Gebze Technical University
Computer Engineering

CSE 331

PROJECT 2 REPORT

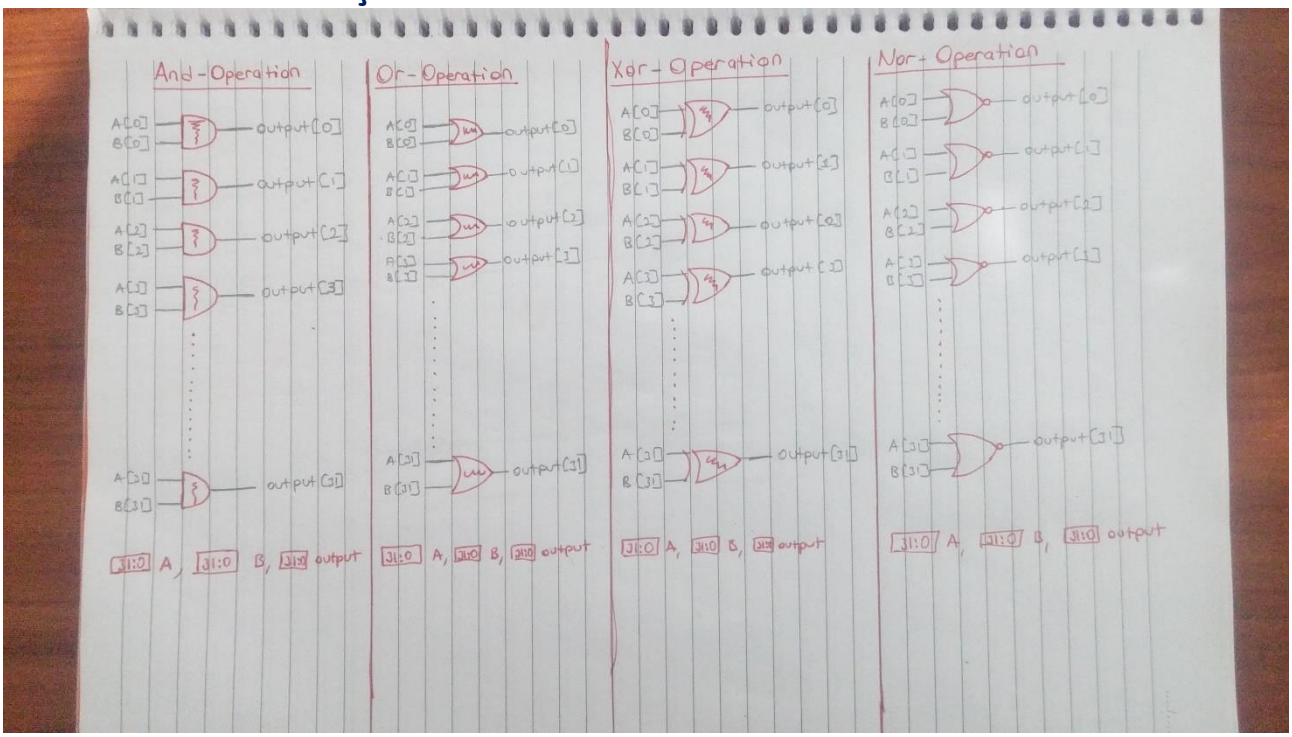
AKIN ÇAM
151044007

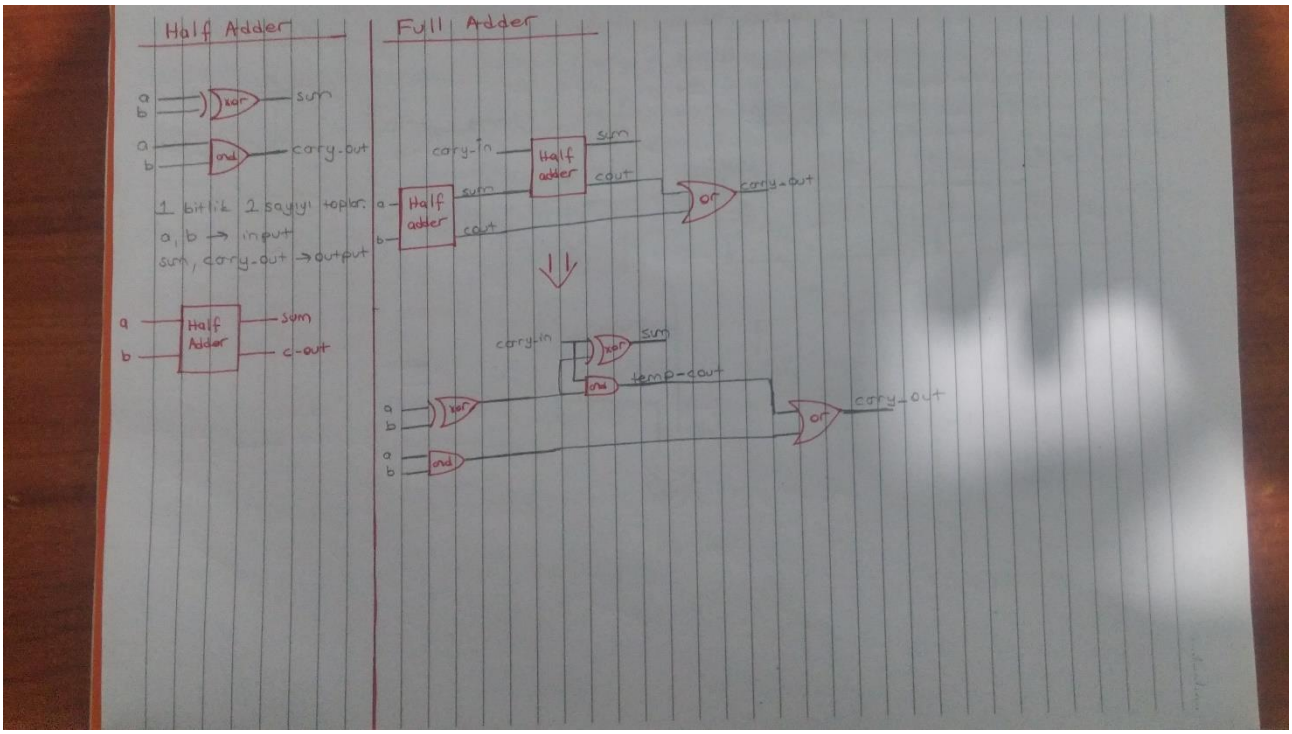
Course Assistant:

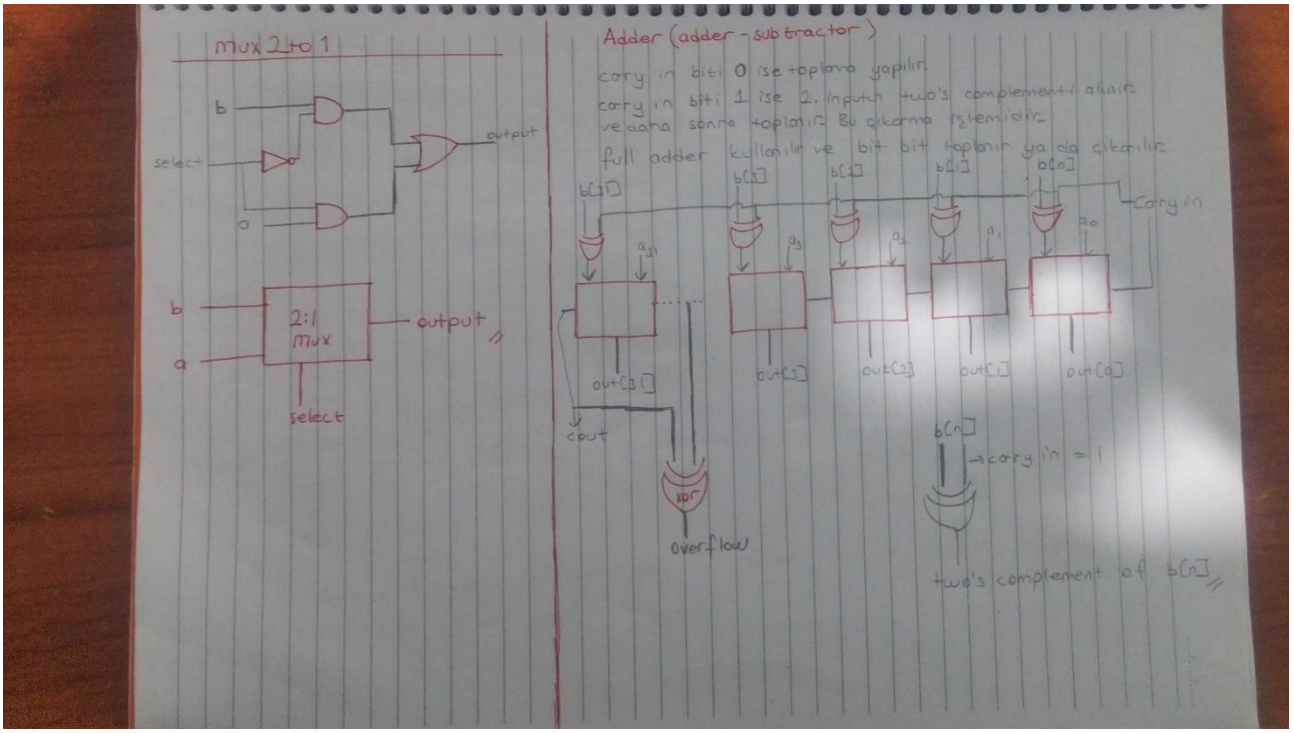
1) Schematic Designs for All Modules

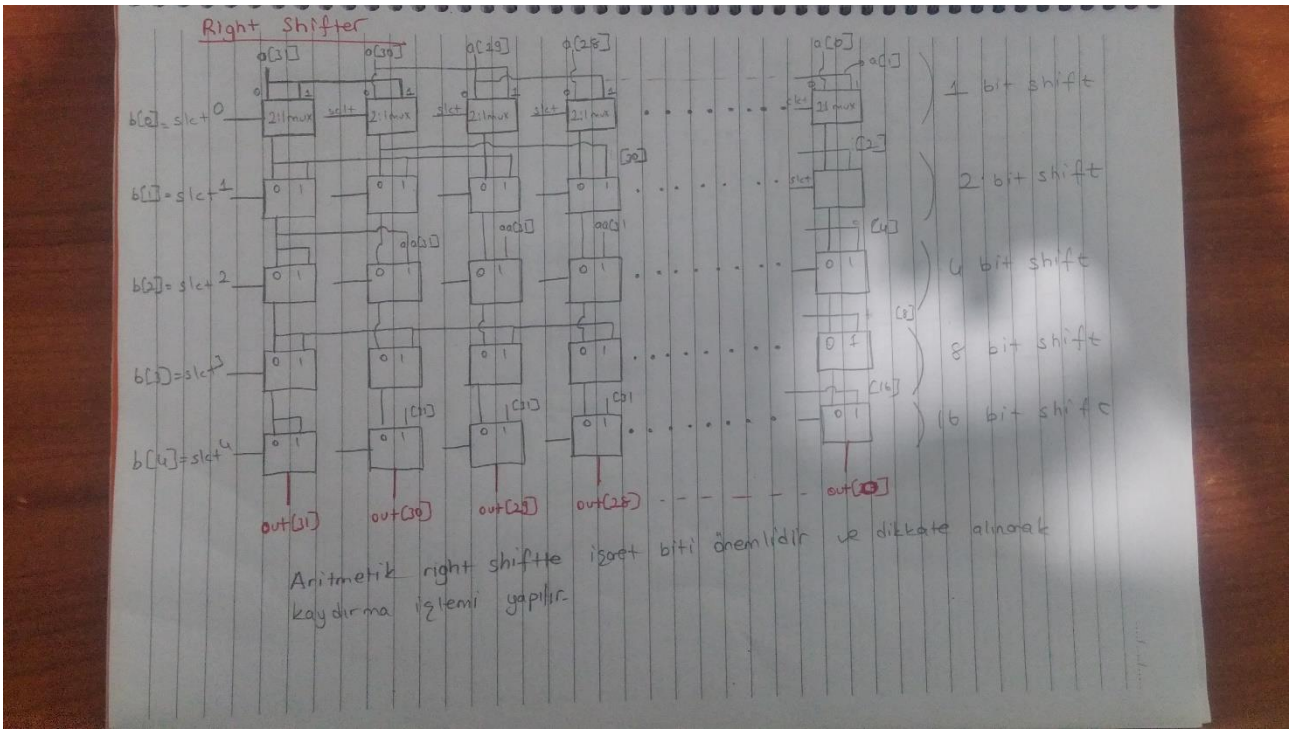
32 bitlik iki sayıyı input olarak alır and, or, xor, nor, adder, subtractor, leftshifter ve aritmetic rightshifter işlemlerinden birini verilen select bitine göre gerçekleştirir.

