ITCS 3181L, Assembly Language Multiplication

Bryan Ritter (No lab partner used for this)

Due Date: 2013-04-18

Course: ITCS 3181L

Simulation Complete: 2013-04-18

# Multiplication

Purpose:

To make a multiplier using ADD, AND, NOT, and PASS; assuming multiplication of two 31 bit unsigned numbers in a 32 bit field into two 32 bit registers.

Analysis:

This program takes in sets of two 31 bit unsigned numbers and multiplies them together into two 32 bit registers, and prints the result onto the console. If the result is zero for the more significant bits half or the most significant bit in the less significant bit section only the value in the lower 31 bits will print. Since this setup can’t directly display more than 32 bit signed numbers, a workaround was done to split the result up into parts: the most significant 32 bits register \*2^32 + , the most significant bit in the least significant bit section \*2^31 + , and the rest of the least significant bits section. The two sides of the equal sign are equal to each other. If a number is less than 2^31 no extra math will be needed.

If one prefers to enter in numbers at run time one can comment out the jump statement lblFixedData around line 21.

Lab Experience:

I originally did this with runtime entered values, and then decided to make this use predetermined values for ease of testing. To do this I used function calls instead of plain jumps. (Which I had read up on how to do) I ran into a problem where the first item was good while the rest weren’t. It took me a while to figure out what was going wrong, but once I figured out I hadn’t re-initialized some variables I should have had it was an easy fix.

At first, I wasn’t sure of the best way to show the results, and after some thought I came with the \*2^32… idea. I tried looking to see how you wanted the results, but didn’t see anything about that so I went with the \*2^32 idea. I found out that I was printing out signed numbers when I came across a number that printed out negative. Printing out unsigned int wasn’t in the list of output types, tried looking it up, and found a number 36 for output type, but that didn’t work. I almost was just going to leave it that way, but then was like it couldn’t be too hard to do a \*2^31 workaround, and changed did it.

For this lab I basically ran out of type ‘t’ registers, and used one type ‘s’, I’m not sure of the difference between the features of the two, but it works, so I’m OK with it. Could you explain what the difference is?

My Code:

.text

.align 2

.globl main

main:

move $t8, $ra ;# using functions so I need to store the return spot.

;# initialize variables

li $t6, 1 ;# amask, aka 'g'

li $t7, 0x80000000 ;# cmask, aka 'h'

li $t9, 0x7fffffff ;# dmask, for semi conversion to signed int

;# 2^32-1=4294967295, 100000000

;# (2^31)-1=2147483647 ;# max number this can handle currently

;# ############################################################

;# pick which style of input you wish to run this program with

;# comment out to run with input prompts instead

;# sorry it's not set to accept more than two numbers to input

;#

j lblFixedData

;# ############################################################

;# get numbers to multiply from keyboard input section

li $v0, 4

la $a0, prompt1 ;# prompt for first number to multiply

syscall

li $v0, 5 ;# read keyboard into $v0 (the first number to multiply)

syscall

move $t0, $v0 ;# first number will now be in $t0, aka 'a'

li $v0, 4

la $a0, prompt2 ;# prompt for the second number

syscall

li $v0, 5 ;# read keyboard into $v0 (the second number to multiply)

syscall

move $t1, $v0 ;# second number is now in $t1, aka 'b'

move $s1, $t1

j lbl2000 ;# skip to running the program

lblFixedData:

li $t0, 2147483647 ;# try with 2^31-1 \* 2^31-1 ;# tests multiplying as many ones as possible

li $t1, 2147483647 ;# 4611686014132420609

jal lbl2000

li $t0, 0 ;# try with zero \* Zero ;# tests multiplying as many zeros as possible

li $t1, 0 ;# 0

jal lbl2000

li $t0, 1 ;# try with 1 \* (2^31-1) ;# does it come back with it's second value?

li $t1, 2147483647 ;# 2147483647

jal lbl2000

li $t0, 2147483647 ;# try with (2^31-1) \* 1 ;# does it come back with the first value?

li $t1, 1 ;# 2147483647

jal lbl2000

li $t0, 5 ;# try with other numbers ;# some random numbers tried out

li $t1, 129 ;# 645

jal lbl2000

li $t0, 1 ;# try with 1 \* 1

li $t1, 1 ;# 1

jal lbl2000

li $t0, 0 ;# try with 0 \* 1

li $t1, 1 ;# 0

jal lbl2000

li $t0, 1 ;# try with 1 \* 0

li $t1, 0 ;# 0

jal lbl2000

li $t0, 2147483647 ;# try with 2147483647 \* 0 ;# does results of zero work?

li $t1, 0 ;# 0

jal lbl2000

li $t0, 0 ;# try with 0 \* 2147483647

li $t1, 2147483647 ;# 0

jal lbl2000

li $v0, 4 ;# going to be print a string

la $a0, noteMessage ;# set to print out this

syscall

;# Note: these next few lines would shown a + -integer, due to these being displayed as

;# signed 32 bit integers instead of unsigned 32 bit integers

;# and being in the range of numbers that would represent a negative signed number

;# this is corrected for in the output with an extra +2^31+, etc...

