Implementing an ISA, part II - Control

David E. Culler
CS61CL
Nov 4, 2009
Lecture 10
Review: TinyMIPS

- **Reg-Reg instructions (op == 0)**
  - `addu`  
    \[ R[rd] := R[rs] + R[rt]; pc:=pc+4 \]
  - `subu`  
    \[ R[rd] := R[rs] - R[rt]; pc:=pc+4 \]

- **Reg-Immed (op != 0)**
  - `lw`  
    \[ R[rt] := Mem[ R[ rs ] + signEx(Im16) ] \]
  - `sw`  
    \[ Mem[ R[ rs ] + signEx(Im16) ] := R[rt] \]

- **Jumps**
  - `j`  
    \[ PC := PC_{31..28} || addr || 00 \]
  - `jr`  
    \[ PC := R[rs] \]

- **Branches**
  - `BEQ`  
    \[ PC := (R[rs] == R[rt]) ? PC + signEx(im16) : PC+4 \]
  - `BLTZ`  
    \[ PC := (R[rs] < 0) ? PC + signEx(im16) : PC+4 \]
Review: DataPath + Control

Asel | Bsel | Dsel | ld | pc | A | B | Ci
---|---|---|---|---|---|---|---
IR | RAM | A | D | A | D
+4 | pc2A | ld_pc | ld_ir | wr | m2D | ld_reg | sx_sel | comp

rs | rt | rd

npc_sel | id_pc | ld Jr | i2D | wr2D | rt_sel | ld_reg | b2D | s2A | s2D
Control State Machine (abstract)

I-Fetch
IR := Mem[pc]

- reset
- \neg reset \& OP == addu
- \neg reset \& OP == lw
- \neg reset \& (OP == beq) \& \neg (EQ)
- \neg reset \& (OP == bneq) \& \neg (N)

AddU
R[rd] := R[rs] + R[rt];
pc := pc + 4

SubU
R[rd] := R[rs] - R[rt];
pc := pc + 4

LW
R[rt] := mem[R[rs] + sx16];
pc := pc + 4

SW
mem[R[rs] + sx16] := R[rt];
pc := pc + 4

J
pc := pc_{31..28} || addr || 00

JR
pc := R[rs]

BR-taken
pc := pc + sx16 || 00

BR-not taken
pc := pc + 4
Ifetch: IR := mem[pc]

- RAM_addr <- A <- PC; (pc2A, ~s2A)
- IR_in <- D <- RAM_data; (~i2D,m2D,~b2D,~s2D)
- IR := IR_in; (1d_ir,~1d_pc,~1d_reg, ~wrt)
Control State Machine

I-Fetch
IR := Mem[pc]

pc2A, ~s2A, ~ir2D, m2D, ~b2D, ~s2D, ld _ir, ~ld_pc, ~ld_reg, ~wrt

AddU
R[rd] := R[rs] + R[rt];
pc := pc + 4

SubU
R[rd] := R[rs] - R[rt];
pc := pc + 4

LW
R[rt] := mem[R[rs] + sx16];
pc := pc + 4

SW
mem[R[rs] + sx16] := R[rt];
pc := pc + 4

J
pc := pc_{31..28} || [addr] || 00

JR
pc := R[rs]

BR-taken
pc := pc + sx16 || 00

BR-not taken
pc := pc + 4

reset & OP==addu
reset & OP==lw
reset & OP==lw
reset & (OP==beq & ~EQ)))
reset & (OP==breg & ~N))

- $npc\_sel=0, ld\_pc, ~pc2A, ~ld\_ir, ~i2D, ~wrt, ~m2D, ~rt\_sel, ld\_reg, ~b2D, ~sx\_sel, ~comp, ~s2A, s2D$
Control State Machine

