CSE 331 Computer Organization

Project 3

R-type Single cycle MIPS with Structural Verilog

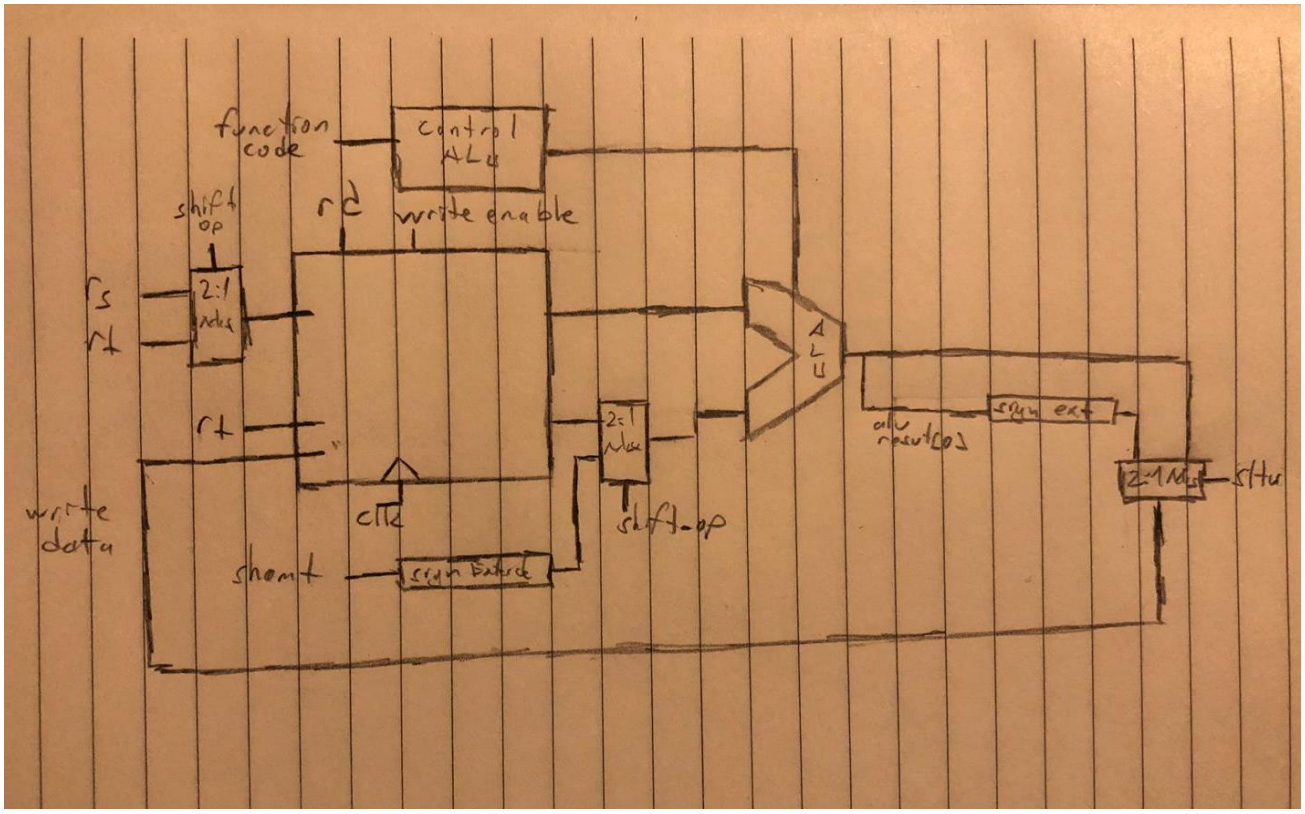
Batuhan TOPALOĞLU 151044026

GTÜ Bilgisayar Mühendisliği Bölümü 2018

1. Şematik Tasarımlar:

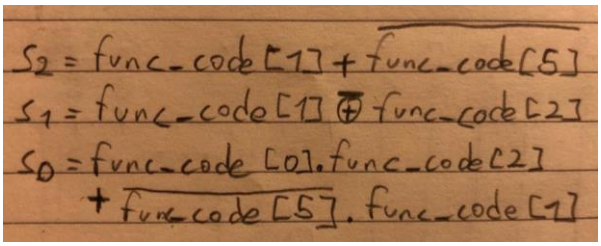
1.1 Data path, mipz32:

Sadece R type instructionları işleyeceğimiz için Lecture_6'da gördüğümüz datapath den biraz daha farklı bir yapı tasarlandı. Burada görüldüğü gibi bir ana control unit ve alu control unit ayrımı yok onun yerine alu nun seçme bacak girişlerini kontrol eden bir unit var. Hafızaya yazma ve hafızadan okuma işlemleri olmadığı için memory bloğu ve onun beraberinde getirdiği bazı seçme elemanları yok. Aşağıda görülen bütün bileşenler ayrı ayrı modül olarak tasarlanarak gerçeklendi. İleriki bölümlerde detaylı olarak açıklanacaklar.

