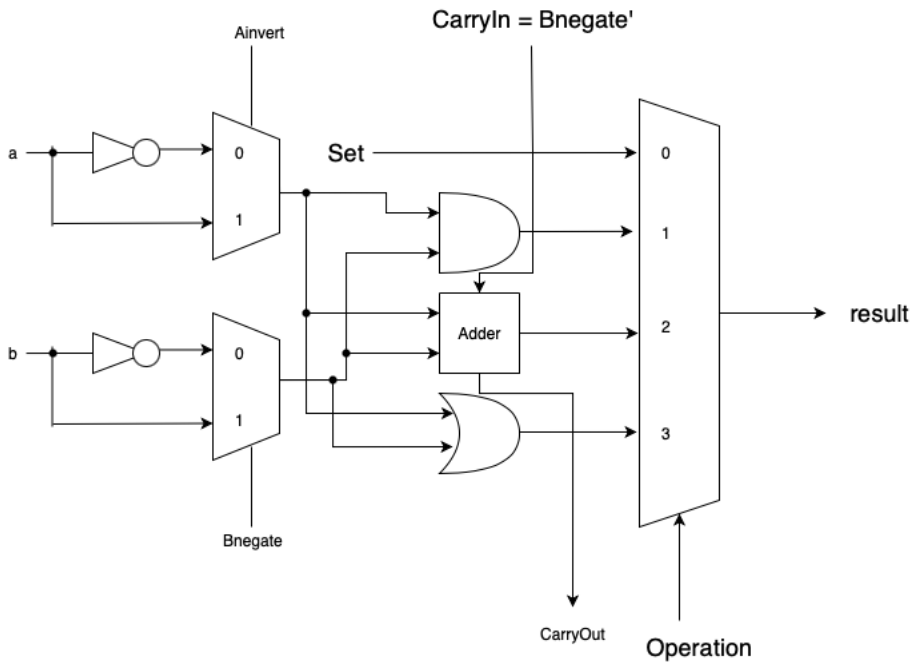
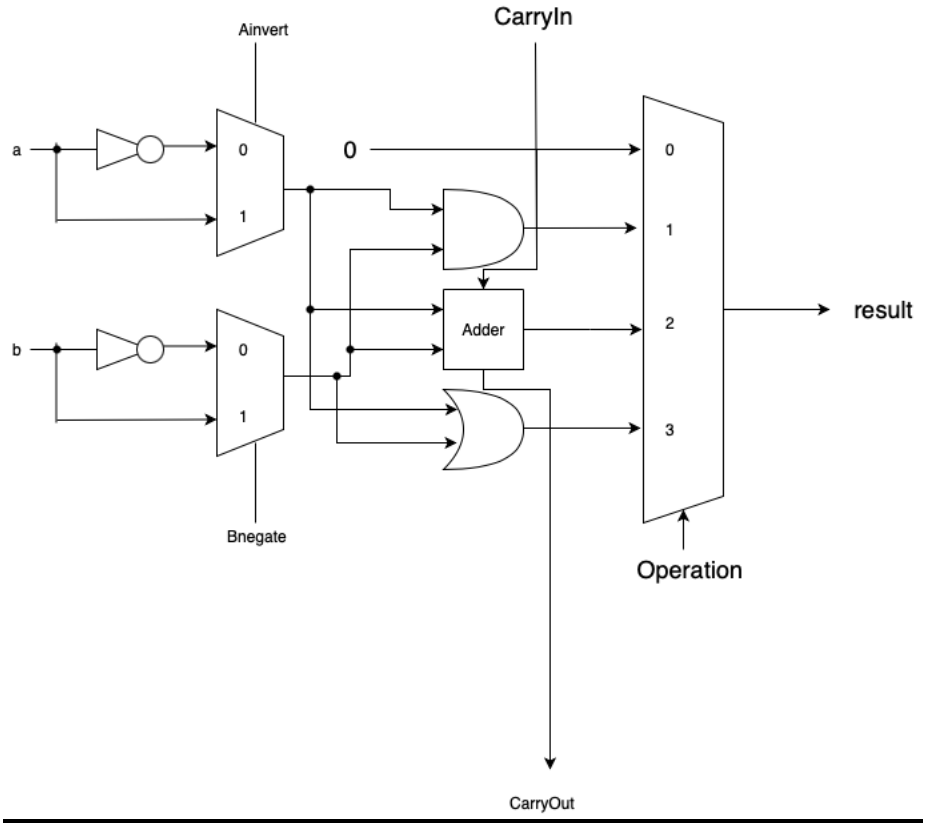
109062320

