ECE 232: Solutions to Test 1

(1) The obvious way to do this is to construct a table listing all 8 combinations of \( A_i, B_i, C_i \), showing that you get the same result for \( C_{i+1} \) whether you define \( P_i = A_i \oplus B_i \) or \( P_i = A_i + B_i \).

A more elegant solution would point out that the only case in which \( A_i \oplus B_i \neq A_i + B_i \) is when \( A_i = B_i = 1 \). But in that case, \( G_i = 1 \) meaning that \( C_{i+1} = 1 \), irrespective of the value of \( P_i \).

Despite what was said in the question, a number of students tried to prove this by providing a few examples, which is not what we wanted. A proof has to be complete and clearly stated: so we were looking for either a truth table or an argument like the one in the preceding paragraph which covers all cases.

(2-4) These questions are direct implementations of the algorithms studied in class: see the examples in the text.

In Question 3, a common mistake was to subtract the bias instead of adding. Note that in this question, you were asked to take a given number and translate it into biased form (not the other way around). We know that if the biased form has an exponent value of \( E_B \), the actual value of the exponent is \( E_{\text{actual}} = E_B - \text{Bias} \). From this, we have \( E_B = E_{\text{actual}} + \text{Bias} \).

In Question 4, some students forgot the reason why we have biased notation at all. Biased notation is meant to ensure that the exponent field can be treated as an unsigned integer so that comparisons between the exponents of different numbers are easy. So, if the result of the biased value is negative, it is either a denormalized number or an underflow (i.e., too small for even a denormalized number).

A few students also multiplied the sign bits! Obviously, the sign bit of the product is the Ex-OR of the sign bits of the multiplier and multiplicand.

(5) There were different versions of this question, which differed only in the constants used. (For example, one version had \( i = i + 19 \), another had \( i = i + 6 \), etc.) I have provided a solution to just one version: for the others, simply substitute the appropriate constant.

The version used here is:

```c
i = 0;
while (a[i] < 75) {
    i = i+19;
    b[i] = b[i] + 35;
}
```

It is good practice to prepare a table of register allocations: this reduces the chances of making a mistake. Here are the register allocations we use in our program:
The program is as follows.

```
add $s2, $zero, $zero # Set i=0
add $t0, $zero, $zero # We will store 4*i in $t0
L1    
add $s3, $s0, $t0     # Form the address of a[i]
lw $t1, 0($s3)    # Load a[i] into register $t1
slt $t2, $t1, 75   # Set $t2=1 if a[i]<75
beq $t2, $zero, Exit # if a[i]>=75 leave the loop
add $s2, $s2, 19  # i=i+19
add $t0, $t0, 78  # 4*i must be incremented by 19*4=78
add $t3, $s1, $t0     # Form the address of b[i]
lw $t4, 0($t3)    # Load b[i] into $t4
add $t4, $t4, 35  # b[i]=b[i]+35
sw $t4, 0($t3)    # Store b[i] into memory
j L1               # Go back to the start of the loop
Exit ....
```

I said in the question that $s2 would hold i: that is why I ended up having to find another register to store 4i, instead of using $s2 itself for that purpose.

Common mistakes were:

- Saving and then restoring registers. This is not a function call, so we don't need to do this. If this fragment of code is within some function, then the registers will be saved at the point of entry into the function, not at the beginning of this code fragment.

- Using the wrong jump instruction to loop back: some people used jal for example. This would overwrite the $ra register (which may, for all we know, already be holding an address that will later be needed).
• Forgetting to account for the fact that a word is four bytes. The correct addresses is actually $s0+4i$ for $a[i]$ (not $s0+i$ and similarly for $b[i]$).

• Forgetting that $i$ needs to be incremented before loading $b[i]$.

• Placing the check for $a[i]<75$ at the end, rather than towards the beginning. If you place the check at the end, the first iteration of the loop will always be executed, even if $a[0]>75$.

I also noticed occasional mistakes in the lw and sw syntax, which was odd, considering that a table of instructions (including the syntax definition) was attached to the test.