

## Study Guide, Patterson & Hennessy, Appendix B, and beginning of Chapter 5

Finally, the hardware and the software are coming together! Here is the outline.

### I. The sequential logic builds the ALU.

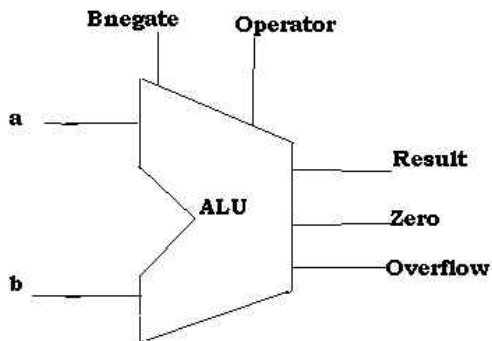
1. Create an adequate set of gates with which to represent all combinational circuits. AND, OR, NOT are an **adequate** set, which can be shown by representing any other boolean function in **disjunctive normal form** (dnf).

I showed that NAND itself is adequate. If someone is interested in a Wednesday presentation, showing that NOR itself is adequate would be one to try. The answer might be on the internet, and that would be fair to use if properly cited.

2. Using that adequate set, create some useful building blocks: **multiplexor** (mux), **decoder**, **encoder**, **half adder** and then eventually a **full adder**.

3. Use these building blocks to create a one-bit ALU. The secret is to choose the bit pattern for the instruction carefully so that the control input (at first we use Binvert and two Operator lines for control) corresponds to the selectors on two muxes. One mux decides whether to pay attention to Binvert; the other decides which simultaneous result is selected for actual output. Put 32 of those one-bit ALUs together to get a 32-bit ALU. Special considerations for ALU0 and ALU31, because the first treats CarryIn specially; the second, treats trapping an overflow.

When the Zero output wire is considered, we always have two outputs, Result and Zero. The ALU does not select which of those to pay attention to. We do that later in the datapath, not in Appendix B.



The resulting diagram looks like this.

Control lines			
Bnegate	Operator	operation	
0	00	AND	\$t3, \$t1, \$t2
0	01	OR	\$t3, \$t1, \$t2
0	10	ADD	\$t3, \$t1, \$t2
1	10	SUB	\$t3, \$t1, \$t2
1	11	SLT	\$t3, \$t1, \$t2
1	00	BEQ	\$t3, \$t1, Location

As you can see, the answer to BEQ is on the Zero line, but it must be the result of subtracting a-b. We don't pay attention to Result when the operation is BEQ. But it must be a-b because that's what beq is testing for zero. Whether we pay attention to Zero or to Result is decided somewhere other than in the ALU.

One can extend this to four Control lines, by adding an Ainvert, so that Ainvert and Binvert can together create a NOR. Appendix B helps you with that.

*You should know what is going on inside that ALU box.*

## II. The sequential logic builds the registers and RAM, and so we have a CPU and memory.

4. Once we have flip-flops for sequential logic, we can save state, hence we can have RAM and a **register file**. We will build a so-called **single-cycle CPU**. We connect ultimately as in the diagram on Page 314. This is terribly complex! So we take it a piece at a time. Here are the page numbers for the pieces:

- 287, 293, 295: Simple.
- 296: Interesting. We see that BEQ instruction has the word offset from PC+4 stored in its low order 16 bits, multiplied by 4 on the fly; we see that branches can be  $\pm$  because the offset is sign-extended; and we can easily imagine that sign-extension is a combinational activity. ALUoperation here has four wires instead of the three we mentioned above because the authors did include an Ainvert line. See the table on page 301.
- 288: We do not see the Control oval in this picture, although there is some control from the Instruction. We will look inside that at a later refinement. We do not see the uppermost Mux and Branch, although here's how it works: The uppermost Mux selects between the "PC+4" answer and the "BEQ is true" answer. The Branch wire is asserted if the instruction is BEQ. The Zero line is asserted if the BEQ instruction is true, so the AND at the top of the diagram makes the Mux do exactly the right thing. Wow!
- 299: We see the Mux driven by the line now named ALUSrc, but not the Mux driven by MemtoReg yet.
- 300: This puts most of the above things into one diagram, leaving out some branch logic from the diagram on page 288, and leaving out some future things to think about: the jump instruction, and control line information. It's not all the way to our goal of page 314. And that in turn is only a step along the way to a *multi-cycle CPU*.

A **single-cycle CPU** requires that instruction memory and data memory be kept separate, since they must both be accessed on the same clock cycle. It is impossible to access the same memory twice on the same clock cycle. The multi-cycle CPU will fix that.