ESE 345: Computer Architecture

Pipelined SIMD Multimedia Unit Design

Part 1: Multimedia Arithmetic Logic Unit

Kyle Han 113799110

Summer Wang 113961753

Table of Contents

About

This project report shows the various testing steps and implementation strategies of the ALU. All registers have been hard coded as 00000, as the registers are not yet specified in this part of the ALU. We will be loading the registers values through the inputs Rs1, Rs2, and Rs3. Rd is our output no matter what. Our Opcode is defined here as “wordIn”, since R3 instructions technically have their own opcode.

Code

All code for this project can be found at the following github repo:

https://github.com/Kyleh2420/Pipelined-SIMD-Multimedia-Unit

For the sake of conciseness, I will not be including the code functions themselves in this document. All code has been written conditionally using only VHDL language that can be synthesized. The hope is to be able to synthesize and actually implement the whole project on a FPGA. For the sake of this project, we have coded the ALU behaviorally.

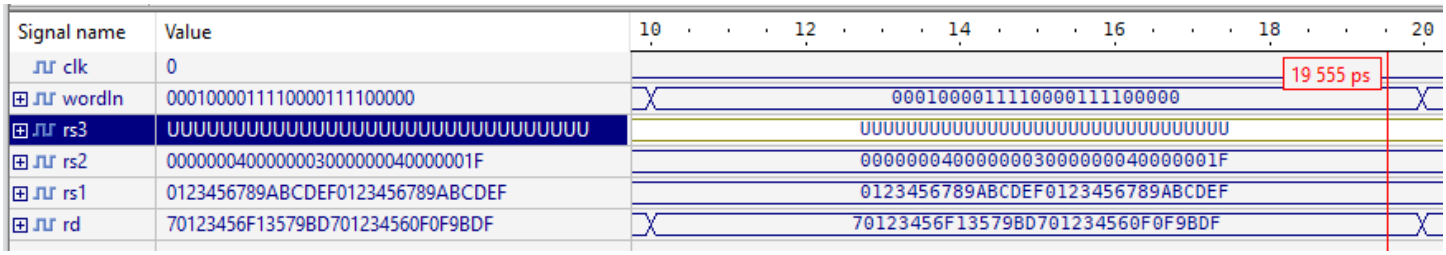
Load Immediate

Load Immediate Test 1

Word In: 0 001 0000 1111 0000 1111 00000

Explanation: Rd, the destination register should now have the 16 bit value defined (0x0F0F0) in the 1st index.

Simulation Result:

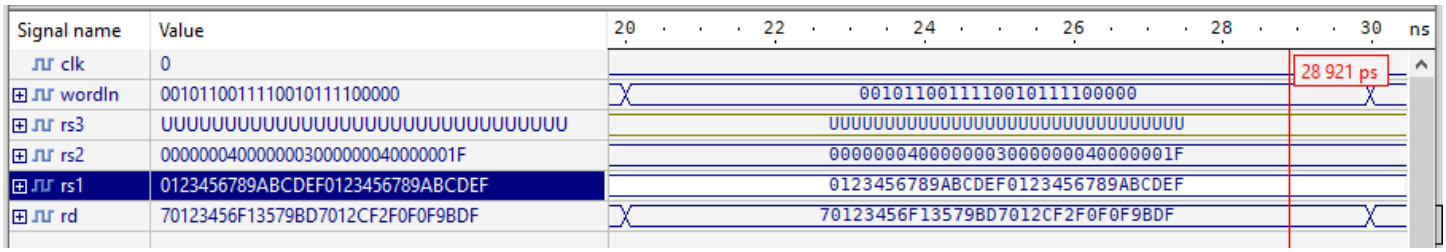


Load Immediate Test 2

Word In: 0 010 1100111100101111 00000

Explanation: Rd, the destination register should now have the 16 bit value defined (0xCF2F) in the 2st index.

Simulation Result:

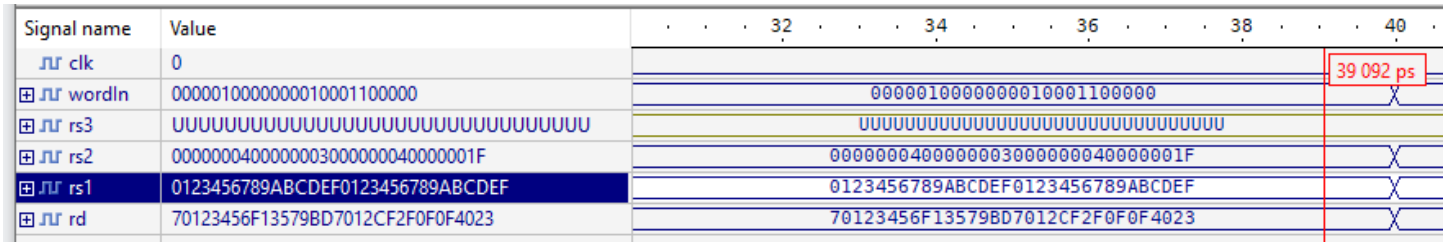


Load Immediate Test 3

Word In: 0 000 1100000000100011 00000

Explanation: Rd, the destination register should now have the 16 bit value defined (0x4023) in the 0st index.

Simulation Result:

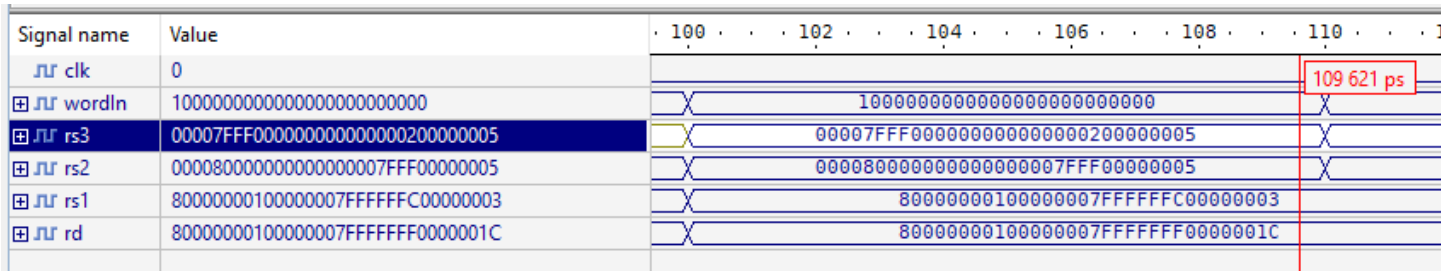

Signal name
Value
32
34
36
38
40
clk
0
39 092 ps
wordln
00000100000000010001100000
rs3
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
rs2
0000000400000003000000040000001F
rs1
0123456789ABCDEF0123456789ABCDEF
rd
70123456F13579BD7012CF2F0F0F4023

R4 Instructions

R4 Instruction: 000 - Integer Multiply-Add Low w/ Saturation
Function: Integer Multiply-Add Low w/ Saturation
WordIn: 1 0 000 00000 00000 00000 00000
Rs1: 0x80000000100000007FFFFFFC00000003
Rs2: 0x00008000000000000007FFF00000005
Rs3: 0x00007FFF0000000000000000200000005
Explanation: Multiply the two low 16 bit values in R2 and R3 together. Then, add the 32-bit value in R1.

