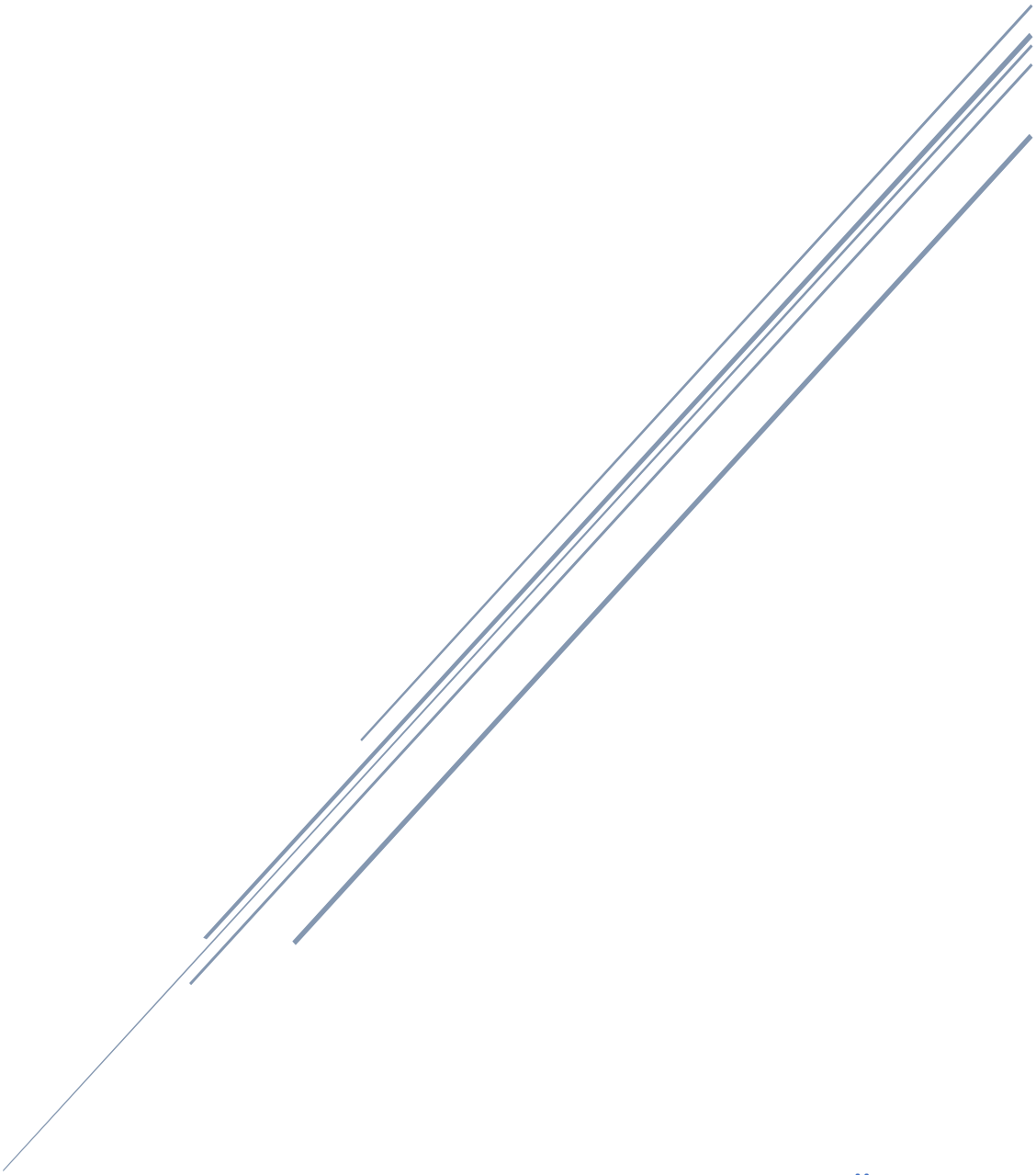
BİL – 331 BİLGİSAYAR ORGANİZASYONU

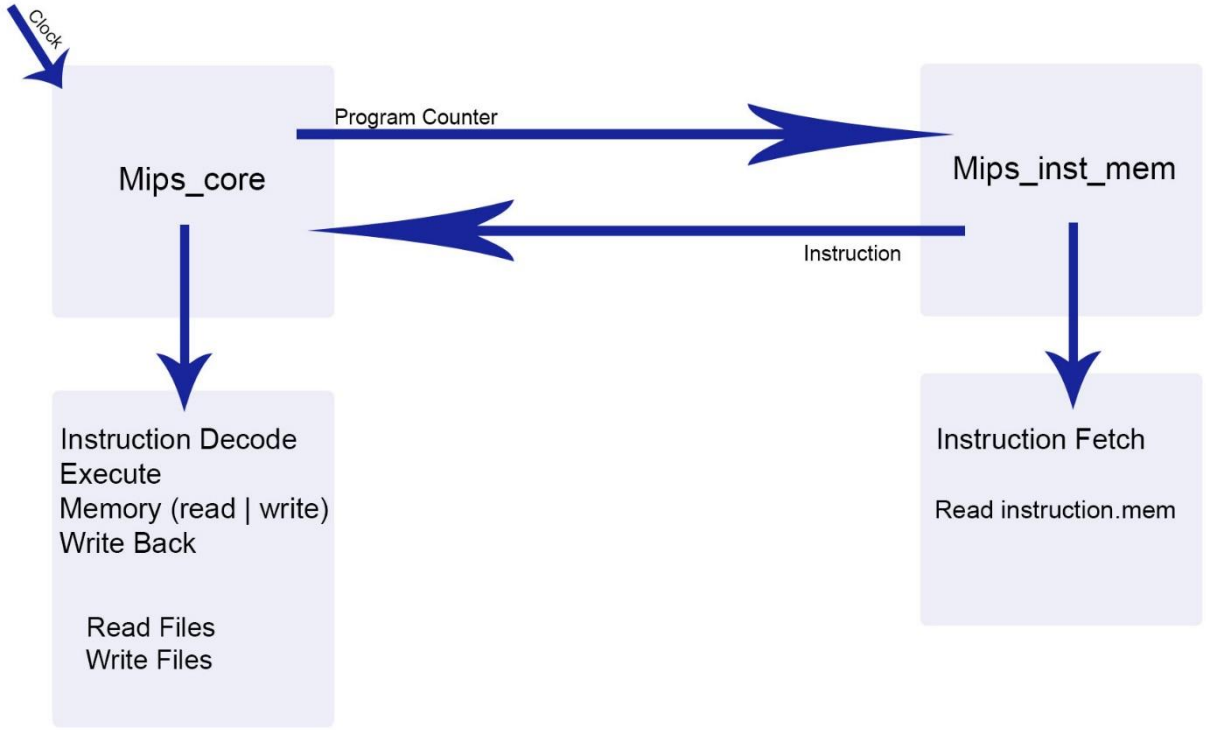
Final Proje Raporu



Lütfullah TÜRKER
141044050

1. INTRODUCTION

1.1 Big Picture



<u>NOT :</u> EĞER MODELSİMDE SİMÜLASYON YAPARKEN SONUÇLAR HEP XXXX ŞEKLİNDE VE "NO SUCH FILE OR DIRECTORY" HATASI VERİYORSA .MEM DOSYALARINI PROJE KLASÖRÜNÜN İÇİNDEKİ SIMULATİON/MODELSIM İÇİNE ATIP TEKRAR SİMÜLASYONU BAŞLATINIZ.

1.2 Bir instruction'ın yaşam döngüsü

Her bir instruction 2 clock sürecektir. **Simülasyonda sonuç çıktısında Clock değerinin 1 olduğu durumdaki çıktılar sonuçlardır.Clock: 1 yazan çıktıları dikkate alınız.**

Programın genel çalışma düzeni ➔

Clock değeri başlangıçta 0 oluyor ve PC değeri mips_inst_mem modülüne giderek fetch işlemi yapılıp instruction mips_core a veriliyor.Daha sonra mips_core modülünde always @(negedge clock) bloğuna giriyor ve instruction decode ve register file işlemleri yapılıyor.Ardından clock değeri 1 oluyor ve always @(posedge clock) bloğuna giriyor ve Execute,Memory read write ve Write Back işlemleri gerekli sinyallere göre yapılıyor ve instruction tamamlanıyor.

Mips_inst_mem modülü mips_core modülünden gelen PC değerine göre gerekli instruction ı instruction memory dosyasından alıp mips_core da decode edilip kullanılması için output olarak geri gönderir.

Mips_core modülü CPU modülüdür. CPU nun merkezi görevini görür.Pipeline stage lerine göre Instruction fetch stage i dışındaki diğer 4 stage gelen clock değerine göre sırasıyla yapılır.

Ayrıntılara sonraki bölümlerde değinilecektir.

1.3 Yapılan Instructionlar

Moodle da bulunan all.pdf dosyasındaki MIPS Reference Data daki Tüm core instructionlar ve ek olarak sra , xori ve xor instructionları implement edilmiştir.

2. METHOD

2.1 Mips_core modülü

Inputlar : clock

Modülün başında tüm değişkenleri tanımlıyoruz.RS,RT ve RD değişkenlerini tanımlarken signed olarak tanımlıyoruz ve unsigned instructionlar için $unsigned fonksiyonunu kullandım.
Sinyallerimizi,PC yi ve write_data yı ilklendiriyoruz. ve registerleri ve memory içeriğini çekmek için registers.mem ve data.mem dosyalarımızı okuyup arraylerimizi dolduruyoruz.
PC değerine göre instruction çeken modülümüzü çağrıyoruz.
Daha sonra always @(negedge clock) bölümüne giriyoruz. Bu always bloğu içinde instruction decode ve register file işlemlerini yapıyorum. Negedge clock yapma sebebim clock 0 olduğu durumda bu always kısmını yapıyor.Ve clock 1 olduğunda yapmıyoruz. Atamalardan bazılarını <= yerine = yaptım çünkü = olan atamalar aşağıda başka bir atama için kullanılacak değişkenlerin ataması. Başka bir değişkenin o değişkeni aynı blokta kullanabilmesi için = ile atama yapmalıyız çünkü <= yapınca always bittiğinde atama yapıldığı için atama yapılmadan önceki değerini alacaktı.

```
// PC değerine göre instruction çeken modülümüzü çağırıyoruz.
mips_instr_mem instructionmem(instruction, PC);

// negedge clock yapma sebebim clock 0 olduğu durumda bu always kısmını yapıyor.Ve clock 1 olduğunda yapmıyoruz.
// bu always bloğu içinde instruction decode ve register file işlemlerini yapıyorum.
always @(negedge clock)
begin
   if (clock === 0)begin
      opCode <= instruction[31:26];
      funct <= instruction[5:0];
      read_reg_1 = instruction[25:21];//rs
      read_reg_2 = instruction[20:16]; //rt
      immediate = instruction[15:0]; // immediate
      jmpAdress <= instruction[25:0]; // J type adress
      shamt <= instruction[10:6];
      signExtImm[31:0] =  {{16{immediate[15]}}, immediate[15:0]}; //SignExtendImmediate
      zeroExtImm[31:0] <=  {{16{1'b0}}, immediate[15:0]}; // ZeroExtendImmediate
      branchAdress[31:0] <= {{14{immediate[15]}}, immediate[15:0], {2'b0}}; // BranchAdress
      read_data_1 = registers[read_reg_1];
      read_data_2 <= registers[read_reg_2];
      mem_address <= read_data_1 + signExtImm ;
// atamalardan bazılarını <= yerine  = yaptım çünkü = olan atamalar aşağıda başka bir atama için kullanılacak değ
// başka bir değişkenin o değişkeni kullanabilmesi için = ile atama yapmalıyız çünkü <= yapınca always bittiğinde
   end
end
```

Bu kısımda instruction parçalara ayrılıp gerekli tüm değişkenler atandıktan sonra clock 1 oluyor ve aşağıdaki always @(posedge clock) bloğuna geçiyor.
Bu always bloğu da posedge olduğu için clock değerinin 1 olduğu durumda çalışacak.
Bu blokda Execute,Memory read write ve Write Back işlemleri yapılıyor.
write_data ya yazma işlemi yapılan instructionlarda signal_reg_write ı 1 yapiyoruz ki registera yazma sinyalini yollayıp gerekli if kısmında yazma işlemi yapılsın.
Sinyalleri başlangıçta yazma işlemi yapmayacağımız için 0 yaptım.
R type olmasi icin opcode 0 olmali ve bunu kontrol edip tipe göre islem yaptım.
Tüm instructionların implementasyonunu tek tek anlatmaya gerek duymuyorum koda bakınca anlaşılır olduğunu düşünüyorum.
opCode a göre tüm instructionları tanımladıktan sonra gelen instruction jump jal veya jr instructionu değilse PC yi 1 arttırıyorum.
instructionın R type veya I type olma durumuna göre RD registeri yerine RT registerine yazılacağı için regDst sinyalinin görevini if else bloklarında kontrol edip write_reg registerimi instruction tipine göre RD veya RT olarak seçiyorum.Seçimi opCode un 0 olup olmamasına göre anlayabiliyorum.
write_reg registeri belirlendikten sonra signal_reg_write sinyali 1 ise registere yazma işlemi yapılacak yani write back stage i yapılacak ve always bloğu sonlanacak.

```
      if (opCode != 6'b000010 && funct != 6'b001000 && opCode != 6'b000011)begin
         PC = PC+1'b1;
      end
// instructionın R type veya I type olma durumuna göre RD registeri yerine RT registerine yazılacağı için
// regDst sinyalinin görevini aşağıdaki if else bloklarında kontrol edip write_reg registerimi instruction
// tipine göre RD veya RT olarak seçiyorum.Seçimi opCode un 0 olup olmamasına göre anlayabiliyorum.
      if (opCode != 6'b000000)begin
         write_reg = instruction[20:16] ; //rt
      end
      else begin
         write_reg = instruction[15:11]; // rd
      end
// write_reg registeri belirlendikten sonra signal_reg_write sinyali 1 ise registere yazma işlemi yapılacak
      if (signal_reg_write && clock) begin
         registers[write_reg] = write_data;
      end
   end
end
```

Arkasından gelen son always bloğunda ise her clockda oluşan sonuçları gerekli dosyalara yazma ve ekrana print etme işlemlerini yapıyorum.

Yani programın çalışma düzeni ==> clock değeri başlangıçta 0 oluyor ve instruction modülünden fetch edilip always @(negedge clock) bloğuna giriyor ve instruction decode ve register file işlemleri yapılıyor.Ardından clock değeri 1 oluyor ve always @(posedge clock) bloğuna giriyor ve Execute,Memory read write ve Write Back işlemleri gerekli sinyallere göre yapılıyor ve instruction tamamlanıyor.Son olarak dosyalara yazma ve ekrana basma işlemi yapılıp sonraki instructiona geçiliyor.

2.2 Mips_inst_mem modülü

Inputlar : program_counter
Outputlar : instruction

Instr_mem adında instructionları tuttuğumuz 256 kapasiteli instruction arrayi oluşturdum. Programın başında modül çağrıldığı an instructionlar dosyadan çekilip arraye yazılıyor.Mips_core da belirlenen PC değerine göre gerekli instruction arrayden çekiliyor.Ve output olarak geri dönüyor.

2.3 Mips_testbench modülü

Başlangıçta clock değişkeni oluşturulup ilk değeri 0 olarak atanıyor.Ve test kısmında instruction sayısı * 2 -1 kadar #50 clock=~clock; yazıyoruz.

3. RESULT

Testbench Sonuçları

```
İnstructions ////////////////////////////

000000 01111 01110 00111 00000 100000  // add
000000 10000 01111 01000 00000 100000  // add
000000 10000 10001 10010 00000 100000  // add
001000 10011 10011 0000000000000001   // addi
100100 10000 01000 0000000000000010   // lbu
100101 10001 01001 0000000000000100   // lhu
000000 10100 10101 01011 00000 101010 // slt
001010 10110 01100 0000000000010000   // slti
101000 10111 01101 0000000000100000   // sb
101001 11000 11001 0000000010000000   // sh
000000 10000 10001 10010 00000 100010 // sub
```

Yukarıdaki instructionlar ile test yaptım.(Template klasöründe verdiğiniz instructionlar)

Aşağıda ModelSim deki test sonuçlarını ayrıntılarıyla görebilirsiniz.Sonuçlarda clock : 1 yazan kısımdaki sonuçları dikkate alınız.

```
VSIM 8> step -current
# Opcode: 000000 , funct: 100000 , RS: 00000000000000000000000001100000, RT: 00000000000000000000000000001100, RD: 00000000000000000000000001101100 , PC: 00000000000000000000000000000000  signal: 1, clock:
1 , RD adress: 00111 , RS adress: 01111 , RT adress: 01110 , SignExtImm: 00000000000000000000111000000100000
# Opcode: 000000 , funct: 100000 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000001100000, RD: 00000000000000000000000001101100 , PC: 00000000000000000000000000000000  signal: 1, clock:
0 , RD adress: 00111 , RS adress: 10000 , RT adress: 01111 , SignExtImm: 00000000000000000100000000100000
# Opcode: 000000 , funct: 100000 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000001100000, RD: 00000000000000000000000001110000 , PC: 00000000000000000000000000000001  signal: 1, clock:
1 , RD adress: 01000 , RS adress: 10000 , RT adress: 01111 , SignExtImm: 00000000000000000100000000100000
# Opcode: 000000 , funct: 100000 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000000010001, RD: 00000000000000000000000001110000 , PC: 00000000000000000000000000000001  signal: 1, clock:
0 , RD adress: 01000 , RS adress: 10000 , RT adress: 10001 , SignExtImm: 11111111111111111001000000100000
# Opcode: 000000 , funct: 100000 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000000010001, RD: 00000000000000000000000000100001 , PC: 00000000000000000000000000000010  signal: 1, clock:
1 , RD adress: 10010 , RS adress: 10000 , RT adress: 10001 , SignExtImm: 11111111111111111001000000100000
# Opcode: 001000 , funct: 000001 , RS: 00000000000000000000000000010011, RT: 00000000000000000000000000010011, RD: 00000000000000000000000000100001 , PC: 00000000000000000000000000000010  signal: 1, clock:
0 , RD adress: 10010 , RS adress: 10011 , RT adress: 10011 , SignExtImm: 00000000000000000000000000000001
# Opcode: 001000 , funct: 000001 , RS: 00000000000000000000000000010011, RT: 00000000000000000000000000010011, RD: 00000000000000000000000000010100 , PC: 00000000000000000000000000000011  signal: 1, clock:
1 , RD adress: 10011 , RS adress: 10011 , RT adress: 10011 , SignExtImm: 00000000000000000000000000000001
# Opcode: 100100 , funct: 000010 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000001110000, RD: 00000000000000000000000000010100 , PC: 00000000000000000000000000000011  signal: 1, clock:
0 , RD adress: 10011 , RS adress: 10000 , RT adress: 01000 , SignExtImm: 00000000000000000000000000000010
# Opcode: 100100 , funct: 000010 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000001110000, RD: 00000000000000000000000000010010 , PC: 00000000000000000000000000000100  signal: 1, clock:
1 , RD adress: 01000 , RS adress: 10000 , RT adress: 01000 , SignExtImm: 00000000000000000000000000000010
# Opcode: 100101 , funct: 000100 , RS: 00000000000000000000000000010001, RT: 00000000000000000000000011110000, RD: 00000000000000000000000000010010 , PC: 00000000000000000000000000000100  signal: 1, clock:
0 , RD adress: 01000 , RS adress: 10001 , RT adress: 01001 , SignExtImm: 00000000000000000000000000000100
# Opcode: 100101 , funct: 000100 , RS: 00000000000000000000000000010001, RT: 00000000000000000000000011110000, RD: 00000000000000000000000000010101 , PC: 00000000000000000000000000000101  signal: 1, clock:
1 , RD adress: 01001 , RS adress: 10001 , RT adress: 01001 , SignExtImm: 00000000000000000000000000000100
# Opcode: 000000 , funct: 101010 , RS: 00000000000000000000000000010100, RT: 00000000000000000000000000010101, RD: 00000000000000000000000000010101 , PC: 00000000000000000000000000000101  signal: 1, clock:
0 , RD adress: 01001 , RS adress: 10100 , RT adress: 10101 , SignExtImm: 00000000000000000101100000101010
# Opcode: 000000 , funct: 101010 , RS: 00000000000000000000000000010100, RT: 00000000000000000000000000010101, RD: 00000000000000000000000000000001 , PC: 00000000000000000000000000000110  signal: 1, clock:
1 , RD adress: 01011 , RS adress: 10100 , RT adress: 10101 , SignExtImm: 00000000000000000101100000101010
# Opcode: 001010 , funct: 010000 , RS: 00000000000000000000000000010110, RT: 00000000000000000000000000100000, RD: 00000000000000000000000000000001 , PC: 00000000000000000000000000000110  signal: 1, clock:
0 , RD adress: 01011 , RS adress: 10110 , RT adress: 01100 , SignExtImm: 00000000000000000000000000010000
# Opcode: 001010 , funct: 010000 , RS: 00000000000000000000000000010110, RT: 00000000000000000000000000100000, RD: 00000000000000000000000000000000 , PC: 00000000000000000000000000000111  signal: 1, clock:
1 , RD adress: 01100 , RS adress: 10110 , RT adress: 01100 , SignExtImm: 00000000000000000000000000010000
# Opcode: 101000 , funct: 100000 , RS: 00000000000000000000000000010111, RT: 00000000000000000000000000001001, RD: 00000000000000000000000000000000 , PC: 00000000000000000000000000000111  signal: 1, clock:
0 , RD adress: 01100 , RS adress: 10111 , RT adress: 01101 , SignExtImm: 00000000000000000000000000100000
# Opcode: 101000 , funct: 100000 , RS: 00000000000000000000000000010111, RT: 00000000000000000000000000001001, RD: 00000000000000000000000000000000 , PC: 00000000000000000000000000001000  signal: 0, clock:
1 , RD adress: 01101 , RS adress: 10111 , RT adress: 01101 , SignExtImm: 00000000000000000000000000100000
# Opcode: 101001 , funct: 000000 , RS: 00000000000000000000000000011000, RT: 00000000000000000000000000011001, RD: 00000000000000000000000000000000 , PC: 00000000000000000000000000001000  signal: 0, clock:
0 , RD adress: 01101 , RS adress: 11000 , RT adress: 11001 , SignExtImm: 00000000000000000000000010000000
# Opcode: 101001 , funct: 000000 , RS: 00000000000000000000000000011000, RT: 00000000000000000000000000011001, RD: 00000000000000000000000000000000 , PC: 00000000000000000000000000001001  signal: 0, clock:
1 , RD adress: 11001 , RS adress: 11000 , RT adress: 11001 , SignExtImm: 00000000000000000000000010000000
# Opcode: 000000 , funct: 100010 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000000010001, RD: 00000000000000000000000000000000 , PC: 00000000000000000000000000001001  signal: 0, clock:
0 , RD adress: 11001 , RS adress: 10000 , RT adress: 10001 , SignExtImm: 11111111111111111001000000100010
# Opcode: 000000 , funct: 100010 , RS: 00000000000000000000000000010000, RT: 00000000000000000000000000010001, RD: 11111111111111111111111111111111 , PC: 00000000000000000000000000001010  signal: 1, clock:
1 , RD adress: 10010 , RS adress: 10000 , RT adress: 10001 , SignExtImm: 11111111111111111001000000100010
```

```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111  1.// 00000000000000000000000001101100
00000000000000000000000000001111  2.// 00000000000000000000000000010010
00000000000000000000000011110000  4.// 00000000000000000000000000010101
00000000000000000000111100000000
00000000000000000001000000000000  5.// 00000000000000000000000000000001
00000000000000000000000100000000  6.// 00000000000000000000000000000000
00000000000000000000000000001001
00000000000000000000000000001100
00000000000000000000000001100000
00000000000000000000000010000
00000000000000000000000010001
00000000000000000000000010010  7.// 11111111111111111111111111111111
00000000000000000000000010011
00000000000000000000000010100
00000000000000000000000010101
00000000000000000000000010110
00000000000000000000000010111
00000000000000000000000011000
00000000000000000000000011001
00000000000000000000000011010
00000000000000000000000011011
00000000000000000000000011100
00000000000000000000000011101
00000000000000000000000011110
00000000000000000000000011111
```

Registers.mem dosyasında instructionlar çalıştırıldıktan sonra değişmesi gereken bazı registerlar yukarıdadır.

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_testbench/test/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000001101100
00000000000000000000000000010010
00000000000000000000000000010101
00000000000000000000111100000000
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000001001
00000000000000000000000000001100
00000000000000000000000001100000
00000000000000000000000000010000
00000000000000000000000000010001
11111111111111111111111111111111
00000000000000000000000000010100
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111
```

Program çalıştırıldıktan sonra oluşan res_register.mem dosyasının içeriği

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_testbench/test/data_mem
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001000
00000000000000000000000000001001
00000000000000000000000000001010
00000000000000000000000000001011
00000000000000000000000000001100
00000000000000000000000000001101
00000000000000000000000000001110
00000000000000000000000000001111
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
```

```
43  00000000000000000000000000000111
44  00000000000000000000000000001000
45  00000000000000000000000000001001
46  00000000000000000000000000001010
47  00000000000000000000000000001011
48  00000000000000000000000000001100
49  00000000000000000000000000001101
50  00000000000000000000000000001110
51  00000000000000000000000000001111
52  00000000000000000000000000010000
53  00000000000000000000000000010001
54  00000000000000000000000000010010
55  00000000000000000000000000010011
56  00000000000000000000000000010100
57  00000000000000000000000000010101
58  00000000000000000000000000010110
59  00000000000000000000000000001001
60  00000000000000000000000000011000
61  00000000000000000000000000011001
62  00000000000000000000000000011010
63  00000000000000000000000000011011
64  00000000000000000000000000011100
65  00000000000000000000000000011101
66  00000000000000000000000000011110
67  00000000000000000000000000011111
68  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
69  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
70  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
71  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
72  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
73  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
74  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
75  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
76  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
77  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
78  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
154  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
155  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
156  xxxxxxxxxxxxxxxx0000000000011001
157  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
158  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
159  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
160  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
161  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
162  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Program çalışmasından sonra oluşan Res_data.mem dosyası içeriği yukardaki gibidir. Instructionlardan birinde sh instructionı vardı ve memory'nin 155. Indexine bir register değerini halfword olarak yazması isteniyordu ve bunu başarıyla yaptığını ekran görüntüsünden görebiliyoruz.