

CSE 141L

Computer Architecture Lab

Spring 2005

Lecture 4
Pramod V. Argade
May 17th, 2005

CSE141: Computer Architecture Lab

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hour:
Mon. 5:00 - 6:00 PM (AP&M 5218)

TAs:
Anjum Gupta: a3gupta@cs.ucsd.edu
Baris Arslan: barslan@cs.ucsd.edu
Kevin Hu: huhu@cs.ucsd.edu
Raid Ayoub: rayoub@cs.ucsd.edu

Textbook: LogicWorks 5, Interactive Circuit Design Software
Capilano Computing Systems, 2004
(Available in UCSD Bookstore.)

Web-page: <http://www.cse.ucsd.edu/classes/sp05/cse141L>

Spring 2005 CSE 141L Course Schedule

Lecture #	Date	Day	Lecture Topic	Lab Due
1	3/29	Tuesday	No Class	
2	4/5	Tuesday	Lab1: 8-bit Processo ISA	-
3	4/12	Tuesday	Lab1 Discussion	-
4	4/19	Tuesday	Lab2: Assembler, ISS	Lab1
5	4/26	Tuesday	LogicWorks 5 Review	-
6	5/3	Tuesday	Lab 3: Data Path	Lab2
7	5/10	Tuesday	Lab 3 Discussion	-
8	5/17	Tuesday	Lab 4: Full CPU	Lab3
9	5/24	Tuesday	Lab4 Discussion	-
6/1 - 6/3			Lab 4 Demo by Students	Lab4

CSE141-L-3

Pramod Argade, Spring '05

Lab Due Dates

- Lab 4 Due:
 - ⇒ Before 6:30 PM Tuesday, May 31st
 - ⇒ You must make an appointment to demonstrate your CPU June 1 - June 3
- You cannot make any changes to the design between the time you submit to your demo to TA
- No late submissions!

CSE141-L-4

Pramod Argade, Spring '05

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!)

CSE141-L-5

Pramod Argade, Spring '05

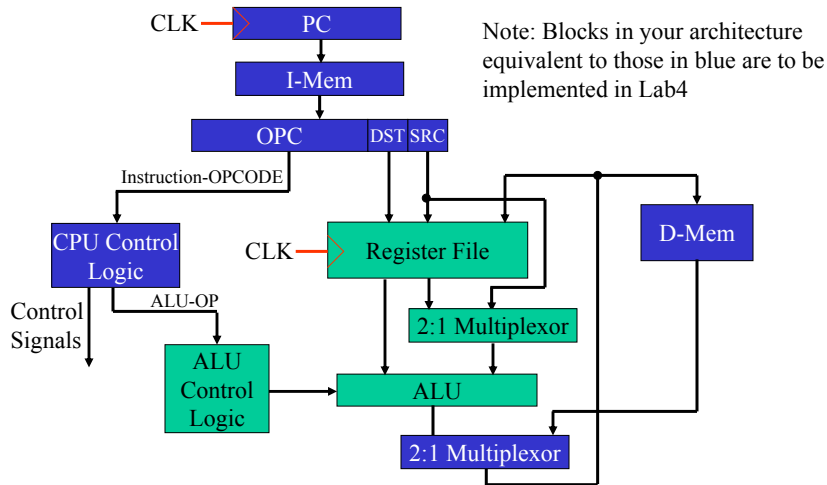
Lab 4 Assignment

- Design logic for complete 8-bit processor architecture
- Use LogicWorks 4 to design logic
- Simulate the operation executing 3 programs from Lab 1

CSE141-L-6

Pramod Argade, Spring '05

Processor Organization



CSE141-L-7

Pramod Argade, Spring '05

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

CSE141-L-8

Pramod Argade, Spring '05

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 as well as all the lower level modules you designed in LogicWorks 4.
- All the LogicWorks 4 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 is most responsible for the length of 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.

CSE141-L-11

Pramod Argade, Spring '05

Lab 4 Grading



- It is your responsibility to make an appointment with one of the TAs before 5/31/05
- Show TA that your CPU design works before end of Friday, June 3rd.
- 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.

CSE141-L-12

Pramod Argade, Spring '05

Useful Hints



- Avoid propagation of unknowns
 - Initialize on reset
- 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!

