1 Problems

1. **Understanding MIPS programs**

```
flathead:
     addi $t0, $a2, 1
loop:
     bge  $t0, $a1, exit
     mul  $t1, $t0, 4
     add  $t1, $t1, $a0
     lw   $t2, 0($t1)
     sub  $t1, $t1, 4
     sw   $t2,  0($t1)
     addi $t0, $t0, 1
     j loop
exit:
     jr   $ra
```

   (a) Translate the `flathead` function above into a high-level language like C or Java.
   Solution:
   This function does stuff with `$a0`, `$a1` and `$a2`, which indicates that there are three arguments. Multiplying (`$a2` +1) by 4 suggests that `$a2` is an index, and adding that to `$a0` for a "`lw`" means `$a0` is an array of 32-bit values (most likely integers or floating-point numbers). The loop exits when `$t0` $\geq$ `$a1`, suggesting that `$a1` is some kind of maximum index. Since nothing is saved in `$v0` before the "`jr`", there is no return value.
   The problem says not to use gotos, so you should have translated the loop instructions into a higher-level "for" or "while" structure. You should also show array manipulations, instead of pointer arithmetic and dereferences.

```
void flathead(int a0[ ],int a1,int a2)
{
     int t0 = a2 + 1;
     while (t0 < a1) {
          a0[t0 - 1] = a0[t0];
          t0++;
     }
}
```

   (b) Describe briefly, in English, what this function does.
   Solution:
   This function takes an array `$a0` and shifts elements (`$a2` + 1) through `$a1` into positions `$a2` through `$a1` − 1. In other words it shifts those elements "downward" or "leftward". You can also think of this as deleting element `$a2`.

2. Compute the result of the following expression and write it in hexadecimal notation:

```
0xdeadbeef | (0x81 << 22)
```

   0x81 = 1000 0001, shifted left 22 bits gives you: 0010 0000 0100 0000 0000 0000 0000 0000

   Since in the above bits are only set in the 1st and 3rd hexidecimal characters, those are the only ones that change (recall that ORing with 0 preserves values).

```
   1101   E  1010   D     B    E    E    F
OR 0010 0000 0100 0000  0000 0000 0000 0000
 = 1111   E  1110   D     B    E    E    F   = FEEDBEEF
```

3. **C to MIPS translation**
   C Code provided for your reference:

```
int count(int A[], int n)
{
     int i;
     int num = 0;

     for(i = 0; i < n; i++){
          if (test(A[i]) == 1) {
               num++;
          }
     }
     return num;
}
```

Solution
The hard part here is preserving registers correctly. The `count` function acts as both a caller and callee. It has to save all the callee-saved registers that it uses and restore them before returning, but it is also responsible for preserving any caller-saved registers that it needs across the call to test. Regardless of whether you use the caller-saved registers, callee-saved registers or a combination of them, you'll have to push stuff onto the stack.

One example implementation is shown on the next page. It uses caller-saved registers for the temporary values `i` and `num`, so those must be saved and restored before and after the call to test. In addition the base address of the array and the array size, which are passed as arguments `$a0` and `$a1`, must also be saved. Finally, recall that the callee-saved register `$ra` has to be preserved because of the nested `jal`.
Translated MIPS code:

```
count:
     subu $sp, $sp, 20
     sw   $ra, 0($sp)      # Save $ra before jal
     li   $t0, 0           # $t0 = i
     li   $t1, 0           # $t1 = num
loop:
     bge  $t0, $a1, exit   # while (i < n)
     sw   $t0, 4($sp)      # Save caller-saved registers
     sw   $t1, 8($sp)      # that we need
     sw   $a0, 12($sp)
     sw   $a1, 16($sp)
     mul  $t2, $t0, 4      # We don't need to save $t2
     addu $t2, $a0, $t2
     lw   $a0, 0($t2)      # $a0 = A[i]
     jal  testfunc         # $v0 = testfunc(A[i])
     lw   $t0, 4($sp)      # Restore saved registers
     lw   $t1, 8($sp)
     lw   $a0, 12($sp)
     lw   $a1, 16($sp)
     beq  $v0, $0, next
     addi $t1, $t1, 1      # num++
next:
     addi $t0, $t0, 1      # i++
     j    loop
exit:
     move $v0, $t1         # return num
     lw   $ra, 0($sp)      # Restore $ra
     addiu $sp, $sp, 20
     jr $ra
```