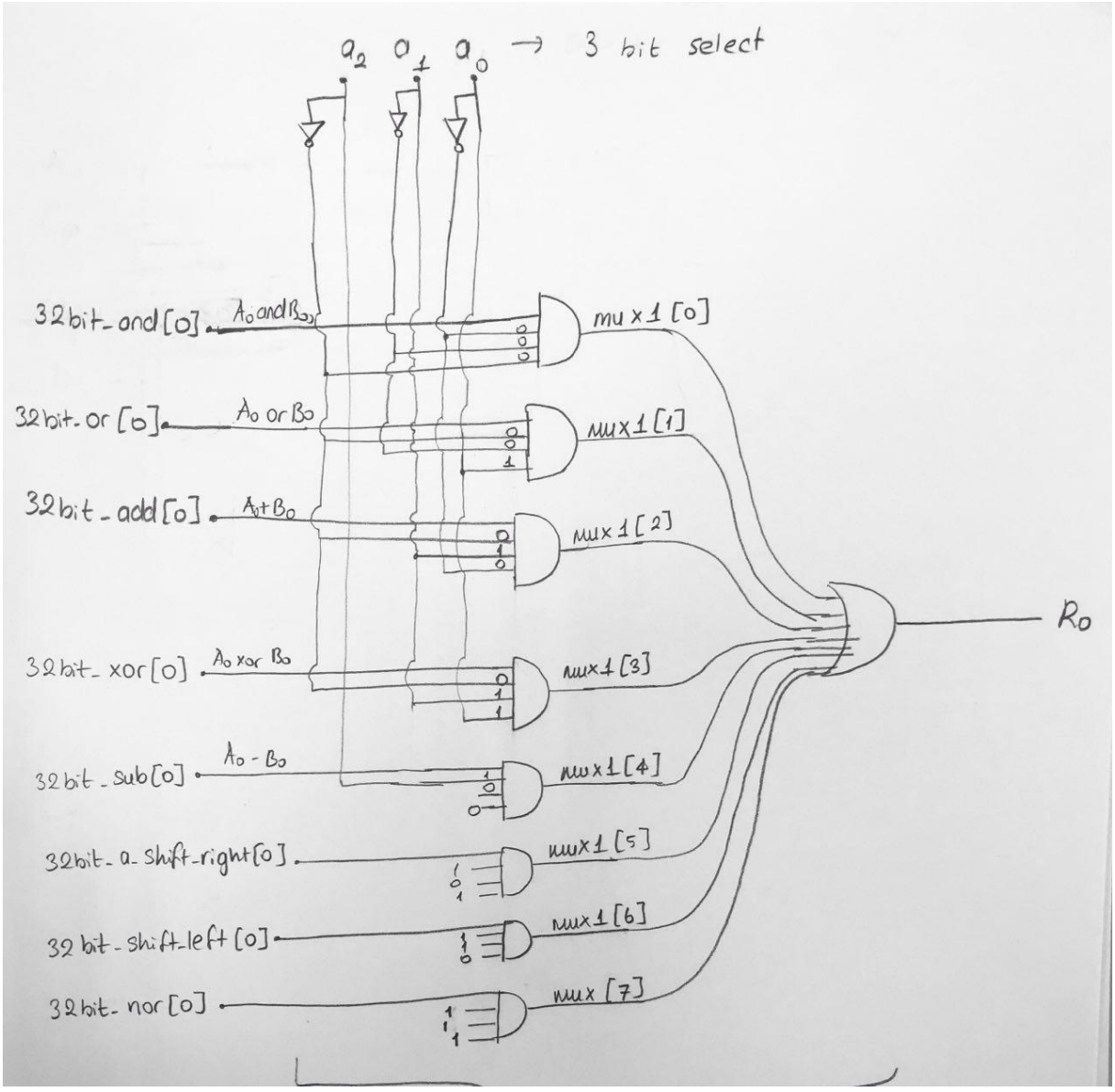
CSE 331 Computer Organization

Homework 2 Rapor

Alu32.v için genel tasarım şu şekildedir:



Görülen şekil mux1 isimli 8:1 bir muxtur oluşacak outputun 0. bitini verir. 32 bit output için alu32.v modülü içinde bu şekilde 32 adet mux kullanılmıştır. İnputlar her bir işlemi 32 bit olarak ayrı bir modülde yaparak 32 bit sonuç üretir. Bu sonuçların her birinin 0. Bitleri mux ile seçilerek istenen işlemin outputunun 0. Bitini oluşturmuş olur.

