Computer Architecture **Name: KEY**
Midterm 1, Spring, 2014

Show Your Work! Point values are in square brackets. There are 35 points possible.

Unless a problem states otherwise, you can use any MIPS instructions, including pseudoinstructions. Be sure to document your code and adhere to the MIPS conventions for saving registers across function calls.

1. Consider two different implementations, M1 and M2, of the same instruction set. There are three classes of instructions (A, B, and C) in the instruction set. M1 has a clock rate of 4 MHz and M2 has a clock rate of 5 MHz. The average number of cycles for each instruction class and their frequencies (for a typical program) are as follows:

Inst Class	M1 CPI	M2 CPI	Frequency
A	2	2	60%
B	1	3	30%
C	4	4	10%

   (a) Calculate the average CPI for each machine. [4 points]

   (b) On a typical program — a program with the indicated instruction frequency — which machine is faster? [2 points]

   (a)

$$\begin{aligned} \text{CPI(M1)} &= \frac{2\times 6 + 1\times 3 + 4\times 1}{10} \\ &= 1.9 \\ \text{CPI(M2)} &= \frac{2\times 6 + 3\times 3 + 4\times 1}{10} \\ &= 2.5 \end{aligned}$$

   (b) Suppose the program has 10^6 instructions. Then

$$\begin{aligned} \text{Runtime(M1)} &= \frac{10^6 \times 1.9}{4\times 10^6} \\ &= 0.475 \text{ sec} \\ \text{Runtime(M2)} &= \frac{10^6 \times 2.5}{5\times 10^6} \\ &= 0.5 \text{ sec} \end{aligned}$$

   So M1 is faster.

2. (a) Convert 182_{10} from decimal to binary [2]
   (b) Convert 821_{16} from hexadecimal to decimal [2]

   (a) $182_{10} = 1011\ 0110_2$

```
182 % 2 =  0  91 % 2 =  1  45 % 2 =  1  22 % 2 =  0  11 % 2 = 1
182 / 2 = 91  91 / 2 = 45  45 / 2 = 22  22 / 2 = 11  11 / 2 = 5

5 % 2 = 1  2 % 2 = 0  1 % 2 = 1
5 / 2 = 2  2 / 2 = 1  1 / 2 = 0
```

   (b) $821_{16} = 2081_{10}$

$$\begin{aligned} 821_{16} &= 8 \times 16^2 + 2 \times 16^1 + 1 \times 16^0 \\ &= 8 * 256 + 32 + 1 \\ &= 2048 + 33 \end{aligned}$$

3. The Bleeblon uses 8-bit two's complement arithmetic.
   (a) What is the smallest value that can be stored in an 8-bit register?
   (b) What is the largest value that can be stored in an 8-bit register?

   Write your answer in base 10. [2]

   (a) $-2^7 = -128$
   (b) $2^7 - 1 = 127$

4. What is the value in `$t0` after the following instructions are executed?

```
addi  $t0, $zero, 0x000010af
srl   $t0, $t0, 2
sll   $t0, $t0, 2
```

You can write your answer in binary or hex. [2]

After the `addi`,

```
$t0 = 0x0000 10af = 0001 0000 1010 1111 (base 2)
```

After the `srl`

```
$t0 = 0x0000 042b = 0000 0100 0010 1011 (base 2)
```

After the `sll`

```
$t0 = 0x0000 10ac = 0001 0000 1010 1100 (base 2)
```

5. Bob has written a MIPS32 assembly language program that contains the following instruction:

```
addi  $t0, $t0, 0x00110ace
```

Unfortunately, Bob doesn't have the MARS simulator installed on his computer: he only has QtSpim, and QtSpim reports that this statement is invalid. Rewrite Bob's code so that QtSpim can assemble it. You should only use core instructions – no pseudoinstructions. (You can assume that registers `$t5`–`$t9` are unused in Bob's code.) [3]

```
lui   $t5, 0x0011
ori   $t5, $t5, 0x0ace
add   $t0, $t0, $t5
```

Note that you can't carry out the `lui` and `ori` on `$t0`: this would destroy the contents of `$t0`, which is what you should be adding `0x110ace` to.

