**Question One (5 pts.)**

**For each question, assume initial register values of:**

**$t0: 20**

**$t1: 50**

**$t2: 60**

**1. After the following, what is $t4?**

**addi $t4, $t2,** 1

$t2 = 60, so $t4 will be 60+1=61.

**2. After the following, what is $t3?**

**addi $t3, $t1, -5**

$t1= 50, so $t3 will be 50−5=45.

**3. After the following, what is $t2?**

**addi $t2, $t2, 6**

$t2 = 60, so $t2 will be 60+6=66.

**4. Type an addi instruction that writes $t5 with the sum of $t4 and 17.**

addi $t5, $t4, 17

**5. Type an instruction that adds 3 to $t4, writing the sum to $t4.**

addi $t4, $t4, 3

**Question Two (5 pts.)**

**Assume initial register values of**

**$t0: 20**

**$t1: 30**

**$t2: 40**

**1. After the following, what is $t0?**

**add $t0, $t1, $t2**

$t1 = 30, $t2 = 40. Adding them together, $t0 = 70.

**2. After the following, what is $t2?**

**add $t0, $t1, $t2**

$t1 = 30, $t2 = 40. Adding them together and storing the result in $t0 does not change the value of $t2. Therefore, $t2 = 40.

**3. After the following, what is $t2?**

**add $t2, $t1, $t0**

$t1 = 30, $t0 = 70. Adding them together and storing the result in $t2, $t2 = 100.

**4. After the following, what is $t2?**

**add $t2, $t0, $t1**

$t0 = 70, $t1 = 30. Adding them together and storing the result in $t2, $t2 = 100.

**5. Type an instruction that writes $t3 with the sum of $t5 and $t6.**

add $t3, $t5, $t6

**Question 3 (4 pts.)**

**Assume initial register values of:**

**$t0: 30**

**$t1: 10**

**$t2: -5**

**$t3: 5**

**1. After the following, what is $t4?**

**sub $t4, $t1, $t3**

$t1 = 10, $t3 = 5. Subtracting $t3 from $t1, $t4 = 5.

**2. After the following, what is $t0?**

**sub $t0, $t0, $t3**

$t0 = 30, $t3 = 5. Subtracting $t3 from $t0, $t0 becomes 25.

**3. After the following, what is $t5?**

**sub $t5, $t1, $t2**

$t1 = 10, $t2 = -5. Subtracting $t2 from $t1, $t5 = 15.

**4. Type an instruction to subtract $t3 from $t4, writing the difference to $t5.**

sub $t5, $t4, $t3

**Question 4 (4 pts.)**

For each question, assume initial register values of:

$t0: 20

$t1: 15

$t2: 15

$t3: 21

1. After the following, what is $t3?

bne $t0, $t1, Cont

addi $t3, $t3, 5

Cont: addi $t2, $t2, 2

Since branch condition is satisfied ($t0 and $t1 are not equal), it skips to the next instruction after the label Cont. Therefore, only addi $t2, $t2, 2 is executed. $t3 remains so, **$t3 = 21**

2. After the following, what is $t3?

bne $t1, $t2, Cont

addi $t3, $t3, 7

Cont: addi $t2, $t2, 3

Since branch condition is not satisfied ($t1 and $t2 are equal), resumes to the next instruction , being addi $t3, $t3, 7 and so on. So**, $t3 = 21 + 7 = 28**

3. After the following, what is $t2?

bne $t1, $t2, Cont

addi $t3, $t3, 7

Cont: addi $t2, $t2, 3

Since branch condition is not satisfied ($t1 and $t2 are equal), resumes to the next instruction , being addi $t3, $t3, 7 and so on. So, **$t2 = 15 + 3 = 18**

4. After the following, what is $t3?

bne $t2, $t1, Cont

addi $t3, $t3, 8

Cont: addi $t3, $t3, 4

Since branch condition is satisfied ($t2and $t1 are not equal), it skips to the next instruction after the label Cont. Therefore, only addi $t3, $t3, 4 is executed. So, **$t3 = 21 + 4 = 25**

**Question 5 (10 pts )**

**Translate the following binary machine code into MIPS assembly instructions (show your work).**

**1) 0x20080000**

- Opcode (6 bits): `001000` (hex: 0x20) - corresponds to the "addi" operation.

- Source register (5 bits): `00000` - corresponds to register $zero.

- Destination register (5 bits): `01000` - corresponds to register $t0.

- Immediate value (16 bits): `0000000000000000` - corresponds to 0.

**assembly instructions: addi $t0, $zero, 0**

**2) 0x0089502A**

- Opcode (6 bits): `001000` (hex: 0x20) - corresponds to the "addi" operation.

- Source register (5 bits): `01001` - corresponds to register $t1.

- Destination register (5 bits): `01010` - corresponds to register $t2.

