CSE331
Computer Organization
3rd Assignment
Report

Date

19/12/21

Author

Gökbey Gazi Keskin

1901042631

Table of Contents

Part 1

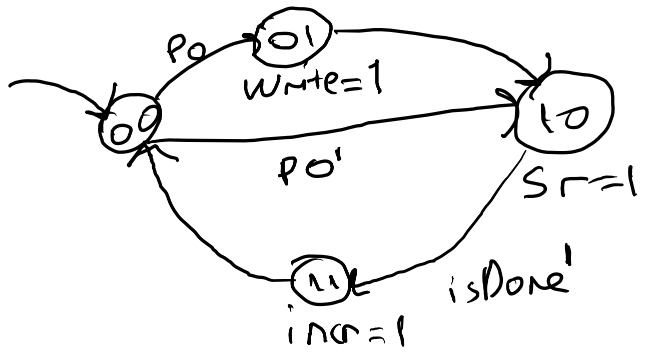
I only made the controller part of the Multiplier.

Controller

Inputs: p0, isDone

Outputs: write, sr, incr

I planned to make a counter in datapath and increment it every time incr is 1. When it is 32 isDone should be 1.



S1	S0	p0	isDone	N1	N0	write	sr	incr
0	0	0	X	1	0	0	0	0
0	0	1	X	0	1	0	0	0
0	1	X	X	1	0	1	0	0
1	0	X	0	1	1	0	1	0
1	1	X	X	0	0	0	0	1

N1:

S1S0/p0isDone	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10	1			1

$S_1' p_0' + S_1'S_0 + S_1S_0' \, isDone'$

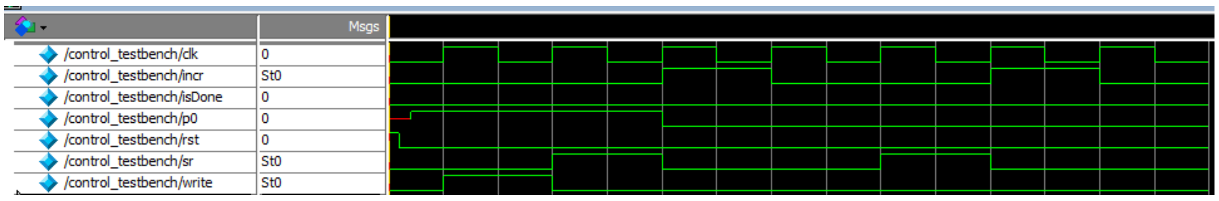
N0:

S1S0/p0isDone	00	01	11	10
00			1	1
01				
11				
10	1			1

$S_1'S_0' p_0 + S_1S_0' \, isDone'$

write: S1'S0 , sr: S1S0', incr S1S0

Testbench for Multiplier Controller


Msgs
/control_testbench/clk 0
/control_testbench/incr St0
/control_testbench/isDone 0
/control_testbench/p0 0
/control_testbench/rst 0
/control_testbench/sr St0
/control_testbench/write St0

```
# time =  2,p0:1, PS:00 NS:01
# time =  5,p0:1, PS:01 NS:10
# time = 15,p0:1, PS:10 NS:11
# time = 25,p0:0, PS:11 NS:00
# time = 35,p0:0, PS:00 NS:10
# time = 45,p0:0, PS:10 NS:11
# time = 55,p0:0, PS:11 NS:00
# time = 65,p0:0, PS:00 NS:10
```

-next state is 01 when present state is 00 and p0 is 1

- next state is 10 when present state is 00 and p0 is 0

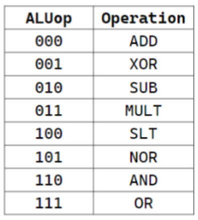
-next state is 10 when present state is 01

-next state is 11 when present state is 10 and isDone is 0

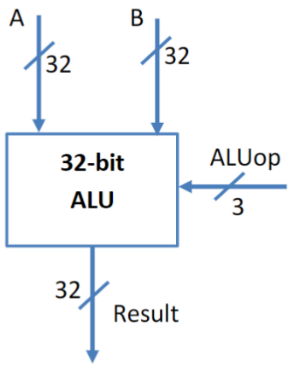
- next state is 00 when present state is 11

Controller works as expected.