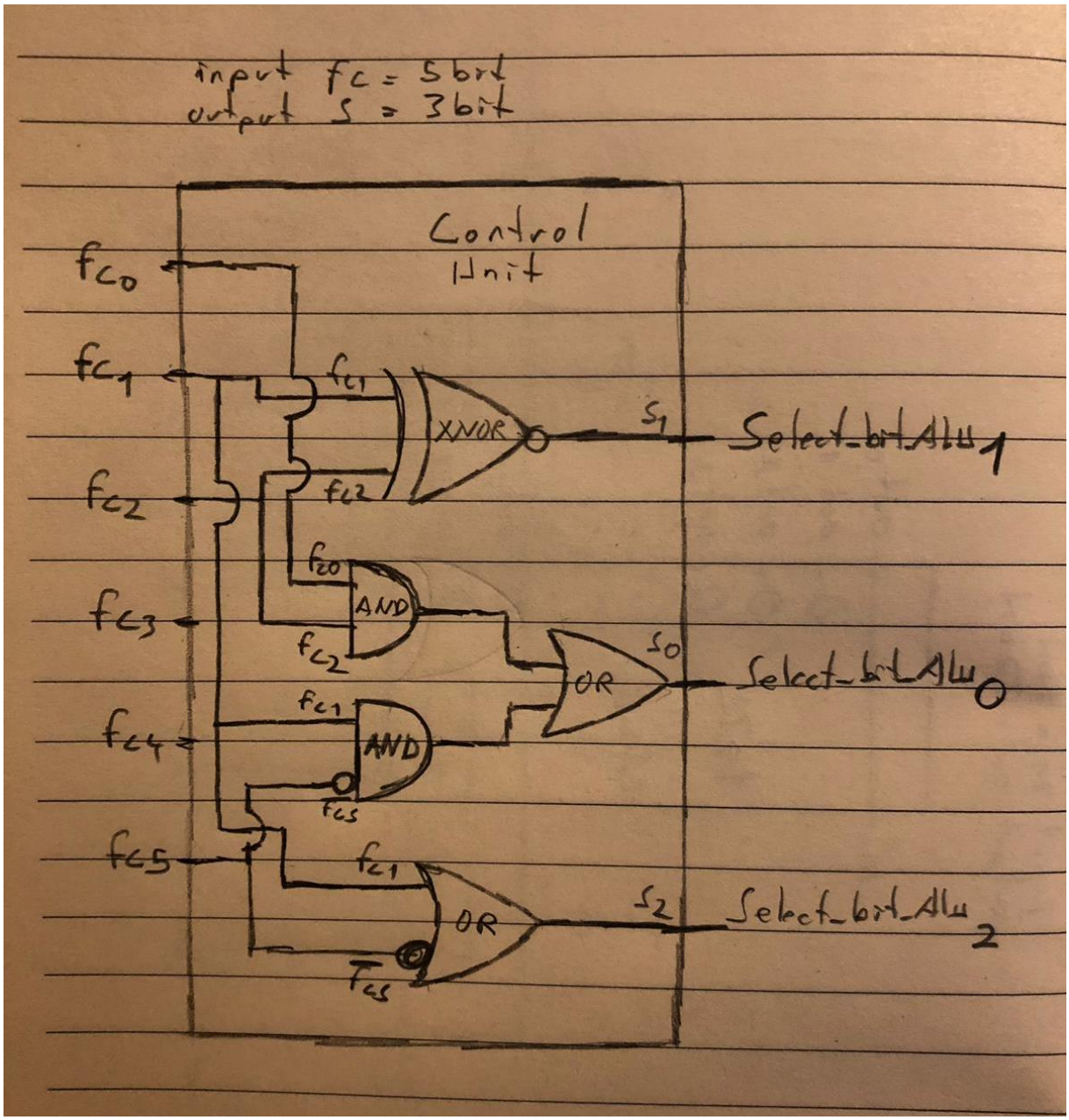


1.2 Control Unit:

Bütün instruction girdilerimiz R-type olacagında instruction ilk 6 biti ile ifade edilen opCode'da bütün R-type lar için aynı olduğundan bu bitlerle ilgili bir işlem yapmıyorum ve opCode'un 6bit 0 olacağının garantilendiğini düşünerek bir tasarım yaptım. Control unit instruction ın son 6 biti ile alu'nun 3bitlik seçme bacaklarının ne olacağına karar veren bir yapı. Control unit in iç yapısını hesaplamak için tablolardan yararlanar 3 bitlik çıkışlar için şu denklemleri buldum:



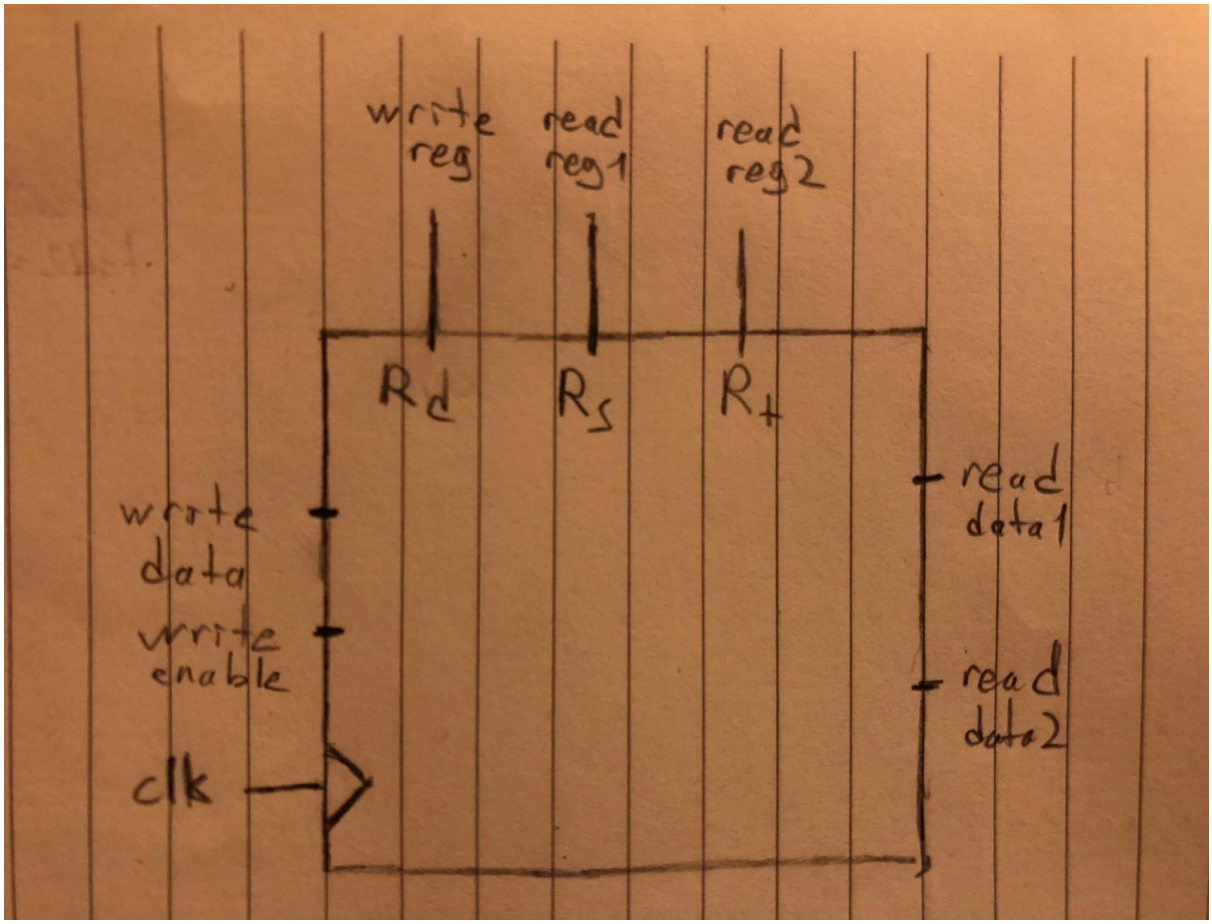
Lojik kapılarla yukarıdaki denklemleri gerçekleyince control unit bu şekilde bir yapıya sahip oldu:



(control unit)

1.3 Register File:

Register file ı buradan şematik olarka nasıl ifade edebileceğimi tam olarak bilemedim ama sanırım bu şekilde bir gösterim yeterlidir. Daha fazla detay iç yapısına giriyor ki onu Modüller başlığı altında bulabilirsiniz.

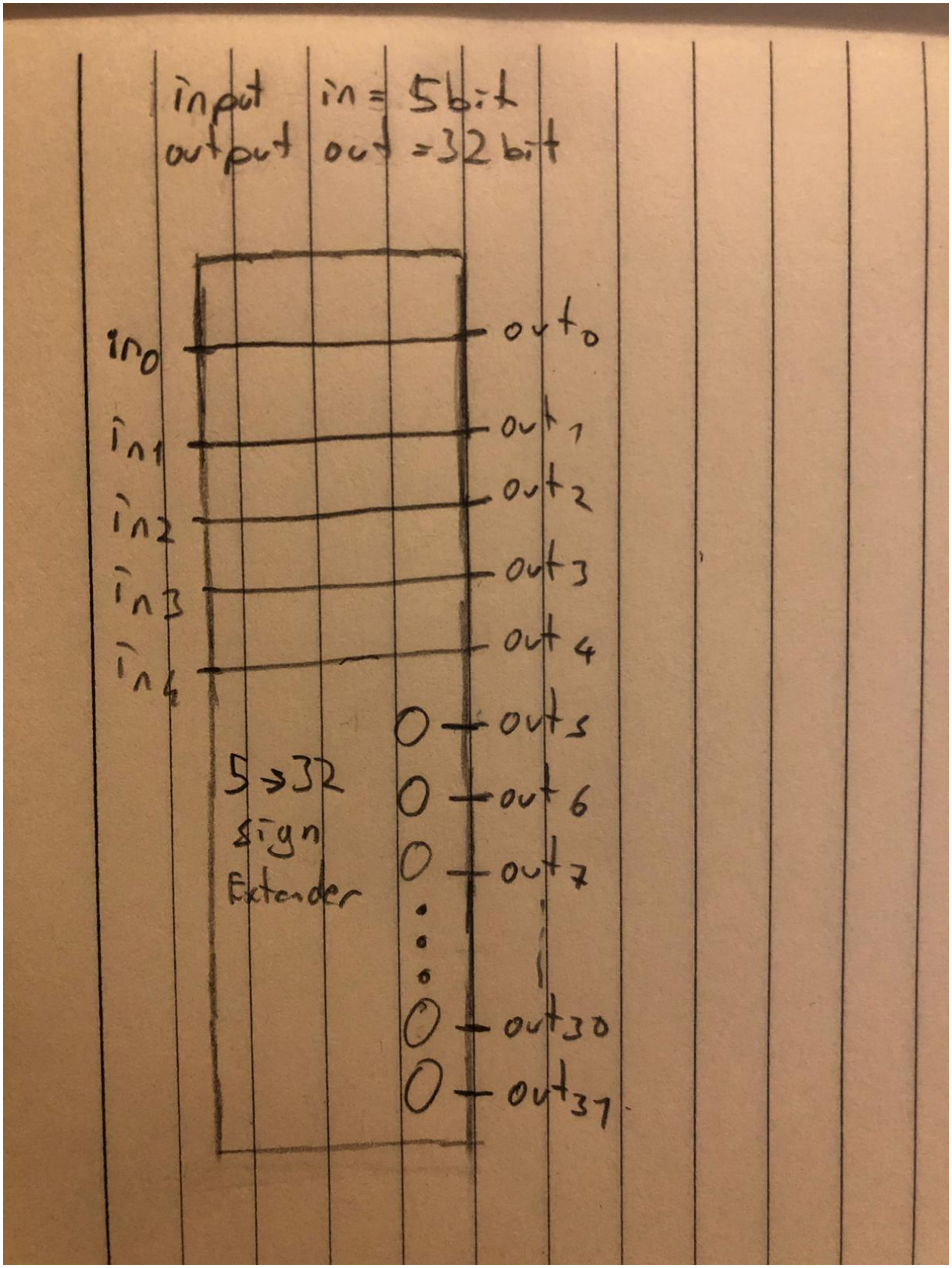


1.4 Sign Extensers:

Bu projede iki farklı sign extender kullandım. İlki shift instructionlarında 5 bitlik shift amountu 32 bit extend etmek için gerekli olan sign extender diğeri ise sltu instructionda alu sonucunun 31. Bitini alıp onu 32 bite extend eden sign extender.

