HW#4 (CSC390); Turn in your code on the Blackboard by 03/11/2016 by 5:00PM

#Q1. Consider the following two C procedures (swap and sort) and their corresponding MIPS assembly codes as shown in the figures 1 and 2, respectively. Using these two C procedures (swap and sort) write a MIPS assembly program that will sort the following array elements in the **ascending order**.

Array= [100 50 75 -1 -50 500 20 40 40 17 19 23 5 7 -20]

```
void swap(int v[], int k)
{
 int temp;
 temp = v[k];
 v[k] = v[k+1];
 v[k+1] = temp;
}
```

Procedure body

```
swap: sll   $t1, $a1, 2        # reg $t1 = k * 4
      add   $t1, $a0, $t1      # reg $t1 = v + (k * 4)
                               # reg $t1 has the address of v[k]
      lw    $t0, 0($t1)        # reg $t0 (temp) = v[k]
      lw    $t2, 4($t1)        # reg $t2 = v[k + 1]
                               # refers to next element of v
      sw    $t2, 0($t1)        # v[k] = reg $t2
      sw    $t0, 4($t1)        # v[k+1] = reg $t0 (temp)
```

Procedure return

```
      jr    $ra                # return to calling routine
```

Figure 1 MIPS assembly code for swap procedure

```
void sort (int v[], int n)
{
   int i, j;
   for (i = 0; i < n; i += 1) {
      for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
           swap(v,j);
      }
   }
}
```

Saving registers

```
sort:   addi    $sp,$sp, -20        # make room on stack for 5 registers
        sw      $ra, 16($sp)# save $ra on stack
        sw      $s3,12($sp)         # save $s3 on stack
        sw      $s2, 8($sp)# save $s2 on stack
        sw      $s1, 4($sp)# save $s1 on stack
        sw      $s0, 0($sp)# save $s0 on stack
```

Procedure body

```
Move parameters
        move    $s2, $a0 # copy parameter $a0 into $s2 (save $a0)
        move    $s3, $a1 # copy parameter $a1 into $s3 (save $a1)

Outer loop
        move    $s0, $zero# i = 0
for1tst:
        slt$t0, $s0, $s3 #  reg $t0 = 0 if $s0 Š $s3 (i Š n)
        beq     $t0, $zero, exit1# go to exit1 if $s0 Š $s3 (i Š n)

Inner loop
        addi    $s1, $s0, -1# j = i - 1
for2tst:
        slti$t0, $s1, 0     # reg $t0 = 1 if $s1 < 0 (j < 0)
        bne     $t0, $zero, exit2# go to exit2 if $s1 < 0 (j < 0)
        sll     $t1, $s1, 2# reg $t1 = j * 4
        add     $t2, $s2, $t1# reg $t2 = v + (j * 4)
        lw      $t3, 0($t2)# reg $t3 = v[j]
        lw      $t4, 4($t2)# reg $t4 = v[j + 1]
        slt     $t0, $t4, $t3 # reg $t0 = 0 if $t4 Š $t3
        beq     $t0, $zero, exit2# go to exit2 if $t4 Š $t3

Pass parameters
and call
        move    $a0, $s2            # 1st parameter of swap is v (old $a0)
        move    $a1, $s1 # 2nd parameter of swap is j
        jal     swap                # swap code shown in Figure 2.25

Inner loop
        addi    $s1, $s1, -1# j -= 1
        j       for2tst             # jump to test of inner loop

Outer loop
exit2:  addi    $s0, $s0, 1         # i += 1
        j       for1tst             # jump to test of outer loop
```

Restoring registers

```
exit1:  lw      $s0, 0($sp)         # restore $s0 from stack
        lw      $s1, 4($sp)# restore $s1 from stack
        lw      $s2, 8($sp)# restore $s2 from stack
        lw      $s3,12($sp)         # restore $s3 from stack
        lw      $ra,16($sp)         # restore $ra from stack
        addi    $sp,$sp, 20         # restore stack pointer
```

Procedure return

```
        jr      $ra                 # return to calling routine
```

Figure 2 MIPS assembly version of sort procedure

What chages you would make to sort the array elements in **descending order**.

Results (ascending order) :

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	-50	-20	-1	5	7	17	19	20
0x10010020	23	40	40	50	75	100	500	0

Results (descending order) :

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	500	100	75	50	40	40	23	20
0x10010020	19	17	7	5	-1	-20	-50	0

Q2. (Read chapter#3 (pages 178-182) of your text book before you solve the following questions)

(a) MIPS assembler generates an exception error when overflow occurs in mathematical operation of signed numbers. Write a MIPS assembly language program that will check overflows during the element by element addition between the array elements (signed numbers) of two vectors A and B and store the results in C vector. If overflow occurs the code will store $0xFFFFFFFF_{hex}$ in the corresponding memory location of the C vector, otherwise store the actual addition results.

Consider the following vectors A and B and store the results C vectors in your coding:

A = [1000000000, 2000000000, 2000000000, -1000000000, -2000000000];

B= [1000000000, -1000000000, 1000000000, -1000000000, -1000000000];

C=[. . . …. …. …];

(b) Write a MIPS assembly language program that will check overflows during the element by element addition between the array elements (unsigned numbers) of two vectors A and B and store the results in C vector. If overflow occurs the code will store 0x00000000 in the corresponding memory location of the C vector, otherwise store the actual addition results.

Consider the following vectors A and B and store the results C vectors in your coding:

A = [1000000000, 2000000000, 2000000000, 1000000000, 2000000000];

B = [3000000000, 3000000000, 1000000000, 4000000000, 2000000000];

C=[. . . …. …. …];

Note: Please turn in your Code on the Blackboard by 03/11/2016 by 5:00PM