**Gebze Technical University**
**Computer Engineering**

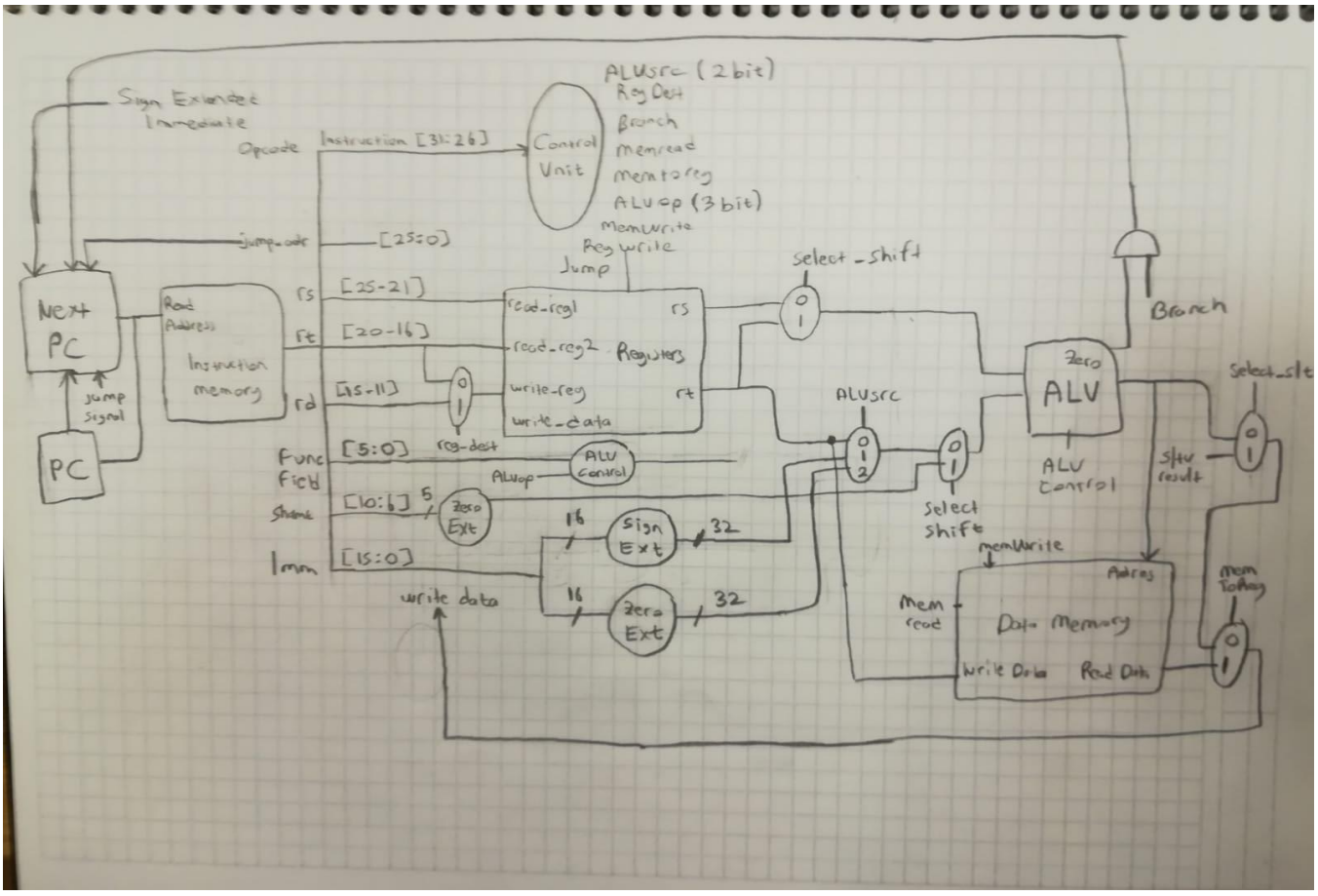
**CSE 331 - 2018**

# HOMEWORK 4 REPORT

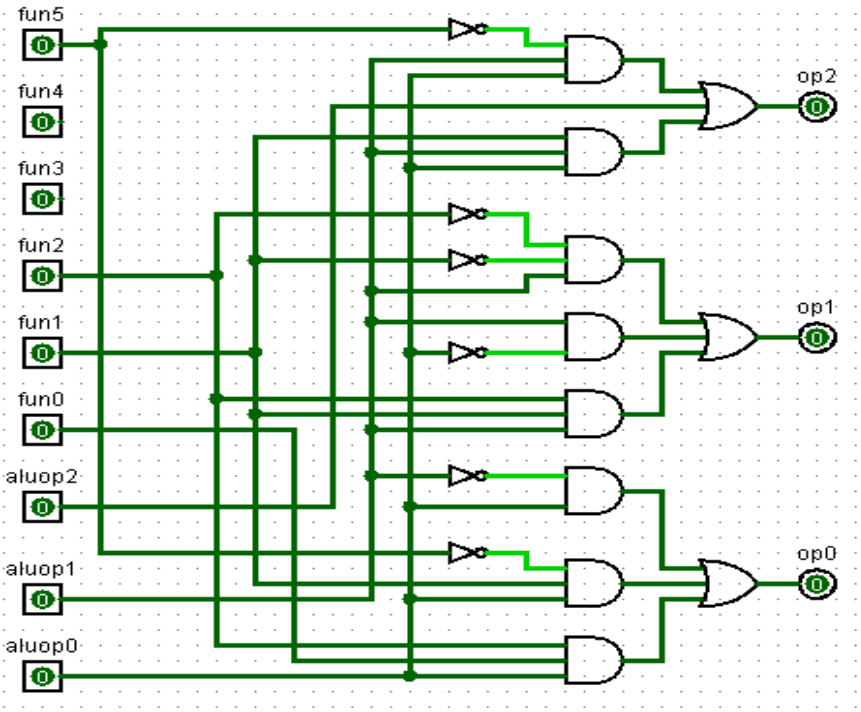
**SERKAN SORMAN**
**151044057**

Course Assistant: Fatma Nur Esirci

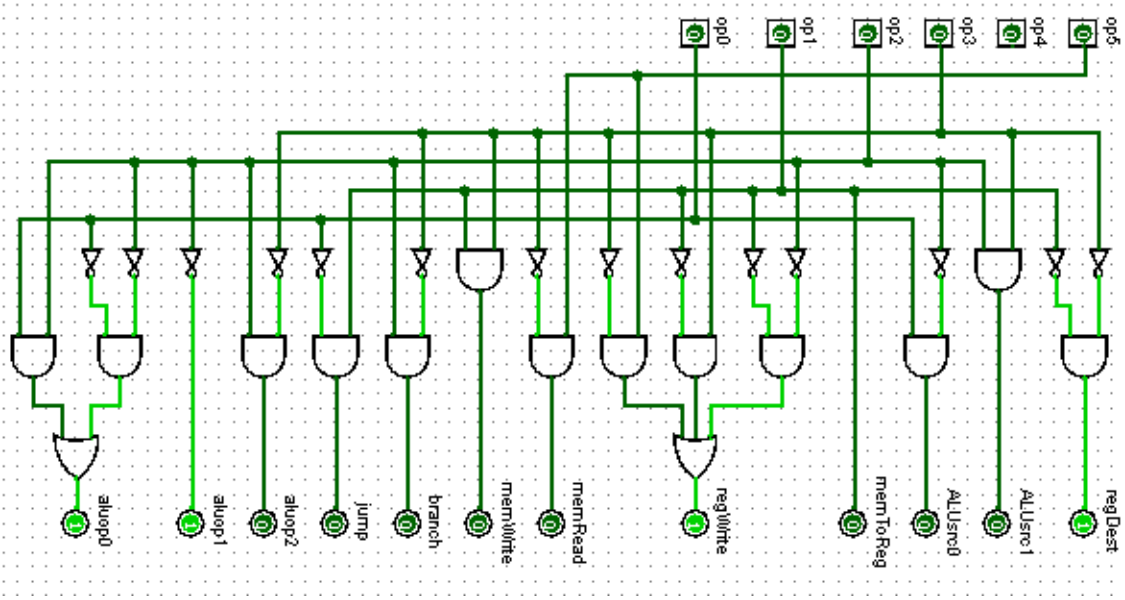
**Module mips32_single_cycle:** Input olarak clock alır. Ardından PC değerine göre instruction memoryden instruction okunur ve opcode,rs,rt,rd,shamt,function field,immediate ve jump adres değerlerine ayrılır. Opcode kontrol unite yollanır ve gerekli sinyaller burada üretilir. Reg dest sinyaline göre sonucun yazılacağı rt ya da rd registerı seçilir. Shamt zero extend edilirken immediate ayrı ayrı zero ve sign extend edilir. Function field ve ALUop alu controle yollanır ve ALUda yapılacak işlem için 3 bitlik op üretilir. Register blokta rs ve rt contentleri okunur. ALU nun ilk inputunun belirlenmesi için instructionun shift olması durumuna bakılır. Bu işlem instructionun tüm opcode larının ve func field[5] in ORlanması ile elde edilir. Yani select shift biti bu durumda 0 olur ve muxdan ilk ALU inputu olarak rt alınır diğer tüm durumlarda ilk ALU inputu rsdir. İkinci ALU inputunun belirlenmesi için 4x1 lik 32 bit mux kullanılır ve rt , sign extend ve zero extendden biri seçilir. ALU ya verilen 3 bit op ile belirtilen işlem yapılır. Rt değeri mem write sinyaline göre memorye yazılmak üzere data memorye verilir. ALudan çıkan sonuç data memorydeki adrese verilir ve mem read sinyaline göre bu adresteki değer read data çıktısı olarak verilir. Ayrıca alunun çıktısı ve sltu result(0 yada 1) muxa sokularak elde