1. Use the register and memory values in the tables below for the next questions. Assume a 32-bit machine. Assume each of the following questions starts from the table values; that is, DO NOT use value changes from one question as propagating into future parts of the question. (25 pts)

<table>
<thead>
<tr>
<th>Registers</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t0</td>
<td>7</td>
</tr>
<tr>
<td>$t1</td>
<td>16</td>
</tr>
<tr>
<td>$t2</td>
<td>5</td>
</tr>
<tr>
<td>$t3</td>
<td>-12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>8712</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>12</td>
<td>-33</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>129</td>
</tr>
<tr>
<td>28</td>
<td>321</td>
</tr>
</tbody>
</table>

1) Give the values of $t0$, $t1$, and $t2$ after this instruction: sub $t1$, $t0$, $t2$

$t0=7$, $t1=2$, $t2=5$

2) What values will be in $t1$ and $t2$ after this instruction is executed: lw $t2$, 8($t1$)

$t1=16$, $t2=129$

3) What value will be in the register $t1$ after this instruction is executed: addi $t1$, $t1$, -20

$t1=-4$

4) What value will be in the register $t2$ after this instruction is executed: sll $t2$, $t2$, 3

$t2=40$

5) What values will be in the registers $t1$ and $t2$ after this instruction is executed: sw $t1$, 4($t2$)
$t1=16, $t2=5

6) (Bonus 5pts) What value will be in the register $t0 after this instruction is executed: srl $t0, $t0, 2 $t1=1

7) (Bonus 5 pts) What binary value will be in $t0 after this instruction is executed: addu $t0, $t0, $t3

11111111 11111111 11111111 11110100
+ 00000000 00000000 00000000 00000111
= 11111111 11111111 11111111 11111011

2. Suppose we have a 32-bit MIPS word containing the value 8E680064_{16}. We would like to know what MIPS machine instruction this word represents.

1) Write the instruction word in binary (2 pts)

<table>
<thead>
<tr>
<th>I-format</th>
<th>$s3</th>
<th>$t0</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>19</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>100011</td>
<td>10011</td>
<td>01000</td>
<td>0000000001100100</td>
</tr>
</tbody>
</table>

2) What is the format of this instruction? (2 pts)

I-format

3) Translate this instruction to MIPS assembly language. (6 pts)

lw $t0, 100($s3)

3. Convert the following MIPS instructions into machine instructions in hexadecimal form. Show every step in the conversion (20 pts).

1) sw $t4, -4($s3)

<table>
<thead>
<tr>
<th>I-format</th>
<th>$s3</th>
<th>$t4</th>
<th>-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>19</td>
<td>12</td>
<td>-4</td>
</tr>
<tr>
<td>101011</td>
<td>10011</td>
<td>01100</td>
<td>11111111 11111100</td>
</tr>
</tbody>
</table>

AE6CFFFFC

2) and $t1, $s1, $s2

<table>
<thead>
<tr>
<th>R-format</th>
<th>$s1</th>
<th>$s2</th>
<th>$t1</th>
<th>0</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17</td>
<td>18</td>
<td>9</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>000000</td>
<td>10001</td>
<td>10010</td>
<td>01001</td>
<td>00000</td>
<td>100100</td>
</tr>
</tbody>
</table>
4. For the following sequence of MIPS instructions, identify all registers used and their values after the code has been executed. (5 pts)

```
addi $t1, $zero, 4
addi $t2, $zero, 12
slt $t3, $t2, $t1
beq $t3, $zero, NEXT
add $s0, $zero, $t3
j END
NEXT: add $s0, $t1, $t2
END:
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s0</td>
<td>16</td>
</tr>
<tr>
<td>$t1</td>
<td>4</td>
</tr>
<tr>
<td>$t2</td>
<td>12</td>
</tr>
<tr>
<td>$t3</td>
<td>0</td>
</tr>
</tbody>
</table>

5. In MIPS assembly, write an assembly language version of the following C code segment:

```c
int A[1000];
int min, i;

min = A[0]
for (i=1; i < 1000; i++) {
    if (A[i] < min) {
        min = A[i];
    }
}
```

At the beginning of this code segment, the base address of array A is in register $s0. Values of min and i are in registers $s1 and $s2, respectively. Avoid the use of multiplication instructions and pseudoinstructions since they are unnecessary. (20 pts)

```
lw $s1, 0($s0)   # min = A[0]
addi $s2, $zero, 1 # i = 1

LOOP:
    slti $t0, $s2, 1000 # Compare with Loop Bound
    beq $t0, $zero, Exit # Exit
    sll $t0, $s2, 2 # (i)*4
    add $t0, $s0, $t0 # address of A[i]
    lw $t0, 0($t0) # $t0<-A[i]
```
6. Convert the C function below to MIPS assembly language. Make sure that your assembly language code could be called from a standard C program (that is to say, make sure you follow the MIPS calling conventions). Avoid the use of multiplication instructions and pseudoinstructions since they are unnecessary. (20 pts)

int abssum(int a, int b)
{
    int sum;

    sum = a + b;

    if (sum < 0)
        return(-sum);
    else
        return(sum);
}

The stack grows downward (toward lower memory addresses). Assume the attribute values a and b stored in $a0 and $a1, respectively. Local variable sum is in $s0,

abssum:
    addi $sp, $sp, -4       # Set up the stack
    sw $s0, 0($sp)           # Save $s0
    add $s0, $a0, $a1        # sum = a+b;
    slt $t0, $s0, $zero     # sum < 0;
    beq $t0, $zero, NEG
    sub $v0, $zero, $s0      # $v0= - sum
    lw $s0, 0($sp)           # Recover $s0
    addi $sp, $sp, 4         # Set stack pointer
    jr $ra

NEG:
    add $v0, $zero, $s0      # $v0= sum
    lw $s0, 0($sp)            # Recover $s0
    addi $sp, $sp, 4          # Set stack pointer
    jr $ra