Part 1: Intro to Assembly

1. [10 points] Translate the following MIPS code into C. Assume that the variables f, g, h, i, and j are assigned to registers $\$s0, \$s1, \$s2, \$s3$, and $\$s4$, respectively. Also assume that the base address for arrays A, and B are stored in $s6, $s7, respectively.

```
sll  $t0, $s0, 2          ##### $t0 = f × 4
add  $t0, $s6, $t0        ##### $t0 = f × 4 + Base(A)
sll  $t1, $s1, 2          ##### $t1 = g × 4
add  $t1, $s7, $t1        ##### $t1 = g × 4 + Base(B)
lw   $s0, 0($t0)          ##### f = A(f)
addi $t2, $t0, 4          ##### $t2 = f × 4 + Base(A) + 4
lw   $t0, 0($t2)          ##### $t0 = A(f + 1)
add  $t0, $t0, $s0        ##### $t0 = A(f) + A(f + 1)
sw   $t0, 0($t1)          #####, B(g) = A(f) + A(f + 1)
```

So, all of this code is just one line in c: $B(g) = A(f) + A(f+1)$

2. Assume that registers $s0, and $s1 hold the value $0x80000000$ and $0xD0000000$, respectively. (0x: means a hexadecimal number)

   a. What is the value of $t0 after execution of the following instruction? Do we have overflow here? Explain your reasoning.

$$add\ \$t0, \$s0, \$s1$$

$t0 = 0x50000000, but it should be 0x150000000, so **we have overflow**, it is a signed addition, and both of these numbers are negative, while the final result is a positive number (sign digit is zero)

Another explanation: The result is a larger bit size than the register it needs to be stored into.

b. What is the value of $t0 after execution of the following instruction? Do we have overflow here? Explain your reasoning.

$$sub\ \$t0, \$s0, \$s1$$

$t0 = 0xB0000000, and there is **no overflow**, it is a signed subtraction, and both of these numbers are negative, while the final result is negative number (sign digit is one)

c. What is the value of $t0 after execution of the following assembly code? Do we have overflow here? Explain your reasoning.

$$add\ \$t0, \$s0, \$s1$$

$$add\ \$t0, \$s0, \$t0$$

$t0 = 0xD0000000, from the first instruction, we know that there is overflow. But for the second one the result is without overflow, but overall, the result would not be correct.

3. Translate the following loop into C. The integer 'i' is held in $t1, $s2 is used to hold an integer called result, and $s0 holds the base address for an integer array MemArray.

```
 addi $t1,  $zero,  0
LOOP:  lw  $s1 ,0($s0)
   add  $s2,  $s2,  $s1
   addi $s0,  $s0,  4
   addi $t1,  $t1,  1
   slti  $t2,  $t1,  100
   bne  $t2,  $s2,  LOOP
```

$\longrightarrow$

```
int i = 0;
// Assume MemArray is an
array of 32-bit words.

for (i=0; i<100; i++) {
result += MemArray[i];
}

return result;
```

Part 2: Intro to MIPS

1.

a. Provide the type and assembly language instruction for the following binary value:

$(0000\ 0010\ 0001\ 0000\ 1000\ 0000\ 0010\ 0000)_{(2)}$

r-type -- $add\ \$s0, \$s0, \$s0$

b. Provide the type and hexadecimal representation of following instruction:

$sw\quad \$t1, 32(\$t2)$

i-type -- 0xAD490020

c. Provide the type, assembly language instruction, and binary representation of instruction described by the following MIPS fields:

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

r-type

$sub\ \$v1, \$v1, \$v0$

0x00621822

2. Show the complied MIPS code for the following C code, assume x and y is in $s1 and $s2 respectively, and the base address of A is in $s3.

y = x + A[2] + A[4]

A[6] = A[5] – y

Answer:

```
lw $t0, 8($s3)

add $t0, $s1, $t0

lw $t1, 16($s3)

add $t0, $t1, $t0

sw $t0, $s2

lw $t2, 20($s3)

sub $t2, $s2, $t2

sw $t2, 24($s3)
```