In this assignment, you will finally design a pipelined implementation of a processor to execute your 10-bit ISA. At a minimum, your design will have a program counter (PC) and fetch unit, an ALU, memory, and some internal storage. The ALU, internal storage, and fetch unit 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.), wires/buses, pipeline latches, etc.
- 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. Instructions to create memory parts in Xilinx will be on the web page.
- You can actually have three ROMs for your three programs, and swap them in and out of your design as needed. Otherwise, you could have a single ROM with three different initial values for the PC.
- I want you to have an init signal that sets the PC to some predetermined value. I also want a done signal that goes high when the `halt` instruction completes.
- I want you to have two counters: one that counts dynamic instructions and one that counts dynamic cycles to execute the program. Both start when the first instruction is fetched (ie, at the `init` signal), and stop counting when the last instruction (`halt`) completes. The cycle counter is easy. The dynamic instruction counter should count instructions that go through a late pipeline stage, so that it does not count flushed instructions, and it should be able to distinguish between real instructions and bubbles.
- 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, I am no longer looking for you to convince me that your design is wonderful (unless it is), but rather I am looking at how effectively you critique/understand 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.
- There are several opportunities to increase your performance in this lab, but you should probably set priorities. Get things working first, then add features that improve CPI (reduce data hazards and control hazards) after.

**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. This would include at least instructions, source and dest register values, memory values and addresses, and the counters.
• 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 factorial product being generated, the even and odd values incrementing ...), 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. You should make an effort to highlight interesting pipeline behavior (e.g., data and control hazards).
• The assembly and machine code for your three programs.
• Answers to the following questions:
  1. Have you made any changes to your ISA from lab 1? What were they? Why did you make them?
  2. What are your dynamic instruction counts for program 1? program 2? program 3?
  3. What are your cycle counts for program 1? program 2? program 3?
  4. What is your CPI for each program? What did you do to reduce CPI? What’s the biggest remaining CPI limiter? How would you reduce CPI further?
  5. What could you have done differently to better optimize for dynamic instruction count?
  6. How successful were you at optimizing for ease of design? Give examples.
  7. What could you have done differently to better optimize for ease of design?
  8. How successful were you at optimizing for low cycle time? What could you have done better?
  9. How easy/difficult would it be to extend your design to a superscalar implementation? Give examples.
 10. What might you have done differently if a priority were ease of programming? Give examples.
 11. What pipeline stage takes longest on your machine (and thus would set the cycle time)?
     (Use what you know to make rough estimates of how long individual blocks would take, if necessary.)

**The inputs**
You will use the same input for all three programs. The inputs are specified in the “memory tutorial” where I give you the code to create a RAM part initialized to the correct values.