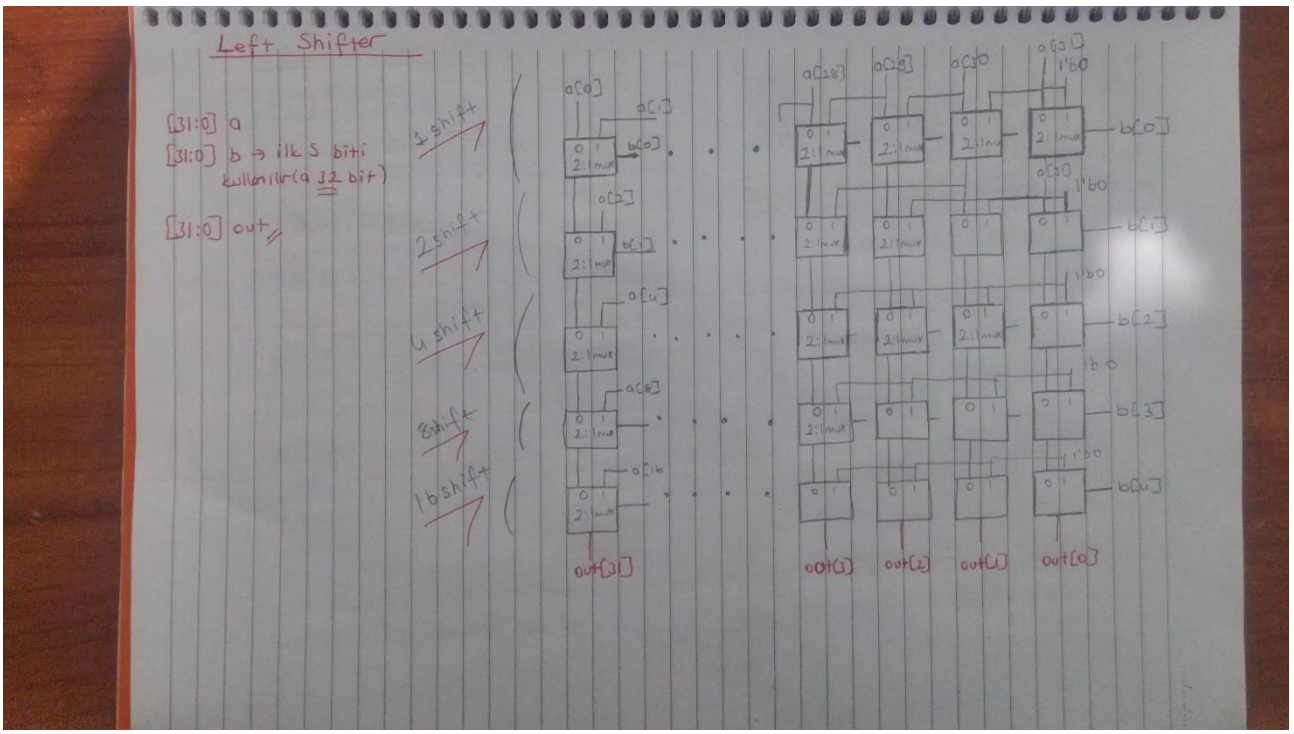
####BONUS PART####
***Overflow durumu eklenmiştir.**
***Zero durumu eklenmiştir.**