1. (15 points) In class we have described a 64-bit ALU whose control input ALUop is composed of 1-bit Ainvert, 1-bit Bnegate, and 2-bit Operation from left to right. Re-design the ALU to meet the following new specification. You need to draw the circuit diagrams of each 1-bit ALU and the 64-bit ALU, similar to those shown in the lecture notes.

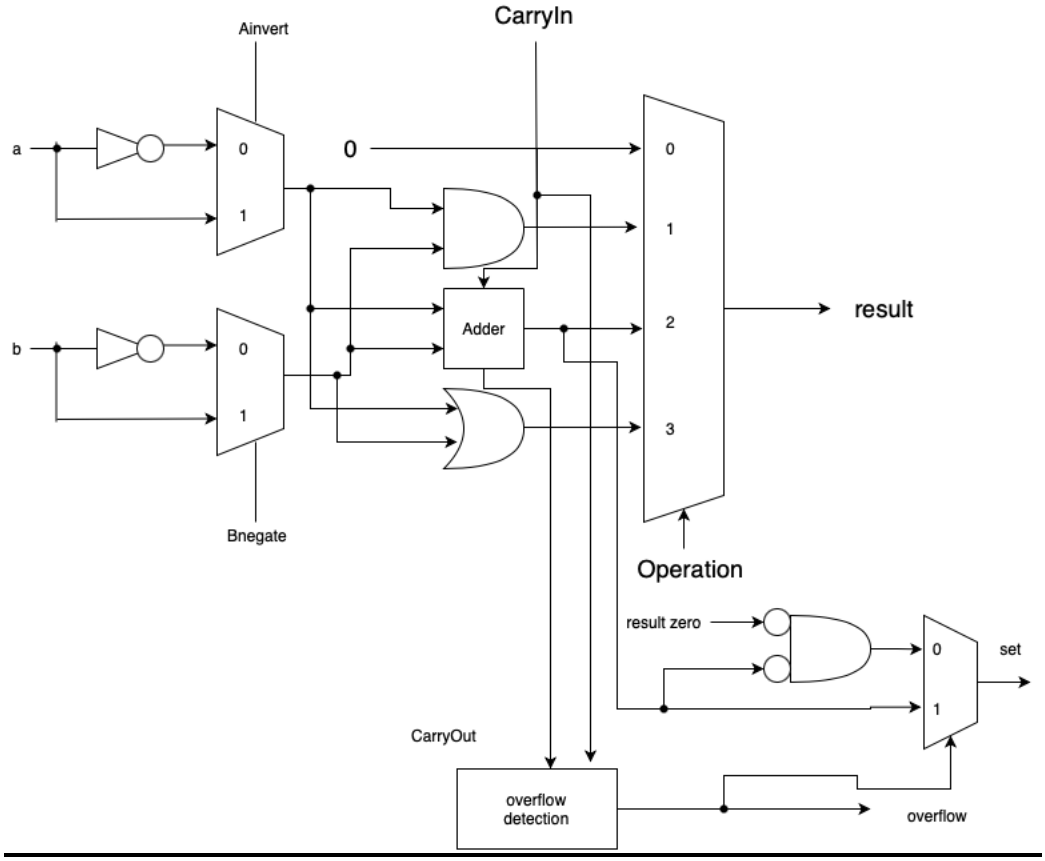
ALU Control (ALUop)	Function
1101	And
1111	Or
1110	Add
1010	Substract
1000	Set greater than
0011	Nand



