Lab 4:
Construct and test Complete 8-bit Processor
Due: 7/31/03
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 4 to design logic
- Simulate the operation executing 3 programs from Lab 1
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 as well as all the lower level modules you designed in LogicWorks 4.
- All the LogicWorks 4 files you created (to be submitted via email to BOTH the TAs).
- 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 in timing 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 T’s before 7/31/03
- Show TA that your CPU design works before above deadline.
- 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

• Avoid propagation of unknowns
  – Initialize on reset

• Build hierarchical design

• Test thoroughly at every level of hierarchy
  – Connect binary switches and hex kepads 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 application
• It is costly to fix bugs in processors
  – Chip mask and fabrication costs
  – System HW and SW redesign
  – Lost market opportunity

Demo of A Complete CPU
A Program to Test 8-bit CPU

```assembly
045 0000: mov r1, $0xf // r1 = $0xf
045 0001: shld r1, r2, 4 // r2 = r1 + 4
045 0002: addi r3, r3, 4 // r3 = r3 + 4
045 0003: mov r2, r2 // r2 = r2
045 0004: addi r1, r1, 1 // M[1] = 0
045 0005: subi r2, r1, 1 // r2 = r2 - r1
045 0006: mrs r2, r1 // should not halt
045 0007: addi r1, r1, 1 // r1 = r1 + 1
045 0008: subi r2, r1, 1 // r2 = r2 - r1
045 0009: b cond r1, r2 // should jump
045 000a: mov r0, r0 // should not halt
045 000b: addi r0, r0, 1 // r0 = r0 + 1
045 000c: b cond r0, r0 // should not halt
045 000d: subi r0, r0, 1 // r0 = r0 - 1
045 000e: b cond r0, r0 // should not halt

A Program to Test 8-bit CPU
```

---

A Program to Test 8-bit CPU

```assembly
045 0000: mov r1, $0xf // r1 = $0xf
045 0001: shld r1, r2, 4 // r2 = r1 + 4
045 0002: addi r3, r3, 4 // r3 = r3 + 4
045 0003: mov r2, r2 // r2 = r2
045 0004: addi r1, r1, 1 // M[1] = 0
045 0005: subi r2, r1, 1 // r2 = r2 - r1
045 0006: mrs r2, r1 // should not halt
045 0007: addi r1, r1, 1 // r1 = r1 + 1
045 0008: subi r2, r1, 1 // r2 = r2 - r1
045 0009: b cond r1, r2 // should jump
045 000a: mov r0, r0 // should not halt
045 000b: addi r0, r0, 1 // r0 = r0 + 1
045 000c: b cond r0, r0 // should not halt
045 000d: subi r0, r0, 1 // r0 = r0 - 1
045 000e: b cond r0, r0 // should not halt
```

---

A Program to Test 8-bit CPU

```assembly
045 0000: mov r1, $0xf // r1 = $0xf
045 0001: shld r1, r2, 4 // r2 = r1 + 4
045 0002: addi r3, r3, 4 // r3 = r3 + 4
045 0003: mov r2, r2 // r2 = r2
045 0004: addi r1, r1, 1 // M[1] = 0
045 0005: subi r2, r1, 1 // r2 = r2 - r1
045 0006: mrs r2, r1 // should not halt
045 0007: addi r1, r1, 1 // r1 = r1 + 1
045 0008: subi r2, r1, 1 // r2 = r2 - r1
045 0009: b cond r1, r2 // should jump
045 000a: mov r0, r0 // should not halt
045 000b: addi r0, r0, 1 // r0 = r0 + 1
045 000c: b cond r0, r0 // should not halt
045 000d: subi r0, r0, 1 // r0 = r0 - 1
045 000e: b cond r0, r0 // should not halt
```

---

A Program to Test 8-bit CPU

```assembly
045 0000: mov r1, $0xf // r1 = $0xf
045 0001: shld r1, r2, 4 // r2 = r1 + 4
045 0002: addi r3, r3, 4 // r3 = r3 + 4
045 0003: mov r2, r2 // r2 = r2
045 0004: addi r1, r1, 1 // M[1] = 0
045 0005: subi r2, r1, 1 // r2 = r2 - r1
045 0006: mrs r2, r1 // should not halt
045 0007: addi r1, r1, 1 // r1 = r1 + 1
045 0008: subi r2, r1, 1 // r2 = r2 - r1
045 0009: b cond r1, r2 // should jump
045 000a: mov r0, r0 // should not halt
045 000b: addi r0, r0, 1 // r0 = r0 + 1
045 000c: b cond r0, r0 // should not halt
Before you leave

• Class on 7/29/03 will be Office hour in AP&M 2444
• Remember to make appointment with TA to show your operational design