Proje yapılırken, oluşturulan her bir modül için testbenchler yazılmıştır. En son alu32 için ayrıca her bir durumu test edip basan testbench oluşturulmuştur.

Alu32 testbench:

```
# select=000
#   A=11000000001111111100000000000000,
#   B=11000000101101000000000000011111,
# out=11000000001101000000000000000000,
# overflow=0,  cout= 1
#
# select=000
#   A=11000000000000000000000000000000,
#   B=11000000000000000000000000011111,
# out=11000000000000000000000000000000,
# overflow=0,  cout= 1
#
# select=000
#   A=11000001010101010000000000000000,
#   B=11000001111100000000000000011111,
# out=11000001010000000000000000000000,
# overflow=0,  cout= 1
#
```

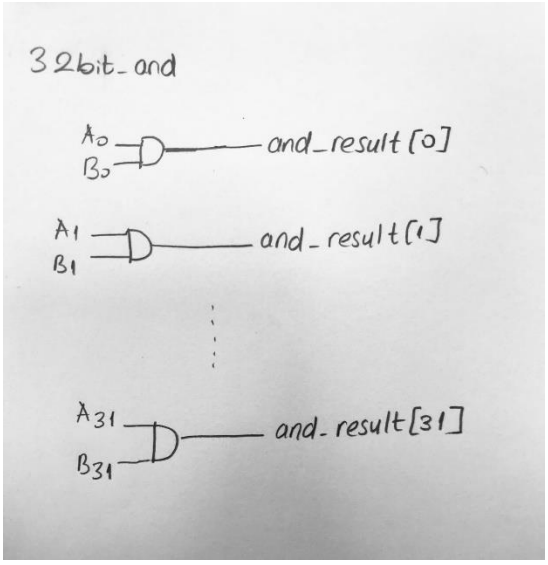
```
# select=001
#   A=11000101010101000000000011111111,
#   B=11110001111100000000000000000000,
# out=11110101111110100000000011111111,
# overflow=0,  cout= 1
#
# select=001
#   A=11000000000000101010101001111111,
#   B=11110001111100111111111100000000,
# out=11110001111100111111111101001111,
# overflow=0,  cout= 1
#
# select=001
#   A=11000000001111111110100011111111,
#   B=11110001111100000000000000000000,
# out=11110001111111111111010000111111,
# overflow=0,  cout= 1
#
```

```
#
# select=010
#   A=11000111111111100000000011101010,
#   B=11110001111100000000000000000000,
# out=10111001110111000000000011101010,
# overflow=0,  cout= 1
#
# select=010
#   A=11000000000101010000000011101010,
#   B=11110001111100000000000000000000,
# out=10110001111110101000000011101010,
# overflow=0,  cout= 1
#
# select=010
#   A=11000000000000000000000011101010,
#   B=11110001111000011111101010101000,
# out=10110001111000011111101110010010,
# overflow=0,  cout= 1
#
```

```
# select=011
#   A=11000000000000000000000011111111,
#   B=11110001111000000000000000000000,
# out=00110001111000000000000011111111,
# overflow=0,  cout= 1
#
# select=011
#   A=11000011111111111111111100011111111,
#   B=11110001111000000000000000000000000,
# out=00110010000111111111111100011111111,
# overflow=0,  cout= 1
#
# select=011
#   A=11000000010101010000000011111111,
#   B=11110001111000000000000000000000000,
# out=00110001101101010000000011111111,
# overflow=0,  cout= 1
#
```

