For full and partial credits, you must show your work and explain all steps.

ρ8 List three of the main characteristics that define an ISA.
&ÕÎÃÔÉÏÎÁÌ 5ÎÉÔÓ
-ÅÍÏÒÙ
)T/ ÓÙÓÔÅÍ

2. The following program tries to copy words from the address in register $a0 to the address in register $a1, counting the number of words copied in register $v0. The program stops copying when it finds a word equal to 0. You do not have to preserve the contents of registers $v1, $a0, and $a1. This terminating word should be copied but not counted.

```
        addi $v0, $zero, 0      # Initialize count
loop:   lw $v1, 0($a0)          # Read next word from source
        sw $v1, 0($a1)          # Write to destination
        addi $a0, $a0, 4        # Advance pointer to next source
        addi $a1, $a1, 4        # Advance pointer to next destination
        beq $v1, $zero, loop    # Loop if word copied != zero
```

```
      addi $v0, $zero, 0   # Initialize count
loop: lw   $v1, 0($a0)     # Read next word from source
     sw $v1, 0($a1)        # Write to destination
     addi $a0, $a0, 4      # Advance pointer to next source
     addi $a1, $a1, 4      # Advance pointer to next destination
     addi $v0, $v0, 1      #increment count by one
     bne $v1, $zero, loop  #Loop if word copied !=zero
     addi $v0, $v0, -1
```

There are multiple bugs in this MIPS program; fix them and turn in a bug-free version

3. Write a MIPS assembly language program for the following C ‘if statement’:

```
if (i ≥ 5)
      A[7] = A[6] + A[5];
else
      k = j × 15;
```

Assume that i, j, k, and base address of A are assigned to $s1, $s2, $s3, and $s6 respectively. You can use temporary registers ($t0, $t1, etc.) if needed. Do not use ‘mul’ or ‘div’ instructions.

```
        lw $t0, 20($s6)   #$t0 is A[5]
        lw $t1, 24($s6)   #$t1 is A[6]
        blt $s1, 5, else
        add $t2, $t0, $t1 #add A[6] to A[5] to $t2
        sw $t2, 28($s6)   #$t2 is put into A[7]
else:   sll $s3, $s2, 4

       add $s3, $s3, $s2
```

4. Translate the following expression into MIPS assembly language. Use less than 6 instructions. For division, assume that there is no remainder. You can use temporary registers if needed.

a. `$t0 = $s0/4 – 5*$s1` ;do not use ‘mul’ or ‘div’ instructions
b. `abs $s1, $s2` ;$s1 = abs[$s2], ‘abs[$s2]’ is an absolute value of $s2.
c. `$s0 = 12345678` ;12345678 (decimal) = 188 × 2^{16} + 24910

```
a. srl $t0, $s0, 2
   sll $t1, $s1, 2
   add $t1, $t1, $s1
   sub $t0, $t0, $t1
b.      blt $s2, $zero, neg
        addi $s1, $s2, 0
        j Exit
   neg: sub $s1, $zero, $s2
   Exit:
c.
      addi $s0, $zero,188
      sll $s0, $s0, 15
      addi $s0, $s0, 24910
```

5. Assume that we would like to expand the MIPS register file to 64 registers and expand the instruction set to contain two times as many instructions.
   a. How this would this affect the size of each of the bit fields in the R-type instructions?
      Assuming that we still have 32-bit instructions
      opcode: 31-25 7-bits to double instructions
      rs:24-19 6-bits for the 64 registers
      rt:18-13 6-bits for the 64 registers
      rd: 13-7 6-bits for the 64 registers
      sa: 6 1-bit because the other bits are used to expand register and double instructions
      fn: 5-0 6-bits kept it the same so ALU could operate the same
   b. How this would this affect the size of each of the bit fields in the I-type instructions?
      Assuming that we still have 32-bit instructions
      opcode: 31-25 7-bits to double instructions
      rs:24-19 6-bits for the 64 registers
      rt:18-13 6-bits for the 64 registers
      imm: 12-0 13-bits because we put the other 3 bits into the other fields

6. The following instruction is not included in the MIPS instruction set:

```
lwn Rd, Rs, Rt
```

The lwn (load word on not-zero) will set Rd = Mem[[Rs]+[Rt]] if [Rt] ≠ 0 (move the content of memory location with address = [Rs]+[Rt] to the register Rd if [Rt] is not equal to zero).

   a. If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?
      R-Type
   b. What is the shortest sequence of MIPS instructions that performs the same operation?

```
    beq $Rt, $zero, Exit
    add $Rs, $Rs, $Rt
    lw $Rd, 0($Rs)
Exit:
```