edilen sonuç read data ile tekrar muxa sokulur ve ve mem to reg sinyaline göre hangi değerin registera yazılacağı seçilmiş olur. Next PC içinde ise eğer alunun zero biti 1 ve branch sinyali 1 ise PC = PC + 1+ Sign extend immediate olarak hesaplanır ve o adrese branch edilir. Eğer jump sinyali 1 ise PC = Jump adres olarak hesaplanır diğer türlü PC = PC +1 olarak hesaplanır ve sıradaki instruction instruction memorye alınır.



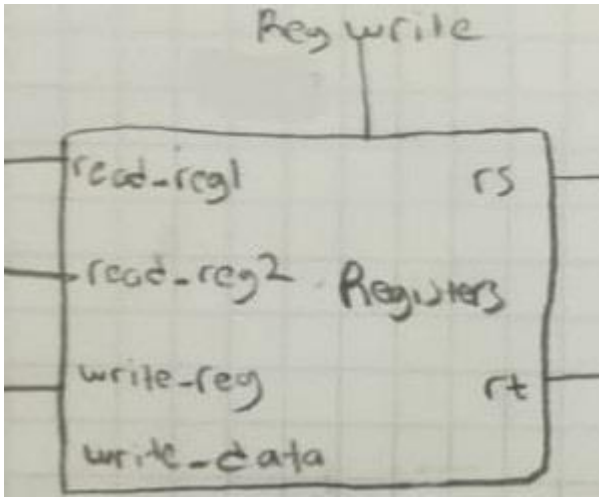
**Module alu_control:** 6 bitlik function field ve 3 bitlik ALUop alır, ALU için 3 bitlik bir koda dönüştürür.




**Module control_unit:** 6 bitlik opcode alır ve çizimde belirtilen sinyalleri üretir.




**Module register_block:** Input olarak rs, rt, write reg, write data ve registera yazma işlemini belirten reg write ve clock sinyali alır. İlk olarak registerlardan rs ve rt nin adreslerine göre contentlerini okur ve output olarak verir. Sonrasında write data yı write regde belirtilen registera yazar.

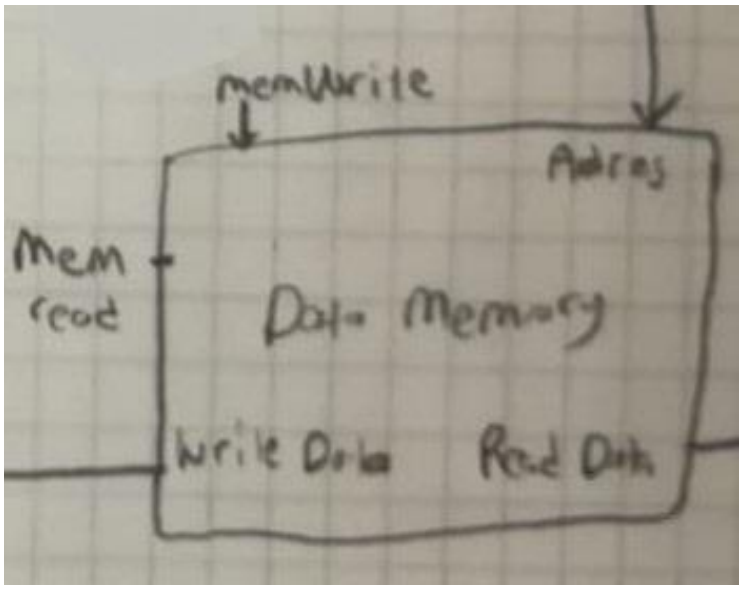



**Module zero_extend:** Input olarak 5 bitlik shamt ı alır ve başına 0 ekleyerek 32 bite extend eder.

**Module zero _extend_immediate:** Input olarak 16 bitlik immediate ı alır ve başına 0 ekleyerek 32 bite extend eder.

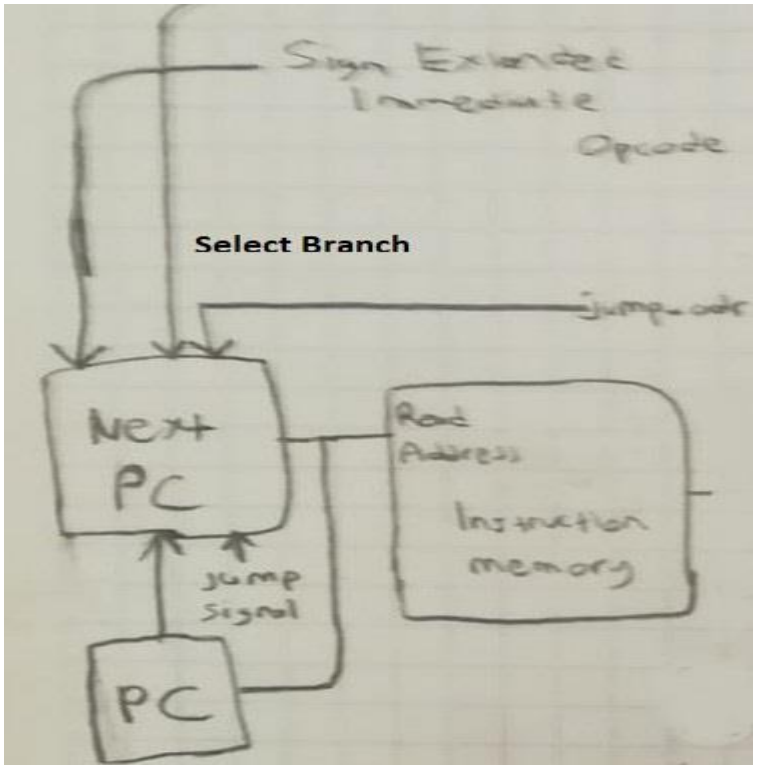
**Module sign_extend:** Input olarak 16 bitlik immediate alır ve sign extend işlemi yapar

**Module data_memory:** Input olarak memory adress, write data, mem read ve mem write sinyallerini alır. Bu sinyallere göre memorye yazma ya da okuma işlemi yapar. Okuma işlemi memory adresindeki bulunan contenti output olarak verir.

