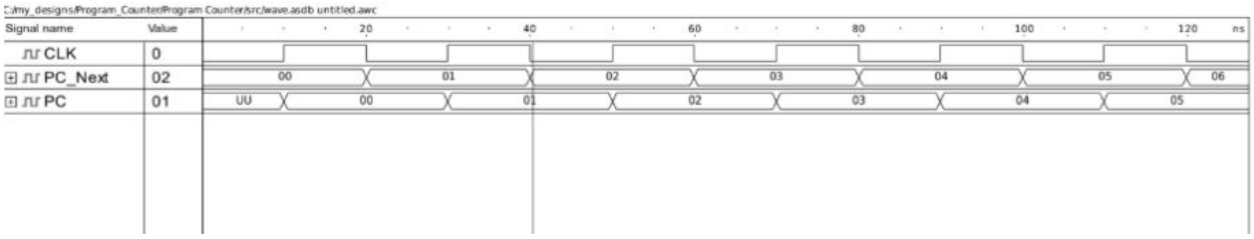
ESE 345 Aaron Lin, Hang Chen

Project Report : Waveform Analysis

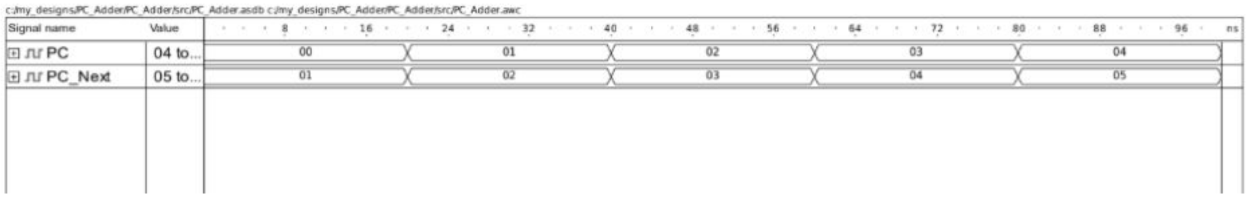
This report will be explaining about the results of the testbenches run for each of the components of the 4-Stage Pipelined Multimedia Unit. The sections are broken up into parts depending on which stage of the pipeline the component belongs to.

IF Stage :

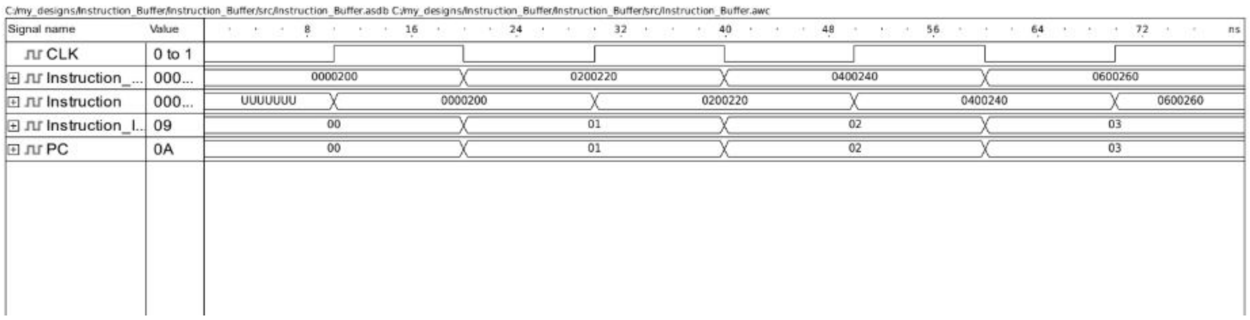
Program Counter: The program counter is increased by 1 at rising edge of the clock



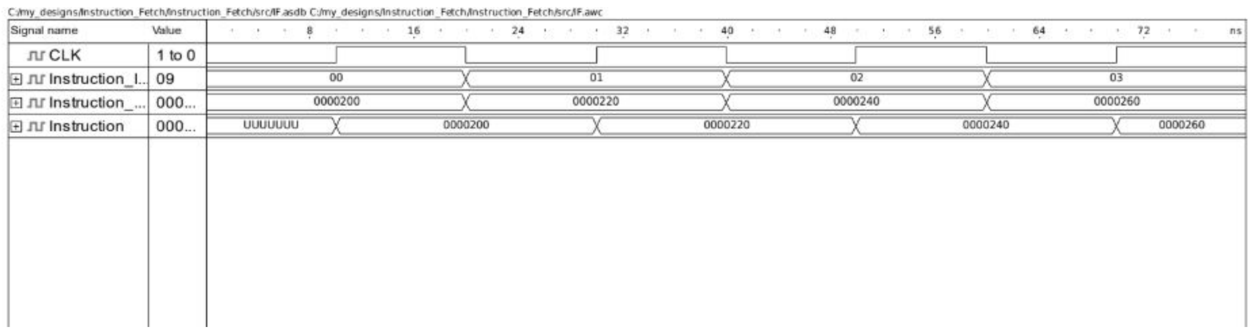
Program Adder: The value of PC is added by 1


c:/my_designs/PC_Adder/PC_Adder/src/PC_Adder.asdb c:/my_designs/PC_Adder/PC_Adder/src/PC_Adder.awc
Signal name
Value
PC 04 to...
PC_Next 05 to...
8 16 24 32 40 48 56 64 72 80 88 96 ns
00 01 02 03 04
01 02 03 04 05

Instruction Buffer: The signal under CLK stores the instruction from the text file. There is also an output instruction signal that is passed to the next stage for decoding.

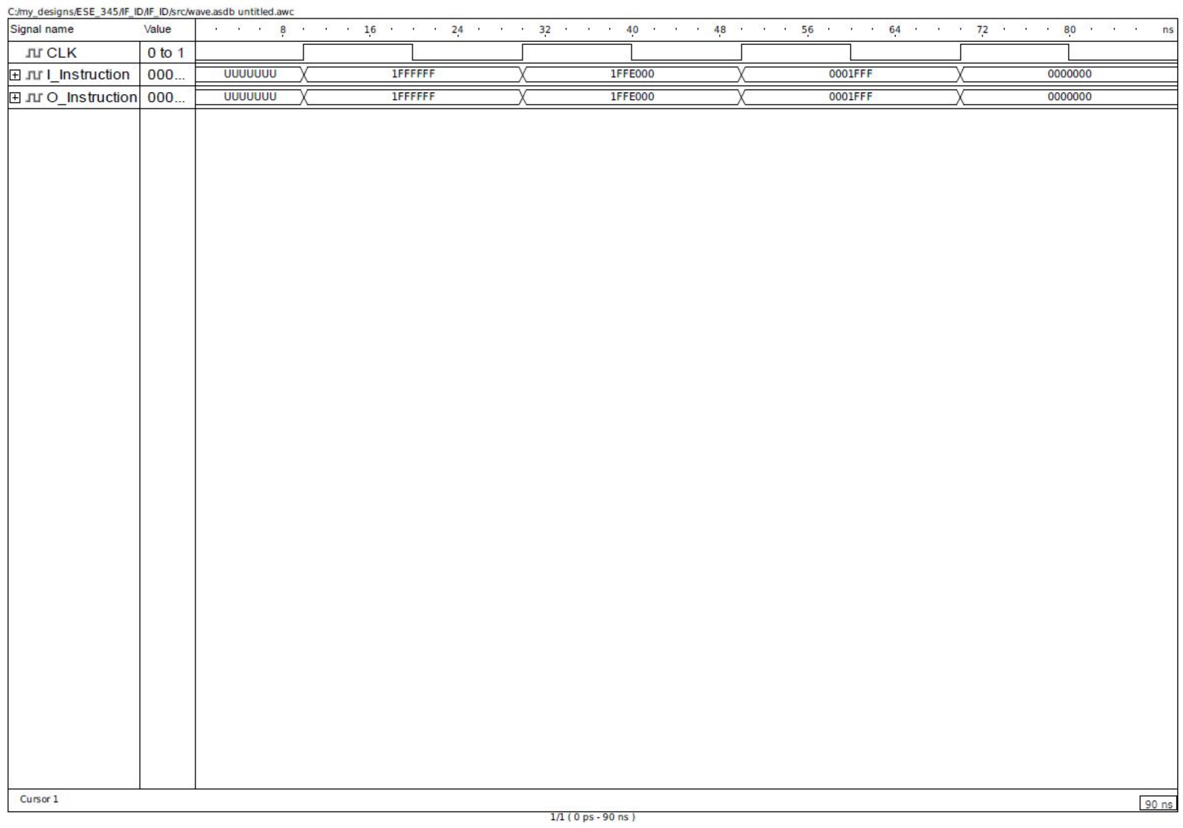

C:/my_designs/Instruction_Buffer/Instruction_Buffer/src/Instruction_Buffer.asdb C:/my_designs/Instruction_Buffer/Instruction_Buffer/src/Instruction_Buffer.awc
Signal name
Value
CLK 0 to 1
Instruction_... 000...
Instruction 000...
Instruction_I... 09
PC 0A
8 16 24 32 40 48 56 64 72 ns
0000200 0200220 0400240 0600260
UUUUUUU 0000200 0200220 0400240 0600260
00 01 02 03
00 01 02 03

