The University of Alabama in Huntsville
ECE Department
Homework #2 Solution
CPE 431/531 01/91/92
Fall 2015

2.2(5), 2.4(15), 2.5(15), 2.7(5), 2.13.1(5), 2.13.2(5), 2.13.3(5), 2.15(5), 2.24(5), 2.26(15), 2.40(5), 3.3(5), 3.6(5), 3.8(5)

2.2 For the following MIPS assembly instructions below, what is a corresponding C statement?

```
add   f, g, h
add   f, i, f
```

```
f = g + h + i;
```

2.4 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       # $t2 = &A[f + 1]
lw    $t0, 0($t2)       # $t0 = A[f + 1]
add   $t0, $t0, $s0     # $t0 = A[f + 1] + f
sw    $t0, 0($t1)       # B[g] = A[f + 1] + f
```

```
B[g] = A[f + 1] + f;
```

2.5 For the MIPS assembly instructions in Exercise 2.4, rewrite the assembly code to minimize the number of MIPS instructions (if possible) needed to carry out the same function.

```
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]
lw    $t0, 4($t2)       # $t0 = A[f + 1]
add   $t0, $t0, $s0     # $t0 = A[f + 1] + f
sw    $t0, 0($t1)       # B[g] = A[f + 1] + f
```

2.7 Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume that the data is stored starting at address 0.

Address	Little Endian	Big Endian
0	12	ab
1	ef	cd
2	cd	ef
3	ab	12

2.13 Assume that $s0 holds the value 128_{ten}.

2.13.1 For the instruction `add $t0, $s0, $s1`, what is the range(s) of values for $s1 that would result in overflow?

Unsigned – we can represent $0 – 2^{32} – 1$
$x + 128 > 2^{32} -1$, $x > 2^{32} - 129$
$x + 128 < 0$, $x < -128$

Signed – we can represent $-2^{31} – 2^{31} - 1$
$x + 128 > 2^{31} – 1$, $x > 2^{31} - 128$
$x + 128 < -2^{31}$, $x < 2^{31} - 128$

2.13.2 For the instruction `sub $t0, $s0, $s1`, what is the range(s) of values for $s1 that would result in overflow?

Unsigned – we can represent $0 – 2^{32} – 1$
$128 - x > 2^{32} -1$, $x < -2^{32} + 129$
$128 - x < 0$, $x > 128$

Signed – we can represent $-2^{31} – 2^{31} - 1$
$128 - x > 2^{31} – 1$, $x < -2^{31} + 129$
$128 - x < -2^{31}$, $x > 2^{31} + 128$

2.13.3 For the instruction `sub $t0, $s1, $s0`, what is the range(s) of values for $s1 that would result in overflow?

Unsigned – we can represent $0 – 2^{32} – 1$
$x - 128 > 2^{32} -1$, $x > 2^{32} + 127$
$x - 128 < 0$, $x < 128$

Signed – we can represent $-2^{31} – 2^{31} - 1$
$x - 128 > 2^{31} – 1$, $x > 2^{31} + 127$
$x - 128 < -2^{31}$, $x < 2^{31} + 128$

2.15 Provide the type and hexadecimal representation of the following instruction: `sw $t1, 32($t2)`

I-type, Opcode = 43 = 10 1011, rs = 0 1010, rt = 01001, offset = 32 = 0000 0000 0010 0000
0x1010 1101 0100 1001 0000 0000 0010 0000 = 0xAD49 0020

2.24 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 of 0x4000 0000? Is it possible to ues the branch-on-equal (`beq`) instruction to set the PC to this same address?

Jump – target address 0100 0000 0000 0000 0000 0000 0000 0000
Current PC 0010 0000 0000 0000 0000 0000 0000 0000
No, it is not possible, the top four bits of the address come from PC = $, so they will be 0010, not 0100.

Beq – Target address has to be in the range0x1FFF 8000 0x2000 7FFF, so no

2.26 Consider the following MIPS loop:

```
LOOP:   slt   $t2, $0, $t1
        beq   $t2, $0, DONE
        subi  $t1, $t1, 1
        addi  $s2, $s2, 2
        j     LOOP
DONE:
```

2.26.1 Assume that the register `$t1` is initialized to the value 10. What is the value in register `$s2` assuming `$s2` is initially zero?

The `addi` is executed 10 times, so `$s2` will be `20` .

2.26.2 For the loop above, write the equivalent C code routine. Assume that the registers `$s1`, `$s2`, `$t1`, and `$t2` are integers `A`, `B`, `i` and `temp`, respectively.

```
for (i = 10; i > 0; i--)
  B = B + 2
```

2.26.3 For the loop written in MIPS assembly above, assume that the register `$t1` is initialized to the value N. How mamy MIPS instructions are executed?

5 instructions are executed N times and 2instructions are executed the last time, so a total of 5 * N + 2.

2.40 If the current value of the PC is 0x0000 0000, can you use a single jump instruction to get to the PC address as shown in Exercise 2.39?

Exercise 2.39 gives the 32-bit constant 0010 0000 0000 0001 0100 1001 0010 0100
The jump can go up to 0x0FFF FFFC, but not to 0x2001 4924

3.3 Convert 5ED4 into a binary number. What makes base 16 (hexadecimal) an attractive numbering system for representing values in computers?

0101 1110 1101 0100 It's easy to convert from hexadecimal to binary and back.

3.6 Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate 185 -122. Is there overflow, underflow, or neither?

```
185    1011 1001          185    1011 1001
122    0111 1010         -122    1000 0110
                                  ---------
                                  0011 1111
```

The result is 63, there is neither overflow or underflow.

3.8 Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185 – 122. Is there overflow, underflow, or neither?

185 cannot be represented as an 8-bit number in sign-magnitude format, the range of numbers that can be represented in 8 bit sign-magnitude format are -127 to 127. So, there is overflow just in one of the operands.