```
# select=100
#   A=11111111000000000000000011111111,
#   B=01110001111000000000000000111110,
# out=10001101001000000000000011000001,
# overflow=1,  cout= 1
#
# select=100
#   A=11111111010101010100000011111111,
#   B=01110001111000000000000000111110,
# out=10001101011101010100000011000001,
# overflow=1,  cout= 1
#
# select=100
#   A=11111111000000001111100001111111,
#   B=01110001111000010101000000111110,
# out=10001101001000001010000011000001,
# overflow=1,  cout= 1
#
# select=101
#   A=11000000000000000000000011111111,
#   B=00000000000000000000000000000011,
# out=11111000000000000000000000011111,
# overflow=1,  cout= 0
#
# select=101
#   A=11000011111111111110000011111111,
#   B=00000000000000000000000000000011,
# out=11111000011111111111110000011111,
# overflow=1,  cout= 0
#
# select=101
#   A=11000000000000000000000011111111,
#   B=00000000000000000000001010101010,
# out=11111111111110000000000000000000,
# overflow=1,  cout= 0
#
```

```
# select=110
#   A=11000000000000000000000011111111,
#   B=00000000001111110000000000001001,
# out=00000000000000001111111000000000,
# overflow=1,  cout= 0
#
# select=110
#   A=11000000000000000000000011111111,
#   B=00000000000000000000000000001000,
# out=00000000000000001111111100000000,
# overflow=1,  cout= 0
#
# select=110
#   A=11000000000000000000000011111111,
#   B=00000000000000000000000000001001,
# out=00000000000000001111111000000000,
# overflow=1,  cout= 0
#
# select=111
#   A=11000000000000000000000011111111,
#   B=11110001111100000010111100011101,
# out=00001110000111111010000100000000,
# overflow=0,  cout= 1
#
# select=111
#   A=11000000000000000000000011111111,
#   B=11110001111100000000000000011111,
# out=00001110000111111111111111100000000,
# overflow=0,  cout= 1
#
# select=111
#   A=11000000101010100000000011111111,
#   B=11110001111100000000000000000000,
# out=00001110000101011111111100000000,
# overflow=0,  cout= 1
#
```



