Intro To Processor Architecture

Assignment 1

Team 33 : Snapdragon
Macharla Harish : 2020102062
Revanth Sai : 2020102068

ALU Functionality:

1. Control [00] 0:

The first control we implemented is Full Adder(64bits)
The module takes A , B as inputs which are 64 bits respectively and returns SUM and CARRY_OVERFLOW as outputs. Overflow but returns 1 if overflow occurs else it returns 0.

```
module Add(A,B,SUM,CARRY_OVERFLOW);
```

Approach:

- We will take two 64 bits inputs $A = A_{64}A_{63}\ldots\ldots A_1A_0$, $B = B_{64}B_{63}\ldots..B_1B_0$.
- Now for getting sum part we calculate $SUM_i = A_i$ XOR B_i XOR C_i. (We assumed $C_i = 0$).
- Now for getting the carry part C_{i+1} = (A_i XOR B_i)AND Ci OR A_i and B_i.
- In this manner we calculate the sum for all 64 bits and the XOR value of C_{63} and C_{64} will tell us whether an overflow has occurred or not.

2. **Control [01] 1:**

The next control we implemented is Subtractor(64bits)
This module also uses the full adder to accomplish subtraction. We take 2's complement of the Number that is getting subtracted and add it to the other digit.

```
module Sub(A,B,result,carry_overflow);
```

Here also A,B are the 64 bit inputs and result returns the subtracted output and carry overflow bit returns if an overflow has occurred.

Approach:

- Here suppose A - B is the operation then we just take the 2's complement of B and add it to A.
- The approach for addition part is same as previous Sum control.
- For getting the 2's complement we just negate ~B and add 1 to it.

3. **Control[10] 2:**

The next control we implemented is the AND(64bits)
This module takes in two 64 bits inputs and returns the And result of them.

```
module And(A,B,OUT);
```

Approach:

- We will take two 64 bits inputs A = $A_{64}A_{63}\ldots\ldots A_1A_0$,
  B = $B_{64}B_{63}\ldots..B_1B_0$.
- Now we will find the AND of A and B using simple & i.e.
  OUTi = A_i and B_i, for example $OUT_1 = A_1$ and B_1,
  $OUT_{63} = A_{63}$ and B_{63}.

4. **Control[11] 3:**

The final control implemented is the XOR(64bits)
This module takes in two 64 bit inputs and returns the Xor result of them.

```
module Xor(A,B,OUT);
```

Approach:

- We will take two 64 bits inputs A = $A_{64}A_{63}\ldots\ldots A_1A_0$,
  B = $B_{64}B_{63}\ldots..B_1B_0$.
- Now we will find the XOR of A and B using simple XOR i.e.
  OUTi = A_i XOR B_i, for example OUT_1 = A_1 XOR B_1,
  OUT_{63} = A_{63} XOR B_{63}.

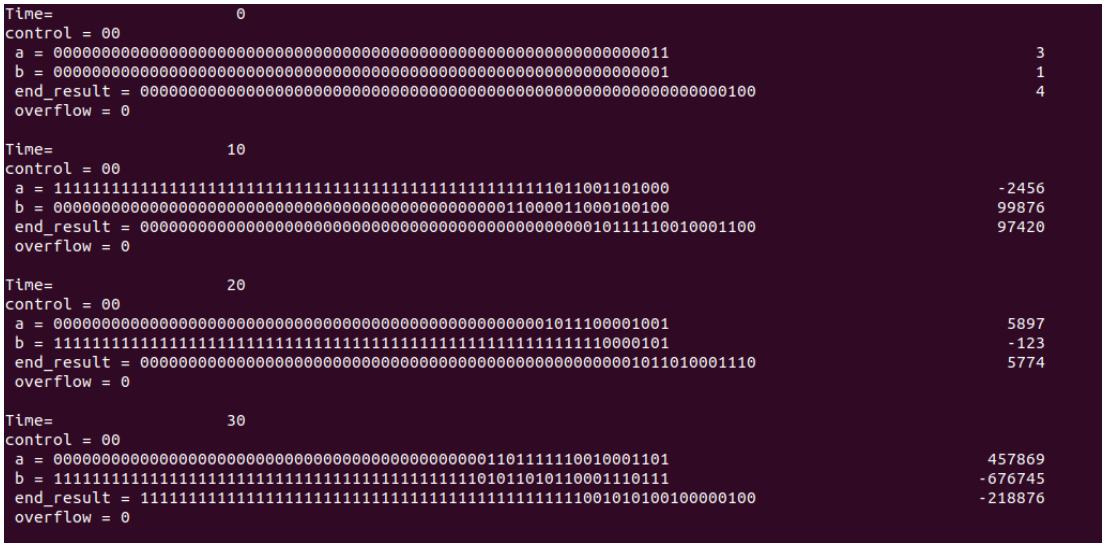
ALU Unit:

Control Input	Function
00	ADD
01	SUB
10	AND
11	XOR

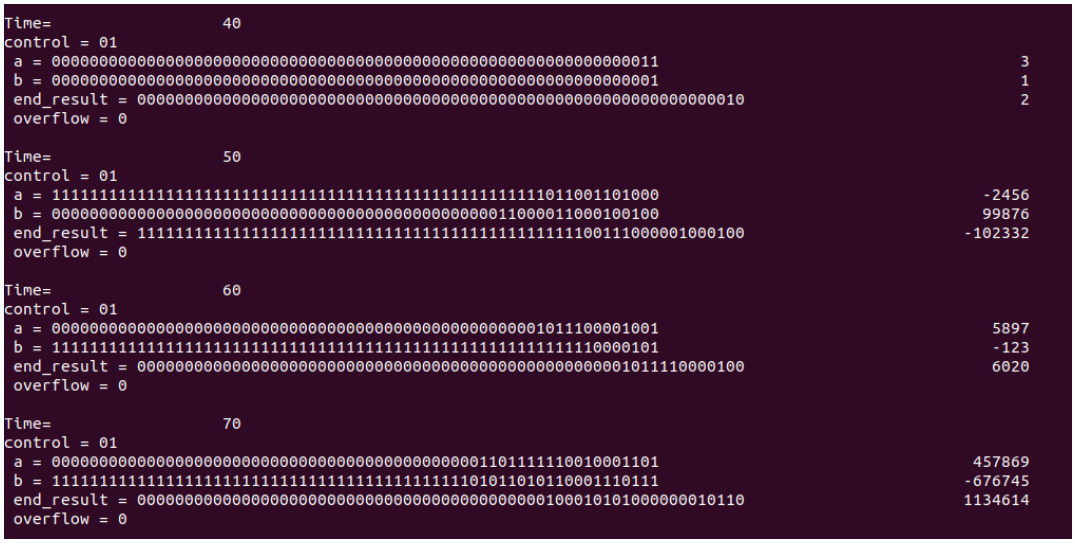
We implemented the above sequence of instructions using case statements in verilog.

Verification and Terminal Results:

The following are terminal results for Adder(64 bits) and we can see that adder gives correct results



The following is the second control subtraction(signed 64 bits) and it also yields correct output. Here we can see that verilog uses 2's complement to represent negative numbers. So all the negative numbers are represented using 2's complement in their binary format.



The following is the third control AND and it also yields expected output.

```
Time=                  80
control = 10
 a = 00000000000000000000000000000000000000000000000000000000000000011                                3
 b = 00000000000000000000000000000000000000000000000000000000000000001                                1
 end_result = 00000000000000000000000000000000000000000000000000000000000000001                       1
 overflow = 0

Time=                  90
control = 10
 a = 11111111111111111111111111111111111111111111111111111011001101000                            -2456
 b = 00000000000000000000000000000000000000000000011000011000100100                                99876
 end_result = 00000000000000000000000000000000000000000000011000011000100000                       99872
 overflow = 0

Time=                  100
control = 10
 a = 00000000000000000000000000000000000000000000000001011100001001                                 5897
 b = 11111111111111111111111111111111111111111111111111111110000101                                -123
 end_result = 00000000000000000000000000000000000000000000000001011100000001                       5889
 overflow = 0

Time=                  110
control = 10
 a = 00000000000000000000000000000000000000000001101111110010001101                              457869
 b = 11111111111111111111111111111111111111111101011010110001110111                             -676745
 end_result = 00000000000000000000000000000000000000000001001010110000000101                      306181
 overflow = 0
```

