Gebze Technical University
Computer Engineering

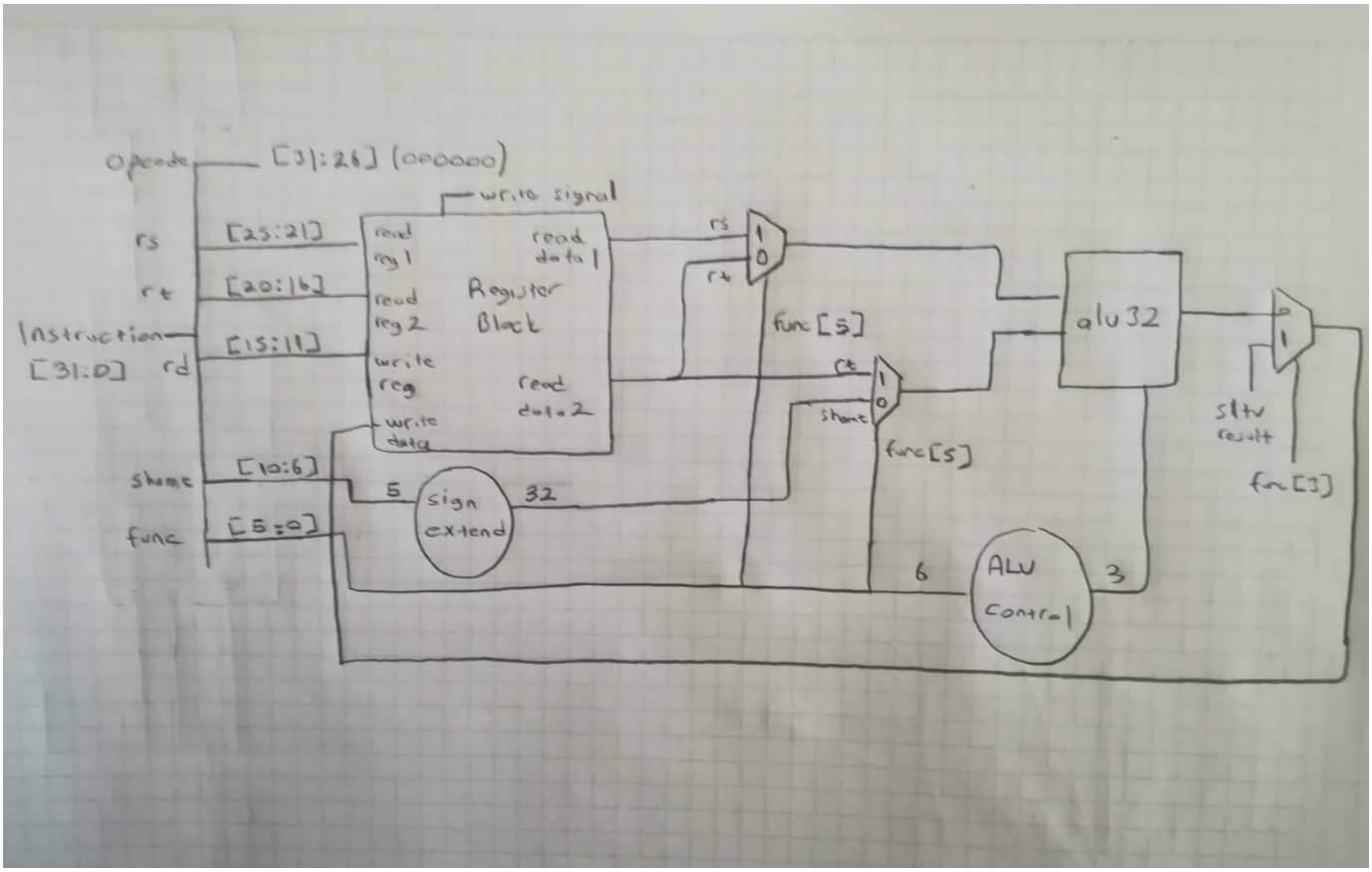
CSE 331 - 2018

HOMEWORK 3 REPORT

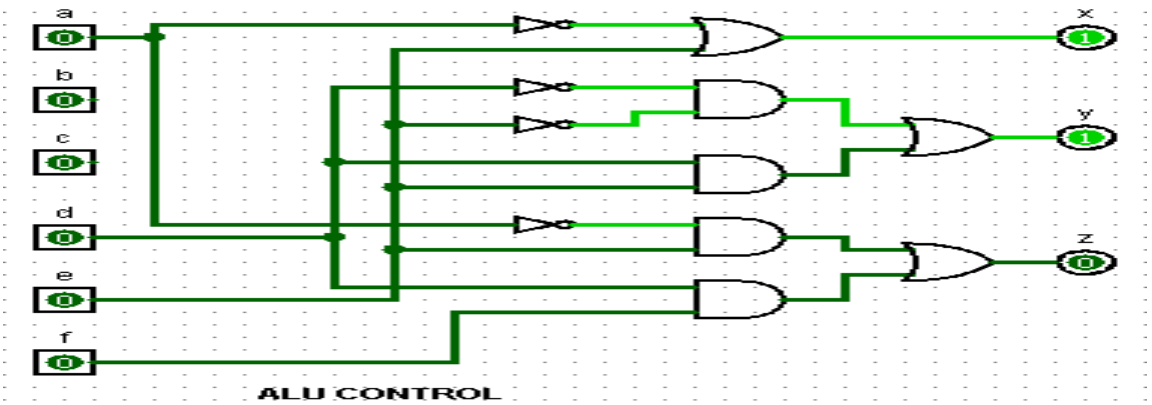
SERKAN SORMAN
151044057

Course Assistant: Fatma Nur Esirci

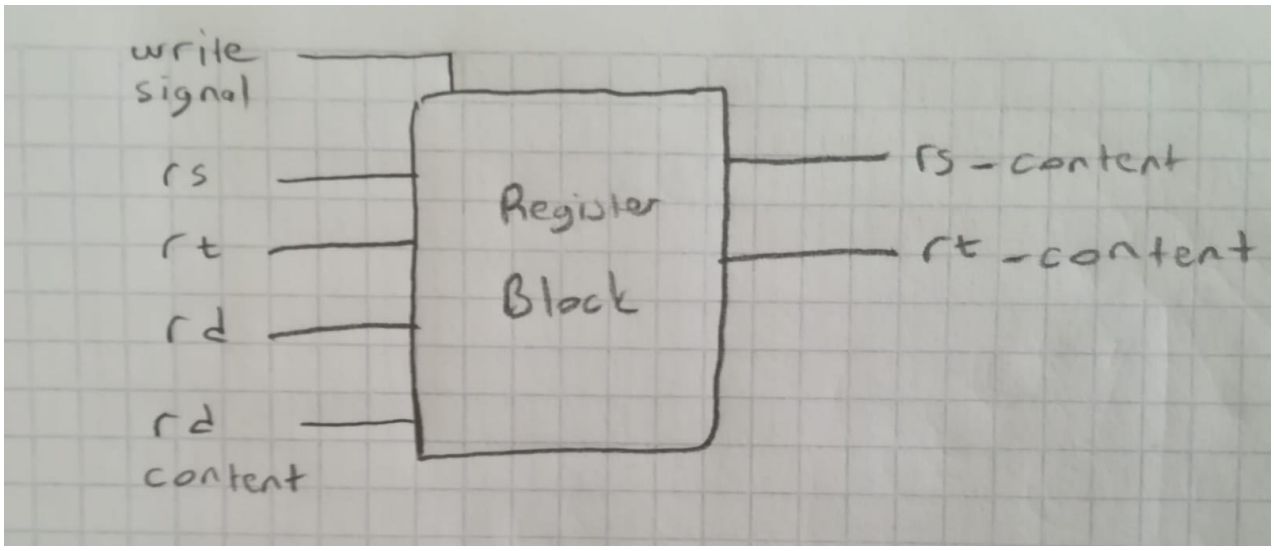
Module mips_core: Input olarak 32 bit bir R-type instruction alır ve bu instructionu opcode, rs, rt, rd, shamt ve function code bileşenlerine ayırır. Register bloğuna rs, rt, rd ve write sinyali 0 olarak yollanır. 5 bitlik shamt ise sign extende yollanarak 32 bite extend edilir. Aynı zamanda 6 bitlik function code alu controle yollanarak 3 bitlik alu opcode elde edilir. Daha sonra register bloğundan elde edilen rs ve rt contentleri instructionun