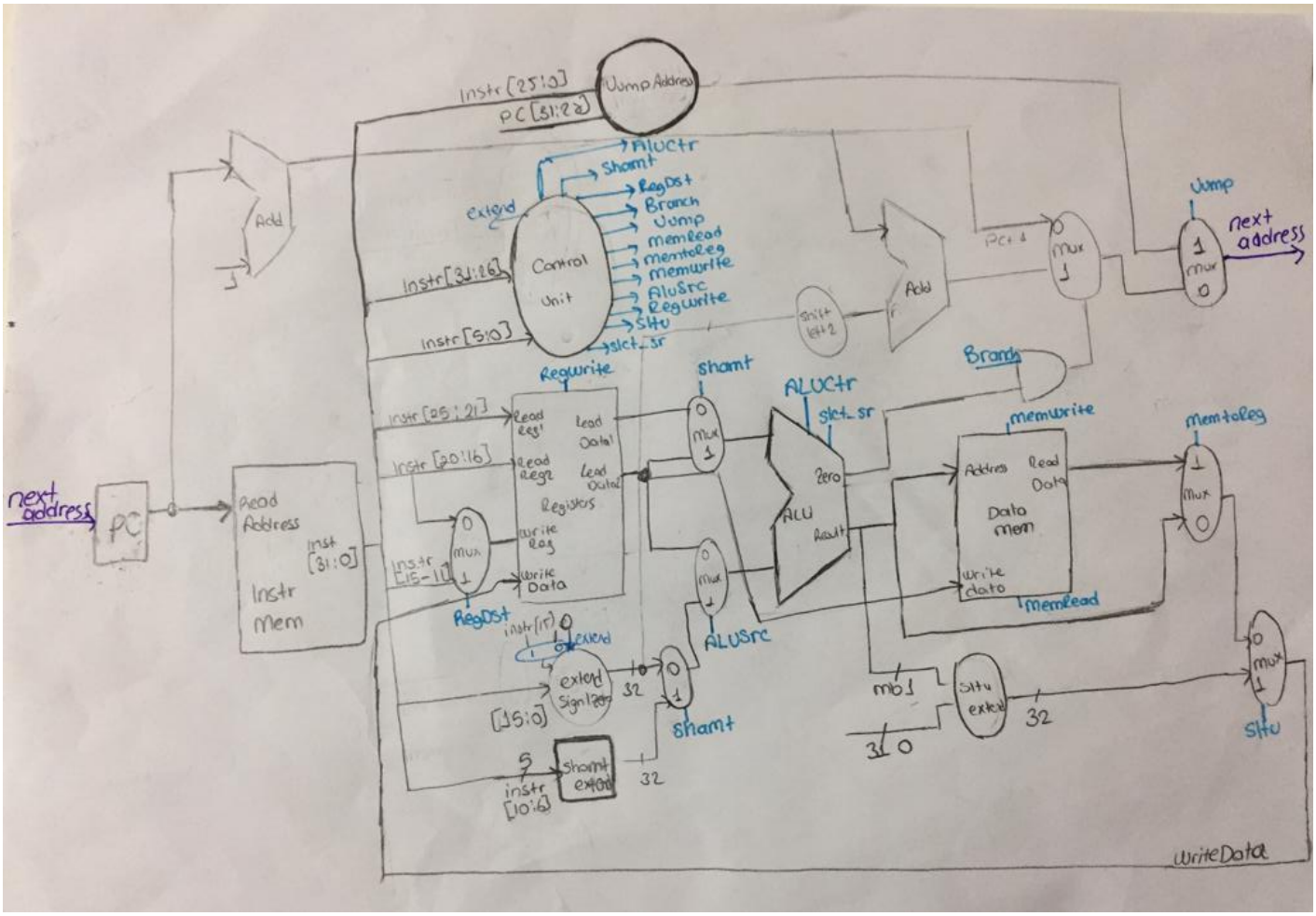
CSE 331 Computer Organization

Final Project – Single cycle MIPS with Structural Verilog

SEVGİ BAYANSALDUZ - 151044076

- **DATAPATH**



Yukarıdaki datapath ders slaytlarındaki datapath`e benzerdir.Ders slaytlarındaki datapath`den farkları şunlardır:

- Control unit modülü aluOp sinyali yerine AluCtr sinyali üretir,bundan dolayı Alu Control modülüne olan ihityaç ortadan kalkar.
- Bu datapath shift işlevlerini destekler. shamt_extend modülü ile instruction içerisindeki shamt değeri extend edilir; extend sonucu Alu`nun 2.girşine gönderilir. 2.datayı seçmek için Alu girişinde shamt sinyali alan bir mux bulunur. Aynı şekilde Alu`ya girecek 1. dataya rt registerından okunan değeri göndermek için de Alu grişinde mux bulunur.
- Bu datapath sltu işlevini destekler. Alu çıkışında sltu_extend moülü bulunur ve registera yazılacak datayı seçmek için sltu sinyali alan ekstra bir mux bulunur.
- Bu datapath ayrıca jump işlevini destekler.Jump_address modülü ile jump adresi hesaplanır ve nextPc modülüne gidecek adresi seçmek için jump sinyali alan ekstra bir mux bulunur.

- **mips32_single_cycle modülü**

```
31
32  //ilk başta 0 verip programı başlatacaktır.
33  //program counteri cağır ve instructionın adresini oku.
34  nextPC p_c(PC,clock,jump,branch_signal,nextAddress);
35  //PC değerine göre instructionı instruction memoryden oku.
36  instruction_mem imem(instruction,PC);
37
38  //Control_unit ile sinyalleri oluştur.
39  control_unit c_u(AluCtr,regDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,branch,
40
41  //destination registerı şeç
42  mux2to1_5bit m(write_reg,instruction[20:16],instruction[15:11],regDst);
43
44  //registerdan veri al.
45  mips_registers mrl(read_data1,read_data2,Result,instruction[25:21],instruction[2
46
47  //sign or zero extendbit
48  mux_2_1_1bit mlbit(extendbit2,extendbit,instruction[15],extendbit);
49  sign_or_zero_extend sze(s_z_extend,instruction[15:0],extendbit2);
50
51  //shamt extend
52  shamt_extend s_ex(shmt_extend,instruction[10:6]);
53
54  //aluData1 seçimi
55  mux_2_1_32bit mux(aluData1,read_data1,read_data2,slct_shamt);
56
57  //aluData2 seçimi
58  mux_2_1_32bit mux1(tempData2,s_z_extend,shmt_extend,slct_shamt);
59  mux_2_1_32bit mux2(aluData2,read_data2,tempData2,ALUSrc);
60
61  //alu32 işlemleri
62  alu32 alu(aluRes,C,V,Zero,aluData1,aluData2,AluCtr,slct_sr);
63
64  ////////sltu
65  sltu_extend slt_mips(sltuRes,aluRes[31],AluCtr[0]);
66
67
68  //datamemory
69  datamemory dm(dataM,aluRes,read_data2, MemWrite,MemRead, clock );
```

Bu modül yukarıda şeması verilen datapathin implement edildiği kısımdır. (Tamamını ekleyemedim.)
Clock 0 olarak bu module gönderilir,clock 1 olduğunda memorylere yazma ve PC değerinin değişmesi işlemleri gerçekleşir.

Başlangıçta PC regiterına başlangıçta testbenchte 32b'0 atanır.
Bu değer instruction_mem modülene gönderilir ve bir instruction alınır.
Bu instruction için control_unit modülünde sinyaller oluşur.
Bu sinyallere göre instruction gerçekleşir.
Instructiona göre yeni PC değeri hesaplanır.Cycle sonunda hesaplanan PC değeri PC registerına yazılır.Eğer memory veya registera yazma olacaksa cycle sonunda yapılır.

- **control_unit modülü**

INSTR	Function Code	Opcode	Alu Action	Alu Ctr	Reg Dst	ALU Src	MemtoReg	RegWrite	Mem Rd	MemWrt	Branch	Jump	shamt	sltu	slct_sr	extend
add	100000	000000	add	010	1	0	0	1	0	0	0	0	0	0	X	1
addu	100001	000000	add	010	1	0	0	1	0	0	0	0	0	0	X	1
nor	100111	000000	nor	111	1	0	0	1	0	0	0	0	0	0	X	1
or	100101	000000	or	001	1	0	0	1	0	0	0	0	0	0	X	1
sltu	101011	000000	sub	100	1	0	0	1	0	0	0	0	0	1	X	1
sll	000000	000000	sll	110	1	1	0	1	0	0	0	0	1	0	X	1
srl	000010	000000	sr	101	1	1	0	1	0	0	0	0	1	0	0	1
sub	100010	000000	sub	100	1	0	0	1	0	0	0	0	0	0	X	1
subu	100011	000000	sub	100	1	0	0	1	0	0	0	0	0	0	X	1
and	100100	000000	and	000	1	0	0	1	0	0	0	0	0	0	X	1
addiu	XXXXXX	001001	add	010	0	1	0	1	0	0	0	0	0	0	X	1
ori	XXXXXX	001101	or	001	0	1	0	1	0	0	0	0	0	0	X	0
andi	XXXXXX	001100	and	000	0	1	0	1	0	0	0	0	0	0	X	0
lw	XXXXXX	100011	add	010	0	1	1	1	1	0	0	0	0	0	X	1
sw	XXXXXX	101011	add	010	X	1	X	0	0	1	0	0	0	0	X	1
beq	XXXXXX	000100	sub	100	X	0	X	0	0	0	1	0	0	0	X	1
j	XXXXXX	000010	#	XXX	X	X	X	0	0	0	0	1	0	0	X	1

Instuctionlara göre sinyaller bu modülde yukarıdaki tabloya göre üretilir.Modülün bir kısmı aşağıdadır.

workspace/control_unit.v

```
4   output [2:0] ALUCtr;
5   output RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,branch,jump,shamt,slct_sr
6
7   wire [18:0]temp;
8   /////////RegDst
9   //RegDst=~op0.~op1.~op2
10  not n(temp[0],opcode[0]);
11  not n1(temp[1],opcode[1]);
12  not n2(temp[2],opcode[2]);
13  and a(RegDst,temp[0],temp[1],temp[2]);
14
15  ////////MEMToReg
16  //MEMToReg=op5.~op3
17
18  not n3(temp[3],opcode[3]);
19  and a1(MemtoReg,temp[3],opcode[5]);
20
21  //////////REGWrite
22  //REGWrite=regdst+op3.~op1+~op3.op5
23  buf na(temp[4],RegDst);
24  and na1(temp[5],opcode[3],temp[1]);
25  and na2(temp[6],temp[3],opcode[5]);
26  or o2(RegWrite,temp[4],temp[5],temp[6]);
27
28  /////////MEMRead
29  //MEMRead=~OP3.OP5
30  and a2(MemRead,temp[3],opcode[5]);
31
32  /////////MemWrite
33  ////MemWrite=op3.op5
34  and a3(MemWrite,opcode[3],opcode[5]);
35
36  /////////branch
37  //branch=op2.~op3
38  and a4(branch,opcode[2],temp[3]);
39
40  ////////jump
41  //jump=op1.~op0
42  and a5(jump,opcode[1],temp[0]);
```

- **instruction_mem modülü**

```
1   //Instruction okuması yapılır, program counter değiştikçe okuma yapar.
2   //Program ilk çalıştırıldığında testbenchte 32b'0 değeri verilir.
3   module instruction_mem(instruction, program_counter);
4
5      input [31:0] program_counter;
6      output reg [31:0] instruction;
7      reg [31:0] instr_mem [255:0];
8
9      always @(*) begin
10        instruction = instr_mem[program_counter];
11     end
12
13  endmodule
14
```

Instruction okuması yapılır, program counter değiştikçe okuma yapar. Program ilk çalıştırıldığında 32b'0 değeri testbenchte verilir.

- **datamemory modülü**

```
1   module datamemory(data, address,write_data ,memWrite,memRea
2
3      output reg [31:0] data;
4      input [31:0] address,write_data;
5      input memRead,memWrite,clk;
6
7      reg [31:0] data_mem [255:0];
8
9      always @(*)begin
10        if (memRead) begin
11           data[31:0] = data_mem[address];
12        end
13     end
14     always @(posedge clk)begin
15        if ( memWrite)begin
16           data_mem[address] = write_data[31:0];
17        end
18     end
19
20  endmodule
```

Testbenchte dosyanın içndekiler data_mem `e yazılır. Daha sonra her clk değişimi ile eğer write sinyali 1 ise memory`e yazma gerçekleşir. Modüle giren sinyallerden biri değişirse ve memRead 1 ise memoryden okuma yapılır.

- **jumpaddress modülü**

workspace/jump_address.

```
6     output[31:0]address;
7
8     //jumpın opcode`u 00 0010 ve instr
9
10    buf b1(address[0],instr[0]);
11    buf b2(address[1],instr[1]);
12    buf b3(address[2],instr[2]);
13    buf b4(address[3],instr[3]);
14    buf b5(address[4],instr[4]);
15    buf b6(address[5],instr[5]);
16    buf b7(address[6],instr[6]);
17    buf b8(address[7],instr[7]);
18    buf b9(address[8],instr[8]);
19    buf b10(address[9],instr[9]);
20    buf b11(address[10],instr[10]);
21    buf b12(address[11],instr[11]);
22    buf b13(address[12],instr[12]);
23    buf b14(address[13],instr[13]);
24    buf b15(address[14],instr[14]);
25    buf b16(address[15],instr[15]);
26    buf b17(address[16],instr[16]);
27    buf b18(address[17],instr[17]);
28    buf b19(address[18],instr[18]);
29    buf b20(address[19],instr[19]);
30    buf b21(address[20],instr[20]);
31    buf b22(address[21],instr[21]);
32    buf b23(address[22],instr[22]);
33    buf b24(address[23],instr[23]);
34    buf b25(address[24],instr[24]);
35    buf b26(address[25],instr[25]);
36    buf b27(address[26],jump_opcode);
37    buf b28(address[27],jump_opcode);
38    buf b29(address[28],pc[0]);
39    buf b30(address[29],pc[1]);
40    buf b31(address[30],pc[2]);
41    buf b32(address[31],pc[3]);
42
43    endmodule
```

Bu modülde jump adresi hesaplanır. 26 bitlik address biti srl ile 28 bite çıkarılır daha sonra ,PC registerının most significant 4 biti ,jump adresinin [31:28] bitlerine eklenir.

- **nextPC modülü**

workspace/nextPC.v

```
1   module nextPC(pc_out,clock,jump,branch_signal,pc_in);
2
3     input clock,branch_signal,jump;
4     input [31:0] pc_in;
5     output reg [31:0] pc_out;
6
7     always @ (posedge clock) begin
8
9         if(jump==1) begin
10           pc_out = pc_in;
11        end
12        // branch
13        else if(branch_signal == 1) begin
14           pc_out = pc_out + pc_in;
15        end
16        else begin
17           pc_out = pc_out+1;
18        end
19
20    end
21
22  endmodule
```

Bu modül input olarak jump sinyali,branch sinyali ve pc_in isimli bir adres alır. Pc_in adresi mips32_single_cycle modülündeki muxlar ile seçilir ,bundan dolayı bu adres PC adresi ,branch adresi veya jump adresi alır.

Aldığı bu adresi ,aldığı sinyallere göre pc_out registerına cycle sonunda atar.

(Program ilk çalıştırıldığında pc_out değeri 32b'0 alır.)

(NOT: Diğer modüller önceki proje raporlarında bulunduğundan bu rapora eklemedim.)

○ TESTBENCH

workspace/mips32_testbench.v

```
    begin
        clk2=0;
        $readmemb("instruction.mem", i0.imem.instr_mem);
        $readmemb("registers.mem", i0.mrl.registers);
        $readmemb("data_mem.mem", i0.dm.data_mem);
        i0.p_c.pc_out= 32'b0;
        index = 0;
    end
    always @(posedge clk2)
    begin

        if(i0.regDst==1)begin
            $display("\nopcode = %b, rs = %b, rt = %b, rd = %b, shamt = %b, funct = %b
instruction[15:11],i0.instruction[10:6],i0.instruction[5:0],index+1);
            $display("AluData1 = %b\nAluData2 =%b", i0.aluData1,i0.aluData2);
            $display("result = %b",R);
        end
        else if((i0.ALUSrc==1 & i0.regDst!=1) | i0.branch==1) begin
            $display("\nopcode = %b, rs = %b, rt = %b,immediate = %b ,index = %d",i0.i
            $display("AluData1 = %b\nAluData2 =%b", i0.aluData1,i0.aluData2);
            $display("result = %b",R);
        end
        else begin
            $display("\nopcode = %b  ,address =%b  ,index = %d",i0.instruction[31:26],
            $display("jump address = %b",i0.jAddress);
        end

        index <= index +1;
        if(index==8'd17)
            begin
            $writememb("regLast.mem", i0.mrl.registers);
            $writememb("dataLast.mem", i0.dm.data_mem);
            $display(" %d tests completed. \n",index+1);
            $finish;
            end

end
endmodule
```

Simulation/modelsim klasöründe bulunan “instruction.mem” dosyasının içeriğini değiştirerek ya da testbenchte bu dosyanın okunduğu yere kendi dosyanızı ekleyerek kodu test edebilirsiniz.
Registerların içerikleri Simulation/modelsim klasöründe bulunan “registers.mem” dosyasından okunur.
Data memory`nin içerikleri Simulation/modelsim klasöründe bulunan “data_mem.mem” dosyasından okunur.

Değişen register ve data memory`i Simulation/modelsim klasöründe bulunan “regLast.mem” ve “dataLaast.mem” klasöründe bulabilirsiniz.

Testbenchteki if else yapıları, farklı tipteki instructionları farklı şekilde ekrana basmaya yarar.

○ instruction.mem

```
1  00000010000100011000100000100000
2  00000010100101011001100000100001
3  00000001001010100111010000100111
4  00000001000010010101000000100101
5  00000010111010111011111000000101011
6  00000000000010100100000010000000
7  00000000000010000100000011000010
8  00000010000100011100000000100010
9  00000010110010001100100000100011
10 00000010101011100100000000100100
11 00100100001001010000000000000010
12 10001100001001000000000000000001
13 10101100011010000000000000000011
14 00110000110001100000000000001001
15 00010000000000000000000000000001
16 00110101000001110011001100001100
17 00001000000000000000000000010001
18 00000010000100011000100000100000
```

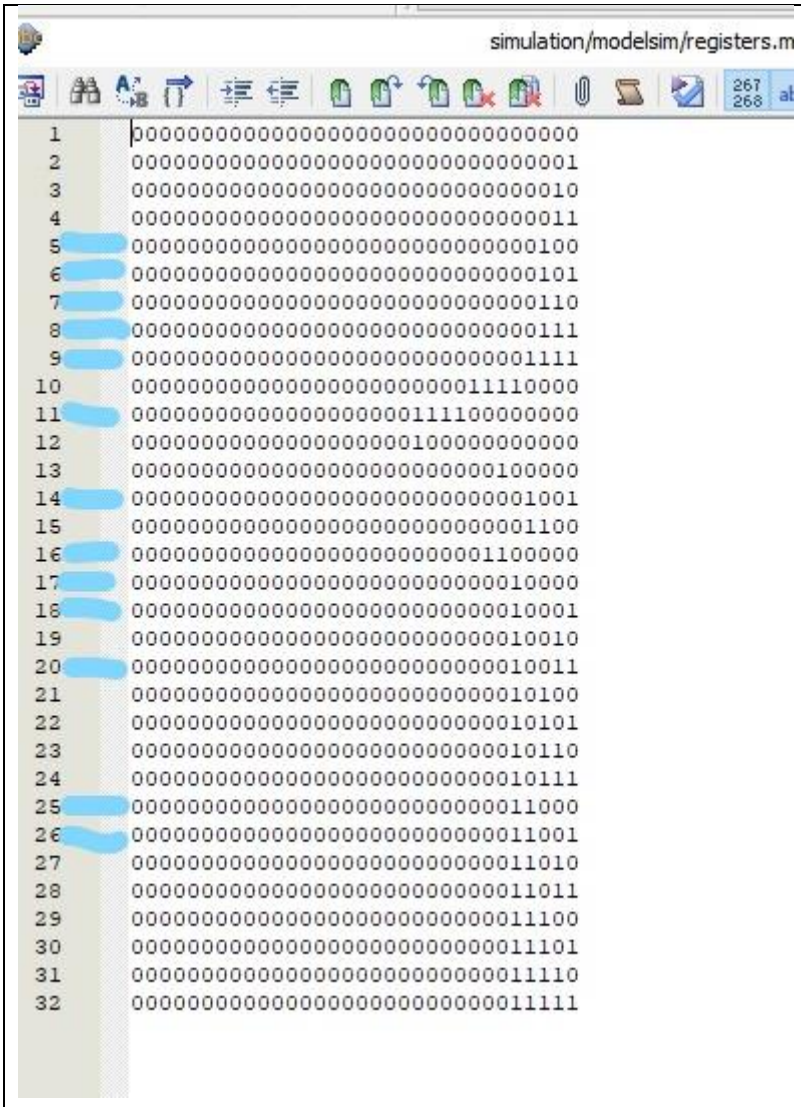
1. add $17, $16, $17
2. addu $19, $20, $21
3. nor $13, $9, $10
4. or $10, $8, $9
5. sltu $15, $22, $23
6. sll $16, $20, 2
7. srl $16, $16, 3
8. sub $24, $16, $17
9. subu $25, $22, $8
10. and $8, $21, $14
11. addiu $5, $1, 2
12. lw $4, 1($1)
13. sw $3 ,3($8)
14. andi $6, $6, 9
15. beq $0, $0, 16. instructiona git
16. ori $7, $8, 0011001100001100
17. jump 18. Instructiona git
18. add $17, $16, $17

○ Simulation Results

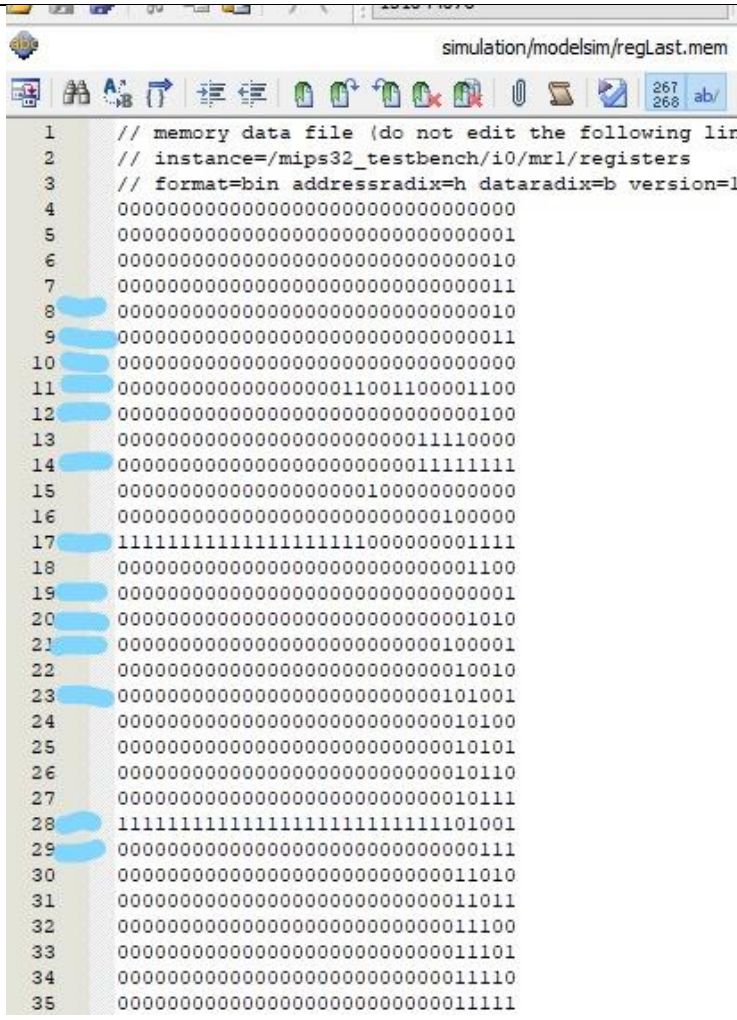
```
sim:/mips32_testbench/result
VSIM 9> step -current
#
# opcode = 000000, rs = 10000, rt = 10001, rd = 10001, shamt = 00000, funct = 100000 ,index =          1
# AluData1 = 00000000000000000000000000010000
# AluData2 =00000000000000000000000000010001
# result = 00000000000000000000000000100001
#
# opcode = 000000, rs = 10100, rt = 10101, rd = 10011, shamt = 00000, funct = 100001 ,index =          2
# AluData1 = 00000000000000000000000000010100
# AluData2 =00000000000000000000000000010101
# result = 00000000000000000000000000101001
#
# opcode = 000000, rs = 01001, rt = 01010, rd = 01101, shamt = 00000, funct = 100111 ,index =          3
# AluData1 = 00000000000000000000000011110000
# AluData2 =00000000000000000000111100000000
# result = 11111111111111111111000000001111
#
# opcode = 000000, rs = 01000, rt = 01001, rd = 01010, shamt = 00000, funct = 100101 ,index =          4
# AluData1 = 00000000000000000000000000001111
# AluData2 =00000000000000000000000011110000
# result = 00000000000000000000000011111111
#
# opcode = 000000, rs = 10110, rt = 10111, rd = 01111, shamt = 00000, funct = 101011 ,index =          5
# AluData1 = 00000000000000000000000000010110
# AluData2 =00000000000000000000000000010111
# result = 00000000000000000000000000000001
#
# opcode = 000000, rs = 00000, rt = 10100, rd = 10000, shamt = 00010, funct = 000000 ,index =          6
# AluData1 = 00000000000000000000000000010100
# AluData2 =00000000000000000000000000000010
# result = 00000000000000000000000001010000
#
# opcode = 000000, rs = 00000, rt = 10000, rd = 10000, shamt = 00011, funct = 000010 ,index =          7
# AluData1 = 00000000000000000000000001010000
# AluData2 =00000000000000000000000000000011
# result = 00000000000000000000000000001010
#
# opcode = 000000, rs = 10000, rt = 10001, rd = 11000, shamt = 00000, funct = 100010 ,index =          8
# AluData1 = 00000000000000000000000000001010
# AluData2 =00000000000000000000000000100001
# result = 11111111111111111111111111101001
#
# opcode = 000000, rs = 10110, rt = 01000, rd = 11001, shamt = 00000, funct = 100011 ,index =          9
# AluData1 = 00000000000000000000000000010110
# AluData2 =00000000000000000000000000001111
# result = 00000000000000000000000000000111
#
# opcode = 000000, rs = 10101, rt = 01110, rd = 01000, shamt = 00000, funct = 100100 ,index =         10
# AluData1 = 00000000000000000000000000010101
# AluData2 =00000000000000000000000000001100
# result = 00000000000000000000000000000100
#
# opcode = 001001, rs = 00001, rt = 00101,immediate = 0000000000000010 ,index =         11
# AluData1 = 00000000000000000000000000000001
# AluData2 =00000000000000000000000000000010
# result = 00000000000000000000000000000011
#
# opcode = 100011, rs = 00001, rt = 00100,immediate = 0000000000000001 ,index =         12
# AluData1 = 00000000000000000000000000000001
# AluData2 =00000000000000000000000000000001
# result = 00000000000000000000000000000010
#
# opcode = 101011, rs = 00011, rt = 01000,immediate = 0000000000000011 ,index =         13
# AluData1 = 00000000000000000000000000000011
# AluData2 =00000000000000000000000000000011
# result = 00000000000000000000000000000110
#
# opcode = 001100, rs = 00110, rt = 00110,immediate = 0000000000001001 ,index =         14
# AluData1 = 00000000000000000000000000000110
# AluData2 =00000000000000000000000000001001
# result = 00000000000000000000000000000000
#
# opcode = 000100, rs = 00000, rt = 00000,immediate = 0000000000000001 ,index =         15
# AluData1 = 00000000000000000000000000000000
# AluData2 =00000000000000000000000000000000
# result = 00000000000000000000000000000000
#
# opcode = 001101, rs = 01000, rt = 00111,immediate = 0011001100001100 ,index =         16
# AluData1 = 00000000000000000000000000000100
# AluData2 =00000000000000000011001100001100
# result = 00000000000000000011001100001100
#
# opcode = 000010  ,address =00000000000000000000010001  ,index =         17
# jump address = 00000000000000000000000000010001
#
```

```
#
# opcode = 000000, rs = 10000, rt = 10001, rd = 10001, shamt = 00000, funct = 100000 ,index =         18
# AluData1 = 00000000000000000000000000001010
# AluData2 =00000000000000000000000000100001
# result = 00000000000000000000000000101011
#          18 tests completed.
#
# ** Note: $finish    : C:/Users/sevgi/Desktop/devam/151044076/workspace/mips32_testbench.v(49)
#    Time: 5250 ps  Iteration: 1  Instance: /mips32_testbench
# 1
# Break in Module mips32_testbench at C:/Users/sevgi/Desktop/devam/151044076/workspace/mips32_testbench.v line 49
```

- **Register**



Registerların ilk durumu



Son durumu

- **Data memory**

simulation/modelsim/data_mem.mem

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001111
10 00000000000000000000000000000000
11 00000000000000000000000000000001
12 00000000000000000000000000000010
13 00000000000000000000000000000011
14 00000000000000000000000000000100
15 00000000000000000000000000000101
16 00000000000000000000000000000110
17 00000000000000000000000000000111
18 00000000000000000000000000001111
19 00000000000000000000000000000011
20 00000000000000000000000000000100
21 00000000000000000000000000000101
22 00000000000000000000000000000110
23 00000000000000000000000000000111
24 00000000000000000000000000001111
25
```

ilk durumu

simulation/modelsim/dataLast.mem

```
1  // memory data file (do not edit the following line - r
2  // instance=/mips32_testbench/i0/dm/data_mem
3  // format=bin addressradix=h dataradix=b version=1.0 wo
4  00000000000000000000000000000000
5  00000000000000000000000000000001
6  00000000000000000000000000000010
7  00000000000000000000000000000011
8  00000000000000000000000000000100
9  00000000000000000000000000000101
10 00000000000000000000000000000100
11 00000000000000000000000000000111
12 00000000000000000000000000001111
13 00000000000000000000000000000000
14 00000000000000000000000000000001
15 00000000000000000000000000000010
16 00000000000000000000000000000011
17 00000000000000000000000000000100
18 00000000000000000000000000000101
19 00000000000000000000000000000110
20 00000000000000000000000000000111
21 00000000000000000000000000001111
22 00000000000000000000000000000011
23 00000000000000000000000000000100
24 00000000000000000000000000000101
25 00000000000000000000000000000110
26 00000000000000000000000000000111
27 00000000000000000000000000001111
28 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
29 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
30 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
31 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
32 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
33 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
34 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
35 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
36 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
37 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
38 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
39 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Son durumu