

FÜRKAN YILDIZ

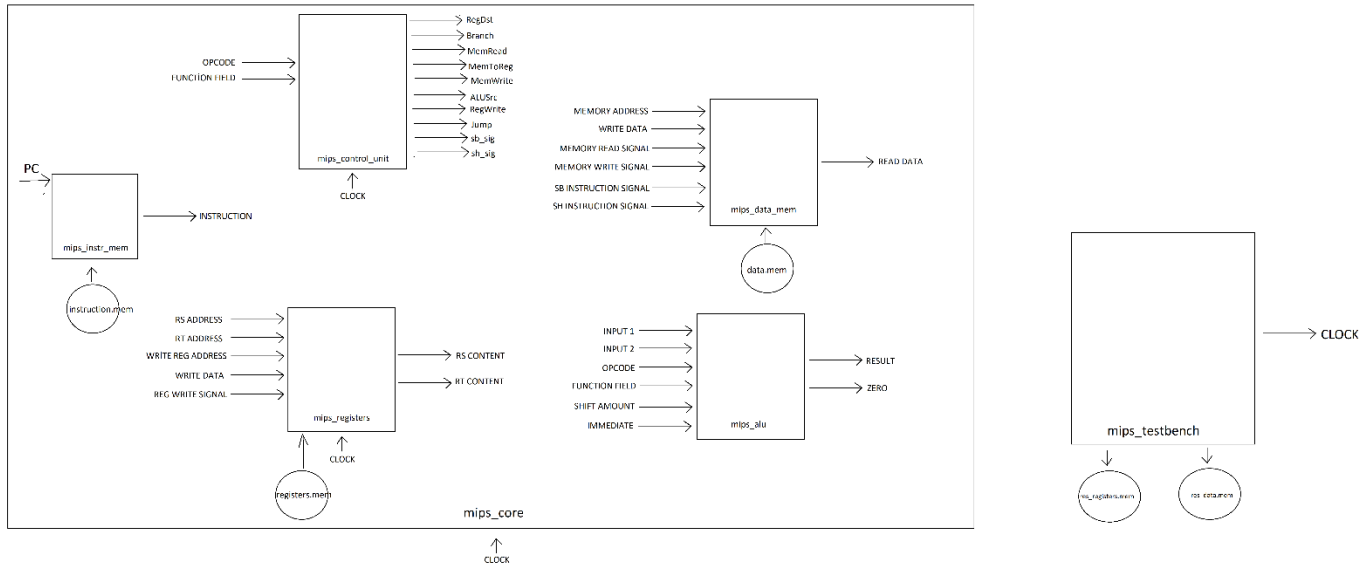
BİL 331 – BİLGİSAYAR ORGANİZASYONU

FİNAL PROJECT REPORT

141044031

1. INTRODUCTION

1.1 BIG PICTURE



mips_core: Testbenchden clock alarak, aldığı bu clock'a göre tüm modülleri çalıştıran ana modüldür. İçerisindeki modüllerin birbirleriyle olan ilişkisini sağlar, input outputlarını birbirlerine bağlar, gerekli zamanda çalıştırır vs.

mips_instr_mem: current PC değerine göre, instruction memory dosyasından(instruction.mem) instruction okuyarak core'ye verir.

mips_registers: Registerden okuma ve registere yazma operasyonları yapılır.

mips_control_unit: Opcode'ye göre instruction için gerekli sinyalleri üretir.

mips_data_mem: Memory de okuma ve yazma operasyonlarını gerçekleştirir.

mips_alu: Aldığı inputlar ile function ve opcodeye göre gerekli işlemleri gerçekleştirir.

mips_testbench: Coreyi çalıştırmak için, instruction sayısına göre clock değişimi yapılır.

instruction.mem: Çalıştırmak istenilen instructionlar bulunur.

registers.mem: Registerlerin contentleri bulunur.

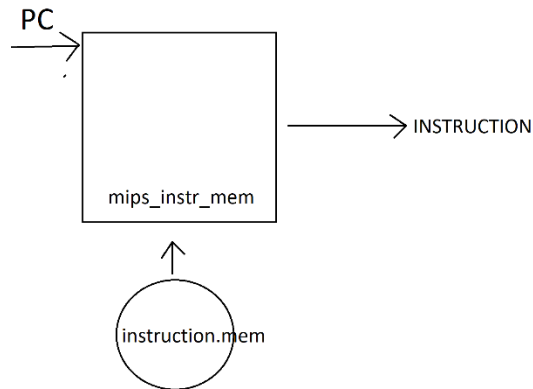
data.mem: Memorylerin contentleri bulunur.

