Assignment #3 Key

(max = 90)

Read the rest of chapter 2 (starting at page 103) in the *Computer Organization and Design* text, including sections 2.15 and 2.21 which appear as **section_2.15.pdf** and **section_2.21.pdf** under Course Materials. I have provided an extensive set of notes ("Notes for Assignment #3) on this reading that can be found under Course Notes. Please refer to these notes as you carefully work through the assigned reading.

Afterwards, submit answers for the following problems:

1. Do Exercise 2.34 on page 171 in the text [CORRECTION: the function declaration for func is "int func(int a, int b);"]. (10 points)

```
f:      addi    $sp, $sp, -12   (here is the most straightforward solution)
        sw      $ra, 8($sp)
        sw      $s1, 4($sp)
        sw      $s0, 0($sp)
        move    $s0, $a2
        move    $s1, $a3
        jal     func
        move    $a0, $v0
        add     $a1, $s0, $s1
        jal     func
        lw      $ra, 8($sp)
        lw      $s1, 4($sp)
        lw      $s0, 0($sp)
        addi    $sp, $sp, 12
        jr      $ra
```

or

```
f:      addi    $sp, $sp, -8   (shorter, but not as straightforward)
        sw      $ra, 4($sp)
        add     $t0, $a2, $a3
        sw      $t0, 0($sp)
        jal     func
        move    $a0, $v0
        lw      $a1, 0($sp)
        jal     func
        lw      $ra, 4($sp)
        addi    $sp, $sp, 8
        jr      $ra
```

2. Recall that you represent hex constants in MIPS code by placing 0x in front of the constant. Do Exercise 2.39 on page 172 in the text. Use hex constants. (2 points)

```
        lui     $t1, 0x2001
        ori     $t1, $t1, 0x4924
```

3. Do Exercise 2.40 on page 172 in the text. This is a simple yes/no answer. Note the word "jump". (1 point)

No. (top 4 bits of destination address would have to be 0000)

4. Do Exercise 2.42 on page 173 in the text. This is a simple yes/no answer. Note the word "branch". (1 point)

   Yes. Range is $0x1FFFF000 \pm 2^{15}$ words = $0x1FFFF000 \pm 2^{17}$
   = 0x1FFFF000 ± 0x20000 = [0x2001F000,0x1FFDF000]
   (address 0x20014924 falls in this interval)

5. Assume that the address of a shared variable, shvar, is in $a1. Also, assume that we would like to place the value of $a2 into the shared variable if it is greater than the current value of the shared variable. Use ll/sc to perform this atomic update. Be sure that you read the course notes before attempting this problem (6 points)

```
try:    ll      $t0, 0($a1)
        slt     $t1, $t0, $a2
        beqz    $t1, skip
        move    $t0, $a2
        sc      $t0, 0($a1)
        beqz    $t0, try
skip:
```

6. Create a file, **test.s**, that has the following 7 lines of code: (14 points)

```
strcpy: lb   $t1, 0($a1)
        sb   $t1, 0($a0)
        beq  $t1, $zero, L1
        addi $a0, $a0, 1
        addi $a1, $a1, 1
        j    strcpy
L1:     jr   $ra
```

   Load this code into QtSpim (don't try to execute it; there is no main program!). Notice that the code begins at location 00400024 in memory. Create a table similar to that on page 115 for these 7 lines of code. Show everything in decimal (including locations). Notice that the constant field in the 6th instruction is exactly 1/4th of the location address of the 1st instruction; this confirms that the jump instruction contains the word address, not the byte address, of the target instruction. NOTE: Your answer for this problem should be as requested above; don't show the change that takes place when making the temporary modifications to the settings described below.

4194340	32	5	9	0		
4194344	40	4	9	0		
4194348	4	9	0	4		
4194352	8	4	4	1		
4194356	8	5	5	1		
4194360	2	1048585				
4194364	0	31	0	0	0	8

Assuming that you carefully read the discussion of PC-relative addressing in the study notes for this assignment, you should have noticed that the constant field for the 3rd instruction is off by one; it should be one less! To see the corrected machine code, check the "Enable Delayed Branches" box inside the MIPS tab in the "Settings" option under "Simulator". Now reinitialize and load the code for **test.s**. The constant field for that 3rd instruction should now be one less (in your table, show the original value … the larger one).

