Computer Organization:
A real processor

MIPS 2000
Background

- We have built a model processor in ActiveHDL
  - You get to make it work
- Heavily based on MIPS2000
  - Described by Patterson & Hennessy
- Single-cycle design
  - All operations take 1 (long) cycle
Instruction Set Specs

- 32 registers
- Load-Store Architecture
- Word Addressing
- 3 Formats for Instructions
  - Register to Register
  - Immediate
  - Jump
Instruction Encodings

- Three principal types (32 bits in each instruction)

<table>
<thead>
<tr>
<th>type</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shift</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(egister)</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>I(mmediate)</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16- immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J(ump)</td>
<td>6</td>
<td></td>
<td></td>
<td>26- immediate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Some of the instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>offset</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>6'h00</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>6'h20</td>
</tr>
<tr>
<td>sub</td>
<td>6'h00</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>6'h22</td>
</tr>
<tr>
<td>and</td>
<td>6'h00</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>6'h24</td>
</tr>
<tr>
<td>or</td>
<td>6'h00</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>6'h25</td>
</tr>
<tr>
<td>slt</td>
<td>6'h00</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>6'h2a</td>
</tr>
<tr>
<td>lw</td>
<td>6'h23</td>
<td>rs</td>
<td>rt</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>6'h2b</td>
<td>rs</td>
<td>rt</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td>6'h04</td>
<td>rs</td>
<td>rt</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>6'h08</td>
<td>rs</td>
<td>rt</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>6'h02</td>
<td>target address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halt</td>
<td>6'h3f</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mips2000
Program Counter

assign offset = {{16{Inst[15]}},Inst}; // sign extend the immediate
assign Branch = Next + offset;
assign Jump = {Next[31:26],Inst[25:0]}; // used by J instruction

// There are 4 possible sources for PC
// 0. PC = Next (Move to next Instruction)
// 1. PC = Next + offset (Conditional Branch)
// 2. PC = Reg (Jump to Register value)
// 3. PC = Next[31:26],jump_target( J instruction)

assign PC = (PCSel[1])?
    ((PCSel[0])? Jump : Reg) :
    ((PCSel[0])? Branch : Next);
Controller

Skeleton Code:

... ADDI: begin
    wrDataSel = 2'bxx;
    mw = 1'bx;
    mr = 1'bx;
    PCSel = 2'bxx;
    srcB = 1'bx;
    regWrite = 1'bx;
    wrRegSel = 2'bxx;
    op = 6'bxxxxxxxx;
end
Register File

// decide which register is the one that might be written to (depends on instruction)
// 00 – rd, 01 – rt, 1X – hardwired to 31 for JAL
assign wrReg = wrRegSel[1] ?
    5'b11111 : (wrRegSel[0] ? rd : rt);

// do two reads and, optionally, one write with the register file
// read two registers and send them to the ALU
assign RegA = RegFile[rs];
assign RegB = RegFile[rt];

// write into a register (but not the register storing our constant 0)
always @(posedge clk) begin
    if (regWrite && (wrReg != 0)) begin
        RegFile[wrReg] = WriteData;
    end
end
// use srcB to select between RegB and Imm
assign B = (srcB === 0)? RegB : Imm;

always @(A or B or op) begin
  case (op)
    6'b000001: result = A + B;
    ...
    default:   result = 32'hxxxxxxxx;
  endcase
  zero = (result == 32'h00000000);
  neg  = result[31];
end
Data Memory

- Address from ALU
- Data from Reg B

- Memory-Mapped I/O
  - SW to xFFFFFFF
  - Buffers / Displays

(See dmemory.v for more)
Miscellaneous

WrRegSel:
   // wrDataSel
   // 00: Out = ALU
   // 01: Out = MEM
   // 1X: Out = PC + 1

SignExtender:
   convert 16-bit to 32-bit
## R-Format Operations

<table>
<thead>
<tr>
<th>Func</th>
<th>Operation</th>
<th>PC</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>$rd = rs + rt$</td>
<td>PC++</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>$rd = rs - rt$</td>
<td>PC++</td>
<td></td>
</tr>
<tr>
<td>SLT</td>
<td>$rd = (rs &lt; rt) ? 1 : 0$</td>
<td>PC++</td>
<td>Set on less than</td>
</tr>
<tr>
<td>JR</td>
<td>No change</td>
<td>PC=rs</td>
<td>Jump to Register</td>
</tr>
</tbody>
</table>
## I-Format Ops:

<table>
<thead>
<tr>
<th>Operation</th>
<th>PC</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI</td>
<td>rt = rs+SE(imm)</td>
<td>PC++</td>
</tr>
<tr>
<td>ORI</td>
<td>rt = rs</td>
<td>imm</td>
</tr>
<tr>
<td>LUI</td>
<td>rt = imm &lt;&lt;&lt; 16</td>
<td>PC++</td>
</tr>
<tr>
<td>LW</td>
<td>rt = MEM[rs+se(imm)]</td>
<td>PC++</td>
</tr>
<tr>
<td>SW</td>
<td>MEM[rs+se(imm)] = rt</td>
<td>PC++</td>
</tr>
<tr>
<td>BEQ</td>
<td>(rs == rt)?</td>
<td>PC+1+(0</td>
</tr>
</tbody>
</table>
### J-Format Ops:

<table>
<thead>
<tr>
<th>Func</th>
<th>Operation</th>
<th>PC</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>PC = target</td>
<td>a.k.a GOTO</td>
<td></td>
</tr>
<tr>
<td>JAL</td>
<td>r31 = PC+1</td>
<td>PC = target</td>
<td>Jump and Link</td>
</tr>
</tbody>
</table>

JAL stores next address, jumps to target (a.k.a fn call)
Final Tips:

- Verilog uses “?” for Don’t Cares
- Waveforms will make things easier
- Be sure to set clk and reset
- Simulation: 14000ns limit-- longer, controller is broken
- Don't use opcode defs from 378 text-- ours are different.
Programming Example

Given: A is an array of size B

Goal: Compute

\[ \sum_{i=0}^{B} A[i] \]

Lets use a for loop …
In Assembly Language:

- Variables $\to$ Registers
- Array Access $\to$ Load (name+offset)
- Minimal Control Structures
  - Branches (A < B, A >= B, A != B)
  - Jumps
C to ASM

High Level Language

C = 0;
for(i = 0; i < B; i = i+1) {
    C = C + A[i];
}

Psuedo-Asm

C = 0;
i = 0;
Loop: bge i, B, Exit
temp = A+i
temp2 = load 0(temp)
C = C + temp2;
i = i + 1;
} Loop
Exit: …
\begin{align*}
\text{C} &= 0; \\
i &= 0; \\
\text{Loop: } &\text{bge i, B, Exit} \\
\text{temp} &= \text{load A[i];} \\
\text{C} &= \text{C + temp;} \\
i &= i + 1; \\
\text{j Loop} \\
\text{Exit: } &\ldots
\end{align*}
\begin{align*}
\text{r3} &= \text{r0, PC++} \\
\text{r4} &= \text{r0, PC++} \\
\text{Loop: } &\text{PC =} \\
\quad &\left( \text{r4} \geq \text{r2} \right) ? \text{Exit : PC+1} \\
\text{r6} &= \text{MEM[r4+r1 ],PC++} \\
\text{r3} &= \text{r3 + r6, PC++} \\
\text{r4} &= \text{r4 + 1, PC++} \\
\text{PC} &= \text{LOOP} \\
\text{Exit: }
\end{align*}