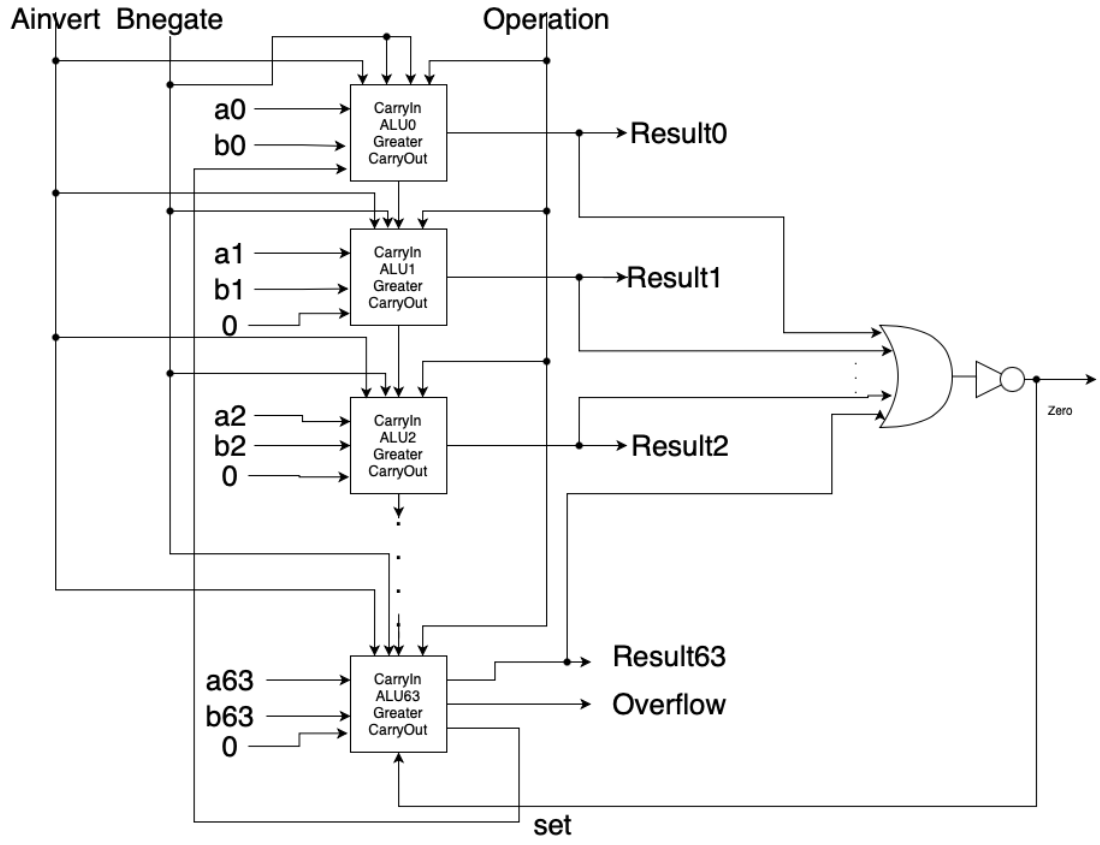
Bit0 in ALU



Bit1~62 ALU



Bit63 ALU



64-bit ALU

Note: 1. the above adder is full adder 2. There is an xor in the overflow detection 3.we have to check the result of A-B isn't equal to 0

2. (36 points) Consider two unsigned binary numbers: M = 1010 and N = 0101.

