16.216: ECE Application Programming

Fall 2012

Programming Assignment #3: Instruction Decoding

Due **Friday, 9/28/12**, 11:59:59 PM

1. Introduction

This assignment will give you practice with the C bitwise operators and conditional statements. Your program will simulate a very simple processor that is controlled using an 8-bit "instruction." The program will decode this instruction to determine what operation should be performed and what values should be used in the computation.

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 ***prog3_decode.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

General description: When C programs are compiled, they are converted to *instructions*—simple operations that processors execute. Most instructions specify an operation to be performed and the data to be used in that operation.

Processors often store data in *registers*—temporary storage locations that are referenced by name or number in the instruction, as shown in the example below. This instruction adds the contents of registers 0 and 1 (the *source operands*) and stores the result in register 2 (the *destination operand*):

ADD R2, R0, R1

In practice, each instruction is encoded as a bit sequence; the processor *decodes* those bits to determine the operation and operands used for each instruction. Each possible operation is assigned a number, or *opcode*—for example, 0 might represent addition. Registers are usually referred to by number.

This program simulates a simple processor with four operations (add, subtract, multiply, divide) and four registers. The "instruction" that you will input uses a total of 8 bits—2 bits for the operation, 2 bits for the destination operand, and 2 bits for each of the two source operands.

The example instruction above would be encoded as 0x21 = $0010\ 0001_2$:

- The first two bits (00) indicate the operation (add)
- The next two bits indicate the destination register number ($10_2 = 2 \rightarrow$ R2)
- The next four bits indicate the two source register numbers (00 and 01 $\rightarrow$ R0 and R1)

General description (cont.): Further instruction examples are shown below:

Encoded inst.	**Opcode**	**Destination**	**Source 1**	**Source 2**	**Actual inst.**
0x3A = 0011 1010_2	00_2 → Add	11_2 → R3	10_2 → R2	10_2 → R2	ADD R3, R2, R2 (R3 = R2 + R2)
0x5C = 0101 1100_2	01_2 → Subtract	01_2 → R1	11_2 → R3	00_2 → R0	SUB R1, R3, R0 (R1 = R3 – R0)
0x89 = 1000 1001_2	10_2 → Multiply	00_2 → R0	10_2 → R2	01_2 → R1	MUL R0, R2, R1 (R0 = R2 * R1)
0xFB = 1111 1011_2	11_2 → Divide	11_2 → R3	10_2 → R2	11_2 → R3	DIV R3, R2, R3 (R3 = R2 / R3)

The actual result of the operation depends on the values stored in each register when the instruction executes. For example, if R2 = 7, then ADD R3, R2, R2 would assign the value 7 + 7 = 14 to R3.

Input: Your program should prompt the user to enter the following:

- Four integers that represent the values stored in R0, R1, R2, and R3.
- An 8-bit hexadecimal value representing the instruction
  - o Use the format specifier `%x` to read a hexadecimal value.
  - o Note that, although the instruction is only 8 bits, you should store it in a variable of type `unsigned int`.

Your program might produce the following first two lines (inputs are underlined):

```
Enter values for 4 integer registers: 1 2 3 4
Enter single byte for instruction (in hex): 0xC6
```

Error checking: You do not have to check any errors in this assignment—assume all inputs will be valid.

Output: Your program should perform the operation specified by the encoded instruction, and print the following on two separate lines:

- Line 1: The registers and operation, in the form:
  `<dest> = <src1> <op> <src2>`
- Line 2: The values used in the calculation and the result, in the form:
  `= <val1> <op> <val2> = <result>`

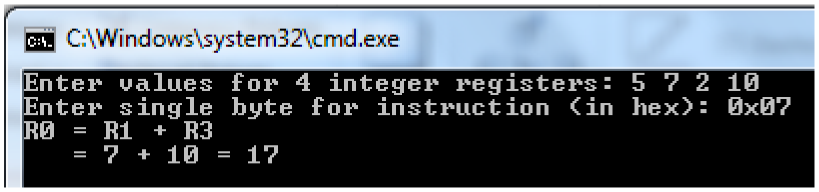
For example, the sample inputs given above would produce the output:

```
R0 = R1 / R2
   = 2 / 3 = 0
```

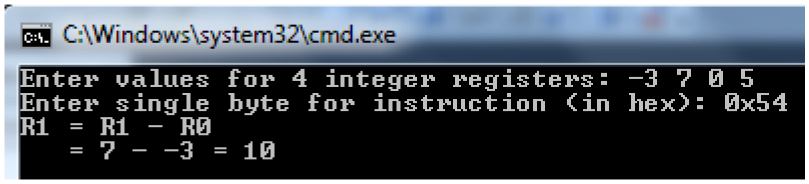
See Section 4: Test Cases for more sample program runs.

4. Test Cases

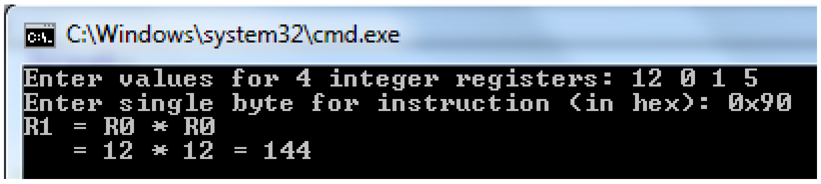
Your output should match these test cases exactly for the given input values. I will use these test cases in grading of your assignment, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create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



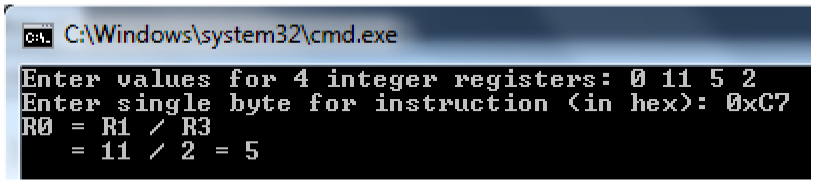
```
Enter values for 4 integer registers: 5 7 2 10
Enter single byte for instruction (in hex): 0x07
R0 = R1 + R3
   = 7 + 10 = 17
```



```
Enter values for 4 integer registers: -3 7 0 5
Enter single byte for instruction (in hex): 0x54
R1 = R1 - R0
   = 7 - -3 = 10
```



```
Enter values for 4 integer registers: 12 0 1 5
Enter single byte for instruction (in hex): 0x90
R1 = R0 * R0
   = 12 * 12 = 144
```



```
Enter values for 4 integer registers: 0 11 5 2
Enter single byte for instruction (in hex): 0xC7
R0 = R1 / R3
   = 11 / 2 = 5
```