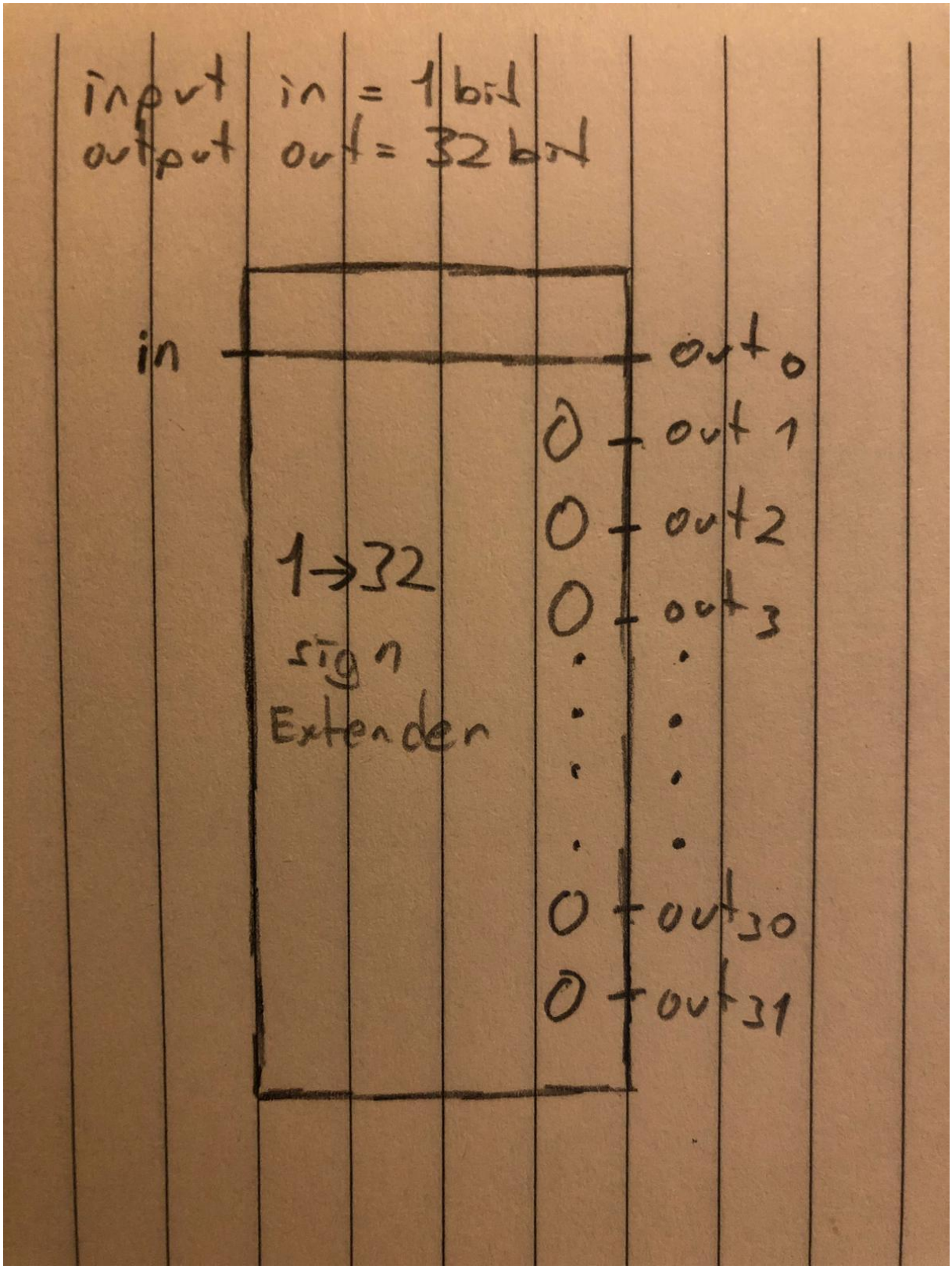
1.4.1 5bit->32bit Sign Extender:

Shift instructionları kullanılan signExtender modülünün iç yapısı:



1.4.2 1bit->32bit Sign Extender:

Sltu instruction ı için kullanılan signExtender_v2 modülünün iç yapısı:

