

1. Complete the InsertionSort subprogram

```

        .data
scores: .word 20, 30, 10, 40, 50, 60, 30, 25, 10, 50
n:      .word 10

        .text
        .globl main

main:
        la $a0, scores
        lw $a1, n
        jal insertionSort

        li $v0, 10
        syscall

end_main:

insertionSort:
        # setup call-frame
        # save the return addr back to main
        # save main's s-registers

        # save address of numbers in $s0
        # save (length-1) in $s1

for_init:
        # save firstUnsortedIndex in $s2

for_loop:
        # fill actual parameters

        # call insert subpgm

end_for:
        # restore return addr. and
        # restore main's s-registers

        # remove call-frame
        # 'jump register' back to main

end_insertionSort:

insert: # insert does not call any subpgms so we'll use $a and $t registers
        # nothing to save on the run-time stack
        move $t0, $a2      # use $t0 for testIndex
        li $t4, 4
while:  blt $t0, 0, end_while
        mul $t1, $t0, $t4
        add $t1, $a0, $t1
        lw $t2, 0($t1)
        ble $t2, $a1, end_while
        sw $t2, 4($t1)
        sub $t0, $t0, 1
        j while
end_while:
        mul $t1, $t0, $t4
        add $t1, $a0, $t1
        sw $a1, 4($t1)
        jr $ra      # 'jump register' back to insertionSort

end_insert:

```

```

main:

        $a0 $a1
InsertionSort(scores, n)

. . .
end main

        $a0 -> $s0          $a1 -> $s1
InsertionSort(numbers - address to integer array, length - integer)
        $s2
integer firstUnsortedIndex
for firstUnsortedIndex = 1 to (length-1) do
        $a0 $a1          $a2
        Insert(numbers, numbers[firstUnsortedIndex], firstUnsortedIndex-1);
end for
end InsertionSort

Insert($a0 - address to integer array, elementToInsert - integer,
      $a2 lastSortedIndex - integer) {
        $t0
integer testIndex;
testIndex = lastSortedIndex
while (testIndex >=0) AND (numbers[testIndex] > elementToInsert )
        numbers[ testIndex+1 ] = numbers[ testIndex ];
        testIndex = testIndex - 1;
end while
end Insert

```

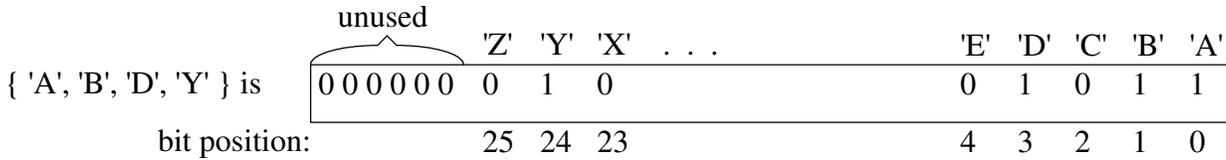
2. If \$5 contains 0xAF000A5D and \$6 contained 0x6, what hexadecimal value would be in \$4 after each of the following?

- a) sll \$4, \$5, 3
- b) sllv \$4, \$5, \$6
- c) sra \$4, \$5, 3
- d) ror \$4, \$5, \$6

3. If \$5 contains 0xA5D and \$6 contained 0x63C, what hexadecimal value would be in \$4 after each of the following?

- a) and \$4, \$5, \$6
- b) ori \$4, \$5, 0xBF
- c) xor \$4, \$5, \$6
- d) nor \$4, \$5, \$6
- e) not \$4, \$5

4. Sometimes you want to manipulate individual bits in a “string of bits”. For example, you can represent a set of letters using a bit-string. Each bit in the bit-string is associated with a letter: bit position 0 with ‘A’, bit position 1 with ‘B’, ..., bit position 25 with ‘Z’. Bit-string bits are set to ‘1’ to indicate that their corresponding letters are in the set. For example, the set { ‘A’, ‘B’, ‘D’, ‘Y’ } would be represented as:



To

determine if a specific ASCII character, say ‘C’ (67₁₀) is in the set, you would need to build a “mask” containing a single “1” in bit position 2.

- a) What instruction(s) could we use to build the mask needed for ‘C’ in \$3?
- b) If a bit-string set of letters is in register \$5, then what instruction(s) can be used to check if the character ‘C’ (using the mask in \$3) is in the set contained in \$5?
- c) If a bit-string set of letters is in register \$5 and another in \$6, then what instruction(s) can be used to calculate in \$7 the union of sets in \$5 and \$6? (i.e., all elements in either set)
- d) If a bit-string set of letters is in register \$5 and another in \$6, then what instruction(s) can be used to calculate in \$7 the intersection of sets in \$5 and \$6? (i.e., all elements in both sets)
- e) If a bit-string set of letters is in register \$5 and another in \$6, then what instruction(s) can be used to calculate in \$7 the set difference: set in \$5 - set in \$6? (i.e., all elements in the left-hand set that are not in the right-hand set)