1.) (12 pts.)
Assume the following C declaration and that the starting address of 'dat' is loaded into register $a0.:

```c
int dat[10] = {4,6,1,9,3,7,2,8,5,5};
```

a.) What is the final value of $s1 when the first code snippet completes.
b.) How many instructions get executed by code snippet

```assembly
add $s1, $zero, $zero
add $t0, $zero, $zero
addi $t1, $zero, 10
NEXT:  beq $t0, $t1, DONE
sll $t2, $t0, 2
add $t3, $a0, $t2
lw $t4, 0($t3)
add $s1, $s1, $t4
sw $t4, 0($t3)
addi $t0, $t0, 1
b  NEXT
DONE:
```

c.) **True / False:** Assuming the $a0 is loaded with the same value, the following code achieves the same answer for $s1.
d.) How many instructions get executed by this code snippet

```assembly
add $s1, $zero, $zero
addi $t1, $zero, 10
sll $t1, $zero, 2
add $a1, $a0, $t1
NEXT:  lw $t4, 0($a0)
addi $s1, $s1, $t4
sw $t4, 0($a0)
addi $a0, $a0, 4
bne $a0, $a1, NEXT
```

**Submission Instructions:** Select your answers on Blackboard
Subroutines and Assembly
2.) (24 pts.) In addition to passing arguments in registers or by using the stack, another possible method (that no one would ever use but is interesting “academically” speaking) is to provide space for the arguments in the code itself (though you should note that this code will NOT assemble using MARS because it does not handle data directives in the text section). However, please study the code below to understand how this method works and answer the given questions. **SHOW ALL VALUES in HEX. Assume that the .data section starts at 0x10010000.**

a.) **Before execution of “jal AVG”,** what are the contents of $t0, $t1, and $t2.
b.) **Before execution of “sra $v0,$v0,1”,** what are the contents of $v0, $t3, & $ra?
c.) **After program execution,** what are the contents of $ra and the word at address RES.

**Submission Instructions:** Enter the hex values into the appropriate blanks.

```assembly
.data
VALS: .half 0xbead, 0xface
RES: .space 4
.text
la $t0, VALS
lh $t1, ($t0)  a.) $t0 = 0x_____
lu $t2, 2($t0)  $t1 = 0x_____
sll $t2, $t2, 16  $t2 = 0x_____
or $t2, $t1, $t2
jal AVG
 lw $t4, RES  b.) $v0 = 0x_____
.word -2  $t3 = 0x_____
.word -6  $ra = 0x_____
sw $v0, ($t4)
li $v0, 10
syscall
AVG: lw $v0, ($ra)  c.) $ra = 0x_____
lw $t3,4($ra)  M[RES] = 0x_____
add $v0,$v0,$t3
sra $v0,$v0,1
addi $ra,$ra,8
jr $ra
```

```
3.) (44 pts.) Consider the following assembly code.
   a.) Show the word content (in hex) of registers $t0,$t1,$sp and the PC:
      I. Just after execution of the jal instruction at location L1
      II. Just after the first execution of the jal instruction at L2
      III. Just after the first jr $ra instruction execution
      IV. After execution of the li $v0,10 instruction
   b.) Show the contents of the stack at its fullest (fill unused locations with 0’s).
   c.) What arithmetic function is this code performing?

   Submission Instructions: Enter the hex values for registers and memory (include the 0x).
   For part c., describe the operation and we will grade it manually.

<table>
<thead>
<tr>
<th></th>
<th>$t0</th>
<th>$t1</th>
<th>$sp</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>I)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>II)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>III)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b.) Initially
   $sp = 0x7ffefffc -->

   |         |        |        |          |
   |         |        |        | 0x7ffeff8 |
   |         |        |        | 0x7ffeff4 |
   |         |        |        | 0x7ffeff0 |
   |         |        |        | 0x7ffeefec|
   |         |        |        | 0x7ffefe8 |
   |         |        |        | 0x7ffefe4 |
   |         |        |        | 0x7ffefe0 |
   |         |        |        | 0x7ffeefdc|

c.) What arithmetic function or operation is this code performing (Hint: try $t1 = 32 or 8$)?
Stack Frames
4.) (20 pts.) Examine the C-style subroutines and the actual “gcc” MIPS assembly output below. Assume that we start execution at the beginning of the “caller” routine. Further, assume that initially, $sp = $fp = 0x7fffeffc, $ra=0x00400018, $gp=0x10018000.

Further assume that i=2 & j=9 for the arguments passed to “caller” and the return address for the call to AVG is 0x0040ce18. Follow the assembly code and fill in the stack data values (in hex) that this trace would create. Show the stack values after completion of the code.

Because of how “gcc” works, the stack frames are a bit more bloated than they need to be and many locations will go unused. Fill unused locations with 0’s. Try to understand the correspondence of variables to their particular locations in the stack frame.

```c
void caller(int i, short j)
{
    unsigned int a=5;
    a = avg(a,3);
}
unsigned int avg(unsigned int x, unsigned int y)
{
    int temp;
    temp = (x+y)/2;
    return temp;
}
```

```assembly
caller:
    subu $sp,$sp,48
    sw $31,40($sp)
    sw $fp,36($sp)
    sw $28,32($sp)
    move $fp,$sp
    sw $4,48($fp)
    li $2,5 # 0x5
    sw $2,24($fp)
    lw $4,24($fp)
    li $5,3 # 0x3
    la $25,avg
    jalr $25
    sw $2,24($fp)
    move $sp,$fp
    lw $31,40($sp)
    lw $fp,36($sp)
    addu $sp,$sp,48
    jr $31

avg:
    subu $sp,$sp,24
    sw $fp,20($sp)
    sw $28,16($sp)
    move $fp,$sp
    sw $4,24($fp)
    sw $5,28($fp)
    lw $3,24($fp)
    lw $2,28($fp)
    addu $2,$3,$2
    srl $2,$2,1
    sw $2,8($fp)
    lw $2,8($fp)
    move $sp,$fp
    lw $fp,20($sp)
    addu $sp,$sp,24
    jr $31
```
Submission Instructions: Enter the hex values for each memory location (include the 0x). Fill unused locations with 0’s.

<table>
<thead>
<tr>
<th>$sp--&gt;$</th>
<th>0x7ffeффc</th>
<th>0x7ffeфф8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0x7ffeфф0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффe8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффe0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффd8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффd0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффc8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффc0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффb8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x7ffeффb0</td>
</tr>
</tbody>
</table>