I-Fetch
IR := Mem[pc]

pc2A, ~s2A, ~ir2D,
m2D, ~b2D, ~s2D, ld _ir, ~ld_pc,
~ld_reg, ~wrt

reset

AddU
R[rd] := R[rs] + R[rt]; pc := pc + 4
npc_sel = 0, ld_pc, ~pc2A, ~ld_ir, ~i2D, ~wrt, ~m2D,
~rt_sel, ld_reg, ~b2D, ~sx_sel, ~comp, ~s2A, s2D

reset & OP = addU

SubU
R[rd] := R[rs] - R[rt];
 pc := pc + 4

reset & OP = subU

LW
R[rt] := mem[R[rs] + sx16];
 pc := pc + 4

reset & OP = lw

SW
mem[R[rs] + sx16] := R[rt];
 pc := pc + 4

reset & OP = sw

J
pc := pc31..28 || addr || 00

reset & (OP = br & ~(EQ))

JR
pc := R[rs]

reset & (OP = br & ~(EQ))

BR-taken
pc := pc + sx16 || 00

reset & (OP = br & ~(EQ))

BR-not taken
pc := pc + 4

reset & (OP = br & ~(EQ))

- $npc\_sel = 0, ld\_pc, \neg pc\_2A, \neg ld\_ir, \neg i2D, \neg wrt, \neg m2D, \neg rt\_sel, ld\_reg, \neg b2D, \neg sx\_sel, comp, \neg s2A, s2D$

- $npc\_sel=0$, $ld\_pc$, $m2D$, $rt\_sel$, $ld\_reg$, $sx\_sel$, $s2A$
Exec SW: Mem[R[rs]+SXim16] := R[rt]
**Exec J:** $\text{PC} := \text{PC}_{31..28} \parallel \text{addr} \parallel 00$

- $\text{npc}_\text{sel}=1$, $\text{ld}_\text{pc}$, $\text{i2D}$
Exec JR: PC := R[rs]

- npc_sel=2, ld_pc, s2D, sx_sel=2
Exec Br Taken: \( PC := PC + SX16 \)

- \( \text{npc\_sel} = 3, \text{ld\_pc}, \text{i2D} \)
## Controller Specification

<table>
<thead>
<tr>
<th>Ifetch</th>
<th>reset</th>
<th>Exec</th>
<th>OP</th>
<th>EQ</th>
<th>N</th>
<th>NSTATE</th>
<th>npc_sel</th>
<th>ld_pc</th>
<th>pcA</th>
<th>ld_ir</th>
<th>i2D</th>
<th>wrt</th>
<th>m2D</th>
<th>rt_sel</th>
<th>ld_reg</th>
<th>b2D</th>
<th>SX_SEL</th>
<th>COMP</th>
<th>S2A</th>
<th>S2D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>E</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Exec

<table>
<thead>
<tr>
<th></th>
<th>reset</th>
<th>Exec</th>
<th>OP</th>
<th>EQ</th>
<th>N</th>
<th>NSTATE</th>
<th>npc_sel</th>
<th>ld_pc</th>
<th>pcA</th>
<th>ld_ir</th>
<th>i2D</th>
<th>wrt</th>
<th>m2D</th>
<th>rt_sel</th>
<th>ld_reg</th>
<th>b2D</th>
<th>SX_SEL</th>
<th>COMP</th>
<th>S2A</th>
<th>S2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>addu</td>
<td>0</td>
<td>1</td>
<td>addu</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>subu</td>
<td>0</td>
<td>1</td>
<td>subu</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>lw</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>0</td>
<td>1</td>
<td>sw</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>j</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>jr</td>
<td>0</td>
<td>1</td>
<td>jr</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BEQ-taken</td>
<td>0</td>
<td>1</td>
<td>BEQ</td>
<td>0</td>
<td>x</td>
<td>I</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BEQ-notake</td>
<td>0</td>
<td>1</td>
<td>BEQ</td>
<td>1</td>
<td>x</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BNEG-taken</td>
<td>0</td>
<td>1</td>
<td>BNEG</td>
<td>x</td>
<td>0</td>
<td>I</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BNEG-notake</td>
<td>0</td>
<td>1</td>
<td>BNEG</td>
<td>x</td>
<td>1</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Adminstration

