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

| IT CLK 0 Image: control of the property of the prope | 03 |
|--|----------------|
| ⊞ лг PC_Next 02 00 (01) 02 (03) 04 (05) | 03 |
| | |
| ⊞ ΛΓPC 01 UU X 00 X 01 X 02 X 03 X 04 X 05 | X 03 X 04 X 05 |

Program Adder: The value of PC is added by 1

| ⊡ n r P C | |
|-----------------------|--|
| | |
| ⊡ // PC Next 05 to 01 | |

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.

| lignal name | Value | Document 1 | | 16 | . 24 | 8 5 8 | 32 | 40 | 48 | 56 | Barra to | 64 | 2 |
|---------------|--------|------------|---------|------|------|-------|------|---------|-----|------|---------------|---------|---------|
| JL CLK | 0 to 1 | | | | | | | | | | | | |
| Instruction | 000 | | 0000200 | | | 0200 | 1220 | X | 040 | 0240 | $\overline{}$ | 0600260 | |
| Instruction | 000 | UUUUUUU | _X | 0000 | 200 | | | 0200220 |) | | 0400240 | Х | 0600260 |
| Instruction_I | 09 | | 00 | | | 0 | 1 | X_ | |)2 | $\overline{}$ | 03 | |
| J.T. PC | 0A | | 00 | | | 0 | 1 | X | (|)2 | X | 03 | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

Final Waveform of IF stage:

| | Value | 8 | | 16 | 4 32 | 40 | 48 5 | 6 64 | 72 |
|------------------|--------|---------|--------|---------|---------|---------|---------|---------|---------|
| ЛГCLK | 1 to 0 | | | | | ŢĹ, | | | |
| Instruction_I | 09 | | 00 | X | 01 | X | 02 | X | 03 |
| ☐ ЛГ Instruction | 000 | 0 | 000200 | X | 0000220 | X | 0000240 | X | 0000260 |
| Instruction | 000 | UUUUUUU | X | 0000200 | χ | 0000220 | χ | 0000240 | 0000260 |
| | | | | | | | | | |

IF/ID Register:

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.

| ignal name | Value | | 16 24 | 52 40 40 | 5,0 | 7.2 |
|------------------|--------|-----------|-----------|----------|---------|-----------|
| JT CLK | 0 to 1 | | | 9 | | |
| ா I_Instruction | 000 | UUUUUUU | 1FFFFFF X | 1FFE000 | 0001FFF | 0000000 |
| лг O_Instruction | 000 | UUUUUUU X | 1FFFFFF X | 1FFE000 | 0001FFF | X 0000000 |
| | | | | | | |
| ursor 1 | | | | | | |

ID Stage:

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

| Signal name | Value | 100 | 16 | | | 24 | | 32 | | | 40 | 1.4 | 1 | 48 | | | 1 | 56 | | | 64 | | | 72 | 11 1 | · ns |
|---------------|-------|-----|----|---|---|----|-----------|----|--------|--------|---------|--------|--------|---------|-------------|-----|---|----|---|---|----|--|---|----|------|------|
| JT W_EN | 1 | | | | | | | | | | | | | | | | | | | | | | Г | | | |
| ⊕ лг I_regA | 04 | | | | | | | | | | | | 00 | | | | | | | | | | | | | |
| ⊕ лг I_regB | 04 | | | | | | | | | | | | 01 | | | | | | | | | | | | | |
| ⊕ лт I_regC | 04 | | | | | | | | | | | | 02 | | | | | | | | | | | | | |
| ⊞ лт I_regD | 04 | | | 0 |) | | \supset | | | | 01 | | | | \supset C | | | | 0 | 2 | | | X | | 00 | |
| ⊕ лг I_DataIn | 00 | | | | | | | | | | 000000 | 000000 | 000000 | 0000000 | 000000 | 064 | | | | | | | | | | |
| ⊕ .rur O_regA | 00 | | | | | | | 3 | 000000 | 000000 | 0000000 | 000000 | 000000 | 000 | | | | | | | | | X | | | |
| ⊕ лг O_regB | 00 | | | | | | | | | | 000000 | 000000 | 000000 | 0000000 | 000000 | 000 | | | | | | | | | | |
| ⊕ .⊓.r O_regC | 00 | | | | | | | | | | 000000 | 000000 | 000000 | 000000 | 000000 | 000 | | | | | | | | | | |

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.

| Signal name | Value | - 2 | 16 | | 2 | 4 | | 32 | - 51 | 23 | 40 | 3 8 | 48 | 10 | * | . 5 | 6 . | 12 | 64 | 40 | (2) | | 72 | 117 | 25. 2 | 80 | * | 2 | 88 | | 53 | 96 | ns |
|-------------------|--------|-----|----|---|---|---|---|----|------|----|----|-----|----|----|---|-----|-----|----|----|----|-----|---|----|-----|-------|----|---|---|----|---|----|----|----|
| JT CLK | 0 to 1 | | | L | | | | | | | | | | | | | | | | | | J | | | | | | | | | | | _ |
| лr W_EN | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | _ |
| лг register_Mux | 0 | | | | | | ٦ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ лг Instruction1 | 3 | | | ? | | | | į. | | | 2 | | | X | | | | | | | | | | 3 | | | | | | | | | |
| | В | | | | | | X | į. | | | | | | X | | | | 5 | | | | X | | | | 6 | | | | X | | 7 | |

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.