1.2 Life cycle of 1 instruction

Testbench'de clock 0 olarak verilir, mips_core çalışmaya başlar, PC'ye 0 atanır. PC contenti değiştiğinden, PC inputuna bağlı olan mips_instr_mem modülü çalışmaya başlar, bu modül ilk çalıştığında "data.mem" dosyasını okur ve içerisinde tuttuğu arraye kaydeder, daha sonra ise bu array üzerinden gerçekleştirir işlemleri, arraye instructionlar kaydedildikten sonra PC'ye göre instruction output olarak coreye verilir. Core bu instructionu parçalara ayırır, opcode, function field,rs,rt,rd adressleri,immediate,jump address,shift amount olarak. Instruction parçalara ayrıldıktan sonra mips_control_unit çalışır. Bu modül clock sadece 0 olduğunda çalışır (negedge) Opcodeye bakarak, instruction için sinyaller üretilir. Daha sonra ise mips_registers modulu çalışır, bu modül clock 0 iken, kendisine gelen rs ve rt addresslerinin contentlerini output olarak verir. Contentler okunduktan sonra ALU'nun çalışma sırası gelir, ALU'nun 1. Inputu her zaman rs contentidir fakat 2. Inputu ALUSrc sinyaline göre rt ya da immediate olarak değişmektedir, bu sinyalin kontrolü ile ALU'ya rs adresi ve 2. Input veriliyor, ALU'da opcode ve function field dikkate alınarak, yapılması gereken işlemler yapılıyor, branch instructionu ise instruction zero outputu, değilse ALUResult kullanılıyor. Ardından mem_address'e ALUResult'ın sonucu atanarak, mips_data_mem modulu çalıştırılıyor. Bu modülde memoryden okuma ya da memorye yazma sinyallerine bakılarak, gerekli okuma ve yazmalar yapılıyor. Bu modül de işini tamamladığında registre yazılacak bir şeyler olabilmesi sebebi ile (Register write sinyali 1) yazılmak istenilen registerin adresine RegDst==1 ise rd, jr operasyonu yapılıyorsa 31. registerin adresi,diğer durumlarda ise rt yazılır. Registere yazılacak dataya ise, lbu,lhu ve jal operasyonları için onların gerektirdiği şeyler, diğer durumlarda ALUResult yazılır. Registere yazma işlemi ise sadece clock 1 olduğunda yapılır çünkü clock 0 da tüm operasyonlar bitirilir, sonucun doğruluğundan emin olunur ve clock 1 e döndürüldüğünde yazma işlemi gerçekleşir. Yazma işlemide gerçekleştikten sonra instructionun datapathde işi bitmiş olur, PC 1 arttırılarak, sıradaki instruction alınır datapath'e.

2.

2.1 : mips_instr_mem

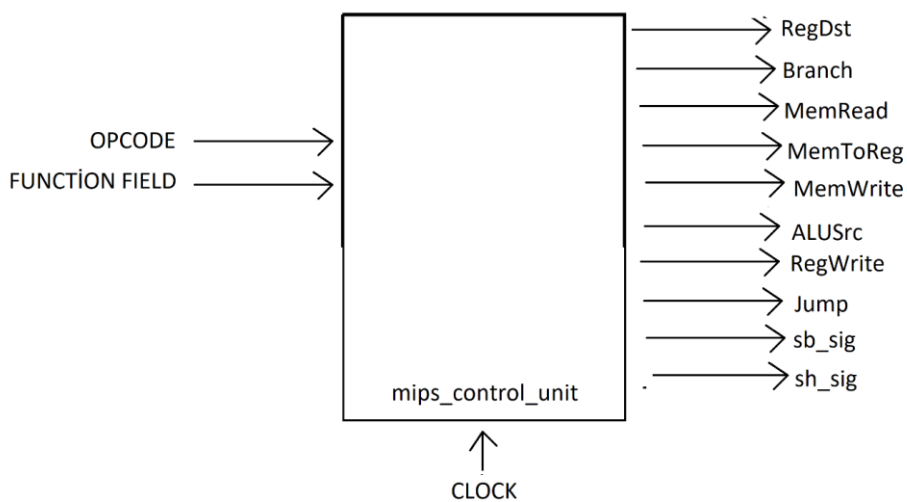


input: Program Counter

output: Instruction

Detailed description: İlk kez çalıştırıldığında “instruction.mem” dosyasını okuyarak, 256 lık bir arraya kaydeder,daha sonra bu array üzerinden işlemleri gerçekleştirir. Input olarak verilen Program Counter’ın değerini arrayin okunmak istenilen indexi kabul ederek, o indexdeki instructionu output olarak verir.