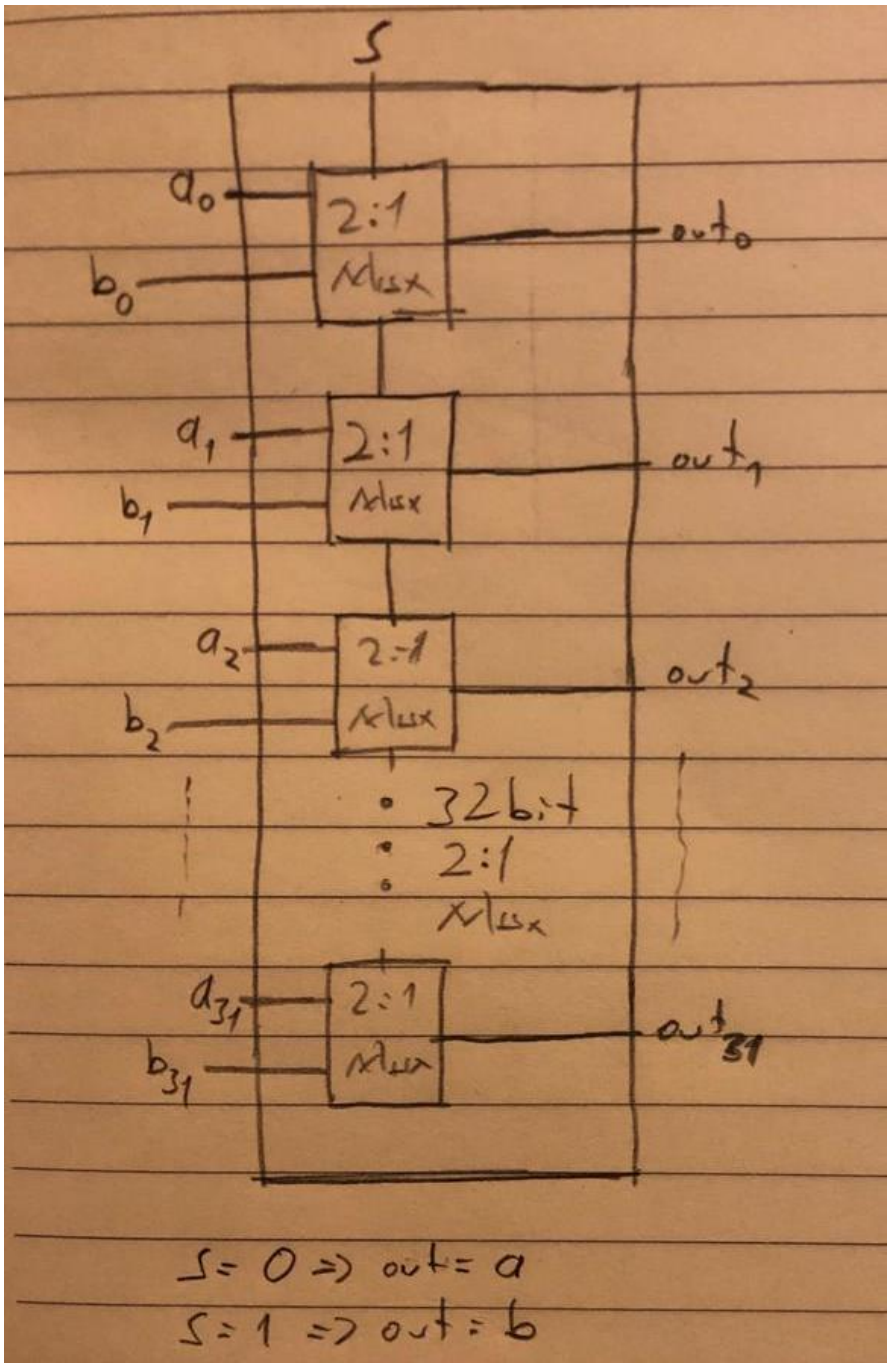


1.5 Mux:

Bu proje alu32 projesi için tasarlanan mux'lardan yararlanılsadad ek olarak tasarlanması gereken 2 mux'a ihtiyaçım oldu. İlki iki tane 32bitlik veri girişi olan bir 2:1 mux , diğeri ise iki tane 5bitlik veri girişi olan bir 2:1 mux.

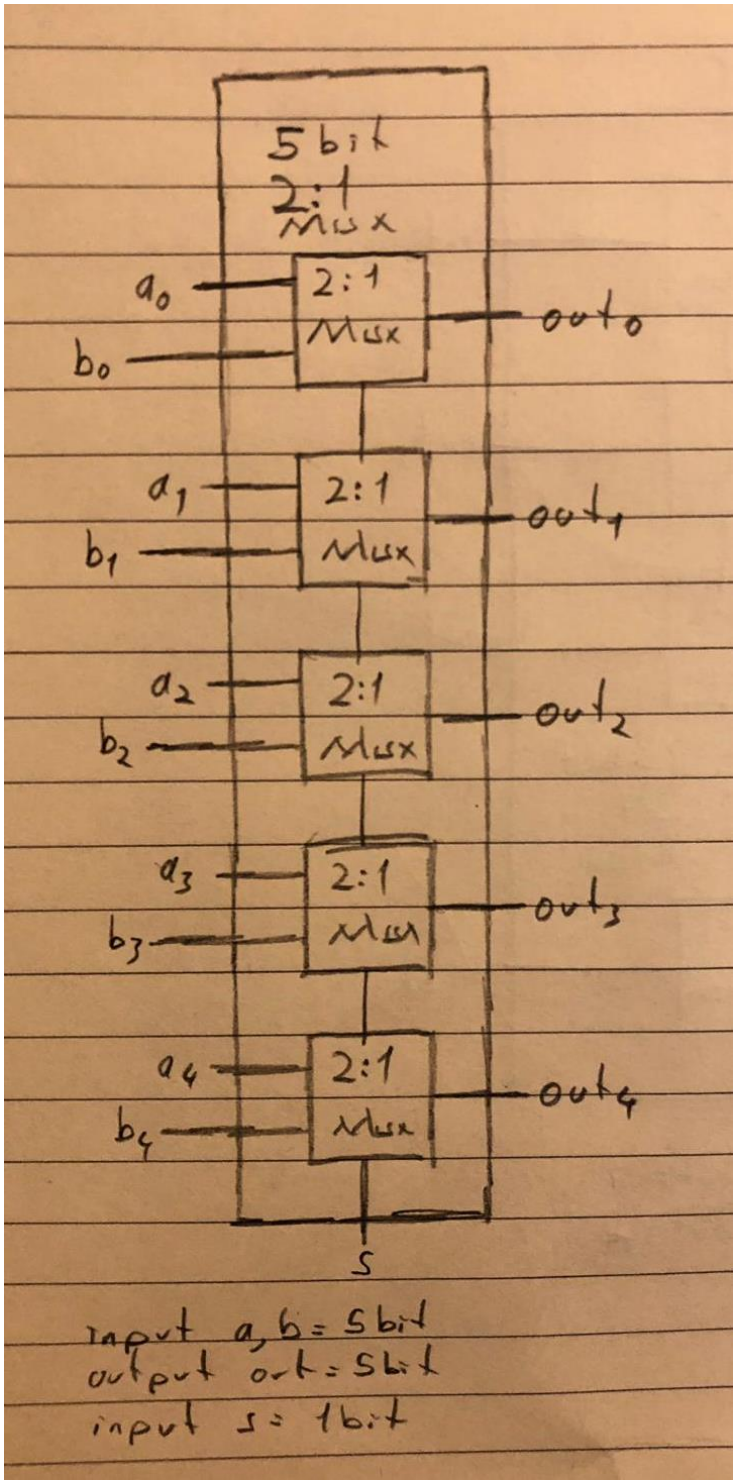
1.5.1 32bit veriler seçen 2:1 mux:

Bu mux alu'nun alt bacak girişi için rt nin verisi ile sign extend edilmiş shamnt değerlerinden instruction tipine göre birini seçmek için ve alu sonu ile sltu için sign extende edilmiş değerlerden birini seçmek için kullanılır.