Part 2



ALUop	Operation
000	ADD
001	XOR
010	SUB
011	MULT
100	SLT
101	NOR
110	AND
111	OR



Required Submodules:

1 – 32 Bit Adder

Inputs: A[32], B[32], C0

Outputs: S, Cout, C30 (C30 is used at set less than module to determine if there is an overflow or not C_n XOR C_{n-1} gives the overflow)

I used the full adder given by the T.A of the lecture.

Required Submodule: 1-bit full adder (which requires 1-bit half adder)

2 – 32 Bit XOR Gate

Inputs: A[32], B[32]

Output: R[32]

I XORed inputs bit by bit

3 – 32 Bit Subtractor

Inputs A[32], B[32]

Outputs: R[32], Cout, CnMinus1 (CnMinus1 is used at set less than module to determine if there is an overflow or not C_n XOR C_{n-1} gives the overflow)

Carry in of the 32-bit adder is constant 1. B is reverted by the 32-bit NOT gate which I implemented.

Required Submodules:

-32-bit adder

-32 bit not gate

4 - 32 Bit Sequential Multiplier (1st part of the Assignment)

Explained in the first part.

5 – Set Less Than

Inputs: A[32], B[32]

Output: S

Required Submodule: 32-bit subtractor

S = (cout xor c_{n-1}) xor sub(A,B)[31] (isOverflow xor most significant bit of the subtraction)

6 – 32 Bit NOR Gate

Inputs: A[32], B[32]

Output: R[32]

I NORed inputs bit by bit

7 – 32 Bit AND Gate

Inputs: A[32], B[32]

Output: R[32]

I ANDed inputs bit by bit

8 – 32 Bit OR Gate

Inputs: A[32], B[32]

Output: R[32]

I ORed inputs bit by bit

9 – 8x1 32 Bit Multiplexer

Inputs: A[32], B[32], C[32], D[32], E[32], F[32], G[32], H[32], S[3]

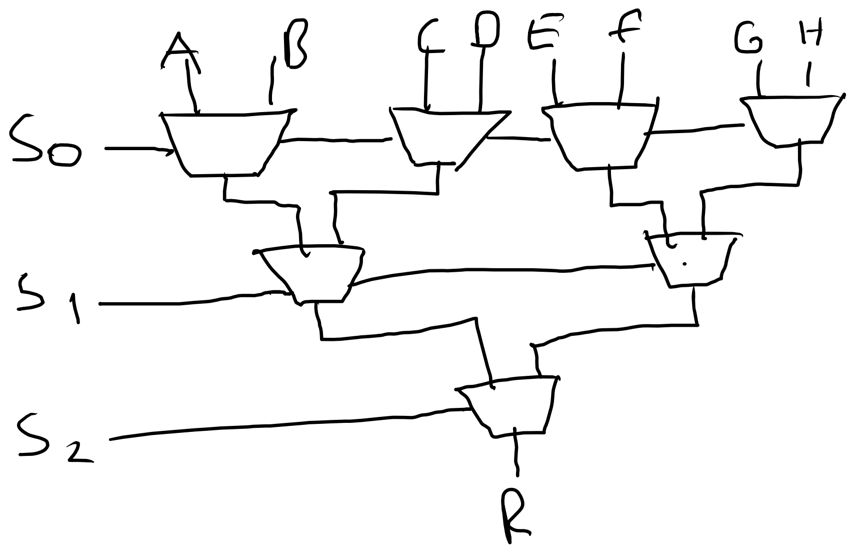
Output: R

First, I made a 2x1 1 bit multiplexer.

R = A and S' + B and S

Then, I made a 2x1 32-bit mux by using 32 2x1 1-bit muxes

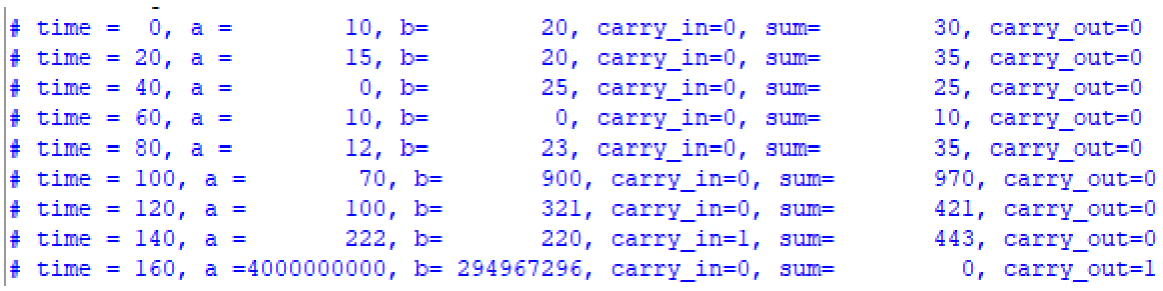
Finally, I made an 8x1 32-bit mux by using 7 2x1 32-bit muxes