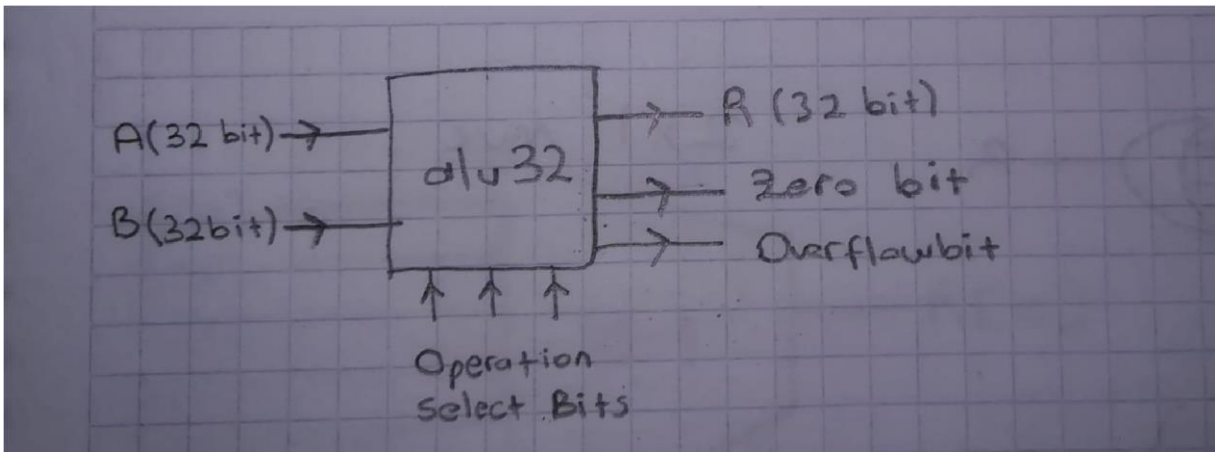


**Module instr_memory:** Input olarak PC alır ve bu PC adresindeki instructionu alarak output olarak verir.

**Module next_PC:** Input olarak PC, sign extended immediate, jump adress, select branch, jump sinyali ve clock alır. Jum sinyali 1 ise Jump adresini hesaplar ve PC ye verir. Branch ise PC = PC + 1+ Sign extend immediate olarak hesaplar. Diğer durumlarda PC = PC + 1 olarak hesaplanır ve PC güncellenir.




**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 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.

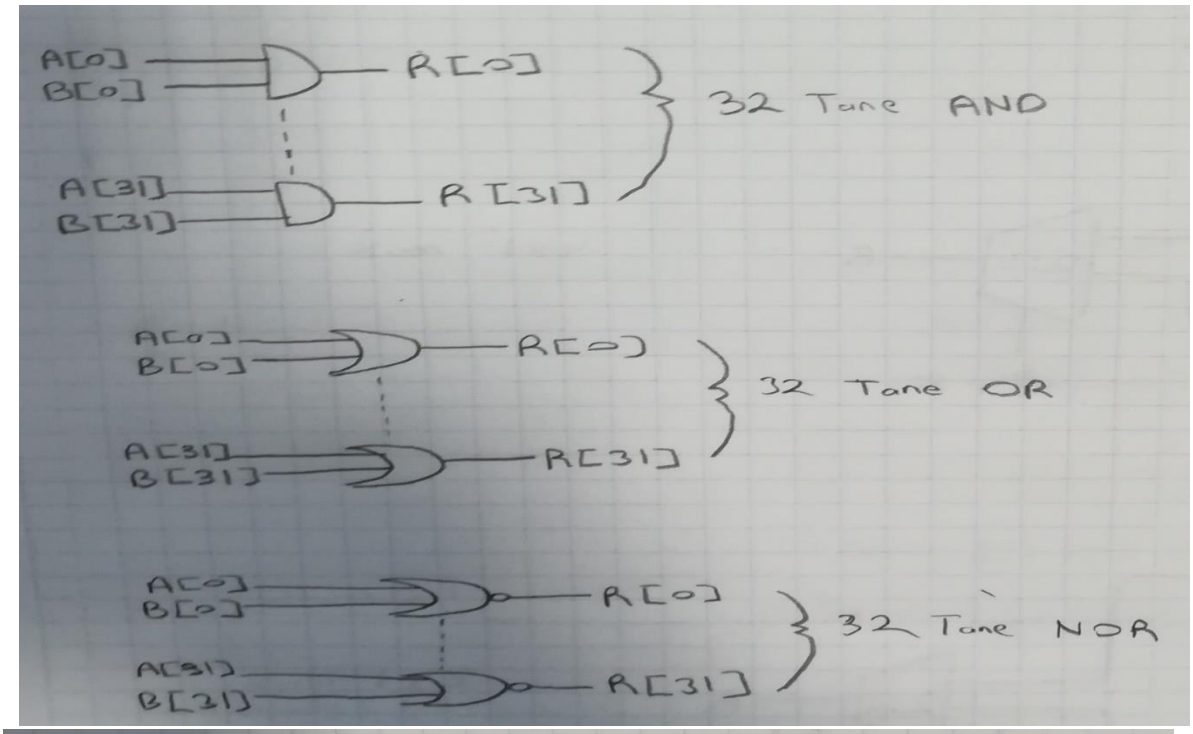


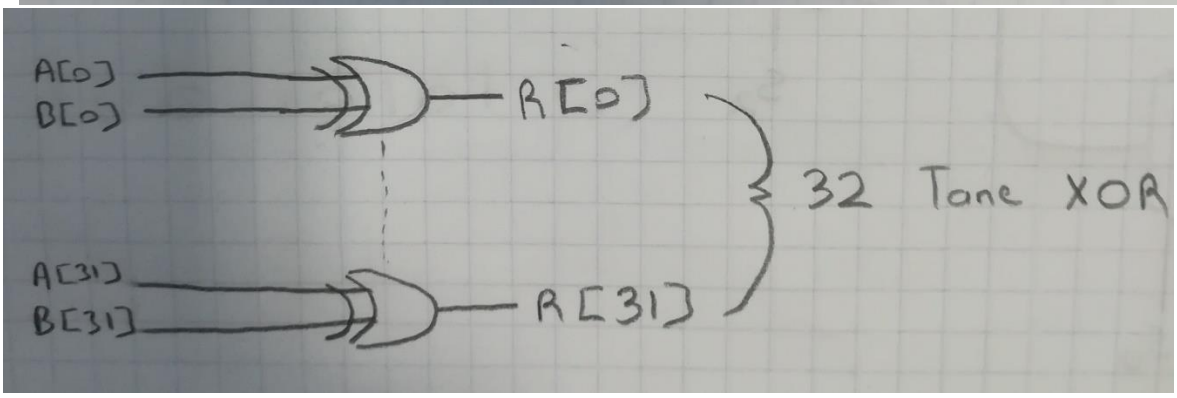
**Module andop**: Input olarak iki adet 32 bit sayı alır. Output olarak iki sayının tüm bitlerinin andlenmiş halini verir.

