

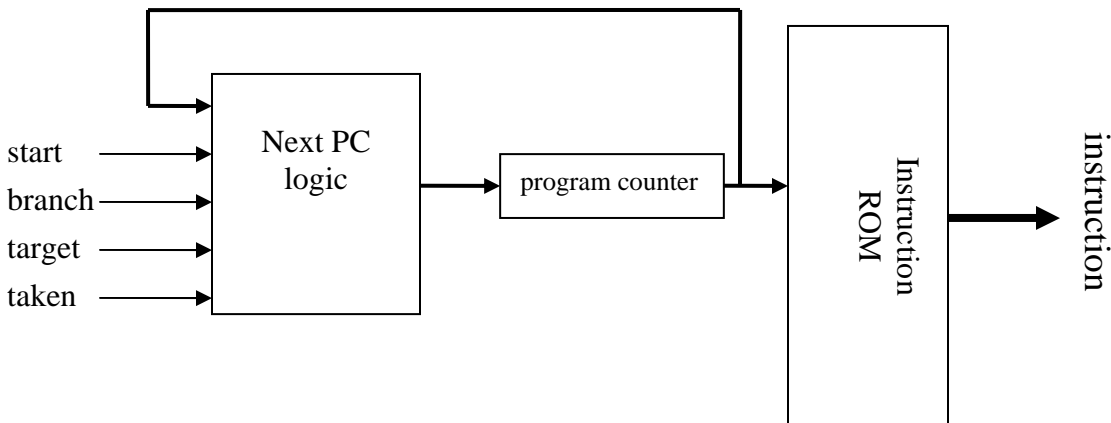
CSE 141 Lab #2: 8-bit CPU – ALU and fetch unit

due Friday, February 6

In this lab assignment, you will design the ALU (arithmetic logic unit) and the fetch unit for your CPU. You will not connect them together yet; you will demonstrate each separately through schematic and timing printouts.

You will have an ALU, possibly something like the schematic (ignore the register file – that will come later). The one shown here is simple and not necessarily reflective of what you will be designing. In this picture, the ALU has two 16-bit inputs, a 16-bit output, some control bits (okay, one). It also has a zero signal as an output.

The fetch unit is the logic responsible for fetching the current instruction from the instruction memory and determining the next program counter (PC). It should look roughly like the following diagram:



The *program counter* is a state element (register). *Instruction ROM* is a ROM memory part that holds your code. It doesn't have to actually hold your code at this point, but it might as well – but it should hold something so we can see the effect of changing PCs. The *next PC logic* takes as input the previous PC and several other signals and calculates the next PC value. The inputs to the next PC logic are: *start* – when asserted during a clock edge, it sets the PC to zero or the starting address of your program (note – if you store all three programs in one ROM, you'll have to do this a little differently). *branch* – when asserted it indicates that the prior instruction was a branch. *taken* – when the instruction is a branch, this signal when asserted indicates the branch was resolved as taken. On non-branch instructions, the next PC should be PC+1. For branch instructions, the new PC is either PC+1 (branch not taken) or *target* (branch taken). If your branches are ALWAYS PC-relative, then you can redefine *target* to be a signed distance rather than an absolute address if you want. Make sure you tell us this is what you're doing. Otherwise, just do it as I described.

You will demonstrate each element of your design in two ways. First, with schematics such as the one shown. Obviously, you must also show all relevant internal circuits. That will either be

further schematics or verilog code. Second, you must demonstrate correct operation of all ALU operations and fetch unit functions with timing diagrams. An example of a (partial) timing diagram is also included; yours will be longer. The timing diagrams, for example, should demonstrate all ALU operations (this includes math to support load address computation, or any other computation required by your design), each with a couple of interesting inputs. Make sure corner/unusual cases are demonstrated, if relevant. If you support instructions that do multiple computations at the same time, you need to demonstrate them happening at the same time unless you've already determined that they will happen in different pipeline stages. Note that you're demonstrating ALU operations, not instructions. So for example, instructions that do no computation (e.g., branch to address in register) need not be demonstrated. There will also be a timing diagram for the fetch unit, showing it doing everything interesting (fetching instructions, taking branches, not taking branches, etc.). The schematics and timing diagrams will be difficult for us to understand without a great deal of annotation. Good organization and use of hierarchical design also help.

All designs will use schematic entry with the Xilinx tools, or Verilog modules. The highest level should be a schematic (as in my simple example). When starting the project, specify the following (in the "New Project Wizard: Device Properties" window): Family: Virtex 2P, Device: XC2VP30, Package: FG676, Speed: -7. Then you can use any logic modules provided (eg, for schematic entry) for that device.

The lab report will contain the following:

1. Introduction and general comments. Overview of report.
2. Summarize your ISA from Lab 1 (operations supported, with full detailed descriptions).
3. A listing of ALU operations you will be demonstrating, including the instructions they are relevant to.
4. Full schematics, hierarchically organized.
5. Timing diagrams. It should be clear everything works. If your presentation leaves doubt, we'll assume it doesn't.

Answer the following questions:

6. How many (and what) ALU operations do you support?
7. Will your ALU be used for non-arithmetic instructions (e.g., address calculation, branches)? If so, how does that complicate your design?
8. Name one thing you could have done differently in your ISA design to make your ALU/register file design easier.
9. What is your most complex instruction, from the standpoint of the ALU?

10. Are your ALU control signals easily derived from your opcode bits? How much translation is needed? How could you have designed your ISA to eliminate (or at least reduce) that translation?

11. Now that your ALU is designed, are there any instructions that would be particularly straightforward to add given the hardware that is already there?

12. Is there anything you could have done in your ISA to make your ALU design job easier?

13. Is there anything you could have done in your ISA to make your fetch unit design job easier?

14. Give a qualitative description of your expected cycle time, as dictated by the ALU design – relatively fast, average, slowwww... (or better, it is the sum of ... access time plus ... plus ...). Explain. What could you do to improve cycle time?