Midterm 1 Report

Group Name: Papermates

October 30, 2017

Group Members (StudentID):Avik Banerjee (40166093), Davis Furukawa (78202260), Eric Cai(66016389), Eric Rodriguez(53173240), Monish Ramadoss(69514430)

We have three inputs in total. Two of these take a 32-bit number as an input. Since most of these functions require two numbers as inputs, we need two 32-bit logic input vars which we call A and B. Our third input is for our op code. The op code acts as a selector and each of its bits references a separate function. These op codes are 4 bits in length. The number of outputs on the other hand is five. These are our carry, overflow, sign, and zero flags plus our general resulting output.

Our carry flags are required for all of our addition based functions since there can be carries if two 1's are added together in the leftmost bit of the two vars resulting in an extra one carried over to the next column.

Our overflow flags which, also account for underflow, are applied to our arithmetic operators of addition and subtraction where if there is an answer from the operation that is out of the maximum/minimum bit size used for the two vars, in our case $2^{31} - 1$ or $-(2^{32})$, it will lead to overflow. In other words if two positive numbers are added to each other and the result is a negative number there will be overflow. The same goes when subtracting. If a positive number is subtracted from a larger positive number and the result is a negative number there will be overflow.

Our sign flags are also applied to our arithmetic operators since there can be events in which the sign of the 32 bit number will be negative. The sign flag will never trigger for the add operation since both inputs are defaulted positive.

Our zero flag is applied to all of our op code functions. If any, when applied on the two vars, result in zero the flag is activated.

Our output is just the final value that results from any of the operations being completed on the vars.

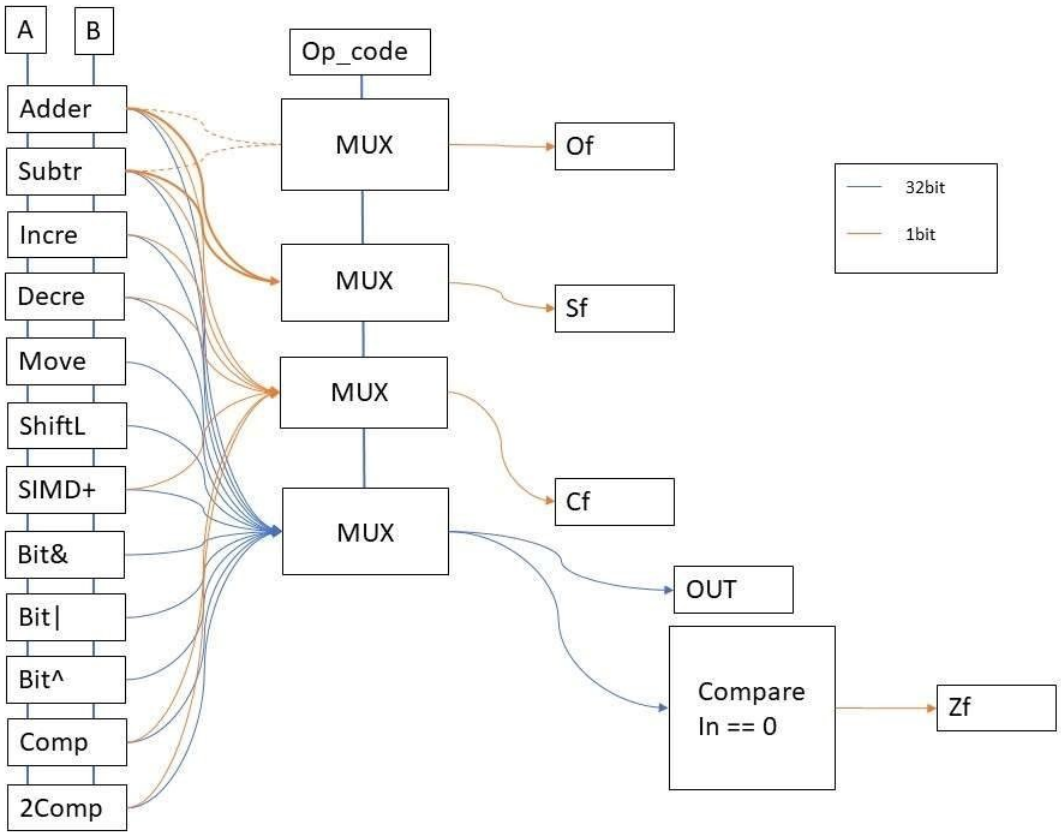
Every sub module takes inputs of A but not all take inputs of B. These outliers are the increments/decrements, bit shifts and compliments. They only take in one input and that input is A since they are only operating on a single number. The other operations require a second number since they utilize both in computing a result. Regarding the outputs, every submodule has the generic output but each one takes inputs of specific flag outputs. All operations except

for move, shift, and the bitwise operators have carry flags since there is a possibility of producing a carry from the leftmost column. Overflow, and sign flags are only applied to arithmetic functions. The bitwise operations and operations on the single var A, do not have any of these flags since their specific functions do not change the numbers in a drastic way and thus will not raise issue when computed. The operations that do require these flags are the arithmetic operations of addition and subtraction. Since they require a combination of two numbers they have a higher risk of affecting the program and moving out of the given bit range. The op-codes of SIMD_ADD, ADD, and SUBTRACTION utilize both these flags.

For base case we forced constants of 0 and 1 and for edge case we forced constants of h'ffffffff and 0 along with other random variables x, y ,z. These forced constants were for

For our simulation we used a number of forced constants. For addition we used 1 for both vars to check the regular addition resulting in 2 and 11111111 and 10000000 to check that our overflow flag and carry flag worked. For subtraction we used 1 for both vars again to check the regular subtraction resulting in 0, 0 and 1 to check that our sign flag worked, and 00000000 and ffffffff to check that our overflow flag worked. For increment to test our carry flag we gave A the value of ffffffff left B as zero since it was a non-factor and to just generally test it we used the value of 1 and again left B as zero for the same reason and the result was 2. The same was done for decrement and move regarding the general test. For 1-bit logical shift left of A we tested 4 for A so we could get concrete evidence of the shift and also let B stay the same resulting in a value of 8. For SIMD_ADD we used the given binary equation on the midterm instructions pdf and it returned the expected output. For bitwise AND we used 1 and 0 to get a value of 0. For bitwise OR we use the same value to obtain a value of 1. For bitwise XOR we had both A and B as 1 returning a value of 0. For compliment we gave A the value of 11111111 in hex and left B alone thus, it returned a value of eeeeeeee. For 2's compliment we used the same value for A and didn't touch B and the result was eeeeeeef. The zero flag will work on any module given that it is triggered within the main module. We gave an example sim of the zero flag with ADD where we had both vars as 0 thus, the result was zero and the flag was triggered.

We had minimal bugs but some nonetheless. These consisted of common syntax errors like instantiation, not defining logic as multiple bits, and using nor instead of ex-or. One bug that was slightly more challenging to correct was making a selector in which we received the error that all functions were not declared when they most definitely were. We ended up having to make temporary outputs from which then we could select the one to actually output. Another challenging bug was an issue we had with our overflow detection flag which in this case was used as an underflow flag since it was in our subtraction operation. We needed to adjust our code to have multiple conditions that had to all be true for the underflow to occur.



Our Block Design