--3rd	--2nd	--1st	--0th
--Testing the normal operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x8000, The lowest 16 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF, The Highest 16 bit number --That is: (32,767*-32,768) = -1,073,709,056 - 2147483647 = Saturate to 8000_0000	--Testing the normal operation --Set the 1st set of 32 bits of rs1 to 0x1000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x0000, 0 --Set the 1st set of 16 bits of rs3 to 0x0000, 0 --That is: (0*0) = 0 - 4,294,967,296 = -4,294,967,296, or 0x1000_0000	--Testing the overflow portion --Set the 1st set of 32 bits of rs1 to 0x7FFF_FFFC, the hightest 32 bit number - 3 --Set the 1st set of 16 bits of rs2 to 0x7FFF, the highest 16 bit --Set the 1st set of 16 bits of rs3 to 0x0002, 2 --That is: (32,767*2) = 65,534 + 4,294,967,296 = 4,295,032,830, which overflowed --Therefore, saturation should set it at 0x7FFF_FFFF	--Set the 0th set of 32 bits of rs1 to 3 --Set the 0th set of 32 bits of rs2 to 5 --Set the 0th set of 32 bits of rs3 to 5 --That is: (5*5) + 3 = 28, which is 0x0_001C

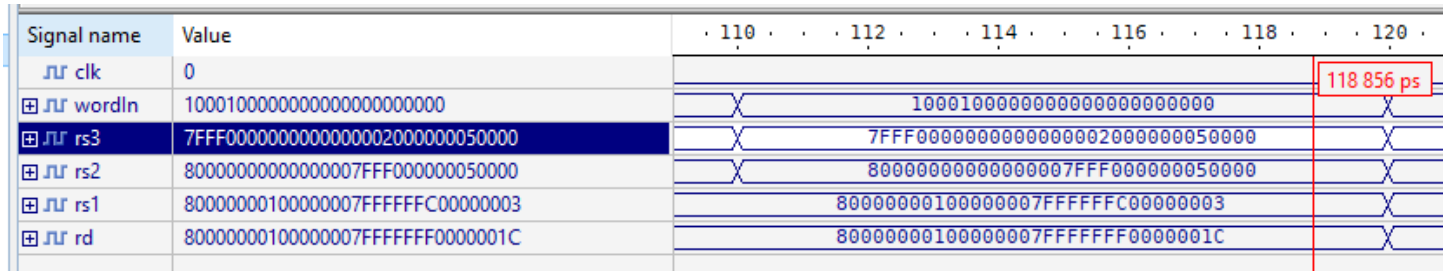
Simulation Result:



R4 Instruction: 001 - Integer Multiply-Add High w/ Saturation
Function: Integer Multiply-Add High w/ Saturation
WordIn: 1 0 001 00000 00000 00000 00000
Rs1: 0x80000000100000007FFFFFFC00000003
Rs2: 0x80000000000000007FFF000000050000
Rs3: 0x7FFF0000000000000002000000050000
Explanation: Multiply the two high 16 bit values in R2 and R3 together. Then, add the 32 bit value in R1.

--3rd	--2nd	--1st	--0th
--Testing the underflow operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x8000, The lowest 16 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF, The Highest 16 bit number --That is: (32,767*-32,768) = -1,073,709,056 - 2147483647 = Saturate to 8000_0000	--Testing the normal operation --Set the 1st set of 32 bits of rs1 to 0x1000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x0000, 0 --Set the 1st set of 16 bits of rs3 to 0x0000, 0 --That is: (0*0) = 0 - 4,294,967,296 = -4,294,967,296, or 0x1000_0000	--Testing the overflow portion --Set the 1st set of 32 bits of rs1 to 0x7FFF_FFFC, the hightest 32 bit number - 3 --Set the 1st set of 16 bits of rs2 to 0x7FFF, the highest 16 bit --Set the 1st set of 16 bits of rs3 to 0x0002, 2 --That is: (32,767*2) = 65,534 + 4,294,967,296 = 4,295,032,830, which overflowed --Therefore, saturation should set it at 0x7FFF_FFFF	--Set the 0th set of 32 bits of rs1 to 3 --Set the 0th set of 32 bits of rs2 to 5 --Set the 0th set of 32 bits of rs3 to 5 --That is: (5*5) + 3 = 28, which is 0x0_001C

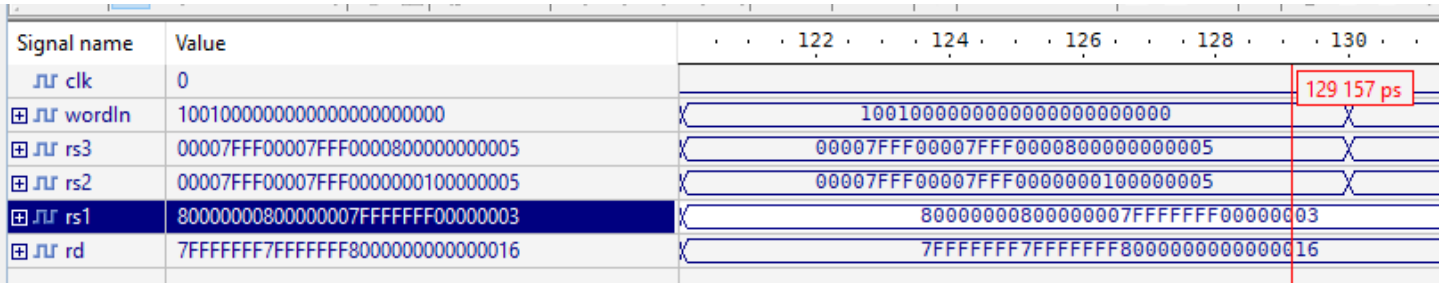
Simulation Result:



R4 Instruction: 010 - Integer Multiply-Subtraction Low w/ Saturation
Function: Integer Multiply-Subtraction Low w/ Saturation
WordIn: 1 0 010 00000 00000 00000 00000
Rs1: 0x800000008000000007FFFFFFF00000003
Rs2: 0x00007FFF00007FFF0000000100000005
Rs3: 0x00007FFF00007FFF0000800000000005
Explanation: Multiply the two Low 16 bit values in R2 and R3 together. Then, subtract the 32 bit value in R1.

--3rd	--2nd	--1st	--0th
--Testing the Overflow operation --Testing the Overflow operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x7FFF, the highest 16 bit --Set the 1st set of 16 bits of rs3 to 0x7FFF, the highest 16 bit --That is: (32,767*32,767) = 1,073,676,289 - -2,147,483,648 = 3,221,159,937, which overflowed --Therefore, saturation should set it at 0x7FFF_FFFF	--Testing the Overflow operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x7FFF, the highest 16 bit --Set the 1st set of 16 bits of rs3 to 0x7FFF, the highest 16 bit --That is: (32,767*32,767) = 1,073,676,289 - -2,147,483,648 = 3,221,159,937, which overflowed --Therefore, saturation should set it at 0x7FFF_FFFF	--Testing the Underflow portion --Set the 1st set of 32 bits of rs1 to 0x7FFF_FFFF, the highest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x0001, 1 --Set the 1st set of 16 bits of rs3 to 0x8000, The lowest 16 bit number --That is: (-32,768*1) = -32768 - 2,147,483,647 = -2,147,516,415, which underflowed --Therefore, saturation should set it at 0x8000_0000	--Set the 0th set of 32 bits of rs1 to 3 --Set the 0th set of 32 bits of rs2 to 5 --Set the 0th set of 32 bits of rs3 to 5 --That is: (5*5) - 3 = 22, which is 0b1_0110

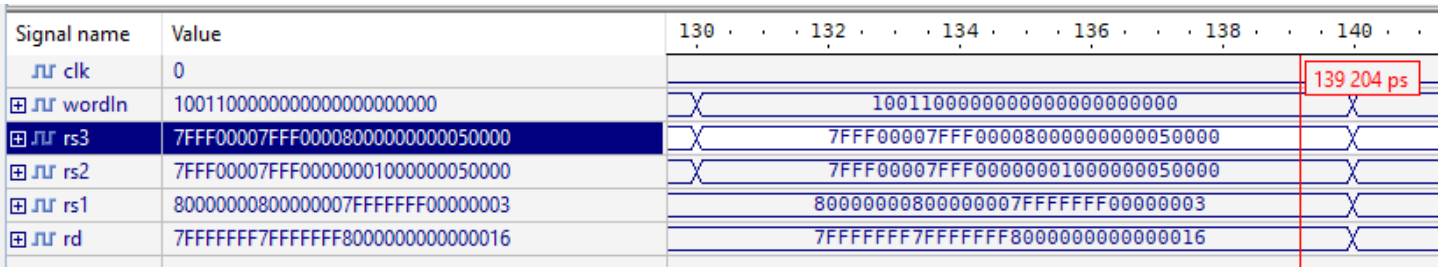
Simulation Result:



R4 Instruction: 011 - Integer Multiply-Subtraction High w/ Saturation
Function: Integer Multiply-Subtraction High w/ Saturation
WordIn: 1 0 010 00000 00000 00000 00000
Rs1: 0x80000000800000007FFFFFFF00000003
Rs2: 0x00007FFF00007FFF0000000100000005
Rs3: 0x00007FFF00007FFF0000800000000005
Explanation: Multiply the two high 16 bit values in R2 and R3 together. Then, subtract the 32 bit value in R1.

