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

Homework 1 - Solutions

Assembly Language
Max Points : 30

Directions

This assignment is due Monday, December 15, 2003 for Section 1 and Tuesday, December 16, 2003 for Section 2. Submit your solutions on a separate sheet of paper.
Hint: Use SPIM.

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.
- Predict the actual assembly language instructions corresponding to a pseudoinstruction.

Problems

1. [10 pts] Problem 3.10 (page 199) from Hennessy and Patterson.

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. [8 pts] Write a documented MIPS assembly language program to calculate the product of two numbers a and b, that are stored in memory. The product must be written back to memory at label c. You may NOT use the MIPS `mult, multu, mul, mulo or muluo` instructions. You MUST use a loop structure. *Hint: 2 * 3 = 2 + 2 + 2. Also, this program does not require the use of arrays.*

The MIPS program implements the following high-level language code fragment:

```
i = 0;
C = 0;
while( i != B){
     C = C + A;
     i = i + 1;
}
```

where A and B are the inputs and C is the product of A and B.

```
          .text
          .globl main
main:
          la   $t0,  A
          lw   $t0, 0($t0)    # Read the value of "A" from
                              # memory

          la   $t1, B
          lw   $t1, 0($t1)    # Read the value of "B" from
                              # memory.

          li   $t3, 1         #  i = 0 (to keep track of
                              #  the value "B"
          li   $t4, 0         #  C = 0 (initialize the
                              #  result to 0)

loop:
          beq  $t3, $t1, done  # while (i is not equal to b
                               # ) continue adding
                               # A to C.
          add  $t4, $t4, $t0   # C= C + A;
          addi $t3, $t3, 1     # i = i + 1
          j loop

done:
          la   $t3, C
          sw   $t4, 0($t3)    # Store the value of "C" in
```

```
                                # memory

          li   $v0, 10
          syscall               # Ready to quit

          .data
A:    .word     5
B:    .word     4
C:    .word     0
```

3. [5 pts] The selection sort algorithm begins by finding the largest element of an array and exchanging it with the last element. It then finds the second largest element and exchanges it with the next to last element. It repeats this process until the array is sorted.

The following Java code implements the algorithm:

```
public static void SelectionSort( int A[], int N ){
     for( int i = N; i > 1; i-- ){
          int max = -1;
          int maxIndex = 0;
          for( int j = 0; j < i; j++ ){
               if( A[ j ] > max ){
                    max = A[ j ];
                    maxIndex = j;
               }
          }
          int temp = A[ i - 1 ];
          A[ i - 1 ] = A[ maxIndex ];
          A[ maxIndex ] = temp;
     }
   }
```

The following is a partial MIPS assembly language program to implement the above selection sort algorithm. This is, in fact, a modified version of p04-2.asm that we have already seen in class. Complete the program by supplying the code required, in the spaces provided, so that the program may implement the selection sort algorithm.

```
# This MIPS assembly language program must be completed to implement
# the selection sort algorithm.
#
# Register usage
#
# $t0 - two uses:
# 1) the address of N
# 2) the value of i
# $t1 - the constant 1
# $t2 - j (the counter)
# $t3 - unused
# $t4 - the base address of A
# $t5 - two uses:
```

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

```
# 1) the address of A[j]
# 2) the value of A[j]
# $t6 - max (the maximum known element)
# $t7 - maxIndex (the index of max)
# $t8 - flag (set to 1 if max < A[j], otherwise 0)

   .text        # Text section of the program (as opposed to data).
   .globl main  # Make MAIN globl so you can refer to it in SPIM.
   .globl loop2 # Make LOOP2 globl so you can refer to it in SPIM
main: # Program starts at MAIN.
#
# Initialization
#
         la $t0, N         # Set $t0 to the address of N
         lw $t0, 0($t0)    # Set $t0 (hereafter called i) to the
                           # value of N
         li $t1, 1         # Set $t1 to 1
         la $t4, A         # Set $t4 to the address of A[j]
loop1:   li $t2, 0   # Set $t2 (hereafter called j) to 0
         # $t5 is assigned in the loop before it is used
         li $t6, -1        # Set $t6 (hereafter called max) to -1
         # $t7 and $t8 are assigned in the loop before they are used
loop2:   beq $t2, $t0, exit2   # Continue into the loop if j < i
                               # else go to exit2

# Load the next element
         sll $t5, $t2, 2   # Set $t5 to j*4
         add $t5, $t5, $t4 # Set $t5 to address of A[j]
         lw  $t5, 0($t5)   # Set $t5 to A[j]

# Update max and maxindex if necessary
         slt $t8, $t6, $t5 # Set flag to 1 if max < A[j], and 0
                           # otherwise
         beq $t8, $0, ok   # Skip update if flag is 0
         add $t6, $t5, $0  # Set max to A[j]
         add $t7, $t2, $0  # Set maxIndex to j
ok:      add $t2, $t2, $t1 # Increment j
         j loop2           # Continue loop


exit2:         # Swap A[maxIndex] and A[i-1];
               # Decrement i and continue to loop1
               # if i > 1 else prepare to exit
```

```
        sll $t7, $t7, 2         # Set $t7 to maxindex*4
        add $t7, $t7, $t4       # Set $t7 to address of
                                # A[maxindex]
        sw $t5, 0($t7)          # Store A[j] in
                                # A[maxindex]
        addi $t0, $t0, -1       # Decrement j
        sll $t5, $t0, 2         # Set $t5 to j*4
        add $t5, $t5, $t4       # Set $t5 to address of
                                # A[j]
        sw $t6, 0($t5)          # Store max in A[j]
        bne $t0, $t1, loop1     # Continue loop if j > 1


        li $v0, 10          # Prepare to exit
        syscall             # ... Exit.

        .data               # Data section of the program.
A:      .word 32, 64, 49, 80, 8
N:      .word 5
```

4. [7 pts] Write a documented MIPS assembly language program that will count the number of occurrences of a number in an array of integers and write the count to memory. The array and the number to be counted are values stored in memory.

The MIPS program will implement the following high level language fragment.

```
numToCount = 5;
count = 0;
sizeOfArray = 6;
i = 0;
while ( i < 6){
        if (A[i] == numToCount) {
                count++;
        }
        i++;
}
```

```
        .text
        .globl main
main:
        la    $t0, NUM
        lw    $t0, 0($t0) # Read number to count from memory

        la    $t1, SIZE
        lw    $t1, 0($t1) # Read size of array from memory.

        li    $t2, 0      # count = 0
        li    $t3, 0      # i = 0
```

```
            la    $t4, A            # Determine address of first
                                    # element in array

loop:
            beq   $t3, $t1, done    # Continue while (i != SIZE)

            # Read A[i] from memory
            sll   $t5, $t3, 2       #  i   * 4 (determine byte value of
                                    # index)
            add   $t6, $t4, $t5     # Determine address of A[i]
            lw    $t7, 0($t6)       # Read A[i] from memory

            bne   $t7, $t0, next    # If (A[i] == NUM), continue
            addi  $t2, $t2, 1       # count = count + 1
next:
            addi  $t3, $t3, 1       # i = i + 1
            j loop
done:
            la    $t0, COUNT
            sw    $t2, 0($t0)       # Write COUNT to memory

            li    $v0, 10
            syscall                 # Get ready to exit

            .data
NUM:        .word       5
SIZE:       .word       6
COUNT:      .word       0
A:          .word       3     5     5     5     6     5
```