1 CMOS logic (4 points)

The foo logic gate has 3 inputs A, B and C, one output X, and its CMOS schematic is represented on Figure 1.

1. Write its truth table, that is, a table in which you give the value of the output for each combination of the inputs.
2. Write the boolean equation of the X output of foo using the \texttt{NOT}, \texttt{AND} and \texttt{OR} operators and parentheses. Do not assume any precedence between the boolean operators, use parentheses to make your equation non ambiguous. Example (of course this is not the correct answer): \(X = ((A \text{ AND} (\text{NOT} (C \text{ OR} B))) \text{ OR} (C \text{ AND} A))\).
3. When assembling logic gates to design a circuit it is convenient to use graphical symbols instead of the CMOS schematics. You already know the
graphical symbols of several logic gates (inverter, 2-inputs NAND, 2-inputs NOR, 2-inputs AND...). Imagine a graphical symbol for the foo logic gate and draw it.

2 Binary representation of numbers (4 points)

There are several ways to represent signed integers using bits. In computer systems, the two most frequently encountered are sign and magnitude and two’s complement. In the following we denote $a_{n-1}a_{n-2}...a_1a_0$ the $n$-bits representation of integer $A$. In both representations $a_0$ is the Least Significant Bit (LSB). In sign and magnitude $a_{n-1}$ is the sign bit. In two’s complement $a_{n-1}$ is the Most Significant Bit (MSB) and has negative weight.

1. Consider integer values 56, 28 and -61 (in base 10). We want to represent these 3 values in base 2 using the sign and magnitude representation, on the same number $m$ of bits (sign bit included). What is the minimum value of $m$ (only one answer expected)?

2. Consider integer values 56, 28 and -61 (in base 10). Represent them in $m$-bits sign and magnitude (where $m$ is your unique answer to the preceding question).

3. A $p$-bits adder is a hardware device that takes two $p$-bits inputs, adds them as if they were unsigned integers, and outputs the $p + 1$-bits result. Example: if $p = 3$ and the inputs are 101 (5 in base 10) and 011 (3 in base 10), a 3-bits adder outputs 1000 (8 in base 10). Suppose that we consider the two inputs of a $p$-bits adder not as unsigned but as signed in two’s complement representation. Is the output of the $p$-bits adder always the correct two’s complement representation of their sum? If yes, explain why. If not, explain why and propose a way to fix the output of the $p$-bits adder such that it becomes the correct two’s complement representation of the sum.

4. What is the pentadecimal (base 15, digits 0, 1, 2,..., 9, A, B, C, D, E) representation of decimal value 407?

5. Consider integer values 56, 28 and -61 (in base 10). We want to represent these 3 values in base 2 using the two’s complement representation, on the same number $n$ of bits. What is the minimum value of $n$ (only one answer expected)?

6. Consider integer values 56, 28 and -61 (in base 10). Represent them in $n$-bits two’s complement (where $n$ is your unique answer to the preceding question).

3 Branch prediction (4 points)

A processor implements the RV32I Instruction Set Architecture (RISC-V, 32 bits, no extension). It is equipped with a branch prediction unit based on the Variant of the Saturating Counter (VSC) branch predictor which diagram is represented on Figure 2.
ST, WT, WN and SN represent Strong Taken, Weak Taken, Weak Not taken and Strong Not taken, respectively. The transition labels t and n represent the actual branch outcomes taken and not taken, respectively. The VSC branch predictor predicts taken when it is in states ST or WT and not taken when it is in states SN or WN.

Consider the following function written in RV32I assembly code (with pseudo-instructions):

1. Explain what the hw function does.
2. Assume our processor executes the hw function and a VSC branch predictor is used to predict the outcome of the bne branch instruction (Line 8). Assume also that the VSC branch predictor is in the ST state when entering the function. Calculate the Misspredictions Per Branch Instruction (MPBI) of the VSC branch predictor at the end of the hw function. Warning: this depends on the value in register a0 when the function is called; consider all cases.

4 Caches (4 points)

Definitions and notations

- The breakdown of a data structure is the partitioning of the data structure in separate fields.

\[1\] The MPBI is the ratio of the number of times the branch instruction was wrongly predicted over the total number of times it was executed. It is a real number between 0 and 1; 0: excellent, 1: catastrophic.
• Extra cache information: control and management information stored in the cache (tags, flags, replacement policy information...)
• Net cache data: copies of memory data stored in the cache, that is, everything except extra cache information.

Questions

We consider a 64-bits computer system with 48 bits byte addresses and 64-bits addressing units (that we call double-words in the following). The computer is equipped with a write-through data cache, 5-ways set associative and 8 double-words per line. The total size of the net cache data of 327680 bytes. Ignore the replacement policy.

1. What is the breakdown of a 48-bits address? For each field specify its bit-width, the indexes of its leftmost and rightmost bits, and explain what its role is. Number the bits as usual: bit 0 is the Least Significant Bit (LSB) and bit 47 is the Most Significant Bit (MSB).
   Example: Bits 12 ... 7 (6 bits): this field blah blah ...

2. What is the breakdown of a cache entry? For each field specify its bit-width and explain what its role is.

3. What is the total size of the cache (extra cache information plus net cache data)?

5 RISC-V 5-stages pipeline (4 points)

A processor implements the RV32I Instruction Set Architecture (RISC-V, 32 bits, no extension). It has the same 5-stages pipelined architecture as seen in class (Fetch, Decode, Execute, Memory, Write-back).

Consider the following clz function written in RV32I assembly code without pseudo-instructions:

```
1 clz : 
2 lw t0 ,0(a0) # t0 <- mem[a0] 
3 andi t1 ,t0 ,1 # t1 <- t0 AND 1 
4 beq t1 ,zero ,even # if t1==0 goto even 
5 add t1 ,t0 ,t0 # t1 <- t0+t0 
6 addi t0 ,t0 ,1 # t0 <- t0+1 
7 addi t0 ,t0 ,1 # t0 <- t0+1 
8 even : 
9 srl t0 ,t0 ,1 # t0 <- t0>>1 
10 sw t0 ,0(a0) # mem[a0] <- t0 
11 jalr zero ,0(ra) # return
```

We run it on our pipelined processor. We assume that the branch prediction unit predicts the branch and jump instructions perfectly and that all fetched instructions are indeed the ones that should be fetched (no need to kill). If there were no pipeline hazards the total execution time of the function would be 13 clock cycles: during clock cycle number 1 the first instruction would in Fetch, during clock cycle 9 the last instruction would be in Fetch and during clock cycle 13 it would be in Write-back (that is, 9 instructions plus 4 clock cycles to let
the last instruction leave the pipeline). To measure the performance overhead due to pipeline hazards we use this 13 clock cycles duration as a reference. We say that a pipeline hazard costs 3 clock cycles if in order to handle it we use a technique that increases the total execution time by 3 clock cycles.

1. Identify all data hazards. For each hazard give the involved register and pair of instructions (use the line numbers).

2. Suppose our pipelined architecture handles the data hazards only with stalls (no bypass logic). How many clock cycles will be added to the reference (13) due to data hazards while executing \texttt{clz}? Warning: this depends on the outcome of the \texttt{beq} instruction (line 4); answer for each of the two cases.

3. Suppose our pipelined architecture handles the data hazards with stalls and the bypass logic we saw in class (Execute-to-Decode, Memory-to-Decode and Write-back-to-Decode). How many clock cycles will be added to the reference (13) due to data hazards while executing \texttt{clz}?