li $t0, 2147483647 ;# try with 2147483647 \* 2

li $t1, 2 ;# 4294967294

jal lbl2000

li $t0, 2 ;# try with 2\* 2147483648 ;# two more than last one

li $t1, 2147483648 ;# 4294967296 ;# note the input number gets truncated to 2147483647, so this number gets changed to 4294967294

jal lbl2000

li $t0, 65536 ;# try with 65536 \* 65536

li $t1, 65536 ;# 4294967296

jal lbl2000

li $t0, 65535 ;# try with 65536 \* 65536 ;# no + -integer part due to this being one more than in the range of values that would give this

li $t1, 65535 ;# 4294967294 ;#fffe0001

jal lbl2000

li $v0, 4 ;# going to be print a string "\n"

la $a0, newLine ;# set to print out this

syscall

endPrgm:jr $t8 ;# $ta contains the return to address

lbl2000:

;# reset the values that get changed in calculations

move $s1, $t1 ;# the calculations put $t1 at zero, so this is to save the number so that it can be put back

li $t2, 0 ;# c, the 'most significant bits'

li $t3, 0 ;# d, the 'least significant bits'

li $t5, 0 ;# f, the temp value for these calculations

li $t4, 32 ;# e, number of bits 4\*16=32, the calculations make $t4 zero, this puts it back

lbl2010:

;# start the math...

and $t5, $t1, $t6 ;# f=band(b, amask) ;# used to get last digit of $t1(aka b) into $t5 (aka f)

beqz $t5, lbl2050 ;# if $t5 (aka f) = 0 skip next line

addu $t2, $t2, $t0 ;# c=c+a

lbl2050:

srl $t1, $t1, 1 ;# rightshift b by 1

and $t5, $t2, $t6 ;# f=band(c,amask) ;# used to get last digit of $t2(aka c) into $t5 (aka f)

beqz $t5, lbl2100 ;# if f is zero skip next line

move $t5, $t7 ;# f = cmask

lbl2100:

srl $t2, $t2, 1 ;# do the right shifting to put the values closer to the right position

srl $t3, $t3, 1

addu $t3, $t3, $t5 ;# do the addition part of the multiplication

sub $t4, $t4, $t6 ;# $t4 controls the number of loops remaining, reduce by 1

bgtz $t4, lbl2010 ;# go to the next set of digits

move $t1, $s1 ;# now put the original $t1 back for printing

printMul: ;# this specific label technically isn't needed, but makes the program easier to read

;# This part prints out two numbers

;# the first being for the 'c' register for the 'more significant values'

;# take this value and multiply by 2^32 then add

;# the second suwmelbeing the 'd' register for the 'less significant valuse'

li $v0, 1 ;# set it up to print an integer

move $a0, $t0 ;# move the integer into the printing spot

syscall ;# make the system call to print it

li $v0, 4 ;# going to be print a string " \* "

la $a0, timesSymbol ;# set to print out this

syscall

li $v0, 1 ;# set it up to print an integer

move $a0, $t1 ;# move the integer into the printing spot

syscall ;# make the system call to print it

li $v0, 4 ;# going to be print a string " = "

la $a0, equalSymbol ;# set to print out this

syscall

beqz $t2, NextValue1 ;# Don't print this if it's zero, since leading zeros can be removed from numbers without changing there value

li $v0, 1 ;# set it up to print an integer

move $a0, $t2 ;# move the integer into the printing spot

syscall ;# make the system call to print it

li $v0, 4 ;# going to be print a string "\*2^32 + "

la $a0, cMessage ;# set to print out this

syscall

NextValue1:

;# DONE: Change this to deal with printing as a signed number, otherwize it would have printed negative numbers

;# if negative then

;# display + 2^31 +

;# replace the first digit with a zero, band with 01111111111111111111111111111111

;# display the resulting number

;# ###

bgez $t3, lastValue ;# if the number dosen't start with a 1 skip this section

li $v0, 4 ;# going to be print a string " + 2^31 + "

la $a0, unSign ;# set to print out this

syscall

and $t3, $t3, $t9 ;# replaces the leading 1 with a zero

lastValue:

li $v0, 1 ;# set it up to print an integer

move $a0, $t3 ;# move the integer into the printing spot

syscall ;# make the system call to print it

li $v0, 4 ;# going to be print a string "\n"

la $a0, newLine ;# set to print out this

syscall

jr $ra ;# end the function call

.data

prompt1:

.asciiz "Enter the first number "

prompt2:

.asciiz "Enter the second number "

prompt3:

.asciiz "The product is

"

prompt4:

.asciiz "

"

timesSymbol:

.asciiz " \* "

equalSymbol:

.asciiz " = "

cMessage:

.asciiz "\*2^32 + "

unSign:

.asciiz "2^31 + "

newLine:

.asciiz "

"

noteMessage:

.asciiz "Please see notes in the program file about the plus negative stuff.

(appprox line 87)

"

Code Execution Results:

2147483647 \* 2147483647 = 1073741823\*2^32 + 1

0 \* 0 = 0

1 \* 2147483647 = 2147483647

2147483647 \* 1 = 2147483647

5 \* 129 = 645

1 \* 1 = 1

0 \* 1 = 0

1 \* 0 = 0

2147483647 \* 0 = 0

0 \* 2147483647 = 0

Please see notes in the program file about the plus negative stuff.

(appprox line 87)

2147483647 \* 2 = 2^31 + 2147483646

2 \* 2147483647 = 2^31 + 2147483646

65536 \* 65536 = 1\*2^32 + 0

65535 \* 65535 = 2^31 + 2147352577