--3rd	--2nd	--1st	--0th
--Testing the Overflow operation --Testing the Overflow operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x7FFF, the highest 16 bit --Set the 1st set of 16 bits of rs3 to 0x7FFF, the highest 16 bit --That is: (32,767*32,767) = 1,073,676,289 - -2,147,483,648 = 3,221,159,937, which overflowed --Therefore, saturation should set it at 0x7FFF_FFFF	--Testing the Overflow operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000, the lowest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x7FFF, the highest 16 bit --Set the 1st set of 16 bits of rs3 to 0x7FFF, the highest 16 bit --That is: (32,767*32,767) = 1,073,676,289 - -2,147,483,648 = 3,221,159,937, which overflowed --Therefore, saturation should set it at 0x7FFF_FFFF	--Testing the Underflow portion --Set the 1st set of 32 bits of rs1 to 0x7FFF_FFFF, the highest 32 bit number --Set the 1st set of 16 bits of rs2 to 0x0001, 1 --Set the 1st set of 16 bits of rs3 to 0x8000, The lowest 16 bit number --That is: (-32,768*1) = -32768 - 2,147,483,647 = -2,147,516,415, which underflowed --Therefore, saturation should set it at 0x8000_0000	--Set the 0th set of 32 bits of rs1 to 3 --Set the 0th set of 32 bits of rs2 to 5 --Set the 0th set of 32 bits of rs3 to 5 --That is: (5*5) - 3 = 22, which is 0b1_0110

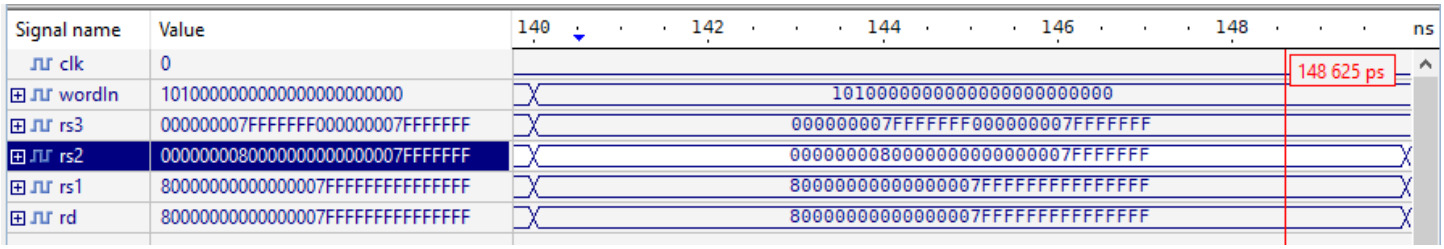
Simulation Result:



R4 Instruction: 100 - Long Multiply-Add Low w/ Saturation (Overflow and Underflow)
Function: Long Multiply-Add Low w/ Saturation
WordIn: 1 0 100 00000 00000 00000 00000
Rs1: 0x800000008000000007FFFFFFF00000003
Rs2: 0x7FFF00007FFF000000010000000050000
Rs3: 0x7FFF00007FFF000080000000000050000
Explanation: Multiply the two low 32 bit values in R2 and R3 together. Then, add the 64 bit value in R1.

--1st --Testing the Underflow portion --Set the 1st set of 32 bits of rs1 to 0x8000_0000_0000_0000, the lowest 64 bit number --Set the 1st set of 16 bits of rs2 to 0x8000_0000, The lowest 32 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --This should saturate at 0x8000_0000_0000_0000	--0th --Overflow Operation --Set the 0th set of 64 bits of rs1 to 0x7FFF_FFFF_FFFF_FFFF, the highest 64 bit number --Set the 0th set of 32 bits of rs2 to 0x7FFF_FFFF --Set the 0th set of 32 bits of rs3 to 0x7FFF_FFFF --This should cap out at 0x7FFF_FFFF_FFFF_FFFF

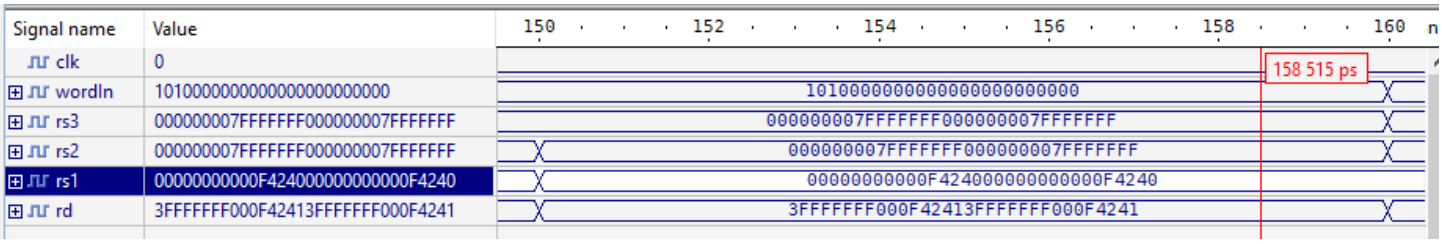
Simulation Result:



R4 Instruction: 100 - Long Multiply-Add Low w/ Saturation (Normal Operation)
Function: Long Multiply-Add Low w/ Saturation
WordIn: 1 0 100 00000 00000 00000 00000
Rs1: 0x800000008000000007FFFFFFF00000003
Rs2: 0x7FFF00007FFF000000010000000050000
Rs3: 0x7FFF00007FFF000080000000000050000
Explanation: Multiply the two Low 32 bit values in R2 and R3 together. Then, add the 64 bit value in R1.

--1st --Normal Operation --Set the 1st set of 64 bits of rs1 to 0x0000_0000_000F_4240, the lowest 64 bit number --Set the 1st set of 32 bits of rs2 to 0x7FFF_FFFF, The highest 32 bit number --Set the 1st set of 32 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --That is: (2147483647 * 2147483647) + 1000000 = 4,611,686,014,133,420,609 --Expect: 3FFFFFFF000F4241	--0th --Normal Operation --Set the 1st set of 64 bits of rs1 to 0x0000_0000_000F_4240, the lowest 64 bit number --Set the 1st set of 32 bits of rs2 to 0x7FFF_FFFF, The highest 32 bit number --Set the 1st set of 32 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --That is: (2147483647 * 2147483647) + 1000000 = 4,611,686,014,133,420,609 --Expect: 3FFFFFFF000F4241

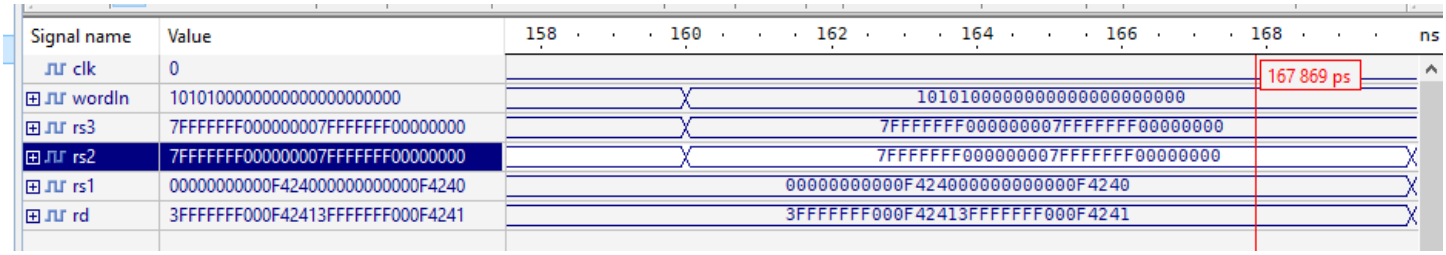
Simulation Result:



R4 Instruction: 101 - Long Multiply-Add High w/ Saturation (Normal Operation)
Function: Long Multiply-Add High w/ Saturation
WordIn: 1 0 101 00000 00000 00000 00000
Rs1: 0x00000000000F4240000000000000F4240
Rs2: 0x7FFFFFFF000000007FFFFFFF00000000
Rs3: 0x7FFFFFFF000000007FFFFFFF00000000
Explanation: Multiply the two high 32 bit values in R2 and R3 together. Then, add the 64 bit value in R1.

