Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Laurence Merkle and Archana Chidanandan

Homework 1 Solutions (Assembly Language)

Directions

This assignment is due Thursday, September 18, 2003 for Sections 1 and 3 and Friday, September 19, 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

- Predict the actual assembly language instructions corresponding to a pseudoinstruction.
- Implement algorithms involving arrays, selection, iteration, and procedural abstraction in assembly language.

Problems

1. [10 pts] Hennessy and Patterson Problem 3.10.

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`	

Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Laurence Merkle and Archana Chidanandan

2. [10 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 list[], int n ){

     for( int i = n; i > 1; i-- ){
          int m = list[ 0 ];
          int k = 0;

          for( int j = 1; j < i; j++ ){
               if( list[ j ] > m ){
                    m = list[ j ];
                    k = j;
               }
          }

          int temp = list[ i - 1 ];
          list[ i - 1 ] = list[ k ];
          list[ k ] = temp;
     }
}
```

Using P03-2.asm as a starting point, write a MIPS program that implements the selection sort algorithm. Be sure to document your program properly. Turn in a hard copy listing of your program.

```
# File:        selection_sort.asm
# Written by:  Larry Merkle, Jan. 15, 2003
#
# This file contains a MIPS assembly language program that implements the
# selection sort algorithm.
#
# Register usage
#
# $t0 - two uses:
#       1) the address of N
#       2) the value of j
# $t1 - the constant 1
# $t2 - i (the counter)
# $t3 - unused
# $t4 - the base address of A
# $t5 - two uses:
#       1) the address of A[i]
#       2) the value of A[i]
# $t6 - max (the maximum known element)
# $t7 - maxindex (the index of max)
# $t8 - flag (set to 1 if max < A[i], otherwise 0)

      .text             # Text section of the program (as opposed to data).
```

Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Laurence Merkle and Archana Chidanandan

```
      .globl main       # Make MAIN 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 j) to the value of N
      li    $t1, 1            # Set $t1 to 1
      la    $t4, A            # Set $t4 to the address of A[i]

loop1:
      li    $t2, 0            # Set $t2 (hereafter called i) 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 loop if i < j

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

      # Update max and maxindex if necessary
      slt   $t8, $t6, $t5     # Set flag to 1 if max < A[i], and 0 otherwise
      beq   $t8, $0, ok       # Skip update if flag is 0
      add   $t6, $t5, $0      # Set max to A[i]
      add   $t7, $t2, $0      # Set maxindex to i

ok:
      add   $t2, $t2, $t1     # Increment i
      j     loop2             # Continue loop

exit2:
      # Swap A[maxindex] and A[j]
      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, 16, 64, 80, 48
N:    .word 5
```

Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Laurence Merkle and Archana Chidanandan

3. [10 pts] Write a procedure *find* in MIPS assembly language, adhering to MIPS register usage conventions. The procedure should take ~~two~~ **three** arguments. The first argument is a pointer to an array of integers~~ and the second argument is an integer value~~, **the second is the number of elements in the array, and the third is an integer value**. *find* must find and return for the ~~second~~ **third** argument, the index value of its first occurrence in the integer array. If the integer value does not exist in the array, then *find* must return -1.

```
# File:        find.asm
# Written by:  Larry Merkle, Sept. 22, 2003
#
# This file contains a MIPS assembly language program that calls procedure
find,
# which returns the index of the first occurrence of a value X in an N-
element array A,
# or -1 if X does not occur in A.  The result is stored in Index.
#
# Register usage
#
# $s0 - the base address of A
# $s1 - two uses:
#       1) the address of N
#       2) the value of N
# $s2 - two uses:
#       1) the address of X
#       2) the value of X
# $s3 - the address of Index

      .text             # Text section of the program (as opposed to data).
      .globl main       # Make MAIN globl so you can refer to it in SPIM.

main:                   # Program starts at MAIN.
#
# Initialization
#

      la    $s0, A            # Set $s0 to the base address of A
      la    $s1, N            # Set $s1 to the address of N
      lw    $s1, 0($s1)       # Set $s1 to the value of N
      la    $s2, X            # Set $s2 to the address of X
      lw    $s2, 0($s2)       # Set $s2 to the value of X
      la    $s3, Index        # Set $s3 to the address of Index

#
# Procedure call
#

      move  $a0, $s0          # Copy value of first parameter
      move  $a1, $s1          # Copy value of second parameter
      move  $a2, $s2          # Copy value of third parameter
      jal   find              # Call procedure
      move  $t0, $v0          # Copy return value

      sw    $t0, 0($s3)       # Store return value in Index

      li    $v0, 10           # Prepare to exit
```

```
      syscall                 #   ... Exit.


# Procedure find returns the index of the first occurrence of a value X in an
N-element
# array A, or -1 if X does not occur in A.
#
# "Public" register usage:
#
# $a0 - the base address of A (the array being searched)
# $a1 - the value of N (the number of elements in A)
# $a2 - the value of X (the value sought)
# $v0 - the index of the first occurrence of X in A
#
# "Private" register usage:
#

# $t0 - the base address of A (the array being searched)
# $t1 - the value of N (the number of elements in A)
# $t2 - the value of X (the value sought)
# $t3 - the index of the first occurrence of X in A
# $t4 - i, the index of the element being compared to X
# $t5 - two uses:
#       1) the address of A[i]
#       2) the value of A[i]
#

find:

#
# Copy values of parameters out of $a registers
#
      move  $t0, $a0
      move  $t1, $a1
      move  $t2, $a2

#
# Initialization
#

      li    $t3, -1           # -1 is the default return value
      li    $t4, 0            # Start searching at index i = 0

#
# Search the array
#

loop:
      beq   $t4, $t1, exit    # Continue the loop if i < N

# Load the next element

      sll   $t5, $t4, 2       # Set $t5 to i*4
      add   $t5, $t5, $t0     # Set $t5 to address of A[i]
      lw    $t5, 0($t5)       # Set $t5 to A[i]

# Compare this element to X
```

```
      beq   $t5, $t2, found   # Exit loop if A[i] = X

# Continue loop if A[i] != x

      addi  $t4, $t4, 1       # Increment i
      j     loop              # Continue loop

#
# Successful search -- assign return value
#

found:

      move  $t3, $t4    # Set $t3 to i

#
# Exit procedure
#

exit:

      move  $v0, $t3    # Copy return value into $v0
      jr    $ra         # Return from procedure

      .data             # Data section of the program.

A:    .word 32, 16, 64, 80, 48
N:    .word 5
X:    .word -3
Index:      .word -2
```