (OP == `Load) ? 1 : 0;
RegWr: <= ((OP == `Store) || (OP == `BEQ)) ? 0 : 1;
RegDst: <= ((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>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>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>nPC_sel</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>0</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>ALUop &lt;2&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALUop &lt;1&gt;</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALUop &lt;0&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 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</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>
</tr>
<tr>
<td>lw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</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>
</tr>
<tr>
<td>jump</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{RegWrite} = \text{R-type} + \text{ori} + \text{lw} = !\text{op}<5> & !\text{op}<4> & !\text{op}<3> & !\text{op}<2> & !\text{op}<1> & !\text{op}<0> \quad (\text{R-type}) \\
+ !\text{op}<5> & !\text{op}<4> & \text{op}<3> & \text{op}<2> & !\text{op}<1> & \text{op}<0> \quad (\text{ori}) \\
+ \text{op}<5> & !\text{op}<4> & !\text{op}<3> & !\text{op}<2> & \text{op}<1> & \text{op}<0> \quad (\text{lw})
\]
PLA Implementation of the Main Control
The Big Picture: Where are We Now?

° The Five Classic Components of a Computer

° 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?

- 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)

- question: register file vs. memory

- => Use a hierarchy of memories

![Diagram of memory hierarchy]

- Processor
- Cache (1 time-period)
- L2 Cache (2-3 time-periods)
- Memory (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!
Basic Limits on Cycle Time

- Next address logic
  - \( PC \leq \text{branch} \ ? \ PC + \text{offset} : PC + 4 \)

- Instruction Fetch
  - \( \text{InstructionReg} \leq \text{Mem}[PC] \)

- Register Access
  - \( A \leq R[rs] \)

- ALU operation
  - \( R \leq A + B \)

Diagram:

- Next PC
- Instruction Fetch
- Operand Fetch
- Exec
- Mem Access
- Reg. File
- Data Mem
- Result Store

Control signals:
- \( nPC\_sel \)
- ExtOp
- ALUSrc
- ALUctr
- MemRd
- MemWr
- RegDst
- RegWr
- MemWr

Diagram labels:
- CS152 / Kubiatowicz
- Lec8.19
- 2/23/04
- ©UCB Spring 2004
Partitioning the CPI=1 Datapath

° Add registers between smallest steps

° Place enables on all registers
Example Multicycle Datapath

Critical Path?

2/23/04
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-rtype (add, sub, . . .)

° Logical Register Transfer

\[
\text{ADDU} \quad R[rd] \leftarrow R[rs] + R[rt]; \quad PC \leftarrow PC + 4
\]

° Physical Register Transfers

\[
\begin{align*}
\text{IR} & \leftarrow \text{MEM}[pc] \\
\text{ADDU} & \quad A \leftarrow R[rs]; \quad B \leftarrow R[rt] \\
S & \leftarrow A + B \\
R[rd] & \leftarrow S; \quad PC \leftarrow PC + 4
\end{align*}
\]
Step 4: Logical immed

- **Logical Register Transfer**
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Logical Register Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORI</td>
<td>R[rt] ← R[rs] OR ZExt(Im16); PC ← PC + 4</td>
</tr>
</tbody>
</table>

- **Physical Register Transfers**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Physical Register Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IR ← MEM[pc]</td>
</tr>
<tr>
<td>ORI</td>
<td>A ← R[rs]; B ← R[rt]</td>
</tr>
<tr>
<td></td>
<td>S ← A or ZExt(Im16)</td>
</tr>
<tr>
<td></td>
<td>R[rt] ← S; PC ← PC + 4</td>
</tr>
</tbody>
</table>
Step 4: Load

- Logical Register Transfer

Logical Register Transfers

LW
R[rt] ← MEM[R[rs] + SExt(Im16)];
PC ← PC + 4

- Physical Register Transfers

Physical Register Transfers

IR ← MEM[pc]
LW
A ← R[rs]; B ← R[rt]
S ← A + SExt(Im16)
M ← MEM[S]
R[rd] ← M;
PC ← PC + 4

Diagram:

Time

Next PC

PC

Inst. Mem

Reg File

IR

E

A

B

Reg. File

Exec

S

M

Mem Access

Data Mem

Reg. File

PC
Step 4: Store

- **Logical Register Transfer**

  - **Logical Register Transfers**
    - \( \text{SW} \): \( \text{MEM}[\text{R}[rs] + \text{SExt(Im16)}] \leftarrow \text{R}[rt] \);
    - \( \text{PC} \leftarrow \text{PC} + 4 \)

