Hw 1) Given 32 bit: 1010 1101 0001 0000 0000 0000 0000 0010

a.) 2's comp signed int = $\boxed{-1391460350}$

b.) MIPS

101011	01000	10000	0000 0000 0000 0010
op	rs	rt	imm
0x2b	8=$tØ	16=$s0	2

sw $sØ, 2($tØ)
# $sØ → MEMORY[$tØ+2]

2. (4 pts) Determine the absolute value of a signed integer. Show the implementation of the following pseudo-instruction using three real instructions:
abs $t1, $t2

```
      addu $t1, $t2, $zero  #interpretation of move pseudo instruction
      bgez $t1, next        #if t1 >= 0 branch to next
      subu $t1, $zero, $t1  #else if neg int take 0-(-int)= +int
next:...
```

3.)

```
#move $t1, $t2
addu $t1, $t2, $zero       #$t1 = $t2 + 0 therefore $t1 = $t2

#clear $t5
and  $t5, $t5, $zero       #$t5 = $t5 AND 0 therefore $t5 = 0

#li $t5, imm32
lui $s1, (upper 16 bits)        #stores upper 16 bits
ori $s1, $s1, (lower 16 bits)   #inserts the lower 16 bits

#addi $t5, $t3, imm32
lui $t5, (upper 16 bits)        #similar function as above
ori $t5, $t5, (lower 16 bits)   #similar function as above
addi $t5, $t5, $t3              #$t5 = $t5 + $t3

#beq $t5, imm32, Label
lui $at, (upper 16 bits)        #stores upper 16 bits in $at
ori $at, $at, (lower 16 bits)   #inserts lower 16 bits in $at
beq $t5, $at, Label             #if $t5 == $at branch to Label

#ble $t5, $t3, Label            #branch if $t5 <= $t3
slt $at, $t3, $t5               #if $t3 < $t5 $at=1
beq $at, $zero, Label           #if $at == 0 branch to Label
#if $t3 < $t5 then at=1 and $at != 0 ...no branch
#if $t5 <= $t3 then at=0 and $at == 0...branches

#bgt $t5, $t3, Label            #branch if $t5 > $t3
slt $at, $t3, $t5               #if $t3 < $t5 $at=1
bne $at, $zero, Label           #if $at != 0 branch to Label
#if $t3 < $t5 then $at=1 and $at != 0 ...branches

#bge $t5, $t3, Label            #branch if $t5 >= $t3
slt $at, $t5, $t3               #if $t5 < $t3 $at=1
beq $at, $zero, Label           #if $at == 0 branch to Label
#works similar to ble but changes places for rs and rt
```

4. (8 pts) Translate the following statements into MIPS assembly language. Assume that a, b, c, and d are allocated in \$s0, \$s1, \$s2, and \$s3. All values are signed 32-bit integers.
   a) `if ((a > b) || (b > c)) {d = 1;}`
   b) `if ((a <= b) && (b > c)) {d = 1;}`

```
#implementation for a: if (($s0 > $s1) || ($s1 > $s2)) {$s3 = 1;}
#pseudocode implementation
     bgt $s0, $s1, L1                #if yes jump to 'if' part (L1)
     ble $s1, $s2, next         #if no skip 'if' part
L1:  li $s3, 1                  #set $s3 to 1
next:...                        #branch here to skip 'if' inst


#implementation with no pseudocode
     slt $at, $s1, $s0          #if $s1 < $s0 $at=1
     bne $at, $zero, L1         #if $at != 0 branch to 'if' part (L1)
     slt $at, $s2, $s1          #if $s2 < $s1 $at=1
     beq, $at, $zero, next      #if $at == 0 skip 'if', branch to
next
L1:  ori $s3, $zero, 1          #OR inst used to set bits
next:...                        #branch here to skip 'if' inst
```

```
#implementation for b: if (($s0 <= $s1)&&($s1 > $s2)) {$s3=1;}
#pseudocode implementation
     bgt $s0, $s1, next         #if $s0>$s1 skip 'if'
     ble $s1, $s2, next         #if $s1<=$s2 skip 'if'
     li $s3, 1                  #set $s3 to 1
next:...                        #branch here to skip 'if' inst


#implementation with no pseudocode
     slt $at, $s1, $s0          #if $s1 < $s0 $at=1
     bne $at, $zero, next       #if $at != 0 skip 'if', branch to
next
     slt $at, $s2, $s1          #if $s2 < $s1 $at=1
     beq $at, $zero, next       #if $at == 0 skip 'if', branch to
next
L1: ori $s3, $zero, 1           #OR inst used to set bits
next:...                        #branch here to skip 'if' inst
```