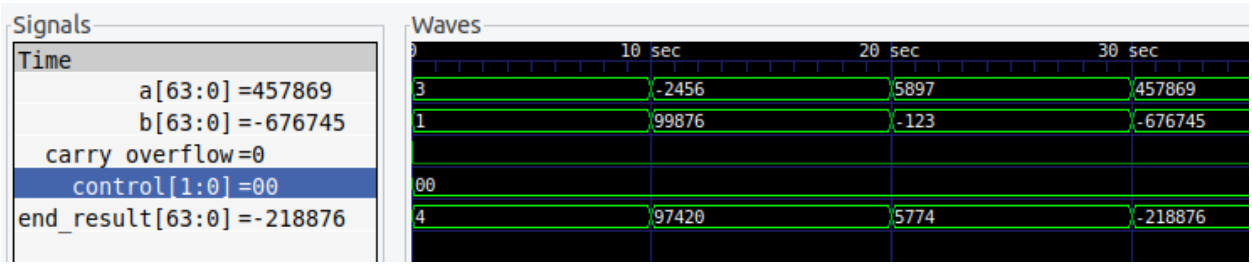
The following is the fourth control XOR and it also yields expected output.

```
Time=                  120
control = 11
 a = 00000000000000000000000000000000000000000000000000000000000000011                                3
 b = 00000000000000000000000000000000000000000000000000000000000000001                                1
 end_result = 00000000000000000000000000000000000000000000000000000000000000010                       2
 overflow = 0

Time=                  130
control = 11
 a = 11111111111111111111111111111111111111111111111111111011001101000                            -2456
 b = 00000000000000000000000000000000000000000000011000011000100100                                99876
 end_result = 11111111111111111111111111111111111111111111111111001110000001001100             -102324
 overflow = 0

Time=                  140
control = 11
 a = 00000000000000000000000000000000000000000000000001011100001001                                 5897
 b = 11111111111111111111111111111111111111111111111111111110000101                                -123
 end_result = 11111111111111111111111111111111111111111111111111111010000010001100               -6004
 overflow = 0

Time=                  150
control = 11
 a = 00000000000000000000000000000000000000000001101111110010001101                              457869
 b = 11111111111111111111111111111111111111111101011010110001110111                             -676745
 end_result = 11111111111111111111111111111111111111111111111001101010000011111010             -831238
 overflow = 0
```

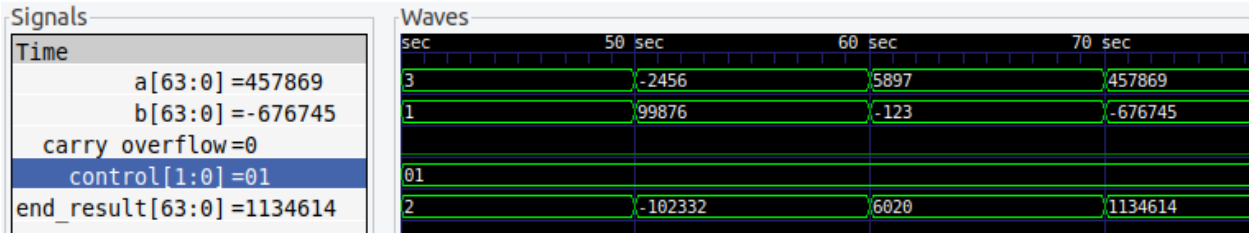
Results:

The following are the GTKWave Outputs

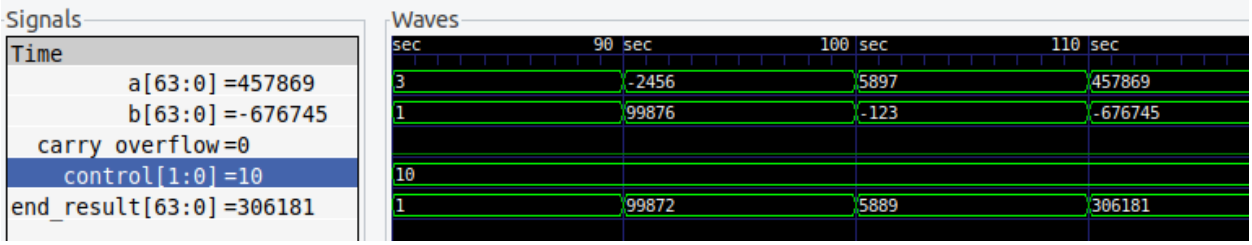
1.Sum



2. Subtraction



3. AND



4. XOR