



GEBZE TEKNİK ÜNİVERSİTESİ
ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x7 Deney Raporu

Komut Parçalama ve Hafıza

Hazırlayanlar
1) 1801022022 – Alperen Karataş
2) 1801022091 – Ogün Uygur Yıldırım

1. Giriş

Bu deney kapsamında verilen 32 bitlik bir komutu istenilen şekilde parçalamak, hafıza oluşturup veri okumak ve tasarlanan devreleri gerçekleyip test edebilmek problemleri üzerinde durulmuştur.

2. Problemler

2.1. Problem I – Komut Ayırıcı

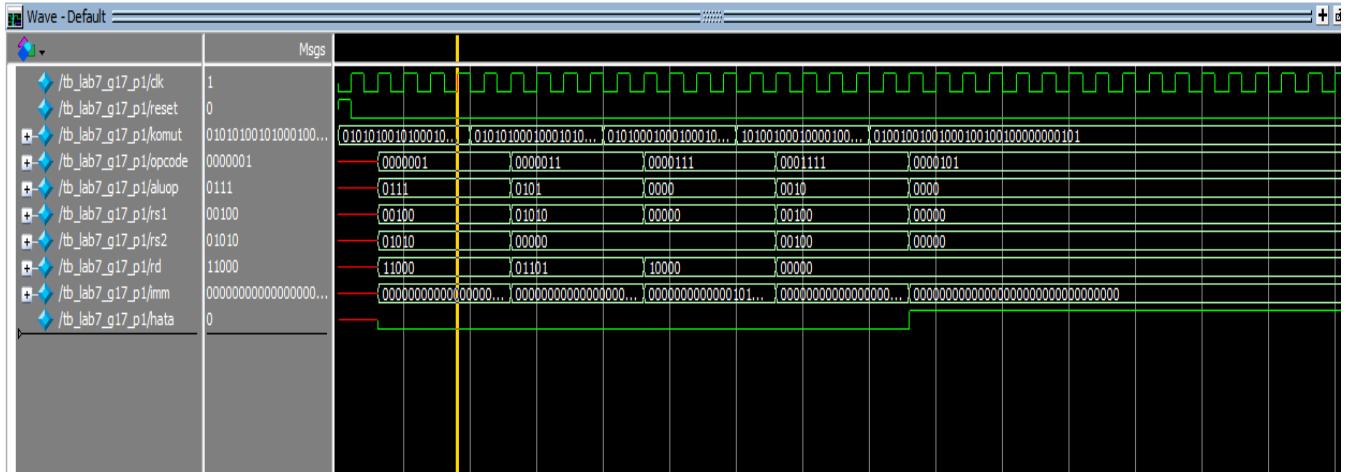
2.1.1. Teorik Araştırma

Bu problemde 32 bitlik bir komutu parçalayabilmek adına decode encoding üzerinde bilgi sahibi olunmuştur.

2.1.2. Deneyin Yapılışı

Bu problemde 32 bitlik bir komutun, dört farklı operasyon tipine göre, her tip kendi içinde bağımsız olmak üzere farklı şekillerde parçalanması yapılmıştır. Uygun if else komutları kullanılarak beş farklı if - else if - else kondisyonu altında parçalama işlemi gerçekleştirilmiştir. Her kondisyonda kullanılmayan portlar veya o portların boşta kalan bitleri latch oluşumunu engellemek adına 0'a atanmıştır. Ayrıca fmax değerinin Quartus Prime programında hesaplanamaması ihtimali de göz önünde bulundurularak devrenin bütün giriş ve çıkış portlarına register eklenmiştir.

Verilen isterlere uygun komut parçalaması yapıldıktan sonra, simülasyon ekranında sonuçları kontrol edebilmek adına yazılan testbench kodu ile test edilen devre karşımıza Şekil 1'deki simülasyon sonuçlarını çıkarmıştır.




Şekil 1. Simülasyon çıktısı

Örnek olarak incelendiğinde, testbench dosyasından gönderilen 32'b01010100101000100111110000000001 komutu, R tipi operasyon koduna uygun olarak bakılıp gerekli parçalama yapıldığında; aluop, rs1, rs2, rd, imm ve hata sinyallerinin olması gerektiği gibi sonuç verdiği görülmektedir.

```
aluop = 0111
rs1   = 00100
rs2   = 01000
rd    = 11000
imm   = 32'b0
hata  = 0
```

Simülasyon ekranındaki sonuçlar gelmeden önce ilk başta oluşan unknown-state durumunu, devrenin giriş çıkışlarına register bağlanması durumundan dolayı gelen sinyalin çok kısa süreli gecikmesine bağlayabiliriz.

Analysis & Synthesis Resource Utilization by Entity										
 <<Filter>>										
	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins
1	lab7_g17_p1	47 (47)	78 (78)	0	0	0	0	0	157	0

Şekil 2. Utilization Report (1)

Virtual Pins	ADC blocks	Full Hierarchy Name	Entity Name	Library Name
0	0	lab7_g17_p1	lab7_g17_p1	work

Şekil 3. Utilization Report (2)

Şekil 2 ve Şekil 3’de görüldüğü üzere devrede 47 adet “Combinational ALUT”, 78 adet lojik register saptanmıştır. Devrenin 157 adet pini vardır.

Slow 1200mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	357.02 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Şekil 4. Fmax tablosu

Şekil 4’ te görüldüğü gibi devrenin max çalışma frekansı 357.02 MHz’dir.

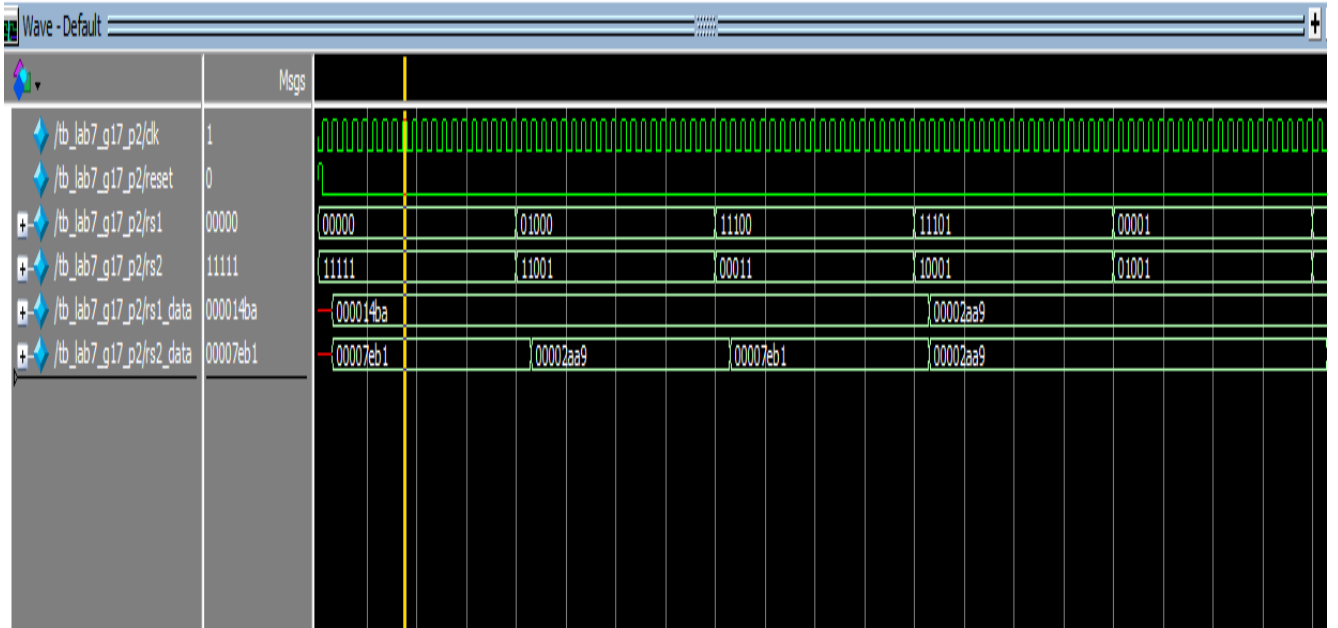
2.2. Problem II - Hafıza oluřturma ve okuma

2.2.1. Teorik Arařtırma

Deneye bařlamadan önce SystemVerilog dilinde hafıza oluřturma ve bu hafızadan data okuma hakkında arařtırmalar yapılmıř ve gereken bilgilere sahip olunmuřtur.

2.2.2. Deneyin Yapılıřı

Bu deneyde öncelikle okuma iřlemi yapılacak olan hafıza oluřturulmuřtur ve sonrasında belirli adreslere göre veriler çekilip hafızadan okuma iřlemi tamamlanmıřtır. Örneğın hafızada bulunan rs1_reg'in tuttuğı değerin ataması rs1_data_reg portuna, rs2_reg'in tuttuğı değerin ataması rs2_data_reg portuna yapılmıřtır. Önceden oluřturulan .mem dosyası içerisindeki veriler hafızada depolanmıřtır ve registerların bu verileri belirtilen iki portun arasına aktarmasıyla okuma saėlanmıřtır.



řekil 6. Simölasyon çıktısı

Analysis & Synthesis Resource Utilization by Entity									
<<Filter>>									
	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	UFM Blocks	DSP Elements	DSP 9x9	DSP 18x18	Pins
1	lab7_g17_p2	1467 (1467)	1103 (1103)	0	0	0	0	0	114

Şekil 7. Utilization Report (1)

Virtual Pins	ADC blocks	Full Hierarchy Name	Entity Name	Library Name
0	0	lab7_g17_p2	lab7_g17_p2	work

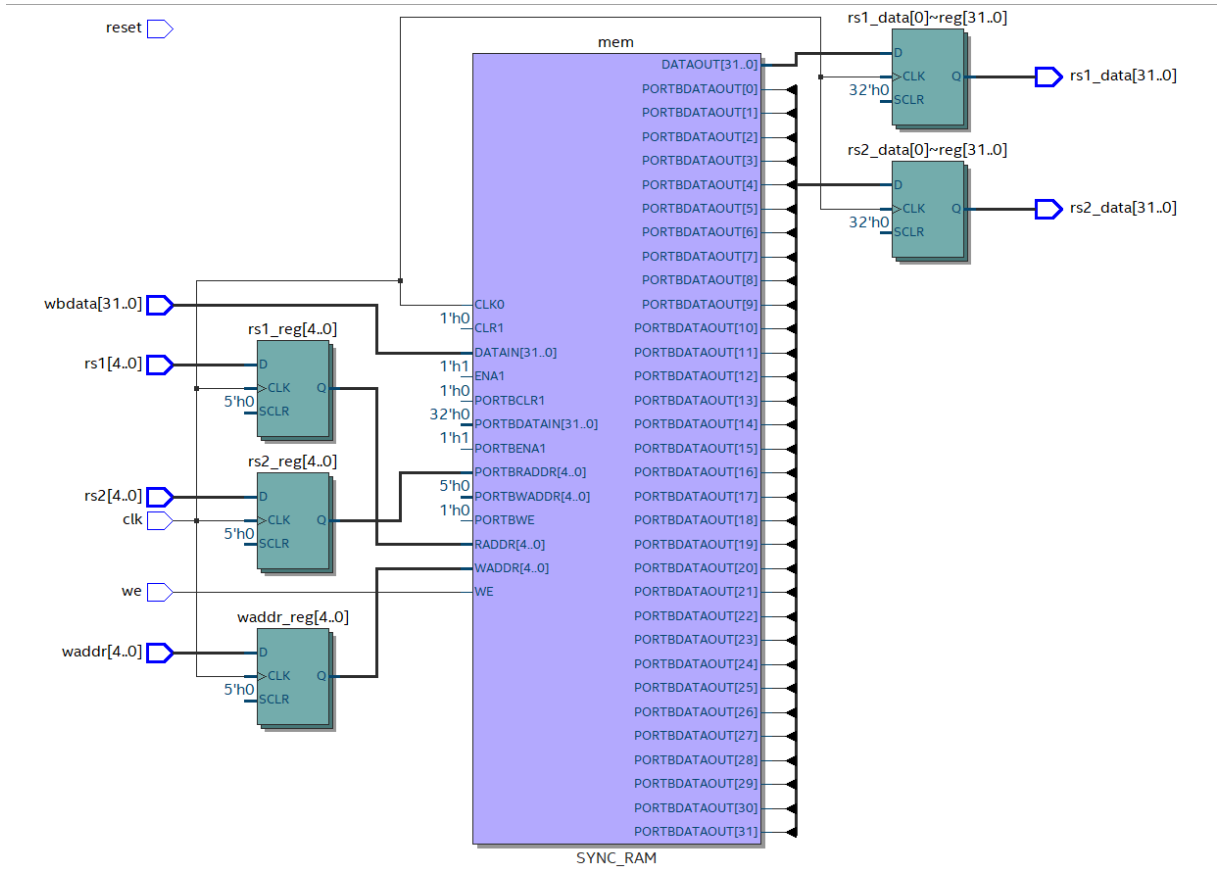
Şekil 8. Utilization Report (2)

Şekil 7 ve Şekil 8’de görüldüğü üzere devrede 1467 adet “Combinational ALUT”, 1103 adet lojik register ve 114 adet pin görülmüştür.

Slow 1200mV 85C Model Fmax Summary			
<<Filter>>			
	Fmax	Restricted Fmax	Clock Name
1	169.46 MHz	169.46 MHz	clk

Şekil 9. Fmax değeri

Şekil 9’ten anlaşılacağı üzere devrenin maksimum çalışma frekansının 169.46 MHz olduğu görülmektedir.



Şekil 10. RTL şeması

Şekil 10’da hafıza devresinin RTL şeması görülmektedir.

2.2.3. Sonuçların Yorumu

Deneyde hafıza oluşturulup ve bu hafızadan veri okunması işlemi yapılmıştır. Şekil 6’da görüldüğü üzere rs1_data ve rs2_data çıkışları binary olarak ekranda göstermiştir. Yazılan kod .mem dosyasındaki verilerle karşılaştırılarak kontrol edilmiştir ve dosyalar içindeki verilere ulaşılmıştır. Devrenin tasarımı şekil 10’da gösterilmiştir ve gerçeğe yakın bir tasarım elde edilmiştir.

Şekil 12.Utilization Report (1)



Pins	Virtual Pins	ADC blocks	Full Hierarchy Name	Entity Name	Library Name
190	0	0	lab7_g17_p3	lab7_g17_p3	work
0	0	0	lab7_g17_...17_p2:uut0	lab7_g17_p2	work

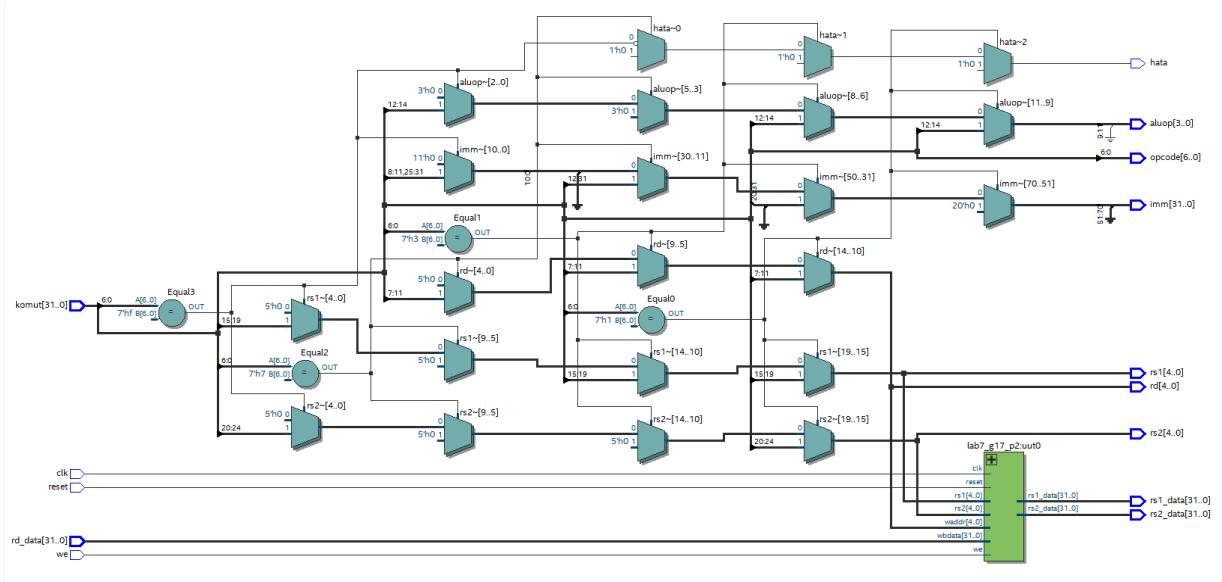
Şekil 13. Utilization Report (2)

Şekil 12 ve Şekil 13’de görüldüğü üzere devrede 1522 adet “Combinational ALUT” saptanmıştır. Devrenin 175 adet pini vardır.

Slow 1200mV 85C Model Fmax Summary			
<<Filter>>			
	Fmax	Restricted Fmax	Clock Name
1	164.26 MHz	164.26 MHz	clk

Şekil 14. Fmax tablosu

Şekil 14’ten anlaşılabacağı üzere devrenin maksimum çalışma frekansının 164.26 MHz olduğu görülmektedir.



Şekil 15. RTL şeması

Şekil 15’te devrenin RTL şeması görülmektedir.

2.3.3. Sonuçların Yorumu

Bu problemde, tasarlanması istenen devre iki farklı tasarımdan oluşmaktadır. Problem 1 devresinde girilen komut belirli parçalara ayrılmıştır, Problem 2 devresinde ise hafızadan okuma gerçekleşmiştir. Bu deneyde girilen komut öncelikle ayrılıp sonrasında okunmuştur ve ekrana bastırılmıştır. Basit bir devre bağlama işleminin haricinde, bu problemde ayrıca rs1_data ve rs2_data portlarına bir önceki problemde rs1 ve rs2'nin tuttuğu değerler atanması görülmüş oldu.

Modülün ALU devresi ile birleştirilmesi

ALU devresi ile birleştirilme işlemi, modül içerisinde ALU devresini “instatiate” etme işlemi ile gerçekleştirilebilir. Gönderilen 32 bitlik komut, operasyon tipine göre parçalanıp gerekli atamalar yapılabilir, yine aynı şekilde operasyon tipine göre de ALU devresinde belirtilen atamalar ve eşitlikler sağlanabilir; istenen işlemler gerçekleştirilebilir. Hafızada tutulan rs1 ve rs2 değerleri de rs1_data ve rs2_data portlarına aktarılabilir. Bu işlemleri devre şemasında görmek için ise, Quartus Prime programında proje açılarak problem 3 dosyası projeye dahil edildikten sonra problem 3 için oluşturulan modülün içerisinde çağırılan hafıza ve ALU devresi, aynı şekilde proje dosyalarına dahil edilir. Kodu compile ettikten sonra Tools → Netlist Viewer → RTL Viewer işlemleri sırasıyla uygulanarak devrenin RTL şemasına ulaşılır.

3. Sonuçlar ve Genel Yorumlar

Bu laboratuvar çalışmasında, istenilen bir sorunun çözümüne ulaşmak için öncelikle problemler hakkında bilgi sahibi olup, sonrasında bunun koda aktarımı tam anlamıyla anlaşılmıştır.

Problemlerde bizden istenilenler, adım adım anlaşılıp sırasıyla kodlanmıştır. İstenilen problemlerde 2 farklı devre tasarlanıp son olarak birbiriyle birleştirilip çalışması istenmiştir. Komut ayırıcı devreyi tasarlamak, if-else komutlarıyla kolayca tasarlanabilirken, hafıza okuma devresi de gerekli örnekler incelenip tasarlanabildi. Problem 3'ü tasarlarırken, başta bazı bağlantıları tasarlayamadık ve sonuca ulaşamadık lakin gerekli çabalar ve uğraşlar sonucunda sonuca ulaşabildik.

Genel anlamda, deney eğitici ve öğretici olmuştur. Bu deney tecrübemizi daha da arttırmıştır.

4. Referanslar

[1] Ders slaytları

[2] Harris and D.Harris, Digital Design and Computer Architecture: ARM Edition, 1st edition. Morgan Kaufmann, 2015

[3] https://github.com/fcayci/sv-digital-design/blob/master/misc/reg_image.mem

KODLAR

```
/*lab7_g17_p1.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab7 - Problem 1

* Komut ayırıcı

*/

module lab7_g17_p1 (
input logic clk, reset,
input logic [31:0] komut,
output logic [6:0] opcode,
output logic [3:0] aluop,
output logic [4:0] rs1,
output logic [4:0] rs2,
output logic [31:0] rs1_data,
output logic [31:0] rs2_data,
output logic [4:0] rd,
output logic [31:0] imm,
output logic hata
);

logic [31:0] komut_reg;
logic [6:0] opcode_reg;
logic [3:0] aluop_reg;
logic [4:0] rs1_reg;
logic [4:0] rs2_reg;
logic [31:0] rs1_data_reg;
logic [31:0] rs2_data_reg;
logic [4:0] rd_reg;
logic [31:0] imm_reg;
logic hata_reg;

always_ff @(posedge clk)

begin

    komut_reg <= komut;
    opcode <= opcode_reg;
    aluop <= aluop_reg;
    rs1 <= rs1_reg;
    rs2 <= rs2_reg;
    rs1_data <= rs1_data_reg;
    rs2_data <= rs2_data_reg;
    rd <= rd_reg;
    imm <= imm_reg;
    hata <= hata_reg;

end
```

```

assign opcode_reg = komut_reg[6:0];

always @(*)
begin
    if(komut_reg[6:0] == 7'b0000001)
    begin

        rs1_reg = komut_reg[19:15];
        rs2_reg = komut_reg[24:20];
        rd_reg = komut_reg[11:7];
        aluop_reg[3] = komut_reg[30]; aluop_reg[2:0] = komut_reg[14:12];
        aluop_reg[3] = 1'b0;
        imm_reg = 32'd0;
        hata_reg = 1'b0;
        rs1_data_reg=32'b0;
        rs2_data_reg=32'b0;

    end

    else if(komut_reg[6:0] == 7'b0000011)
    begin

        rs1_reg = komut_reg[19:15];
        rd_reg = komut_reg[11:7];
        aluop_reg[2:0] = komut_reg[14:12]; aluop_reg[3] = 1'b0;
        imm_reg[11:0] = komut_reg[31:20];
        imm_reg[31:12] = 20'b0;
        hata_reg = 1'b0;
        rs2_reg = 5'b00000;
        rs1_data_reg=32'b0;
        rs2_data_reg=32'b0;
    end

    else if(komut_reg[6:0] == 7'b0000111)
    begin

        rd_reg = komut_reg[11:7];
        imm_reg[19:0] = komut_reg[31:12];
        hata_reg = 1'b0;
        rs1_reg = 5'b00000;
        rs2_reg = 5'b00000;
        aluop_reg = 4'b0000;
        rs1_data_reg=32'b0;
        rs2_data_reg=32'b0;
    end
end

```

```

else if(komut_reg[6:0] == 7'b0001111)
    begin

        rs1_reg = komut_reg[19:15];
        rs2_reg = komut_reg[24:20];
        aluop_reg[2:0] = komut_reg[14:12]; aluop_reg[3] = 1'b0;
        imm_reg[12:6] = komut_reg[31:25]; imm_reg[5:1] =
komut_reg[11:7];
        imm_reg[1] = 1'b0;
        imm_reg[19:13] = 7'b0; imm_reg[0] = 1'b0;
        hata_reg = 1'b0;
        rd_reg = 5'b00000;
        rs1_data_reg=32'b0;
        rs2_data_reg=32'b0;
    end

    else
    begin

        hata_reg = 1;
        rs1_reg = 5'b0;
        rs2_reg = 5'b0;
        rd_reg = 5'b0;
        aluop_reg = 4'b0;
        imm_reg = 32'b0;
        rs1_data_reg = 32'b0;
        rs2_data_reg = 32'b0;

    end

end

endmodule

```

```

/*tb_lab7_g17_p1.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygur Yıldırım

* ELM235 2020 Bahar Lab7 - Problem 1

* Komut ayırıcı testbench

*/

`timescale 1ns/1ps
module tb_lab7_g17_p1();
logic clk,reset;
logic [31:0] komut;
logic [6:0] opcode;
logic [3:0] aluop;
logic [4:0] rs1;
logic [4:0] rs2;
logic [4:0] rd;
logic [31:0] imm;
logic hata;

lab7_g17_p1 uut0 (.clk(clk), .reset(reset), .komut(komut),
.opcode(opcode), .aluop(aluop), .rs1(rs1), .rs2(rs2), .rd(rd),
.imm(imm), .hata(hata));

always begin
clk=0; #10;
clk=1; #10;
end

initial begin
reset=1; #10;
reset=0;
end

initial begin
komut = 32'b01010100101000100111110000000001; #100 // R
komut = 32'b01010100010001010101011010000011; #100 // I
komut = 32'b01010001000100010101100000000111; #100 // U
komut = 32'b10100100010000100010110000001111; #100 // B
komut = 32'b01001001001000100100100000000101; #100 // diger kosullar
#1000;
$stop;
end
endmodule

```

```

/*lab7_g17_p2.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygur Yıldırım

* ELM235 2020 Bahar Lab7 - Problem 2

* Hafıza oluşturma ve okuma

*/

module lab7_g17_p2 (
    input  logic clk, reset,
    input  logic we,
    input  logic [4:0] waddr,
    input  logic [31:0] wdata,
    input  logic [4:0] rs1,
    input  logic [4:0] rs2,
    output logic [31:0] rs1_data,
    output logic [31:0] rs2_data
);

    logic [31:0] mem[0:31];
    logic we_reg;
    logic [4:0] waddr_reg;
    logic [31:0] wdata_reg;
    logic [4:0] rs1_reg;
    logic [4:0] rs2_reg;
    logic [31:0] rs1_data_reg;
    logic [31:0] rs2_data_reg;

    initial begin
        $readmemh("a.txt",mem);
        $readmemh("b.txt",mem);

        end

    always_comb
    begin
        rs1_data_reg=mem[rs1_reg];
        rs2_data_reg=mem[rs2_reg];
    end

    always_ff @(posedge clk)
    begin
        if (we) begin mem[waddr_reg] <= wdata; end
        we_reg <= we;
        waddr_reg<= waddr;
        wdata_reg<=wdata;
        rs1_reg<=rs1;
        rs2_reg<=rs2;
        rs1_data<=rs1_data_reg;
        rs2_data<=rs2_data_reg;

        end

endmodule

```

```

/*tb_lab7_g17_p2.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygur Yıldırım

* ELM235 2020 Bahar Lab7 - Problem 2

* Hafıza oluşturma ve okuma testbench

*/

`timescale 1ns/1ps

module lab7_g17_p2 (
    input  logic clk, reset,
    input  logic we,
    input  logic [4:0] waddr,
    input  logic [31:0] wdata,
    input  logic [4:0] rs1,
    input  logic [4:0] rs2,
    output logic [31:0] rs1_data,
    output logic [31:0] rs2_data
);

    logic [31:0] mem[0:31];

    logic we_reg;
    logic [4:0] waddr_reg;
    logic [31:0] wdata_reg;
    logic [4:0] rs1_reg;
    logic [4:0] rs2_reg;
    logic [31:0] rs1_data_reg;
    logic [31:0] rs2_data_reg;

    initial begin
        $readmemh("reg_image.mem", mem);
    end

    always_comb
    begin
        rs1_data_reg=mem[rs1_reg];
        rs2_data_reg=mem[rs2_reg];
    end

    always_ff @(posedge clk)
    begin
        if (we) mem[waddr_reg] <= wdata; end
        we_reg <= we;
        waddr_reg<= waddr;
        wdata_reg<=wdata;
        rs1_reg<=rs1;
        rs2_reg<=rs2;
        rs1_data<=rs1_data_reg;
        rs2_data<=rs2_data_reg;
    end

endmodule

```



```

/*lab7_g17_p3.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygur Yıldırım

* ELM235 2020 Bahar Lab7 - Problem 3

* Modül birleştirme

*/

module lab7_g17_p3 (
input logic clk,reset,
input logic we ,
input logic [31:0] rd_data,
input logic [31:0] komut,
output logic [6:0] opcode,
output logic [3:0] aluop,
output logic [4:0] rs1,
output logic [4:0] rs2,
output logic [31:0] rs1_data,
output logic [31:0] rs2_data,
output logic [4:0] rd,
output logic [31:0] imm,
output logic hata
);

lab7_g17_p2 uut0 (
.clk(clk),
.reset(reset),
.rs1_data(rs1_data),
.rs2_data(rs2_data),
.rs1(rs1),
.rs2(rs2),
.we(we),
.wbdata(rd_data),
.waddr(rd)
);

assign opcode = komut[6:0];

always @(*)
begin
    if(komut[6:0] == 7'b0000001)
    begin

        rs1 = komut[19:15];
        rs2 = komut[24:20];
        rd = komut[11:7];
        aluop[3] = komut[30]; aluop[2:0] = komut[14:12];
        aluop[3] = 1'b0;
        imm = 32'd0;
        hata = 1'b0;
        /*rs1_data=32'b0;
        rs2_data=32'b0;*/

    end
end

```

```

else if(komut[6:0] == 7'b0000011)
begin

    rs1 = komut[19:15];
    rd = komut[11:7];
    aluop[2:0] = komut[14:12]; aluop[3] = 1'b0;
    imm[11:0] = komut[31:20];
    imm[31:12] = 20'b0;
    hata = 1'b0;
    rs2 = 5'b00000;
    /*rs1_data=32'b0;
    rs2_data=32'b0;*/
end

else if(komut[6:0] == 7'b0000111)
begin

    rd = komut[11:7];
    imm[19:0] = komut[31:12];
    hata = 1'b0;
    rs1 = 5'b00000;
    rs2 = 5'b00000;
    aluop = 4'b0000;
    /*rs1_data=32'b0;
    rs2_data=32'b0;*/
end

else if(komut[6:0] == 7'b0001111)
begin

    rs1 = komut[19:15];
    rs2 = komut[24:20];
    aluop[2:0] = komut[14:12]; aluop[3] = 1'b0;
    imm[12:6] = komut[31:25]; imm[5:1] = komut[11:7];
    imm[1] = 1'b0;
    imm[19:13] = 7'b0; imm[0] = 1'b0;
    hata = 1'b0;
    rd = 5'b00000;
    /*rs1_data=32'b0;
    rs2_data=32'b0;*/
end

else begin

    hata = 1;
    rs1 = 5'b0;
    rs2 = 5'b0;
    rd = 5'b0;
    aluop = 4'b0;
    imm = 32'b0;
    /*rs1_data = 32'b0;
    rs2_data = 32'b0;*/

end
end

endmodule

```

<pre> /*tb_lab6_g17_p3.sv *Hazırlayanlar: * Alperen Karataş - Ogün Uygur Yıldırım * ELM235 2020 Bahar Lab7 - Problem 3 * Modül birleştirme testbench */ `timescale 1ns/1ps module tb_lab7_g17_p3(); logic clk,reset; logic we; logic [31:0] rd_data; logic [31:0] komut; logic [6:0] opcode; logic [3:0] aluop; logic [4:0] rs1; logic [4:0] rs2; logic [31:0] rs1_data; logic [31:0] rs2_data; logic [4:0] rd; logic [31:0] imm; logic hata; lab7_g17_p3 uut0(.clk(clk), .reset(reset), .we(we), .komut(komut), .opcode(opcode), .aluop(aluop), .rs1(rs1), .rs2(rs2), .rs1_data(rs1_data), .rs2_data(rs2_data), .rd(rd), .rd_data(rd_data), .imm(imm), .hata(hata)); always begin clk=0; #10; clk=1; #10; end </pre>	<pre> initial begin reset=1; #10; reset=0; end initial begin komut = 32'b01010100101000100111110000000001; #100 komut = 32'b01010100010001010101011010000011; #100 komut = 32'b01010001000100010101100000000111; #100 komut = 32'b10100100010000100010110000001111; #100 komut = 32'b01001001001000100100100000000101; #100 we = 1'b1; #1000; \$stop; end endmodule </pre>
---	---