Homework 3 (due Feb 4)

Design of a simple processor

The processor we will design will have the following components:

- 8 32-bit registers R_0, R_1, …, R_7 for holding data.
- 256 16-bit registers P_0, P_1, …, P_{255} for holding a program.
- A 32-bit adder.
- An 8-bit register PC that will serve as a program counter.

It is useful to have the constant values 0 and 1 available for use. So we set registers R_0 and R_1 to hold the values 0 and 1, respectively, permanently.

The processor will have five types of instructions: *add*, *negate*, *load*, and *jump if zero*. A program consists of a sequence of instructions stored in the 256 program registers. Each of these registers holds 16 bits. The 16 bits of an instruction specify the type of instruction and its operands. We need two bits to specify the instruction; the first two of the 16 bits (positions 0 and 1) will specify the instruction type. The formats of the 4 instructions are as follows.

The add instruction adds the contents of two registers R_a and R_b and stores the result in register R_c. The indices a, b, and c are specified in bit positions 2-4, 5-7, and 8-10, respectively. Bit positions 11-15 will be ignored. The add instruction also increments the program counter by 1. The add instruction

$$add\ R_a, R_b \rightarrow R_c$$

is encoded as follows.

0	1	2 3 4	5 6 7	8 9 10	11	12	13	14	15
0	0	a	b	c	0	0	0	0	0

The negate instruction replaces R_a with $-R_a$, using the two's complement representation. The index a is specified by bit positions 2-4. The negate instruction also increments the program counter by 1. The negate instruction

$$neg\ R_a$$

is encoded as follows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	a			0	0	0	0	0	0	0	0	0	0	0

The load instruction loads an 8-bit number d into the 8 low-order bit positions of register R_a. The index a is specified by bit positions 2-4. The value d is specified in binary by the 8 low-order positions of the instruction; that is, positions 8-15. The effect of the instruction is to set the value in register R_a to d; note that the 24 high-order bits of R_a are set to 0. Also, the program counter is incremented by 1. The load instruction

$$lod\ d \rightarrow R_a$$

is encoded as follows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	a			0	0	0	d							

The jump if zero instruction changes the program counter register to the value specified by an 8-bit number d, if register R_a is 0; otherwise the program counter PC is incremented by 1, as usual. The jump if zero instruction

$$jiz\ R_a \rightarrow d$$

is encoded as follows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	a			0	0	0	d							

1. Design the entire processor using basic components that we have studied. These components include gates (unclocked and clocked), adders, multiplexers, demultiplexers, and registers. You may use as many adders, multiplexers, demultiplexers, and registers of any size as you need. You may also design and use your own circuits as you wish. You may also use the constant values 0 and 1 as inputs to any of your circuits.

   Show your design using a diagram. You need not show the design of the basic components we have already built. For instance, if you are using a 4-way 1-bit multiplexer, you may simply draw a picture depicting the multiplexer. Make your design as simple as possible, without worrying about optimizing the size of the circuit.

   Assume that the registers $P_0, \ldots, P_{255}$ already contain the instructions to run, and their contents do not need to be changed.

   (*Hint:* The program counter should serve as the control input to a multiplexer, that selects the instruction to execute. The registers should serve as data inputs to one or more multiplexers.)

2. Write a program for the above processor to multiply two numbers, which are stored in registers R_2 and R_3. The final result is to be placed in register R_4. Don't worry about overflows. Since our processor does not have a halt instruction, you can end your program by going into an infinite loop.

The following example program takes two numbers, say x and y, which are stored in registers R_2 and R_3, respectively, computes $x + 15 - y$, stores the result in R_4, and enters into an infinite loop.

$$\begin{array}{lll}
0 & lod & 15 \to R_5 \\
1 & add & R_2, R_5 \to R_6 \\
2 & add & R_3, R_0 \to R_5 \\
3 & neg & R_5 \\
4 & add & R_6, R_5 \to R_4 \\
5 & jiz & R_0 \to 5
\end{array}$$

3. Write a program for the above processor to compute F_n, the nth Fibonacci number. Assume that n is stored in register R_2, and that the final answer is stored in R_3. Again, don't worry about overflows; and you can end your program by going into an infinite loop.