(a) (8 points) Write down each step of M × N according to version 1 of the multiply algorithm.

iteration	multiplier	multiplicand	product
0(initial)	0101	0000 1010	0000 0000
1	010**1**	0000 1010	0000 1010
2	001**0**	0001 0100	0000 1010
3	000**1**	0010 1000	0011 0010
4	000**0**	0101 0000	0011 0010

(b) (8 points) Write down each step of M × N according to version 2 of the multiply algorithm.

iteration	multiplicand	Left product	Right product	description
0	1010	0000	010**1**	
1	1010	1010	0101	Add mul
1	1010	0101	001**0**	Shift right
2	1010	0010	100**1**	Shift right
3	1010	1100	100**1**	Add mul
3	1010	0110	010**0**	Shift right
4	1010	0011	0010	Shift right

(c) (10 points) Write down each step of M ÷ N according to version 1 of the divide algorithm. (1010 / 0101)

iteration	quotient	divisor	remainder
0(initial)	0000	0101 0000	0000 1010
1	0000	0101 0000	**1**011 1010
1	0000	0101 0000	0000 1010
1	0000	0010 1000	0000 1010
2	0000	0010 1000	**1**110 0010
2	0000	0010 1000	0000 1010
2	0000	0001 0100	0000 1010
3	0000	0001 0100	**1**111 0110
3	0000	0001 0100	0000 1010
3	0000	0000 1010	0000 1010
4	0000	0000 1010	**0**000 0000
4	000**1**	0000 1010	0000 0000
4	0001	0000 0101	0000 0000
5	0001	0000 0101	**1**111 1011
5	0010	0000 0010	0000 0000

(d) (10 points) Write down each step of M ÷ N according to version 2 of the divide algorithm

step	Remainder (rem | quot)	div
0	0000 1010	0101
1.1	0001 0100	
1.2	**1**100 0100	
1.3b	0010 1000	
2.2	**1**101 1000	
2.3b	0101 0000	
3.2	**0**000 0000	
3.3a	0000 0001	
4.2	**1**011 0001	
4.3a	0000 0010	
Done	0000 0010	

3. (20 points) Consider two decimal numbers: X = 785.3125 and Y = −13.125.

(a) (10 points) Write down X and Y in the IEEE 754 single precision format. You must detail how you get the answers, or you will receive 0 point.

X:

785 => 0011 0001 0001 (continue to divide 2)

0.3125 => 0. 0101 (0.6250 , 1.25 -> 0.25 , 0.5 , 1.0 -> 0

continue to multiply 2 and when result exceed 1, minus 1)

785.3125 => 0011 0001 0001. 0101 = 1.100 0100 0101 0100 0 x 2^9

Sign bit = 0

exponent => 0000 1001 + 0111 1111 = 1000 1000

Fraction => 100 0100 0101 0100 0000 0000

sign	exponent	fragment
0	1000 1000	100 0100 0101 0100 0000 0000

Y:

13 = 0000 1101 (continue to divide 2)

0.125 = 1/8 = 2^(-3) => 0.001

13.125 => 1101.0010 = 1.101 0010 x 2^3

Sign bit = 1

Exponent => 0000 0011 + 0111 1111 = 1000 0010

Fragment => 101 0010 0000 0000 0000 0000

sign	exponent	fragment

1	**1000 0010**	**101 0010 0000 0000 0000 0000**

(b) (5 points) Assuming X and Y are given in the IEEE 754 single precision format, show all the steps to perform X + Y and write the result in the IEEE 754 single precision format.

X: 0 ,1000 1000 ,100 0100 0101 0100 0000

=> + 1.100 0100 0101 0100 x 2^(136 – 127)

Y: 1 ,1000 0010 ,101 0010 0000 0000 0000

=> - 1.101 0010 x 2^(130 – 127)

X + Y = 1.100 0100 0101 0100 x 2^9 – 1.101 001 x 2^3

= 1.100 0100 0101 0100 x 2^9 – 0.000 0011 0100 1000 x 2^9

= 1.100 0001 0000 1100 x 2^9

(add significands, and the result shows that no underflow happen)

Sign bit = 0

Exponent => 0000 1001 + 0111 1111 = 1000 1000

Fragment => 100 0001 0000 1100 0000

sign	exponent	fragment
0	**1000 1000**	**100 0001 0000 1100 0000 0000**

(c) (5 points) Assuming X and Y are given in the IEEE 754 single precision format, show all the steps to perform X × Y and write the result in the IEEE 754 single precision format.

