16.216: ECE Application Programming

Fall 2012

Programming Assignment #8: Instruction Decoding and File I/O

Due **Wednesday, 11/28/12**, 11:59:59 PM

1. Introduction

In this assignment, you will work with files to handle input and output. Your program will expand upon the instruction decoding concept introduced earlier, modifying your program so that it reads its initial register values and list of instructions from input files, and it prints its results to an output file.

2. Deliverables

Submit your source file directly to Dr. Geiger (Michael_Geiger@uml.edu) as an e-mail attachment. Ensure your source file name is ***prog8_files.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

Note: a "starter" version of the .c file, which essentially contains an outline of what your main program should look like as well as some code and variables to be used, is available on the website. You may use this file as a starting point for your own code.

3. Specifications

NOTE: See Section 5 for a full breakdown of instruction encoding, possible opcodes, and detailed information about the program inputs.

Variables: Your program should contain, at a minimum, the following variables:

- `unsigned int inst`: The current instruction
- `int regs[32]`: Array of register values. If you have an operation that uses R0, R5, and R22, you'll access regs[0], regs[5], and regs[22].
- `unsigned int opcode`: Instruction field indicating operation to be performed.
- `unsigned int dest`: Instruction field that holds destination register number.
- `unsigned int src1`: Instruction field with number of first source operand.
- `unsigned int src2`: Instruction field with number of second source operand.
- `unsigned int shamt`: Instruction field that holds shift amount; only used for left and right shift operations.

You will also need to add variables to allow you to handle two different input files—one binary file, one text file—and one output file. You may need additional variables to successfully complete the program.

Program outline: The program should perform the following operations:

- Prompt the user to enter the name of a binary input file. This file will contain 32 integer values—open the file and read these values into the `regs[]` array to provide the initial register values.
    - Sample binary files: `invals1.bin`, `invals2.bin`, and `invals3.bin`.
    - Note that these files, as well as the text input file(s), should be placed in the same directory as your source code or executable.
- Prompt the user to enter the names of two text files: one to be used for program input; the other to be used for program output. Open each of these files.
    - The input file will contain a series of "instructions"—32 bit unsigned integers in hexadecimal format.
    - The sample program input file is `p1.txt`; additional programs will be posted shortly.
- Set up a loop that will repeatedly read a single instruction from the program input file, stopping when it reaches the end of the file.
- For each instruction:
    - Decode the instruction into the appropriate fields. Note that the code to perform the decoding is provided in the starter file.
    - Perform the appropriate operation on the source values. See Section 5 for a list of operations and corresponding opcodes.
    - Print the following output to the open text output file:
        - Line 1: The instruction, printed in the form:

          ```
          INSTRUCTION <#>: <inst>
          ```

          The number printed indicates how many instructions have been read thus far.
        - Line 2: The registers and operation, in the form:

          ```
          <dest> = <src1> <op> <src2>
          ```

        - Line 3: The values used in calculation and the result, in the form:

          ```
                 = <val1> <op> <val2> = <result>
          ```

          For example, the first instruction may generate the following output:

          ```
          INSTRUCTION 0: 0x04011000
          R0 = R1 + R2
             = 1 + 2 = 3
          ```

    - Store the instruction result in the appropriate `regs[]` array element.
        - Note: I suggest storing the result after printing the output, to ensure the output is correct. For example, if you have the instruction:

          ```
          R17 = R17 + R17
          ```

          You can only print the correct source values on the next line if you don't overwrite `regs[17]` until after you've printed the output.

Error checking: If your program cannot open any of the three files (binary input file, text input file, text output file), print an error message and open the program.

You may assume all input files are formatted correctly and therefore will not generate any errors.

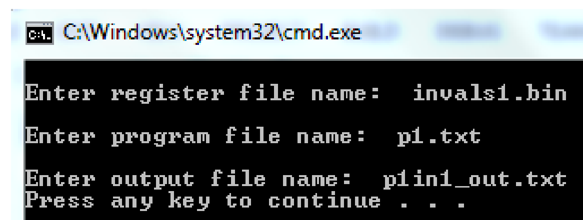
Hints: Most hints for this program can be found in one of two sources:

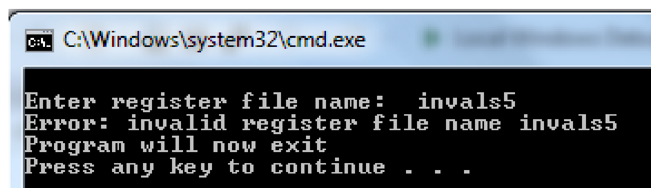
- The starter file, which contains an outline of the program.
- The lecture slides, code, and recording from Monday, 11/19.

