Assignment #2 Key

(max = 90)

Read pages 62 through 102 of the *Computer Organization and Design* text (we will come back to the remainder of Chapter 1 in a later assignment). I have provided an extensive set of notes ("Notes for Assignment #2) 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. This is an adaptation of Exercise 2.3 from page 165 in the textbook. Assume that variables f, g, h, i and j are assigned to registers $s0, $s1, $s2, $s3 and $s4, respectively. Also, assume that the base address of the arrays A and B are in registers $s6 and $s7. For each of the following C statements, what is the corresponding MIPS code? (10 points total)

   a. f = g – A[4]; (2 points)

   ```
   lw    $s0, 16($s6)      [obviously, there are acceptable variations]
   sub   $s0, $s1, $s0
   ```

   b. f = B[i]; (3 points)

   ```
   sll   $s0, $s3, 2       [ditto]
   add   $s0, $s7, $s0
   lw    $s0, 0($s0)
   ```

   c. B[8] = A[i-j]; (5 points)

   ```
   sub   $t0, $s3, $s4     [ditto]
   sll   $t0, $t0, 2
   add   $t0, $t0, $s6
   lw    $t0, 0($t0)
   sw    $t0, 32($s7)
   ```

2. What is the decimal value of the following? (4 points total … 2 points each)

   a. 1010111_2 (show calculation)

   $2^6 + 2^4 + 2^2 + 2^1 + 2^0 = 64 + 16 + 4 + 2 + 1 = 87_{10}$

   b. $ab9_{16}$ (show calculation)

   $10 * 16^2 + 11 * 16^1 + 9 * 16^0 = 2560 + 176 + 9 = 2745_{10}$ or

   $1010\ 1011\ 1001_2 = 2^{11} + 2^9 + 2^7 + 2^5 + 2^4 + 2^3 + 2^0$

   $= 2048 + 512 + 128 + 32 + 16 + 8 + 1 = 2745_{10}$

3. Given the 32-bit binary number: (detailed calculations not required)

   $1000\ 1001\ 1000\ 1001\ 0000\ 1011\ 0000\ 0011_2$ (5 points total)

   a. Write this number in hexadecimal. (1 point)

      $89890b03_{16}$

   b. As an unsigned integer, what is this number's decimal value? (1 point)

      2307459843_{10}

   c. As a two's complement integer, what is the decimal value? Explain process, but detailed calculations not required. (3 points)

      Number is negative.
      One's complement: 0111 0110 0111 0110 1111 0100 1111 1100
      Add 1: + 1
      0111 0110 0111 0110 1111 0100 1111 1101
      Value of this number: 1987507453
      Value of original number: -1987507453_{10}

4. Given the following hexadecimal number: (detailed calculations not required)

   $abcd1234_{16}$ (5 points total)

   a. Express this number in binary. (1 point)

      $1010\ 1011\ 1100\ 1101\ 0001\ 0010\ 0011\ 0100_2$

   b. As an unsigned integer, what is this number's decimal value? (1 point)

      2882343476_{10}

   c. As a two's complement integer, what is the decimal value? Explain process, but detailed calculations not required. (3 points)

      Number is negative.
      One's complement: 0101 0100 0011 0010 1110 1101 1100 1011
      Add 1: + 1
      0101 0100 0011 0010 1110 1101 1100 1100
      Value of this number: 1412623820
      Value of original number: -1412623820_{10}

5. Changing "44" to "4", do Exercise 2.19.1 from page 168 in the text. (3 points)

After instruction 1, we have
$t2 = AAAAAAA0$_{16}$ = 1010 1010 1010 1010 1010 1010 1010 0000$_2$.
Then "or" with $t1 = 0001 0010 0011 0100 0101 0110 0111 1000$_2$.
This gives $t2=1011 1010 1011 1110 1111 1110 1111 1000$_2$ = BABEFEF8$_{16}$.

6. Do Exercise 2.19.3 from page 168 in the text. (3 points)

After instruction 1, $t2 = 0001 0101 0101 0101 0101 0101 0101 0101$_2$.
Then do an "and" with 0000 0000 0000 0000 1111 1111 1110 1111$_2$.
This gives $t2 = 0000 0000 0000 0000 0101 0101 0100 0101$_2$ = 00005545$_{16}$.

7. For each of the following instructions, indicate the content of each instruction field (in decimal), the binary representation of the instruction in memory (show as 8 groups of 4 bits) and the equivalent hexadecimal representation of the instruction: (10 points)

```
sw      $t9, 60($sp)
```

opcode = 43, rs = 29, rt = 25, immediate = 60 (in decimal)
opcode = 101011, rs = 11101, rt = 11001,
immediate = 0000 0000 0011 1100 (in binary)
putting bits together and grouping by 4 gives us
1010 1111 1011 1001 0000 0000 0011 1100 = afb9 003c$_{16}$

```
add     $a2, $s6, $fp
```

opcode = 0, rs = 22, rt = 30, rd = 6, shamt = 0 (unused), funct = 32 (in decimal)
opcode = 000000, rs = 10110, rt = 11110, rd = 00110, shamt = 00000,
funct = 100000 (in binary)
putting bits together and grouping by 4 gives us
0000 0010 1101 1110 0011 0000 0010 0000 = 02de 3020$_{16}$

8. Recall that I have placed the **leaf_example.s** driver program from the "Notes on Assignment #2" document under Course Materials. Download that file onto your machine, get into QtSpim and load the driver program.

   a. The first line in the actual leaf_example procedure is

```
addi    $sp, $sp, -4
```

   At what address is that instruction stored? What is the hexadecimal representation of the instruction? Is this an I-format or R-format instruction? What is the value (in binary) of the constant field? (4 points)

   The instruction is stored at address 00400024$_{16}$.
   The instruction in Hexadecimal is 23bdfffc$_{16}$.

