ITCS 3181 Second Summer

Lab Report 02

August 5, 2016

Justin Le

Joshua Locklear

**Purpose and Experience:**

The purpose of the assignment was to recreate an algorithm that allows for multiplication using the add and shift method which will be explain in detail later. Throughout the lab, our general goal was to attempt to get the assignment done logically, and our secondary goal was to figure out how we could do this with as few lines of code as possible.

Throughout this process, we first started writing the process on paper. Doing this was beneficial to us because without computer assistance and simulation, we were able to plan the code in a way that allowed us to understand the process much better. Next, during the coding portion of the lab wrote our code that we had developed on paper. However, we didn’t properly account for the fact that the code needed to perform this operation would require two registers.

Now we would like to speak to you on the design of our asm code designed for multiplication. Our program consist of the main function, the Multiply function, loop functions, shift functions, end function, and a print function. However, in implementing our design, our code was having issues because we only used one register to store answers. This deficiency caused us to get only one half of the answer while the other half was not properly recorded. To fix this, we decided it was impossible unless we added a new register. However, this task presented it’s own problem. Though, through trial and error, this problem was quickly solved. We were able to work through our problems by creating a new shift function for shifting registers that we used to store our answer.

However, design and the previous problem were not the only problems we faced. Another dilemma that we had encountered was the fact that once we started adding, we found out that we had an overflow that was not accounted for. To solve this, instead of just using the add function, we decided to use the addu function. The addu function’s purpose is to allow for addition, but unsigned without overflow. In the end this solved our major problems and we were able to complete our assignment, while completing both of our goals.

Throughout the first set of labs, learning assembly on the fly was not very fun but once it was understood it became easier. Assignment delegation was given to each person on a basis of who knew how to do the particular part as for the lab report it was split in half.

**Technical explanation of add-shift multiplication algorithm:**

There are only four rules for multiplying binary integers. What make the multiplication difficult is the need to add the partial products obtained by multiplying the upper and lower integer. Binary multiplication can be performed by scanning the bits of the multiplier right to left. Then one would add the multiplicand to the product depending on whether the bit is 1 or 0 then left shifting the multiplicand after each test. Left shifting shifts each bit one position to the left while adding a zero to the right end and discarding the left most bit. Right shifting would do the opposite of this technique.

Example:

1011

101

-------

1011

0000

1011

--------

110111

**Code:**

#Justin Le

#Joshua Locklear

main:

#prompt for multiplicand

li $v0,4 # code for print\_string

la $a0,getA # point $a0 to prompt string

syscall # print the prompt

#acquire multiplicand

li $v0,5 # code for read\_int

syscall # get an int from user --> returned in $v0

move $s0,$v0 # move the resulting int to $s0

move $s5,$s0 # copy of multiplicand to use in multu

#prompt for multiplier

li $v0,4 # code for print\_string

la $a0,getB # point $a0 to prompt string

syscall # print the prompt

#acquire multiplier

li $v0,5 # code for read\_int

syscall # get an int from user --> returned in $v0

move $s1,$v0 # move the resulting int to $s0

move $s6,$s1 # copy of multiplier to use in multu

jal MyMult

j print

MyMult:

move $s3, $0 # lw product

move $s4, $0 # hw product

beq $s1, $0, done

beq $s0, $0, done

move $s2, $0 # extend multiplicand to 64 bits

loop:

andi $t0, $s0, 1 # LSB(multiplier)

beq $t0, $0, next # skip if zero

addu $s3, $s3, $s1 # lw(product) += lw(multiplicand)

sltu $t0, $s3, $s1 # catch carry-out(0 or 1)

addu $s4, $s4, $t0 # hw(product) += carry

addu $s4, $s4, $s2 # hw(product) += hw(multiplicand)

next:

# shift multiplicand left

srl $t0, $s1, 31 # copy bit from lw to hw

sll $s1, $s1, 1

sll $s2, $s2, 1

addu $s2, $s2, $t0

srl $s0, $s0, 1 # shift multiplier right

bne $s0, $0, loop

done:

jr $ra

print:

# print result string

li $v0,4 # code for print\_string

la $a0,result # point $a0 to string

syscall # print the result string

# print out the result

li $v0,1 # code for print\_int

move $a0,$s4 # put result in $a0

syscall # print out result

li $v0,4 # code for print\_string

la $a0,space # point $a0 to string

syscall # print the result string

li $v0,1 # code for print\_int

move $a0,$s3 # put result in $a0

syscall # print out result

# print the line feed

li $v0,4 # code for print\_string

la $a0,endLine # point $a0 to string

syscall # print the linefeed

.data

getA: .asciiz "Please enter the first number: "

getB: .asciiz "Please enter the second number: "

space: .asciiz " "

result: .asciiz "The product is: "

endLine: .asciiz "\n"

.text

Examples and answers:

|  |  |  |  |
| --- | --- | --- | --- |
| First Number: | 1143330295 | 1122430295 | 1212430291 |
| Second Number: | 999999223 | 855559223 | 844449213 |
| Product | 266202121 -1483441327 | 223588568 -552578639 | 238380349 2088244679 |