| Signal name | Value | | 16 24 | 32 40 | 48 | 56 64 | 72 80 ns |
|-------------|-------|------|-------|-------|----|-------|----------|
| лг Selector | 1 | | 70. | | | | |
| ± лг Input0 | 02 | UU X | 00 | Х | | 02 | |
| ⊕ лг Input1 | 0F | υυ χ | | 1F | | | ∑ 0F |
| ⊕ лг Output | 0F | υυ χ | 00 | χ 02 | χ | 1F | X OF |

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.

| Signal name | Value | 8 | 16 24 | 32 40 48 | 56 64 | 72 80 |
|------------------|--------|-------|---|---|---|---|
| лг СLК | 0 to 1 | | | | | |
| JL I_W_EN | 0 | | | | | 1 |
| лι O_W_EN | 0 | | | | | 1 |
| ⊕ лг I_regA | 00 | UU | \ 1F | OF OF | 07 | X 00 |
| ⊡ лг I_regB | 1F | UU | X OF | 07 | X00 | (1F |
| ⊞ лт I_regC | 0F | UU | X 07 | 00 | N 1F | QF 0F |
| ⊞ лт I_regD | 07 | UU | X 00 | 1F | OF OF | 07 |
| ⊞ ЛГO_regA | 00 | UU |) 1F | OF | 07 | 00 |
| ⊕ лґ O_regB | 1F | UU | X 0F | 07 |) 00 | |
| ⊕ лт O_regC | 0F | UU | X 07 | 00 | | QF OF |
| ⊞ лг O_regD | 07 | UU | X 00 | 1F | OF OF | V 07 |
| ⊞ .T.I_ALUop | 80000 | UUUUU | 00001 | 00002 | 00004 | 80000 |
| ⊞ ЛГO_ALUop | 80000 | UUUUU | X 00001 | 00002 | X 00004 | 00008 |
| ⊞ лт I_regA_Data | 000 | | 000000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 00000000000000000000000000000000000000 | 000000000000000000000000000000000000000 |
| | 000 | | X 000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 00000000000000000000000000000000000000 | 000000000000000000000000000000000000000 |
| | 000 | | X 000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 000000000000000000000000000000000000000 |
| ⊕ лг O_regA_Data | 000 | | X 000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | (000000000000000000000000000000000000 | X 000000000000000000000000000000000000 |
| ⊕ лл O_regB_Data | 000 | | 000000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 00000000000000000000000000000000000000 | 000000000000000000000000000000000000000 |
| | 000 | | (8000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 000000000000000000000000000000000000000 | 000000000000000000000000000000000000000 |
| | | | | | | |
| Cursor 1 | | | | 1/1 (0 ps - 90 ns) | | 90.0 |

EX Stage:

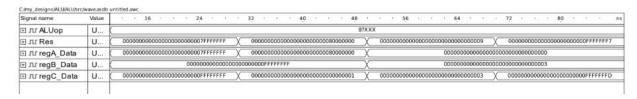
LDI: ALuop holds the index of register and the value to be loaded into regA_Data. As can be seen, the Res signal holds the new value combining ALUop and regA_Data.

| Signal name | Value | 12 16 20 24 28 | 32 36 40 44 48 ns |
|-----------------|-------|--|--|
| ⊞ лг ALUop | 70 | A0000 | 10064 |
| ⊕ лrRes | 02 | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |
| ⊕ лг regA_Data | 00 | FFFFFFFFFFFFFFFFFFFFF0000 X | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |
| ⊕ лл regB_Data | U | υυσυσυσυσυσυσυσυσυσυσυσυσυσυσυσυσυσυσυσυ | JUUUUUUUUUUUUU |
| ⊞ .⊓r regC_Data | U | UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | JUUUUUUUUUUUU |
| | | | |
| | | | |

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



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

| Signal name | Value | · 16 · · · 24 · · · 32 · · · 40 · · · 48 · · · 56 · · · 64 · · · 72 · · · 80 · · · ns |
|----------------|-------|---|
| ⊕ лг ALUop | 8? | 87XXX |
| ⊕ лг Res | FF | 0000000000000000000007FFFFFF X 0000000000 |
| ⊕ лг regA_Data | 00 | 00000000000000000000007FFFFFF X 0000000000 |
| ⊕ лг regB_Data | 00 | 0000000000000000000000000FFFFFFF X 00000000 |
| | FF | 0000000000000000000FFFFFF X 00000000000 |
| | | |

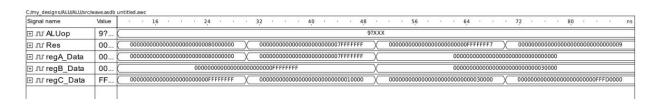
Signed Integer Multiply-Subtract Low with Saturation: The low 16 bits of each word in regB and regC are multiplied and subtracted from corresponding 32 bit regA.