X: 0 ,1000 1000 ,100 0100 0101 0100 0000

=> + 1.100 0100 0101 0100 x 2^(136 – 127)

Y: 1 ,1000 0010 ,101 0010 0000 0000 0000

=> - 1.101 0010 x 2^(130 – 127)

a. add exponents : 9 + 3 = 12, biased: 12 + 127 = 139

b. multiply significands

1.100 0100 0101 01 x 1.101 0010 0000 00

--> 13 + 13 = 26 (小數點後總共 26 位)

multiplier	multiplicand	product
1101 001**0 0000 00**	0000 0000 0000 0011 0001 0001 0101	0000 0000 0000 0000 0000 0000 0000
0000 0001 1010 0**1**	0000 0001 1000 1000 1010 1**000 0000**	0000 0001 1000 1000 1010 1000 0000
0000 0000 1101 **00**	0000 0011 0001 0001 0101 0000 000**0**	0000 0001 1000 1000 1010 1000 0000
0000 0000 0011 0**1**	0000 1100 0100 0101 0100 0000 00**00**	0000 1101 1100 1101 1110 1000 0000
0000 0000 0001 1**0**	0001 1000 1000 1010 1000 0000 000**0**	0000 1101 1100 1101 1110 1000 0000
0000 0000 0000 1**1**	0011 0001 0001 0101 0000 0000 000**0**	0011 1110 1110 0010 1110 1000 0000
0000 0000 0000 0**1**	0110 0010 0010 1010 0000 0000 000**0**	1010 0001 0000 1100 1110 1000 0000

c. **Normalize result & check for over/underflow**

Result = 10.1000 0100 0011 0011 1010 0000 00 * 2^12

= 1.0100 0010 0001 1001 1101 *2^13

d. **Round and renormalize if necessary --> no change**

e. **Determine sign : + * - = -**

Sign bit : 1

Exponent => 0000 1101 + 0111 1111 = 1000 1100

Fragment => 0100 0010 0001 1001 1101 000

sign	exponent	fragment
1	1000 1100	0100 0010 0001 1001 1101 000

4. (9 points) Let W be the hexadecimal pattern 0xFF8E0E13. You must detail how you get the answers, or you will receive 0 point.

(a) (3 points) What decimal number does W represent if it is a two's complement integer?

Convert to binary--> 1111 1111 1000 1110 0000 1110 0001 0011

Because the sign bit is 1 --> negative!

Two's complement -->0000 0000 0111 0001 1111 0001 1110 1101

Convert to hexadecimal -> 0x0071F1ED

Convert to decimal -> 13 + 14*16 + 16^2 + 15*16^3 + 16^4 + 7*16^5

= 7467501

The answer is -7467501

(b) (3 points) What decimal number does W represent if it is an IEEE 754 floating-point number?

sign	exponent	fragment
1	11111111	0001 1100 0001 1100 0010 011

Exponent in decimal = 255 , fraction != 0 ==> the object is not a number(NaN)

(c) (3 points) Is it possible for W to be a RISC-V instruction? If yes, what is the corresponding assembly instruction? If not, why?

1111 1111 1000 1110 0000 1110 0 0010011

Firstly, we observe the last 7 bits since opcode of all of instructions is situated at 0~6.

I-type	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	lwu	0000011	110	n.a.
	addi	0010011	000	n.a.
	Slli	0010011	001	000000
	xori	0010011	100	n.a.
	srli	0010011	101	000000
	srai	0010011	101	010000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.