6. The assembler has assigned the following byte addresses to instructions in a MIPS32 program.

```
0x0040 0060   start: beq  $t0, $t1, branch
0x0040 0064          add  $t2, $t2, $t1
0x0040 0068          addi $t1, $t1, 1
0x0040 006c          j    start
0x0040 0070  branch: addi $v0, $zero, 1
```

Find the machine language translations of the `beq` and `j` instructions. Write your answers in hexadecimal. [6]

The `beq` instruction:

```
     Opcode  rs     rt    Immed
Dec      4     8     9     3
Bin 000100  01000 01001  0000 0000 0000 0011
Bin 0001 0001 0000 1001 0000 0000 0000 0011
Hex 0x1109 0003
```

The `j` instruction:

```
     Opcode  Address
Hex    2     0010 0018
Bin 000010  00 0001 0000 0000 0000 0001 1000
Hex  0x0810 0018
```

7. Bob has written a program that runs for 10 seconds, but it spends 8 seconds in the function `Scribble`. So Bob starts trying to improve the performance of `Scribble`. [3]

   (a) If he reduces the time in scribble to 2 seconds, what is the overall performance improvement of his program?

   (b) If he reduces the time in scribble to 1 second, what is the overall performance improvement of his program?

   (c) If Bob keeps working on making Scribble run faster (but he doesn't work on the other parts of the program), is there a limit to the possible overall performance improvement of the program? If so, what is it? If not, in two sentences or less, why not?

   (a)

$$\text{Improvement} = \frac{10}{2+2} = 2.5$$

   (b)

$$\text{Improvement} = \frac{10}{2+1} = 3.33$$

   (c) Yes. Since he's only working on improving `Scribble`, he can't do better than no time spent in it.

$$\text{Improvement} = \frac{10}{2+0} = 5$$

   (This is an instance of Amdahl's Law.)

8. Sally has written the following assembly language function to calculate

$$1 + 2 + 3 + \cdots + (n-1) + n.$$

Unfortunately, the program crashes with an "arithmetic overflow," and Sally has no idea what the problem is. As a last resort, she decides to ask Prof X for help. Prof X says that the algorithm is correct; the problem is with her stack management and her use of registers. But she won't tell Sally more.

Help Sally out. Fix the function without modifying the basic algorithm. Your solution must use recursion. [4]

```
        ##############################################################
        # Sum Function
        #    $a0 = n >= 1
        # Return 1 + 2 + . . . + n
        #
sum:    subi    $sp, $sp, 4
        sw      $s0, 0($sp)

        move    $s0, $a0
        li      $t0, 1
        beq     $a0, $t0, done  # Go to done if n = $a0 = 1

        # n > 1
        subi    $a0, $a0, 1     # $a0 = n-1
        jal     sum             # Find Sum(n-1)
        add     $s0, $s0, $v0   # sum = n + Sum(n-1)

done:   move    $v0, $s0
        lw      $s0, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
```

Sally isn't saving the return address in `$ra`. So the address assigned to `$ra` in the final `jal` is used for all of the returns, effectively putting the program in an infinite loop.

She should replace the first 2 lines with the following code:

```
sum:    subi    $sp, $sp, 8
        sw      $ra, 4($sp)
        sw      $s0, 0($sp)   # unchanged
```

She should also replace the last 3 lines with the following code:

```
        lw      $s0, 0($sp)   # unchanged
        lw      $ra, 4($sp)
        addi    $sp, $sp, 8
        jr      $ra           # unchanged
```

9. Translate the following C function into MIPS32 assembly. [5]

```
void Copy(int a[], int b[], int n) {
   int i;

   for (i = 0; i < n; i++)
      b[i] = a[i];
}
```

```
        ################################################################
        # Copy function
        #    $a0 is the address of the beginning of list a (in)
        #    $a1 is the address of the beginning of list b (out)
        #    $a2 is n (in)
        # Note:  $a1 isn't changed:  the block of memory it refers
        #    to is changed
        #
        # Since this is a leaf function, we can use t registers and
        #    and we don't need to push any registers onto the stack
copy:
        # Main for loop
        li      $t0, 0            # $t0 = i = 0
lp_tst: bge     $t0, $a2, lpdone  # If  i = $t0 >= $a2 = n
                                  #    branch out of loop.
                                  #    Otherwise continue.
        # Load a[i]
        sll     $t1, $t0, 2       # $t1 is byte offset of a[i]
        add     $t1, $t1, $a0     # $t1 is absolute address of a[i]
        lw      $t2, 0($t1)       # $t2 = a[i]

        # Store a[i] in b[i]
        sll     $t1, $t0, 2       # $t1 is byte offset of b[i]
        add     $t1, $t1, $a1     # $t1 is absolute address of b[i]
        sw      $t2, 0($t1)       # b[i] = a[i]

        # Increment i
        addi    $t0, $t0, 1       # i++
        j       lp_tst            # Go to the loop test

lpdone: jr      $ra               # return
```