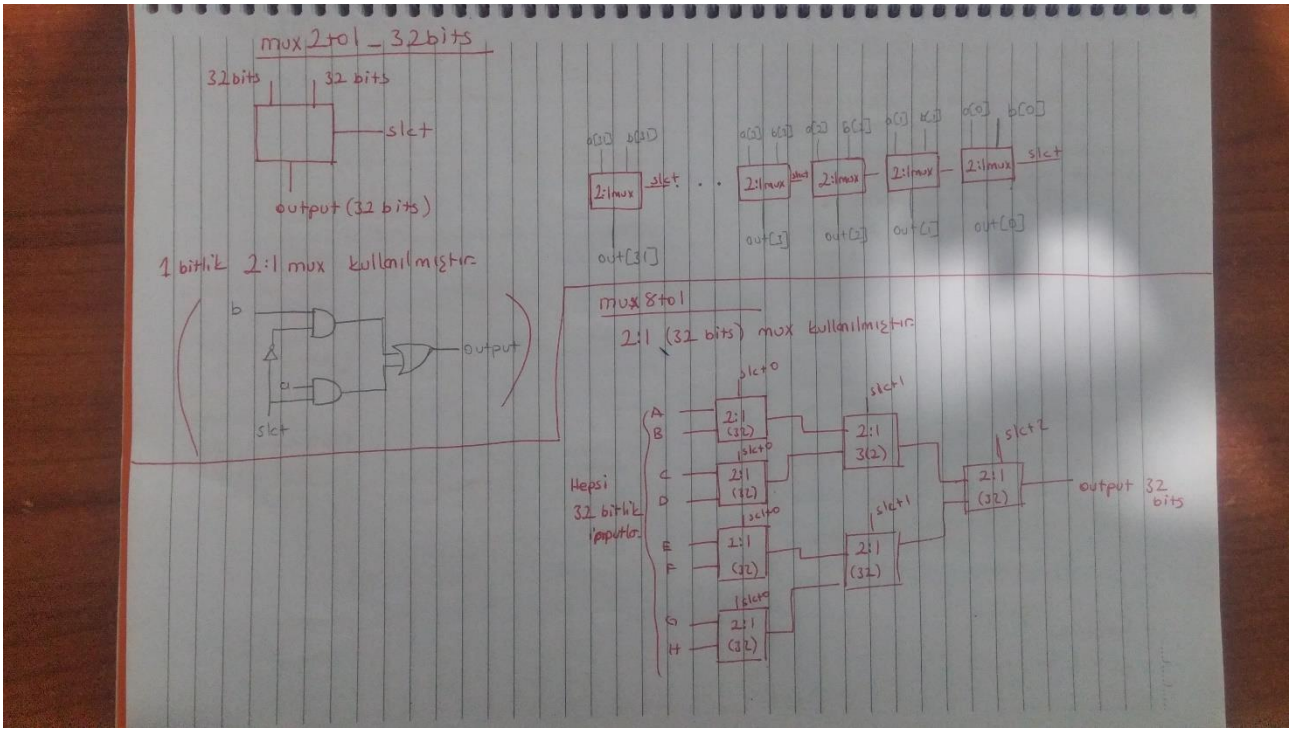


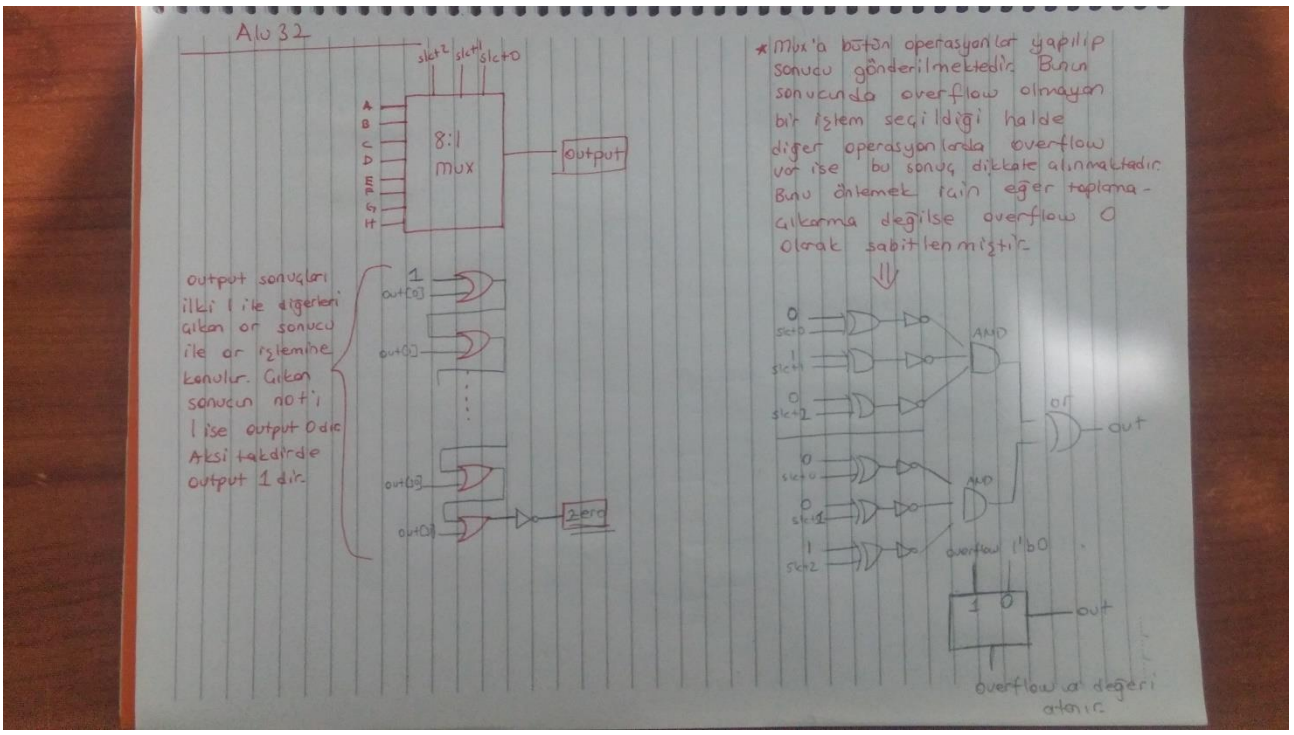



Mux 2 to 1
b
select
0
output
b
a
2:1 mux
output
select
Adder (adder - subtractor)
cary in biti 0 ise toplama yapilir
cary in biti 1 ise 2. inputun two's complementi alinir
ve daha sonra toplanir. Bu cikarma islemidir.
full adder kullanilir ve bit bit toplanir ya da cikarilir
Cary in
cout
xor
overflow
cary in = 1
two's complement of b[n]


Right Shifter
1 bit shift
2 bit shift
4 bit shift
8 bit shift
16 bit shift
Aritmetik right shiftte isaret biti önemlidir ve dikkate alinarak
kaydirma islemi yapilir.


Left Shifter
[31:0] a
[31:0] b → ilk 5 biti kullanilir (a 32 bit)
[31:0] out
1 shift
2 shift
4 shift
8 shift
16 shift
2:1 mux
1'b0
out[3]
out[2]
out[1]
out[0]


mux 2to1 _ 32bits
32bits
32 bits
slct
output (32 bits)
1 bitlik 2:1 mux kullanilmistir
b
a
output
slct
mux 8to1
2:1 (32 bits) mux kullanilmistir
slct0
slct1
slct2
A
B
C
D
E
F
G
H
Hepsi 32 bitlik inputlar
output 32 bits



2) Verilog Modules and Their Description