For those of you who did not attend class on 11/19, I strongly encourage you to look over the code developed during that in-class programming exercise and to watch the lecture recording for step-by-step details of how we developed the program. Your program for this assignment will be similar in many ways to that exercise.

4. Test Cases

The screenshots below show the console input and output from two different program runs—one in which all files open successfully, and another in which the user provides an invalid input file name:





The more useful test cases can be found on the web page, in the form of input and output files. As of 11/20, there is one program input file (`p1.txt`) and three binary input files (`invals1.bin`, `invals2.bin`, `invals3.bin`), as well as output files showing the results of running this program with each set of input values (`p1in1_out.txt`, `p1in2_out.txt`, `p1in3_out.txt`). Additional program input and output files will be added shortly.

Your output files should match these test cases exactly for the given input values. I will use these test cases in grading of your lab, but may also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

5. Additional Details

Instruction encoding (fields are described in Section 3):

31 26	25 21	20 16	15 11	10 6	5 0
opcode (6 bits)	dest (5 bits)	src1 (5 bits)	src2 (5 bits)	shamt (5 bits)	UNUSED

Opcode values and operations to be performed. Note that most operations use two source registers, but the left and right shift operations (opcodes 5 and 6) use the shift amount field as the second operand:

<table>
<tr><th>Opcode</th><th>Operation</th><th>Example instruction</th></tr>
<tr><td>1</td><td>Add</td><td>0x04011000
R0 = R1 + R2</td></tr>
<tr><td>2</td><td>Subtract</td><td>0x08642800
R3 = R4 - R5</td></tr>
<tr><td>3</td><td>Multiply</td><td>0x0cc74000
R6 = R7 * R8</td></tr>
<tr><td>4</td><td>Divide</td><td>0x112a5800
R9 = R10 / R11</td></tr>
<tr><td>5</td><td>Left shift</td><td>0x158d0200
R12 = R13 << 8</td></tr>
<tr><td>6</td><td>Right shift</td><td>0x19cf00c0
R14 = R15 >> 3</td></tr>
<tr><td>7</td><td>Bitwise AND</td><td>0x1e119000
R16 = R17 & R18</td></tr>
<tr><td>8</td><td>Bitwise OR</td><td>0x2274a800
R19 = R20 | R21</td></tr>
<tr><td>9</td><td>Bitwise XOR</td><td>0x26d7c000
R22 = R23 ^ R24</td></tr>
</table>

Contents of binary input files: Values listed should be loaded directly into `regs[]` array; if read correctly, each element of `regs[]` will start with the value shown:

	invals1.bin	invals2.bin	invals3.bin
regs[0]	0	-65536	5
regs[1]	1	-32768	1785
regs[2]	2	-16384	-333
regs[3]	3	-8192	117
regs[4]	4	-4096	-906
regs[5]	5	-2048	10000
regs[6]	6	-1024	-1978
regs[7]	7	-512	26
regs[8]	8	-256	12
regs[9]	9	-128	-24
regs[10]	10	-64	2007
regs[11]	11	-32	2111
regs[12]	12	-16	-2112
regs[13]	13	-8	718
regs[14]	14	-4	-3
regs[15]	15	-2	-88
regs[16]	16	-1	15
regs[17]	17	0	3
regs[18]	18	1	-916
regs[19]	19	2	-12345
regs[20]	20	4	1000000
regs[21]	21	8	2552
regs[22]	22	16	13
regs[23]	23	32	14
regs[24]	24	64	86
regs[25]	25	128	17
regs[26]	26	256	119
regs[27]	27	512	-890
regs[28]	28	1024	-5152
regs[29]	29	2048	33
regs[30]	30	4096	16
regs[31]	31	8192	8