Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hours: Tue. 7:30 - 8:30 PM (Center 109)
Wed. 4:30 - 5:30 PM (AP&M 4141)

TA: Hua (Kevin) Yang (hu@cs.ucsd.edu)
Office Hour: Mon/Tue 3 - 4:30 PM

Tutors:
Anthony Choi (puchoi@cs.ucsd.edu)
Office Hour: Tue/Wed 1:30 - 3 PM

Diann Shaw (ddshaw@cs.ucsd.edu)
Office Hour: Tue 4 - 5:30 PM, Thu 4:30 - 6

Hou Choi (hochoi@cs.ucsd.edu)
Office Hour: Mon 1:30 - 3 PM, Fri. 3 - 4:30

Lecture: Tue. 6 - 7:15 PM, Center 109

Reference Book: LogicWorks4, Capilano Computing Systems
Available in Bookstore

Web-page: http://www.cse.ucsd.edu/classes/su04/cse141L
<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Time</th>
<th>Room</th>
<th>Topic</th>
<th>Assignment Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tue. 6/29</td>
<td>6 - 7:15 PM</td>
<td>Center 109</td>
<td>Assignment 1: ISA</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Tue. 7/6</td>
<td>6 - 7:15 PM</td>
<td>Center 109</td>
<td>Assignment 2: ISS</td>
<td>#1</td>
</tr>
<tr>
<td>3</td>
<td>Tue. 7/13</td>
<td>6 - 7:15 PM</td>
<td>Center 109</td>
<td>Assignment 3: Datapath</td>
<td>#2</td>
</tr>
<tr>
<td>4</td>
<td>Tue. 7/20</td>
<td>6 - 7:15 PM</td>
<td>Center 109</td>
<td>Assignment 4: CPU</td>
<td>#3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No Lecture</td>
<td></td>
</tr>
</tbody>
</table>

July 26 - 29: Students demonstrate their working CPU Design to TAs by appointment. Assignment #4 reports are due during the demonstration.
Lab Due Dates

• Lab 2 Due:
  ⇒ Before 6:00 PM Tuesday, July 13
• No late submissions!

• I plan to disclose a range of clock cycles from your Lab 2 reports (no names)
Microprocessor Design Steps

✓ Design Instruction Set Architecture (ISA)
✓ Code applications
  • Develop software generation tools
  • Develop instruction set simulator (ISS)
  • Design datapath, verify it
  • Design the Processor, simulate logic
  • Verify the processor
  • Fabricate the chip
Develop Assembler for your ISS

• Assembler converts assembly file to hex code

• Language choice for assembler:
  – C, C++, Java or Perl

• Command line:
  assembler mean.s

• Output generated:
  – mean.dis (annotated with hex code)
  – mean.imi (memory image in hex)
Assembler Code

- Process command line options
- Open input file and output files
- Parse each line
  - Gather fields: label, instruction mnemonic, …
  - Detect errors and report them
  - Assemble instruction
  - Generate output to .dis
  - Generate output to .imi
- Close files
Assembler Input and Output

• Input

    // mean.s
    // r0 used to index array
    ...
    000 start: mov r3, $0    // r3 = 0, has address of a[0]
    001    ld r0, *r3    // r0 points to a[0]
    002    mov r2, $0    // r2 = sum_low = 0

• Output

    // mean.dis
    // r0 used to index array
    ...
    0x0c 000 start: mov r3, $0    // r3 = 0, has address of a[0]
    0xe3 001    ld r0, *r3    // r0 points to a[0]
    0x08 002    mov r2, $0    // r2 = sum_low = 0
Are your 3 programs functionally correct?
General Architecture Challenges

• ISA
  – What is the performance?
    • How to find it out for huge programs?
  – How to optimize the ISA?
    • How to test “what if” scenarios?

• Design
  – How to verify the design?
  – What is the golden reference for behavior?
Answer:
Instruction Set Simulator (ISS)
What is an ISS?

- Software model of your processor
- Runs on a PC or workstation
- Is aware of:
  - Processor’s internal resources
  - ISA
  - Size and characteristics of I-Mem & D-Mem
- Provides
  - Run-time statistics
  - Debug capabilities
What does an ISS do?

- **Input:**
  - Instruction and data memory image

- **Output:**
  - Instruction trace
  - Run-time statistics
  - Number of instructions executed
  - Utilization of resources and instructions

...
What is a Reset?