Testbenches for ALU and its submodules:

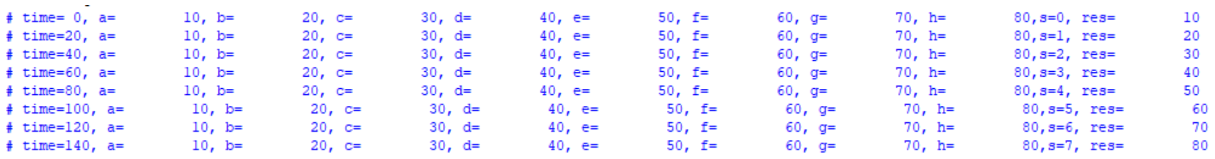
(All the submodule testbenches are included the file.)

32-bit adder



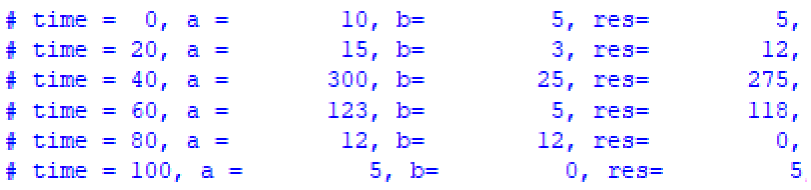
```
# time =   0, a =         10, b=        20, carry_in=0, sum=         30, carry_out=0
# time = 20, a =         15, b=        20, carry_in=0, sum=         35, carry_out=0
# time = 40, a =          0, b=        25, carry_in=0, sum=         25, carry_out=0
# time = 60, a =         10, b=         0, carry_in=0, sum=         10, carry_out=0
# time = 80, a =         12, b=        23, carry_in=0, sum=         35, carry_out=0
# time = 100, a =         70, b=       900, carry_in=0, sum=        970, carry_out=0
# time = 120, a =        100, b=       321, carry_in=0, sum=        421, carry_out=0
# time = 140, a =        222, b=       220, carry_in=1, sum=        443, carry_out=0
# time = 160, a =4000000000, b= 294967296, carry_in=0, sum=          0, carry_out=1
```

32-bit 8x1 mux



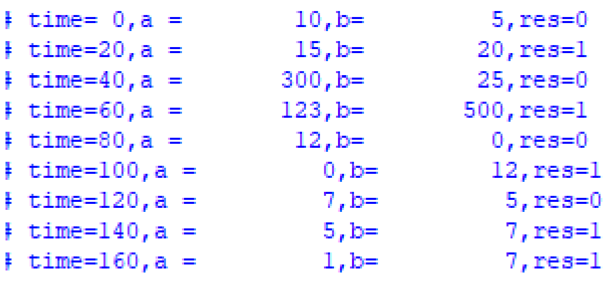
```
# time= 0, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=0, res=        10
# time=20, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=1, res=        20
# time=40, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=2, res=        30
# time=60, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=3, res=        40
# time=80, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=4, res=        50
# time=100, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=5, res=        60
# time=120, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=6, res=        70
# time=140, a=        10, b=        20, c=        30, d=        40, e=        50, f=        60, g=        70, h=        80,s=7, res=        80
```

32-bit subtractor



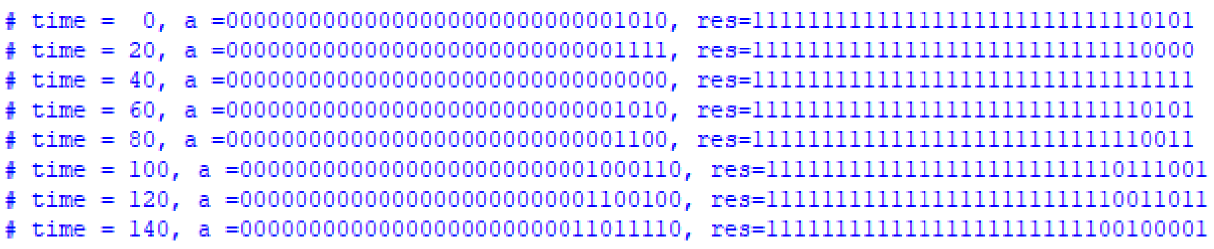
```
# time =   0, a =         10, b=          5, res=          5,
# time = 20, a =         15, b=          3, res=         12,
# time = 40, a =        300, b=         25, res=        275,
# time = 60, a =        123, b=          5, res=        118,
# time = 80, a =         12, b=         12, res=          0,
# time = 100, a =          5, b=          0, res=          5
```

