CSE 141L
Computer Architecture Lab
Fall 2005

Lecture 4
Pramod V. Argade
November 15th, 2005

---

Fall 2005 CSE 141L Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Lab Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9/27</td>
<td>Tuesday</td>
<td>No Class</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10/4</td>
<td>Tuesday</td>
<td>Lab1: 8-bit Processor ISA</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10/11</td>
<td>Tuesday</td>
<td>Lab1 Discussion</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10/18</td>
<td>Tuesday</td>
<td>Lab2: Assembler, ISS</td>
<td>Lab1</td>
</tr>
<tr>
<td>5</td>
<td>10/25</td>
<td>Tuesday</td>
<td>LogicWorks 5 Review</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11/1</td>
<td>Tuesday</td>
<td>Lab 3: Data Path</td>
<td>Lab2</td>
</tr>
<tr>
<td>7</td>
<td>11/8</td>
<td>Tuesday</td>
<td>Lab 3 Discussion</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>11/15</td>
<td>Tuesday</td>
<td>Lab 4: Full CPU</td>
<td>Lab3</td>
</tr>
<tr>
<td>9</td>
<td>11/22</td>
<td>Tuesday</td>
<td>Lab 4 Discussion</td>
<td></td>
</tr>
<tr>
<td>11/29 - 12/2</td>
<td></td>
<td></td>
<td>Lab 4 Demo by Students</td>
<td>Lab4</td>
</tr>
</tbody>
</table>
Lab Due Dates

- Lab 4 Due:
  - Before 6:00 PM Tuesday, November 29th
  - You must make an appointment to demonstrate your between CPU November 29th through December 2nd
- You cannot make any changes to the design between the time you submit to your demo to TA
- No late submissions!

Microprocessor Design Steps

- Design Instruction Set Architecture (ISA)
- Develop software generation tools
- Code applications
- Develop instruction set simulator (ISS)
- Design datapath, verify it
- Design the Processor, simulate logic
- Verify the processor
- (Fabricate the chip: not in this class!)
**Lab 4 Assignment**

- Design logic for complete 8-bit processor architecture
- Use LogicWorks 5 to design logic
- Simulate the operation executing 3 programs from Lab 1

**Processor Organization**

Note: Blocks in your architecture equivalent to those in blue are to be implemented in Lab4
What you must include

- Reset logic
- Program Counter Logic
  - Non-control transfer instructions
  - Control transfer instructions
  - Halt instruction
- Data path modified for Load/Store instructions
- Control logic
- Instruction counter
  - Initialize on reset
  - Freeze on HALT instruction

What you will turn in for this Lab

- Summary of your ISA from Lab 1 and assembly code with machine code for 3 programs.
- Printed schematics for the top level CPU in LogicWorks 5.
- All the LogicWorks 5 files you created (to be submitted via electronic submission).
- Answers to following questions.
Questions

• What changes did you make in your original ISA and why?
• What is instruction count for each one of the three programs? How do the numbers compare with those for the ISS. If the numbers are different, why?
• What are the strengths of your design?
• What are the deficiencies of your design?
• Which instruction will determine the clock frequency of your processor, i.e. is responsible for the critical path?
• Which instruction is most expensive in terms of the number of gates required?

Question Continued

• Having gone through a complete CPU design experience, what would you do differently in your ISA to:
  – Decrease static and dynamic instruction count
  – Simplify data path design
  – Simplify CPU design
• If you were to pipeline the execution, what would the pipeline stages be? Give at least three issues that will complicate the design of your processor.
Lab 4 Grading

- It is your responsibility to make an appointment with one of the TAs before 11/29/05
- Show TA that your CPU design works before end of Friday, December 2\textsuperscript{nd}.
- You should test the programs using the data patterns given in Lab1.
- The TAs may test your design for correct functionality using their own data files that satisfy the constraints outlined in Lab 1.

Useful Hints

- Build hierarchical design
- Test thoroughly at every level of hierarchy
  - Connect binary switches and hex keypads to provide inputs
  - Connect binary and hex displays to observe behavior
- Write an assembly program to test individual instructions in your CPU
  - Self-checking programs are ideal!
What is Functional Verification?

- Making sure that your design is functionally correct!
  - Reset behavior
  - Instructions
    - Addressing modes
    - Algorithms in hardware, e.g. setting carry
    - Corner cases
    - Control transfer: branch/jump
  - Memory access
  - Special features
    - e.g. HALT in our case

Importance of Verification

- General purpose processor must run, without a flaw, any application running on it!
- Programmers will use the CPU in ways you never imagined!
- Processor may be used in mission critical applications
- It is costly to fix bugs in processors
  - Chip mask and fabrication costs
  - System HW and SW redesign
  - Lost market opportunity
A Program to Test 8-bit CPU

```
0x0000 start: mov r3, $0xf    // r3 = 0xf
0x0001 add r1, r3      // r1 = 0xf0
0x0002 add r3, $3      // r3 = 0x12
0x0003 mov r2, r3      // r2 = 0x12
0x0004 ushr4 r3, r2      // r3 = 1
0x0005 st r2, *r3     // M[1] = 0x12
0x0006 add r1, r3      // r1 = 0xf1, test corruption of RAM
0x0007 mov r2, $0      // r2 = 0
0x0008 sub r2, $1      // r2 = 0xff
0x0009 xor r2, r1      // r2 = 0xe
0x000a ld r1, *r3     // r1 = 0x12
0x000b ushr4 r2, r1      // r2 = 1
0x000c cmpeq r2, $2      // flag = 0
0x000d mov r0, 0xf     // r0 = 0xf
0x000e add r0, $2      // r0 = 0x11, addr of cont1
0x000f jmpf *r0          // should jump
0x0010 add r2, $1      // should not exec. r2 = 1
0x0011 cmpeq r2, $1      // flag = 1
0x0012 add r0, $3      // r0 = 0x19
0x0013 add r0, $2      // r0 = 0x1b
0x0014 jmpf cont3        // should not jump to cont3
0x0015 halt                // should not halt
0x0016 jmpf cont2        // should not jump to cont2
0x0017 cmphi r1, r2       // flag = 0
0x0018 add r0, $3       // r0 = 0x1e
0x0019 add r0, $3       // r0 = 0x21
0x001a jmpf cont5        // should not jump to cont5
0x001b add r0, $1       // r0 = 0x22, address of cont5
0x001c jmp *r0          // jump to cont5
0x001d halt                // should not halt
0x001e jmpf cont4        // should not jump to cont4
0x001f cmphi r1, r2       // flag = 1
0x0020 add r0, $3       // r0 = 0x21
0x0021 add r0, $3       // r0 = 0x21
0x0022 jmpf cont3        // should not jump to cont3
0x0023 add r0, $1       // r0 = 0x22
0x0024 sub r2, $1       // r2 = 0
0x0025 sub r3, $1       // r3 = 0
0x0026 st r2, *r3      // M[0] = 0
0x0027 halt                // should not halt
0x0028 halt                // should not halt
```

A Program to Test 8-bit CPU
Before you leave

• Remember to make appointment with the TAs to show your operational design