- Immediate value (16 bits): `0010101010101010` (hex: 0x2A2A) - corresponds to 42.

**assembly instructions: slti $t2, $t1, 42**

**3) 0x01094020**

- Opcode (6 bits): `000000` (hex: 0x00) - corresponds to the "add" operation.

- Source register (5 bits): `01001` - corresponds to register $t1.

- Destination register (5 bits): `01000` - corresponds to register $t0.

- Target register (5 bits): `01010` - corresponds to register $t2.

- Function code (6 bits): `100000` (hex: 0x20) - corresponds to the "add" operation.

**assembly instructions : add $t0, $t1, $t2**

**4) 0xA0A90000**

- Opcode (6 bits): `101000` (hex: 0x0A) - corresponds to the "or" operation.

- Source register (5 bits): `01010` - corresponds to register $t2.

- Destination register (5 bits): `01001` - corresponds to register $t1.

- Target register (5 bits): `01000` - corresponds to register $t0.

- Function code (6 bits): `000000` (hex: 0x00) - corresponds to the "or" operation.

**assembly instructions : or $t1, $t0, $t2**

**5) 0x03E00008**

- Opcode (6 bits): `000111` (hex: 0x0E) - corresponds to the "jr" operation.

- Source register (5 bits): `00000` - corresponds to register $zero.

- Destination register (5 bits): `11111` - corresponds to register $ra.

- Target address (16 bits): `0000000000001000` (hex: 0x08) - corresponds to the address.

**assembly instructions: jr $ra**

**Question 6 (5 pts.)**

**Write the following strings using ASCII encoding. Write your final answers in hexadecimal. a and b are worth 1 pt. and c is worth 3 pts.**

**a) SOS**

* + 'S' is 0x53
  + 'O' is 0x4F
  + 'S' is 0x53

Answer: **0x53 0x4F 0x53**

**b) Cool!**

* + 'C' is 0x43
  + 'o' is 0x6F
  + 'o' is 0x6F
  + 'l' is 0x6C
  + '!' is 0x21

**Answer: 0x43 0x6F 0x6F 0x6C 0x21**

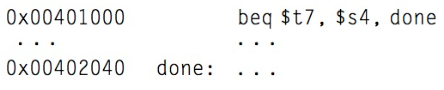
**c) CIS221 Digital Logic**

* + 'C' is 0x43
  + 'I' is 0x49
  + 'S' is 0x53
  + '2' is 0x32
  + '2' is 0x32
  + '1' is 0x31
  + ' ' is 0x20 (space)
  + 'D' is 0x44
  + 'i' is 0x69
  + 'g' is 0x67
  + 'i' is 0x69
  + 't' is 0x74
  + 'a' is 0x61
  + 'l' is 0x6C
  + ' ' is 0x20 (space)
  + 'L' is 0x4C
  + 'o' is 0x6F
  + 'g' is 0x67
  + 'i' is 0x69
  + 'c' is 0x63

Answer: **0x43 0x49 0x53 0x32 0x32 0x31 0x20 0x44 0x69 0x67 0x69 0x74 0x61 0x6C 0x20 0x4C 0x6F 0x67 0x69 0x63**

**Question 7 (3 pts)**

Convert the following beq assembly instruction into machine code. Instruction addresses are given to the left of each instruction (show your work).



* Opcode for beq is 0x04.
* Source register $t7 is 0x1F (31 in decimal).
* Target register $s4 is 0x14 (20 in decimal).

Offset needs to be calculated based on the target address relative to the next instruction. In this case, the offset is 0x02040 - 0x01000 = 0x01040, and since it's a signed offset, it needs to be represented in two's complement form.

* Convert 0x01040 to binary: 0000 0001 0000 0100 0000
* Represent in two's complement: 1111 1110 1111 1100 00
* opcode | rs | rt | offset
* 000100 | 11111 | 10100 | 1111111111111000

Making it so, In hexadecimal, this becomes: 0x1003FFFC

**Question 8 (4 pts)**

Convert the following jal assembly instruction into machine code. Instruction addresses are given to the left of each instruction (show your work).



To convert the jal (jump and link) assembly instruction into machine code, we need to calculate the offset=(target address−current PC​)/4 −1

* Opcode (6 bits): **0x3** (opcode for **jal**)
* Target Address (funcfunc): 0x0041147c
* Current PC: 0x00403000

Offset=(0*x*0041147*c*−0*x*00403000​)/4 − 1

Offset = (0*x*0000*E*47*C*​)/4 − 1

offset=0*x*0000391*F*−1

offset=0x0000391*E*

* Machine Code=Opcode∣∣Offset
* Machine Code=0*x*3∣∣0*x*0000391*E*

After combining all our data, the final machine code is: 0*x*3000391*E*