--1st --Normal Operation --Set the 1st set of 64 bits of rs1 to 0x0000_0000_000F_4240, the lowest 64 bit number --Set the 1st set of 32 bits of rs2 to 0x7FFF_FFFF, The highest 32 bit number --Set the 1st set of 32 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --That is: (2147483647 * 2147483647) + 1000000 = 4,611,686,014,133,420,609 --Expect: 3FFFFFFF000F4241	--0th --Normal Operation --Set the 1st set of 64 bits of rs1 to 0x0000_0000_000F_4240, the lowest 64 bit number --Set the 1st set of 32 bits of rs2 to 0x7FFF_FFFF, The highest 32 bit number --Set the 1st set of 32 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --That is: (2147483647 * 2147483647) + 1000000 = 4,611,686,014,133,420,609 --Expect: 3FFFFFFF000F4241

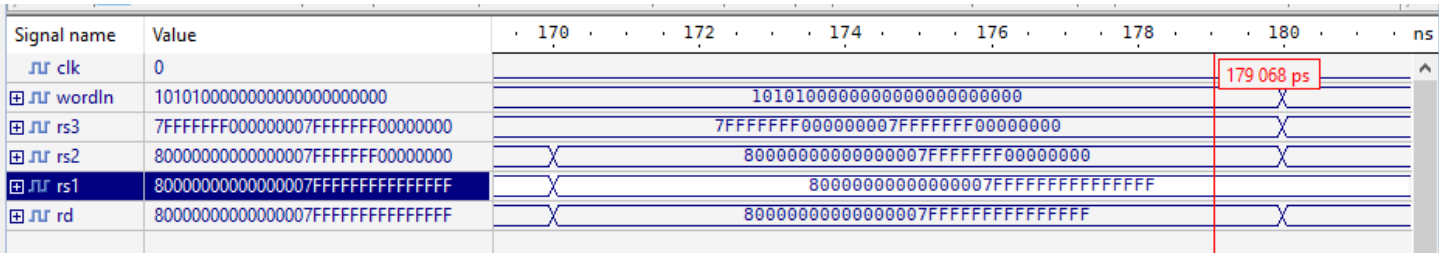
Simulation Result:



R4 Instruction: 101 - Long Multiply-Add High w/ Saturation (Overflow and Underflow)
Function: Long Multiply-Add High w/ Saturation
WordIn: 1 0 101 00000 00000 00000 00000
Rs1: 0x80000000000000007FFFFFFFFFFFFFFF
Rs2: 0x80000000000000007FFFFFF00000000
Rs3: 0x7FFFFFF000000007FFFFFF00000000
Explanation: Multiply the two high 32 bit values in R2 and R3 together. Then, add the 64 bit value in R1.

--1st --Testing the Underflow portion --Set the 1st set of 32 bits of rs1 to 0x8000_0000_0000_0000, the lowest 64 bit number --Set the 1st set of 16 bits of rs2 to 0x8000_0000, The lowest 32 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --This should saturate at 0x8000_0000_0000_0000	--0th --Overflow Operation --Set the 0th set of 64 bits of rs1 to 0x7FFF_FFFF_FFFF_FFFF, the highest 64 bit number --Set the 0th set of 32 bits of rs2 to 0x7FFF_FFFF --Set the 0th set of 32 bits of rs3 to 0x7FFF_FFFF --This should cap out at 0x7FFF_FFFF_FFFF_FFFF

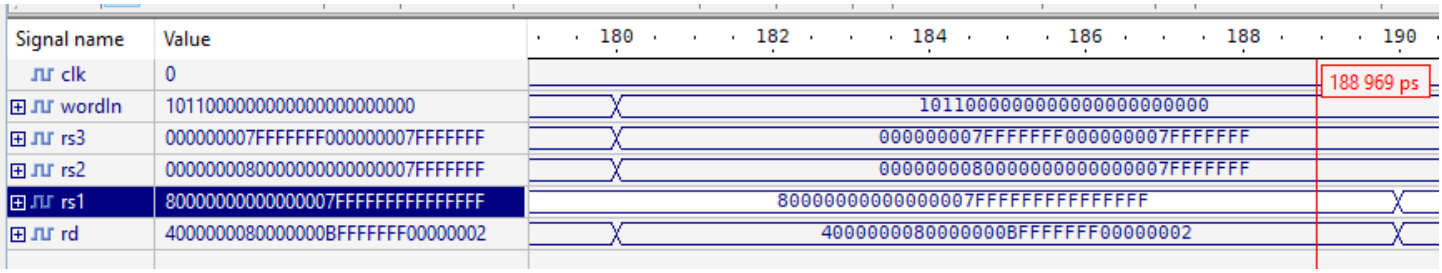
Simulation Result:



R4 Instruction: 110 - Long Multiply-Subtraction Low w/ Saturation (Normal Operation)
Function: Long Multiply-Subtraction Low w/ Saturation
WordIn: 10110000000000000000000000
Rs1: 0x80000000000000007FFFFFFFFFFFFFFF
Rs2: 0x00000000800000000000000007FFFFFF
Rs3: 0x000000007FFFFFFF000000007FFFFFFF
Explanation: Multiply the two low 32 bit values in R2 and R3 together. Then, subtract the 64 bit value in R1.

--1st --Testing the Normal operation --Set the 1st set of 32 bits of rs1 to 0x8000_0000_0000_0000, the lowest 64 bit number --Set the 1st set of 16 bits of rs2 to 0x8000_0000, The lowest 32 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --This should result in to 0xBFFF FFFF 0000 0002	--0th --Normal Operation --Set the 0th set of 64 bits of rs1 to 0x7FFF_FFFF_FFFF_FFFF, the highest 64 bit number --Set the 0th set of 32 bits of rs2 to 0x7FFF_FFFF --Set the 0th set of 32 bits of rs3 to 0x7FFF_FFFF --This should result in 0x4000 0000 8000 0000

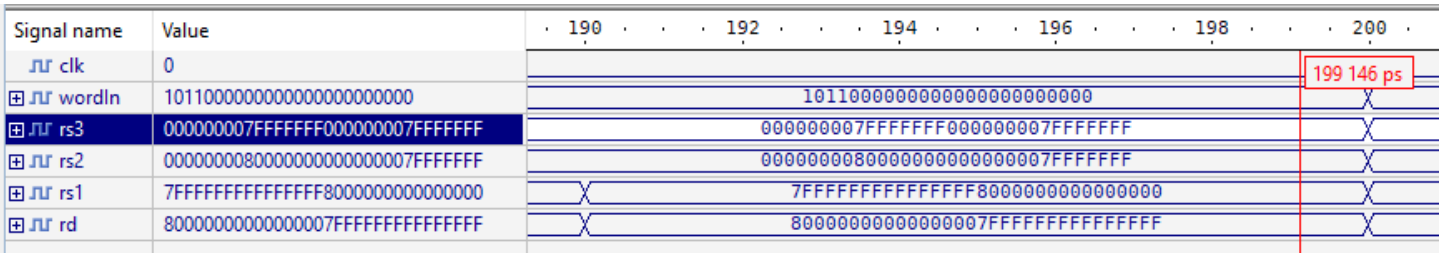
Simulation Result:



R4 Instruction: 110 - Long Multiply-Subtraction Low w/ Saturation (Overflow and Underflow)
Function: Long Multiply-Subtraction Low w/ Saturation
WordIn: 10110000000000000000000000
Rs1: 0x7FFFFFFFFFFFFFFF8000000000000000
Rs2: 0x000000008000000000000007FFFFFFF
Rs3: 0x000000007FFFFFFF000000007FFFFFFF
Explanation: Multiply the two low 32 bit values in R2 and R3 together. Then, subtract the 64 bit value in R1.

