CSSE 232 – Computer Architecture I
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Homework 1 Solutions

1. (18 points)

Pseudoinstruction	Actual Instructions
`move $t5, $t3`	`add $t5, $t3, $zero`
`clear $t5`	`add $t5, $zero, $zero`
`li $t5, big`	`lui $t5, upper(big)` `ori $t5, $t5, lower(big)`
`li $t5, small`	`addi $t5, $zero, small`
`beq $t5, small, L`	`addi $at, $zero, small` `beq $t5, $at, L`
`ble $t5, $t3, L`	`slt $at, $t3, $t5` `beq $at, $zero, L`

2. (7 points)

```
find:   li      $v0, -1
        li      $t0, 0
        beq     $t0, $a1, exit
loop:   sll     $t1, $t0, 2
        add     $t1, $t1, $a0
        lw      $t1, 0($t1)
        beq     $t1, $a2, found
        add     $t0, $t0, 1
        bne     $t0, $a1, loop
        j       exit

found:  move    $v0, $t0

exit:   jr      $ra
```

3. (15 points)

```
# File:         hmwk3-1.asm
# Written by:   Larry Merkle, Dec. 10, 2002
#
# Modified by: Archana Chidanandan, Dec. 11, 2003
#               a. Name changed from "Move" to "MoveObject"
```

```
#
# Modified by: Archana Chidanandan, Sept. 9, 2004
#               a. Upadated descriptions of the procedures.
#
# Given a current position component and a velocity component, the
# MIPS procedure MoveObject calculates a new position component.  It
# assumes periodic boundaries at positions zero and eleven in both
# directions, so a ball that goes off of one edge comes back at the
# opposite edge.
#
###############################################################################
#
# NOTE:  The procedure adheres STRICTLY to the MIPS register usage conventions.
#
###############################################################################
#
#
#
# Register usage in MoveObject:
#
# $a0 - Position component
# $a1 - Velocity component
# $v0 - Updated position component
#


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

main:
#-------------------------------------------------------------------------
# Initialize the x and y positions and velocities of the "ball".
# Call MoveBall repeatedly (infinite loop) to update the x and y
# positions of the "ball".
#-------------------------------------------------------------------------

        sub     $sp, $sp, 32    # Create a 8-word frame.
        sw      $ra, 4($sp)     # Save $ra
        sw      $s0, 8($sp)     # Save $s0
        sw      $s1, 12($sp)    # Save $s1
        sw      $s2, 16($sp)    # Save $s2
        sw      $s3, 20($sp)    # Save $s3

        li      $s0, 1          # initialize x position
        li      $s1, 2          # initialize x velocity
        li      $s2, 1          # initialize y position
```

```
        li      $s3, -3         # initialize y velocity

loop:   move    $a0, $s0        # get x position
        move    $a1, $s1        # get x velocity
        move    $a2, $s2        # get y position
        move    $a3, $s3        # get y velocity
        jal     MoveBall
        j       loop

        lw      $s3, 20($sp)    # Restore $s3
        lw      $s2, 16($sp)    # Restore $s2
        lw      $s1, 12($sp)    # Restore $s1
        lw      $s0, 8($sp)     # Restore $s0
        lw      $ra, 4($sp)     # Restore $ra
        add     $sp, $sp, 32    # Undo the 8-word frame.
        jr      $ra             # Return


MoveBall:
#-------------------------------------------------------------------------------
# Call MoveObject twice. First, to update the x position of the "ball".
# Second, to update the y position of the "ball".
#-------------------------------------------------------------------------------

        sub     $sp, $sp, 16    # Create a 4-word frame.
        sw      $ra, 4($sp)     # Save $ra
        sw      $s0, 8($sp)     # Save $s0
        sw      $s1, 12($sp)    # Save $s1

        move    $s0, $a2
        move    $s1, $a3
        jal     MoveObject

        move    $a0, $s0
        move    $a1, $s1
        move    $s0, $v0
        jal     MoveObject

        move    $v1, $v0
        move    $v0, $s0

        lw      $s1, 12($sp)    # Restore $s1
        lw      $s0, 8($sp)     # Restore $s0
        lw      $ra, 4($sp)     # Restore $ra
        add     $sp, $sp, 16    # Undo the 4-word frame.
```

```
        jr      $ra             # Return


MoveObject:

# Note: You may not modify this procedure.

#-----------------------------------------------------------------------------
#
# Given a current position component and a velocity component, this
# MIPS procedure calculates a new position component.  It assumes
# periodic boundaries at positions zero and eleven in both directions,
# so a ball that goes off of one edge comes back at the opposite edge.
#
#-----------------------------------------------------------------------------
        add     $t0, $a0, $0    # position component
        add     $t1, $a1, $0    # velocity component

        add     $t0, $t0, $t1   # update component ignoring walls
                                # i.e. add the velocity to the position

cklb:
        slt     $t2, $t0, $0    # set flag if component is less than zero
        beq     $t2, $0, ckub   # if flag is clear, ball didn't "cross" wall
                                # at zero
        addi    $t0, $t0, 11    # bring ball back in the other side
        j       cklb            # a fast ball could "cross" more than once

ckub:
        slti    $t2, $t0, 11    # set flag if component is not at least eleven
        bne     $t2, $0, exit   # if flag is set, ball didn't "cross" wall
                                # at eleven
        addi    $t0, $t0, -11   # bring ball back in the other side
        j       ckub            # a fast ball could "cross" more than once

exit:

        move    $v0, $t0        # set return value

        # Wipe out other non-preserved registers for the fun of it
        li      $t0, -1
        li      $t1, -1
        li      $t2, -1
        li      $t3, -1
        li      $t4, -1
```

```
        li      $t5, -1
        li      $t6, -1
        li      $t7, -1
        li      $t8, -1
        li      $t9, -1
        li      $a0, -1
        li      $a1, -1
        li      $a2, -1
        li      $a3, -1
        li      $v1, -1

        jr      $ra             # return to the calling procedure
```