

Computing Machinery I

Assignment 2

Integer Multiplication using Add and Shift Operations

Create an ARMv8 assembly language program that implements the following integer multiplication program:

```
#include <stdio.h>
#define FALSE 0
#define TRUE 1

int main()
{
    int multiplier, multiplicand, product, i, negative;
    long int result, temp1, temp2;

    // Initialize variables
    multiplicand = -16843010;
    multiplier = 70;
    product = 0;

    // Print out initial values of variables
    printf("multiplier = 0x%08x (%d)  multiplicand = 0x%08x (%d)\n\n",
           multiplier, multiplier, multiplicand, multiplicand);

    // Determine if multiplier is negative
    negative = multiplier < 0 ? TRUE : FALSE;

    // Do repeated add and shift
    for (i = 0; i < 32; i++) {
        if (multiplier & 0x1) { // If LSB multiplier == 1: (If (nonzero))
            product = product + multiplicand;
        }

        // Arithmetic shift right the combined product and multiplier
        multiplier = multiplier >> 1;
        if (product & 0x1) { // If LSB product == 1
            multiplier = multiplier | 0x80000000;
        } else {
            multiplier = multiplier & 0x7FFFFFFF;
        }
        product = product >> 1;
    }

    // Adjust product register if multiplier is negative
    if (negative) {
        product = product - multiplicand;
    }

    // Print out product and multiplier
    printf("product = 0x%08x  multiplier = 0x%08x\n",
           product, multiplier);
}
```

```

// Combine product and multiplier together
temp1 = (long int)product & 0xFFFFFFFF;
temp1 = temp1 << 32;
temp2 = (long int)multiplier & 0xFFFFFFFF;
result = temp1 + temp2;

// Print out 64-bit result
printf("64-bit result = 0x%016lx (%ld)\n", result, result);

return 0;
}

```

Be sure to use 32-bit registers for variables declared using *int*, and 64-bit registers for variables declared using *long int*. Use *m4* macros to name the registers to make your code more readable. Name the program *assign2a.asm*. Optimize your code so that it uses as few instructions as possible.

Also run the program in *gdb*, displaying the contents of key registers as the program executes; you should show that the algorithm is working as expected. Capture the *gdb* session using the Unix command *script* and name the output file *script.txt*.

Create a second version of the program (called *assign2b.asm*) that uses 522133279 for the multiplicand, and 200 for the multiplier. Also create a third version of the program (called *assign2c.asm*) that uses -252645136 for the multiplicand, and -256 for the multiplier.

Other Requirements

Make sure your code is properly formatted into columns, is readable and fully documented, and includes identifying information at the top of each file. You must comment each line of assembly code. Your code should also be well designed: make sure it is well organized, clear, and concise.

New Skills Needed for this Assignment:

- Use of bitwise logical and shift operations
- Use of branching and condition code tests
- Understanding of hexadecimal and binary numbers

Submit the following:

1. Your assembly source code files for the 3 programs, and your script output file. Use the *Assignment 2* Dropbox Folder in D2L to submit electronically. The TA will assemble and run your programs to test them. Be sure to name your programs and script file as described above.

Computing Machinery I

Assignment 2 Grading

Student:_____

Functionality:

Correct calculation of product	4	_____	
Use of bit masking	6	_____	
Use of shift instructions	3	_____	
Display to screen	3	_____	
Optimization	3	_____	
Script showing <i>gdb</i> session	3	_____	
Complete documentation and commenting	4	_____	
Design quality	2	_____	
Version 2	2	_____	
Version 3	2	_____	
Total	32	_____	_____%