ELEC 5200-001/6200-001 (Spring 2019)
Homework 2 - Solution
Assigned 01/30/19, due 02/06/19

Note: This homework will be easier if you have completed the assigned reading for Chapter 2. You also might find the MPIS reference data sheet useful (located on the green insert inside your textbook).

Question 1: Show how the value 0x EFBEEDFE would be arranged in the memory of a big-endian machine. Assume that the data is stored starting at address 0. Show your answer in hexadecimal. (2 pts)

Address	0x03	0x02	0x01	0x00
Byte	FE	ED	BE	EF

Question 2: Show how the binary value 1101 1110 1010 1101 1111 1010 1100 1110 would be arranged in the memory of a little-endian machine. Assume that the data is stored starting at address 0. Show your answer in hexadecimal. (2 pts)

Address	0x03	0x02	0x01	0x00
Byte	DE	AD	FA	CE

Question 3: Provide the type, assembly language instruction, and binary representation of the instruction described by the following MIPS fields: op=0, rs=9, rt=12, rd=20, shamt=0, funct=39 (dec) (4 pts)

Type: R-type (register)

Assembly instruction: `nor $s4, $t1, $t4`

Binary representation:

opcode	rs	Rt	rd	shamt	funct
000000	01001	01100	10100	00000	100111

or, without the tabular format: 0000 0001 0010 1100 1010 0000 0010 0111

Question 4: Consider the following assembly code written using the MIPS instruction set:

```
        add     $s2,  $zero, $zero
        addi    $t4, $s2, 1
        add     $s3,  $s2, $t4
        sw      $s2, 0($s1)
        sw      $s3, 4($s1)
        addi    $s1,  $s1,  8
        addi    $s0,  $zero,  1

LOOP:   add     $s0,  $s0,  $t4  ++n
        lw      $t0,  -4($s1)  t0 = result [n-1]
        lw      $t1,  -8($s1)   t1 = result [n-2]
        add     $t2,  $t0,  $t1
        sw      $t2, 0($s1)   t2 = result[n]
        addi    $s1,  $s1,  4 result++
        slti    $t3,  $s0,  50
        beq     $t3,  $t4,  LOOP
```

Translate the code above into C. Assume that the 32-bit (4 byte) C-language integer `n` is the loop variable and is located in register `$s0`. Register `$s1` is initialized to the base address of the integer array named `Result`. (5 pts)

```
Result[0] = 0;

Result[1] = 1;

for (n=1; n < 50; ++n)

{

   Result[n] = Result[n-1] + Result[n-2];

}
```

Note: slight variations of the loop body may be acceptable. ex:

Result[0] = Result[-1] + Result[-2];

Result = Result + 1;

(Extra Credit: 2 points) What famous mathematical result does this code generate?

The first 50 Fibonacci numbers!

Question 5: If the current value of the PC is 0x12345678 can you use a single MIPS **jump** instruction to get to PC address 0xABCD1234? Explain your answer. (5 pts)

No. The jump address is calculated by concatenating the 4 most significant bits of the PC with the 26-bit address in the jump instruction (actually 28 bits because it is a word address, thus the two LSBs are 0). Since the 4 most significant bits of the PC are 0001 you cannot generate a target address with the most significant bits of 1010.

Note: Other answers may be acceptable. ex: Target address is more than 2^26 words(or 2^28 bytes) away from the PC.

Question 6: If the current value of the PC is 0x0FFFFFF0 can you use a single MIPS **branch** instruction to get to PC address 0x10015678? Explain your answer. (5 pts)

Yes. Branch addresses in MIPS are calculated relative to the PC. Branch instructions contain a 16 bit branch offset. You can therefore jump +/- 2^15 words from the instruction **after** the current value of the PC. The furthest branch addresses are located 2^15 words from PC + 4. PC + 4 is 0x0FFFFFF4. 2^15 words **added** to PC+4 gives us 0x0FFFFFF4 + 0x00007FFF * 4 = 0x1001FFF0. 2^15 words **subtracted** from PC+4 gives us 0x0FFFFFF4 - 0x00008000 * 4 = 0x0FFDFFF4 (remember two's complement math). Therefore, from a PC value of 0x0FFFFFF0 we can jump to any address between 0x0FFDFFF4 and 0x1001FFF0. The target address is inside this range, thus we can get there using a single branch instruction.