1) and_op : **32 tane and operatörü kullanılmıştır. Inputların karşılıklı bitlerine and işlemi gerçekleştirilmiş ve sonuç 32 bitlik bir değişkene her işlem sonucu tek tek atanıp(index- index) output elde edilmiştir.**

2) or_op : : **32 tane or operatörü kullanılmıştır. Inputların karşılıklı bitlerine or işlemi gerçekleştirilmiş ve sonuç 32 bitlik bir değişkene her işlem sonucu tek tek atanıp(index- index) output elde edilmiştir.**

3) xor_op : **32 tane xor operatörü kullanılmıştır. Inputların karşılıklı bitlerine xor işlemi gerçekleştirilmiş ve sonuç 32 bitlik bir değişkene her işlem sonucu tek tek atanıp(index- index) output elde edilmiştir.**

4) nor_op : **32 tane nor operatörü kullanılmıştır. Inputların karşılıklı bitlerine nor işlemi gerçekleştirilmiş ve sonuç 32 bitlik bir değişkene her işlem sonucu tek tek atanıp(index- index) output elde edilmiştir.**

5) half_adder : **1 bitlik iki sayıyı toplar. Half adder da xor ve and operatörleri kullanılmıştır.**

6) full_adder : **1 bitlik iki sayıyı toplar. 2 tane half adder toplamıştır. Most significant biti önemlidir.(işaret biti)**