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/v | wave.asdb | untitled.awc |
|-----------------------------|-----------|--|
| Signal name | Value | 1.6 · 2.4 · 3.2 · 4.0 · 4.8 · 5.6 · 6.4 · 7.2 · 8.0 · ns |
| ⊕ лг ALUop | 9? | 97XXX |
| ⊕ л/ Res | 00 | 00000000000000000000000000000000000000 |
| ⊕ лл regA_Data | 00 | 00000000000000000000000000000000000000 |
| ⊕ лл regB_Data | 00 | 00000000000000000000000000000000000000 |
| ⊕ лл regC_Data | FF | 000000000000000000000000FFFFFF X 00000000 |
| | | |
| 1 | 1 | |

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



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

| Signal name | Value | · 16 · · 24 · · 32 · · 40 · · 48 · · 56 · · 64 · · 72 · · 80 · · · ns |
|----------------|-------|---|
| ± лг ALUop | U | A7XXX |
| ± лг Res | U | 000000000000007FFFFFFFFF X 0000000000000 |
| ± лг regA_Data | U | 000000000000007FFFFFFFFF X 0000000000000 |
| ⊕ лг regB_Data | U | 0000000000000000FFFFFFFFFF X 0000000000 |
| ± лг regC_Data | U | 000000000000000FFFFFFFFFF X 00000000000 |

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.

$$((-1) * (-1)) + \max \text{ positive} = \max \text{ positive},$$
 $((-1) * 1) + \max \text{ negative} = \max \text{ negative}$ $(3 * 3) + 0 = 9,$ $((-3) * 3) + 0 = -9$

| Signal name | Value | . 16 · · · 24 · · · 32 · · · 40 · · · 48 · · · 56 · · · 64 · · · · 72 · · · 80 · · · · · · · · · · · · · · · |
|----------------|-------|--|
| ⊞ лг ALUop | A? | A7XXX |
| ⊕ лr Res | FF | 000000000000007FFFFFFFFFF X 000000000000 |
| ⊕ лг regA_Data | 00 | 00000000000007FFFFFFFFF X 00000000000000 |
| ⊕ лг regB_Data | 00 | 00000000000000000FFFFFFFFFFF X 000000000 |
| | FF | 0000000000000FFFFFFFFFF X 0000000000000 |

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.

May positive ((1) * (1)) = may positive ((1) * 1) =

Max negative -
$$((-1) * (-1))$$
 = max negative,
0 - $(3 * 3)$ = -9,

Max positive -
$$((-1) * 1) = \max positive$$

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



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/w | vave.asdb | untitled.awc |
|-----------------------------|-----------|---|
| Signal name | Value | 16 · · · · 24 · · · · 32 · · · · 40 · · · · 48 · · · · 56 · · · · 64 · · · · 72 · · · · 80 · · · · ns |
| ⊕ лг ALUop | B? | B?XXX |
| ⊕ л/ Res | 00 | 0000000000000000000000000 \ 0000000000 |
| ⊕ лг regA_Data | 00 | 000000000000000000000000000 X 000000000 |
| ⊕ лг regB_Data | 00 | 0000000000000000000FFFFFFFFFFF X 00000000 |
| ⊞ л⊥ regC_Data | FF | 000000000000000FFFFFFFFF X 000000000000 |
| | | |

R3 Instructions:

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 FFFFFFF+3 = 100000002 and truncated to 00000002 dude to overflow.

| Signal name | Value | - | 10 | 8 | 600 | 8.0 | 0.7 | 16 | 1 | + | . 2 | 4 | 100 | | 32 | | 38 8 | 40 | 9 | | | 48 | | 1 | 56 | 30 | + | ns |
|----------------|-------|---|-------|---|--------------|-----|------|--------|--------|---------|--------|--------|-------|-----------|--------|-------|--------|------|------|--------|--------|---------|--------|--------|----|----|---|----|
| πr ALUop | ?? | | UUUUU | | X | | | | | | | | | | | | ???XX | | | | | | | | | | | |
| ∄ лư Res | 00 | | | | \supset | | 0000 | 000000 | 000000 | 0000000 | 000200 | 000003 | | \supset | | | | | 0000 | 000200 | 000003 | 8000000 | 000000 | 000000 | | | | |
| ∄ лг regA_Data | 00 | | | | \mathbb{X} | | 0000 | 000000 | 000000 | OOFFFF | FFFF00 | 000003 | | \supset | | | | | FFFF | FFFF00 | 000003 | 8000000 | 000000 | 000000 | | | | |
| ± лг regB_Data | 00 | | | | \propto | | 0000 | 000000 | 000000 | 0000000 | 000300 | 000000 | | \supset | | | | | 0000 | 000300 | 000000 | 0000000 | 000000 | 000000 | | | | _ |
| ∄ лг regC_Data | U | | | | | | | | | | | UUUUU | UUUUU | JUUUUI | JUUUUI | JUUUL | JUUUUL | JUUU | | | | | | | | | | |

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.

| Signal name | Value | | 12 | 4 + | ×. | 16 | X | | | 20 | - | 4 | | 24 | | 221 | | 28 | | | 3 | 2 . | | | 36 | i. | 4 | 40 | 100 | 44 | - 1 | 20 | 48 |
|-----------------|-------|--|--|-----|----|----|---|--|--|----|---|---|--|----|---|------|------|------|------|-----|------|------|------|----|----|----|---|----|---------|----|-----|----|----|
| ⊞ лг ALUop | ?? (| | | | | | | | | | | | | | | | | | 777 | XX | | | | | | | | | | | | | |
| ⊕ лư Res | 00 (| | 00000000000000000000000000000000000000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ .ru regA_Data | FF (| Α | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ лл regB_Data | 00 | 00000000000000000000000000000000000000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ лгregC Data | U | | | | | | | | | | | | | | U | UUUU | UUUL | JUUU | UUUU | UUU | JUUU | UUUI | JUUU | UU | | | | | | | | | |

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.

| Signal name | Value | 12 16 20 | 36 40 44 48 n |
|----------------|-------|---|---|
| | ?? | . (777XX | |
| ⊞ ли Res | 80 | . (00000000000000000000000000000000000 | 000000000000000000FFFD00000000 |
| | 80 | . (000000000000000000000000000000000000 | 000000000000000000000000000000000000000 |
| ⊕ лг regB_Data | FF | . (00000000000000000000000000000000000 | 0000000000000000FFFDFFD00000000 |
| | U | 000000000000000000000000000000000000000 | J |
| | | | |