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

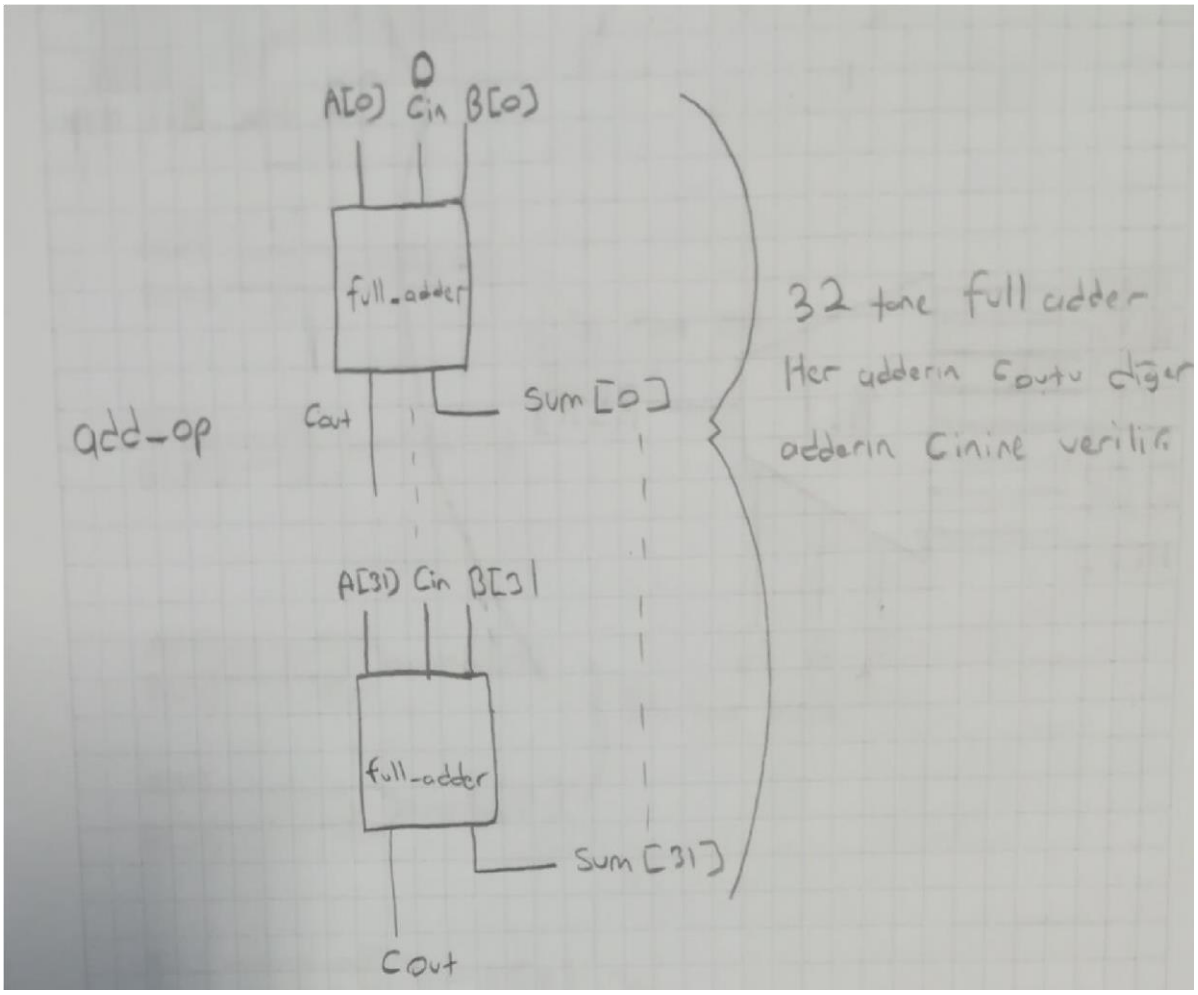
**Module xorp**: Input olarak iki adet 32 bit sayı alır. Output olarak iki sayının tüm bitlerinin xorlanmış halini verir.

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

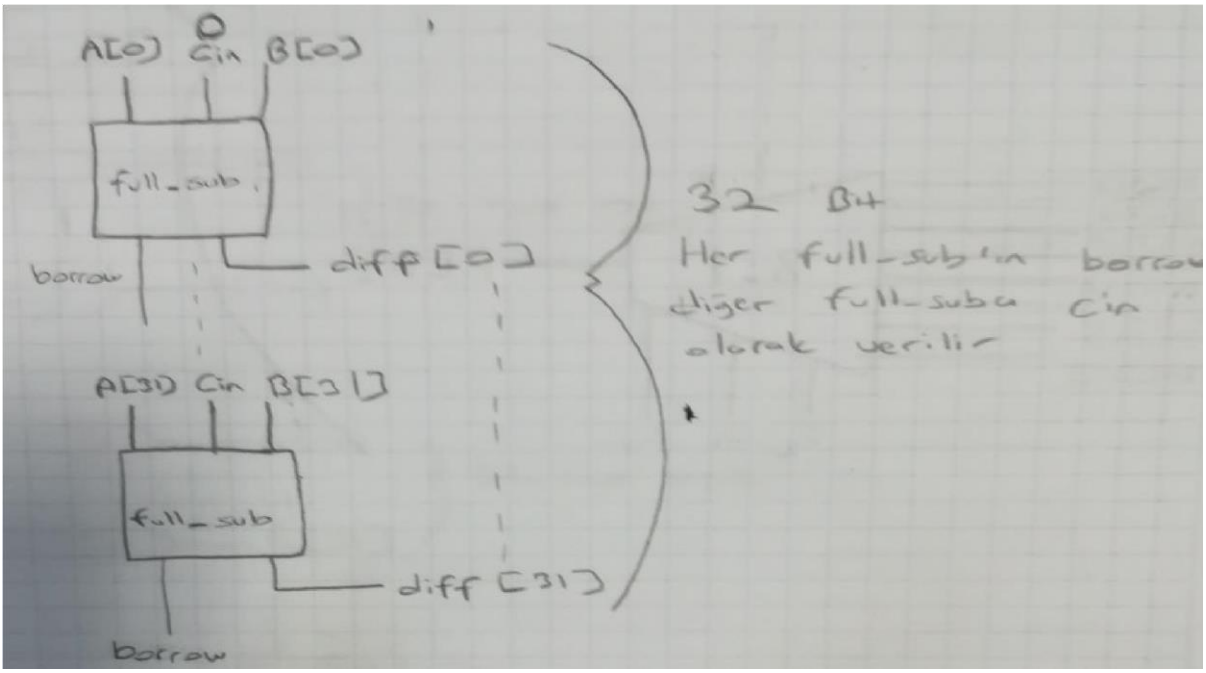


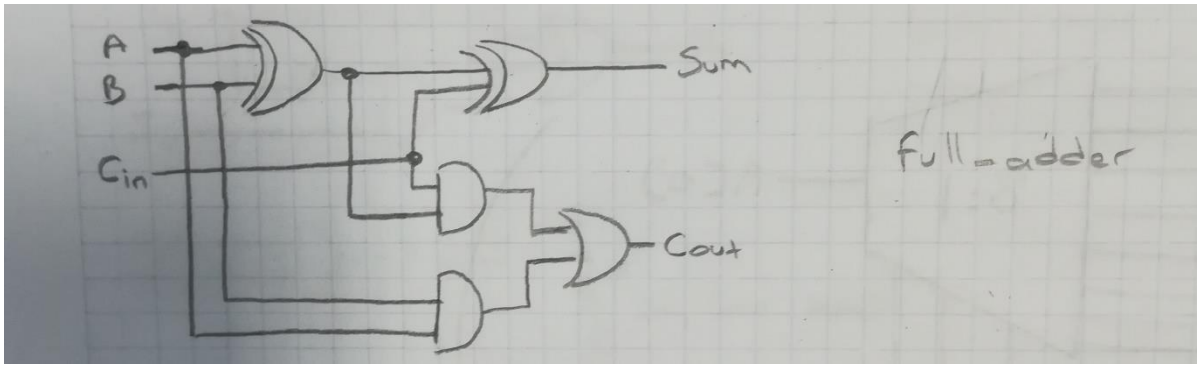

**Module addop:** Input olarak iki adet 32 bit sayı alır ve bu sayılar birer bit olarak full_addera yollanır. Output olarak carry out ve iki sayının toplamını verir.