• HW7 due midnight

• Mid Term 2 Monday 11/9
  – 5:30 – 7:30 RM: 145 Dwinelle
  – alternate Friday 11/4 3:00-5:00 rm TBA
  – Review session Thurs

• Project 3
  – incremental lab check offs

• Flex lab mon (9-5) and tues (9-1)
  – midterm final prep
  – project 3 help
Controller Implementation

clk
reset

exec

op
eq
n

npc_sel

s2D
Combinational Logic per Ctrl Point

- exec → pc2A
- exec → IdPC
- exec
- op → wrt
Multiplexor Control

I 0 1 0 0 0 0 0 1 0 0 0 1
I 0 1 0 0 0 0 0 1 0 0 1 0 1
I 3 1 0 0 1 0 0 x 0 0 x 0 0

op
Faster Clock

- Clock Period > Longest path from reg out to input + reg delay
Multi-Cycle Controller State Machine

I-Fetch
IR := Mem[pc]

Op / Dcd
A := R[rs], B := R[rt]

AddU
S := A + B;
pc := pc + 4

SubU
S := A - B;
pc := pc + 4

LW
S := A + sx16;
pc := pc + 4

SW
S := A + sx16;
pc := pc + 4

J
pc := pc31..28||addr||00

JR
pc := R[rs]

BR-taken
pc := pc + sx16||00

BR-not taken
pc := pc + 4

R[rd] := S;

read MAR := S;

wrt MAR := S; MDR := B'
Time State control

- Move control word through the stages
- Decode per stage
- Active stage moves “around the ring”
More regular multi-cycle execution

I-Fetch
IR := Mem[pc]

Op / Dcd
A := R[rs], B := R[rt]

AddU
S := A + B;
pc := pc + 4
S' := S;
R[rd] := S;

SubU
S := A - B;
pc := pc + 4
S' := S;
R[rd] := S';

LW
S := A + sx16;
pc := pc + 4
read
MAR := S;
R[rd] := D

SW
S := A + sx16;
pc := pc + 4
wrt
MAR := S;
MDR := B'

J
pc := pc31..28 || addr || 00

JR
pc := R[rs]

BR-taken
pc := pc + sx16 || 00

BR-not taken
pc := pc + 4
Sequence of Multi-step Operations

- Operation implemented as sequence of steps on distinct resources
  - wash => dry => fold
- Multiple independent Operations
Technology Trends

- Clock Rate: ~30% per year
- Transistor Density: ~35%
- Chip Area: ~15%
- Transistors per chip: ~55%
- Total Performance Capability: ~100%

by the time you graduate...
- 3x clock rate (>10 GHz)
- 10x transistor count (100 Billion transistors)
- 30x raw capability

- plus 16x dram density,
- 32x disk density (60% per year)
- Network bandwidth, ...
Pipelining

- Overlap consecutive operations
Definition: Performance

• Performance is in units of things per sec
  – bigger is better

• If we are primarily concerned with response time

\[
\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}
\]

"X is n times faster than Y" means

\[
n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(Y)}
\]
Pipeline Performance

• N operations performed in k steps each
• Sequential Time: N*k
• Lower bound: N (1 every cycle)
• Pipeline Time: k – 1 + N

• Bound on Speedup on k-stage pipeline < k

• Speedup(k,N) = Time(1,N)/Time(k,N)
  = N*k / (N+k-1)
  ≈ N / (1+k/N)

• StartUp Cost: k-1
• Peak Rate
• Half Power point
Performance Trends

- Supercomputers
- Mainframes
- Minicomputers
- Microprocessors

MIPS R3000
Processor Performance
(1.35X before, 1.55X now)
Pipelined control
Pipelined Instruction Execution

• Fetch Instruction Every cycle
• Launch into a pipeline

• What if they are not independent?
  – structural hazards
    » two operations need to use same resource
  – data dependence
    » later instruction needs to use the value produce by an earlier on

• Detect
• Wait till hazard clears
Pipelined “Bubble”