7) 2to1mux: **verilen bir bitlik iki sayı arasında select bitine göre output verir.**

8) Adder : **Verilen 32 bitlik iki sayıyı op_type bitine göre toplar ya da çıkarır. Eğer çıkarma işlemi ise ikinci sayının her bitini xor ile işleme koyar(sayıyı eksi yapar two's complement işlemi gerçekleştirilir.) ve toplar. overflow durumu kontrol edilmiştir. carry_out ile son carry_in için xor işlemi uygulanmıştır Sonuç 1 ise overflow olmuştur.**

9) Rightshifter : **32 bitlik bir sayıyı aritmetic right shift yapar.1,2,4,8,16 shift olmak üzere 5 bölümden oluşur.(her bölümde 32 adet 2*1 mux bulunur.) Shift işlemi kadar most significant bit out değişkenine eklenir. Daha sonra most signifant**

bitinden itibaren her bit bir kez kayacak şekilde bir out da bir sonraki yere eklenir. b bitinin ilk 5 biti kullanılır(çünkü a 32 bittir). eğer bit bir ise shift yapılır değilse aynen temp değişkene atanır. Olay örgüsü 4 bitlik sayı üzerinden anlatmak gerekirse 1000 olduğunu düşünelim 1 shift için most significant bit aşağı aynı şekilde geçer daha sonra most significant bit mux ta bir önceki mux un inputuna bağlanır. Daha sonra diğer bitlerde bu şekilde devam eder ve sonuç 1100 olarak elde edilir

10) Leftshifter : **32 bitlik bir sayıyı left shift yapar.1,2,4,8,16 shift olmak üzere 5 bölümden oluşur.(her bölümde 32 adet 2*1 mux bulunur.) Least signifant bitinden itibaren her bit bir kez kayacak şekilde bir out da bir sonraki yere eklenir. b bitinin ilk 5 biti kullanılır(çünkü a 32 bittir). eğer bit bir ise shift yapılır değilse aynen temp değişkene atanır. shift işlemi kadar başa sıfır eklenir(least siginificant bit) Olay örgüsü 4 bitlik sayı üzerinden anlatmak gerekirse 0001 olduğunu düşünelim; 1 shift içinleast significant bit mux ta bir sonraki mux un inputuna bağlanır. Daha sonra diğer bitlerde bu şekilde devam eder ve sonuç 0010 olarak elde edilir**

11) mux2to1_32bits : **32 bitlik iki sayı ile and işlemi yapılır ve sonuç result değişkenine yazılır.**

12) mux8to1 : **32 bitlik 8 input alır ve 3 bitlik slct bitine göre result değişkeni ile output u verir. 32 bitlik input için 2to1mux_32bits module kullanılmıştır.**

13) alu32 : **Input A,B(32 bits), slct(3 bits) Output result(32bits) ,overflow,zero,carry_out(1 bits) 8*1 ile istenilen işlemi seçer. zero durumu için işlemin sonucundaki bitler([31:0] output) or işlemi yapılmıştır. sonuç un not işlemi uygulanır. Eğer zero 0 ise işlemin sonucu 0 dır. Ayrıca işlemler aynı anda yapıldığı için seçilen işlemde overflow olmamış olabilir. Bunun için işlem toplama ya da çıkarma işlemi (010,100 selects) ise overflow bitinin kendisi 2*1 mux ile seçilir aksi takdirde overflow 0 kabul edilir.**

14) alu32_testbench **: Her operatör için 3 test toplamda 24 test işlemi bulunmaktadır.**