Örneğin; 32bit_and modülü içerisinde her bir bit kendi içinde and'lenerek 32 bit and_result sonucu hesaplanır ve alu'ya verilir. Alu, bu sonucun 0. Bitini 8:1'lik muxun ilk girişine verir. 1. Bitini ise, 2. Muxun ilk girişine verir. 2. Biti 3. Muxun ilk girişine verir ve bu şekilde 32 bit için aynı işlem devam eder. Muxun ilk girişine verilme sebebi A and B işlemi için verilecek selectin 000 olmasıdır.

```
# time =  0,
# A =10100101101000101111010001101001O,
# B=11111111111111111111111111111111,
# and_result=10100101101000101111010001101001O
```

=> AND

32bit_or, 32bit_xor ve 32bit_nor modüllerinin içi, 32bit_and modülüyle hemen hemen aynıdır. And kapısının yerine sırasıyla; or, xor ve nor kapıları kullanılarak tasarlanmıştır. Alu'da ise selectlerine bağlı olarak and işlemindeki gibi bitlerine ayrılarak gereken mux girişlerine verilirler.

```
# time =  0,
# A =10100101101000101111010001101001O,
# B=11111111111111000000000000000000,
# or_result=11111111111111101111010001101001O
```

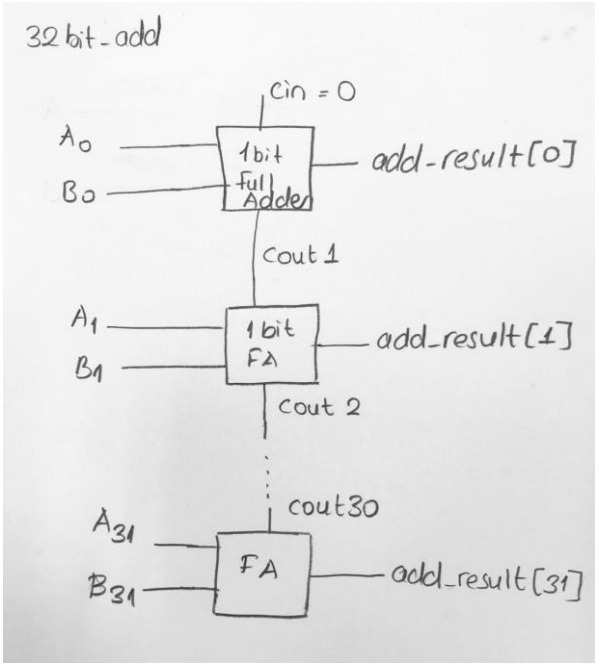
=> OR

```
# time =  0,
# A =10100101101000101111010001101001O,
# B=11111111111111111111111111111111,
# xor_result=01011010010111010001011100101101
```

=>XOR

```
# time =  0,
# A =10100101101000101111010001101001O,
# B=11111111111111000000000000000000,
#  nor_result=00000000000000010001011100101101
```

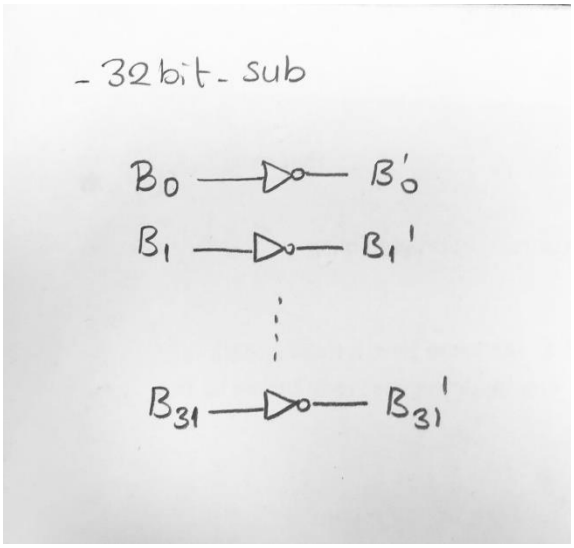
=>NOR



32bit_add modülü için ps1'de anşatılmış olan 4 bit adder yardımcı modülleri kullanılmıştır. Bunlar: half_adder ve full_adder modülleridir. 32bit_add içinde ise A ve B sayılarının her bir biti kendi arasında full_adder ile toplanır. İki sayının toplamı 32 biti aştığı takdirde cout biti 1 olur ve alu32'ye gönderilir output ile de basılır.

```
# select=010
#   A=11000000000000000000000011101010,
#   B=11110001111000000000000000000000,
# out=10110001111000000000000011101010,
# overflow=0,  cout= 1
```

```
# time =  0,
# A =10100101101000101111010001101001O,
# B=00000000000000000000000000000111,
# add_result=10100101101000101111010001101l001
```