shift olma durumunu belirlemek üzere muxa sokulur seçici olarak function codun msb si verilir (function codun msb si 0 ise instruction shifttir) ve eğer instruction shift ise rs yerine rt, rt yerine ise shamt aluya verilir. Alu controlden elde edilen 3 bitlik opcode aluya verilerek gerekli işlem yapılır ve sonuç hesaplanır. Alunun çıktısı sltu instructionın sonucu (32 bit 0 ya da 32 bit 1) ile muxa sokulur ve tekrar register bloğuna dönülerek write dataya verilir ve write signali 1 olarak yollanır. Sonuç olarak instructiondaki işlem sonucu dosyadaki rd registerına yazılır.



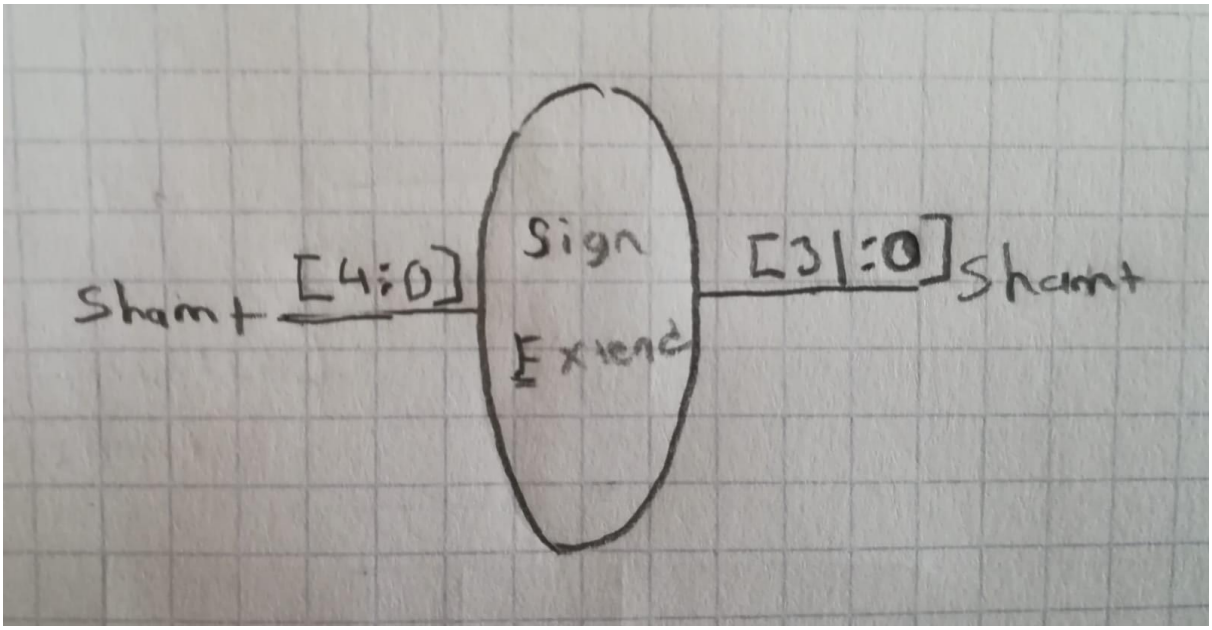
Module alu_control: 6 bitlik opcode u alır ve alu için 3 bitlik bir koda dönüştürür.