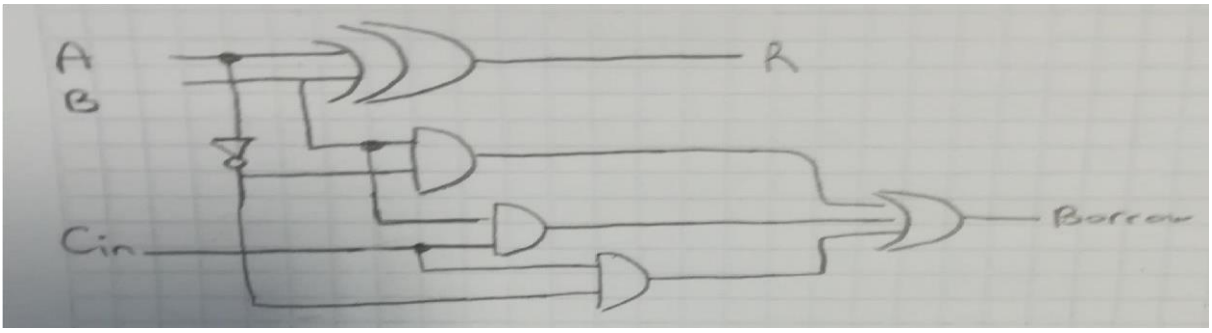

**Module subop:** Input olarak iki adet 32 bit sayı alır ve bu sayılar birer bit olarak full_suba yollanır. Output olarak borrow ve iki sayının farkını verir.



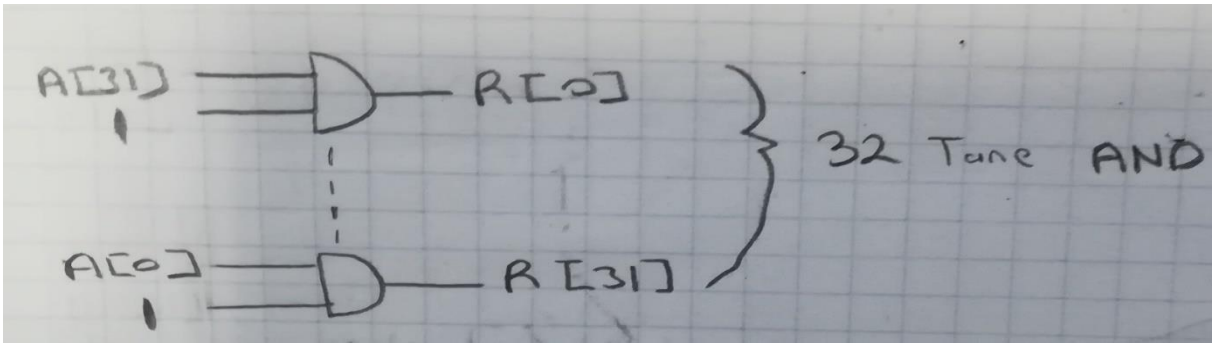
**Module full_adder:** Input olarak iki adet 1 bit sayı ve 1 bit carry in alır. Output olarak carry out ve iki bitin toplamını verir.



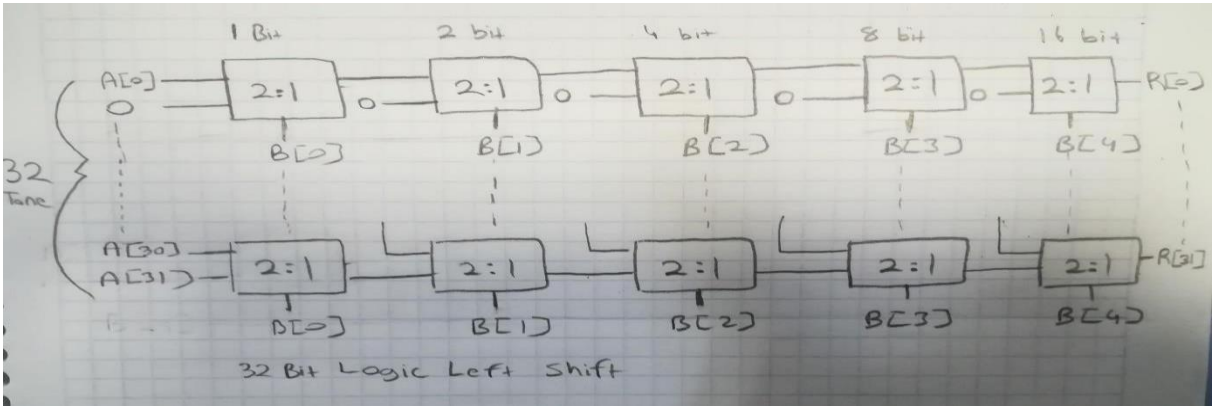

**Module full_sub:** Input olarak iki adet 1 bit sayı ve 1 bit carry in alır. Output olarak borrow ve iki bitin farkını verir.