AND: Res store the AND result of regA Data and regB Data.

| Signal name | Value | * 1 | | 8 | (6) | 16 | - 1 | -1 | 24 | 4 | - 1 | | 32 | 1 | 400 | 10 | 40 | 4. | (-) | 48 | (4) | | 56 | | ns |
|--------------------|-------|-----|-------|---|-----------|----|-----|----|----|-------|------|------|--------|-------|--------|-------|--------|-------|-----|----|-----|--|----|--|----|
| ∃ .π. ALUop | U | | UUUUU | | \supset | | | | | | | | | | ???> | X | | | | | | | | | |
| ∄ лг Res | U | | | | \supset | | | | | | | F | FFFFF | F0000 | 000000 | 00000 | 000000 | 00000 | | | | | | | |
| ∄ лг regA_Data | U | | | | \supset | | | | | | | F | FFFFFF | F0000 | 000000 | 00000 | 00FFF | FFFFF | | | | | | | |
| ⊕ лг regB_Data | U | | | | \supset | | | | | | | F | FFFFF | F0000 | 00000F | FFFFF | FF000 | 00000 | | | | | | | |
| | U | | | | | | | | UL | JUUUL | UUUU | UUUU | UUUUL | JUUUU | UUUU | UUUU | U | | | | | | | | |

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

| □ IT ALUop ?? UUUUU X ???XX □ IT Res FF X FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF | gnal name | Value | | 16 24 32 40 48 56 |
|--|--------------|-------|-------|---|
| □ πregA_Data 00 | лг ALUop | ?? | 00000 | mxx |
| | лгRes | FF | | FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |
| ⊕ nregB_Data U uuuuuuuuuuuuuuuuuuuuuuuuu | лг regA_Data | 00 | | 000000000000000000FFFFFF |
| | лг regB_Data | U | | บบบนบนบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบ |
| □ rr regC_Data U Uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu | лг regC_Data | U | | บบบบบบบบบบบบบบบบบบบบบบบบบบบบบบบ |

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 FFFFFFE, 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 FFFFFFF which has no 0 in it. Thus Res is 0 for the least significant word.

| Value | 10 | 0.000 | 8 | | | 10 | 16 | 4 | 6 | | 2,4 | | | | 32 | | | * 1 | 40 | | ** | 100 | 48 | 6.0 | 100 | | 5,6 | | | ns |
|-------|-------------|---------|----------------------|---|--|----|----|---|---|---|-----|-----------------------------------|---------------------|------|--|--|--|--|--|---|--|--|--|--|--|---|---|--|---|---|
| U | | UUUUU | | \supset X | | | | | | | | | | | | | ???X | Х | | | | | | | | | | | | |
| U | | | | \supset X | \(\) 0000020000001000000F0000000 \(\) 0000000000000000000000000000000000 | | | | | | | | | | | | | | | | | | | | | | | | | |
| U | | | | \supset X | X 00000020000000100000000000000000000000 | | | | | | | | | | | | | | | | | | | | | | | | | |
| U | | | | | | | | | | | UL | JUUUU | UUUU | JUUU | UUUUI | JUUUU | JUUUUL | JUUU | U | | | | | | | | | | | |
| U | | | | | | | | | | | UL | JUUUU | UUUU | JUUU | UUUUI | JUUUU | JUUUUL | JUUU | U | | | | | | | | | | | |
| | U U U | U U U U | U UUUUUU U U U UUUUU | U UUUUUU UUUUU UUUUU UUUUU UUUUU UUUUU UUUU | U UUUUU X U X U X U X | U | U | U | U | U | U | U UUUUU X U X U X U X U U U X U X | U U X U X U X U U X | U | U UUUUU X U X 0 U X 0 U X 0 U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X U X 0000000 U X 0000000 U X 0000000 U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X U X 00000200000 U X 00000000FFF U X 00000000FFF U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X ???X U X 0000002000000010 U X 00000000FFFFFFE0 U X 00000000FFFFFFE0 U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X 777XX U X 000000200000001000000 U X 00000000FFFFFFE0018 U X 00000000FFFFFFE0018 U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X 7777XX U X 00000020000000100000000000000000000000 | U UUUUU X ???XX U X 00000020000001000000000000000000000000 | U UUUUU X ???XX U X 00000020000000100000000000000000000000 | U UUUUU X ???XX U X 00000200000010000000000000000000000000 | U UUUUU X ???*XX U X 00000020000000100000000000 U X 00000000FFFFFFE0018000FFFFFFF U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X 777XX U X 00000020000000100000000000000000000000 | U UUUUU X 7777XX U X 00000020000000100000000000000000 U X 00000000FFFFFFE0018000FFFFFFF U X 00000000FFFFFFFE0018000FFFFFFF U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU | U UUUUU X ???xxx U X 00000020000000100000000000000000000000 | U UUUUU X ???XX U X 00000020000001000000000000000000000000 | U UUUUU X 7777XX U X 00000200000010000000000000000000000000 | U UUUUU X 7777XX U X 0000002000000010000000000 U X 00000000FFFFFFFE U X 00000000FFFFFFFF U UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU |

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

| C:/my_designs/ALU/ALU/src/\ | wave.asdb | untitled.aw | c | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|-----------|-------------|-------|----|-----------|---|-----|--------|---------|--------|-------|--------|-------|------------|-------|-------|-------|------|-----|-------|-------|-------|--------|-------|-------|----|-----|--|------|
| Signal name | Value | - 3 | | 8 | 8 - | 10 | 59 | 1,6 | 0 | 7 | | 2,4 | | | 3,2 | | 100 | 19 | 4,0 | 1 | 10 | | 4,8 | - | 4 | 61 | 5,6 | | · ns |
| | ?? | | UUUUU | Ų. | \supset | | | | | | | | | | | | ???) | (X | | | | | | | | | | | |
| ⊕ лr Res | 00 | | | | | X 80018001000300030000000000000 X 0000000000000 | | | | | | | | | | | | | | 01000 | 30003 | 3 | | | | | | | |
| ⊕ лг regA_Data | 00 | | | | | X 0001FFFE00030000000000000000000000000000 | | | | | | | | | | | | | | | FE000 | 30000 |) | | | | | | |
| | 00 | | | | | | 800 | 07FFF0 | 0000003 | 300000 | 00000 | 000000 |) | $=$ χ | | | | | 00 | 00000 | 00000 | 30000 | 3000FF | FF000 | 00003 | 3 | | | |
| ⊞ лг regC_Data | U | | | | | | | | | | | UUUU | UUUUL | JUUUU | JUUUU | UUUUL | JUUUU | UUUU | U | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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.a | awc | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------|-----------|------------|-----|-----|---|--|--|--|----|--|--|--|----|------|-------|-------|--------|-------|--------|-------|--------|------|---|----|--|-----|--|------|
| Signal name | Value | | | | 8 | | | | 16 | | | | 24 | | | | 3,2 | | | | 40 | | | 48 | | 5,6 | | · ns |
| ⊞ лг ALU op | U | | UU | UUU | | \supset X | | | | | | | | | | | | | ?? | ?XX | | | | | | | | |
| ⊕ лır Res | U | | | | | \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ лл regA_Data | U | | | | | X 000000000000000000000000000000000000 | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ лг regB_Data | U | | | | | \exists X | | | | | | | | | | (| 000000 | 00000 | 000003 | BFFFF | FFFFFF | FFFF | F | | | | | |
| ⊕ лг regC_Data | U | | | | | | | | | | | | Ü | UUUU | JUUUI | JUUUI | JUUUU | UUUU | UUUU | UUUL | JUU | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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/v | wave.asdb | untitled.awo | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|-----------|--------------|-------|---|-------------------|----|----|-----|------|-----|-----|------|------|------|--------|-------|-------|-------|--------|-------|----|---|----|----|----|---|-----|----|---|--------|
| Signal name | Value | - 6 | | ė | O. | 7. | 16 | į r | - 65 | 970 | 2,4 | 77 | 7 | 7 | 3,2 | ¥. | - 70 | 500 | 4.0 | - 1 | 37 | × | 48 | ** | 10 | 0 | 5,6 | 39 | В | · ns |
| ∄ JJ ALUop | U | | UUUUU | | \supset | | | | | | | | | | | | ???? | XX | | | | | | | | | | | | |
| ⊞ лг Res | U | | | | X 00000007FFFFFFF | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⊕ лг regA_Data | U | | - 1 | | \supset | | | | | | | | | 00 | 000000 | 38000 | 00000 | 80000 | 0007FF | FFFFF | | | | | | | | | | |
| ⊕ лг regB_Data | U | | - | | \supset | | | | | | | | | 00 | 000000 | OFFF1 | FFFE | 7FFFF | FFF7FF | FFFFF | | | | | | | | | | |
| ⊕ лг regC_Data | U | | | | | | | | | | UU | UUUU | UUUU | UUUU | UUUUL | JUUUL | JUUUL | JUUUL | JU | | | | | | | | | | | |
| | | | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | \neg |

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

| C:/my_designs/ALU/ALU/src/ | wave.asdb | untitled.a | WC | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------|-----------|------------|-----|----|----|---------------|--|-----|----|----|-----|-----|-------|-------|------|-------|-------|--------|-------|-------|-------|---|--------|----|----|-----|--|------|
| Signal name | Value | | | | ė. | 34 | | 1,6 | 10 | 65 | - 1 | 2,4 | 14 | 3. | 9. | 3,2 | - 6 | | | 40 | 19 | 3 | 48 | 10 | 65 | 5,6 | | · ns |
| ⊞ лг ALUop | U | | UUU | υU | | \mathcal{X} | | | | | | | | | | | | ???XX | | | | | | | | | | |
| ⊕ лг Res | U | | | | | \supset | | | | | | | | | F | FFFFF | FFFFF | FFFF00 | 00000 | OFFF | FFFFF | | | | | | | |
| ⊕ лг regA_Data | U | | | | | \supset | | | | | | | | | 0 | 00000 | OFFFF | FFFF00 | 0000 | 0FFFI | FFFFF | | | | | | | |
| ⊞ лг regB_Data | U | | | | | \supset | | | | | | | | | F | FFFFF | FFFFF | FFFF00 | 00000 | 00000 | 00000 | | | | | | | |
| ⊕ лг regC_Data | U | | | | | | | | | | | UL | JUUUL | JUUUU | UUUU | UUUUI | JUUUU | UUUUU | JUUU | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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 (FFFFFFF) 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/v | vave.asdb | untitled.aw | vC . | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|-----------|-------------|-------|----|-------------|--|-----|----|-----|-------|------|------|--------|-------|-------|-------|-------|-------|---|---|-----|--|-----|-----|---|---|------|
| Signal name | Value | | | | | | 1,6 | 0) | 2.4 | 1 | | 1 | 3,2 | 8 | - | | 4.0 | | | 1 | 4.8 | | - 6 | 5,6 | 1 | 9 | · ns |
| ⊞ .T.r AL.Uop | U | | UUUUU | 10 | =X | | | | | | | | | | ??? | XX | | | | | | | | | | | |
| ⊞ лг Res | U | | | | = | | | | | | | 0 | 000000 | 00000 | 0001F | 00000 | 01E00 | 00020 |) | | | | | | | | |
| ⊞ лг regA_Data | U | | | | \supset X | | | | | | | 0 | 000000 | OFFFF | FFFFE | FFFE7 | FFFFF | FFFFF | | | | | | | | | |
| ⊞ лг regB_Data | U | | | | | | | | UL | IUUUU | UUUU | UUUU | IUUUUI | JUUUL | JUUUI | JUUUL | JU | | | | | | | | | | |
| ⊞ лг regC_Data | U | | | | | | | | UL | IUUUU | UUUU | UUUL | IUUUUI | JUUUL | JUUUI | JUUUL | JU | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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/v | vave.asdb | untitled.aw | C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|-----------|-------------|-----|----|----|---------------|-----|------|--------|-------|-------|-------|--------|-------|------|-----------|-------|-------|-------|-------|-------|-------|-------|-------|--------|--------|------|-----|-----|-----|-----|------|
| Signal name | Value | 7 | × | Ŧ | ė. | 75 | 100 | 19 | 16 | 34 | × | 77 | 2,4 | 10 | - 61 | N. | 3,2 | 34 | - 1 | ¥ | 4.0 | T | 70 | - 1 | 4,8 | 9 | - 17 | 100 | 5,6 | - 1 | 100 | · ns |
| | U | | UUU | UU | | \mathcal{X} | | | | | | | | | | | | | ??? | XX | | | | | | | | | | | | |
| ⊞ лл Res | U | | | | | X | | 7C00 | 000000 | 00000 | 00000 | 00000 | 000000 | 0000 | | \supset | | | | | 3 | E0000 | 00000 | 00000 | 000000 | 000000 | 0000 | 00 | | | | |
| ⊞ лι regA_Data | U | | | | | \mathcal{X} | | | | | | | | | | E | 00000 | 00000 | 00000 | 00000 | 00000 | 00000 | 13 | | | | | | | | | |
| ⊞ лг regB_Data | U | | | | | \mathcal{X} | | 0000 | 00000 | 00000 | 00000 | 00000 | 000000 | 0003 | | \supset | | | | | 0 | 00000 | 00000 | 00000 | 000000 | 000000 | 0001 | 84 | | | | |
| ⊞ лл regC_Data | U | | | | | | | | | | | | UU | IUUUU | UUUU | UUUU | UUUU | JUUUL | JUUU | UUUUI | JU | | | | | | | | | | | |
| | | | | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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/v | rave.asdb | untitled.aw | C: | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|-----------|-------------|-------|---|---|--------|----|-----|----|-----|---|----|-----|------|------|------|--------|-------|----------|-------|-------|-------|---|-----|----|----|----|----|-----|---|-----|------|
| Signal name | Value | (18) | | | 8 | 48 | 10 | 100 | 16 | 100 | × | 15 | 2,4 | 75 | 3 | 29 | 3,2 | | S4 | - 4 | 0 | × | ā | 100 | 48 | 97 | -5 | 40 | 5,6 | 9 | (%) | · ns |
| ⊕ лг ALUop | U | | UUUUI | U | | χ | | | | | | | | | | | | | ???XX | | | | | | | | | | | | | |
| ⊕ лr Res | U | | | | | X | | | | | | | | | | 00 | 03E000 | 003E0 | 000003E0 | 00000 | 0000 | 3C00 | | | | | | | | | | |
| | U | | | | | X | | | | | | | | | | E | 000000 | 3E000 | 00003E00 | 00003 | C000 | 00003 | | | | | | | | | | |
| ⊕ лг regB_Data | U | | | | | X | | | | | | | | | | 00 | 000000 | C000 | 0008000 | 00004 | 10000 | 00034 | | | | | | | | | | |
| | U | | | | | | | | | | | | UU | JUUU | UUUU | UUUU | UUUUL | UUUL | JUUUUUL | UUU | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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/R | 3_15.asdb | C:/my | desig | ns/AL | .U/R3_ | 15/R | 3_15.a | awc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------|-----------|-------|-------|-------|--------|------|--------|------|-------|-----|-------|------|-----|------|------|-----|---|------|------|------|------|------|-----------|------|------|------------|-----|----|---|-----|------|-----|------|-------|------|-----|--------|-----|-----|---|---|---|---|----|
| Signal name | Value | * | 1,2 | 60 | | 60 | 1,6 | | 80 | | 20 | | - | • | e : | 2,4 | • | 3.5 | | 2,8 | | | * | | 3,2 | A) | | | 3 | 6 | 66 | | | 40 | 0.0 | | •0 · 0 | | 4.4 | • | , | 4 | 8 | ns |
| | ?? | | | | | | | | | | ??C?) | K | | | | | | | | | | | \subset | | | | | | | | | | | ??D? | K | | | | | | | | | |
| ⊕ лr Res | 00 | | | | | | 00 | 0000 | 03000 | 000 | 03000 | 0000 | 030 | 0000 | 0003 | 0 | | | | | | | \subset | | | | | | | 000 | 0030 | 000 | 0003 | 30000 | 0000 | 300 | 0000 | 030 | 0 | | | | | |
| ⊕ лг regA_Data | E0 | | | | | | | | | | | | | | | | | E | 0000 | 003E | 0000 | 0003 | BE000 | 0000 | 3C0 | 0000 | 003 | | | | | | | | | | | _ | | | | | | |
| ⊕ лг regB_Data | U | | | | | | | | | | | | | | | | U | JUUL | UUU | IUUU | JUU | UUU | JUUU | IUUL | JUUL | JUUI | JUU | UU | | | | | | | | | | | | | | | | |
| | U | | | | | | | | | | | | | | | | U | JUUL | UUL | IUUU | JUU | UUU | JUUU | IUUL | JUUL | JUUI | JUU | UU | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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/w Signal name | vave.asdb Value | untitled.awc | 32 36 40 44 48 . ns |
|--|--------------------|--|----------------------------------|
| ⊕ лг ALUop | ?? | | 777XX |
| ⊕ лґ Res | 00 | 000000000000000000000000004FFFC | X 000000000000000004FFFC00000000 |
| | FF | 00000000000000000000000000000000000000 | X 000000000000000FFF00030000000 |
| ⊕ лг regB_Data | 00 | 000000000000000000000000003FFFF | X 000000000000000003FFFF00000000 |
| | U | UUUUUUUUUUUUUUU | UUUUUUUUUUUUUUUUU |
| | | | |
| | | | |