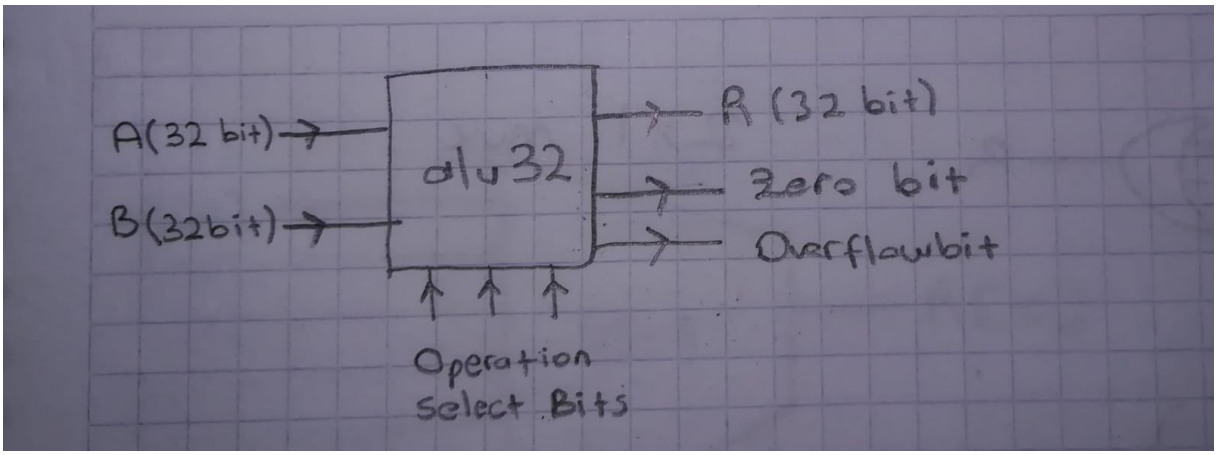
Module register_block: Input olarak rs, rt, rd, yazılacak resultu ve rd registerına yazma işlemini belirten write signali alır. Dosyadan rs ve rt nin adreslerine göre contentlerini okur ve output olarak verir. Write signali 1 iken öncelikle rd registerının r0 olmadığı kontrol edilir ve dosyadaki rd registerına result yazılır.



Module sign_extend: Input olarak 5 bitlik shamt ı alır ve başına 0 ekleyerek 32 bite extend eder.



Module alu32: Input olarak iki adet 32 bit sayı ve hangi işlemin yapılacağını belirten 3 adet sinyal biti alır. Gelen 3 sinyal bitine göre seçilen işlemi yapar. Output olarak sonucu verir. (Bu projede overflow ve zero bitine ihtiyaç olmadığı için kaldırılmıştır) . Ayrıca SRL işleminin yapılabilmesi için alu içine mux konulup select biti 0 olarak ayarlanmıştır. Böylece ShiftR modülü kullanılarak select bitine göre SRL ve SRA işlemleri yapılabilecektir.