Image4-c-1: the instructions that the opcode is 0010011

By the above image, we can see that if W is a RISC-V instruction, then the instruction belongs to I-type

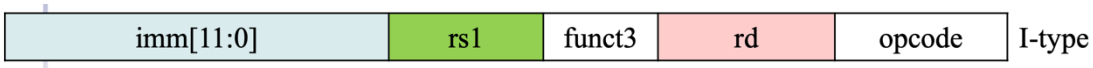


Image4-c-2: I-type instruction

1111 1111 1000 11100 000 11100 0010011

By Image4-c-2, we can recognize which are the bits belong to.

What's more, we can also ensure that it's possible for W to be a RISC-V instruction.

By func3 and opcode, we know that the instruction is "addi",

while rs1 = rd = "x28", immediate = -8

(sign bit = 1--> negative, 2's complement: 0000 0000 0111 + 1 = 0000 0000 1000 -->8)

The corresponding assembly instruction is "addi x28, x28, -8"

5. (20 points) Consider a new floating-point number representation that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the faction is 10 bits long. A hidden 1 to the left of the binary point is assumed. In this representation, any 16-bit binary pattern having 00000 in the exponent field and a non-zero fraction indicates a denormalized number: $(-1)^S \times (0 + \text{Fraction}) \times 2^{-14}$. Write the answers of (a), (b) and (c) in scientific notation, e.g., 1.0101×2^2.

(a) (3 points) What is the smallest positive "normalized" number, denoted as a0?

Sign bit = 1

Exponent = 00001 (because 11111 and 00000 are reserved)

-->actual exponent = 1-15 = -14

Fragment = 00000 00000

a0 = 1.00000 00000 x 2^(-14)

(b) (6 points) What is the largest positive "denormalized" number, denoted as a1? What is the second largest positive "denormalized" number, denoted as a2?

a1 = 0.11111 11111 x 2^(-14)

(the largest positive denormalized number's fragment = 11111 11111)

a2 = 0.11111 11110 x 2^(-14)

(Similarly, the second largest one have to be the one closest to a1,

So the fragment will be 11111 11110)

(c) (4 points) Find the differences between a0 and a1, and between a1 and a2. Also describe what you observe and any implication from them.

a0 – a1 = 0.00000 00001 x 2^(-14)

a1 – a0 = 0.00000 00001 x 2^(-14)

By the above equation, we see that the difference between a0 and a1

is equal to the one between a1 and a2.

Besides, we know that the smallest positive de-normalize number is

0.00000 00001 x 2^(-14) = 2^(-24) and the difference between this and the smallest positive normalized number is 2^(-10)

By representing de-normalized numbers, we can allow a number to degrade in significance until it becomes 0 (gradual underflow), and the equality of difference between a0 and a1 and the one between a1 and a2 is necessary for us in order to present the numbers between smallest de-normalized number and smallest normalized number.

(d) (3 points) What decimal number does the binary pattern 1011110110100111 represent?

1 01111 01101 00111

Sign bit: 1

Exponent : 01111 -->actual exponent = 15 – 15 = 0

Fragment : 01101 00111

In binary: -1.01101 00111

Convert to decimal:-(1 + 2^(-10)x(1 + 2 + 4 + 32 + 128 + 256)

= - (1024 + 256 + 128 + 32 + 4 + 2 + 1)x2^(-10)

= -1447x2^(-10)

= - 1.4130859375 ~= - 1.41

(e) (4 points) Let U be the nearest representation of the decimal number 1.24; that is, U has the smallest approximation error. What is U? What is the actual decimal number represented by U?

Try to convert 1.24 to binary:

1-->1

0.24--> 0.00111 10101 110..... -->0.00111 10110

(0.48, 0.96, 1.92->0.92,1.84->0.84,1.68->0.68,1.36->0.36,0.72,

1.44->0.44,0.88,1.76->0.76,1.52->0.52,1.04->0.04)

Sign bit:0
Exponent:00000 + 01111 = 01111
Fragment:00111 10110
1.00111 10110 -->2^(-9)x(1 + 2 + 8 +16 + 32 + 64 + 512)
= 2^(-9) x 635
= 1.240234375