Lab 2 Report

ECE 154A

Vishal Seenivasan

1. Hours spent: I spent around 4-5 hours coding this lab, mostly on the divide function, which I found trickier than the multiplication function.
2. mult.s:

```
##############################################################################
# File: mult.s
# Skeleton for ECE 154a project
##############################################################################

        .data
student:
        .asciz "Student"      # Place your name in the quotations in place of Student
        .globl  student
nl:     .asciz "\n"
        .globl nl



op1:    .word 100                           # change the multiplication operands
op2:    .word 1000                  # for testing.



        .text

        .globl main
main:                               # main has to be a global label
        addi    sp, sp, -4          # Move the stack pointer
        sw      ra, 0(sp)           # save the return address

        mv      t0, a0              # Store argc
        mv      t1, a1              # Store argv

# a7 = 8 read character
#  ecall

        li      a7, 4               # print_str (system call 4)
```

```
        la      a0, student           # takes the address of string as an argument
        ecall

        slti    t2, t0, 2             # check number of arguments
        bne      t2, zero, operands
        j       ready

operands:
        la      t0, op1
        lw      a0, 0(t0)
        la      t0, op2
        lw      a1, 0(t0)



ready:
        jal     multiply              # go to multiply code

        jal     print_result          # print operands to the console




                                      # Usual stuff at the end of the main
        lw      ra, 0(sp)             # restore the return address
        addi    sp, sp, 4

        li       a7, 10
        ecall


multiply:
##############################################################################
# Your code goes here.
# Should have the same functionality as running
#       mul     a2, a1, a0
# assuming a1 and a0 stores 8 bit unsigned numbers
##############################################################################
        addi a2, zero, 0 #a2 = 0
        add s1, zero, a1 #Temp multiplicand, s1 = a1
```

```
        add s2, zero, a0 #Temp multiplier, s2 = a0

        #Loop variables - t3=i, t4=32
        addi s3, zero, 0 #s3 = 0
        addi s4, zero, 32 #s4 = 0

for: bge s3 s4 done #for(s3, s3 <s4, s3++)
        #Test multiplier0
        andi s5 s2 0x0001 #s5 = s2 && 0x0001
        beq s5 zero shift #if(s5 < 0)

        add a2, a2, s1 #Add multiplicand to product, a2 = a2+s1
shift:
        slli s1, s1, 1 #Shift multiplicand left s1<<1
        srli s2, s2, 1 #Shift muliplier right, s2>>1
        addi s3, s3, 1 #Increment i, s3++
        j for
done:

###############################################################################
# Do not edit below this line
###############################################################################
        jr      ra



print_result:

# print string or integer located in a0 (code a7 = 4 for string, code a7 = 1 for integer)
        mv      t0, a0
        li      a7, 4
        la      a0, nl
        ecall

# print integer
        mv      a0, t0
        li      a7, 1
        ecall
# print string
        li      a7, 4
```

```
        la      a0, nl
        ecall

# print integer
        li      a7, 1
        mv      a0, a1
        ecall
# print string
        li      a7, 4
        la      a0, nl
        ecall

# print integer
        li      a7, 1
        mv      a0, a2
        ecall
# print string
        li      a7, 4
        la      a0, nl
        ecall

        jr      ra
```

3. div.s:

```
################################################################################
# File: div.s
# Skeleton for ECE 154a project
################################################################################

        .data
student:
        .asciz "Student"      # Place your name in the quotations in place of Student
        .globl  student
nl:     .asciz "\n"
        .globl nl



op1:    .word 0                       # divisor for testing
op2:    .word 144                     # dividend for testing



        .text

        .globl main
main:                                 # main has to be a global label
        addi    sp, sp, -4            # Move the stack pointer
        sw      ra, 0(sp)             # save the return address

        mv      t0, a0                # Store argc
        mv      t1, a1                # Store argv

        li      a7, 4                 # print_str (system call 4)
        la      a0, student           # takes the address of string as an argument
        ecall

        slti    t2, t0, 2             # check number of arguments
        bne      t2, zero, operands
        j       ready

operands:
        la      t0, op1
        lw      a0, 0(t0)
```

```
        la      t0, op2
        lw      a1, 0(t0)


ready:
        jal     divide                  # go to divide code

        jal     print_result            # print operands to the console

                                        # Usual stuff at the end of the main
        lw      ra, 0(sp)               # restore the return address
        addi    sp, sp, 4

        li       a7, 10
        ecall


divide:
###############################################################################
# Your code goes here.
# Should have the same functionality as running
#       divu    a2, a1, a0
#       remu     a3, a1, a0
# assuming a1 is unsigned divident, and a0 is unsigned divisor
###############################################################################


#Check for division by 0
        beq a0, zero, end #if(a0 == 0)

        add s0, a1, zero #Work register for dividend, s0 = a1
        add s1, a0, zero #Work register for divisor, s1 = a0
        add a2, zero, zero #s2 = 0
        add a3, zero, a1 #a3 = a1
        lui s3, 0x40000 #s3 = 0x40000000

        slli s1, s1, 8  #Shift divisor to left 8 bits, s1 << 8
```

```
	#Loop values
	add a4, zero, zero #index #a4 = 0
	addi a5, zero, 9 #limit #a5 = 9

for: bge a4 a5 end #for(a4, a4 < a5, a4++)
	sub a3, a3, s1 #Subtract divisor from remainder, a3-=s1
	blt a3 zero else #if(a3 >= 0)
	slli a2, a2, 1 #Shift quotient, a2 << 1
	addi a2, a2, 1 #Set rightmost bit to 1, a2 += 1
	j else_end
else: #else
	add a3, a3, s1 #Restore remainder #a3 += s1
	slli a2, a2, 1 #Shift quotient, a2 << 1
else_end:
	srli s1, s1, 1 #s1 >> 1
	addi a4, a4, 1 #a4++
	j for
end:
###############################################################################
# Do not edit below this line
###############################################################################
	jr	ra



# Prints a0, a1, a2, a3
print_result:
	mv	t0, a0
	li	a7, 4
	la	a0, nl
	ecall

	mv	a0, t0
	li	a7, 1
	ecall
	li	a7, 4
	la	a0, nl
	ecall

	li	a7, 1
```

```
mv      a0, a1
ecall
li      a7, 4
la      a0, nl
ecall

li      a7, 1
mv      a0, a2
ecall
li      a7, 4
la      a0, nl
ecall

li      a7, 1
mv      a0, a3
ecall
li      a7, 4
la      a0, nl
ecall

jr ra
```

4. Feedback: It would be helpful if the lab instructions highlighted the fact that we’re only building multiplication/division functions for 8 bit operands. I wasted time trying to build the division function for the default 32 bit operands.