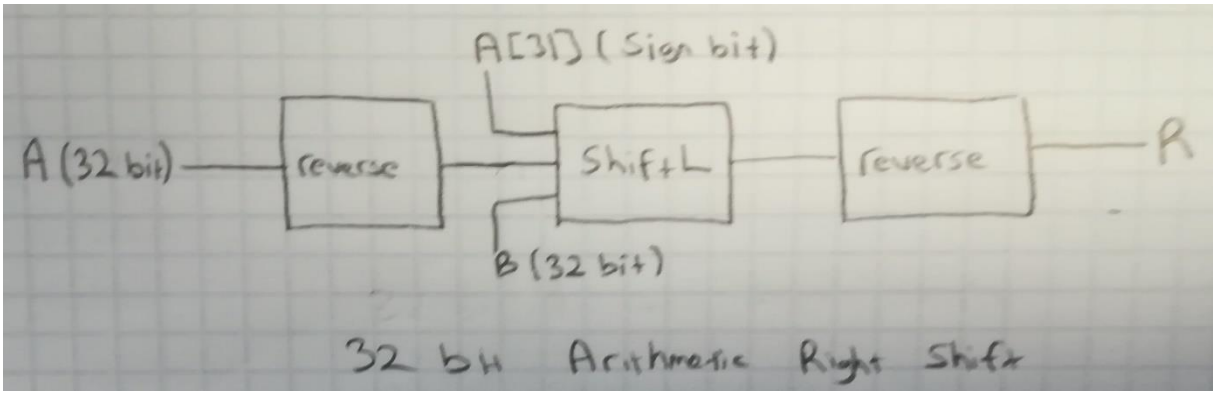

**Module reverse**: Input olarak verilen 32 bit sayının ters çevrilmiş halini output olarak verir.



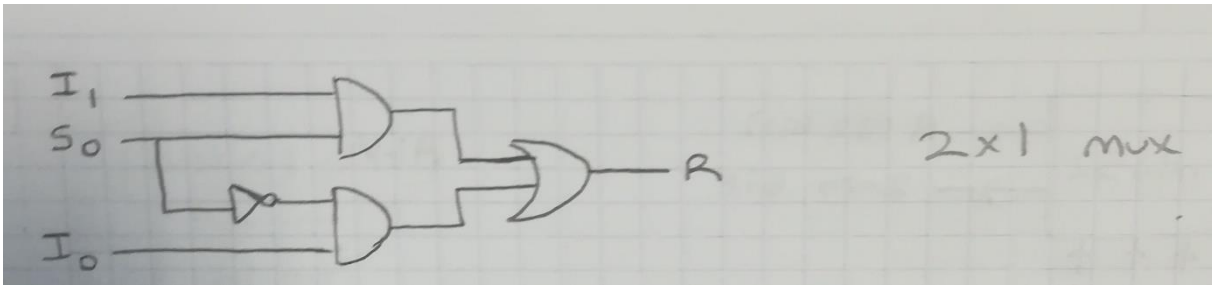

**Module ShiftL**: Input olarak Logic Left Shift edilecek 32 bit bir sayı, Shift edilecek basamak sayısını belirten ikinci bir 32 bit sayı ve Shift işleminde kaydırılan bitlerin yerine konacak biti belirten bit alınır (Logic left shift yapıldığında 0 yollanır). Output olarak verilen 32 bit sayının Left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

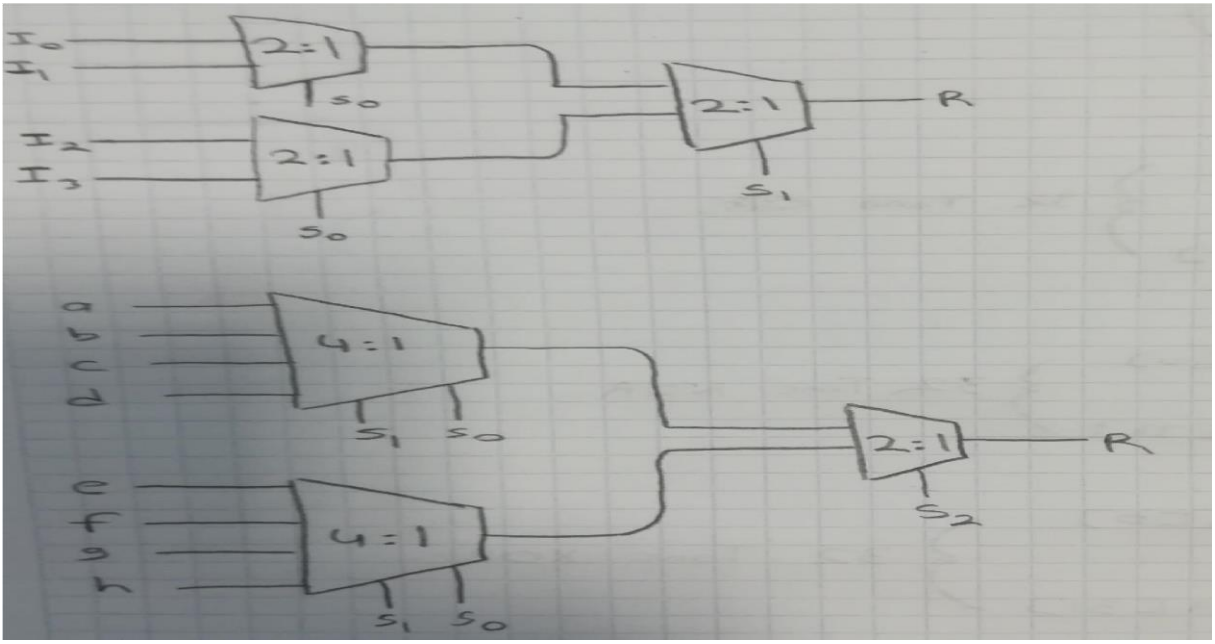



**Module mux2x1**: İki adet birer bit input ve bir seçici alarak inputlardan birini verir.

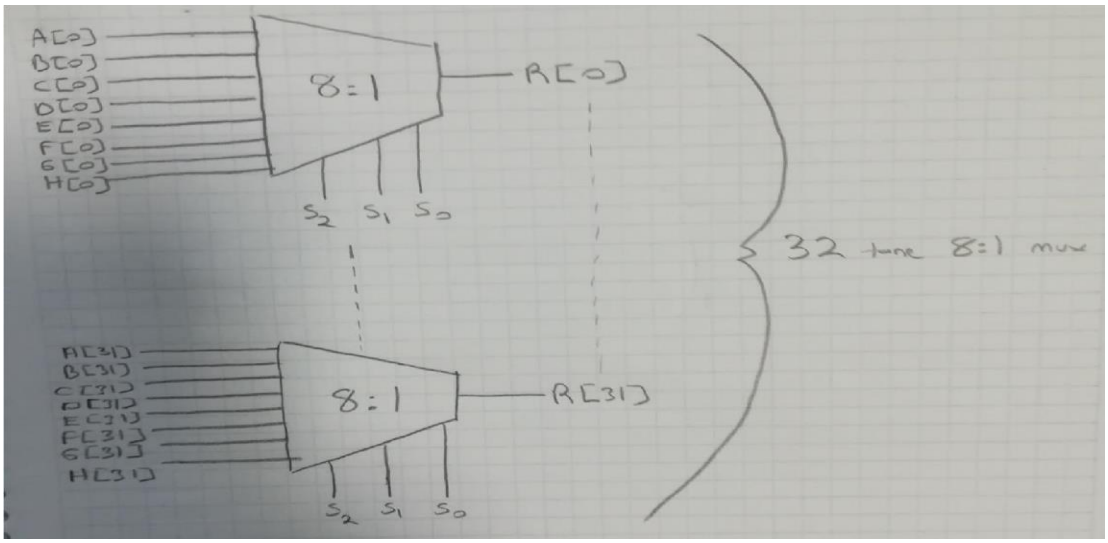



**Module mux4x1**: Dört adet birer bit input ve iki seçici alarak inputlardan birini verir. Üç adet 2:1 mux kullanılarak tasarlanmıştır.

