COMPUTER ORGANIZATION

HOMEWORK 3

ÖMER FARUK BİTİKÇİOĞLU
161044010

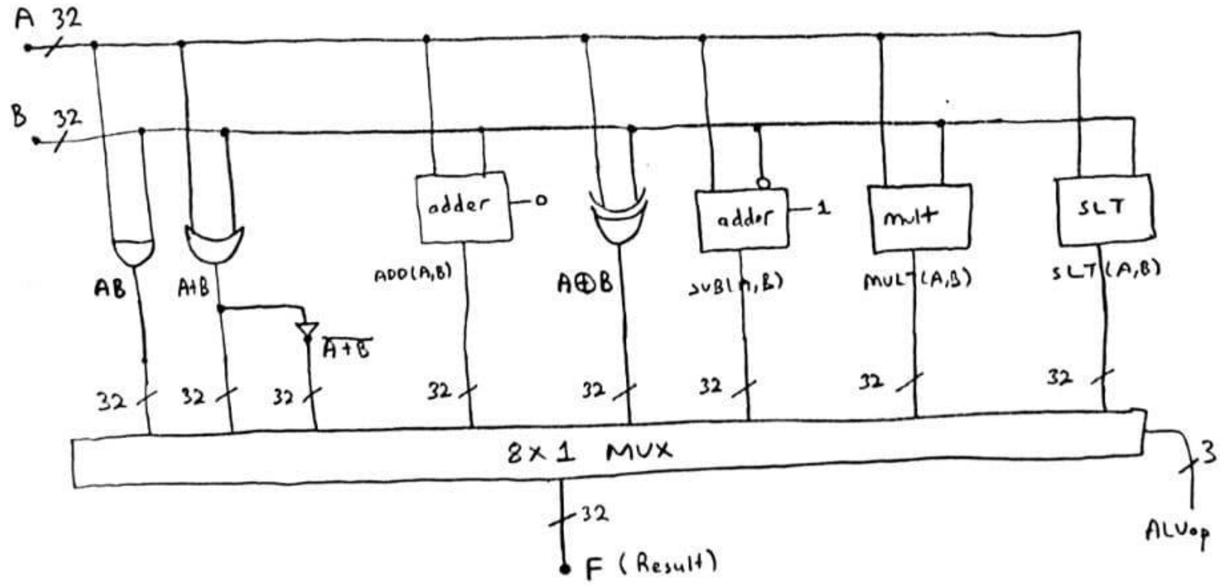
Firstly, I found formulas for adder. Then I sketched alu32 on paper. Then I started to implement all the components to combine with alu32. To choose a value I implemented an 8x1 multiplexer.

Fadder

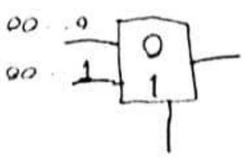
a	b	cin	sum	cout
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

$$
\begin{aligned}
sum &= ab c_{in} + ab' c_{in}' + a'b c_{in}' + a'b' c_{in} \\
&= c_{in}(ab + a'b') + c_{in}'(ab' + a'b) \\
&= c_{in}(a \oplus b)' + c_{in}'(a \oplus b) \\
&= c_{in} \oplus (a \oplus b)
\end{aligned}
$$

$$
\begin{aligned}
c_{out} &= ab c_{in} + ab c_{in}' + ab' c_{in} + a'b c_{in} \\
&= ab(c_{in} + c_{in}') + c_{in}(ab' + a'b) \\
&= ab + c_{in}(a \oplus b)
\end{aligned}
$$



S_2	S_1	S_0	Op	
0	0	0	ADD	$S_2' S_1' S_0' D_0$
0	0	1	XOR	$+ S_2' S_1' S_0 D_1$
0	1	0	SUB	$+ S_2' S_1 S_0' D_2$
0	1	1	MULT	$+ S_2' S_1 S_0 D_3$
1	0	0	SLT	$+ S_2 S_1' S_0' D_4$
1	0	1	NOR	$+ S_2 S_1' S_0 D_5$
1	1	0	AND	$+ S_2 S_1 S_0' D_6$
1	1	1	OR	$+ S_2 S_1 S_0 D_7$



Adder / full adder

To implement 32-bit adder, I used full adder and call it in a loop. To use carry in and find the exact carry out I separated the adding first and last bits.