Final Waveform of IF stage:


C:/my_designs/Instruction_Fetch/Instruction_Fetch/src/IF.asdb C:/my_designs/Instruction_Fetch/Instruction_Fetch/src/IF.awc
Signal name
Value
8 16 24 32 40 48 56 64 72 ns
CLK
1 to 0
Instruction_I...
09
00 01 02 03
Instruction_...
000...
0000200 0000220 0000240 0000260
Instruction
000...
UUUUUUU 0000200 0000220 0000240 0000260

<u>IF/ID Register</u> :

At the rising edge of the clock, the IF/ID pipeline register takes the input (I_Instruction as shown in the waveform) from the IF stage and passes the data (O_Instruction as shown in the waveform) to the ID stage. Notice that I_instruction and O_instruction are the same.


C:/my_designs/ESE_345/IF_ID/IF_ID/src/wave.asdb untitled.awc
Signal name
Value
8 16 24 32 40 48 56 64 72 80 ns
CLK
0 to 1
I_Instruction
000...
UUUUUUU 1FFFFFF 1FFE000 0001FFF 0000000
O_Instruction
000...
UUUUUUU 1FFFFFF 1FFE000 0001FFF 0000000
Cursor 1
90 ns
1/1 (0 ps - 90 ns)

<u>ID Stage</u> :

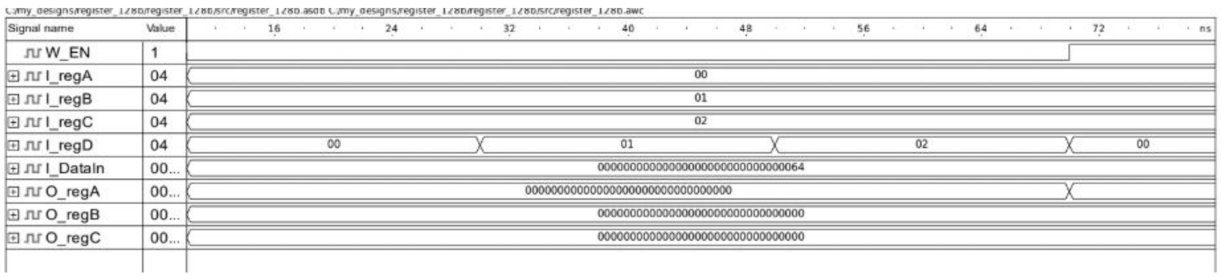
RegisterFile: W_EN shows whether data need to be written into the register file.

I_regA, I_regB, and I_regC indicate which registers need to be read

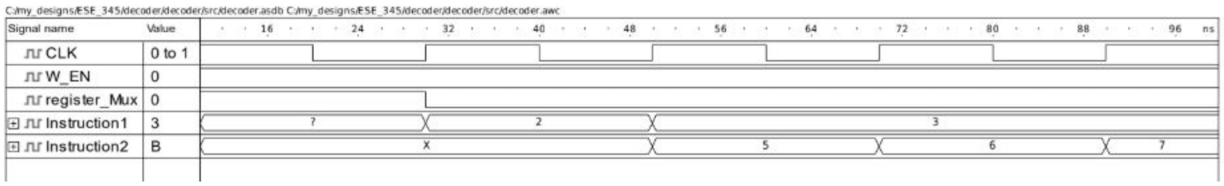
O_regA, O_regB and O_regC contains the data in each of the respective registers above

I_regD indicate the register that need to be written

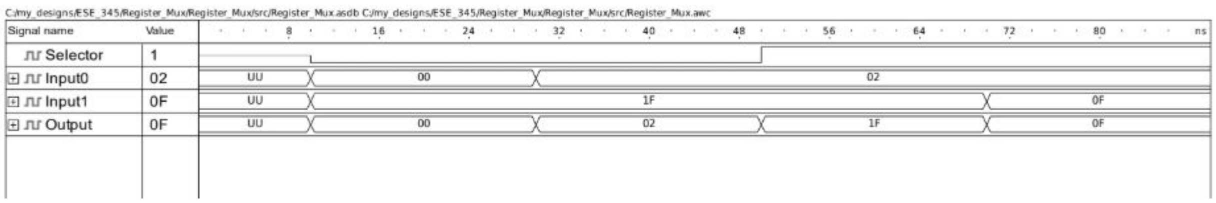
I_Datain contains the data need to be written into the register



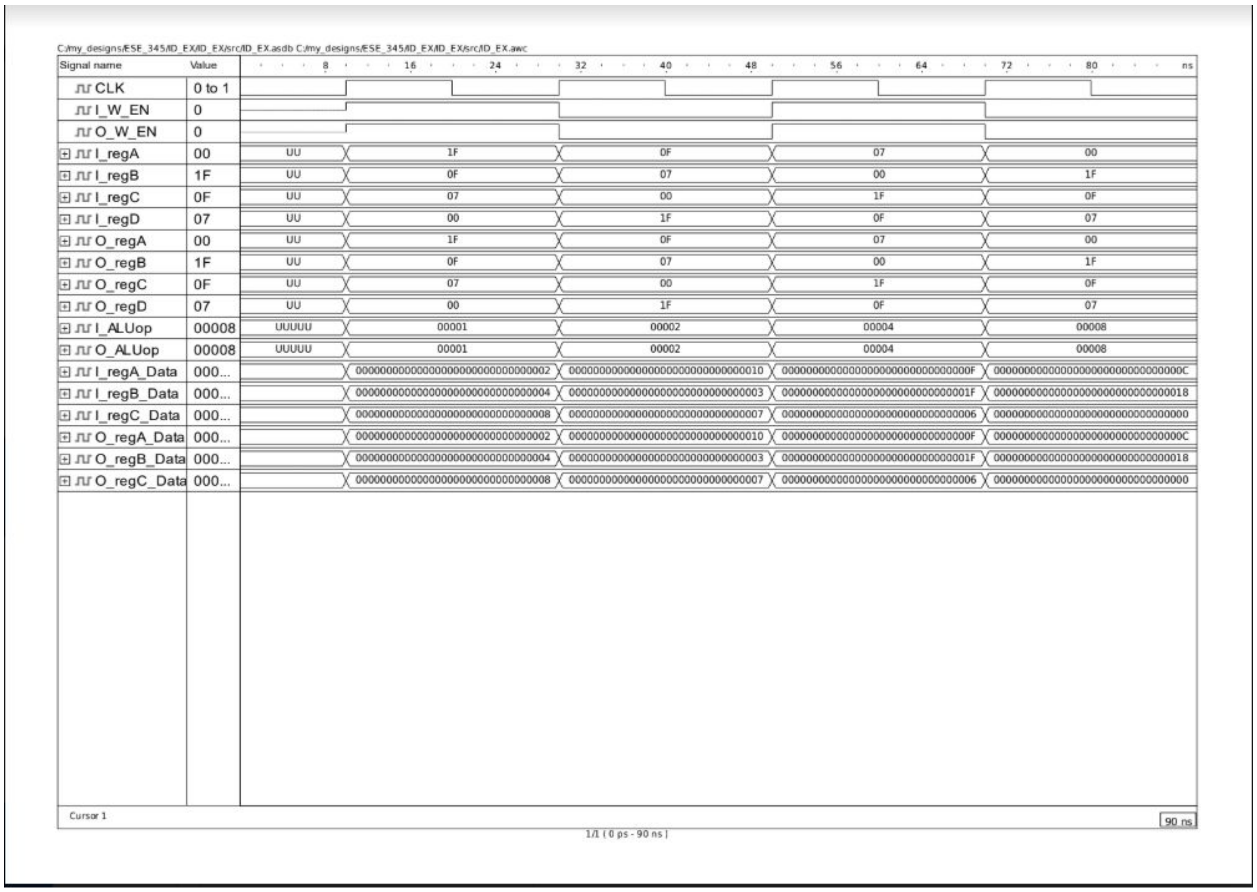
RegisterControlUnit: Instruction1 is the msb 2 bits of the Instruction, and Instruction2 is the opcode bits. With these two, the Control Unit determines if W_EN needs to be asserted, and if the selector bit register_Mux should be asserted. Asserted register_Mux means the instruction is ldi and the register to read from is rd. Asserted W_EN means there is a register to write to.



RegisterMux: As can be seen the Output of the register mux is selected based on the selector. When selector is 0, output will have the same value as Input0. Similarly, when selector is 1, output will have the same value as Input1.

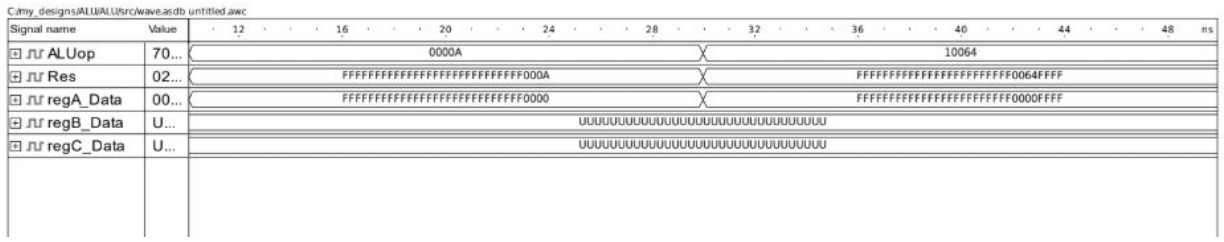


ID/EX Register :Relevant signals are passed from ID stage to EX stage via this register. At the rising edge of the clock, the ID/EX pipeline register takes the inputs (I_W_EN, I_regA, I_regB, I_regC, I_regD, I_ALUop, I_regA_Data, I_regB_Data and I_regC_Data as shown in the waveform) from the ID stage and pass these data(O_W_EN, O_regA, O_regB, O_regC, O_regD, O_ALUop, O_regA_Data, O_regB_Data and O_regC_Data as shown in the waveform) it to the EX stage. As can be seen the data coming out from the register are the same as the data coming into the register.



EX Stage :

LDI: ALuop holds the index of register and the value to be loaded into regA_Data. As can be seen, the Res signal holds the new value combining ALUop and regA_Data.



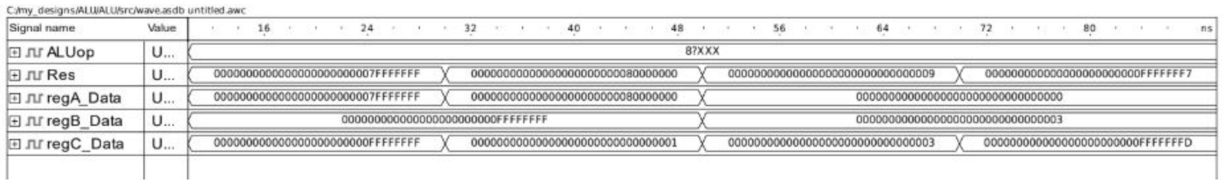
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   12  16  20  24  28  32  36  40  44  48 ns
ALUop         70...   0000A                              | 10064
Res           02...   FFFFFFFFFFFFFFFFFFFFFFFFFFFF000A   | FFFFFFFFFFFFFFFFFFFFFFFF0064FFFF
regA_Data     00...   FFFFFFFFFFFFFFFFFFFFFFFFFFFF0000   | FFFFFFFFFFFFFFFFFFFFFFFF0000FFFF
regB_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
regC_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

R4 Instructions :

Signed Integer Multiply-Add Low with Saturation : The low 16 bits of each word in regB and regC are multiplied and added into corresponding 32 bit regA.

((-1) * (-1)) + max positive = max positive, (1 * -1) + max negative = max negative

(3 * 3) + 0 = 9 , ((-3) * 3) + 0) = -9

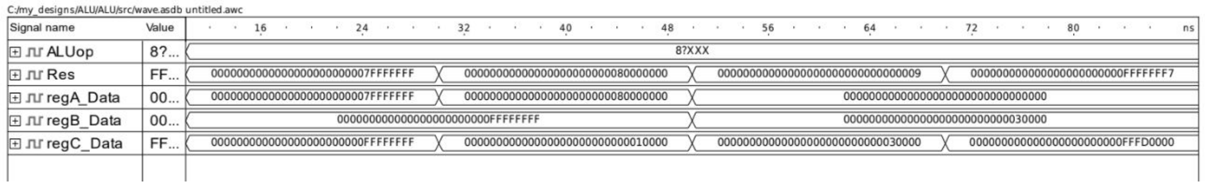


```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   16  24  32  40  48  56  64  72  80 ns
ALUop         U...    8?XXX
Res           U...    0000000000000000000000007FFFFFFF | 00000000000000000000000080000000 | 00000000000000000000000000000009 | 000000000000000000000000FFFFFFF7
regA_Data     U...    0000000000000000000000007FFFFFFF | 00000000000000000000000080000000 | 00000000000000000000000000000000
regB_Data     U...    000000000000000000000000FFFFFFFF                                    | 00000000000000000000000000000003
regC_Data     U...    000000000000000000000000FFFFFFFF | 00000000000000000000000000000001 | 00000000000000000000000000000003 | 000000000000000000000000FFFFFFFD
```

Signed Integer Multiply-Add High with Saturation: The high 16 bits of each word in regB and regC are multiplied and added into corresponding 32 bit regA.

((-1) * (-1)) + max positive = max positive, (1 * -1) + max negative = max negative

(3 * 3) + 0 = 9 , ((-3) * 3) + 0) = -9

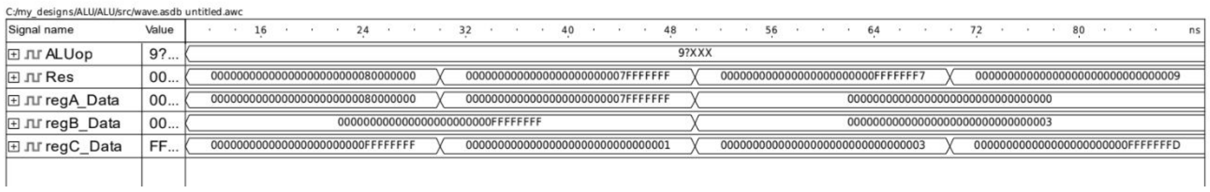


```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   16  24  32  40  48  56  64  72  80 ns
ALUop         8?...   8?XXX
Res           FF...   0000000000000000000000007FFFFFFF | 00000000000000000000000080000000 | 00000000000000000000000000000009 | 000000000000000000000000FFFFFFF7
regA_Data     00...   0000000000000000000000007FFFFFFF | 00000000000000000000000080000000 | 00000000000000000000000000000000
regB_Data     00...   000000000000000000000000FFFFFFFF                                    | 00000000000000000000000000030000
regC_Data     FF...   000000000000000000000000FFFFFFFF | 00000000000000000000000000010000 | 00000000000000000000000000030000 | 000000000000000000000000FFFD0000
```

Signed Integer Multiply-Subtract Low with Saturation: The low 16 bits of each word in regB and regC are multiplied and subtracted from corresponding 32 bit regA.

Max negative - ((-1) * (-1)) = max negative, max positive - ((-1) * 1) = max positive

0 - (3 * 3) = -9, 0 - (3 * (-3)) = 9

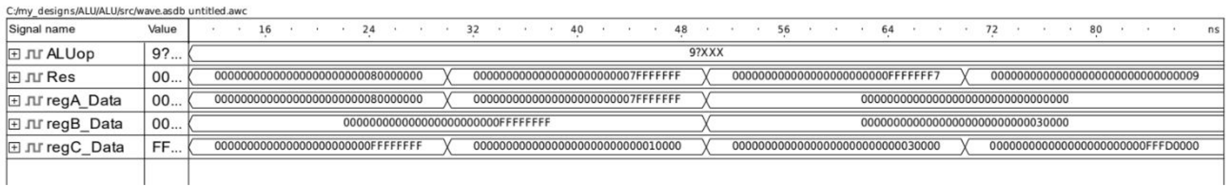


```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   16  24  32  40  48  56  64  72  80  ns
ALUop         9?...   9?XXX
Res           00...   00000000000000000000000080000000 | 0000000000000000000000007FFFFFFF | 000000000000000000000000FFFFFFF7 | 00000000000000000000000000000009
regA_Data     00...   00000000000000000000000080000000 | 0000000000000000000000007FFFFFFF | 00000000000000000000000000000000
regB_Data     00...   000000000000000000000000FFFFFFFF | 00000000000000000000000000000003
regC_Data     FF...   000000000000000000000000FFFFFFFF | 00000000000000000000000000000001 | 00000000000000000000000000000003 | 000000000000000000000000FFFFFFFD
```

Signed Integer Multiply-Subtract High with Saturation: The high 16 bits of each word in regB and regC are multiplied and subtracted from corresponding 32 bit regA.

Max negative - ((-1) * (-1)) = max negative, max positive - ((-1) * 1) = max positive

0 - (3 * 3) = -9, 0 - (3 * (-3)) = 9

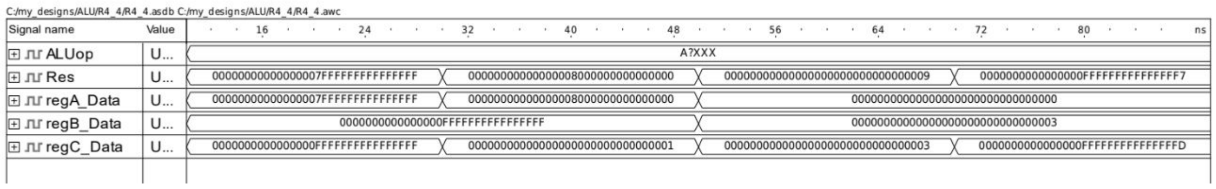


```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   16  24  32  40  48  56  64  72  80  ns
ALUop         9?...   9?XXX
Res           00...   00000000000000000000000080000000 | 0000000000000000000000007FFFFFFF | 000000000000000000000000FFFFFFF7 | 00000000000000000000000000000009
regA_Data     00...   00000000000000000000000080000000 | 0000000000000000000000007FFFFFFF | 00000000000000000000000000000000
regB_Data     00...   000000000000000000000000FFFFFFFF | 00000000000000000000000000030000
regC_Data     FF...   000000000000000000000000FFFFFFFF | 00000000000000000000000000010000 | 00000000000000000000000000030000 | 000000000000000000000000FFFD0000
```

Signed Long Integer Multiply-Add Low with Saturation: The low 32 bits of each word in regB and regC are multiplied and added into corresponding 64 bit regA.

((-1) * (-1)) + max positive = max positive, ((-1) * 1) + max negative = max negative

(3 * 3) + 0 = 9, ((-3) * 3) + 0 = -9

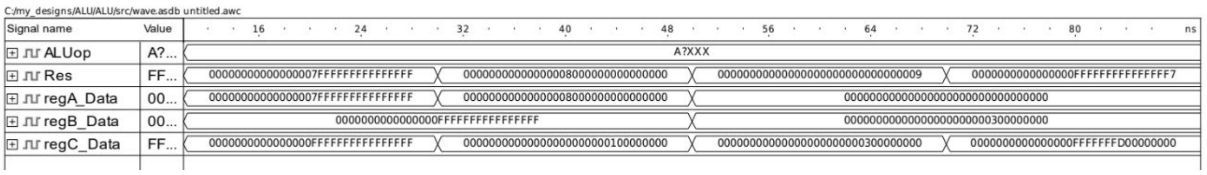


```
C:/my_designs/ALU/R4_4/R4_4.asdb C:/my_designs/ALU/R4_4/R4_4.awc
Signal name   Value   16  24  32  40  48  56  64  72  80  ns
ALUop         U...    A?XXX
Res           U...    00000000000000007FFFFFFFFFFFFFFF | 00000000000000008000000000000000 | 00000000000000000000000000000009 | 0000000000000000FFFFFFFFFFFFFFF7
regA_Data     U...    00000000000000007FFFFFFFFFFFFFFF | 00000000000000008000000000000000 | 00000000000000000000000000000000
regB_Data     U...    0000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000000000003
regC_Data     U...    0000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000000000001 | 00000000000000000000000000000003 | 0000000000000000FFFFFFFFFFFFFFFD
```

Signed Long Integer Multiply-Add High with Saturation: The high 32 bits of each word in regB and regC are multiplied and added into corresponding 64 bit regA.

((-1) * (-1)) + max positive = max positive, ((-1) * 1) + max negative = max negative

(3 * 3) + 0 = 9, ((-3) * 3) + 0 = -9

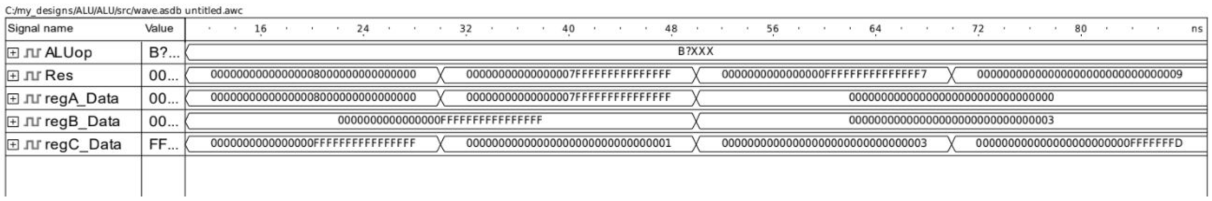


```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   16   24   32   40   48   56   64   72   80   ns
ALUop          A?...   A?XXX
Res            FF...   0000000000000007FFFFFFFFFFFFFFF | 00000000000000008000000000000000 | 00000000000000000000000000000009 | 000000000000000FFFFFFFFFFFFFFF7
regA_Data      00...   0000000000000007FFFFFFFFFFFFFFF | 00000000000000008000000000000000 | 00000000000000000000000000000000
regB_Data      00...   000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000300000000
regC_Data      FF...   000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000100000000 | 00000000000000000000000300000000 | 000000000000000FFFFFFFD00000000
```

Signed Long Integer Multiply-Subtract Low with Saturation: The low 32 bits of each word in regB and regC are multiplied and subtracted from corresponding 64 bit regA.

Max negative - ((-1) * (-1)) = max negative, Max positive - ((-1) * 1) = max positive

0 - (3 * 3) = -9, 0 - (3 * (-3)) = 9

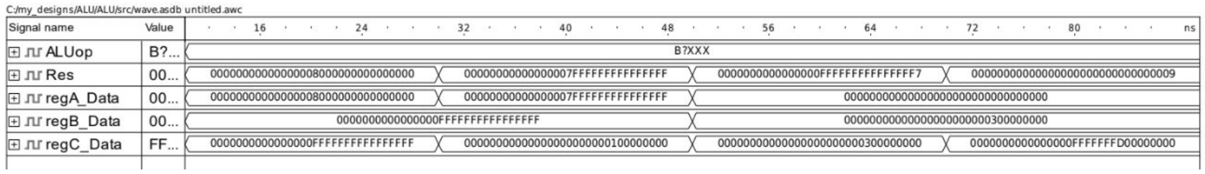


```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   16   24   32   40   48   56   64   72   80   ns
ALUop          B?...   B?XXX
Res            00...   00000000000000008000000000000000 | 0000000000000007FFFFFFFFFFFFFFF | 000000000000000FFFFFFFFFFFFFFF7 | 00000000000000000000000000000009
regA_Data      00...   00000000000000008000000000000000 | 0000000000000007FFFFFFFFFFFFFFF | 00000000000000000000000000000000
regB_Data      00...   000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000000000003
regC_Data      FF...   000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000000000001 | 00000000000000000000000000000003 | 000000000000000000000000FFFFFFFD
```

Signed Long Integer Multiply-Subtract High with Saturation: The high 32 bits of each word in regB and regC are multiplied and subtracted from corresponding 64 bit regA.

Max negative - ((-1) * (-1)) = max negative, Max positive - ((-1) * 1) = max positive

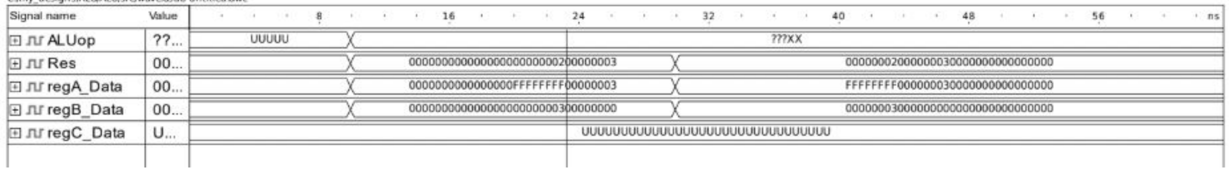
0 - (3 * 3) = -9, 0 - (3 * (-3)) = 9



```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   16   24   32   40   48   56   64   72   80   ns
ALUop          B?...   B?XXX
Res            00...   00000000000000008000000000000000 | 0000000000000007FFFFFFFFFFFFFFF | 000000000000000FFFFFFFFFFFFFFF7 | 00000000000000000000000000000009
regA_Data      00...   00000000000000008000000000000000 | 0000000000000007FFFFFFFFFFFFFFF | 00000000000000000000000000000000
regB_Data      00...   000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000300000000
regC_Data      FF...   000000000000000FFFFFFFFFFFFFFFF | 00000000000000000000000100000000 | 00000000000000000000000300000000 | 000000000000000FFFFFFFD00000000
```

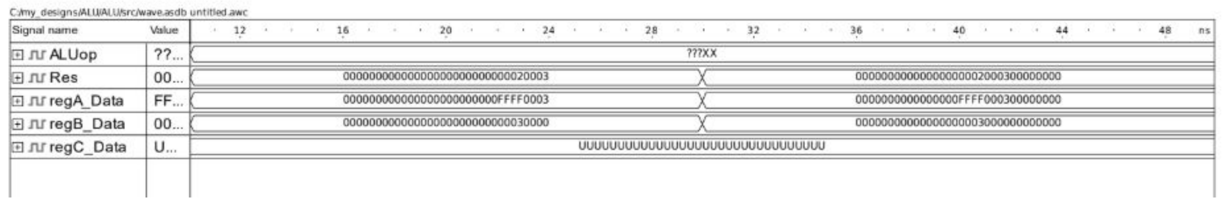
<u>R3 Instructions</u> :

A(add): Res store the result of regA_Data and regB_Data. Noticed that the least significant packed 32 bits yield a 0+3=3. The 2nd least significant packed 32 bits produce FFFFFFFF+3 = 100000002 and truncated to 00000002 dude to overflow.



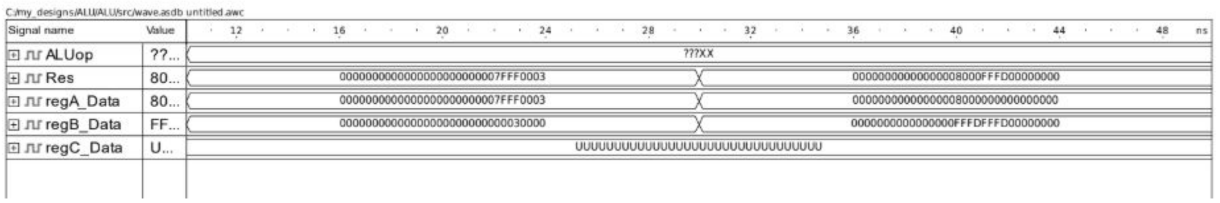
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8   16   24   32   40   48   56   ns
ALUop          ??...   UUUUU | ???XX
Res            00...         | 00000000000000000000000200000003 | 00000002000000030000000000000000
regA_Data      00...         | 0000000000000000FFFFFFFF00000003 | FFFFFFFF000000030000000000000000
regB_Data      00...         | 00000000000000000000000300000000 | 00000003000000000000000000000000
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

AH(add halfword): Res store the result of regA_Data and regB_Data. Each Packed 16 bit half word consists of a pair(2 value = 2 bytes = 16 bits) from each register. 00+03 = 03 as the least significant pair. FFFF+0003 = 10002 truncated to 0002 since only the least significant 16 bits are preserved.



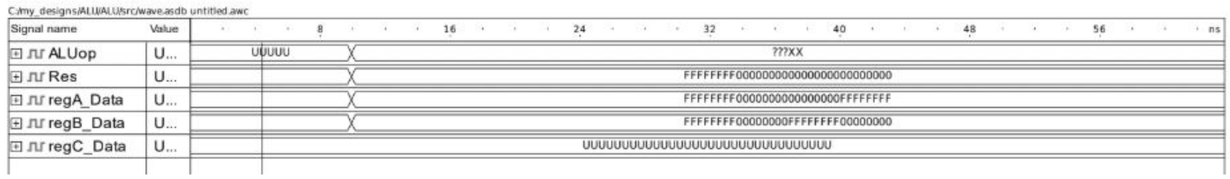
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   12   16   20   24   28   32   36   40   44   48 ns
ALUop          ??...   ???XX
Res            00...   00000000000000000000000000020003   00000000000000000002000300000000
regA_Data      FF...   000000000000000000000000FFFF0003   0000000000000000FFFF000300000000
regB_Data      00...   00000000000000000000000000030000   00000000000000000003000000000000
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

AHS(add halfword with saturated): 16 bits from regA is added with the corresponding 16 bits from regB. As can be seen, 0 + 3 = 3, 3 + max positive = max positive, -3 + 0 = -3, and -3 - max negative = max negative.



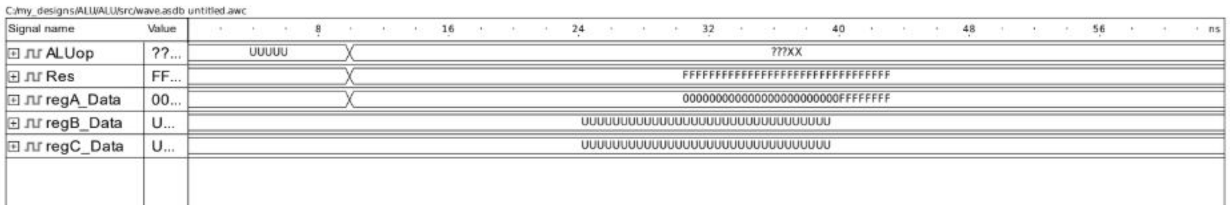
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   12   16   20   24   28   32   36   40   44   48 ns
ALUop          ??...   ???XX
Res            80...   0000000000000000000000007FFF0003   00000000000000008000FFFD00000000
regA_Data      80...   0000000000000000000000007FFF0003   00000000000000008000000000000000
regB_Data      FF...   00000000000000000000000000030000   0000000000000000FFFDFFFD00000000
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

AND: Res store the AND result of regA_Data and regB_Data.



```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8   16   24   32   40   48   56 ns
ALUop          U...    UUUUU   ???XX
Res            U...            FFFFFFFF000000000000000000000000
regA_Data      U...            FFFFFFFF0000000000000000FFFFFFFF
regB_Data      U...            FFFFFFFF00000000FFFFFFFF00000000
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

BCW(Broadcast word): Rightmost 32 bits of regA_Data(rs1) has value of FFFFFFFF(32 bits of 1). The values are stored into all the 32 bits section of Res.



```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8   16   24   32   40   48   56 ns
ALUop          ??...   UUUUU   ???XX
Res            FF...           FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
regA_Data      00...           000000000000000000000000FFFFFFFF
regB_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

CLZ(count leading zeros in words): The counting result is stored in Res.

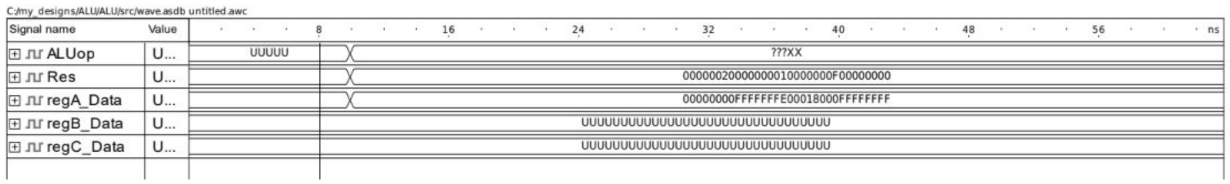
Word = 4 bytes = 32 bits. Each data has 4 bits. When counting in words, the register can be broken into 4 small groups each having 4 byte(32 bits).

First group is 00000000, noticed that each 0 is made up of 0000. Thus Res has 20 = 32 count for 0 in the most significant word.

Second group is FFFFFFFE, F = 1111 while E = 1110. Thus Res has 1 count for 0 in the second most significant word.

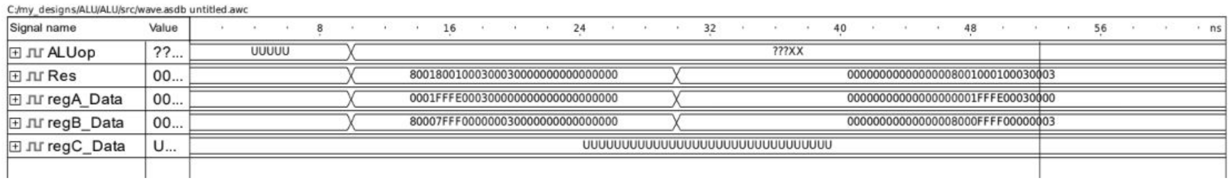
Third group is 00018000, again each 0 within the group = 0000,1 = 0001 and 8 = 1000. However, only the 0 before value 1 in 0001 are counted. Thus, Res has e 3*4+3 = 15 = F count of leading 0 in the third most significant word.

Fourth group is FFFFFFFF which has no 0 in it. Thus Res is 0 for the least significant word.



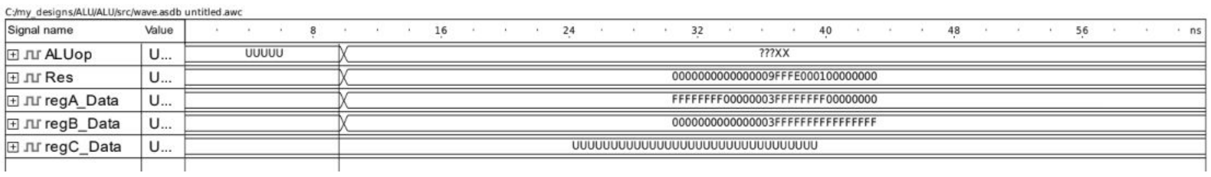
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU  ???XX
Res            U...    00000020000000010000000F00000000
regA_Data      U...    00000000FFFFFFFE00018000FFFFFFFF
regB_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

ABSDB(absolute difference of bytes): Res store the absolute value of (regB_Data - regA_Data)



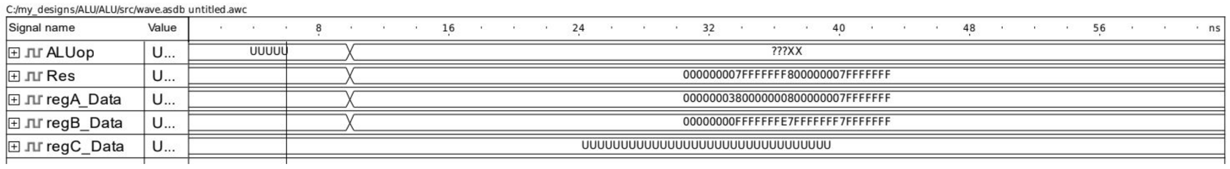
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          ??...   UUUUU  ???XX
Res            00...   80018001000300030000000000000000  00000000000000008001000100030003
regA_Data      00...   0001FFFE000300000000000000000000  000000000000000001FFFE00030000
regB_Data      00...   80007FFF000000030000000000000000  00000000000000008000FFFF00000003
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

MPYU(multiply unsigned): Res stores the result of (regA_Data * regB_Data) unsigned. Only the multiplication result of lower 16 bits of packed 32 bits are stored into Res. The higher 16 bits of the packed 32 bits are set to be 0 in Res. The multiplication are FFFF*0000 = 0000, FFFF*FFFF = 0001 after truncated and 0003*0003 = 0009.



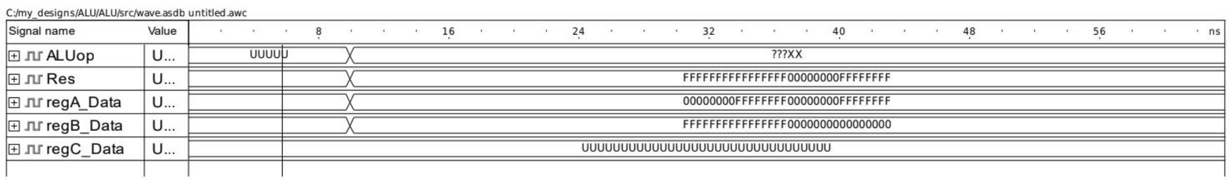
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU  ???XX
Res            U...    0000000000000009FFFE000100000000
regA_Data      U...    FFFFFFFF00000003FFFFFFFF00000000
regB_Data      U...    0000000000000003FFFFFFFFFFFFFFFF
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

MSGN: (multiply sign): regA is multiplied by the sign of regB. As can be seen, max positive multiplied by a positive value gives a max positive, max negative multiplied by a positive gives a max negative, max negative multiplied by a negative gives a max positive, and anything multiplied by 0 gives 0.



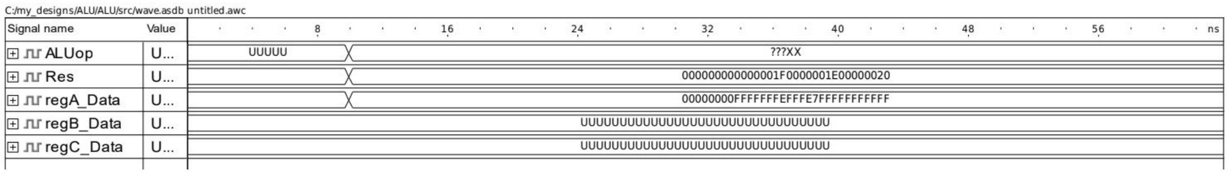
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU  ???XX
Res            U...    000000007FFFFFFF800000007FFFFFFF
regA_Data      U...    00000003800000008000000007FFFFFFF
regB_Data      U...    00000000FFFFFFFE7FFFFFFF7FFFFFFF
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

OR: Res store the result OR operation of regA_Data and regB_Data



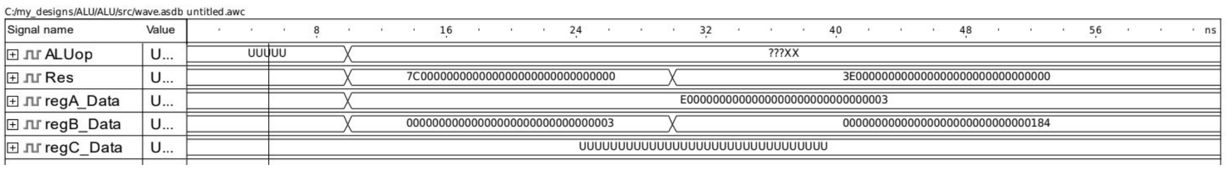
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU   ???XX
Res            U...    FFFFFFFFFFFFFFFF00000000FFFFFFFF
regA_Data      U...    00000000FFFFFFFF00000000FFFFFFFF
regB_Data      U...    FFFFFFFFFFFFFFFF0000000000000000
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

POPCNTW(count ones in words): 1 word = 4 bytes = 32 bits. Each word consists of 8 values in regA_Data . Taking the lowest word from regA_Data for example, these are (FFFFFFFF) which has 4*8 = 32 counts of value 1. The lowest words in Reg contain the value 0020 = 0010 0000, which is 32.



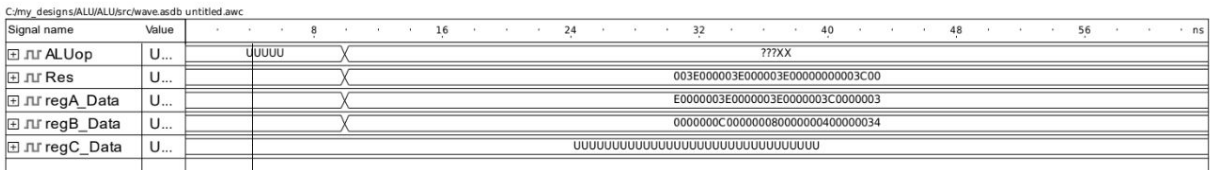
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU   ???XX
Res            U...    000000000000001F0000001E00000020
regA_Data      U...    00000000FFFFFFFEFFFE7FFFFFFFFFFF
regB_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

ROT(rotate bits right): The contents in regA_Data are rotated to the right by 3(value from regB_Data). regA_Data has two significant values: E = 1110, 3 = 0011. By rotating to the right by 3, the least 3 bits(011) of 3 from regA_Data are moved to the front of E which result in 0111 1100 = 7C as can be seen in Res.