--1st --Testing the Underflow Operation --Set the 1st set of 32 bits of rs1 to 0x7FFF_FFFF_FFFF_FFFF, the highest 64 bit number --Set the 1st set of 16 bits of rs2 to 0x8000_0000, The lowest 32 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --This should saturate to 0x8000 0000 0000 000	--0th --Overflow Operation --Set the 0th set of 64 bits of rs1 to 0x8000_0000_0000_0000, the lowest 64 bit number --Set the 0th set of 32 bits of rs2 to 0x7FFF_FFFF --Set the 0th set of 32 bits of rs3 to 0x7FFF_FFFF --This should result in 0x7FFF_FFFF_FFFF_FFFF

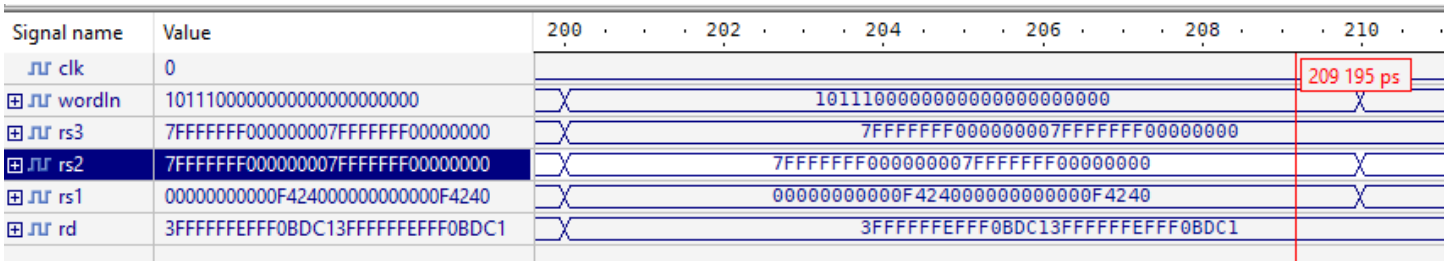
Simulation Result:



R4 Instruction: 111 - Long Multiply-Subtraction High w/ Saturation (Normal Operation)
Function: Long Multiply-Subtraction High w/ Saturation
WordIn: 10111000000000000000000000
Rs1: 0x00000000000F424000000000000F4240
Rs2: 0x7FFFFFFF000000007FFFFFFF00000000
Rs3: 0x7FFFFFFF000000007FFFFFFF00000000
Explanation: Multiply the two high 32 bit values in R2 and R3 together. Then, subtract the 64 bit value in R1.

--1st --Normal Operation --Set the 1st set of 64 bits of rs1 to 0x0000_0000_000F_4240, the lowest 64 bit number --Set the 1st set of 32 bits of rs2 to 0x7FFF_FFFF, The highest 32 bit number --Set the 1st set of 32 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --That is: (2147483647 * 2147483647) - 1000000 = 4,611,686,014,133,420,609 --Expect: 3FFFFFFF000F4241	--0th --Normal Operation --Set the 1st set of 64 bits of rs1 to 0x0000_0000_000F_4240, the lowest 64 bit number --Set the 1st set of 32 bits of rs2 to 0x7FFF_FFFF, The highest 32 bit number --Set the 1st set of 32 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --That is: (2147483647 * 2147483647) + 1000000 = 4,611,686,014,131,420,609 --Expect: 3FFFFFFEFFF0BDC1

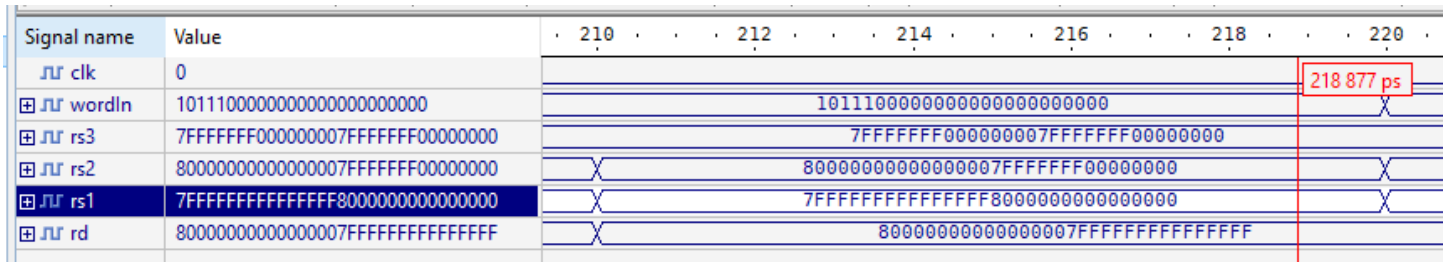
Simulation Result:



R4 Instruction: 111 - Long Multiply-Subtraction High w/ Saturation (Overflow and Underflow)
Function: Long Multiply-Subtraction High w/ Saturation
WordIn: 10111000000000000000000000
Rs1: 0x7FFFFFFFFFFFFFFF8000000000000000
Rs2: 0x80000000000000007FFFFFFF00000000
Rs3: 0x7FFFFFFF000000007FFFFFFF00000000
Explanation: Multiply the two high 32 bit values in R2 and R3 together. Then, subtract the 64 bit value in R1.

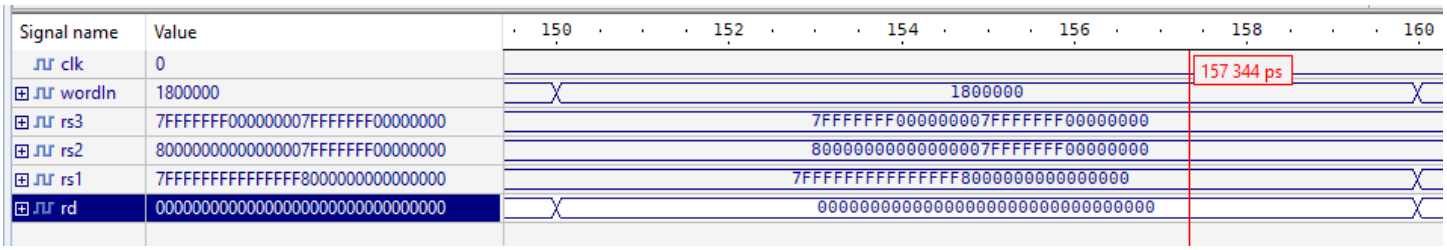
--1st --Testing the Underflow Operation --Set the 1st set of 32 bits of rs1 to 0x7FFF_FFFF_FFFF_FFFF, the highest 64 bit number --Set the 1st set of 16 bits of rs2 to 0x8000_0000, The lowest 32 bit number --Set the 1st set of 16 bits of rs3 to 0x7FFF_FFFF, The highest 32 bit number --This should saturate to 0x8000 0000 0000 0000	--0th --Overflow Operation --Set the 0th set of 64 bits of rs1 to 0x8000_0000_0000_0000, the lowest 64 bit number --Set the 0th set of 32 bits of rs2 to 0x7FFF_FFFF --Set the 0th set of 32 bits of rs3 to 0x7FFF_FFFF --This should result in 0x7FFF_FFFF_FFFF_FFFF

Simulation Result:



R3 Instructions

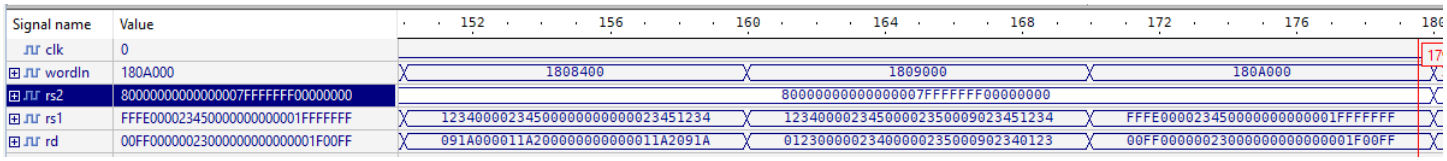
R3 Instruction: 0000 - NOP
Function: NOP
Word In: 1 1 0000 0000 00000 00000 00000
Explanation: The ALU in this case will output only all 0s. This is to verify that the NOP is actually being selected. However, this function will really be implemented through the register file, and not the ALU.
Simulation Result:



R3 Instruction: 0001 - SHRHI (Test 1)
Function: Shift right halfword immediate. The immediate value is provided in the rs2 field (13 downto 10)
Word In: 1 1 0000 0001 00000 00000 00000
Rs1: 0x1234 0000 2345 0000 0000 0000 2345 1234
Rs2 Field: 0b0001
Explanation: Rotate each variable by 1.
Expect: 0x091A 0000 11A2 0000 0000 0000 11A2 091A

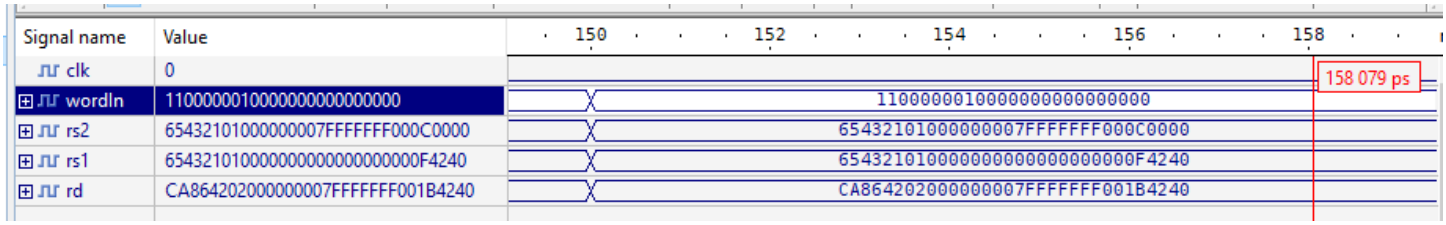
R3 Instruction: 0001 - SHRHI (Test 2)
Function: Shift right halfword immediate. The immediate value is provided in the rs2 field (13 downto 10)
Word In: 1 1 0000 0001 00000 00000 00000
Rs1: 0x1234 0000 2345 0000 0000 0000 2345 1234
Rs2 Field: 0b0100
Explanation: Shift each index right by 4
Expect: 0x0123 0000 0234 0000 0000 0000 0234 0123

R3 Instruction: 0001 - SHRHI (Test 3)
Function: Shift right halfword immediate. The immediate value is provided in the rs2 field (13 downto 10)
Word In: 1 1 0000 0001 00000 00000 00000
Rs1: 0xFFFE 0000 2345 0000 0000 0000 1FFF FFFF
Rs2 Field: 0b1000
Explanation: Shift each index right by 8
Expect: 0x00FF 0000 0023 0000 0000 0000 001F 00FF
Simulation Result:

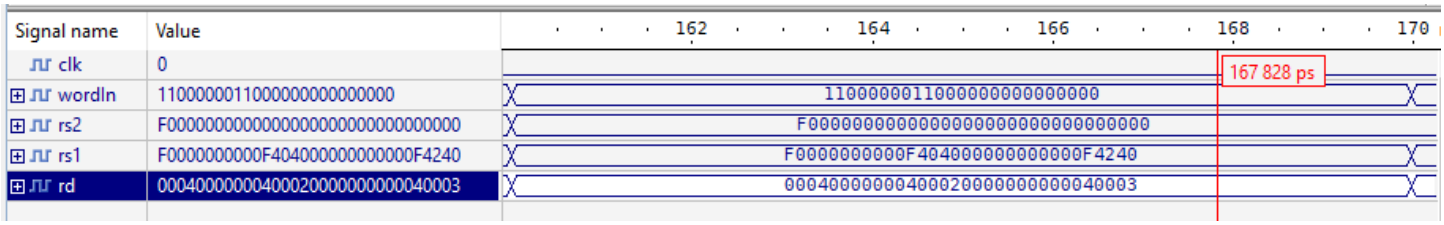


R3 Instruction: 0010 - AU
Function: Add Unsigned
Word In: 1 1 0000 0010 00000 00000 00000
Rs1: 0x65432101000000000000000000F4240

Rs2: 0x65432101000000007FFFFFFF000C0000
Explanation: Add every 32 bit values: expected value is 0xCA864202 00000000 7FFFFFFF 001B4240
Simulation Result:



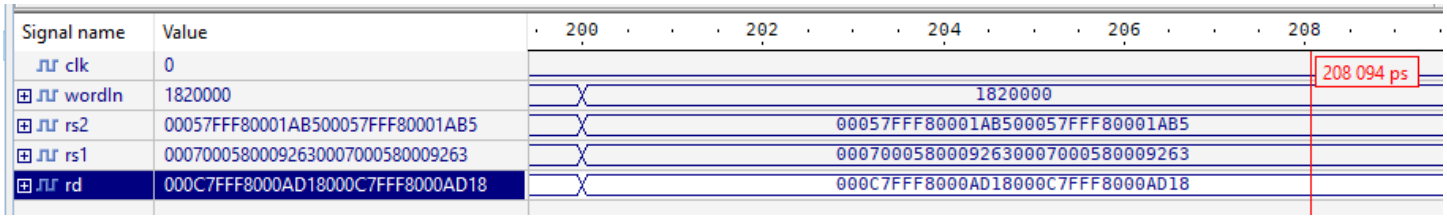
R3 Instruction: 0011 - CNT1H
Function: Count the number of 1s in each halfword
Word In: 1 1 0000 0011 00000 00000 00000
Rs1: 0xF0000000000F4040000000000000F4240
Explanation: Add up the 1s in each halfword. Expected value is 0004 0000 0004 0002 0000 0000 0004 0003
Simulation Result:



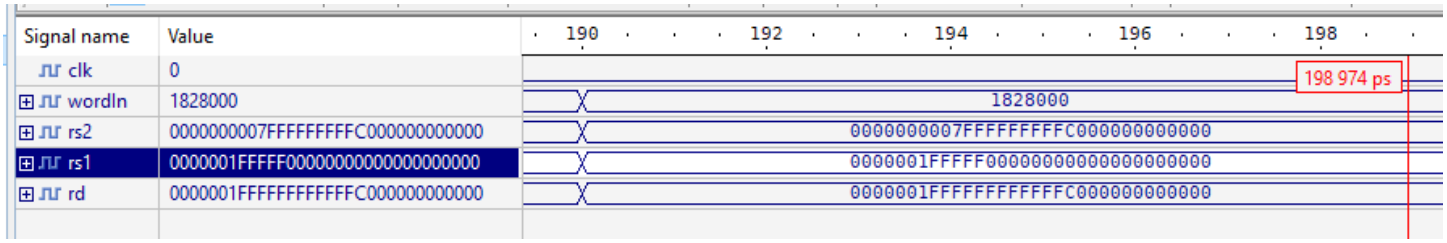
R3 Instruction: 0100 - AHS
Function: Add halfword saturated
Word In: 1 1 0000 0100 00000 00000 00000
Explanation: Add each corresponding halfword together. Employ saturation.

--3rd --Normal Operation --Set 3rd set of rs2 to 0x0005, which is dec 5 --Set 3rd set of rs1 to 0x0007, which is dec 7 --Answer should be placed in last 16 bits of rd --Answer is 12, which is 000C	--2nd --Overflow Condition --Set 2nd set of rs2 to 0x7FFF, which is most postive number --Set 2nd set of rs1 to 0x0005, which is 5 --Answer should be placed in last 16 bits of rd --Answer should overflow - you can't get anymore positive --Saturation will kick in, answer is 0x7FFF	--1st --Underflow Operation --Set last set of rs2 to 0x8000, which is most negative number --Set last set of rs1 to 0x8000, which is most negative number --Answer should underflow - 16 bits can't get more negative than 8000 --Answer should 0x8000	--0th --Normal Operation --Set zeroth set of rs2 to 0x1AB5, which is 6,837 --Set zeroth set of rs1 to 0x9263, which is 37475 --Answer should be placed in zeroth 16 bits of rd --Answer is 44,312, answer is 0xAD18