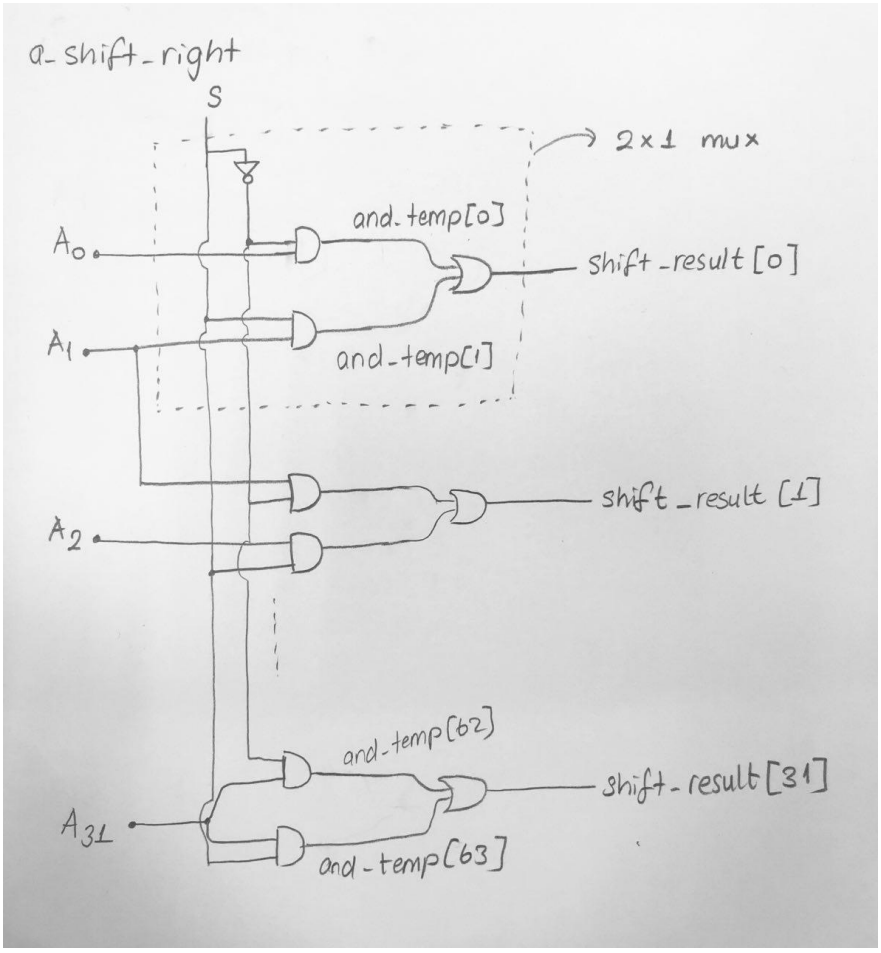
_32bit_sub modülünde çıkarma işlemi

A-B= A+ (-B) ve -B = B'nin 2's complimentı +1 olduğundan önce B'nin her bitinin tersi not gate'i ile bulunur. Sonrasında 32bit_adder modulu kullanılarak 1 ile toplanır ve sonucu da A inputu ile toplanılarak çıkarma işlemi yapılmış olunur. Sayıların işaret biti farklı olduğu durumlar için overflow biti tasarıma uygun olarak yapıldı fakat test edilmedi

```
# select=100
#   A=11111111000000000000000011111111,
#   B=01110001111000000000000000111110,
# out=10001101001000000000000011000001,
# overflow=1,  cout= 1
#
```

```
# select=100
#   A=11000000000000000000000011111111,
#   B=11110001111000000000000000111110,
# out=11001110001000000000000011000001,
# overflow=0,  cout= 1
#
```

```
# time =  0,
#  A=11110000000000000000000000000111,
#  B=00000000000000000000000000000011,
# sub_result=11110000000000000000000000000100
# overflow=1
```

