Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Prof. Archana Chidanandan

Homework 1 - Solutions
Assembly Language and Machine Language
Max Points: 45 points

Directions

This assignment is due Friday, December 17. Submit your solutions on a separate sheet of paper. You may use SPIM and a calculator as required.

Learning Objectives

In the process of completing this homework assignment, students will develop their abilities to

- Implement algorithms involving arrays, selection and iteration abstraction in assembly language.
- Write programs with procedures.
- Use the stack to store values in procedure calls.
- Predict the actual assembly language instructions corresponding to a pseudo-instruction.
- Interpret a sequence of binary data, given the rules for conversion.

Problems

1. [10 pts] For each pseudoinstruction listed below, give a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use `$at` for some of the sequences. "`big`" refers to a 32-bit immediate value and "`small`" refers to a 16-bit immediate value.

Pseudoinstruction	Actual Instructions	Notes
move $t5, $t3	add $t5, $t3, $zero	addu is OK
clear $t5	add $t5, $zero, $zero	addu is OK
li $t5, small	addi $t5, $zero, small	ori and addiu are OK
li $t5, big	lui $t5, $big_{31:16}$ ori $t5, $t5, $big_{15:0}$	

lw $t5, big($t3)	lui $at, $big_{31:16}$
	addu $at, $at, $t3
	lw $t5, $big_{15:0}$($at)
		Credit will be given for add

addi $t5, $t3, big	lui $at, $big_{31:16}$ ori $at, $at, $big_{15:0}$ add $t5, $t3, $at	
beq $t5, small, l	addi $at, $zero, small beq $t5, $at, l	ori is OK
beq $t5, big, l	lui $at, $big_{31:16}$ ori $at, $at, $big_{15:0}$ beq $t5, $at, l	

ble $t5, $t3, l	slt $at, $t3, $t5 beq $at, $zero, l
bgt $t5, $t3, l	slt $at, $t3, $t5 bne $at, $zero, l
bge $t5, $t3, l	slt $at, $t5, $t3 beq $at, $zero, l

2. [10 pts] Write a documented MIPS assembly language program that will use a procedure "CountSmallerInt" to count the number of values in an integer array that are smaller than a given number. The procedure will accept ~~two~~ *three* input parameters: the address of the array, ~~and~~ the number to compare the elements of the array with *and the size of the array*. The procedure must return the count. In "main", write the count to memory. The array values are present in memory, while the number to compare with must be accepted as an input from the console. Prompt the user for this value with a meaningful message. *Note:* Figure A.9.1 (on the CD) has the list of syscall codes.

Please see attached hmwk1-p2.asm.

3. [10 points] Given the bit pattern below,

1000 1111 1110 1111 1100 0000 0000 0000

what does it represent, assuming that is it

a. a two's complement integer?

Since the MSB is a "1", the number is a negative number. Invert each bit, add 1 to the result and then convert the binary number to the decimal equivalent.

Answer: -1880113152

b. an unsigned integer?

All the bits contribute towards the magnitude which in decimal is 2414854144.

c. a sign-magnitude integer?

The MSB is a "1". This implies that the number is negative. Now, ignore the MSB and determine the decimal equivalent of the remaining 31 bits.

Answer: -267370496

d. a MIPS instruction?

opcode = 100011 = 35 => lw
rs = 11111 => $31($ra)
rt = 01111 => $15($t7)
16-bit signed immediate = 1100 0000 0000 0000 = -16384
Therefore, the MIPS instruction is:
lw $t7, -16384($ra)

4 a. [10 points] Complete the MIPS procedure `power` in the following pages by

filling in the provided spaces with MIPS assembly language statements such that

- The procedure returns x^y, where x and y are its parameters, and
- It follows the MIPS register usage conventions.

Assume the existence of another MIPS procedure `product` that returns the product of its two parameters.

b. [5 points] Complete the MIPS main procedure that follows procedure power by filling in the provided spaces with MIPS assembly language statements such that

- The program calls the `power` procedure to compute 3^5, and
- It follows the MIPS register usage conventions.

Read all the provided parts of the program, before you begin.

```
#
# Procedure power calculates x^y using repeated multiplication, where x
# and y are the two parameters of the procedure, and returns that value
#
#
#  In the procedure, register usage:
#
#  $t0 - x
#  $t1 - y
#  $t2 - count
#  $t3 - value

                .text
power:

#
# Procedure entrance and initialization
#
```

```
            move $t0, $a0      # $t0 = x
            move $t1, $a1      # $t1 = y
            addi  $sp, $sp, -4
            sw    $ra, 0($sp)  # Store $ra on the stack because power
                               # calls product and hence $ra will get
                               # over-written.
```

```
            addi  $t2, $zero, 1   # count = 1
            addi  $t3, $zero, 1   # value = 1

#
# For each of y iterations, set value = value * x
#

loop:
            bgt   $t2, $t1, exit1   # if (count > y) done with loop
```

```
                                                  # Call "product"
            move $a0, $t2       #$a0 = count      # to calculate
                                                  # value * x.
```

```
            move $a1, $t3       #$a1 = value
            addi $sp, $sp, -4   #Save $t2 on the stack
            sw    $t2, 0($sp)   #as it may get over-written
                                #in product
            jal   product
            lw    $t2, 0($sp)   # Restore count to $t2
            addi  $sp, $sp, 4   # Restore stack pointer
            move $t3, $v0      # $t3 = new "value" returned
                                # by "product"


            addi  $t2, $t2, 1    # count = count + 1
            j     loop


#
# Exit the procedure
#
exit1:
      move $v0, $t3     # Move the calculated value to $v0

      lw    $ra, 0($sp) # Restore the value of $ra
      addi  $sp, $sp, 4 # Restore the value of $sp

      jr  $ra

#
# Main program starts here
#
main:
      li    $s0, 3      # x
      li    $s1, 5      # y


                                                    # Call "power"
     move $a0, $s0                                  # to calculate
     move $a1, $s1                                  # x^y

      jal   power

     move $t1, $v0
     li    $v0, 10
     syscall
```