CSE141-L-13

Pramod Argade, Spring '05

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

CSE141-L-14

Pramod Argade, Spring '05

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 application
- It is costly to fix bugs in processors
 - Chip mask and fabrication costs
 - System HW and SW redesign
 - Lost market opportunity

CSE141-L-15

Pramod Argade, Spring '05

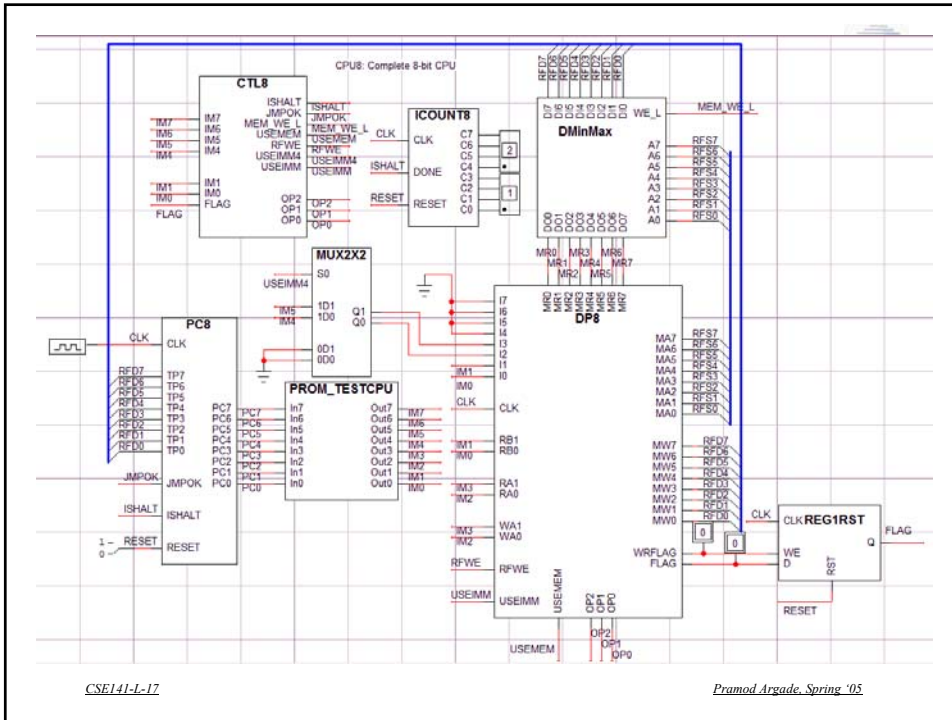
A Program to Test 8-bit CPU



```
0x3f 0000 start: mov   r3, $0xf // r3 = 0xf
0xb7 0001      sh14  r1, r3 // r1 = 0xf0
0x5f 0002      add   r3, $3 // r3 = 0x12
0x8b 0003      mov   r2, r3 // r2 = 0x12
0xae 0004      ushr4 r3, r2 // r3 = 1
0xfb 0005      st    r2, *r3 // M[1] = 0x12
0x97 0006      add   r1, r3 // r1 = 0xf1, test corruption of RAM
0x06 0007      mov   r2, $0 // r2 = 0
0x69 0008      sub   r2, $1 // r2 = 0xff
0xd9 0009      xor   r2, r1 // r2 = 0xe
0xe7 0010      ld    r1, *r3 // r1 = 0x12
0xa9 0011      ushr4 r2, r1 // r2 = 1
0x7a 0012      cmpeq r2, $2 // flag = 0
0x33 0013      mov   r0, 0xf // r0 = 0xf
0x52 0014      add   r0, $2 // r0 = 0x11, addr of cont1
0x40 0015      jmpf  *r0 // should jump
0x59 0016      add   r2, $1 // should not exec. r2 = 1
0x79 0017 cont1: cmpeq r2, $1 // flag = 1
0x53 0018      add   r0, $3 // r0 = 0x14
0x52 0019      add   r0, $2 // r0 = 0x16, address of cont2
0x41 0020      jmpt  *r0 // jmp cont2
0x43 0021      halt // should not halt
0xc9 0022 cont2: cmphi r2, r1 // flag = 0
0x53 0023      add   r0, $3 // r0 = 0x19
0x52 0024      add   r0, $2 // r0 = 0x1b
0x40 0025      jmpf  *r0 // should jump to cont3
0x43 0026      halt // should not halt
0xc6 0027 cont3: cmphi r1, r2 // flag = 1
0x53 0028      add   r0, $3 // r0 = 0x1e
0x53 0029      add   r0, $3 // r0 = 0x21
0x40 0030      jmpf  cont4 // should not jump to cont4
0x51 0031      add   r0, $1 // r0 = 0x22, address of cont5
0x42 0032      jmp  *r0 // jump to cont5
0x43 0033 cont4: halt // should not halt
0x69 0034 cont5: sub   r2, $1 // r2 = 0
0x6d 0035      sub   r3, $1 // r3 = 0
0xfb 0035      st    r2, *r3 // M[0] = 0
0xfb 0035      halt
```

CSE141-L-16

Pramod Argade, Spring '05



Before you leave



- Discussion on Lab 4 next Tuesday
- Remember to make appointment with the TAs to show your operational design