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. F In the calling convention we discussed in class, all parameters were passed on the stack and the return value was returned on the stack.
   b. T In C, on a 32-bit x86 machine, the expression (1 << 31) results in a negative integer.
   c. T In assembly, performing a right arithmetic shift operation, SAR, on a negative number results in a negative number.
   d. F In x86 assembly, CALL FOO and JMP FOO are equivalent.
   e. F Caches on CPU chips were developed in order to compensate for the slow access to data on a disk.
   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 Pipeline stalls are unrelated to cache misses.
   i. T In a register file, the number of bits carried by each Read Select input is the log of the number of registers in the register file.
   j. T In a register file, the number of bits carried by the Write Data line is the number of flip-flops contained within each register in the register file.

2. Short Answer. Fill in the answer on this sheet.
   a. Give a value for a and for b which will cause the two procedures below to return different results. Answer: \( a = 1 \quad b = 2 \) (if there are none, write NONE).

   ```
   typedef int BOOL;
   typedef int BOOL;
   
   int foo(BOOL a, BOOL b) {
     if (a && b) return 1;
     else return 0;
   }
   
   int bar(BOOL a, BOOL b) {
     if (a & b) return 1;
     else return 0;
   }
   ```

   b. Why is it important for the size of a direct-mapped cache to be a power of two?

   Answer: So that computing the \((\text{address MOD size\_of\_cache})\) operation entails simply selecting some of the bits of the address.
3.
   a. What does the XOR (exclusive-OR) instruction in x86 assembly do?
   **Performs the bitwise exclusive-OR of its operands, where for each bit, the result is 1 if exactly one of the two inputs is 1.**
   b. What is the result of executing the instruction
      \[
      \text{xor} \quad \text{eax, eax} \quad (\text{xor} \ %\text{eax}, %\text{eax} \text{ in AT&T syntax})
      \]
   **eax is set to zero.**
   c. Draw a circuit to compute the XOR of two inputs using only AND, OR, or NOT gates.
   ![XOR Circuit](image)
   d. Write a sequence of x86 instructions to swap the values of the eax and edx registers, using only xor instructions and no other registers or memory locations. Prove that your answer is correct (you may assume that xor is associative and commutative).

**See the comments for the proof.**

\[
\begin{align*}
\text{xor} \quad \text{eax, edx} & \quad \# \text{now eax holds (A XOR D)} \\
\text{xor} \quad \text{edx, eax} & \quad \# \text{now edx holds (D XOR (A XOR D)) = ((D XOR D) XOR A) = A} \\
\text{xor} \quad \text{eax, edx} & \quad \# \text{now eax holds ((A XOR D) XOR A) = ((A XOR A) XOR D) = D}
\end{align*}
\]

4.
   a. Write the shortest complete C program you can that will crash with a “Segmentation fault” or “bus error” (depending on the system you are using).
   **These crashes are due to dereferencing an invalid address.**

```c
main()
{  int *p = 0;
    (*p)++;
}
```
   b. On a machine with a 16MB four-way set-associative cache where each cache entry is one word (32 bits), write a short C program that will generate a large number of conflict cache misses.

   To generate conflict cache misses on a machine with a four-way set-associative cache, one could sequentially access five memory locations that map to the same set. In this case, since each set holds four words, and each word is 4 bytes, there are 1M sets. Thus, addresses that are 1M words apart will map to the same set. Declaring 5 arrays that are 1M words apart, then accessing the corresponding elements of the 5 arrays will result in a conflict cache miss.
int main()
{
    int a[1<<20], b[1<<20], c[1<<20], d[1<<20], e[1<<20];
    int i, j;
    j = j + a[i] + b[i] + c[i] + d[i] + e[i];
    j = j + a[i];  //conflict miss here.
}

5. Suppose that on a computer with a primary (L1) cache and secondary (L2) cache, the following holds:
   • the primary cache has a 90% hit rate for both data and instructions,
   • in cases where there is a primary cache miss, the secondary cache has a hit rate of 96% for both data and instructions,
   • if an instruction does not cause a cache miss, executing the instruction takes one cycle,
   • in the case of a primary cache miss but a secondary cache hit, the penalty is 10 cycles,
   • in the case of a primary cache and secondary cache miss, the penalty for having to access main memory is 200 cycles

What is the overall execution time (in cycles) of a program that executes N instructions, where 25% of the instructions access data in memory?

The number of instruction fetches that miss L1 cache but hit L2 cache = (N * .1 * .96)
The number of instruction fetches that miss both L1 and L2 cache = (N * .1 * .04)
The number of data fetches that miss L1 cache but hit L2 cache = (.25N * .1 * .96)
The number of data fetches that miss both L1 and L2 cache = (.25N * .1 * .04)

Thus, the total number of cycles is:
= 3.2N

6. Build a circuit with three inputs and one output, where the output is 1 if an odd number of the inputs is asserted (carry a 1 value) and the output is 0 if there is an even number of asserted inputs. For example, if exactly one of the inputs carries a 1, then the output would be 1 but if exactly two of the inputs carry a 1, the output would be 0.
7. a. The following x86 code will cause pipeline stalls on the simple pipelined processor discussed in class. Indicate which of the instructions will cause a stall and why. You don’t need to show the actual cycle-by-cycle execution of the instructions.

```assembly
#Intel  #AT&T
mov  eax,1     mov  $1,%eax
add  eax,edx   add  %edx,%eax    #stall, waiting
                      #for value in eax
add  [ebp+8],7   add  $7,8(%ebp)
mov  ecx,[ebp+8] mov  8(%ebp),%ecx #stall, waiting for
                      #value in [ebp+8]
sub  eax,ecx    sub  %ecx,%eax   #stall, waiting for
                      #value in ecx
xor  ebx,ebx    xor  %ebx,%ebx
```

b. Re-arrange the above code (but don’t eliminate any instructions) so that it produces the same result but causes the fewest stalls possible.

```assembly
#Intel  #AT&T
mov  eax,1     mov  $1,%eax
add  [ebp+8],7   add  $7,8(%ebp)
add  eax,edx   add  %edx,%eax
mov  ecx,[ebp+8] mov  8(%ebp),%ecx
xor  ebx,ebx    xor  %ebx,%ebx
sub  eax,ecx    sub  %ecx,%eax
```