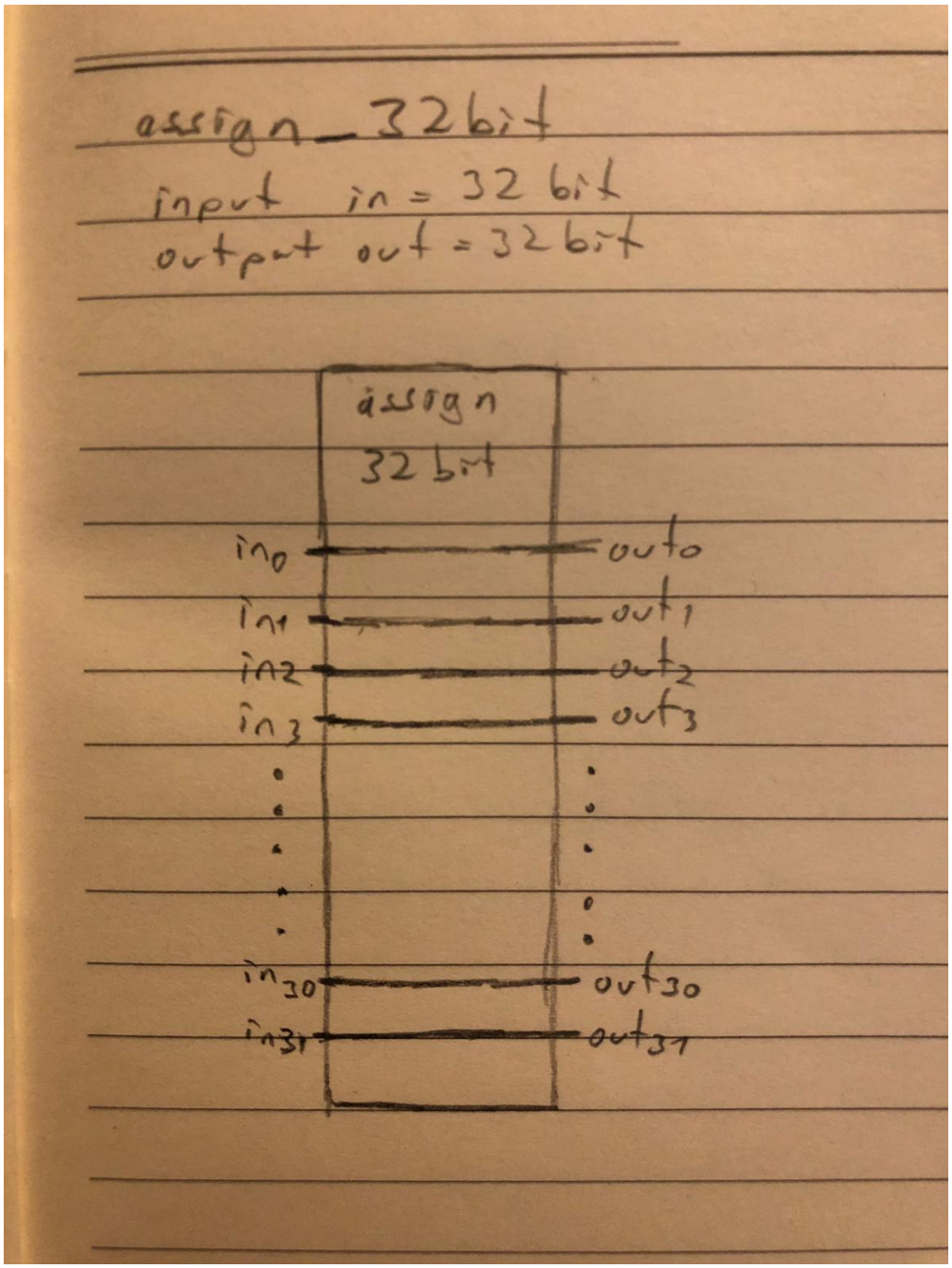
1.5.2 5bit veriler seçen 2:1 mux:

Bu mux register file ın rs read_reg1 girişi için rs veya rt değerlerinden hangisinin gireceğine karar vermek için kullanılır.



1.6 32bit assign:

32bitlik iki değeri birbirine atama işlemi daha modüler hale getirmek için tasarlanan basit birt modül. Result datasını write_data değerine atamak için kulllanılıyor.



2. Modüller:

2.1 module mips32(instruction, clk, result);

İnstruction 32bitlik işleyeceğimiz instruction, clk register file için gerekli olan clock girişi ve result instruction işlendikten sonra rd register ına yazılacak olan 32bit değer.

Bu modülde şematik tasarımda görülen bileşenler için tasarladğım diğer modüllerin gerekli sayıda objeleri oluşturularak ve gerekli giriş çıkış bacakları ayarlanarak gerçeklendi.

2.2 module mips_registers (read_data_1, read_data_2, write_data, read_reg_1, read_reg_2, write_reg, signal_reg_write, clk);

Bu modül input olarak gelen read_reg_1 ve read_reg_2 5bitlik adreslerin register bloğunda tuttukları verileri alır ve read_data_1 ve read_data_2 outputlarına atar. Eğer signal_reg_write 1 ise ki bu tasarım için sürekli bir olmalı çünkü sürekli rd registerına yazma yapılıyor ve aynı zamanda clk yükselen kenarda ise çünkü sadece yükselen kenarda registera yazma işlemi yapılacak, input olarak alıdığı write_data verisini yine input olarak aldığı write_reg register adresinin gösterdiği register bloğundaki registera yazar.