Full adder testbench results:

```
# time =  0, a =0, b=0, carry_in=0, sum=0, carry_out=0
# time = 20, a =0, b=0, carry_in=1, sum=1, carry_out=0
# time = 40, a =0, b=1, carry_in=0, sum=1, carry_out=0
# time = 60, a =0, b=1, carry_in=1, sum=0, carry_out=1
# time = 80, a =1, b=0, carry_in=0, sum=1, carry_out=0
# time = 100, a =1, b=0, carry_in=1, sum=0, carry_out=1
# time = 120, a =1, b=1, carry_in=0, sum=0, carry_out=1
# time = 140, a =1, b=1, carry_in=1, sum=1, carry_out=1
```

Adder testbench results:

```
# time =  0, a =11110100011111110100010000001000, b=10110001110100101000110100001000, carry_in=0, sum=10100110010100011101000100010000, carry_out=1
# time = 20, a =11110100011111110100010000001000, b=10110001110100101000110100001000, carry_in=1, sum=10100110010100011101000100010001, carry_out=1
# time = 40, a =00011101100001001000000000010110, b=01010100101011100010100001000111, carry_in=0, sum=01110010001100101010100001011101, carry_out=0
# time = 60, a =00011101100001001000000000010110, b=01010100101011100010100001000111, carry_in=1, sum=01110010001100101010100001011110, carry_out=0
# time = 80, a =10001010111110000110111101011001, b=00110001101100001100010111011010, carry_in=0, sum=10111100101010010011010100110011, carry_out=0
# time = 100, a =10001010111110000110111101011001, b=00110001101100001100010111011010, carry_in=1, sum=10111100101010010011010100110100, carry_out=0
# time = 120, a =11111111100011000111100101111100, b=10111110000011000101001100010100, carry_in=0, sum=10111101100110001100110010010000, carry_out=1
# time = 140, a =11111111100011000111100101111100, b=10111110000011000101001100010100, carry_in=1, sum=10111101100110001100110010010001, carry_out=1
# time = 160, a =11000000001100011010110111111001, b=01111000010001011110110011010011, carry_in=0, sum=00111000011101111001101011001100, carry_out=1
# time = 180, a =11000000001100011010110111111001, b=01111000010001011110110011010011, carry_in=1, sum=00111000011101111001101011001101, carry_out=1
# time = 200, a =10011010111110011011000111011101, b=00111010101001111101001010001110, carry_in=0, sum=11010101101000011000010001101011, carry_out=0
# time = 220, a =10011010111110011011000111011101, b=00111010101001111101001010001110, carry_in=1, sum=11010101101000011000010001101100, carry_out=0
# time = 240, a =11011111110101101010101000010101, b=10100010001100110101000101001010, carry_in=0, sum=10000010000010011111101101011111, carry_out=1
# time = 260, a =11011111110101101010101000010101, b=10100010001100110101000101001010, carry_in=1, sum=10000010000010011111101101100000, carry_out=1
# time = 280, a =01010000111100100001101010110110, b=00110111111101001011101001010100, carry_in=0, sum=10001000111001101101010100001010, carry_out=0
# time = 300, a =01010000111100100001101010110110, b=00110111111101001011101001010100, carry_in=1, sum=10001000111001101101010100001011, carry_out=0
```

Or32 / Xor32 / And32

I used dynamic programming to solve these. I downgraded the problem to 16-bit, 8-bit and 4-bit then combine the results.

