Please answer questions 1 and 2 on this paper and put all other answers in the blue book.

1. True/False. Please circle the correct response.
   
   a. T  In the C and assembly calling convention that we used for this class, arguments to a function call are pushed in reverse (i.e. right-to-left) order.
   
   b. F  In C, for the expression (x | THE_MASK), where THE_MASK has at least one bit that is not zero, the result will be zero if all the bits of x are zero.
   
   c. T  In x86 assembly, the instruction mov eax, [ebp+12] (or movl 12(%ebp), %eax in AT&T syntax) will add 12 to the value contained in the ebp register in order to compute a memory address.
   
   d. T  In x86 assembly on a 32-bit machine, the instruction pop eax adds 4 to the value of the esp register.
   
   e. T  In assembly, performing a right arithmetic shift operation, SAR, on a negative number results in a negative number.
   
   f. T  Any circuit constructed using only AND, OR, or NOT gates can also be constructed using only NAND gates (which perform an AND and then a NOT).
   
   g. T  Branch prediction is used to predict which way a conditional branch (jump) instruction will go in order to avoid pipeline stalls.
   
   h. F  A multiplexer uses 2^N select lines to select from N input lines.
   
   i. T  A 32-bit adder can be used for subtraction if each bit of the second operand is first sent through a (32-bit) NOT gate and the carry-in to the adder is set to 1.
   
   j. T  In an unclocked latch, setting both the S and the R inputs to 0 will cause the output Q to retain its current value.

2. Write the number AB31 hex in binary: 1010 1011 0011 0001.
   b. Write the number 73 decimal in hex: 49 and in binary: 0100 1001.
   c. log 16G = __34__.
   d. In order to access all bytes in a 64MB memory, an address must have at least __26__ bits.

3. Write a C procedure, int foo(int x), that returns the index of the most significant bit of x whose value is 1. For example, if bit 15 of x is the most significant bit of x whose value is 1, then foo should return 15. If no bits are 1, then foo should return -1.

   ```c
   int foo(int x)
   {
       int i;
       for(i=31; i>=0; i--) {
           if (x & (1<<i))
               return i;
   ```
4. Write an x86 assembly procedure `bar` that takes two parameters, an integer `n` and a pointer `p` to an integer array (i.e. `p` is the address of the start of the array), and adds up the first `n` integers of the array and returns the result.

**Intel Syntax**

```
_bar:  
push   ebp
      mov   ebp,esp

      mov   eax,0       #result is initially 0
      mov   ecx,[ebp+8] #ecx holds n
      mov   edx,[ebp+12] #edx holds p

        TOP:
      cmp   ecx,0       #if n=0
      je    DONE  #jump out of loop
      add   eax,[edx]   #result = result + *p
      dec   ecx         #n--
      add   edx,4       #p++
      jmp   TOP  #jump to top of loop

DONE:
      pop   ebp
      ret
```

**AT&T Syntax**

```
_bar:  
push   %ebp
      mov   %esp,%ebp

      mov   $0,%eax       #result is initially 0
      mov   8(%ebp),%ecx  #ecx holds n
      mov   12(%ebp),%edx #edx holds p

        TOP:
      cmp   $0,%ecx       #if n=0
      je    DONE  #jump out of loop
      add   (%edx),%eax   #result = result + *p
      dec   %ecx          #n--
      add   $4,%edx       #p++
      jmp   TOP  #jump to top of loop

DONE:
      pop   %ebp
      ret
```
5.  
   a. Write in C a recursive version of the bubble sort procedure (hint: it should take as 
      parameters an array a[] and an integer size that gives the size of the portion of the array 
      that is currently unsorted).

```c
void bubble(int a[], int size)
{
    if (size <= 1)
        return;
    int i;
    for(i=0;i<size-1;i++)   //bubble the largest element to the end
        if(a[i] > a[i+1]) {
            int temp = a[i];
            a[i] = a[i+1];
            a[i+1] = temp;
        }
    bubble(a, size-1);     //now sort the rest of array
}
```

   b. Write the recurrence relation that gives the running time, T(n), of the bubble sort 
      algorithm.

   T(1) = 1
   T(n) = T(n-1) + (n-1)

   c. Solve the recurrence relation in order to determine the asymptotic complexity, expressed 
      in “big-Oh” notation, for bubble sort.

   T(n) = T(n-1) + (n-1) = T(n-2) + n-2 + n-1 = T(n-3) + n-3 + n-2 + n-1 = 1 + 2 + ... + n-1
   = n(n-1)/2 = O(n^2)

6.  
   a. Build, from AND, OR, and NOT gates, a circuit that represents the two-bit “less-than” 
      function. That is, it has two two-bit inputs, A and B, and a single one-bit output, R, such 
      that R is true when A < B. [Hint: enumerate the possible inputs for which the output is 
      true.]

   One way to do this is to write out the truth table, where the four inputs are A1 A0 B1 and 
   B0, then generate the Boolean formula for those rows where the result is 1, simplify if 
   possible, and then build the circuit according to the Boolean formula. Another way is to 
   notice:
   If A = 00, then A < B if either bit of B is 1
   If A = 01, then A < B if B1 is 1
   If A = 10, then A < B if B=11

   Encoding the above observations as a Boolean formula gives:
   R = ((NOT A1) AND (NOT A0) AND (B1 OR B0)) OR
      ((NOT A1) AND A0 AND B1) OR
      (A1 AND (NOT A0) AND B1 AND B0)
   Although this could be simplified a bit further, it is easy enough to build as is:
b. As you saw in class, a clocked latch is built from an unclocked latch as shown below. Why are flip-flops used for storing bits in a CPU rather than clocked latches?

With a clocked latch, the value being stored (and output at Q) can change as long as the clock is up. In a system, such as the one illustrated below,

where the data stored in a sequential circuit is being used as input to a combinational circuit (such as an ALU) and the output of the combinational circuit is being sent back to the combinational circuit for storage, if the sequential circuit consisted of clocked latches then, as long as the clock was up, the data coming in to the sequential circuit could overwrite the data that was stored – while the combinational circuit is still performing its computation. What is needed instead, and provided by flip-flops, is for the data coming in to the sequential circuit to be prevented from overwriting the old data until the clock falls (and presumably, the combinational circuit has completed the computation).

c. Build from gates, multiplexers, decoders, and/or adders (including 32-bit versions of each) a circuit that takes two 32-bit inputs, X and Y, and outputs the value of the larger of X and Y. That is, it computes the equivalent of the following C code:

\[
\text{output} = X > Y ? X : Y;
\]
The comparison is accomplished by performing a subtraction, \( Y - X \), and then testing if the result is negative by using the highest bit of the result as the selector for a multiplexer whose inputs are \( X \) and \( Y \).

![Diagram](image)

7. As you probably remember from the project, the MIPS jalr instruction (jump-and-link, MIPS' version of the call instruction),

\[
jalr \ $rs
\]

has the following effect:

\[
\text{register}[31] = \text{pc} + 4 \\
\text{pc} = \text{register}[rs]
\]

Build the data path for this instruction. Assume that the MIPS register file, the PC register, and any adder or ALU you want already exists (i.e. you don't need to build any of these, you can use them). Show all the wires that are involved in executing the instruction and be sure to the label the values that are carried on any wires that you draw.