2.3 module control_unit (select_bits_ALU, function_code);

Bu modül input olarak aldığı instruction ın son 6 bitlik kısmı olan function_code un bitlerini belirlenen mantıksal denklemlere sokarak 3 bitlik select_bits_ALU değerini hesaplar ve output olarak verir.

2.4 module _32bit_2_1mux (out, a, b, s);

Bu modül input olarak gelen iki 32 bitlik a ve b değerlerinden bir bitlik s değerine göre birini seçip yine 32 bitlik out değerine atar. (S 1 ise b, 0 ise a)

2.5 module _5bit_2_1mux (out, a, b, s);

Bu modül input olarak gelen iki 5 bitlik a ve b değerlerinden bir bitlik s değerine göre birini seçip yine 5 bitlik out değerine atar. (S 1 ise b, 0 ise a)

2.6 module signExtender (out, in);

input [4:0] in ; output [31:0] out;

Bu modüle input olarak gelen 5 bitlik in değerini alır ve output olarak verilecek 32 bitlik out değerinin ilk 5 bitine koyar, out un boşta kalan diğer 27 bitine ise '0' değeri atar. Bu sayede 5 bitlik bir değeri 32 bitlik bir forma değerini değiştirmeden sokmuş oluruz.

2.7 module signExtender_v2(out, in);

input in; output [31:0] out;

Bu modüle input olarak gelen 1 bitlik in değerini alır ve output olarak verilecek 32 bitlik out değerinin ilk bitine koyar, out un boşta kalan diğer 31 bitine ise '0' değeri atar. Bu sayede 1 bitlik bir değeri 32 bitlik bir forma değerini değiştirmeden sokmuş oluruz.

2.8 module assign_32bit(out,in);

input [31:0] in; input [31:0] out;

Bu modülü sadece 32 bitlik iki değerini birbirine kolayca koplayamak için tasarladım. İnput olarak alınan 32 bitlik in değeri bit bit buf() ile kopyalanır ve output olan 32 bitlik değere atanır.

**Derste Alp Hoca bir önceki projede tasarlanan alunun sra sının değiştirilerek srl olarak kullanılabileceğini söylemişti o yüzden sra yerine srl modülüm var. Onu açıklamaya gerek olmadğını düşünüyorum.

3.Modelsim Simülasyon Sonuçları:

10 farklı instruction tip olduğu için 10 farklı test instruction barındıran bir test modülü var. O modülde test edilen instructionlar:

Test instructionları:

00000000100001011000000000100010

00000000100001101000100000100011

00000000111010001001000000100100

00000001001010101001100000100111

00000001001010101010000000100101

00000000100001011010100000100000

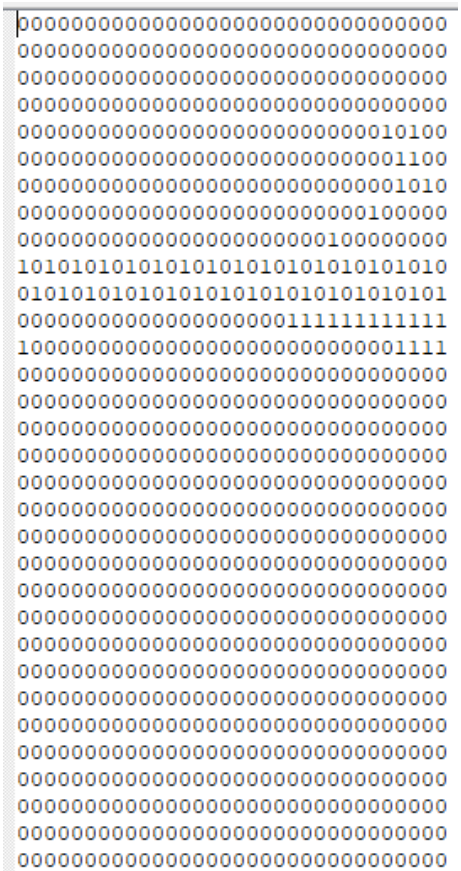
00000001000010001011000000100001

00000000101001111100100000101011

00000000000010111011100111000000

00000000000011001100000100000010

Register bloğunun instructionlar çalıştırılmadan önceki hali:



İlk 4 register boş.

4.register = 20

5.register = 12

6.register = 20

7.register = 32

8.register = 256

9. register = 10101010101010101010101010101010

10. register = 01010101010101010101010101010101

11. register = 00000000000000000000111111111111

12.register = 10000000000000000000000000001111

Değerlerine sahip. 16.register ve ondan sonraki 9 register

İnstrucitonların sonuçlarının yazılması için kullanılacak.

Sırayla 4. ve 5. Registerlar sub işlemi,

4. ve 6. Registerlar subu işlemi;

7. ve 8. Registerlar ile and işlemi;

9. ve 10. Registerlar ile nor işlemi;

9. ve 10. Registerlar ile or işlemi;

4. ve 5. Registerlar ile add işlemi;