And32 testbench results:

```
# time =  0, inp1= 11110100011111110100010000001000, inp2= 10110001110100101000110100001000, and= 10110000010100100000010000001000
# time = 40, inp1= 00011101100001001000000000010110, inp2= 01010100101011100010100001000111, and= 00010100100001000000000000000110
# time = 60, inp1= 10001010111110000110111101011001, inp2= 00110001101100001100010111011010, and= 00000000101100000100010101011000
# time = 80, inp1= 11111111100011000111100101111100, inp2= 10111110000011000101001100010100, and= 10111110000011000101000100010100
# time = 100, inp1= 11000000001100011010110111111001, inp2= 01111000010001011110110011010011, and= 01000000000000011010110011010001
```

Or32 testbench results:

```
# time =  0, inp1= 11110100011111110100010000001000, inp2= 10110001110100101000110100001000, or= 11110101111111111100110100001000
# time = 40, inp1= 00011101100001001000000000010110, inp2= 01010100101011100010100001000111, or= 01011101101011101010100001010111
# time = 60, inp1= 10001010111110000110111101011001, inp2= 00110001101100001100010111011010, or= 10111011111110001110111111011011
# time = 80, inp1= 11111111100011000111100101111100, inp2= 10111110000011000101001100010100, or= 11111111100011000111101101111100
# time = 100, inp1= 11000000001100011010110111111001, inp2= 01111000010001011110110011010011, or= 11111000011101011110110111111011
```

Xor32 testbench results:

```
# time =  0, inp1= 11110100011111110100010000001000, inp2= 10110001110100101000110100001000, xor= 01000101101011011100100100000000
# time = 40, inp1= 00011101100001001000000000010110, inp2= 01010100101011100010100001000111, xor= 01001001001010101010100001010001
# time = 60, inp1= 10001010111110000110111101011001, inp2= 00110001101100001100010111011010, xor= 10111011010010001010101010000011
# time = 80, inp1= 11111111100011000111100101111100, inp2= 10111110000011000101001100010100, xor= 01000001100000000010101001101000
# time = 100, inp1= 11000000001100011010110111111001, inp2= 01111000010001011110110011010011, xor= 10111000011101000100000100101010
```

Sub32

To subtract B from A, I get the two’s complement of B, then I added it to A. So, I didn’t need to implement a new component. I used adder.

Sub32 testbench results:

```
# time =  0, A= 11110100011111110100010000001000, B= 10110001110100101000110100001000, sub= 01000010101011001011011100000000
# time = 40, A= 00011101100001001000000000010110, B= 01010100101011100010100001000111, sub= 11001000110101100101011111001111
# time = 60, A= 10001010111110000110111101011001, B= 00110001101100001100010111011010, sub= 01011001010001111010100101111111
# time = 80, A= 11111111100011000111100101111100, B= 10111110000011000101001100010100, sub= 01000001100000000010011001101000
# time = 100, A= 11000000001100011010110111111001, B= 01111000010001011110110011010011, sub= 01000111111010111100000100100110
```

Not32

Simply I used the not gate for 32 times in a loop.

Not32 testbench results:

```
# time =  0, inp= 11110100011111111010001000001000, not= 00001011100000000101110111110111
# time = 20, inp= 10110001110100101000110100001000, not= 01001110001011010111001011110111
# time = 40, inp= 11110100011111111010001000001000, not= 00001011100000000101110111110111
# time = 60, inp= 10110001110100101000110100001000, not= 01001110001011010111001011110111
# time = 80, inp= 00011101100001001000000000010110, not= 11100010011110110111111111101001
# time = 100, inp= 01010100101011100010100001000111, not= 10101011010100011101011110111000
# time = 120, inp= 10001010111110000011011101011001, not= 01110101000001111100100010100110
# time = 140, inp= 00110001101100001100010111011010, not= 11001110010011110011101000100101
# time = 160, inp= 11111111100011000111100101111100, not= 00000000011100111000011010000011
# time = 180, inp= 10111110000011000101001100010100, not= 01000001111100111010110011101011
# time = 200, inp= 11000000001100011010110111111001, not= 00111111110011100101001000000110
# time = 220, inp= 01111000010001011110110011010011, not= 10000111101110100001001100101100
```

Set on Less Than

I subtracted B from A, then checked for the most significant bit. If A < B, then the result of A-B must be negative (most significant bit = 1).