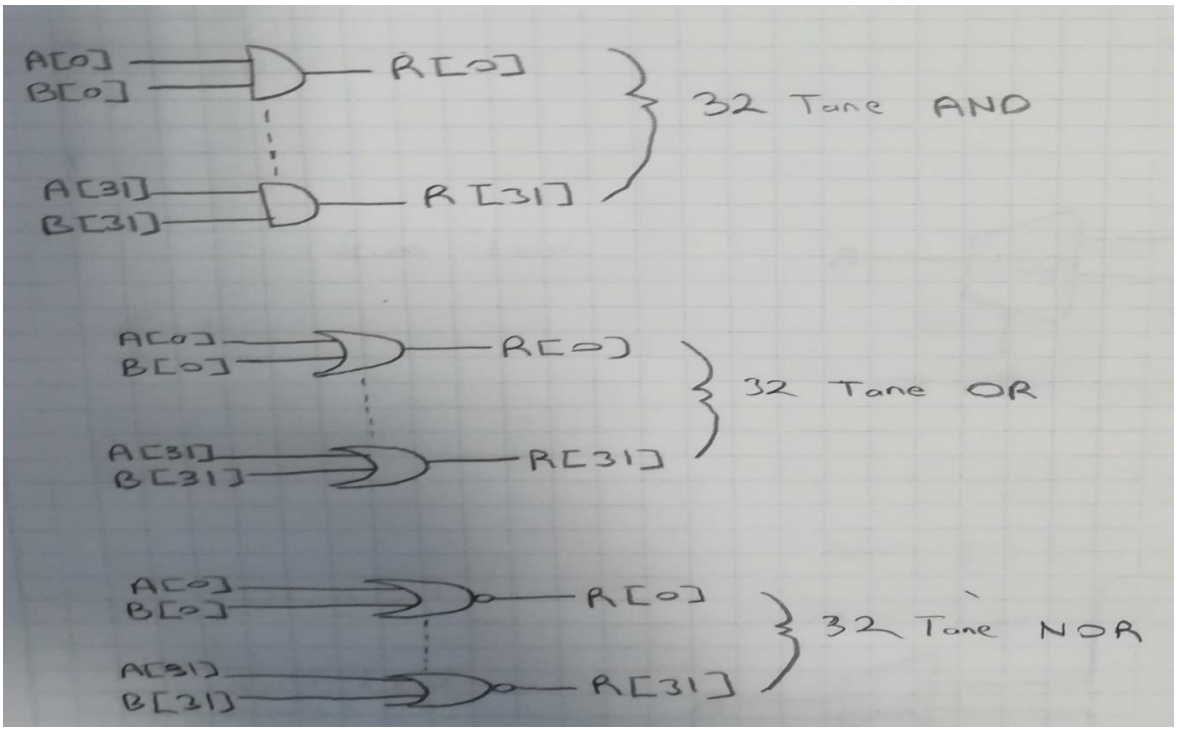


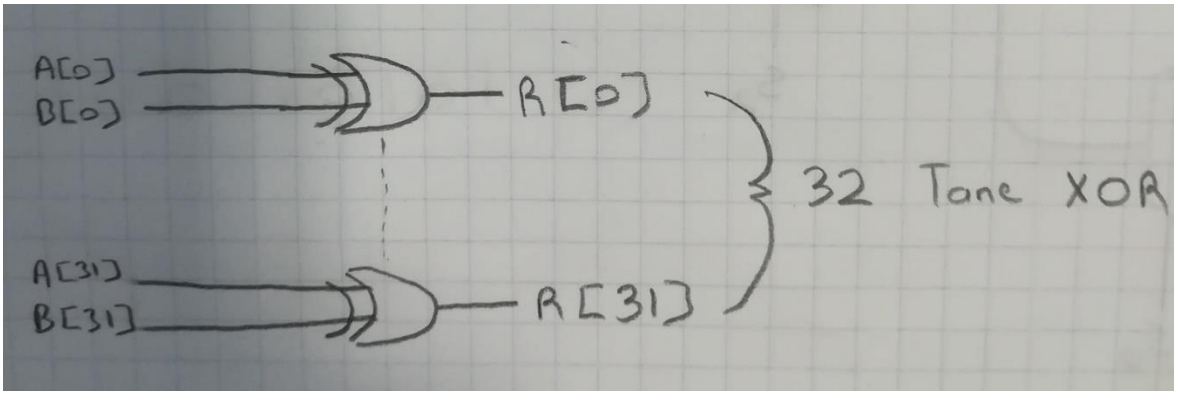
Module andop: Input olarak iki adet 32 bit sayı alır. Output olarak iki sayının tüm bitlerinin andlenmiş halini verir.

Module orop: Input olarak iki adet 32 bit sayı alır. Output olarak iki sayının tüm bitlerinin orlanmış halini verir.

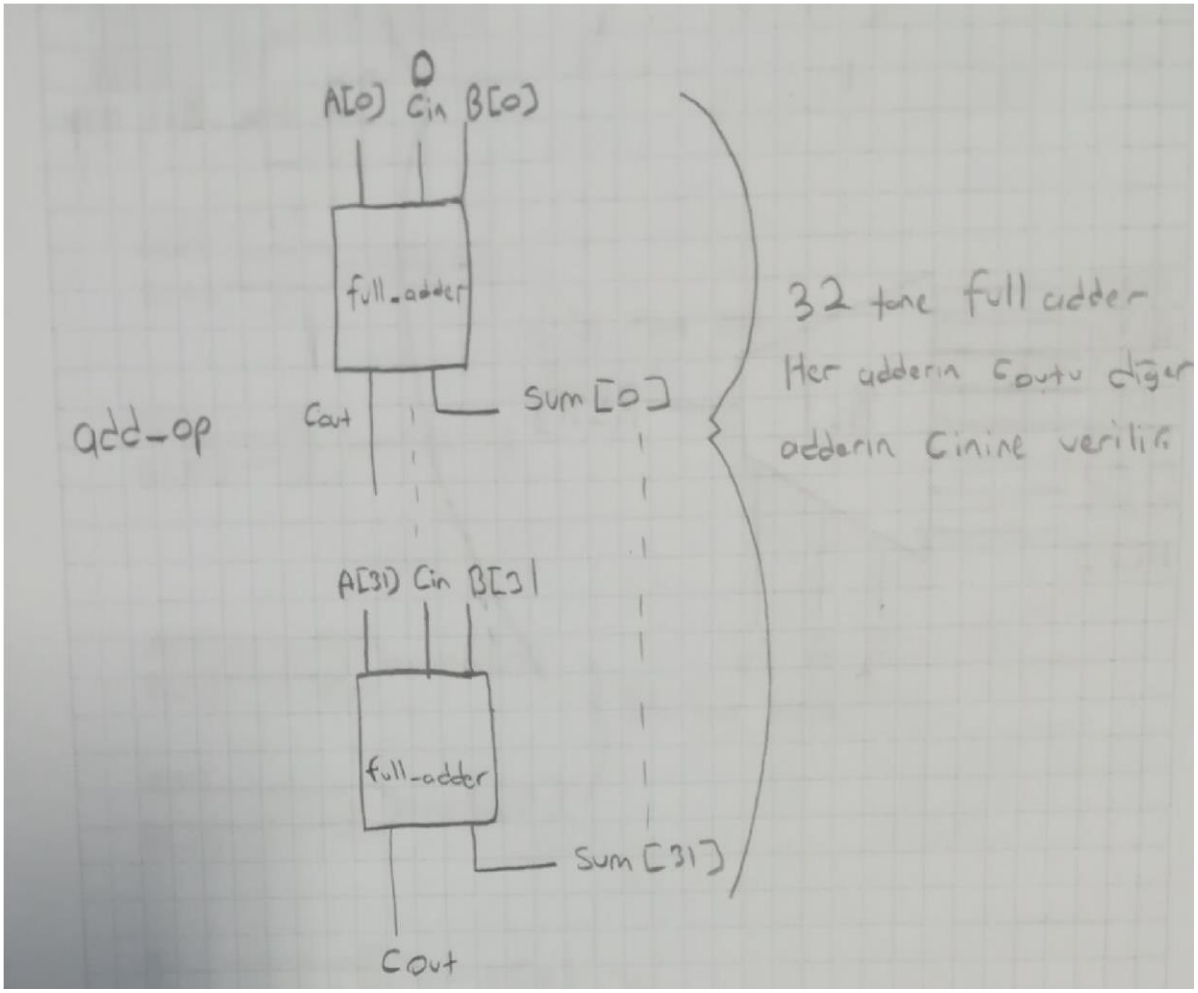
Module xorp: Input olarak iki adet 32 bit sayı alır. Output olarak iki sayının tüm bitlerinin xorlanmış halini verir.

Module norop: Input olarak iki adet 32 bit sayı alır. Output olarak iki sayının tüm bitlerinin norlanmış halini verir.