THIS IS IMPORTANT: You need to immediately (right now, so you won't forget!) uncheck the "Enable Delayed Branches" box. Otherwise, QtSpim will not simulate your code properly. To make sure that you have things back to normal, reinitialize and load **test.s** one more time and verify that the constant field for the 3rd instruction is back at its "one too big" state.

7. A long time ago, a friend of mine showed me a proof having to do with what he called "Polish" sequences. To generate a Polish sequence, you start with any positive integer. You square the individual digits of this number and add those squares, producing the next number in the sequence. This procedure repeats (theoretically) forever.

   Here are some Polish sequences generated for a few positive integers:

   308, 73, 58, 89, 145, 42, 20, 4, 16, 37, 58, …
   5121, 31, 10, 1, 1, …
   27, 53, 34, 25, 29, 85, 89, 145, 42, 20, 4, 16, 37, 58, 89, …
   12345678, 204, 20, 4, 16, 37, 58, 89, 145, 42, 20, …

   It can actually be proven that the Polish sequence for an arbitrary positive integer will always terminate in the infinite sequence

   1, 1, 1, …

   or in the following repeating sequence of eight integers

   20, 4, 16, 37, 58, 89, 145, 42, 20, …

   Of course, the repeating sequence would not have to begin with 20 … it could begin with any of the 8 numbers that appear in that sequence.
   You are to write a program that will allow us to test this claim. Below, I am starting you out with a little driver program (also included under Course Materials as **Polish.s**). I want you to add two procedures: (1) one called "terms" (called by the main program) whose job is to display the first 16 terms in a sequence that starts with the value that it is passed (in $a0) and (2) a function called "Polish" that will be called by the procedure "terms" … this function will actually calculate and return (in $v0) the next term in a sequence, given that the current term is the value that is passed to it (in $a0). Remember that since you have

procedures calling other procedures, you need to be concerned with preserving and restoring certain values on the stack.

Here is the little driver program:

```
# Your name/date
# Appropriate documentation

# insert your terms procedure and your Polish function here

# Driver program provided by Stephen P. Leach -- written 11/12/17

main:  la     $a0, intro    # print intro
       li     $v0, 4
       syscall

loop:  la     $a0, req      # request value of n
       li     $v0, 4
       syscall

       li     $v0, 5        # read value of n
       syscall

       ble    $v0, $0, out  # if n is not positive, exit

       move   $a0, $v0      # set parameter for terms procedure

       jal    terms         # call terms procedure

       j      loop          # branch back for next value of n

out:   la     $a0, adios    # display closing
       li     $v0, 4
       syscall

       li     $v0, 10       # exit from the program
       syscall

       .data
intro: .asciiz  "Welcome to the Polish sequence tester!"
req:   .asciiz  "\nEnter an integer (zero or negative to exit): "
adios: .asciiz  "Come back soon!\n"
```

Start with this code and add your procedures. You may want to add a few items in the data segment (to assist the terms procedure in producing the appropriate output), but don't change the portion of the code that I am providing. Don't forget to document your code! We haven't officially covered some of the needed arithmetic operations, but the following should be enough information to get you going (the last two are actually pseudoinstructions):

```
       mul    $t1, $t2, $t3 # $t1 = $t2 * $t3
       div    $t1, $t2, $t3 # $t1 = $t2 / $t3 (truncated)
       rem    $t1, $t2, $t3 # $t1 = $t2 % $t3 (remainder from $t2 / $t3)
```

Feel free to use other instructions and pseudoinstructions that appear in Appendix A. Submit a separate file called **Polish.s** as well as placing your code in this assignment submission; the Mentor will clarify what I mean by this. (56 points)