3) Modelsim Simulation Results

```
VSIM 5> step -current
# A and B OPERATION
#
#  Input1-->11111111111111111111111111111111
#  Input2-->00000000000000000000000000000000
#
#  result-->00000000000000000000000000000000
#  overflow-->0
#  zero-->1
#
#
#  Input1-->10101010101010101010101010101010
#  Input2-->01010101010101010101010101010101
#
#  result-->00000000000000000000000000000000
#  overflow-->0
#  zero-->1
#
#
#  Input1-->00000000000011111111111111111111
#  Input2-->00000000000011111111111111111111
#
#  result-->00000000000011111111111111111111
#  overflow-->0
#  zero-->0
#
#
# A or B OPERATION
#
#  Input1-->11111111111111111111111111111111
#  Input2-->00000000000000000000000000000000
#
#  result-->11111111111111111111111111111111
#  overflow-->0
#  zero-->0
#
#
#  Input1-->10101010101010101010101010101010
#  Input2-->01010101010101010101010101010101
#
#  result-->11111111111111111111111111111111
#  overflow-->0
#  zero-->0
#
#
#  Input1-->00011100011100011100011100011100
#  Input2-->00000000000000000000000000000001
#
#  result-->00011100011100011100011100011101
#  overflow-->0
#  zero-->0
#
```

```
#
# A+B OPERATION
#
#  Input1-->00000000000000000000000000001111
#  Input2-->00000000000000000000000000001111
#
#  result-->00000000000000000000000000011110
#  overflow-->0
#  zero-->0
#
#
#  Input1-->10000000000000000000000000000000
#  Input2-->10000000000000000000000000000000
#
#  result-->00000000000000000000000000000000
#  overflow-->1
#  zero-->1
#
#
#  Input1-->01010101010101010101010101010101
#  Input2-->01010101010101010101010101010101
#
#  result-->10101010101010101010101010101010
#  overflow-->1
#  zero-->0
#
#
# A xor B OPERATION
#
#  Input1-->11111111111111111111111111111111
#  Input2-->00000000000000000000000000000000
#
#  result-->11111111111111111111111111111111
#  overflow-->0
#  zero-->0
#
#
#  Input1-->10101010101010101010101010101010
#  Input2-->01010101010101010101010101010101
#
#  result-->11111111111111111111111111111111
#  overflow-->0
#  zero-->0
#
#
#  Input1-->10101010101010101010101010101010
#  Input2-->00000000000000000000000000000000
#
#  result-->10101010101010101010101010101010
#  overflow-->0
#  zero-->0
#
```

```
#
# A-B OPERATION
#
#  Input1-->10101010101010101010101010101010
#  Input2-->10101010101010101010101010101010
#
#  result-->00000000000000000000000000000000
#  overflow-->0
#  zero-->1
#
#
#  Input1-->00000000000000000000000010000000
#  Input2-->00000000000000000000000001010101
#
#  result-->00000000000000000000000000101011
#  overflow-->0
#  zero-->0
#
#
#  Input1-->01111111111111111111111111111111
#  Input2-->11000000000000000000000000000000
#
#  result-->10111111111111111111111111111111
#  overflow-->1
#  zero-->0
#
#
# A>>B OPERATION
#
#  Input1-->10000000000000000000000000000000
#  Input2-->00000000000000000000000000000011
#
#  result-->11110000000000000000000000000000
#  overflow-->0
#  zero-->0
#
#
#  Input1-->10000000000000001111111111111111
#  Input2-->00000000000000000000000000010000
#
#  result-->11111111111111110000000000000000
#  overflow-->0
#  zero-->0
#
#
#  Input1-->00000000000000000000000000000111
#  Input2-->00000000000000000000000000000001
#
#  result-->00000000000000000000000000000011
#  overflow-->0
#  zero-->0
#
```

```
#
# A<<B OPERATION
#
#  Input1-->11111111111111111111111111111111
#  Input2-->00000000000000000000000000011111
#
#  result-->11111111111111110000000000000000
#  overflow-->0
#  zero-->0
#
#
#  Input1-->00000000000000000000000000000111
#  Input2-->00000000000000000000000000000101
#
#  result-->00000000000000000000000011100000
#  overflow-->0
#  zero-->0
#
#
#  Input1-->11110000111100001111000011110000
#  Input2-->00000000000000000000000000001100
#
#  result-->11110000111100001000000000000000
#  overflow-->0
#  zero-->0
#
#
# A nor B OPERATION
#
#  Input1-->11111111111111111111111111111111
#  Input2-->00000000000000000000000000000000
#
#  result-->00000000000000000000000000000000
#  overflow-->0
#  zero-->1
#
#
#  Input1-->10101010101010101010101010101010
#  Input2-->01010101010101010101010101010101
#
#  result-->00000000000000000000000000000000
#  overflow-->0
#  zero-->1
#
#
#  Input1-->11111111111111110000000000000000
#  Input2-->00000000000000000000000000000000
#
#  result-->00000000000000001111111111111111
#  overflow-->0
#  zero-->0
#
```