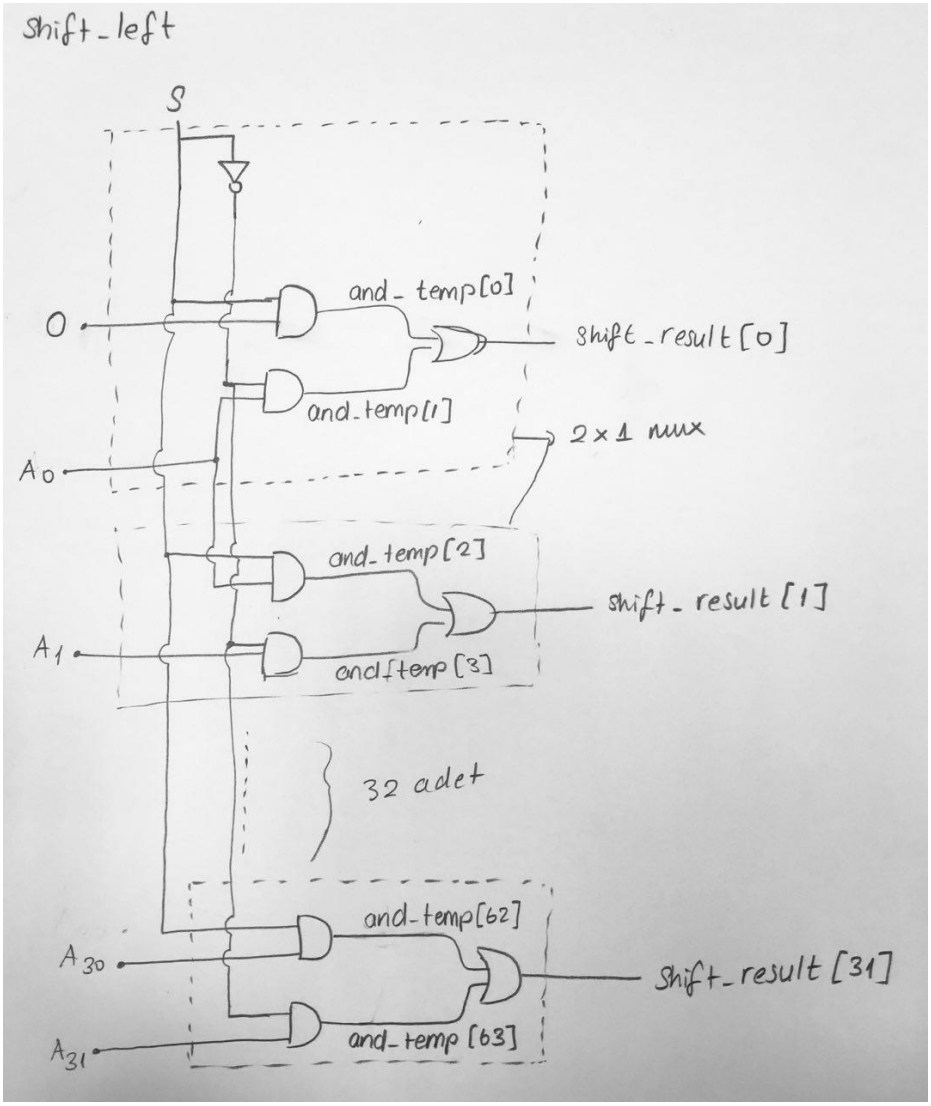


a_shift_right: Aritmetik shift right işlemininin 1. Seviyesini yapan modüldür. 1 adet select alır. Aslında burada son mux'u yapmayıp A31 i direk çıkarabilirdik çünkü her halükârda A31 çıkacaktı.

32 bit B sayısı için 2^n=32 => n=5 seviye olmalıdır. Select 1 verildiğinde 1 shift eder. 2 shift etmek için ise bu modülü shift edilmiş sonuç gönderilerek select 1 verilir ve tekrar çağırılır. Yani alu32 modülünün içinde bu işlem her seferinde 31 defa tekrarlanır. Select ise B sayısına bağlıdır çünkü B kadar shift edilmesi isteniyor.

```
# select=101
#    A=11000000000000000000000011111111,
#    B=00000000000000000000000000000011,
# out=11111100000000000000000000011111,
```

```
# time =  0,
# A =11000000000000000000000000000011,
# S=1
# asr_result=11100000000000000000000000000001
```



left_shift: Logical left shift işlemi yapan modüldür. Right shiftte shift işlemi sonrası oluşan yerlere 32. Bitteki sayı geliyordu. Burada ise farklı olarak boşluklar 1. bitten itibaren oluşacağı için buralara 0 gelmesi gerek. Bu modül 1 select alır ve 1. Seviyede shift yapar. Bu modulun alu32 içerisinde 32 kez çağırılmasıyla 32 kez shift yapılabilinmişmiş olunur.

```
# Shifting left: time =  0,
# A =11000000000000000000000000000011,
# S=1
# out=10000000000000000000000000000110
```

```
# select=110
#   A=11000000000000000000000011111111,
#   B=00000000000000000000000000000011,
# out=00000000000000000000011111111000,
```

Zeynep Hamza

151044011