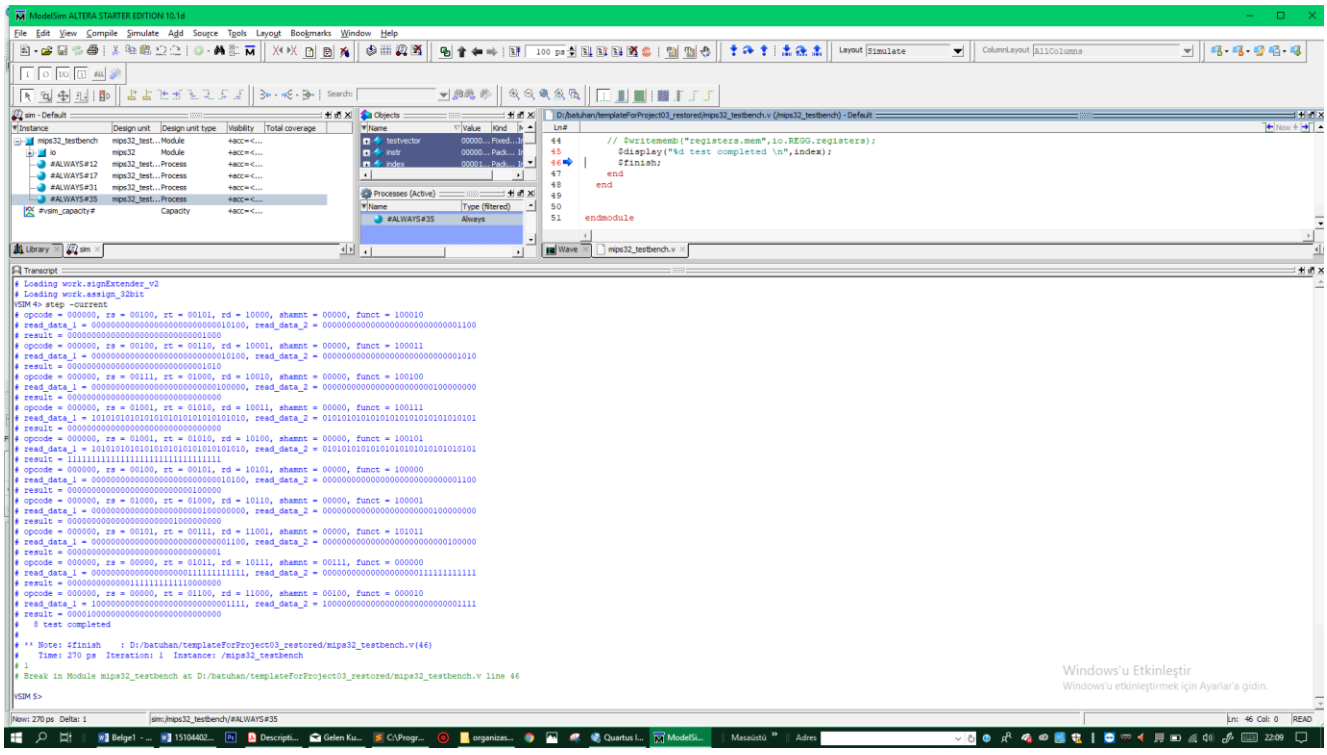
8. ve 8. Registerlar ile addu işlemi;

5. ve 7.registerlar ile sltu işlemi;

11. register ve shamt 7 ile sll işlemi;

12. register ve shamt 4 ile srl işlemi;

Modelsim test sonuçu ekran görüntüsü:



Sadece çıktı olan kesit :

```
# opcode = 000000, rs = 00100, rt = 00101, rd = 10000, shamnt = 00000, funct = 100010
# read_data_1 = 00000000000000000000000000010100, read_data_2 = 00000000000000000000000000001100
# result = 00000000000000000000000000001000
# opcode = 000000, rs = 00100, rt = 00110, rd = 10001, shamnt = 00000, funct = 100011
# read_data_1 = 00000000000000000000000000010100, read_data_2 = 00000000000000000000000000001010
# result = 00000000000000000000000000001010
# opcode = 000000, rs = 00111, rt = 01000, rd = 10010, shamnt = 00000, funct = 100100
# read_data_1 = 00000000000000000000000000100000, read_data_2 = 00000000000000000000000100000000
# result = 00000000000000000000000000000000
# opcode = 000000, rs = 01001, rt = 01010, rd = 10011, shamnt = 00000, funct = 100111
# read_data_1 = 10101010101010101010101010101010, read_data_2 = 01010101010101010101010101010101
# result = 00000000000000000000000000000000
# opcode = 000000, rs = 01001, rt = 01010, rd = 10100, shamnt = 00000, funct = 100101
# read_data_1 = 10101010101010101010101010101010, read_data_2 = 01010101010101010101010101010101
# result = 11111111111111111111111111111111
# opcode = 000000, rs = 00100, rt = 00101, rd = 10101, shamnt = 00000, funct = 100000
# read_data_1 = 00000000000000000000000000010100, read_data_2 = 00000000000000000000000000001100
# result = 00000000000000000000000000100000
# opcode = 000000, rs = 01000, rt = 01000, rd = 10110, shamnt = 00000, funct = 100001
# read_data_1 = 00000000000000000000000100000000, read_data_2 = 00000000000000000000000100000000
# result = 00000000000000000000001000000000
# opcode = 000000, rs = 00101, rt = 00111, rd = 11001, shamnt = 00000, funct = 101011
# read_data_1 = 00000000000000000000000000001100, read_data_2 = 00000000000000000000000000100000
# result = 00000000000000000000000000000001
# opcode = 000000, rs = 00000, rt = 01011, rd = 10111, shamnt = 00111, funct = 000000
# read_data_1 = 00000000000000000000111111111111, read_data_2 = 00000000000000000000111111111111
# result = 00000000000001111111111110000000
# opcode = 000000, rs = 00000, rt = 01100, rd = 11000, shamnt = 00100, funct = 000010
# read_data_1 = 10000000000000000000000000001111, read_data_2 = 10000000000000000000000000001111
# result = 00001000000000000000000000000000
#   8 test completed
```

Register blogunu instructionlar çalıştırıldıktan sonraki hali:

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000010100
00000000000000000000000000001100
00000000000000000000000000001010
00000000000000000000000000100000
00000000000000000000000100000000
10101010101010101010101010101010
01010101010101010101010101010101
00000000000000000000111111111111
10000000000000000000000000001111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000001000
00000000000000000000000000001010
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111111111111111111111
00000000000000000000000000100000
00000000000000000000001000000000
00000000000001111111111110000000
00001000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

16-> sub

17-> subu

18-> and

19->nor

20->or

21->add

22->addu

23->sll

24->srl

25->sltu çıktıları.