Set-less-then



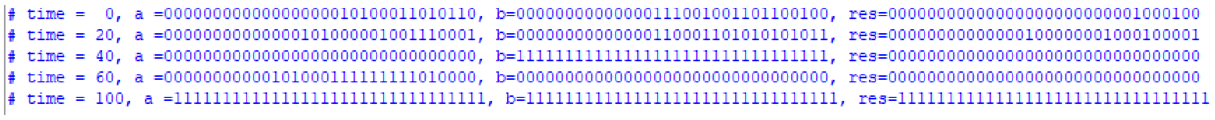
```
# time= 0,a =         10,b=          5,res=0
# time=20,a =         15,b=         20,res=1
# time=40,a =        300,b=         25,res=0
# time=60,a =        123,b=        500,res=1
# time=80,a =         12,b=          0,res=0
# time=100,a =          0,b=         12,res=1
# time=120,a =          7,b=          5,res=0
# time=140,a =          5,b=          7,res=1
# time=160,a =          1,b=          7,res=1
```

32-bit not



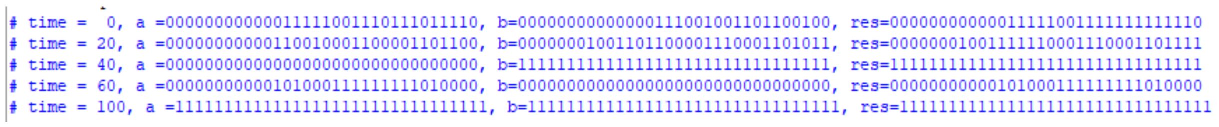
```
# time =   0, a =00000000000000000000000000001010, res=11111111111111111111111111110101
# time = 20, a =00000000000000000000000000001111, res=11111111111111111111111111110000
# time = 40, a =00000000000000000000000000000000, res=11111111111111111111111111111111
# time = 60, a =00000000000000000000000000001010, res=11111111111111111111111111110101
# time = 80, a =00000000000000000000000000001100, res=11111111111111111111111111110011
# time = 100, a =00000000000000000000000001000110, res=11111111111111111111111110111001
# time = 120, a =00000000000000000000000001100100, res=11111111111111111111111110011011
# time = 140, a =00000000000000000000000011011110, res=11111111111111111111111100100001
```

32-bit add

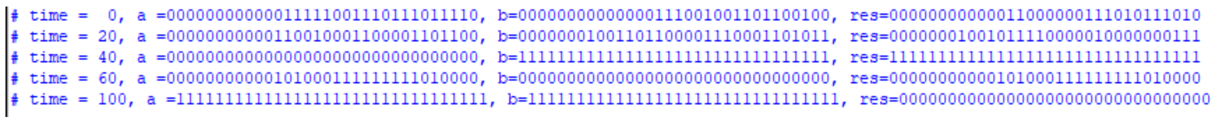


```
# time =   0, a =00000000000000000010100011010110, b=00000000000000011100100110110100, res=00000000000000000000000001000100
# time = 20, a =00000000000000010100000100111000 1, b=00000000000000011000110101010101 1, res=00000000000000010000000100010000 1
# time = 40, a =00000000000000000000000000000000, b=11111111111111111111111111111111, res=00000000000000000000000000000000
# time = 60, a =00000000000101000111111111010000, b=00000000000000000000000000000000, res=00000000000000000000000000000000
# time = 100, a =11111111111111111111111111111111, b=11111111111111111111111111111111, res=11111111111111111111111111111111
```

