

CSE 141L Chien

Lab #3: 8-bit CPU Internals

Due 5pm, Friday, November 15, 2002 (turn in to lab tutors)

In this assignment, you will finally design a single-cycle implementation of a processor to execute your 8-bit ISA. At a minimum, your design will have a program counter (PC), a PC incrementer, an ALU, memory, and probably some internal storage. The ALU and internal storage should not be significantly changed from the last lab. The CPU design will execute the programs you wrote (hopefully correctly) for Lab 1.

Some things to keep in mind for this lab:

- Use hierarchical design (subcircuits) to make your design easier to understand and think about. The highest-level schematic should mostly be functional elements (register file, alu, memory, etc.) and wires/buses.
- Isolate control. Follow the text's lead by generating all control signals in one place from the opcode. This should make the design easier conceptually, at least.
- You will create a ROM to hold instruction memory and a RAM to hold data memory. Both will be initialized – the ROM(s) to hold your programs, the RAM to hold your input data.
- There will be ONLY ONE ROM, and ONLY ONE RAM for all three programs!
- You should make a 4:1 mux to control the starting PC for each program such that: {(mux control)->(PC value)} 00->(normal execution), 01->(PC value for program 1) 10->(PC value for program 2) 11->(PC value for program 3)
- The above control bits will be set by hand, before executing the first program, and after each program has finished executing. You will set the mux control to an appropriate value, step 1 clock cycle, and then set mux control to 00, and run the programs. The above steps will ensure that their programs can run sequentially, using 1 instruction memory (ROM).
- You should include a cycle counter, that will count from 0, to n, for each program, thus when the you change your PC values for each program, you can also reset the counter in that cycle. The counter should be wide enough to show all of the cycles for each program, which might be as high as two thousand or more. It should count up until the done signal goes high.
- There should be a done signal that goes high at the *halt* instruction.
- Keep in mind that you may have to debug your programs! Think about how to make your life easier before it happens.
- In the questions for the lab report, we are no longer looking for you to convince me that your design is wonderful (unless it is), but rather we are looking at how effectively you critique your own design.
- Remember that it is your responsibility to convince us that your CPU works, not our responsibility to figure that out ourselves. Providing sufficient and clear results and information is crucial.

What you turn in:

- A review of your ISA.
- Schematics of all circuits (including those presented in lab 2), hierarchically organized. The highest-level design needs to have all of the signals necessary to demonstrate correct program execution via the timing diagram.
- A timing diagram for each program demonstrating correct operation and other important data. It should at a minimum show results being generated (e.g., the accumulating sum of neighboring ones, etc.), the cycle counter, the PC. It need not show the whole execution of the program (particularly if it takes over 100 cycles or so!), but certainly the beginning and end and some execution of the main loop. It, once again, should be heavily annotated so we can figure out what is going on.
- The assembly and machine code for your three programs.
- Answers to the following questions:

1. Have you made any changes to your ISA? What were they? Why did you make them?
2. What are your dynamic instruction counts for program 1? program 2? program 3? (from your cycle counter)
3. What could you have done differently to better optimize for dynamic instruction count?
4. How successful were you at optimizing for ease of design? Give examples.
5. What could you have done differently to better optimize for ease of design?
6. How easy/difficult would it be to extend your design to a multicycle implementation? A pipelined implementation? Give examples.
7. What might you have done differently if a priority was ease of programming? Give examples.
8. What instruction takes longest on your machine (and thus would set the cycle time)? (Use rough estimates, e.g. assume each device introduces a constant delay).
9. What might you have done differently if a priority was short cycle time? (again, use rough estimates). Give examples.

The inputs

You have to use the same input for all three programs. In fact, you have to use the modified output after program 1 as input to program 2, etc.,:

Initial memory location values:

[Address/address range(dec)]:value(hex)

[0]: A7 [1]: 0D [2-15]: 00

[16-127]: as follows: (repeat for 7 times)

[16]: 6F	[17]: D8	[18]: A8	[19]: 5C
[20]: 4D	[21]: 8F	[22]: 13	[23]: EA
[24]: 47	[25]: 6A	[26]: 2C	[27]: 20
[28]: 7A	[29]: 48	[30]: CF	[31]: 5E

[128-191]: as follows:

[128]: 15	[129]: 10	[130]: 33	[131]: 2A
[132]: 2D	[133]: 0B	[134]: 16	[135]: 11
[136]: 31	[137]: 27	[138]: 2E	[139]: 2C
[140]: 19	[141]: 3A	[142]: 30	[143]: 05
[144]: 0D	[145]: 35	[146]: 15	[147]: 37
[148]: 2B	[149]: 38	[150]: 14	[151]: 16
[152]: 22	[153]: 21	[154]: 37	[155]: 35
[156]: 09	[157]: 00	[158]: 1E	[159]: 13
[160]: 07	[161]: 3C	[162]: 1B	[163]: 2A
[164]: 0B	[165]: 1F	[166]: 2C	[167]: 0C
[168]: 27	[169]: 1C	[170]: 3B	[171]: 3F
[172]: 38	[173]: 01	[174]: 07	[175]: 39
[176]: 31	[177]: 13	[178]: 0D	[179]: 1D
[180]: 17	[181]: 0F	[182]: 2C	[183]: 1D
[184]: 1F	[185]: 02	[186]: 2F	[187]: 24
[188]: 1C	[189]: 11	[190]: 21	[191]: 3E

The rest of the data should be 00 initially:

[192-255]: 00

Design Guidelines

1. Do not use any built-in Xilinx logic except: registers, memories, d flip-flops (if needed), and any basic logic functions such as: and, or, xor, etc. You should create all other parts such as muxes, demuxes, adders, etc. You can use any basic logic unit, such as AND, OR, XOR, NOT, and other BASIC logic gates. Using these gates, you can build anything, and everything more complex, through a careful use of macros YOU have created, to build up more complex units. For example, to build a ripple carry adder, you would: make a half adder macro -> using the half adder above, you would make a full adder macro -> then, you would combine these full adders in series, to make a ripple carry adder, that you can then use in your ALU to add, or subtract.
2. For lab report, include **all** xilinx schematics that you have created in a hierarchical manner, starting from the top level, such as the whole ALU, and down to a half-adder, or mux, if was created.
3. All intermediate, and final busses should have an output on the timing simulation, unless there is absolutely no need. This will demonstrate a progression of values through the circuit, and prove that the circuit works properly.
- 4: The timing diagrams included with the report, should be one per instruction, should be clearly annotated, so when the assignments can be graded with no confusion about what is going on.
- 5: **You should not change your ISA design** from lab to lab. That is, you should continue with the design you developed for Lab #1 and #2. If changes are absolutely critical, propose a minimal change for formal approval by the tutors – well before you need to turn your lab in.
6. To control the data flow, and data paths, you should use muxes and demuxes. For example, if there are two uses for a piece of hardware, you'd have a multiplexor at the input and a demultiplexor at the output.
7. You must use the Xilinx tools, not any other.
8. You should combine bit lines into busses, if you have values that travel from one component to another, and have more than one bit associated with it. This is a suggestion, but it will make you life easier, and your circuits cleaner.