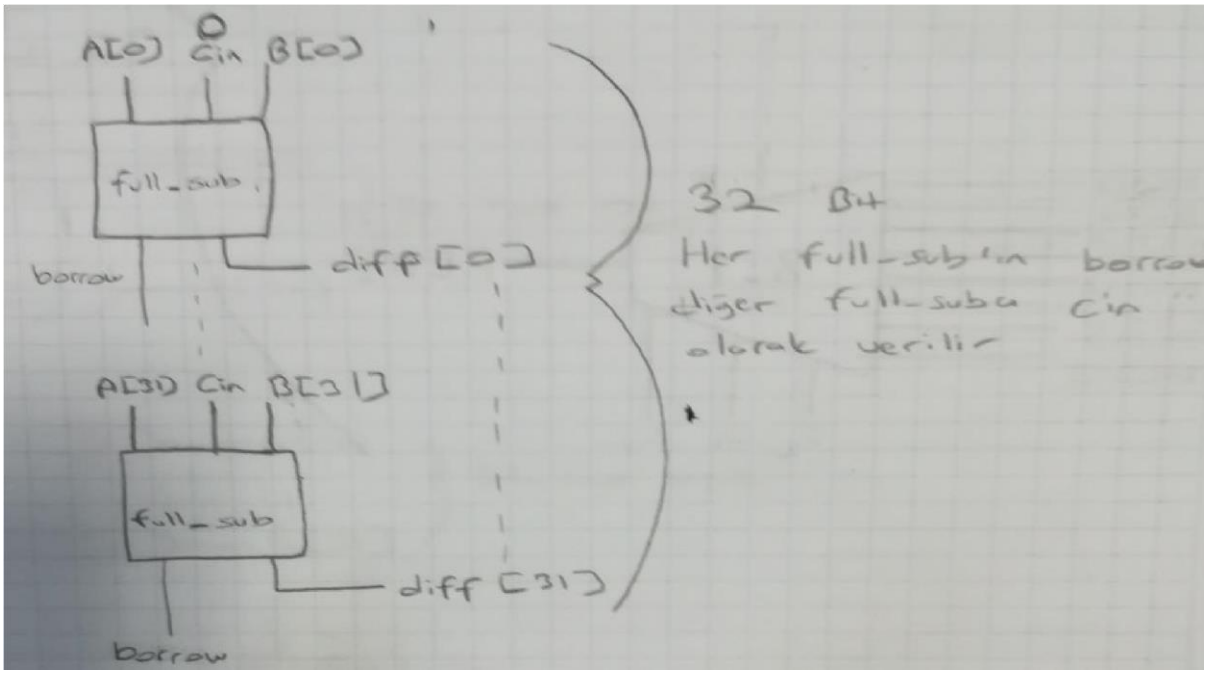




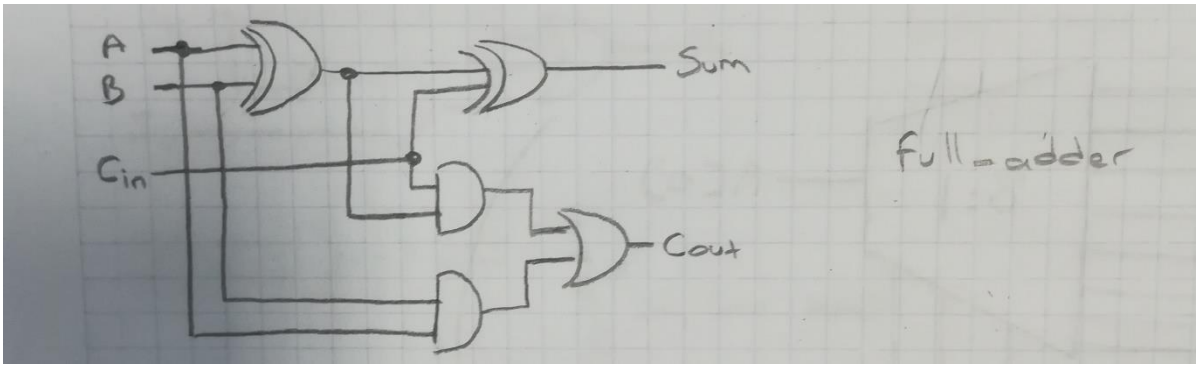
Module addop: Input olarak iki adet 32 bit sayı alır ve bu sayılar birer bit olarak full_addera yollanır. Output olarak carry out ve iki sayının toplamını verir.



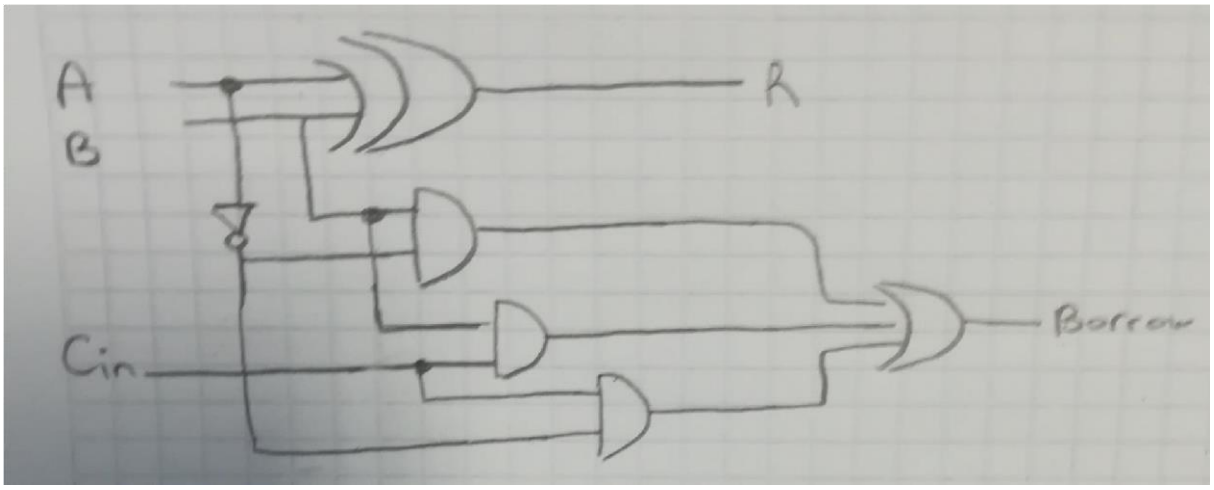
Module subop: Input olarak iki adet 32 bit sayı alır ve bu sayılar birer bit olarak full_suba yollanır. Output olarak borrow ve iki sayının farkını verir.