2.2 : mips_control_unit



Inputs: opcode, function field

Outputs: RegDst, Branch, MemRead, MemToReg, MemWrite, ALUSrc, RegWrite, Jump, sb_sig, sh_sig

Detailed description: Aslında sadece opcode alması yeterlidir fakat, sadece “jr” instructionunu içinde doğru sinyaller üretebilmek için function field da verilmiştir. (“jr” R type bir instruction olmasına rağmen diğer R type’ler gibi davranmaz) Opcode kontrolü yaparak, instructionların işlerini doğru şekilde yapabilmeleri için gerekli sinyalleri üretir. Clock 0 olduğunda, sinyaller üretilir(dizaynıma göre, clock 0 iken tüm işlemler yapılır, 1 olduğunda registerlere yazma yapılır.) Bu sinyalleri üretme işlemleri şu şekildedir:

RegDst sinyali sadece R type instruction ise 1 dir, diğer instructionlarda 0’dır.

Branch sinyali sadece beq ve bne instructionları için 1 , diğer tüm instructionlarda 0’dır.

MemRead ve MemToReg sinyalleri sadece lbu, lhu, lw ve ll instructionları için 1 , diğer tüm instructionlarda 0’dır.

MemWrite sinyali, sadece sb,sh ve sw instructionları için 1 , diğer tüm instructionlarda 0’dır.

ALUSrc sinyali, rtype instruction,beq, ve bne instructionları için 0, diğer tüm instructionlarda 1 dir.

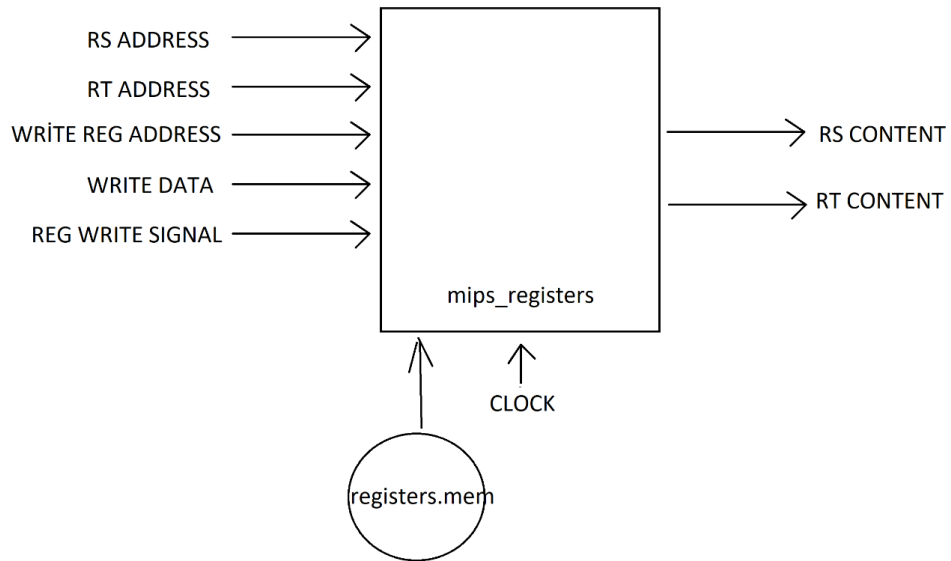
RegWrite sinyali, beq,bne,sb,sh,sw,sc ve j instructionları için 0, diğer tüm instructionlarda 1 dir.

Jump sinyali, sadece j ve jal instructionları için 1 , diğer tüm instructionlarda 0’dır.

sb_sig sinyali, sadece sb instructionu için 1, diğer tüm instructionlarda 0’dır.

sh_sig sinyali, sadece sh instructionu için 1, diğer tüm instructionlarda 0’dır.

2.3 mips_registers



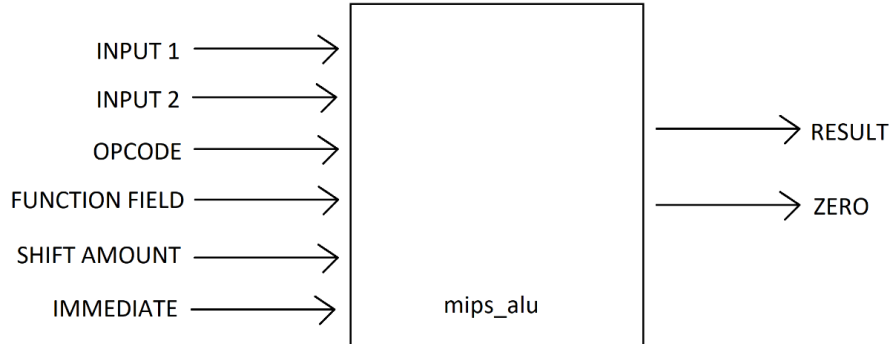
Inputs: RS address, RT address, Write Register Address, Write Data, Register Write Signal

Outputs: Rs Content, RT Content

Detailed description: İlk kez çalıştırıldığında “registers.mem” dosyasını okuyarak, 32 lık bir arraye kaydeder,daha sonra bu array üzerinden işlemleri gerçekleştirir. Negedge clock geldiğinde aldığı rs ve rt adresin contentlerini okuyarak output olarak verir, posedge clock geldiğinde ise “register write signal” i kontrol eder, 1 ise write register address’e, write data’yı yazar. Posedge ve negedge olarak ayırmamın sebebi ise, sistemin negedge

işlemlerini tamamlayıp, verilerin doğruluğunun kesinleştiğinde (posedgede) verileri registre yazmak.

2.4 mips_alu

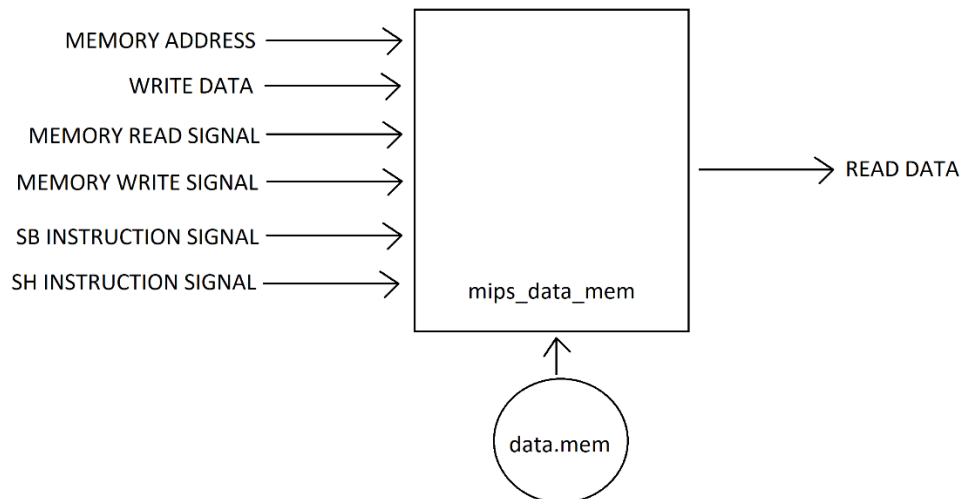


Inputs: Input1(işlem yapılacak 1. değer), Input2(işlem yapılacak 2. değer), Opcode, Function Field, Shift Amount, Immediate

Outputs: Result, Zero

Detailed description: Opcode ve function field kontrolü yaparak, instruction için yapılması gereken işlemi anlar (toplama, çıkartma, and'leme ...) ve bu işlemi gerçekleştirir. Branch instructionları için "zero" outputu kullanılırken, ALU'yu kullanan diğer tüm instructionlar için "result" outputu kullanılır.

2.5 mips_data_mem



Inputs: Memory addressi, Write data, Memory Read Signal, Memory Write Signal, SB instruction signal, SH instruction signal

Outputs: Read data

Detailed description: İlk kez çalıştırıldığında “data.mem” dosyasını okuyarak, 256 lık bir arraye kaydeder, daha sonra bu array üzerinden işlemleri gerçekleştirir. Always bloğu ile çalışır, bu always bloğu Memory addressi, Write data, Memory Read Signal, Memory Write Signal girdilerine bağlıdır, bunlarda değişim meydana geldiğinde çalışır. Çalıştığında memory read ve memory write sinyallerini kontrol eder, memory read sinyali 1 ise, arrayin, memory adresinci indisindeki datayı, read data’ya output olarak verir. Memory write sinyali 1 ise, sb ve sh sinyalleri kontrol edilir, sb sinyali varsa memorynin memory adresinci indisindeki son 8 bit’e yazılacak data yazılır, sh sinyali varsa son 16 bite yazılacak data yazılır, sh veya sb sinyali olmadan sadece write sinyali var ise, o indisin tamamına yazılacak data yazılır.

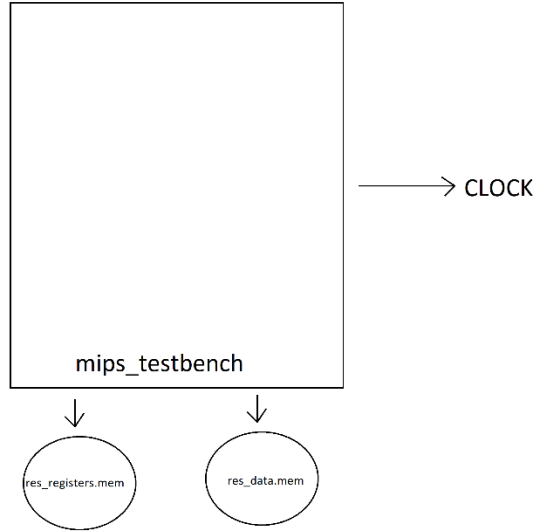
2.6 mips_core

Inputs: Clock

Outputs: Yok.

Detailed description: Ana modüldür. Testbench’den aldığı 0 clock’u ile çalışmaya başlar. Tüm modüllerin birbirleriyle senkron bir şekilde, input outputlarını birbirlerine vererek çalışmalarını sağlar. İlk kez çalıştığında PC’ 0 initialize edilir. PC’de bir değişim olduğundan mips_instr_mem modülünü çalıştırır, bu modülden aldığı instruction outputunu opcode, function field, rs adres, rt adres, rd address, immediate, shift amount, address olarak parçalara ayırır. Aldığı mips_control_unit’e ileterek, negedge clockta instruction için sinyallerin üretilmesini sağlar. Instruction parçalandığında rs ve rt addressleri değişmiş olduğundan register modulu çağırılır ve yine negedge clockta registerden contentleri okunur. Okunan bu contentleri ALUSrc sinyaline bakarak ALU’ya aktarır (rs-rt ya da rs-immediate aktaracağına karar verir ALUSrc ile) ALU’dan ALUResult ve Zero olarak 2 output alır. ALUResult outputunu mips_data_mem modülüne input olarak verir ve memory ile ilgili işlem gerçekleşecekse orada gerçekler, memoryOutputu alınır bu unitten. Registerlere yazma işlemi varsa bunu gerçekleştirmek için yazılacak register adresine RegDst sinyaline bakılarak, rt, rd adresleri ya da jr instructionu için 31. Registerin adresi verilir. Yazılacak data için ise, lbu, lhu ve jal instructionları için onların gerektirdikleri (örneğin lbu için {24'b0, memOutput[7:0]}) , diğer instructionlar için ise ALUResult yazılır. 0 clock ile tüm işlemler sonlanır ve clock değişimi gerçekleşir, clock değişiminde Register Write sinyali 1 ise daha önceden ayarladığımız inputlar doğrultusunda registre yazma işlemi gerçekleşir. Bu işlemin ardından instruction’un datapath’a durmasına artık gerek yoktur, tüm işlemleri bitirilmiştir. PC değişip, yeni instruction okunmalıdır. Bu değişim ise controllerle yapılır, j yada jal instructionu için PC, jumpAdresse eşitlenir, jr instructionu için rs’in contentine eşitlenir, branch instructionları için (ALU’nun Zero outputu da 1 olmak zorunda bunun gerçekleşebilmesi için) PC’ye PC + signExtend ataması yapılır. (normalde branchAdress ataması yapılması gerekir fakat bizim instructionlarımız 4’un katı olarak artmadıkları için 4 ile çarpmaya gerek yoktur, bu yüzden signExtend ataması yapıldı.) Diğer instructionlar ise PC 1 arttırılır ve yeni instruction okunarak her şey en baştan başlar.

2.7 mips_testbench



Inputs: Yok

Outputs: Clock

Detailed description: mips_core modulunun çalışması için ona clock verir. Verdiği clock sayısı instruction sayısının 2 katı'dır.

2.8

3. RESULTS

3.1 Test1

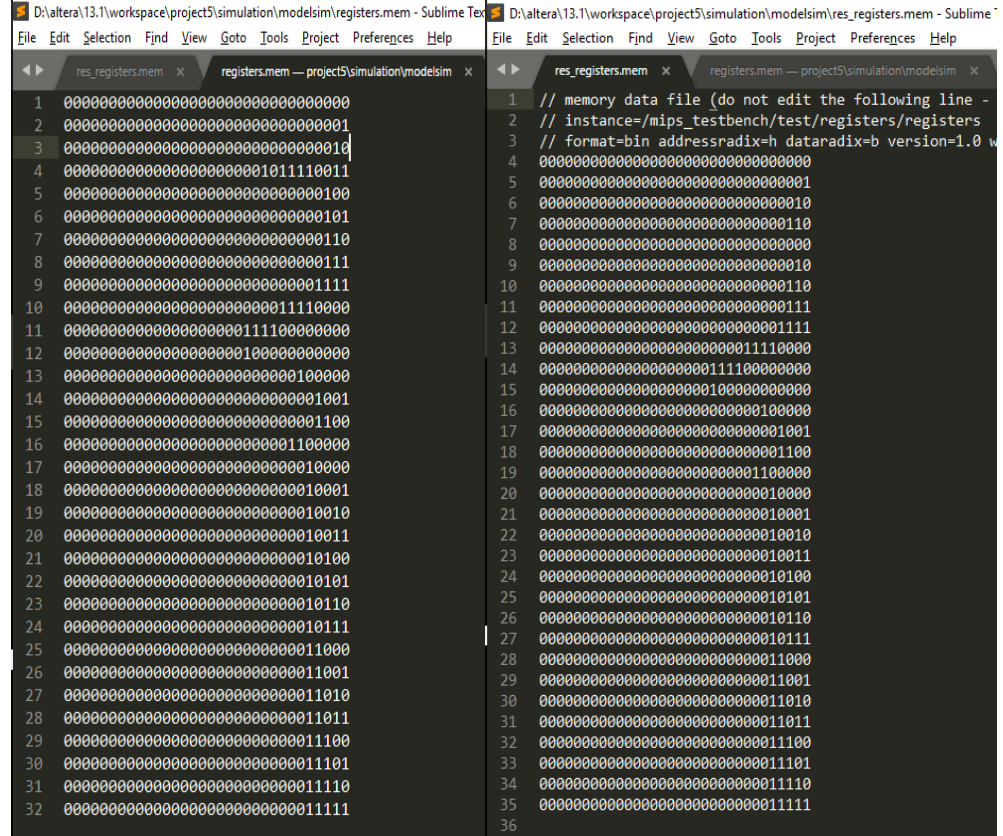
```
add 2 4 3 ,reg(2) + reg(4) -> reg(3) , 00000000010001000001100000100000
andi 7 4 010 , reg(7) & imm(0) -> reg(4) , 00110000111001000000000000000000
bne 3 4 210 ,if(reg(3)-reg(4) != 0) then Brach, 00010100011001000000000000000010
ori 10 12 1110 , reg(10) | imm(11)->reg(12),001101010100111000000000000001011
beq 0 4 210 , if(reg(0)-reg(4) == 0)then Brach, 00010000000001000000000000000010
lw 11 13 210 , mem(11 + imm(2)) -> reg(13), 10001101011011010000000000000010
lw 12 5 210 , mem(reg(12) + imm(2))->reg(5), 10001101100001010000000000000010
```

Beklentiler:

İkinci ve dördüncü registerlerin toplamı olan 6₁₀ yı 3. Registere yazmasını,
Yedinci register ile 0 'ı andleyip sonucu olan 0'ı 4. Registere yazmasını
Ucuncu register ile dördüncü registelerin contentlerinin eşit olmadığından 2 adres
ileri atlayıp ori yi geçerek beq instructionunu yapmasını
Beq instructionunda eşitlikten dolayı 2 adres atlayarak son lw instructionu
gerçekleştirmesini ve memory'nin 34. İndisindeki(reg (12) = 32) 14'ü 5. Registere
yazmasını

Sonuç:

Soldaki resim registerlerin initialize değerleri, sağdaki son halleri.



```
1 00000000000000000000000000000000
2 00000000000000000000000000000001
3 00000000000000000000000000000010
4 00000000000000000000000000000011
5 00000000000000000000000000000100
6 00000000000000000000000000000101
7 00000000000000000000000000000110
8 00000000000000000000000000000111
9 00000000000000000000000000000111
10 00000000000000000000000000000111
11 00000000000000000000000000000111
12 00000000000000000000000000000111
13 00000000000000000000000000000111
14 00000000000000000000000000000111
15 00000000000000000000000000000111
16 00000000000000000000000000000111
17 00000000000000000000000000000111
18 00000000000000000000000000000111
19 00000000000000000000000000000111
20 00000000000000000000000000000111
21 00000000000000000000000000000111
22 00000000000000000000000000000111
23 00000000000000000000000000000111
24 00000000000000000000000000000111
25 00000000000000000000000000000111
26 00000000000000000000000000000111
27 00000000000000000000000000000111
28 00000000000000000000000000000111
29 00000000000000000000000000000111
30 00000000000000000000000000000111
31 00000000000000000000000000000111
32 00000000000000000000000000000111

1 // memory data file (do not edit the following line -
2 // instance=mips_testbench/test/registers/registers
3 // format=bin addressradix=h data radix=b version=1.0 w
4 00000000000000000000000000000000
5 00000000000000000000000000000001
6 00000000000000000000000000000010
7 00000000000000000000000000000110
8 00000000000000000000000000000111
9 00000000000000000000000000000111
10 00000000000000000000000000000111
11 00000000000000000000000000000111
12 00000000000000000000000000000111
13 00000000000000000000000000000111
14 00000000000000000000000000000111
15 00000000000000000000000000000111
16 00000000000000000000000000000111
17 00000000000000000000000000000111
18 00000000000000000000000000000111
19 00000000000000000000000000000111
20 00000000000000000000000000000111
21 00000000000000000000000000000111
22 00000000000000000000000000000111
23 00000000000000000000000000000111
24 00000000000000000000000000000111
25 00000000000000000000000000000111
26 00000000000000000000000000000111
27 00000000000000000000000000000111
28 00000000000000000000000000000111
29 00000000000000000000000000000111
30 00000000000000000000000000000111
31 00000000000000000000000000000111
32 00000000000000000000000000000111
```

3.2 Test2

```
slt 2 4 3 reg(3) <-(reg(2)<reg(4))?1:0, 00000000010001000001100000101010
j 510 00001000000000000000000000000000101
lw 11 13 210,mem(reg(11)+imm(2))->reg(13),10001101011011010000000000000010
lw 11 14 210,mem(reg(11)+imm(2))->reg(14),10001101110011010000000000000010
sw 11 15 210 mem(reg(11)+imm(2))<-reg(15),10101101111011010000000000000010
slti 12 5 2310, 001010011000010100000000000010111
sw 14 13 210,mem(reg(14)+imm(2))<-reg(13),10101101110011010000000000000010
```

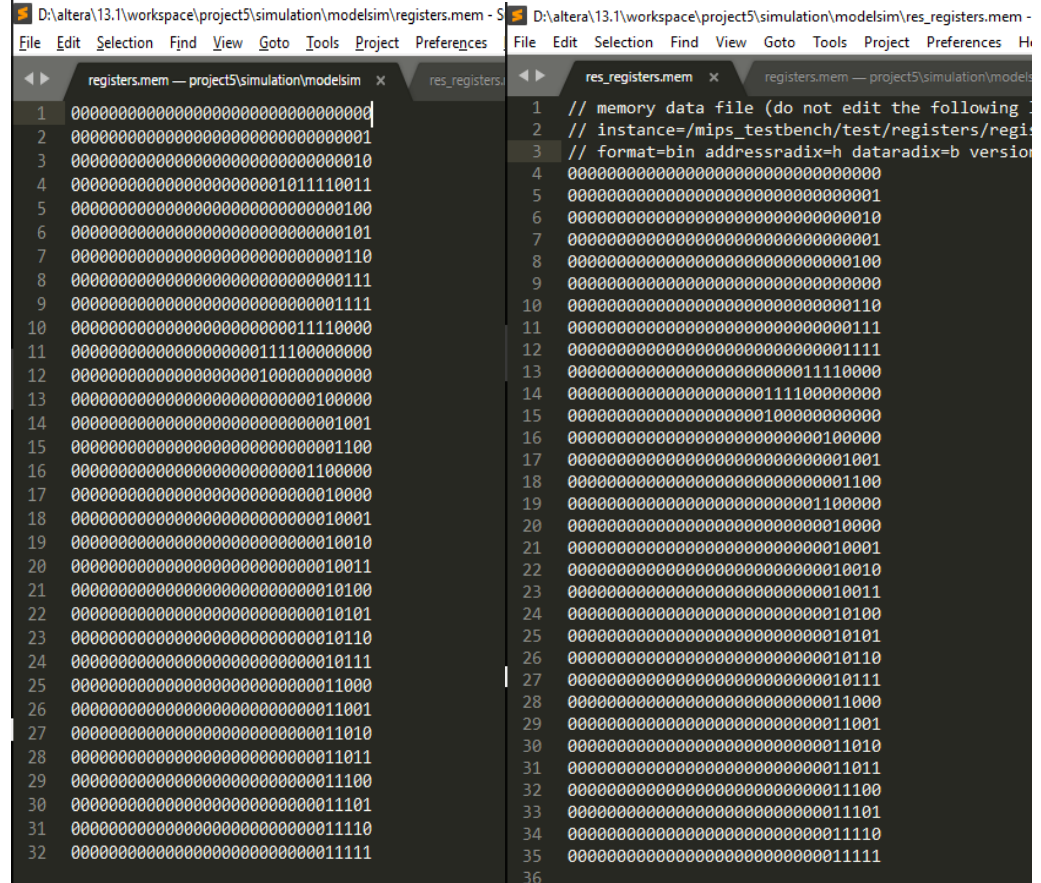
Beklentiler:

Slt işleminin sonucu olan 1'i 3. Registere yazacak.

PC'yi 5 yapacak ve direk slti instructionuna atlayacak. 12. Registerde 32 yazdığından 32>23, 5. Registere 0 yazılacak.

Sw işlemini yapacak. Reg(14) == 12 olduğundan 13. Registerin contenti, 14. Data memory indisine yazılacak.

Sonuçlar:



```
D:\altera\13.1\workspace\project5\simulation\modelsim\registers.mem - S
File Edit Selection Find View Goto Tools Project Preferences
registers.mem — project5\simulation\modelsim x
1 00000000000000000000000000000000
2 00000000000000000000000000000001
3 00000000000000000000000000000010
4 00000000000000000000000000000011
5 00000000000000000000000000000100
6 00000000000000000000000000000101
7 00000000000000000000000000000110
8 00000000000000000000000000000111
9 00000000000000000000000000000111
10 00000000000000000000000000000111
11 00000000000000000000000000000111
12 00000000000000000000000000000100
13 00000000000000000000000000000100
14 00000000000000000000000000000101
15 00000000000000000000000000000110
16 00000000000000000000000000000110
17 00000000000000000000000000000100
18 00000000000000000000000000000101
19 00000000000000000000000000000101
20 00000000000000000000000000000101
21 00000000000000000000000000000100
22 00000000000000000000000000000101
23 00000000000000000000000000000110
24 00000000000000000000000000000111
25 00000000000000000000000000000100
26 00000000000000000000000000000101
27 00000000000000000000000000000110
28 00000000000000000000000000000111
29 00000000000000000000000000000100
30 00000000000000000000000000000101
31 00000000000000000000000000000110
32 00000000000000000000000000000111

D:\altera\13.1\workspace\project5\simulation\modelsim\res_registers.mem -
File Edit Selection Find View Goto Tools Project Preferences H
res_registers.mem x
1 // memory data file (do not edit the following
2 // instance=mips_testbench/test/registers/regi
3 // format=bin addressradix=h dataradix=b version
4 00000000000000000000000000000000
5 00000000000000000000000000000001
6 00000000000000000000000000000010
7 00000000000000000000000000000001
8 00000000000000000000000000000010
9 00000000000000000000000000000000
10 00000000000000000000000000000011
11 00000000000000000000000000000011
12 00000000000000000000000000000011
13 00000000000000000000000000000011
14 00000000000000000000000000000011
15 00000000000000000000000000000010
16 00000000000000000000000000000010
17 00000000000000000000000000000010
18 00000000000000000000000000000010
19 00000000000000000000000000000010
20 00000000000000000000000000000010
21 00000000000000000000000000000010
22 00000000000000000000000000000010
23 00000000000000000000000000000010
24 00000000000000000000000000000010
25 00000000000000000000000000000010
26 00000000000000000000000000000010
27 00000000000000000000000000000010
28 00000000000000000000000000000010
29 00000000000000000000000000000010
30 00000000000000000000000000000010
31 00000000000000000000000000000010
32 00000000000000000000000000000010
33 00000000000000000000000000000010
34 00000000000000000000000000000010
35 00000000000000000000000000000010
36
```

D:\altera\13.1\workspace\project5\simulation\modelsim\data.mem

File Edit Selection Find View Goto Tools Project Preferences

data.mem x res_registers.mem x registers.mem

1 00000000000000000000000000000000

2 00000000000000000000000000000001

3 00000000000000000000000000000010

4 00000000000000000000000000000011

5 00001000010000100000000000000111

6 00000000000000000000000000000101

7 00000000000000000000000000000110

8 00000000000000000000000000000111

9 00000000000000000000000000001000

10 00000000000000000000000000001001

11 00000000000000000000000000001010

12 00000000000000000000000000001011

13 00000000000000000000000000001100

14 00000000000000000000000000001101

15 00000000000000000000000000001110

16 00000000000000000000000000001111

17 00000000000000000000000000001000

18 00000000000000000000000000001001

19 00000000000000000000000000001010

20 00000000000000000000000000001011

21 00000000000000000000000000001010

22 00000000000000000000000000001010

23 00000000000000000000000000001011

24 00000000000000000000000000001011

25 00000000000000000000000000001100

26 00000000000000000000000000001100

27 00000000000000000000000000001101

28 00000000000000000000000000001101

29 00000000000000000000000000001110

30 00000000000000000000000000001110

31 00000000000000000000000000001111

32 00000000000000000000000000001111

33 00000000000000000000000000000000

34 00000000000000000000000000000001

35 00000000000000000000000000000010

36 00000000000000000000000000000011

D:\altera\13.1\workspace\project5\simulation\modelsim\res_data.mem -

File Edit Selection Find View Goto Tools Project Preferences

res_data.mem x data.mem x res_registers.mem x

1 // memory data file (do not edit the followi

2 // instance=/mips_testbench/test/Mem/data_me

3 // format=bin addressradix=h data radix=b ver

4 00000000000000000000000000000000

5 00000000000000000000000000000001

6 00000000000000000000000000000010

7 00000000000000000000000000000011

8 00001000010000100000000000000111

9 00000000000000000000000000000101

10 00000000000000000000000000000110

11 00000000000000000000000000000111

12 00000000000000000000000000001000

13 00000000000000000000000000001001

14 00000000000000000000000000001010

15 00000000000000000000000000001011

16 00000000000000000000000000001100

17 00000000000000000000000000001101

18 00000000000000000000000000001001

19 00000000000000000000000000001111

20 00000000000000000000000000001000

21 00000000000000000000000000001001

22 00000000000000000000000000001010

23 00000000000000000000000000001011

24 00000000000000000000000000001010

25 00000000000000000000000000001011

26 00000000000000000000000000001011

27 00000000000000000000000000001011

28 00000000000000000000000000001100

29 00000000000000000000000000001101

30 00000000000000000000000000001101

31 00000000000000000000000000001101

32 00000000000000000000000000001110

33 00000000000000000000000000001111

34 00000000000000000000000000001111

35 00000000000000000000000000001111

36 00000000000000000000000000000000

37 00000000000000000000000000000001