1. Examine the following snippet of MIPS code. (Assume register $a0 is initialized with the base address of an array of words in memory.)

addi $t0, $zero, 0
addi $t2, $zero, 0
addi $t1, $zero, 5
L: add $t3, $a0, $t2
sw $t1, 0($t3)
addi $t2, $t2, 4
addi $t0, $t0, 1
slti $t4, $t0, 100
bne $t4, $zero, L
done:

(a) Comment this code snippet.
(b) Indicate the contents of the five temporary registers ($t0 - $t4) when the “done” label is reached.
(c) Describe what this bit of code is doing. You may use either a couple English sentences or pseudocode.

Solution:

a) From the given snippet we can conclude the following:

$t0 <= \text{counter}, t2 <= \text{index}, t1 <= \text{value}, t3 <= \text{address of an element}$

addi $t0, $zero, 0  # zero stored in t0
addi $t2, $zero, 0  # zero stored in t2
addi $t1, $zero, 5  # value 5 stored in t1
L: add $t3, $a0, $t2  # $t3 = $a + $t2 where $t3$ is a pointer to the next address relative to $a0$
sw $t1, 0($t3)      # store 5 into next array location i.e array[$t3]=5
addi $t2, $t2, 4   # $t2 = $t2 + 4 increasing the address relative to $a0$
addi $t0, $t0, 1   # $t0 = $t0 + 1 counter is increased by 1
slti $t4, $t0, 100  # if ($t0 < 100) => $t4=1 else => $t4=0
bne $t4, $zero, L   # if $t4 != 0 , branch to L, if counter is less than 100, continue looping
done:  # ”done” label

(b) Indicate the contents of the five temporary registers ($t0 - $t4) when the “done” label is reached.
$t0=100
\$t1=5 \\
\$t2=400 \\
\$t3=\$a0 + 396 \rightarrow \text{address of element a[99]} \\
\$t4=0 \\

(c) Describe what this bit of code is doing. You may use either a couple English sentences or pseudo-code.

The code fills the first 100 elements of an array with the value “5”. i.e.

```c
int array[100]; //each array element is 4 byte long hence “int”
for(i=0;i<100,i++){
    array[i]=5
}
```

2. Write MIPs instructions to implement the following code snippet. Assume that amount is stored in \$s0 and fee is stored in \$s1.

```
switch (amount) {
    case 20: fee=2; break;
    case 50: fee=3; break;
    case 100: fee=5; break;
    default: fee=0;
}
```

Solution:

The above function if seen are nothing but a bunch of if statements and hence can be treated like 3 if statements as follows:

```c
    {
        fee=0;
        if(amount== 20)
            fee=2;
        if(amount== 50)
            fee=3;
        if(amount== 100)
            fee=5;
    }
```

The above can be written in MIPS code as follows:

```
addi \$t0, \$zero, 20
addi \$t1, \$zero, 50
addi \$t2, \$zero, 100
beq \$s0, \$t0, TWENTY      # jump if \$0 equals \$t0 (20)
beq \$s0, \$t1, FIFTY      # jump if \$0 equals \$t1 (50)
```
beq $s0, $t2, HUNDRED  # jump if $0 equals $t0 (100)
addi $v0, $s1, 0
j END  # not equal to 20, 50 or 100 and hence END

TWENTY:
    addi $v0, $s1, 2
    j END

FIFTY:
    addi $v0, $s1, 3
    j END

HUNDRED:
    addi $v0, $s1, 5
END:

3. Write MIPS instructions to implement the following small application.

```c
int main() {
    int y;
    y = average(2,3,4,5);
}
```

```c
int average(int f, int g, int h, int i) {
    int sum = f + g + h + i;
    int avg = sum / 4;
    return avg;
}
```

Solution:

The simplest way to approach this problem would be to look at the “average” function and figure out its requirements. It will need four int variables f, g, h and i and also the return address to jump back to the next instruction after it was called.

In order to do the above we store the four values i.e. f, g, h and i in $a0, $a1, $a2 and $a3. Also the return address is in the stack created each time the function is called.

The following is a simple MIPS code:

```mips
MAIN:
    addi $a0, $zero, 2
    addi $a1, $zero, 3
    addi $a2, $zero, 4
```
addi $a3, $zero, 5        #initializing the parameters to be passed
jal AVERAGE             #Calling AVERAGE function
addi $s0, $v0, 0        #average of f, g, h, i is stored in ‘$s0’
jr $ra                  #Return from main i.e. exit the program

AVERAGE:
addi $sp, $sp, -4       #Save $s0 on stack
sw $s0, 0($sp)          
add $t0, $a0, $a1
add $t1, $a2, $a2
add $t3, $t0, $t1      #f+g+h+i
srl $v0, $t3, 2         #Division by 2 is equivalent to shifting to right by 2
lw $s0, 0($sp)
addi $sp, $sp, 4       #Restore $s0
jr $ra                  #Return to main