• What happens on PC when you hit
  – Power button (cold boot)
    • Does extensive Power On Self Test (POST)
  – Reset button (warm boot)

• Reset is a special pin on a processor

• Processor and hardware is initialized to a known state
  – Program Counter (PC) set to a known value
  – Internal registers set to a known value
  – Instruction execution starts
Structure of ISS

- Written in C++ or Java (C is ok too)
- Define a class (say, Proc)
  - Members are resources of the processor
    - PC, register file, I-Mem, D-Mem, …
Methods in ISS

• loadIMem(): loads instruction memory image
  – Reads hex machine code from a text file
• loadDMem(): loads data memory image
  – Reads hex data from a text file
• reset(): reset the processor
  – Initialize internal state of the processor
  – Set PC = 0 and start execution
Methods in ISS: Runloop()

- Executes instructions
- May take an argument
  - Number of instructions to execute
  - Enables single stepping
- Generates instruction trace
  - Disassembles instruction
  - Prints results and other side effects
Components of runloop()

- **Fetch:**
  - Read instruction from I-Mem at location pointed to by PC

- **Decode:**
  - Analyze instruction
  - Find opcode and other fields, such as, register(s), immediate data, branch offset, etc.
  - Note destination for the result
  - Read operand(s) from register(s) (if any)
Components of runloop()

• Execute:
  – Fetch operand(s) from register(s)
  – Do ALU operation
  – Read D-Mem
  – Select result: either ALU output or data read from D-Mem
  – Resolve whether a branch is taken or not
Components of runloop()

- Memory/Register Writeback
  - Write result to register/D-Mem
- Set next PC
  - If instruction is not a control transfer instruction, or branch is not taken
    - PC = PC +1
  - Else
    - PC = target of the branch
Components of runloop()

- Generate instruction trace
  - Disassemble instruction
  - Print instruction side effects
- Stop execution if HALT instruction
- Else continue with the next instruction
Example Instruction Trace

0x5d 009: ADD   R3, $0x1  { Result = 0x 3 }  
0xfb 010: ST    R2, *R3   { Addr = 0x3, Result = 0x15 }  
0x18 011: MOV   R2, $0x4  
0x65 012: SUB   R1, $0x  { Result = 0x 6 }  
0x74 013: CMPEQ R1, $0x0  { Flag = 0 }  
0x33 014: MOV   R0, $0xf  
0x53 015: ADD   R0, $0x3  { Result = 0x12 }  
0x51 016: ADD   R0, $0x1  { Result = 0x13 }  
0x40 017: JMPF  *R0  
0xf6 019: ST    R1, *R2   { Addr = 0x4, Result = 0x6 }  
0x19 020: MOV   R2, $0x5
Utility Methods in ISS

• dumpIMem():
  – Dumps contents of instruction memory
  – Optionally takes start and end address

• dumpDMem()
  – Dumps contents of data memory
  – Optionally takes start and end address

• Both useful for debugging the program
Main()

• Instantiate class Proc
• Load I-Mem, D-Mem
• Reset the processor
• Interactive loop
  – Receive user command
  – Execute runloop()
  – Dump memory
Extra Credit

• The default mode for iss is non-interactive
• You may implement user-interactive mode
  – May have primitive functionality of gdb
  – e.g.
    • Load .imi, .dmi files
    • Step one or more instructions
    • Set break point
    • Inspect/set a memory location
What you will turn in for Lab 2

- Summary of your ISA from Lab 1 as well as assembly listing for the three programs, including the machine code for each instruction.
- Complete code for the Assembler and ISS, including all the header files and makefile.
- Complete instruction trace for the three programs executing correctly on the ISS.
- Data memory dump before and after running each one of the three programs. Also, dynamic instruction count for each one of the three programs.
- Answer to the questions below.
- Use “turnin” script (instructions are on the web)
Tips for debugging programs

• Typical problems
  – Machine code translation errors
  – ISS bugs
  – Logic errors in your program code
  – Infinite loops
  – Branch to invalid I-Mem location
  – Data fetch from invalid D-Mem location
  – Branch target may change if you modify code

• Print instruction side effects in trace!
You should have completely functional three programs at the end of Lab2!
Homework: Review LogicWorks 4

• Head start on Lab 3 where:
  – You will design datapath
    • Register file, ALU and other elements
    • Implement and test logic

• No submission!
LogicWorks 4 Review

• Various components available in Standard Libraries.
  – D Flip-flop, logic gates, multiplexors, registers, adders, clock, binary switch, binary display, hex keypad, hex display, etc.

• How to connect a bus to various components.

• How to define a subcircuit bottom up
  – Create a subcircuit
  – Use it to define the pins on the parent symbol.

• How to simulate a circuit and generate waveforms.
What you need for Lab2

- UNIX workstations
  - Accounts
- Tools
  - GCC
  - GDB or DDD debugger
Lab Due Dates

• Lab 2 Due:
  ⇒ Before 6:00 PM Friday, July 13th

• No late submissions!

• I plan to disclose a range of clock cycles from your Lab 2 reports (no names)