- **Physical Register Transfers**

  - **Physical Register Transfers**
    - \( \text{IR} \leftarrow \text{MEM}[\text{pc}] \)
    - \( \text{SW} \): \( \text{A} \leftarrow \text{R}[rs]; \text{B} \leftarrow \text{R}[rt] \)
    - \( \text{S} \leftarrow \text{A} + \text{SExt(Im16)} \);
    - \( \text{MEM}[S] \leftarrow \text{B} \)
    - \( \text{PC} \leftarrow \text{PC} + 4 \)
Step 4: Branch

- **Logical Register Transfer**

  inst Logical Register Transfers

  \[
  \text{BEQ} \quad \text{if } R[rs] == R[rt] \\
  \text{then } PC <= PC + 4 + \text{SExt(Im16)} || 00 \\
  \text{else } PC <= PC + 4
  \]

- **Physical Register Transfers**

  inst Physical Register Transfers

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

---

2/23/04

©UCB Spring 2004
Alternative datapath (book): Multiple Cycle Datapath

- Minimizes Hardware: 1 memory, 1 adder
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"

R-type
S <= A fun B
S <= A or ZX
S <= A + SX

ORi

LW

S <= A + SX
M <= MEM[S]
R[rd] <= S
PC <= PC + 4
R[rt] <= S
PC <= PC + 4
R[rt] <= M
PC <= PC + 4
MEM[S] <= B
PC <= Next(PC, Equal)

SW

M <= MEM[S]
MEM[S] <= B
PC <= PC + 4

BEQ

PC <= Next(PC, Equal)

"instruction fetch"

"decode / operand fetch"
```
### Traditional FSM Controller

#### Truth Table

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

#### Diagram

- **Equal**
  - State: 4
  - next State: 11
  - control points

- **datapath State**
  - **op**: 6
  - **next State**: 11
  - **control points**
Step 5 ⇒ (datapath + state diagram ⇒ control)

° Translate RTs into control points
° Assign states

° Then go build the controller
Recall Multicycle Datapath

Critical Path?
Mapping RTs to Control Points

```
IR <= MEM[PC]
imem_rd, IRen

A <= R[rs]
B <= R[rt]
Aen, Ben, Een

A <= R[rs]
B <= R[rt]
Aen, Ben, Een

PC <= PC + 4

S <= A fun B
ALUfun, Sen

S <= A or ZX

S <= A + SX

S <= A + SX

S <= A + SX

PC <= Next(PC,Equal)

M <= MEM[S]

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

R[rd] <= S
RegDst, RegWr, PCen

R[rt] <= S
PC <= PC + 4

R[rt] <= M
PC <= PC + 4

ORi

LW

SW

BEQ

Write-back Memory

Execute

©UCB Spring 2004

2/23/04
```
## (Mostly) Detailed Control Specification (missing $\Rightarrow 0$)

<table>
<thead>
<tr>
<th>State</th>
<th>Op field</th>
<th>Eq</th>
<th>Next</th>
<th>IR</th>
<th>PC en sel</th>
<th>Ops A B E</th>
<th>Exec Ex Sr ALU S</th>
<th>Mem R W M</th>
<th>Write-Back M-R Wr Dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>??????? ?</td>
<td>0001</td>
<td>0011</td>
<td>1</td>
<td>1 1 1</td>
<td>-all same in Moore machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>BEQ x</td>
<td>0011</td>
<td>0000</td>
<td>1</td>
<td>0</td>
<td>x 0 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>R-type x</td>
<td>0011</td>
<td>0000</td>
<td>1</td>
<td>1 1 1</td>
<td>x 0 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>ORI x</td>
<td>0011</td>
<td>0000</td>
<td>1</td>
<td>1 1 1</td>
<td>x 0 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>LW x</td>
<td>0111</td>
<td>0000</td>
<td>1</td>
<td>1 1 1</td>
<td>x 0 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>SW x</td>
<td>1011</td>
<td>0000</td>
<td>1</td>
<td>1 1 1</td>
<td>x 0 x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BEQ**

- 0011 xxxxxxx 0
- 0011 xxxxxxx 1

**R:**

- 0100 xxxxxxx x
- 0101 xxxxxxx x

**ORi:**

- 0110 xxxxxxx x
- 0111 xxxxxxx x

**LW:**

- 1000 xxxxxxx x
- 1001 xxxxxxx x
- 1010 xxxxxxx x

**SW:**

- 1011 xxxxxxx x
- 1100 xxxxxxx x
- 1100 xxxxxxx x
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&lt;sub&gt;i&lt;/sub&gt; for type</th>
<th>Frequency</th>
<th>CPI&lt;sub&gt;i&lt;/sub&gt; x freq&lt;sub&gt;i&lt;/sub&gt;</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

⇒ microprogramming

![Diagram of controller design](image-url)
Example: Jump-Counter

op-code

Map ROM

Counter

zero
inc
load

None of above: Do nothing (for wait states)
Using a Jump Counter

