CECS 341 - Practice problems 2

1. 2.1 [5] <COD §2.2> For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables f, g, h, and i are given and could be considered 32-bit integers as declared in a C program. Use a minimal number of MIPS assembly instructions.

   ```
   f = g + (h – 5);
   ```

2. 2.4 [5] <COD §§2.2, 2.3> For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

   ```
   sll  $t0, $s0, 2          # $t0 = f * 4
   add  $t0, $s6, $t0        # $t0 = &A[f]
   sll  $t1, $s1, 2          # $t1 = g * 4
   add  $t1, $s7, $t1        # $t1 = &B[g]
   lw   $s0, 0($t0)          # f = A[f]
   addi $t2, $t0, 4
   lw   $t0, 0($t2)
   add  $t0, $t0, $s0
   sw   $t0, 0($t1)
   ```

3. 2.7 [5] <COD §2.3> Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

4. 2.11 [5] <COD §§2.2, 2.5> For each MIPS instruction, show the value of the opcode (OP), source register (RS), and target register (RT) fields. For the I-type instructions, show the value of the immediate field, and for the R-type instructions, show the value of the destination register (RD) field. The instructions are shown in the table:

`addi $t0, $s6, 4`	
`add  $t1, $s6, $0`	
`sw   $t1, 0($t0)`	
`lw   $t0, 0($t0)`	
`add  $s0, $t1, $t0`	

5. 2.12 Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.

   2.12.1 [5] <COD §2.4> What is the value of $t0 for the following assembly code?

add $t0, $s0, $s1

2.12.2 [5] <COD §2.4> Is the result in $t0 the desired result, or has there been overflow?

6. 2.14 [5] <COD §§2.2, 2.5> Provide the type and assembly language instruction for the following binary value: 0000 0010 0001 0000 1000 0000 0010 0000two

7. 2.15 [5] <COD §§2.2, 2.5> Provide the type and hexadecimal representation of following instruction: sw $t1, 32($t2)

8. 2.16 [5] <COD §2.5> Provide the type, assembly language instruction, and binary representation of the instruction described by the following MIPS fields:

   op = 0, rs = 3, rt = 2, rd = 3, shamt = 0, funct = 34

9. 2.17 [5] <COD §2.5> Provide the type, assembly language instruction, and binary representation of the instruction described by the following MIPS fields:

   op = 0x23, rs = 1, rt = 2, const = 0x4

10. 2.18 Assume that we would like to expand the MIPS register file to 128 registers and expand the instruction set to contain four times as many instructions.

    2.18.1 [5] <COD §2.5> How this would this affect the size of each of the bit fields in the R-type instructions?

    2.18.2 [5] <COD §2.5> How this would this affect the size of each of the bit fields in the I-type instructions?

    2.18.3 [5] <COD §§2.5, 2.10> How could each of the two proposed changes decrease the size of an MIPS assembly program? On the other hand, how could the proposed change increase the size of an MIPS assembly program?

11. 2.24 [5] <COD §2.7> Suppose the program counter (PC) is set to 0x2000 0000. Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as 0x4000 0000? Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to this same address? Why?

12. Consider a given branch instruction:
beq $s0,$s1, L1
and the case that the target address L1 is large and needs 20 bits to represent its value. Write a code that would performs the equivalent operation, but can jump to L1.

13. The MIPS assembly code below implements the following C code.

```
int fib(int n){
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fib(n − 1) + fib(n − 2);}
```

Write missing piece of code that will correct assure stack handling.
// add your code here:

```
  fib: …
          bgt  $a0, $0, test2      # if n>0, test if n=1
          add  $v0, $0, $0         # else fib(0) = 0
          j rtn                    #
   test2: addi $t0, $0, 1          #
          bne  $t0, $a0, gen       # if n>1, gen
          add  $v0, $0, $t0        # else fib(1) = 1
          j rtn
   gen:   subi $a0, $a0,1          # n-1
          jal  fib                 # call fib(n-1)
          add  $s0, $v0, $0        # copy fib(n-1)
          sub  $a0, $a0,1          # n-2
          jal  fib                 # call fib(n-2)
          add  $v0, $v0, $s0       # fib(n-1)+fib(n-2)
```

// add your code here:
rtn: …

14. Describe the operation that the following instruction performs:
lui rt, constant

15. Define a basic block

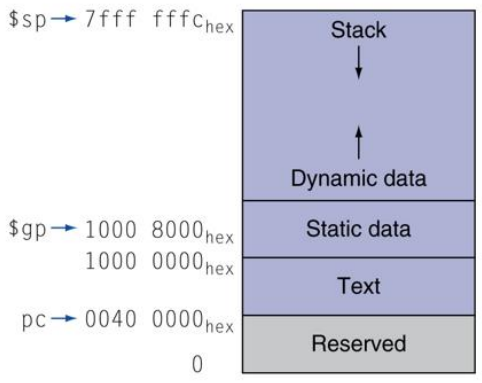
16. Signed and unsigned number conversion, such as:

Convert binary number $(110)_2$ to a decimal number, if we assume that the number is:

a. Unsigned number:

b. Signed number using 2's compliment representation

17. For a given memory layout, define each portion and explain its use.



$sp → 7fff fffc hex
Stack
Dynamic data
$gp → 1000 8000 hex
Static data
1000 0000 hex
Text
pc → 0040 0000 hex
Reserved
0