Here is a sample execution from my solution:

```
        Welcome to the Polish sequence tester!
        Enter an integer (zero or negative to exit): 308
        First 16 terms: 308 73 58 89 145 42 20 4 16 37 58 89 145 42 20 4
        Enter an integer (zero or negative to exit): 5121
        First 16 terms: 5121 31 10 1 1 1 1 1 1 1 1 1 1 1 1 1
        Enter an integer (zero or negative to exit): 27
        First 16 terms: 27 53 34 25 29 85 89 145 42 20 4 16 37 58 89 145
        Enter an integer (zero or negative to exit): 12345678
        First 16 terms: 12345678 204 20 4 16 37 58 89 145 42 20 4 16 37 58 89
        Enter an integer (zero or negative to exit): 0
        Come back soon!
```

```
# Stephen P. Leach -- 11/12/17
# Polish.asm - functions that produce the fist 16 terms of the
#       Polish sequence starting with a given positive integer,
#       along with a simple driver program to test the code.


# Procedure terms --- written by Stephen P. Leach -- 11/12/17
#       Computes and prints the first 16 numbers in the Polish
#       sequence for a given starting number.  The actual
#       generation of a term is done through calling another
#       function, Polish.
# Register use:
#       $a0     integer parameter from calling routine, for
#               function Polish and for syscall
#       $v0     integer parameter for syscall and return
#               value from function Polish
#       $s0     saved value of current term
#       $s1     number of terms produced

terms:  addi    $sp, $sp, -12  # save $ra, $s0 and $s1 on stack
        sw      $ra, 8($sp)
        sw      $s1, 4($sp)
        sw      $s0, 0($sp)

        move    $s0, $a0       # save current term in $s0
        li      $s1, 16        # number of terms to be produced

        la      $a0, text      # print beginning text for sequence
        li      $v0, 4
        syscall

loop1:  move    $a0, $s0       # print current term
        li      $v0, 1
        syscall

        addi    $s1, $s1, -1   # decrement term counter
        beq     $s1, $0, exit  # and exit, if done

        la      $a0, blank     # otherwise, print blank after term,
        li      $v0, 4
        syscall

        move    $a0, $s0       # compute next term
        jal     Polish         # by calling the Polish function

        move    $s0, $v0       # return value becomes current term

        j       loop1          # and continue loop

exit:   lw      $s0, 0($sp)    # restore values from stack
        lw      $s1, 4($sp)
        lw      $ra, 8($sp)
        addi    $sp, $sp, 12
```

```
        jr      $ra             # return to calling routine


# function Polish -- written by Stephen P. Leach -- 11/12/17
#       Given the current term (in $a0), this function computes
#       and returns (in $v0) the next term in the Polish
#       sequence (sum of the squares of the digits).
# Register use:
#       $a0     value of the current term
#       $v0     return value (next term)
#       $t0     the value 10
#       $t1     temporary calculations

Polish: move    $v0, $zero      # initialize next term to 0
        li      $t0, 10         # and $t0 to 10

loop2:  rem     $t1, $a0, $t0   # $t1 is units digit of current term
        mul     $t1, $t1, $t1   # square that value
        add     $v0, $v0, $t1   # add this value to next term

        div     $a0, $a0, $t0   # discard units digit of current term

        bne     $a0, $0, loop2 # continue if non-zero digits remain

        jr      $ra             # return to calling routine


# Driver program provided by Stephen P. Leach -- written 11/12/17

main:   la      $a0, intro      # print intro
        li      $v0, 4
        syscall

loop:   la      $a0, req        # request value of n
        li      $v0, 4
        syscall

        li      $v0, 5          # read value of n
        syscall

        ble     $v0, $0, out    # if n is not positive, exit

        move    $a0, $v0        # set parameter for terms procedure

        jal     terms           # call terms procedure

        j       loop            # branch back for next value of n

out:    la      $a0, adios      # display closing
        li      $v0, 4
        syscall

        li      $v0, 10         # exit from the program
        syscall

        .data
intro:  .asciiz "Welcome to the Polish sequence tester!"
req:    .asciiz "\nEnter an integer (zero or negative to exit): "
adios:  .asciiz "Come back soon!\n"

text:   .asciiz "First 16 terms: "
blank:  .asciiz " "
cr:     .asciiz "\n"
```

Your assignment is due by 11:59 PM (Eastern Time) on the assignment due date (consult Course Calendar on course website).