Lecture #11: Final Project

Kunle Olukotun
EE183
February 19, 2003

Lab Issues

- Lab 3 Questions?
  - Memory mapped I/O
  - EXT1
  - Due on Friday @ 6pm
- ASM183
- Modelsim
  - Verilog simulation without synthesis
  - Tutorial and installation instructions EE183 webpage
Final Lab

- Start thinking about final lab now!
- Need 1 paragraph description by tonight
- Email to David and I
- Project meetings/feedback by Feb 24

<table>
<thead>
<tr>
<th>Lab</th>
<th>Pre-Lab Report Due</th>
<th>Demo Due by 5 pm</th>
<th>Final Report Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan 17</td>
<td>Jan 24</td>
<td>Jan 27</td>
</tr>
<tr>
<td>2</td>
<td>Jan 31</td>
<td>Feb 7</td>
<td>Feb 10</td>
</tr>
<tr>
<td>3</td>
<td>Feb 14</td>
<td>Feb 21</td>
<td>Feb 24</td>
</tr>
<tr>
<td>4</td>
<td>Feb 28</td>
<td>Mar 14</td>
<td>Mar 17</td>
</tr>
</tbody>
</table>

- Bagle Day March 14

Processor Based SOC

- Processors controls hardwired functions
- Memory level communication
  - Memory mapped I/O
  - VGA example
- More software development
- Added flexibility
  - Decreased design time
  - Adapt to changing standards
- Embedded processors
  - MIPS, ARM, PowerPC
Application Specific Instruction-set Processor (ASIP)

- Specialized datapath (FUs)
- Register level communication
- ISA changes
- Advantages
  - Raises design abstraction: C instead of Verilog
  - Eliminates most control logic
  - Added flexibility
- Configurable processors
  - Tensilica
  - ARC
- Final project

Project Examples

- Video game
  - Gamepad and VGA memory mapped I/O
- Keyboard to video character generator
- Fractals with a processor
  - Add multiply unit
- Simple audio processor
  - Interface to audio on XStend board
  - Display on VGA
- Voice disguiser
  - Multiply unit, audio, FIR filter
- DES encryption
- Graphics (line/circle/square/triangle) hardware
- Extra credit for creative applications
Final Project

Part I

Add procedure call and return to your processor
Use register 0 as the link register
Change ASM183 to accommodate your new instructions

Part II

Design an SOC with memory mapped I/O devices or functional units
Design an ASIP by adding a specialized instructions to your 12-bit processor for any application you wish

Control Transfer Instructions

Conditional and unconditional jumps with absolute addresses

Instruction format

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OP</td>
<td>COND</td>
<td>JUMP ADDRESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instructions

<table>
<thead>
<tr>
<th>OPbin</th>
<th>operation</th>
<th>mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Jump False</td>
<td>JF. cond JPC</td>
</tr>
<tr>
<td>01</td>
<td>Jump True</td>
<td>JT. cond JPC</td>
</tr>
<tr>
<td>10</td>
<td>Uncond. Jump</td>
<td>J</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDbin</th>
<th>condition</th>
<th>mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>ALU result negative</td>
<td>.NEG</td>
</tr>
<tr>
<td>0101</td>
<td>ALU result zero</td>
<td>.ZERO</td>
</tr>
<tr>
<td>0110</td>
<td>ALU carry</td>
<td>.CARRY</td>
</tr>
<tr>
<td>0111</td>
<td>ALU result negative or zero</td>
<td>.NEGE0</td>
</tr>
<tr>
<td>0000</td>
<td>TRUE</td>
<td>.TRUE</td>
</tr>
<tr>
<td>1000</td>
<td>External Condition</td>
<td>.EXT</td>
</tr>
</tbody>
</table>

• Extra points for making conditional branch PC relative
• Requires assembler changes

• Could add up to 8 external conditions

Condition codes are only set by ALU instructions
Incremental Digital Integrator

Y Register (n bits)

\[ \text{Y Register (n bits)} \]

Adder

\[ \text{Adder} \]

R Register (n bits)

\[ \text{R Register (n bits)} \]

\[ D \]

\[ Z \]

\[ \text{Incremental Digital Integrator} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]

\[ \text{Y Register (n bits)} \]

\[ \text{R Register (n bits)} \]

\[ \text{Adder} \]
Define two new instructions

**INTEG**
RC, RA, RB  
RC = RA + RB and overflow left in a special register Z

**ADDZ**  
RA, RA  
RA = RA +1 if Z > 0  
RA = RA -1 if Z < 0  
RA = RA if Z = 0

New ASM183 Code for Incremental Digital Integrator

```
// R1 = Y  
// R2 = R  
ADDDZ R1, R1  
INTEG R2, R1, R2
```