```
IR <= MEM[PC] 0000 inc

A <= R[rs] 0001
B <= R[rt]

S <= A fun B 0100
S <= A or ZX 0110
S <= A + SX 1000 inc
S <= A + SX 1011

R[rd] <= S 0101
R[rt] <= S 0111
R[rt] <= M 1100 inc
MEM[S] <= B 1011 inc
MEM[S] <= B 1011
PC <= Next(PC) 0011 inc
PC <= PC + 4 1100 inc
PC <= PC + 4 1010 inc
```

“instruction fetch”
“decode”

2/23/04
Our Microsequencer

- Map ROM
- Micro-PC
- Z I L
- Datapath control
- Taken
- Op-code
## Microprogram Control Specification

<table>
<thead>
<tr>
<th>µPC</th>
<th>Equal</th>
<th>Next</th>
<th>IR</th>
<th>PC en sel</th>
<th>Ops A B</th>
<th>Exec Ex Sr ALU S</th>
<th>Mem R W M</th>
<th>Write-Back M-R Wr Dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>x</td>
<td>inc</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>x</td>
<td>load</td>
<td></td>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>zero</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>zero</td>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>x</td>
<td>inc</td>
<td>1 0</td>
<td>0 1 fun 1</td>
<td></td>
<td></td>
<td>0 1 1</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>x</td>
<td>zero</td>
<td>1 0</td>
<td>0 0 or 1</td>
<td></td>
<td></td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>x</td>
<td>inc</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>x</td>
<td>zero</td>
<td>1 0</td>
<td>1 0 add 1</td>
<td></td>
<td></td>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>x</td>
<td>inc</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>x</td>
<td>inc</td>
<td>1 0</td>
<td>1 0 add 1</td>
<td></td>
<td></td>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>x</td>
<td>zero</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td>1 1 0</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>x</td>
<td>inc</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>x</td>
<td>zero</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Microprogramming is a fundamental concept

- implement an instruction set by building a very simple processor and interpreting the instructions
- essential for very complex instructions and when few register transfers are possible
- overkill when ISA matches datapath 1-1
Microprogramming

- Microprogramming is a convenient method for implementing structured control state diagrams:
  - Random logic replaced by microPC sequencer and ROM
  - Each line of ROM called a $\mu$-instruction:
    contains sequencer control + values for control points
  - limited state transitions:
    branch to zero, next sequential,
    branch to $\mu$-instruction address from dispath ROM

- Horizontal $\mu$Code: one control bit in $\mu$-Instruction for every control line in datapath

- Vertical $\mu$Code: groups of control-lines coded together in $\mu$-Instruction (e.g. possible ALU dest)

- Control design reduces to Microprogramming
  - Part of the design process is to develop a “language” that describes control and is easy for humans to understand
“Macroinstruction” Interpretation

User program plus Data

this can change!

one of these is mapped into one of these

AND microsequence

e.g., Fetch Calc Operand Addr Fetch Operand(s) Calculate Save Answer(s)
Designing a Microinstruction Set

1) Start with list of control signals
2) Group signals together that make sense (vs. random): called “fields”
3) Place fields in some logical order
   (e.g., ALU operation & ALU operands first and microinstruction sequencing last)
4) To minimize the width, encode operations that will never be used at the same time
5) Create a symbolic legend for the microinstruction format, showing name of field values and how they set the control signals
   • Use computers to design computers
Recap: Multicycle datapath (book)
Step 1 ⇒ Start with List of control signals

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Effect when deasserted</th>
<th>Effect when asserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUSelA</td>
<td>1st ALU operand = PC</td>
<td>1st ALU operand = Reg[rs]</td>
</tr>
<tr>
<td>RegWrite</td>
<td>None</td>
<td>Reg. is written</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>Reg. write data input = ALU</td>
<td>Reg. write data input = memory RegDst</td>
</tr>
<tr>
<td></td>
<td>Reg. dest. no. = rt</td>
<td>Reg. dest. no. = rd</td>
</tr>
<tr>
<td>MemRead</td>
<td>None</td>
<td>Memory at address is read,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDR &lt;= Mem[addr]</td>
</tr>
<tr>
<td>MemWrite</td>
<td>None</td>
<td>Memory at address is written</td>
</tr>
<tr>
<td>IorD</td>
<td>Memory address = PC</td>
<td>Memory address = S</td>
</tr>
<tr>
<td>IRWrite</td>
<td>None</td>
<td>IR &lt;= Memory</td>
</tr>
<tr>
<td>PCWrite</td>
<td>None</td>
<td>PC &lt;= PCSource</td>
</tr>
<tr>
<td>PCWriteCond</td>
<td>None</td>
<td>IF ALUzero then PC &lt;= PCSource</td>
</tr>
<tr>
<td>PCSource</td>
<td>PCSource = ALU</td>
<td>PCSource = ALUout</td>
</tr>
<tr>
<td>ExtOp</td>
<td>Zero Extended</td>
<td>Sign Extended</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Value</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUOp</td>
<td>00</td>
<td>ALU adds</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>ALU subtracts</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>ALU does function code</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>ALU does logical OR</td>
