1. (25 points) Given the following data segment declaration, what values are printed by each of the syscalls?

```
x:       .word    13
list:    .word    4,13,-5,6
tree:    .word    x,list
name:    .asciiz  "Patterson and Hennessey"
```

(a) `lw $t0,x`  (b) `la $t5,name`

```
ori $a0,$t0,0xff
li $v0,1
syscall
```

ANSWER (a): 255  ANSWER (b): rson and Hennessey

ANSWER (c): 8  ANSWER (d): 13

(c) `li $t0,4`

```
li $s1,0
li $t1,12
```

(loop) `beq $t0,$t1,done`

```
lw $s6,list($t0)
add $s1,$s1,$s6
add $t0,$t0,4
j loop
```

ANSWER (c): 8

(d) `lw $t3,tree`

```
li $v0,1
lw $s6,list($t0)
syscall
add $s1,$s1,$s6
add $t0,$t0,4
j loop
done:  li $v0,1
move $a0,$s1
syscall
```

2. (35 points) Short answer.

(a) List two reasons why programmers seldom use assembly language. List two reasons why they might need to.

ANSWER: Seldom use because hard to debug, slow to write, tedious. Do use because a compiler might not be available, a short inner loop of code may be critical for performance, it may be necessary to meet real time constraints

(b) `add`, `sub`, and `slt` all have the same op code (0). Why? How does the machine distinguish between them?

ANSWER: Because they are all performed in the ALU. By the function field.

(c) Strings can be entered into a.data segment using either the directive `.asciiz` or the directive `.ascii`. What is the difference give an example of a situation in which you would choose each alternative.
ANSWER: .asciiz terminates the string with a special character, .ascii does not. You would use .asciiz for strings that will be printed (the syscall for a print string prints from a given address until the termination character is encountered) and .ascii for substrings that will be inserted into longer strings.

(d) Assume that FOO is a procedure with 10 arguments that does not call another procedure and uses only $t registers. FOO still must manipulate the runtime stack. Explain why and how.

ANSWER: FOO must pass its parameters on the stack because there are too many parameters for $a0 - $a3. The parameters are stacked by the calling procedure and then FOO is called; when FOO starts up it loads its parameters from the top 10 positions in the stack.

(e) What is a pseudoinstruction? Give an example of one that we have been using from MIPS and show what code (in assembly language) that instruction might generate. (Your answer should be in assembly language and need not be exactly the same as the code that the SPIM assembler would produce but should be plausible.)

ANSWER: A pseudoinstruction is an assembly language instruction that does not exactly correspond to a machine language instruction but is provided as a convenience for the programmer. An example is

```
blt $t1,$t2,loop
slt $1,$t1,t2
bnez $t1,loop
```

(f) Write a .data segment definition that describes the following ternary tree structure. Assume -1 is used to represent the null pointer.

```
Tree:        100    ANSWER:   node1:   .word 100,node2,node3,node4
/ | \                  node2:   .word 50,-1,-1,-1
/ | \                  node3:   .word 200,node5,-1,node6
50 200 0    node4:   .word 0,-1,-1,-1
/ \                      node5:   .word 150,-1,-1,-1
150 250                  node6:   .word 250,-1,-1,-1
```

(g) What do we mean by sign extension? When is it useful?

ANSWER: When a signed integer of less than 32 bits (a byte or halfword) is loaded into a 32 bit register, the high order bit is filled across all of the unfilled, high order bits to preserve the numeric value of the original integer in 2s complement form.

.(25 points) Write a program segment that finds the smallest element in an array of 10 elements called LIST and then swaps that element with the first element of LIST. Thus if the original array is

```
45 56 7 8 1 57 8 10 3 27
```

it should be changed to

```
45 56 7 8 0 57 8 10 3 27
```

If there is more than one smallest value, change just one (any one) of the occurrences. Your program segment can use the following .data segment

```
.data
LIST .space 40
.globl __start
.text
__start:
```
You do not have to write the code to read values into the array, just the program segment that finds and changes the smallest values. You can assume all registers are available; use them in accordance with standard conventions.

# ANSWER: program to zero out the least element of an array

.data
list: .word 45, 56, 7, 8, 1, 57, 8, 10, 3, 27
.globl __start
.text
__start:
  la $t0,list # $t0 holds the current loc. of smallest
  move $t1,$t0 # $t2 will hold the current place
  add $t2,$t1,4 # to check
  add $t1,$t1,40 # $t1 will hold the end of list
  # loop through the array
  loop: beq $t1,$t2,done # check if reached the end
       lw $s1,0($t0) # no, load smallest value so far
       lw $s2,0($t2) # load current value to check
       bge $s2,$s1,loopagain # compare 'em
       move $t0,$t2 # new smallest found
  loopagain: add $t2,$t2,4 # increment index
    j loop # go back around
  # been through entire array ...
  done: sw $0,0($t0) # zero out lowest element

# end of required segment

4. (15 points) Recursive procedures must use the runtime stack. Assume you have written a recursive procedure called CALLEMAGAIN. CALLEMAGAIN uses registers $t0, $s0, and $s3. Write the segment of code that handles the runtime stack on entry to CALLEMAGAIN. Have it stack only those registers that are necessary. Write the corresponding segment of code that handles the exit from CALLEMAGAIN, manipulating the stack and jumping to the return address appropriately.

ANSWER, ENTRY: sub $sp,$sp,16
               sw $ra,0($sp)
               sw $t0,4($sp)
               sw $s0,8($sp)
               sw $s3,12($sp)

ANSWER, EXIT: lw $ra,0($sp)
               lw $t0,4($sp)
               lw $s0,8($sp)
               lw $s3,12($sp)
               add $sp,$sp,16