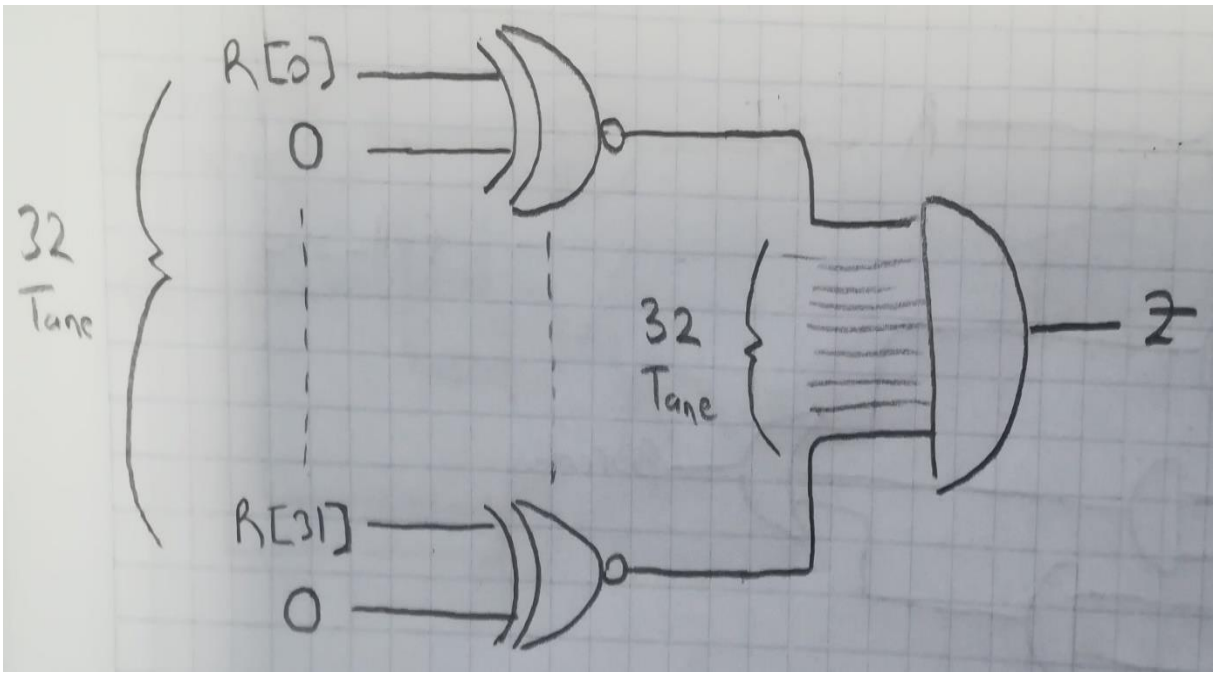
**Module mux8x1:** Sekiz adet birer bit input ve üç seçici alarak inputlardan birini verir. İki adet 4:1 mux ve 1 adet 2:1 mux kullanılarak tasarlanmıştır.



**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 alu_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

```
# Instruction: ac7f0007, ALU Result : 0000000a, RS Content: 00000003, RT Content: 0000001f, RegDest: 11111 ,PC : 00000000 ,clock :0
# Instruction: 8c560003, ALU Result : 00000005, RS Content: 00000002, RT Content: 00000000, RegDest: 10110 ,PC : 00000001 ,clock :1
# Instruction: 8c560003, ALU Result : 00000005, RS Content: 00000002, RT Content: 00000000, RegDest: 10110 ,PC : 00000001 ,clock :0
# Instruction: 351700f0, ALU Result : 000000ff, RS Content: 0000000f, RT Content: 00000000, RegDest: 10111 ,PC : 00000002 ,clock :1
# Instruction: 351700f0, ALU Result : 000000ff, RS Content: 0000000f, RT Content: 00000000, RegDest: 10111 ,PC : 00000002 ,clock :0
# Instruction: 24348000, ALU Result : ffff8001, RS Content: 00000001, RT Content: 00000000, RegDest: 10100 ,PC : 00000003 ,clock :1
# Instruction: 24348000, ALU Result : ffff8001, RS Content: 00000001, RT Content: 00000000, RegDest: 10100 ,PC : 00000003 ,clock :0
# Instruction: 33f5000a, ALU Result : 0000000a, RS Content: 0000001f, RT Content: 00000000, RegDest: 10101 ,PC : 00000004 ,clock :1
# Instruction: 33f5000a, ALU Result : 0000000a, RS Content: 0000001f, RT Content: 00000000, RegDest: 10101 ,PC : 00000004 ,clock :0
# Instruction: 00234820, ALU Result : 00000004, RS Content: 00000001, RT Content: 00000003, RegDest: 01001 ,PC : 00000005 ,clock :1
# Instruction: 00234820, ALU Result : 00000004, RS Content: 00000001, RT Content: 00000003, RegDest: 01001 ,PC : 00000005 ,clock :0
# Instruction: 00445021, ALU Result : 00000006, RS Content: 00000002, RT Content: 00000004, RegDest: 01010 ,PC : 00000006 ,clock :1
# Instruction: 00445021, ALU Result : 00000006, RS Content: 00000002, RT Content: 00000004, RegDest: 01010 ,PC : 00000006 ,clock :0
# Instruction: 00e25822, ALU Result : 00000005, RS Content: 00000007, RT Content: 00000002, RegDest: 01011 ,PC : 00000007 ,clock :1
# Instruction: 00e25822, ALU Result : 00000005, RS Content: 00000007, RT Content: 00000002, RegDest: 01011 ,PC : 00000007 ,clock :0
# Instruction: 00c36023, ALU Result : 00000003, RS Content: 00000006, RT Content: 00000003, RegDest: 01100 ,PC : 00000008 ,clock :1
# Instruction: 00c36023, ALU Result : 00000003, RS Content: 00000006, RT Content: 00000003, RegDest: 01100 ,PC : 00000008 ,clock :0
# Instruction: 00a66824, ALU Result : 00000004, RS Content: 00000005, RT Content: 00000006, RegDest: 01101 ,PC : 00000009 ,clock :1
# Instruction: 00a66824, ALU Result : 00000004, RS Content: 00000005, RT Content: 00000006, RegDest: 01101 ,PC : 00000009 ,clock :0
# Instruction: 00a67025, ALU Result : 00000007, RS Content: 00000005, RT Content: 00000006, RegDest: 01110 ,PC : 0000000a ,clock :1
# Instruction: 00a67025, ALU Result : 00000007, RS Content: 00000005, RT Content: 00000006, RegDest: 01110 ,PC : 0000000a ,clock :0
# Instruction: 00a67827, ALU Result : fffffff8, RS Content: 00000005, RT Content: 00000006, RegDest: 01111 ,PC : 0000000b ,clock :1
# Instruction: 00a67827, ALU Result : fffffff8, RS Content: 00000005, RT Content: 00000006, RegDest: 01111 ,PC : 0000000b ,clock :0
# Instruction: 00a48082, ALU Result : 00000001, RS Content: 00000005, RT Content: 00000004, RegDest: 10000 ,PC : 0000000c ,clock :1
# Instruction: 00a48082, ALU Result : 00000001, RS Content: 00000005, RT Content: 00000004, RegDest: 10000 ,PC : 0000000c ,clock :0
# Instruction: 00a788c0, ALU Result : 00000038, RS Content: 00000005, RT Content: 00000007, RegDest: 10001 ,PC : 0000000d ,clock :1
# Instruction: 00a788c0, ALU Result : 00000038, RS Content: 00000005, RT Content: 00000007, RegDest: 10001 ,PC : 0000000d ,clock :0
# Instruction: 0046902b, ALU Result : 00000001, RS Content: 00000002, RT Content: 00000006, RegDest: 10010 ,PC : 0000000e ,clock :1
# Instruction: 0046902b, ALU Result : 00000001, RS Content: 00000002, RT Content: 00000006, RegDest: 10010 ,PC : 0000000e ,clock :0
# Instruction: 10210002, ALU Result : 00000000, RS Content: 00000001, RT Content: 00000001, RegDest: 00000 ,PC : 0000000f ,clock :1
# Instruction: 10210002, ALU Result : 00000000, RS Content: 00000001, RT Content: 00000001, RegDest: 00000 ,PC : 0000000f ,clock :0
# Instruction: 08000011, ALU Result : 00000000, RS Content: 00000000, RT Content: 00000000, RegDest: 00000 ,PC : 00000012 ,clock :1
# Instruction: 08000011, ALU Result : 00000000, RS Content: 00000000, RT Content: 00000000, RegDest: 00000 ,PC : 00000012 ,clock :0
# Instruction: 0044c821, ALU Result : 00000006, RS Content: 00000002, RT Content: 00000004, RegDest: 11001 ,PC : 00000011 ,clock :1
# Instruction: 0044c821, ALU Result : 00000006, RS Content: 00000002, RT Content: 00000004, RegDest: 11001 ,PC : 00000011 ,clock :0
```

Başlangıç instructionu hariç her instruction ve içerikleri clock 0 ve clock 1 olmak üzere iki kere print edilmiştir.