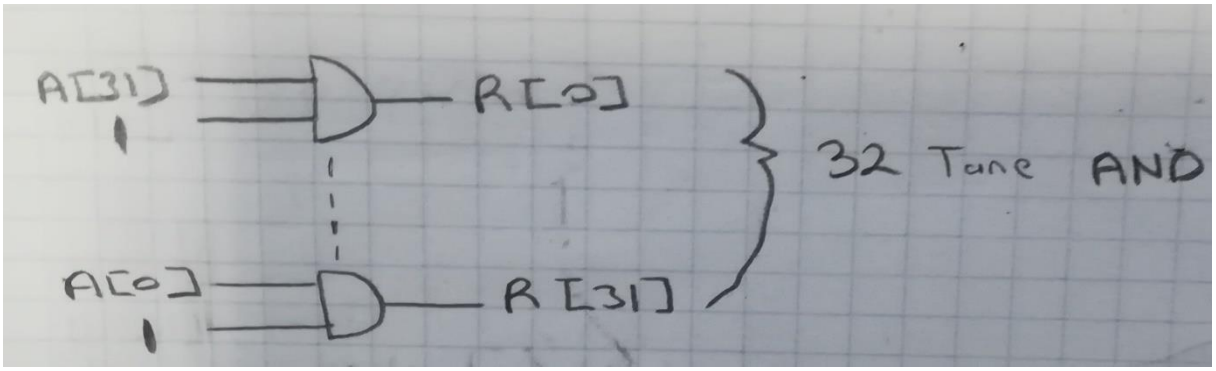
Module full_adder: Input olarak iki adet 1 bit sayı ve 1 bit carry in alır. Output olarak carry out ve iki bitin toplamını verir.



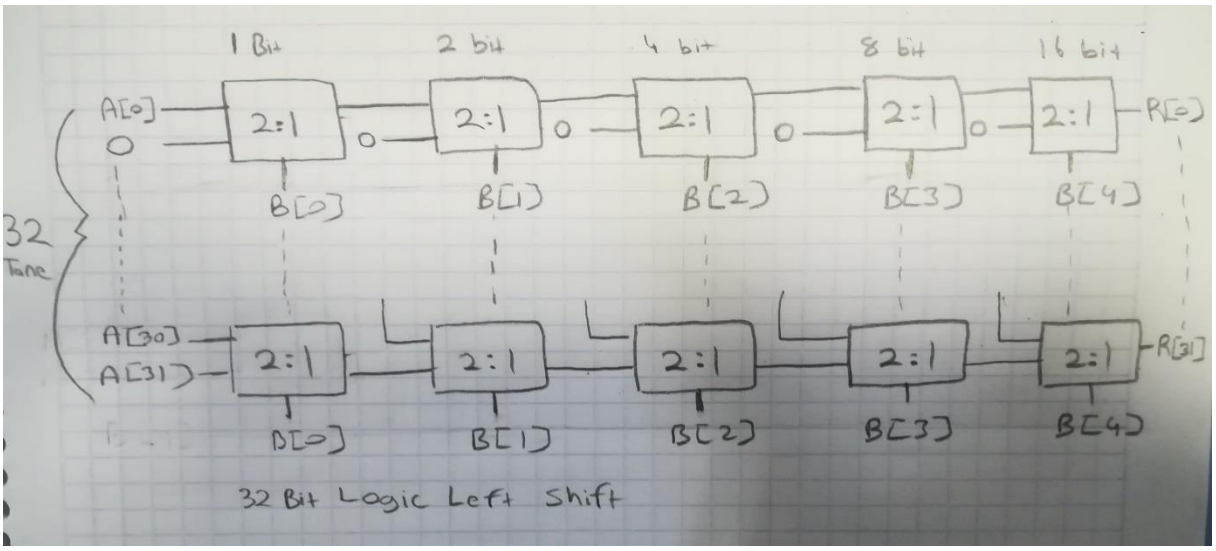
Module full_sub: Input olarak iki adet 1 bit sayı ve 1 bit carry in alır. Output olarak borrow ve iki bitin farkını verir.



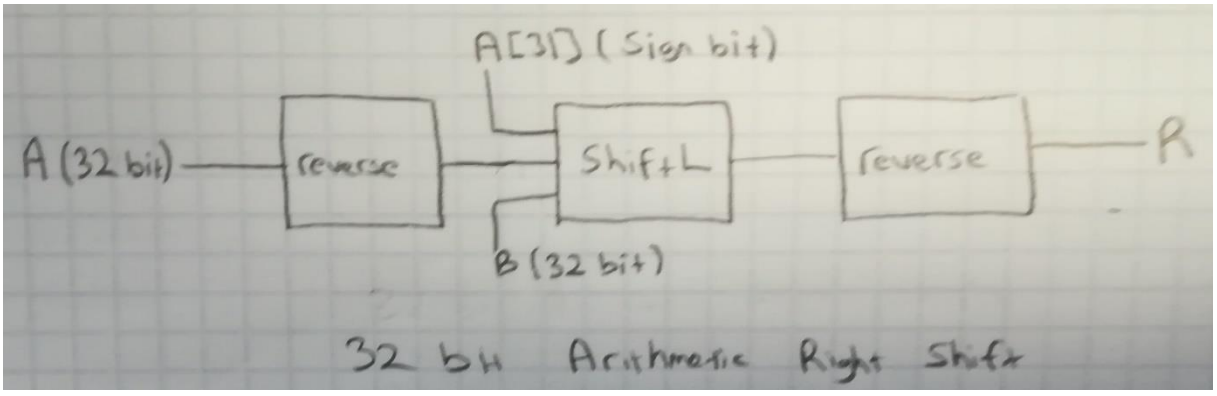
Module reverse: Input olarak verilen 32 bit sayının ters çevrilmiş halini output olarak verir.