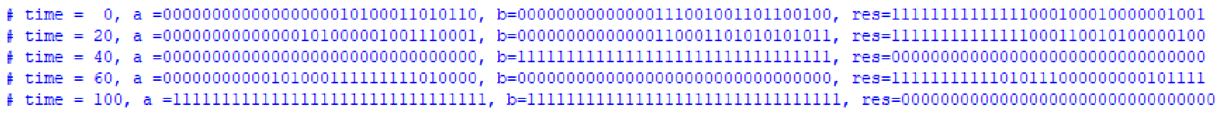
32-bit or


```
# time =  0, a =00000000000001111100111011101110, b=00000000000000111001001101100100, res=00000000000001111100111111111110
# time = 20, a =00000000000011001000110000110110, b=00000001001101100001110001101011, res=00000001001111110001110001101111
# time = 40, a =00000000000000000000000000000000, b=11111111111111111111111111111111, res=11111111111111111111111111111111
# time = 60, a =00000000000010100011111111010000, b=00000000000000000000000000000000, res=00000000000010100011111111010000
# time = 100, a =11111111111111111111111111111111, b=11111111111111111111111111111111, res=11111111111111111111111111111111
```

32-bit xor

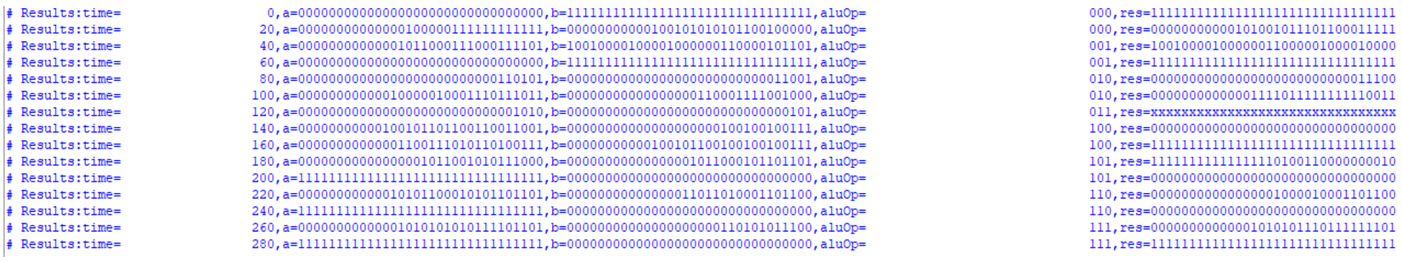

```
# time =  0, a =00000000000001111100111011101110, b=00000000000000111001001101100100, res=00000000000001100000011101011101
# time = 20, a =00000000000011001000110000110110, b=00000001001101100001110001101011, res=00000001001011110000010000000111
# time = 40, a =00000000000000000000000000000000, b=11111111111111111111111111111111, res=11111111111111111111111111111111
# time = 60, a =00000000000010100011111111010000, b=00000000000000000000000000000000, res=00000000000010100011111111010000
# time = 100, a =11111111111111111111111111111111, b=11111111111111111111111111111111, res=00000000000000000000000000000000
```

32-bit nor

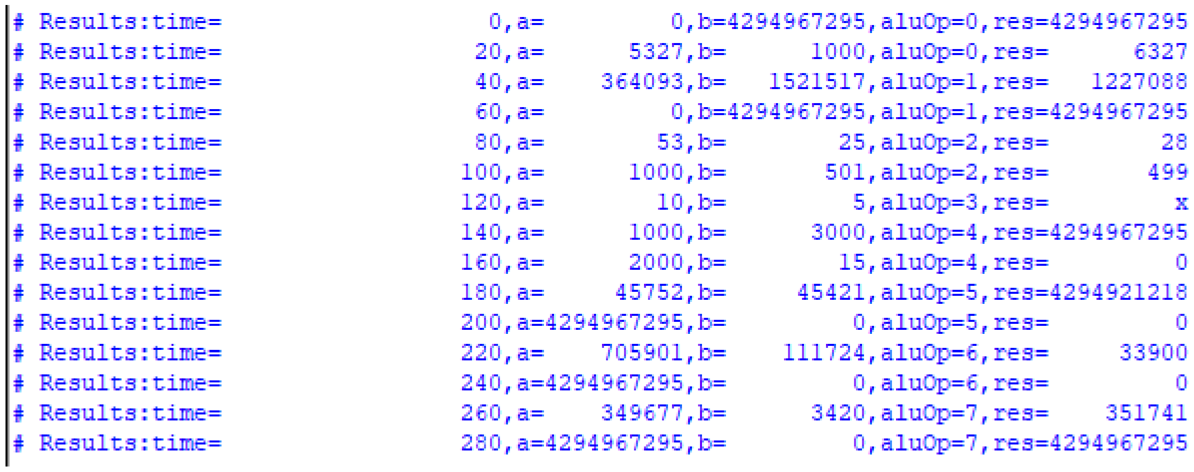

```
# time =  0, a =00000000000000000101000110101110, b=00000000000000111001001101100100, res=11111111111111000100010000001001
# time = 20, a =00000000000000101000001001110001, b=00000000000000110001101010101011, res=11111111111111000110010100000100
# time = 40, a =00000000000000000000000000000000, b=11111111111111111111111111111111, res=00000000000000000000000000000000
# time = 60, a =00000000000010100011111111010000, b=00000000000000000000000000000000, res=11111111111101011100000000101111
# time = 100, a =11111111111111111111111111111111, b=11111111111111111111111111111111, res=00000000000000000000000000000000
```