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

| Signal name | Value | | | 8 | | - | 100 | 16 | | | | 2.4 | - | | 20 | 3,2 | - | | | 40 | | 23 | | 48 | 10 | 23 | | 5,6 | 1.5 | 10 | · ns |
|----------------|-------|-----|----|---|---|---|------|--------|--------|-------|-------|-------|------|------|------|-------|-------|-------|------|----|-------|--------|-------|-------|--------|--------|---|-----|-----|----|------|
| ⊕ лг ALUop | U | UUU | UU | | X | | | | | | | | | | | | | ??? | ΧX | | | | | | | | | | | | |
| ⊕ лг Res | U | | | | X | | 0000 | 000000 | 000000 | 00000 | 00004 | FFFFF | FFC | | X | | | | | 0 | 00000 | 04FFFI | FFFFC | 00000 | 00000 | 00000 | 0 | | | | |
| ⊞ лг regA_Data | U | | | | X | | 0000 | 000000 | 000000 | OFFFF | FFFF | 00000 | 003 | | X | | | | | F | FFFF | FF000 | 00003 | 00000 | 000000 | 000000 |) | | | | |
| ⊕ лг regB_Data | U | | | | X | | 0000 | 000000 | 000000 | 00000 | 00003 | FFFFF | FFF | | X | | | | | 0 | 00000 | 03FFFI | FFFFF | 00000 | 000000 | 000000 |) | | | | |
| | U | | | | | | | | | | | UU | UUUU | UUUU | JUUU | JUUUL | JUUUL | JUUUI | UUUU | JU | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

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.

| Signal name | Value | 12 | 16 20 24 | 28 | 32 40 44 48 ns |
|----------------|-------|----|---|-----------------|--------------------------------|
| ± лг ALUop | ?? | | | ???XX | |
| ± лґ Res | 7F | | 00000000000000000000000000000000000000 | Х | 0000000000000007FFF00030000000 |
| ± лг regA_Data | FF | | 000000000000000000000000000000000000000 | Χ | 000000000000000FFDFFD00000000 |
| ± лг regB_Data | 7F | | 000000000000000000000000000000000000000 | X | 0000000000000007FFF00000000000 |
| ± лг regC_Data | U | | υυυυυυ | UUUUUUUUUUUUUUU | UUUUUUUUUUU |

NAND: Res stores the result of regA_Data NAND regB_Data

| Signal name | Value | 8 | | 40 | 8 | | | 16 | 41 | 2.4 | | 37 | | 3,2 | | 80 | | 40 | 40 | 7 | 48 | 10 | | 56 | | · ns |
|----------------|-------|---|-----|-----|---|-----------|--|----|----|-----|-------|-------|------|--------|-------|-------|-------|--------|-------|---|----|----|--|----|--|------|
| ⊞ лг ALUop | U | | UUI | JUU | | \supset | | | | | | | | | | ???X | X | | | | | | | | | |
| ⊕ лг Res | U | | | | | \supset | | | | | | | F | FFFFFF | F0000 | 0000F | FFFF | FFFFF | FFFFF | | | | | | | |
| ⊞ лг regA_Data | U | | | | | \supset | | | | | | | 0 | 000000 | OFFFF | FFFF0 | 00000 | 000FFF | FFFFF | | | | | | | |
| ⊞ лл regB_Data | U | | | | | X | | | | | | | F | FFFFFF | FFFFF | FFFF0 | 00000 | 000000 | 00000 | | | | | | | |
| ⊞ лл regC_Data | U | | | | | | | | | U | UUUUL | JUUUU | UUUU | UUUU | JUUUU | UUUU | UUUU | IU | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | |

EX/WB Register: 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.