Sadece gerekli olduğu yerde gösterilmek üzere her instructionun alu result, rs rt contenti reg dest vs. print ettirilmiştir. Bu printleri sadece gerekli instructionlar için dikkate alın.

Instructionların okunduğu arrayin boyutu 32 instruction alacak şekilde initialize edilmiştir. Dosyadan 32'den fazla instruction okunmak istendiğinde bu değer de ona göre değiştirilmelidir.

Instructionlar çalıştırıldıktan sonra kolay okunurluk açısından, başlangıç olarak register dosyası içinde okuma yapılacak registerlar (R0 dan R8e kadar ve R31) haricindeki tüm registerların contentleri 0 olarak verilmiştir. Aynı şekilde data memory dosyasında da okuma yapılacak 5. Adresteki 5 değeri haricinde diğer tüm adreslerdeki contentler 0 olarak ayarlanmıştır.

Instruction memorydeki tüm instructionlar aşağıda açıklanmıştır. Yukarıdaki belirtilen bazı hexedecimal değerler aşağıda decimal olarak gösterilmiştir

**1.Instruction) AC7F0007**

**sw instructionu**: rs content = 3, immediate = 7. RT olarak kullanılan R31 registerı içindeki 31 değerini data memorydeki 3+7 (RS+Sign extended immediate) adresine yazar. Test çıktısında görüldüğü gibi ALU result 10 dur. Data memorydeki 10 adresine 31 değeri yazılmış olur.

**2.Instruction) 8C560003**

**lw instructionu**: rs content = 2, rt adres = 22, immediate = 3. Data memorydeki 2+3 (RS+Sign extended immediate) adresindeki 5 değeri RT ye yazılır. Test çıktısında görüldüğü gibi ALU result 5 dir. Data memorydeki 5 adresinde bulunan 5 değeri RT olarak kullanılan register 22 ye yazılmış olur.

**3.Instruction) 351700F0**

**ori instructionu**: rt adres = 23, rs adres = 8, rs content = 0000000f, immediate = 000000f0. RS olarak kullanılan R8 registerı içindeki 0000000f değeri ile zero extend edilmiş immediate 000000f0 değeri or lanmış ve ALU resultda görüldüğü gibi 000000ff elde edilmiş ve RT olarak kullanılan register 23 e yazılmıştır.

**4.Instruction) 24348000**

**addiu instructionu**: rt adres = 20, rs content = 1, immediate = 8000. RS olarak kullanılan R1 registerı içindeki 1 değeri ile sign extend edilmiş immediate toplanarak register 20 ye yazılır. Sign extend edilmiş immediate ve rs contenti toplamı ALU resultunda görüldüğü gibidir ( FFFF8001).

**5.Instruction) 33F5000A**

**andi instructionu**: rt adres = 21, rs adres = 31, immediate = 000A. RS olarak kullanılan R31 içindeki 0000000F ile zero extended immediate olan 0000000A değerleri andlenerek ALU resultunda görüldüğü gibi 0000000A elde edilir ve register 21 e yazılır.

**6.Instruction) 00234820**

**add instructionu**: RS olarak kullanılan R1 registerı içindeki 1 değeri ile RT olarak kullanılan R3 registerı içindeki 3 değeri toplanır ve RD olarak belirtilen R9 a yazılır. R9 a 4 yazılmış olur.

**7.Instruction) 00445021**

**addu instructionu**: RS olarak kullanılan R2 registerı içindeki 2 değeri ile RT olarak kullanılan R4 registerı içindeki 4 değeri toplanır ve RD olarak belirtilen R10 a yazılır. R10 a 6 yazılmış olur.

**8.Instruction) 00E25822**

**sub instructionu**: RS olarak kullanılan R7 registerı içindeki 7 değeriden RT olarak kullanılan R2 registerı içindeki 2 değeri çıkarılır ve RD olarak belirtilen R11 e yazılır. R11 e 5 yazılmış olur.

**9.Instruction) 00C36023**

**subu instructionu**: RS olarak kullanılan R6 registerı içindeki 6 değeriden RT olarak kullanılan R3 registerı içindeki 3 değeri çıkarılır ve RD olarak belirtilen R12 ye yazılır. R12 ye 3 yazılmış olur.

**10.Instruction) 00A66824**

**and instructionu**: RS olarak kullanılan R5 registerı içindeki 5 değeri ile RT olarak kullanılan R6 registerı içindeki 6 değeri andlenir ve RD olarak belirtilen R13 ye yazılır. R13 ye 4 yazılmış olur.

**11.Instruction) 00A67025**

**or instructionu**: RS olarak kullanılan R5 registerı içindeki 5 değeri ile RT olarak kullanılan R6 registerı içindeki 6 değeri orlanır ve RD olarak belirtilen R14 e yazılır. R14 e 7 yazılmış olur.

**12.Instruction) 00A67827**

**nor instructionu**: RS olarak kullanılan R5 registerı içindeki 5 değeri ile RT olarak kullanılan R6 registerı içindeki 6 değeri norlanır ve RD olarak belirtilen R15 e yazılır. R15 e FFFFFFF8 yazılmış olur.

**13.Instruction) 00A48082**

**slr instructionu**: RT olarak kullanılan R4 registerı içindeki 4 değeri 2 basamak(shamt) sağa logic shift edilip RD olarak belirtilen R16 ya yazılmıştır. R16 ya 1 yazılmış olur.

**14.Instruction) 00A788C0**

**sll instructionu**: RT olarak kullanılan R7 registerı içindeki 7 değeri 3 basamak(shamt) sola logic shift edilip RD olarak belirtilen R17 ye yazılmıştır. R17 ye 00000038 yazılmış olur.

**15.Instruction) 0046902B**

**sltu instructionu**: RS olarak kullanılan R2 registerı içindeki 2 değeri ile RT olarak kullanılan R6 registerı içindeki 6 değeri karşılaştırılarak(2<6) RD olarak belirtilen R18 e 1 yazılmıştır.

**16.Instruction) 10210002**

**beq instructionu**: immediate = 2. RS olarak kullanılan R1 ile RT olarak kullanılan R1 değerlerinin eşit olması sonucu PC + 1 + signextend immediate (+3) ileriye branch yapılır. Böylece PC counter test çıktısında görüldüğü gibi 0000000F den 00000012 ye yani 15 den 18 e geçer. Aradaki iki instruction(**0023C020** ve **0044C821**) atlanıp 18 de bulunan jump instructionundan devam edilmiş olur.

**19.Instruction) 08000011**

**jump instructionu**: jump adress olarak belirtilen 00000011 adresine jump eder. Bu adres jump instructionun bir gerisi yani 17 dir. Böylece branch sonucu atlanan iki instructiondan ikincisine (**0044C821**) jump edilir ve buradan devam edilmiş olur.

**18.Instruction) 0044C821**

**addu instructionu**: RS olarak kullanılan R2 registerı içindeki 2 değeri ile RT olarak kullanılan R4 registerı içindeki 4 değeri toplanır ve RD olarak belirtilen R25 e yazılır. R25 e 6 yazılmış olur.

Sonuç olarak **0044C821** instructionu çalıştırıldıktan sonra test durdurulur ve **0023C020** instructionı atlanmış olur böylece R24 registerına R1 ve R3 registerlarındaki değerlerin toplamı yazılamaz R24 registerı eski contenti olan 0 olarak kalır.