</tr>
<tr>
<td>ALUSelB</td>
<td>00</td>
<td>2nd ALU input = 4</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>2nd ALU input = Reg[rt]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2nd ALU input = extended, shift left 2</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>2nd ALU input = extended</td>
</tr>
</tbody>
</table>

2/25/04 ©UCB Spring 2004
Step 2 ⇒ Group together related signals
3&4) Microinstruction Format: unencoded vs. encoded fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Width</th>
<th>Control Signals Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wide</td>
<td>narrow</td>
</tr>
<tr>
<td>ALU Control</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>SRC1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SRC2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>ALU Destination</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Memory</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Memory Register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PCWrite Control</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Sequencing</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Total width</td>
<td>24</td>
<td>15</td>
</tr>
</tbody>
</table>
### Step 5 ⇒ Group into Fields, Order and Assign Names

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Values for Field</th>
<th>Function of Field with Specific Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALU</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td>ALU adds</td>
</tr>
<tr>
<td>Subt.</td>
<td></td>
<td>ALU subtracts</td>
</tr>
<tr>
<td>Func</td>
<td></td>
<td>ALU does function code</td>
</tr>
<tr>
<td>Or</td>
<td></td>
<td>ALU does logical OR</td>
</tr>
<tr>
<td><strong>SRC1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td>1st ALU input &lt;= PC</td>
</tr>
<tr>
<td>rs</td>
<td></td>
<td>1st ALU input &lt;= Reg[rs]</td>
</tr>
<tr>
<td><strong>SRC2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2nd ALU input &lt;= 4</td>
</tr>
<tr>
<td>Extend</td>
<td></td>
<td>2nd ALU input &lt;= sign ext. IR[15-0]</td>
</tr>
<tr>
<td>Extend0</td>
<td></td>
<td>2nd ALU input &lt;= zero ext. IR[15-0]</td>
</tr>
<tr>
<td>Extshft</td>
<td></td>
<td>2nd ALU input &lt;= sign ex., sl IR[15-0]</td>
</tr>
<tr>
<td>rt</td>
<td></td>
<td>2nd ALU input &lt;= Reg[rt]</td>
</tr>
<tr>
<td><strong>dest(ination)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rd ALU</td>
<td></td>
<td>Reg[rd] &lt;= ALUout</td>
</tr>
<tr>
<td>rt ALU</td>
<td></td>
<td>Reg[rt] &lt;= ALUout</td>
</tr>
<tr>
<td>rt Mem</td>
<td></td>
<td>Reg[rt] &lt;= Mem</td>
</tr>
<tr>
<td><strong>Mem(ory)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read PC</td>
<td></td>
<td>Read memory using PC</td>
</tr>
<tr>
<td>Read ALU</td>
<td></td>
<td>Read memory using ALUout for addr</td>
</tr>
<tr>
<td>Write ALU</td>
<td></td>
<td>Write memory using ALUout for addr</td>
</tr>
<tr>
<td><strong>Memreg</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR</td>
<td></td>
<td>IR &lt;= Mem</td>
</tr>
<tr>
<td><strong>PCwrite</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCwr</td>
<td></td>
<td>PC &lt;= PCSource</td>
</tr>
<tr>
<td>PCSrc</td>
<td></td>
<td>IF Zero then PCSource &lt;= ALUout else ALU</td>
</tr>
<tr>
<td>PCWrCond</td>
<td></td>
<td>IF Zero then PC &lt;= PCSource</td>
</tr>
<tr>
<td><strong>Seq(uencing)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq</td>
<td></td>
<td>Go to next sequential µinstruction</td>
</tr>
<tr>
<td>Fetch</td>
<td></td>
<td>Go to the first microinstruction</td>
</tr>
<tr>
<td>Dispatch</td>
<td></td>
<td>Dispatch using ROM.</td>
</tr>
</tbody>
</table>
Quick check: what do these fieldnames mean?

**Destination:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>RegWrite</th>
<th>MemToReg</th>
<th>RegDest</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>---</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>01</td>
<td>rd ALU</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>rt ALU</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>rt MEM</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**SRC2:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>ALUSelB</th>
<th>ExtOp</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>---</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>001</td>
<td>4</td>
<td>00</td>
<td>X</td>
</tr>
<tr>
<td>010</td>
<td>rt</td>
<td>01</td>
<td>X</td>
</tr>
<tr>
<td>011</td>
<td>ExtShft</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>Extend</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>Extend0</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
Specific Sequencer for our Microcoding

Sequencer-based control unit from last lecture

- Called “microPC” or “µPC” vs. state register

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>fetch</td>
<td>Next µaddress = 0</td>
</tr>
<tr>
<td>01</td>
<td>dispatch</td>
<td>Next µaddress = dispatch ROM</td>
</tr>
<tr>
<td>10</td>
<td>seq</td>
<td>Next µaddress = µaddress + 1</td>
</tr>
</tbody>
</table>

ROM:

- Opcode: Dispatch state
  - 000000: Rtype (0100)
  - 000100: BEQ (0010)
  - 001101: ORI (0110)
  - 100011: LW (1000)
  - 101011: SW (1011)
Recap: Finite State Machine (FSM) Spec

IR <= MEM[PC]
PC <= PC + 4

ALUout <= PC + SX