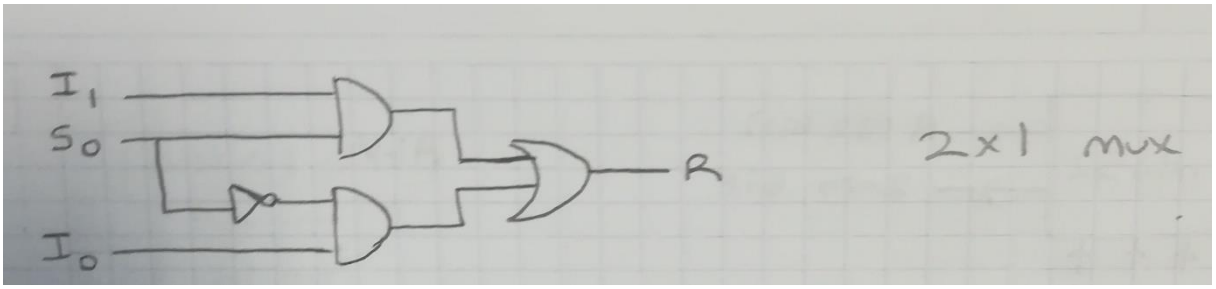
Module ShiftL: Input olarak Logic Left Shift edilecek 32 bit bir sayı, Shift edilecek basamak sayısını belirten ikinci bir 32 bit sayı ve Shift işleminde kaydırılan bitlerin yerine konacak biti belirten bit alınır (Logic left shift yapıldığında 0 yollanır). Output olarak verilen 32 bit sayının Left Logic Shift edilmiş hali verilir.



Module ShiftR: Input olarak Arithmetic Right Shift edilecek 32 bit bir sayı, Shift edilecek basamak sayısını belirten ikinci bir 32 bit sayı alır. Verilen sayı reverse edilip Logic Left Shift yapılır (Kaydırılan bitlerin yerine konması için sign biti yollanır). Ardından tekrar reverse edilir ve Output olarak verilir.

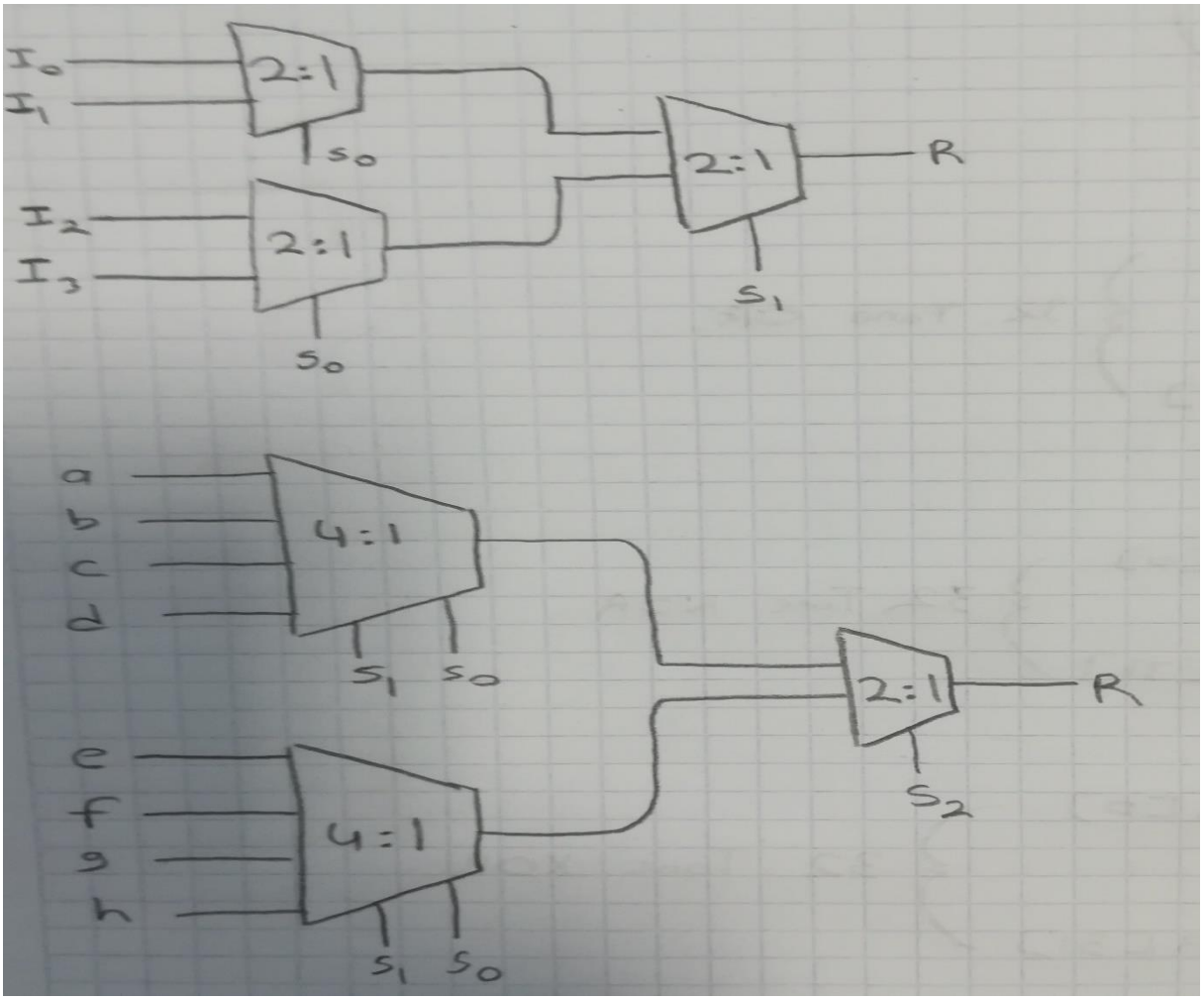


Module mux2x1: İki adet birer bit input ve bir seçici alarak inputlardan birini verir.