| Signal name | Value | 8 | | | - 8 | 16 | | 13 | | - 2 | 24 | | 8 | | 32 | 2 ' | | * | 4 | 40 | | | 10 | | 48 | 3 | | 5 8 | 1 | 56 | 5 | * | 35 | 64 | | 1 | | 72 | 3 | | | 8 | 0 | | 0.0 | 20 | ns |
|-------------|--------|----|-----------|----|-----|------|-----|-----|-----|-----|-----|-----|-----|----|-----|------|-----|------|-----|----|-----|-----|-----|-----|-----|-----|---------------|-----|-----|-----|-----|-----|-----|-----|-----|------|-----|------|------|------|-----|------|-----|-----|-----|------|------|
| JL CLK | 0 to 1 | | | | | | | | | | | | | | | | | | | l | | | | | | j | | | | | | 1 | | | | | | | | | | | | | | | |
| .πι_W_EN | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | Г | | | | | | | | | | | | | | | | | | | | |
| JT O_W_EN | 0 | | | | | | | | | | | | | | | | | | | | | | | | | - 4 | Г | | | | | | | | | | | | | | | | | | | | |
| ⊕ лг I_Res | 000 | | \supset | FF | FFF | FFFF | FFF | FFF | FFF | FF | FFF | FFF | FFF | EX | FFF | FFFF | FFF | FFFF | FFF | FO | 000 | 000 | 000 | 000 | 000 | 00 | (00 | 000 | 000 | 000 | 000 | OFF | FFF | FFF | FFF | FFF | EX | 0000 | 0000 | 0000 | 000 | 000 | 000 | 000 | 000 | 0000 | 000 |
| ⊡ лг O_Res | 000 | | | FF | FFF | FFFF | FFF | EX | FFF | FFFF | FFF | FFFF | FFF | FO | 000 | 000 | 000 | 000 | 000 | 00) | (00 | 000 | 000 | 000 | 000 | OFF | FFF | FFF | FFF | FFFF | EX. | 0000 | 0000 | 0000 | 000 | 0000 | 000 | 000 | 000 | 0000 | 0000 |
| ⊕ лг I_regD | 00 | UU | | | | | | 11 | | | | | | X | | | | | | 18 | | | | | | | $\overline{}$ | | | | | 07 | | | | | X | | | | | 00 | 0 | | | | |
| ⊕ лгO regD | 00 | UU | 7 | | | | | 1 | 5% | | | | | X | | | | | | 18 | | | | | | 7 | $\overline{}$ | | | | | 07 | | | | | X | | | | | 00 | 0 | | | | _ |

4-Stage Pipelined Multimedia Unit:

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.

| Signal name | Value | | 128 | 136 14 | 14 9 9 6 8 | 152 160 | 168 | 176 |
|--------------------|--------|---------|-----------------------|---|---|---|---------|---------|
| | 00 (| | 06 | Х | 07 | X | 08 | |
| Instruction | 000 | | 0000082 | X | 00000C3 | X | 1018824 | |
| JT CLK | 0 | | | | | | | |
| Instruction_i | UUU | 0000041 | $\overline{}$ | 0000082 | X | 00000C3 | X | 1018824 |
| л Instruction_i | UUU | 0000000 | \square X \square | 0000041 | X_ | 0000082 | X | 00000C3 |
| JT ALU_Output | יייטטע | | UUUUUUUU | 000000000000000000000000000000000000000 | X | UUUUUUUUUUUUUUUUUUUUUUUUUUUU | | |
| .rr Data_inWB | UUU | | | UUUUUUUUUUUUUU | 000000000000000000000000000000000000000 | 00 | X | |
| JT ALUop_inID | UU | | | | 00 | | | |
| лг ALUop_inEX | UU | | | | 00 | | | |
| J Input_rs3 | UUU | | | UUL | | UUUUUUUUU0000 | | |
| л Input_rs2 | UUU | | | UUU | | UUUUUUUUU0000 | | |
| Ju Input_rs1 | UUU | | | UUUUUUUUUUUUUUU | 000000000000000000000000000000000000000 | 00 | =X $=$ | |
| лг W_En_inWB | U | | | | | | | |
| л/ Forward_Mu | 0 | | | | | | | |
| лг Forward_Mu | 0 | | | | 69 | | | |
| лг Forward_Mu | 0 | | | | | | | |
| Signal name | Value | 184 | | 92 200 | 208 | 216 224 | 2 | 12 |
| ☐ ☐ Instruction_I | 00 | | 09 | X | 0A | X | OB | |
| ∃ лг Instruction | 000 | | 1218826 | X | 0200082 | X | 02000C3 | |
| JT CLK | 0 | | | | | | | |
|] лг Instruction_i | UUU | 1018824 | X | 1218826 | X_ | 0200082 | X_ | 02000C3 |
|] лг Instruction_i | UUU | 00000C3 | X_ | 1018824 | X_ | 1218826 | X_ | 0200082 |
| J.T. ALU_Output | UUU | | X_ | 000000000000000000000000000000000000000 | | XXXXXXXXXXXXXXXXXXXXXXXXXXXXX | X_ | |
| лг Data_inWB | UUU | 11000 | X_ | 000000000000000000000000000000000000000 | 000004 X | 000000000000000000000000000000000000000 | X_ | |
| ЛГ ALUop_inID | UU | 00 | X | 03 | X_ | 43 | X_ | 40 |
| TT ALUOP_inEX | UU | | | 00 | X_ | 03 | X_ | 43 |
|] Jul Input_rs3 | UUU | | | | UUUUUUUUUUUUUUU | | X_ | |
|] л₁ Input_rs2 | UUU | | | | 000000000000000000000000000000000000000 | | X | |
| Tr Input_rs1 | UUU | | | | 000000000000000000000000000000000000000 | 0000000000000 | | |
| | U | | | | | | | |
| лг W_En_inWB | | | | | | | | |
| лг Forward_Mu | 0 | | | | | | | |
| | 0 | | | | | | | |