```
0000
```

```
0001
```

```
0100
```
```
ALUout <= A fun B
```
```
0110
```
```
ALUout <= A or ZX
```
```
1000
```
```
ALUout <= A + SX
```
```
1010
```
```
BEQ
```
```
If A = B then PC <= ALUout
```
```
0010
```

```
0101
```
```
R[rd] <= ALUout
```
```
MEM[ALUout] <= B
```
```
1100
```
```
R[rt] <= M
```
```
1010
```
```
MEM[ALUout] <= B
```
```
1100
```

```
0111
```
```
ORi
```
```
ALUout <= A + SX
```
```
1011
```

```
1100
```
```
SW
```
```
MEM[ALUout] <= B
```
```
1100
```

```
```
```
```
```
```
```
```
```
```
```
## Microprogram it yourself!

<table>
<thead>
<tr>
<th>Addr</th>
<th>ALU</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Dest.</th>
<th>Memory</th>
<th>Mem. Reg.</th>
<th>PC</th>
<th>Write</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Fetch:
0000: Add PC 4 Read PC IR ALU Seq

### Decode:
0001: Add PC Extshft Dispatch

### BEQ:
0010: Subt. rs rt ALUoutCond. Fetch

### Rtype:
0100: Func rs rt Seq
0101: rd ALU Fetch

### ORI:
0110: Or rs Extend0 Seq
0111: rt ALU Fetch

### LW:
1000: Add rs Extend Seq
1001: Add rs Extend Read ALU Seq
1010: Add rs Extend rt MEM Fetch

### SW:
1011: Add rs Extend Seq
1100: Add rs Extend Write ALU Seq

2/25/04 ©UCB Spring 2004
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
- ROM

"hardwired control"

"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 digram
° 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!