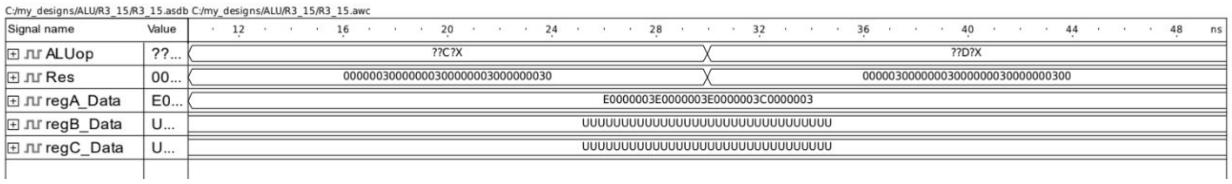
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU   ???XX
Res            U...    7C000000000000000000000000000000   3E000000000000000000000000000000
regA_Data      U...    E0000000000000000000000000000003
regB_Data      U...    00000000000000000000000000000003   00000000000000000000000000000184
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

ROTW(rotate bits in word): Content of regA_Data are shifted based on the 5 least significant bits in regB_Data. These 5 bits (10100=20) came from 34 = 0011 0100. It means that contents in regA_Data are rotated by 20 bits or 5 indexes (20/4=5) to the left and stored into Res.



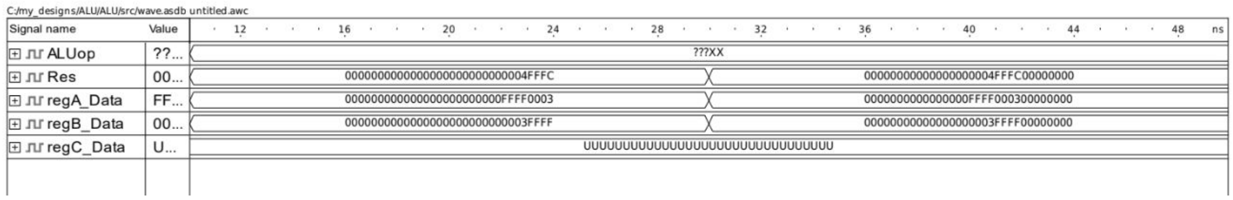
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name    Value   8  16  24  32  40  48  56  ns
ALUop          U...    UUUUU   ???XX
Res            U...    003E000003E000003E00000000003C00
regA_Data      U...    E0000003E0000003E0000003C0000003
regB_Data      U...    0000000C000000080000000400000034
regC_Data      U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

SHLHI(shift left halfword immediate): Each 16 bit of regA is shifted left by the value of rs2 in the instruction field.



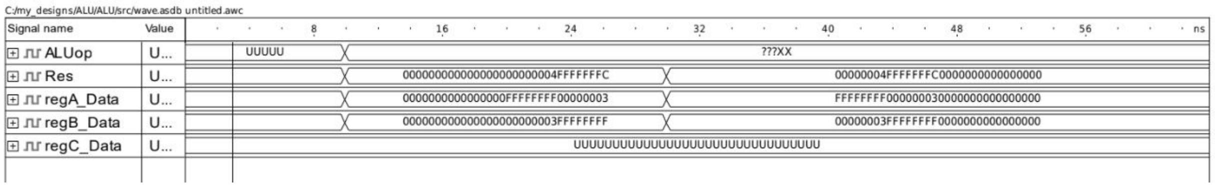
```
C:/my_designs/ALU/R3_15/R3_15.asdb C:/my_designs/ALU/R3_15/R3_15.awc
Signal name   Value   12  16  20  24  28  32  36  40  44  48  ns
ALUop         ??...   ??C?X                              ??D?X
Res           00...   0000003000000030000000300000030   0000030000000300000003000000300
regA_Data     E0...   E0000003E0000003E0000003C0000003
regB_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
regC_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

SFH(subtract from halfword): halfword = 16 bits = 4 data points in the register. Res store the result of (RegA_Data - RegB_Data)



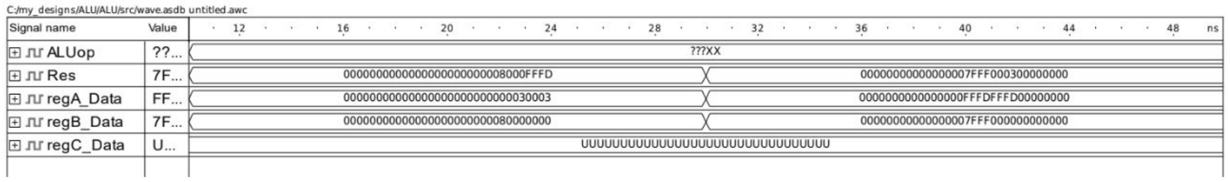
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   12  16  20  24  28  32  36  40  44  48  ns
ALUop         ??...   ???XX
Res           00...   000000000000000000000000004FFFC    00000000000000000004FFFC00000000
regA_Data     FF...   0000000000000000000000FFFF0003     000000000000000FFFF000300000000
regB_Data     00...   000000000000000000000000003FFFF    00000000000000000003FFFF00000000
regC_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

SFW(subtract from word): word = 32 bits = 8 data points in the register. Res store the result of (RegA_Data - RegB_Data)



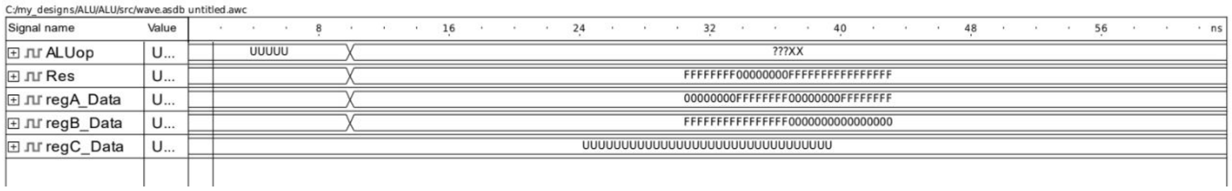
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   8  16  24  32  40  48  56  ns
ALUop         U...    UUUUU   ???XX
Res           U...    0000000000000000000000004FFFFFFFC    00000004FFFFFFFC0000000000000000
regA_Data     U...    0000000000000000FFFFFFFF00000003     FFFFFFFF00000003000000000000000
regB_Data     U...    0000000000000000000000003FFFFFFFF    00000003FFFFFFFF0000000000000000
regC_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

SFHS(subtract from halfword saturated): regB is subtracted by regA. As can be seen, 0 - 3 = -3, max negative - 3 = max negative, 0 - (-3) = 3, and max positive - (-3) = max positive.



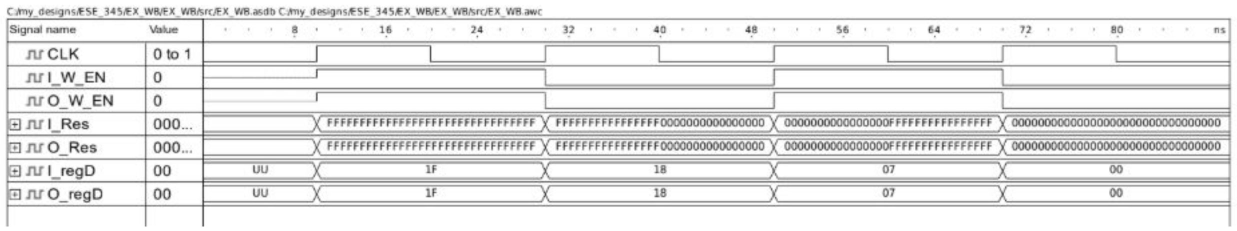
```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   12  16  20  24  28  32  36  40  44  48  ns
ALUop         ??...   ???XX
Res           7F...   000000000000000000000000008000FFFD   00000000000000007FFF000300000000
regA_Data     FF...   00000000000000000000000000030003    000000000000000FFFDFFFD00000000
regB_Data     7F...   000000000000000000000000080000000   00000000000000007FFF000000000000
regC_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

