2.19 Assume the following register contents:

```
$t0 = 0xAAAAAAAA, $t1 = 0x12345678
```

First, we transform $t0 and $t1 from Hex to Binary

$t0 = 10101010101010101010101010101010

$t1 = 00010010001101000101011001111000

2.19.1 [5] <§2.6> For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

```
sll $t2, $t0, 44
or  $t2, $t2, $t1
```

We translate instruction to high level language

reg $t2 = reg $t0 << 44 bits

reg $t2 = reg $t2 | reg $t1

The sll instruction is wrong because the largest shift amount is 31 in R-type instruction. We use remainder "13" to solve this question.

$t2 = 0101 0101 0101 0101 0100 0000 0000 0000

$t1 = 0001 0010 0011 0100 0101 0110 0111 1000

$t2 = 0101 0111 0111 0101 0101 0110 0111 1000

Some of students still use 44 to solve this question, the answer is

$t2 = 0000 0000 0000 0000 0000 0000 0000 0000

$t1 = 0001 0010 0011 0100 0101 0110 0111 1000

$t2 = 0001 0010 0011 0100 0101 0110 0111 1000

2.19.2 [5] <§2.6> For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

```
sll  $t2, $t0, 4
andi $t2, $t2, -1
```

We translate instruction to high level language

reg $t2 = reg $t0 << 4 bits

reg $t2 = reg $t2 & (-1)

$t2 = 1010 1010 1010 1010 1010 1010 1010 0000

$t2 = 1010 1010 1010 1010 1010 1010 1010 0000

(-1) = 1111 1111 1111 1111 1111 1111 1111 1111

$t2 = 1010 1010 1010 1010 1010 1010 1010 0000 = 0xAAAAAAA0

2.23 [5] <§2.7> Assume `$t0` holds the value `0x00101000`. What is the value of `$t2` after the following instructions?

```
        slt  $t2, $0,  $t0
        bne  $t2, $0,  ELSE
        j    DONE
ELSE:   addi $t2, $t2, 2
DONE:
```

We translate instruction to high level language

$t0 = 100000001000000000000 = 1052672

If $0 < reg $t0 (0<1052672) then reg $t2 = 1

If reg $t0 != $0 then go to ELSE (reg $t2 = reg $t2 + 2)

else go to DONE

Step 1 : $t2 = 1

Step 2 : go to ELSE

Step 3 : $t2 = $t2 + 2 = 3

2.27 [5] <§2.7> Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `$s0`, `$s1`, `$t0`, and `$t1`, respectively. Also, assume that register `$s2` holds the base address of the array D.

```
for(i=0; i<a; i++)
    for(j=0; j<b; j++)
        D[4*j] = i + j;
```

```
          add $t0, $t0, $zero
          add $t1, $t1, $zero
LOOP1 :   slt $t2, $t0, $s0
          beq $t2, $zero, EXIT
          add $t1, $zero, $zero
          addi $t0, $t0, 1
LOOP2 :   slt $t3, $t1, $s1
          beq $t3, $zero, LOOP1
          add $t4, $t0, $t1
          muli $t5, $t1, 4
          sll $t5, $t5, 2
          add $t5, $t5, $s2
          sw $t4, $t5($s2)
          addi $t0, $t1, 1
          j LOOP2
```

2.34 Translate function `f` into MIPS assembly language. If you need to use registers `$t0` through `$t7`, use the lower-numbered registers first. Assume the function declaration for `func` is "`int f(int a, int b);`". The code for function `f` is as follows:

```
int f(int a, int b, int c, int d){
  return func(func(a,b),c+d);
}
```

f : *addi* $sp, $sp, 8

sw $ra, 4($sp)

sw $s0, 0($sp)

move $s0, $a2

jal func

move $a0, $v0

move $a1, $s0

jal func

lw $ra, 4($sp)

lw $s0, 0($sp)

addi $sp, $sp, 8

jr $ra