

CS220: Lab#8A

1. [10 points] Implement a register file having 32 registers where each register is 16-bit wide. The register file interface needs to be designed such that it has two read ports and one write port. You need to write an interface module to access the register file. The module should have the following inputs: two read addresses, one write address, one write data, and three bits that specify which of the three ports have valid address inputs. The outputs of the module are the two read data values. Initialize the registers with zero values. Write a top-level instruction processor module to demonstrate that the register file works. This module defines eight commands or instructions to interact with the register file access module. These commands are listed below along with the binary encoding of them in parentheses.

- No read, one write (000)
- One read, no write (001)
- Two reads, no write (010)
- One read, one write (011)
- Two reads, one write (100)
- Two reads, add two read values, write the result, ignore overflow (101)
- Two reads, subtract second read value from the first value, write the result, ignore overflow (110)
- One read, shift the read value left, write the result (111)

Take the three-bit command as input using slide switches. Depending on the command, your hardware should accept further inputs. For example, if the command is 010, you will have to enter the addresses of two registers also. If the command is 000, you will have to enter the address of a register and the 16-bit data that will be written to the register. If the command is 100, you will have to enter the addresses of two read registers and one write register, and the 16-bit value to be written. If the command is 101, 110, or 111, you will have to enter the addresses of two read registers and one write register. If the command is 111, you also have to enter the shift amount (maximum 15). Use whatever means you know of to accept the inputs. Also, keep a provision of aborting an input midway and starting over. For commands 101, 110, and 111, the writing of the result should be done after the operation completes. Assume that the operation takes 16 cycles. Assume that register read takes two cycles and register write takes two cycles. On execution of each command, the output will have to be shown in the LCD. For each command, the output is defined in the following. Feel free to introduce additional modules.

- If command is 000, the first line of output is the address of the register in binary and the second line of output is the 16-bit value written.
- If command is 001, the first line of output is the address of the register in binary and the second line of output is the 16-bit value read.
- If command is 010, the first line of output is the first 16-bit value read and the second line of output is the second 16-bit value read.
- If command is 011, the first line of output is the address of the register read in binary and the second line of output is the 16-bit value read.

- If command is 100, the first line of output is the first 16-bit value read and the second line of output is the second 16-bit value read.
- If command is 101 or 110 or 111, the first line of output is the address of the register written in binary and the second line of output is the 16-bit value written.

Test program: Notice that the designed hardware is a miniature processor with a register file and eight instructions. It has a 16-bit adder/subtractor and a left shifter. In the following, I list the C statements of a program along with translation into assembly language of this miniature processor. I use the binary command encoding as the instruction names. The registers are numbered \$0 to \$31. All numbers in the following are listed in decimal. Use 16-bit two's complement representation for binary conversion. Enter the instructions of the following program one by one and show the TA the corresponding output.

C statements	Assembly language instructions
short a, b, c, d;	
a = 17;	000 \$1 17
printf("%h", a); b = -9;	011 \$1 \$2 -9
printf("%h %h", a, b); c = 65;	100 \$1 \$2 \$3 65
printf("%h %h", b, c);	010 \$2 \$3
d = 8*c - 512*(a + b);	111 \$3 \$5 3 // 8*c
	101 \$1 \$2 \$4 // a + b
	111 \$4 \$4 9 // 512*(a + b)
	110 \$5 \$4 \$6 // d
printf("%h", d);	001 \$6