ALU

Test in binary format (for binary operations):


```
# Results:time=                    0,a=00000000000000000000000000000000,b=11111111111111111111111111111111,aluOp=                    000,res=11111111111111111111111111111111
# Results:time=                   20,a=00000000000000010000011111111111,b=00000000000100101010101100100000,aluOp=                    000,res=00000000000101001011101100011111
# Results:time=                   40,a=00000000000001011000111000111101,b=10010000100001000000110000101101,aluOp=                    001,res=10010000100000011000001000010000
# Results:time=                   60,a=00000000000000000000000000000000,b=11111111111111111111111111111111,aluOp=                    001,res=11111111111111111111111111111111
# Results:time=                   80,a=00000000000000000000000000110101,b=00000000000000000000000000011001,aluOp=                    010,res=00000000000000000000000000011100
# Results:time=                  100,a=00000000000010000010001110111011,b=00000000000000000110001111001000,aluOp=                    010,res=00000000000001111011111111110011
# Results:time=                  120,a=00000000000000000000000000001010,b=00000000000000000000000000000101,aluOp=                    011,res=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# Results:time=                  140,a=00000000000100101101100110011001,b=00000000000000000000000100100111,aluOp=                    100,res=00000000000000000000000000000000
# Results:time=                  160,a=00000000000001100111010110100111,b=00000000000100101100100100100111,aluOp=                    100,res=11111111111111111111111111111111
# Results:time=                  180,a=00000000000000001011001010111000,b=00000000000000001011000101101101,aluOp=                    101,res=11111111111111110100110000000010
# Results:time=                  200,a=11111111111111111111111111111111,b=00000000000000000000000000000000,aluOp=                    101,res=00000000000000000000000000000000
# Results:time=                  220,a=00000000000010101100010101101101,b=00000000000000011011010001101100,aluOp=                    110,res=00000000000000001000010001101100
# Results:time=                  240,a=11111111111111111111111111111111,b=00000000000000000000000000000000,aluOp=                    110,res=00000000000000000000000000000000
# Results:time=                  260,a=00000000000001010101010111101101,b=00000000000000000000110101011100,aluOp=                    111,res=00000000000001010101110111111101
# Results:time=                  280,a=11111111111111111111111111111111,b=00000000000000000000000000000000,aluOp=                    111,res=11111111111111111111111111111111
```

In decimal format(for arithmetic operations):


```
# Results:time=                    0,a=         0,b=4294967295,aluOp=0,res=4294967295
# Results:time=                   20,a=      5327,b=      1000,aluOp=0,res=      6327
# Results:time=                   40,a=    364093,b=   1521517,aluOp=1,res=   1227088
# Results:time=                   60,a=         0,b=4294967295,aluOp=1,res=4294967295
# Results:time=                   80,a=        53,b=        25,aluOp=2,res=        28
# Results:time=                  100,a=      1000,b=       501,aluOp=2,res=       499
# Results:time=                  120,a=        10,b=         5,aluOp=3,res=         x
# Results:time=                  140,a=      1000,b=      3000,aluOp=4,res=4294967295
# Results:time=                  160,a=      2000,b=        15,aluOp=4,res=         0
# Results:time=                  180,a=     45752,b=     45421,aluOp=5,res=4294921218
# Results:time=                  200,a=4294967295,b=         0,aluOp=5,res=         0
# Results:time=                  220,a=    705901,b=    111724,aluOp=6,res=     33900
# Results:time=                  240,a=4294967295,b=         0,aluOp=6,res=         0
# Results:time=                  260,a=    349677,b=      3420,aluOp=7,res=    351741
# Results:time=                  280,a=4294967295,b=         0,aluOp=7,res=4294967295
```