NAND: Res stores the result of regA_Data NAND regB_Data



```
C:/my_designs/ALU/ALU/src/wave.asdb untitled.awc
Signal name   Value   8  16  24  32  40  48  56  ns
ALUop         U...    UUUUU   ???XX
Res           U...    FFFFFFFF00000000FFFFFFFFFFFFFFFF
regA_Data     U...    00000000FFFFFFFF00000000FFFFFFFF
regB_Data     U...    FFFFFFFFFFFFFFFF0000000000000000
regC_Data     U...    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

<u>EX/WB Register</u>: This register takes in the I_W_EN, I_Res, and I_regD signals and outputs them onto the Forwarding Unit and Register File in the ID stage. As can be seen, on every rising clock edge, the corresponding output signal for each of these input signals are the same. This shows correct expected behavior of the EX/WB Register.


C:/my_designs/ESE_345/EX_WB/EX_WB/src/EX_WB.asdb C:/my_designs/ESE_345/EX_WB/EX_WB/src/EX_WB.awc
Signal name
Value
8 16 24 32 40 48 56 64 72 80 ns
CLK 0 to 1
I_W_EN 0
O_W_EN 0
I_Res 000...
O_Res 000...
I_regD 00
O_regD 00
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF0000000000000000
0000000000000000FFFFFFFFFFFFFFFF
00000000000000000000000000000000
UU
1F
18
07
00

<u>4-Stage Pipelined Multimedia Unit</u> :

This is a part of the entire waveform of the pipeline. The first two signals are responsible for the loading of the assembly converted into binary format instructions into the Instruction Buffer. The first one is the index in the Instruction Buffer to load into, and the second one is the Instruction itself. Clearly, the first signal is increasing by 1s indicating the correct location in the Instruction Buffer the instruction is being loaded into. The second signal is also correct because their order of loading corresponds exactly with that in the assembly file.

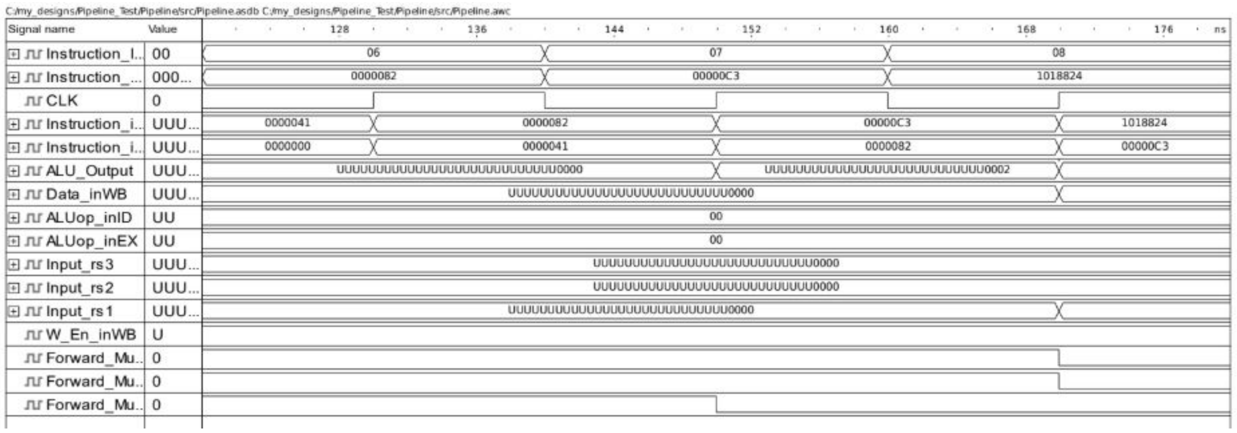
The two Instruction_i signals are the instructions entering the IF/ID register, and the instructions entering the register file respectively. As can be seen, the instruction entering the register file is occurring one clock cycle after the one entering the IF/ID register. This is a correct indicator of a working pipeline from the IF stage to the ID stage.

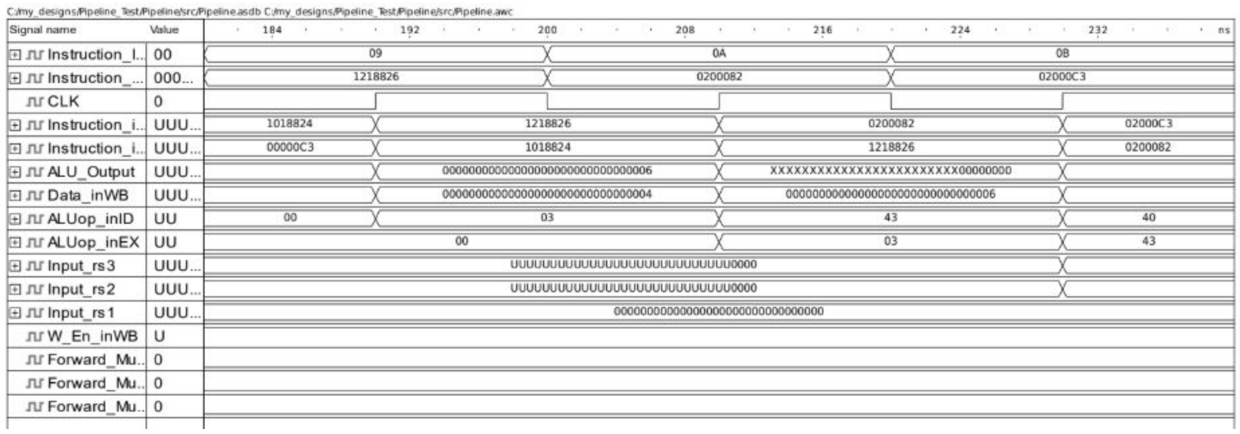
The Instruction 0000041 represents the loading of 2 into the left most significant 16 bits of rd. As can be seen in the ALUOutput, although not entirely correct in the representation of the data in the register, we can see that the semi-correct output from the ALU appears one cycle after the instruction enters the register file or ID stage. This is once again also a correct indicator of a working pipeline from the ID stage to the EX stage. If we look at the signals ALUop_inID and ALUop_inEX, we can also see a correct indicator of a working pipeline from ID to EX stage, as the ALUop shows one cycle from one another.

The ALU_Output signal occurs in the EX stage, whereas the Data_inWB occurs in the WB stage. The Data_inWB signal occurs one cycle after ALU_Output, which is correct. These two signals represent the same data. Once again, this indicates the correct behavior of a pipeline from the EX stage to the WB stage.

In conclusion, the pipeline can finish executing a single instruction in 4 cycles, as expected. Although the pipeline behavior observed is correct, the data being manipulated is still flawed. This could be because of the failure to write the proper data to the register, and the forwarding of improper data. In addition, this could further be caused by the problems rising from putting the pipeline together, such as the ability for the stages to work together synchronously. As can be seen, in the previous waveforms, the individual components accomplish their tasks successfully.

The next step should be to examine each stage over again, but with the slow incorporation of other components one at a time.


C:/my_designs/Pipeline_Test/Pipeline/src/Pipeline.asdb C:/my_designs/Pipeline_Test/Pipeline/src/Pipeline.awc
Signal name
Value
128 136 144 152 160 168 176 ns
Instruction_I.. 00 — 06, 07, 08
Instruction_... 000... — 0000082, 00000C3, 1018824
CLK 0
Instruction_i.. UUU... — 0000041, 0000082, 00000C3, 1018824
Instruction_i.. UUU... — 0000000, 0000041, 0000082, 00000C3
ALU_Output UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000, UUUUUUUUUUUUUUUUUUUUUUUUUUUU0002
Data_inWB UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000
ALUop_inID UU — 00
ALUop_inEX UU — 00
Input_rs3 UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000
Input_rs2 UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000
Input_rs1 UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000
W_En_inWB U
Forward_Mu.. 0
Forward_Mu.. 0
Forward_Mu.. 0


C:/my_designs/Pipeline_Test/Pipeline/src/Pipeline.asdb C:/my_designs/Pipeline_Test/Pipeline/src/Pipeline.awc
Signal name
Value
184 192 200 208 216 224 232 ns
Instruction_I.. 00 — 09, 0A, 0B
Instruction_... 000... — 1218826, 0200082, 02000C3
CLK 0
Instruction_i.. UUU... — 1018824, 1218826, 0200082, 02000C3
Instruction_i.. UUU... — 00000C3, 1018824, 1218826, 0200082
ALU_Output UUU... — 00000000000000000000000000000006, XXXXXXXXXXXXXXXXXXXXXXXX00000000
Data_inWB UUU... — 00000000000000000000000000000004, 00000000000000000000000000000006
ALUop_inID UU — 00, 03, 43, 40
ALUop_inEX UU — 00, 03, 43
Input_rs3 UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000
Input_rs2 UUU... — UUUUUUUUUUUUUUUUUUUUUUUUUUUU0000
Input_rs1 UUU... — 00000000000000000000000000000000
W_En_inWB U
Forward_Mu.. 0
Forward_Mu.. 0
Forward_Mu.. 0