This is an I-format instruction.
The constant field (in binary) is $1111\ 1111\ 1111\ 1100_2$.

b. Run the program using the values 19, 64, -23 and 72 (in that order). What is the value returned by leaf_example? (1 point)

The value returned by leaf_example is 34 (decimal)

c. Even though the program has completed, the value of i (-23) should still be in register $a2. How is that value represented in hexadecimal? What is the binary equivalent of that value? (2 points)

$a2 = $ffffffe9_{16}$
$a2 = $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 1001_2$

d. Reinitialize and load the **leaf_example.s** code. Sometimes it is convenient to inspect the registers at a particular place in the execution of the code. Notice that the instruction immediately after the "jal leaf_example" instruction is "move $s0, $v0" (move is actually a pseudo-instruction; it is replaced by an "addu" instruction). This instruction is stored at address 0x004000c0. Also notice that the value returned by the leaf_example procedure should be in register $v0 when this instruction is about to execute. We can check this out by setting a breakpoint in the code. You can set a breakpoint by right-clicking on the line of code and selecting "set breakpoint". Now execute the program, using the values 10, 20, 30 and 40 (in that order). The program will pause when it is about to execute the instruction at your breakpoint. Select the "Abort" option. What value is in $v0 at this point? What values are in $a0 through $a3? (5 points)

$v0 = $fffffd8_{16}$
$a0 = a_{16}
$a1 = 14_{16}
$a2 = $1e_{16}$
$a3 = 28_{16}

9. Recall that I have placed the code for the fact function from the "Notes on Assignment #2" document under Course Materials (as **fact.s**). Download that file onto your machine. Starting with that code, add a driver program that will allow you to test the factorial function. Here is a sample execution of my program:

```
Welcome to the factorial tester!
Enter a value for n (or a negative value to exit): 0
0! is 1
Enter a value for n (or a negative value to exit): 6
6! is 720
Enter a value for n (or a negative value to exit): 9
9! is 362880
Enter a value for n (or a negative value to exit): 11
```

```
11! is 39916800
Enter a value for n (or a negative value to exit): 20
20! is -2102132736
Enter a value for n (or a negative value to exit): 100
100! is 0
Enter a value for n (or a negative value to exit): 3
3! is 6
Enter a value for n (or a negative value to exit): -1
Come back soon!
```

Your program should produce output that mirrors mine. Be sure to document your code (as I did in **leaf_example.s**). Submit a separate file called **fact.s** as well as placing your code in this assignment submission; the Mentor will clarify what I mean by this.

```
# Stephen P. Leach -- 10/07/15
# fact key.s - a simple driver program that tests the recursive
#      factorial function
# Register use:
#      $a0    parameter for fact and syscall
#      $v0    syscall parameter and fact return value
#      $t0    temporary calculation
#      $s0    saved value of n
#      $s1    saved value of fact(n)

fact:  slti   $t0, $a0, 1         # test for n < 1
       beq    $t0, $zero, L1      # if n >= 1, go to L1

       li     $v0, 1        # return 1
       jr     $ra           # return to instruction after jal

L1:    addi   $sp, $sp, -8  # adjust stack for 2 items
       sw     $ra, 4($sp)   # save the return address
       sw     $a0, 0($sp)   # save the argument n

       addi   $a0, $a0, -1  # n >= 1; argument gets (n - 1)
       jal    fact          # call fact with (n - 1)

       lw     $a0, 0($sp)   # return from jal: restore argument n
       lw     $ra, 4($sp)   # restore the return address
       addi   $sp, $sp, 8   # adjust stack pointer to pop 2 items

       mul    $v0, $a0, $v0 # return n * fact (n - 1)

       jr     $ra           # return to the caller

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

       slt    $t0, $v0, $zero     # if n < 0, exit
       bne    $t0, $zero, out
```

```
        move   $s0, $v0      # save value of n

        move   $a0, $v0      # place value of n in $a0

        jal    fact          # call fact function

        move   $s1, $v0      # save value returned by fact

        move   $a0, $s0      # display result
        li     $v0, 1        # print value of n
        syscall

        la     $a0, ans      # print text part of answer
        li     $v0, 4
        syscall

        move   $a0, $s1      # print fact(n)
        li     $v0, 1
        syscall

        la     $a0, cr       # print carriage return
        li     $v0, 4
        syscall

        j      loop          # branch back and 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 factorial tester!\n"
req:   .asciiz       "Enter a value for n (or a negative value to exit): "
ans:   .asciiz       "! is "
cr:    .asciiz       "\n"
adios: .asciiz       "Come back soon!\n"
```

Look carefully at the execution results. Clearly there is a problem! There is no way that 20 factorial is a negative number! What is the largest value of n that we can use and get a correct result? What is that correct result (you might want to verify with a calculator)? Notice that I get a value of 0 for 100 factorial. Likewise, for 101 factorial, and so on. What is the largest value of n that produces a non-zero result (of course, it is still an incorrect result)? What do you think the problem is here? (38 points)

The program calculates 12! correctly (479,001,600), but not 13!
33! produces the value -2,147,483,648. 34! and all larger factorials produce 0. Factorial grows very quickly; values above 12! cannot be represented using only 32 bits. The reason that we get 0 for 34! and above will need to wait until we get into chapter 3, but we will readdress the question at that time.

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