5. (8 pts) Consider the following fragment of C code:

```
for (i=0; i<=100; i=i+1) { a[i] = b[i] + c; }
```

Assume that a and b are arrays of words and the base address of a is in $a0 and the base address of b is in $a1. Register $t0 is associated with variable i and register $s0 with c. Write the code in MIPS.

```
      and $t0, $t0, $zero        #clears $t0 to use as iterator
      addi $t1, $zero, 404       #sets $t1 to 404(max iterator val)
loop: add $t2, $a1, $t0          #$t2 = B base addr + iterator
      lw $t2, 0($t2)             #$t2 = B[i]
      addu $t2, $t2, $s0         #$t2 = B[i] + C
      add $t3, $a0, $t0          #$t3 = A base addr + iterator
      sw $t2, 0($t3)             #store sum of B[i] + C in A[i]
      addi $t0, $t0, 4                #$t0 = $t0 + 4 (i++)
      bne $t0, $t1, loop         #if iterator != 404 branch to loop
```

6. (8 pts) Add comments to the following MIPS code and describe in one sentence what it computes. Assume that $a0 is used for the input and initially contains n, a positive integer. Assume that $v0 is used for the output.

```
begin:     addi $t0, $zero, 0         #clears space for storage
           addi $t1, $zero, 1         #sets iterator to 1
loop:      slt $t2, $a0, $t1          #is n < iterator?
           bne $t2, $zero, finish     #if n < i branch to finish
           add $t0, $t0, $t1          #else store value of i in $t0
           addi $t1, $t1, 2                #iterator = iterator + 2
           j loop                     #repeat loop
finish:    add $v0, $t0, $zero        #output = stored value of $t0
```

This code starts with 1 and sums all the odd ints less than or equal to input $a0.

7. (12 pts) The following code fragment processes an array and produces two important values in registers $v0 and $v1. Assume that the array consists of 5000 words indexed 0 through 4999, and its base address is stored in $a0 and its size (5000) in $a1. Describe what this code does. Specifically, what will be returned in $v0 and $v1?

```
        add $a1, $a1, $a1       #a1 = 5,000 + 5,0000 = 10,000
        add $a1, $a1, $a1       #a1 = 10,000 + 10,000 = 20,000
        add $v0, $zero, $zero   #clears $v0 to use to sum values
        add $t0, $zero, $zero   #clears $t0 to use as iterator
outer:  add $t4, $a0, $t0       #$t4 = A base address + iterator
        lw $t4, 0($t4)          #$t4 = A[i]
        add $t5, $zero, $zero   #clears $t5 to use as a counter
        add $t1, $zero, $zero   #clears $t1 to use as iterator2
inner:  add $t3, $a0, $t1       #$t3 = A base address + iterator2
        lw $t3, 0($t3)          #$t3 = A[j]
        bne $t3, $t4, skip      #if A[j] != A[i] branch to skip
        addi $t5, $t5, 1        #else counter++
skip:   addi $t1, $t1, 4        #point to next A[j]
        bne $t1, $a1, inner     #if iterator2 != 20,000 loop to inner
        slt $t2, $t5, $v0       #is counter < $v0?
        bne $t2, $zero, next    #if $v0 is greater jump to next
        add $v0, $t5, $zero     #else $v0 = counter value
        add $v1, $t4, $zero     #     $v1 = A[i]
next:   addi $t0, $t0, 4        #point to next A[i]
        bne $t0, $a1, outer     #if iterator != 20,000 branch to outer
```

(1) Goes from A[0] to A[4999] using first iterator...A[i]
(2) Goes from A[0] to A[4999] using a second iterator...A[j]
(3) If elem in A[i] matches elem in A[j] increase counter, else j++ and check next element
(4) If counter value for current check is greater than stored value: store the counter value and store the value for A[i], else i++ and check next element

$v0 will return the highest counter value and $v1 will return the index of the elem that generated the highest counter value. So the code is finding which element appears the most frequently and how many times it appears.