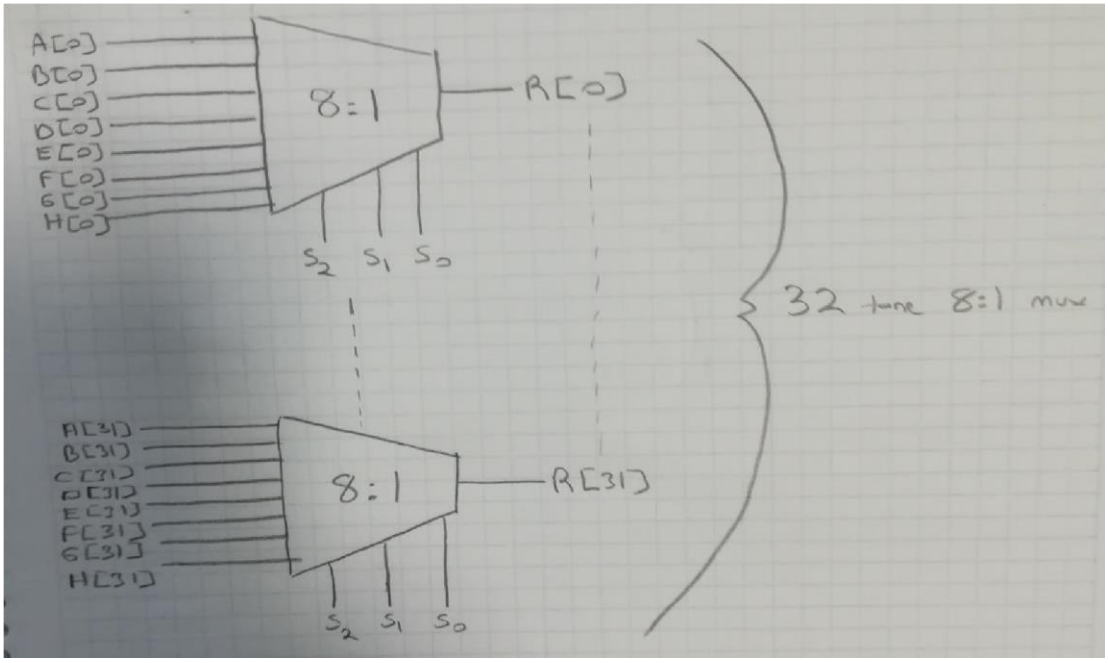


Module mux4x1: Dört adet birer bit input ve iki seçici alarak inputlardan birini verir. Üç adet 2:1 mux kullanılarak tasarlanmıştır.

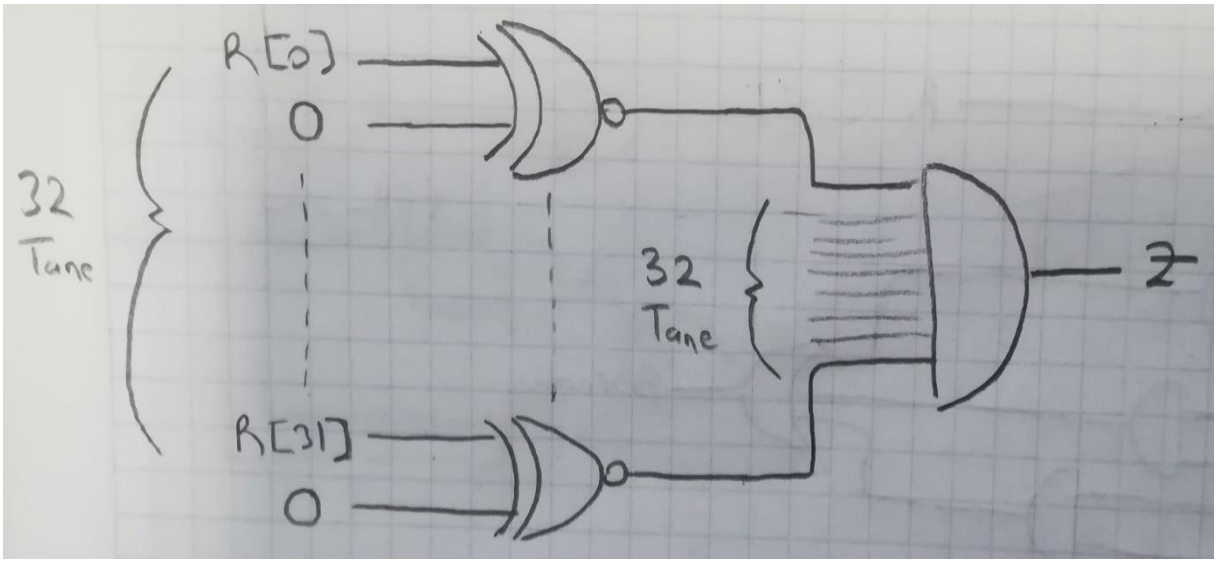
Module mux8x1: Sekiz adet birer bit input ve üç seçici alarak inputlardan birini verir. İki adet 4:1 mux ve 1 adet 2:1 mux kullanılarak tasarlanmıştır.



Module mux8x1_32: 32 bit 8 sayı ve 3 seçici biti input olarak alarak bu 32 bitlik 8 sayıdan birini (Hesaplanmış 8 operasyon sonucundan biri) verir.



Module zero: Input olarak verilen 32 bit sayının tüm bitlerinin 0 olup olmadığını kontrol eder. Tüm bitleri sıfır ise Zero bitini output olarak 1 verir.



Modelsim Test Results

Test benchte arda arda çalıştırılan 10 instructionun sonuçlarının yazıldığı dosyada her instructionun çalıştırılmasının ardından dosyadaki registerların değerleri değiştiği için kolay okunabilirlik açısından R1 ve R7 arası rs ve rt registerı olarak kullanılrken R9 ve R18 arası rd registerı olarak kullanılmıştır. Test bench içinde her instructionun yanında işlem yapılan registerlar ile sonucun yazıldığı rd registerları belirtilmiştir.

```
# time =  0, Instruction = 00000000001000110100100000100000, Result = 00000000000000000000000000000100
# time = 10, Instruction = 00000000010001000101000000100001, Result = 00000000000000000000000000000110
# time = 20, Instruction = 00000000111000100101100000100010, Result = 00000000000000000000000000000101
# time = 30, Instruction = 00000000110000110110000000100011, Result = 00000000000000000000000000000011
# time = 40, Instruction = 00000000101001100110100000100100, Result = 00000000000000000000000000000100
# time = 50, Instruction = 00000000101001100111000000100101, Result = 00000000000000000000000000000111
# time = 60, Instruction = 00000000101001100111100000100111, Result = 11111111111111111111111111111000
# time = 70, Instruction = 00000000101001001000000010000010, Result = 00000000000000000000000000000001
# time = 80, Instruction = 00000000101001111000100011000000, Result = 00000000000000000000000000111000
# time = 90, Instruction = 00000000010001101001000000101011, Result = 00000000000000000000000000000001
```