SLT testbench results:

```
# time =  0, A= 11110100011111111010001000001000, B= 10110001110100101000110100001000, slt= 00000000000000000000000000000000
# time = 20, A= 11111010111110000110111101011001, B= 00000001101100001100010111011010, slt= 00000000000000000000000000000001
# time = 40, A= 00100011101100010010110101010001, B= 00100101101100101100010111011000, slt= 00000000000000000000000000000001
# time = 60, A= 11111111100011000111100101111100, B= 10111110000011000101001100010100, slt= 00000000000000000000000000000000
# time = 80, A= 11000000001100011010110111111001, B= 00001000010001011110110011010011, slt= 00000000000000000000000000000001
```

8x1 Multiplexer

This module selects the desired bit and sends it to F. The formula is on the bottom of the paper that I put on the first page of this report.

8x1mux testbench results:

```
# time =  0, D0= 1, D1= 0, D2= 0, D3= 0, D4= 0, D5= 1, D6= 1, D7= 1, S= 000, F= 1
# time = 20, D0= 0, D1= 1, D2= 1, D3= 0, D4= 1, D5= 1, D6= 1, D7= 1, S= 001, F= 1
# time = 40, D0= 1, D1= 0, D2= 1, D3= 0, D4= 0, D5= 0, D6= 0, D7= 1, S= 010, F= 1
# time = 60, D0= 0, D1= 1, D2= 0, D3= 1, D4= 1, D5= 1, D6= 1, D7= 0, S= 011, F= 1
# time = 80, D0= 0, D1= 1, D2= 1, D3= 1, D4= 1, D5= 1, D6= 0, D7= 1, S= 100, F= 1
# time = 100, D0= 1, D1= 0, D2= 0, D3= 0, D4= 1, D5= 0, D6= 1, D7= 1, S= 101, F= 0
# time = 120, D0= 0, D1= 0, D2= 1, D3= 1, D4= 0, D5= 0, D6= 0, D7= 0, S= 110, F= 0
# time = 140, D0= 0, D1= 1, D2= 0, D3= 0, D4= 1, D5= 1, D6= 0, D7= 0, S= 111, F= 0
```

ALU32

I combined all the modules and selected the desired results with multiplexer.

Alu32 testbench results:

```
# time =  0, A= 11110100011111111010001000001000, B= 10110001110100101000110100001000, ALUop= 000, F= 10100110010100011101000100010000
# time = 20, A= 11111010111110000110111101011001, B= 00000001101100001100010111011010, ALUop= 001, F= 11111011010010001010101010000011
# time = 40, A= 00100011101100010010110101010001, B= 00100101101100101100010111011000, ALUop= 010, F= 11111101111111110011001110111001
# time = 60, A= 11111111100011000111100101111100, B= 10111110000011000101001100010100, ALUop= 011, F= xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# time = 80, A= 11000000001100011010110111111001, B= 00001000010001011110110011010011, ALUop= 100, F= 00000000000000000000000000000001
# time = 100, A= 00111111110101110111011010011011, B= 10000100100111011110111110110011, ALUop= 101, F= 01000000001000000000000001000100
# time = 120, A= 10110001111100011011010000101111, B= 11011010000000110110000001001111, ALUop= 110, F= 10010000000000010010000000001111
# time = 140, A= 00100011000100000000011111110011, B= 10001011110101000101001101000110, ALUop= 111, F= 10101011110101000101011111110111
```

Last words

- All the implemented modules work fine.
- Nor is simply not of or.
- If you are getting wrong results, it probably because of overflowing 32-bits.
- Multiplier is not implemented.