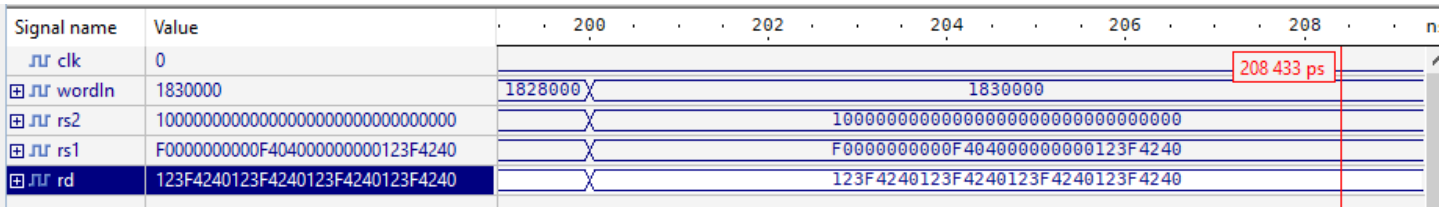
Simulation Result:



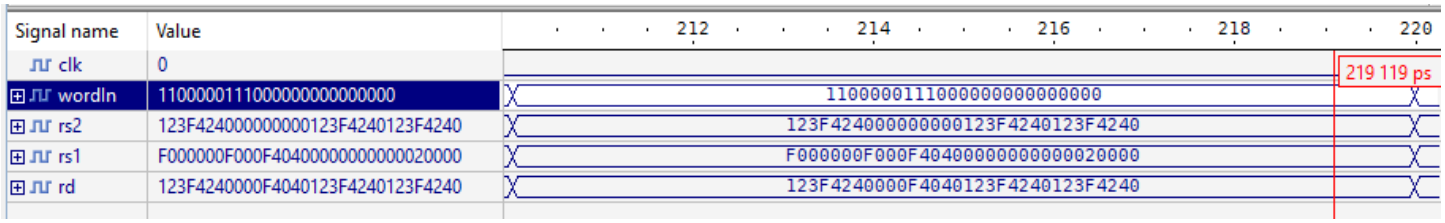
R3 Instruction: 0101 - OR
Function: Bitwise Logical Or
Word In: 1 1 0000 0101 00000 00000 00000
Explanation: We expect to see only the 1s from R2 and R1 to propagate down to Rd
Simulation Result:



R3 Instruction: 0110 - BCW
Function: Broadcast the 0th 32 bits of Rs1 to each of the other 3 indices.
Word In: 1 1 0000 0110 00000 00000 00000
Rs1: 0xF0000000000F4040000000000123F4240
Explanation: Copy 0x123F 4240 to everything else. Expect 0x123F 4240 123F 4240 123F 4240 123F 4240
Simulation Result:

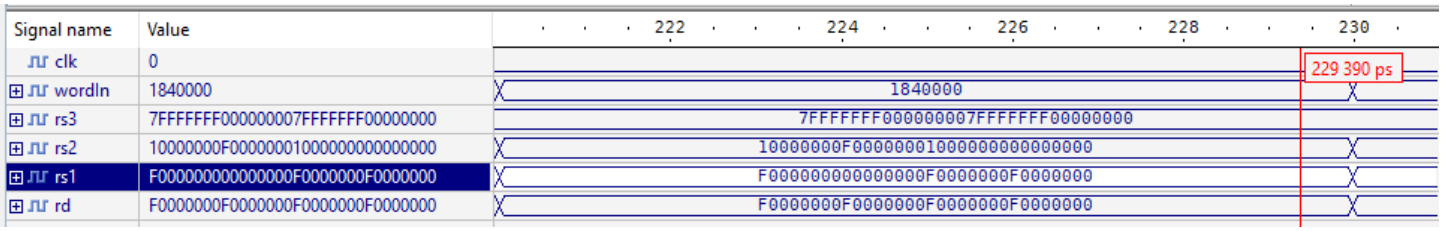


R3 Instruction: 0111 - MAXWS
Function: For each 32 bit word slot, select the maximum signed value to be placed in Rd.
Word In: 1 1 0000 0111 00000 00000 00000
Rs1: 0xF000000F000F40400000000000020000
Rs2: 0x123F4240000000000123F4240123F4240
Explanation: Expect to see the highest for each slot. Expect: 123F 4240 000F 4040 123F 4240 123F 4240
Simulation Result:



R3 Instruction: 1000 - MINWS
Function: For each 32 bit word slot, select the most minimum signed value to be placed in Rd

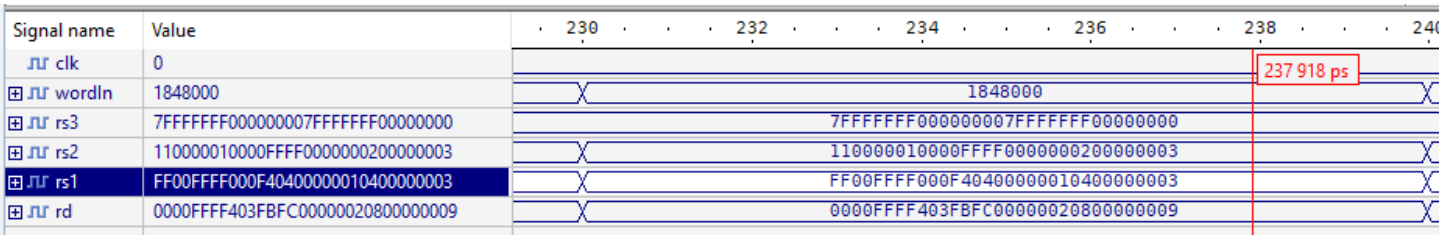
Word In: 1 1 0000 1000 00000 00000 00000
Rs1: 0xF000000000000000F0000000F0000000
Rs2: 0x10000000F0000000100000000000000
Explanation: F would represent a negative number, while 1 represents a positive. Thus, we should only see the F functions in the register Rd.
Simulation Result:



R3 Instruction: 1001 - MLHU
Function: Multiply Low Unsigned
Word In: 1 1 0000 1001 00000 00000 00000
Rs1: 0xFF00FFFF000F404000000010400000003
Rs2: 0x1100000010000FFFF0000000200000003
Explanation:

--3rd --FFFF * 0001 = FFFF [Hex] --65535 * 1 = 65535 [Dec]	--2nd --4040 * FFFF = 1043F3BBFC0 [Hex] --16448 * 65535 = 1,077,919,680 [Dec]	--1st --104 * 2 = 208 [Hex] --260 * 2 = 520 [Dec]	--0th --3*3 = 9

Simulation Result:

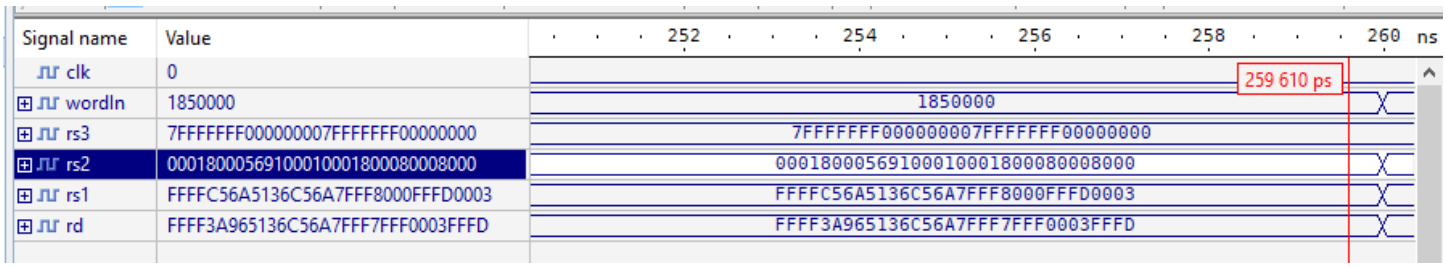


R3 Instruction: 1010 - MLHSS
Function: Multiply by Sign Saturated
Word In: 1 1 0000 1010 00000 00000 00000
Explanation:

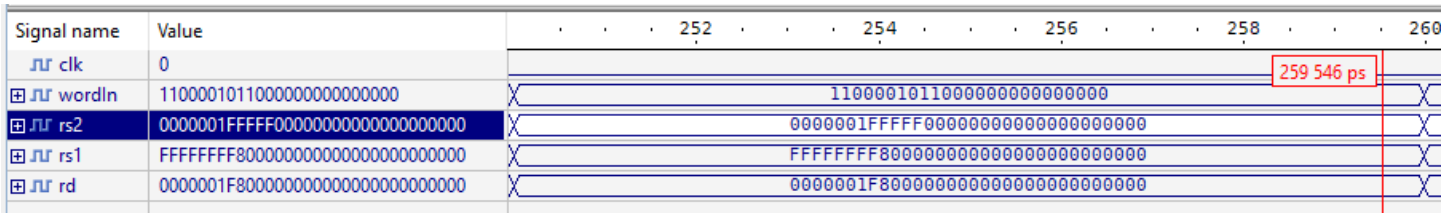
--7th --FFFF * 0001 = FFFF[Hex]	--6th -- C56A * 8000 = 3A96 [Hex]	--5th -- 5136 * 5691 = 5136 [Hex]	--4th -- C56A * 0001 = C56A [Hex]	--3rd --7FFF * 0001 = 7FFF [Hex]	--2nd --saturate	--1st -- -3 * -1 = 3	--0th -- 3*-1 = -3

-- -1 * 1 = -1 [Dec]	-- -14998 * -1 = 14998 [Dec]	-- 5430 * 1 = 5430 [Dec]	-- -14998 * 1 = -14998 [Dec]	-- 32767 * 1 = 32767 [Dec]	-- 8000 * 8000 = 7FFF [Hex] -- -32768 * -1 = 32767 [Dec]	-- FFFD * 8000 = 3 [Hex]	-- 3*8000 = FFFD [hex]

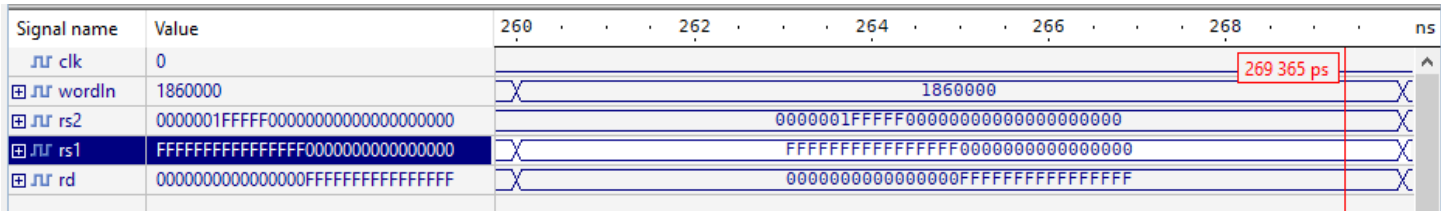
Simulation Result:



R3 Instruction: 1011 - AND
Function: Bitwise And Rs1 and Rs2
Word In: 1 1 0000 1011 00000 00000 00000
Explanation: We load some 1s into both Rs1 and Rs2. The bits which overlap should be shown in Rd.
Simulation Result:

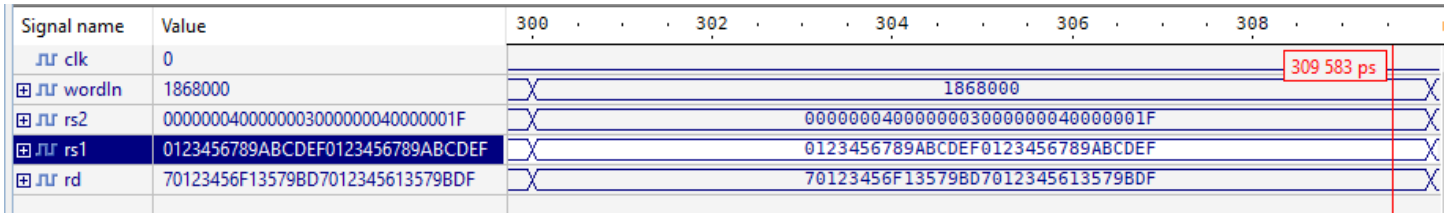


R3 Instruction: 1100 - INVB
Function: Bitwise invert Rs1
Word In: 1 1 0000 1100 00000 00000 00000
Explanation: We load in 1s in some part of Rs1. We expect to see the entire register inverted
Simulation Result:



R3 Instruction: 1101 ROTW
Function: Rotate each 32 bit word in Rs1 by the corresponding value in Rs2.
Word In: 1 1 0000 1101 00000 00000 00000
Rs1: 0x0123 4567 89AB CDEF 0123 4567 89AB CDEF
Rs2: 0x0000 0004 0000 0003 0000 0004 0000 001F
Explanation: We're rotating the words by 4, 3, 4, and 31, respectively.

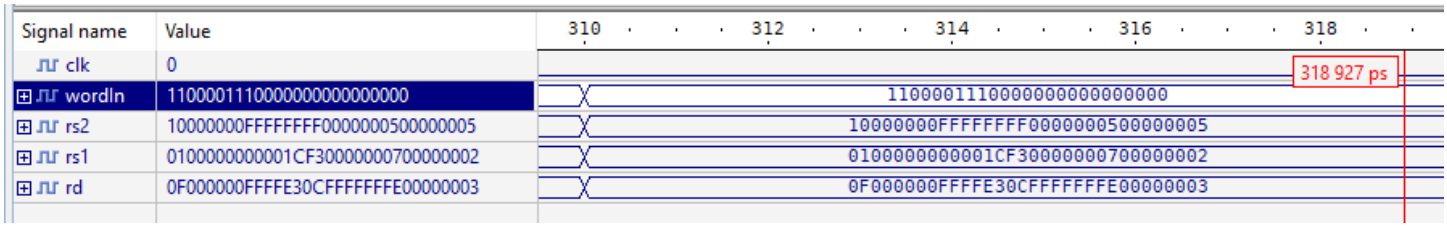
Expecting: 0x 7012 3456 F135 79BD 7012 3456 1357 9BDF
Simulation Result:



R3 Instruction: 1110 SFWU
Function: Subtract from Word Unsigned
Word In: 1 1 0000 1110 00000 00000 00000
Explanation:

--1st --Testing edge case for SFWU --Set last set of rs2 to 0x1000_0000, which is dec 16,777,216 --Set last set of rs1 to 0x0100_0000, which is dec 268,435,456 --Answer should be placed in last 32 bits of rd. --Answer is 0x0F00_0000, or 251,658,240	--3rd --Set last set of rs2 to 0xFFFF_FFFF, which is dec 4,294,967,295 --Set last set of rs1 to 0x0000_1CF3, which is dec 7,411 --Answer is negative when done in decimal. --Answer is should underflow to FFFF_E30C	--1st --Testing edge case for SFWU --Set last set of rs2 to 0x0000_0005, which is dec 5 --Set last set of rs1 to 0x0000_0007, which is dec 7 --Answer is negative when done in decimal. --Answer is should underflow to FFFF_FFFE	--0th --Set 0th set of rs1 to 0x0100_0000, which is dec 16,777,216 --Set 0th set of rs2 to 0x1000_0000, which is dec 268,435,456 --Answer should be placed in last 32 bits of rd --Answer is 0x0F00_0000, or 251,658,240

Simulation Result:



R3 Instruction: 1111 - SFHS
Function: Subtract from Halfword Saturated
Word In: 1 1 0000 1111 00000 00000 00000
Rs1: 0x0007800080000001000780008000001
Rs2: 0x00057FFF8000800000057FFF80008000
Explanation: These 4 test cases are repeated in indices 7, 6, 5, 4

--3rd --Normal Operation --Set 3rd set of rs2 to 0x0005, which is dec 5 --Set 3rd set of rs1 to 0x0007, which is dec 7 --Answer should be placed in last 16 bits of rd --Answer is -2, which is FFFE	--2nd --Overflow Condition --Set 2nd set of rs2 to 0x7FFF, which is most postive number --Set 2nd set of rs1 to 0x8000, which is most negative number --Answer should be placed in last 16 bits of rd --Answer should overflow - you can't get anymore positive --Saturation will kick in, answer is 0x7FFF	--1st --Regular Operation --Set last set of rs2 to 0x8000, which is most negative number --Set last set of rs1 to 0x8000, which is most negative number --Answer should be placed in 1st 16 bits of rd --Answer should 0x0000	--0th --Underflow Condition --Set zeroth set of rs2 to 0x8000, which is most negative number --Set zeroth set of rs1 to 0x0001, which is 1 --Answer should be placed in zeroth 16 bits of rd --Answer should underflow - you can't get any more negative --Saturation will kick in, answer is 0x8000

Simulation Result: