GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELM235

LOJİK DEVRE TASARIM LABORATUVARI

LAB 0x8 Deney Raporu

Tek Ritimli işlemci Tasarımı

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

1. Giriş

Bu deney kapsamında tek ritimli işlemci hakkında çalışılmıştır. Gerekli araştırmalar yapılıp deneye başlanmıştır.

2. Problemler

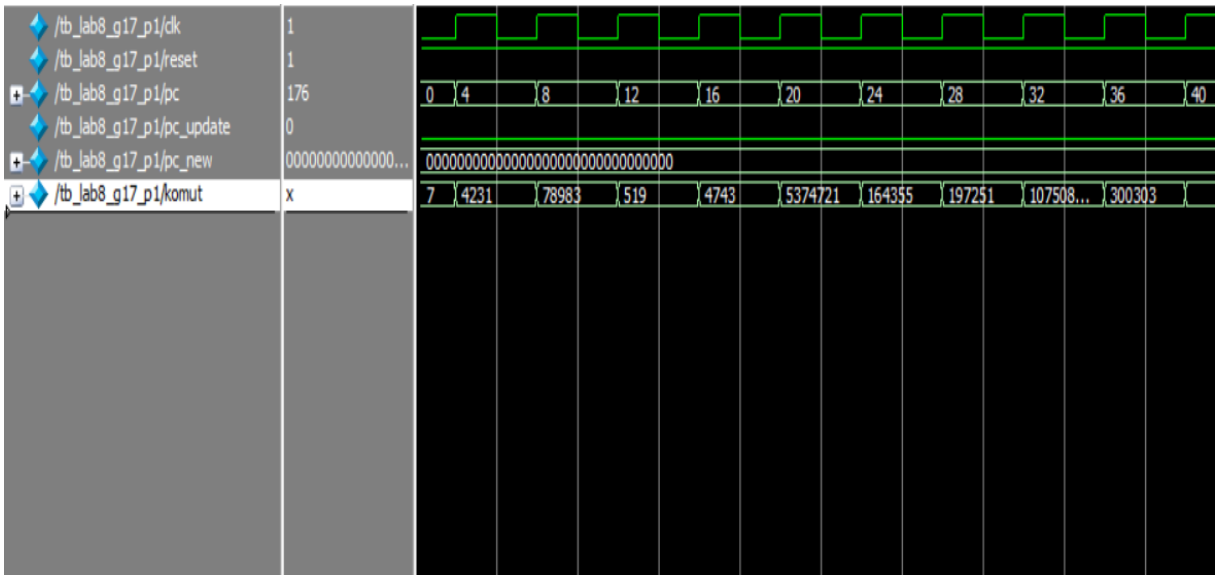
2.1. Problem I – Instruction Fetch Ünitesi

2.1.1. Teorik Araştırma

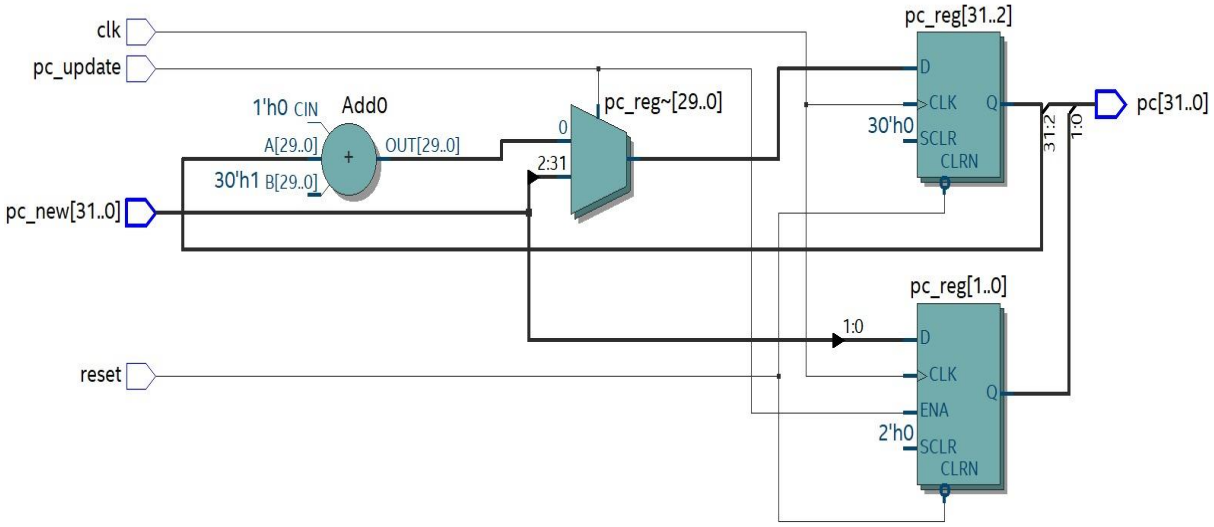
Deneye başlanılmadan önce fetch ünite devresi hakkında nasıl çalıştığına dair araştırmalar yapılmıştır.

2.1.2. Deneyin Yapılışı

Deneye yapılışında işlemcinin fetch ünitesinin tasarımı için gönderilen pc sinyali, pc_update sinyali aktif ise dışarıdan gelen değer pc değerine aktarılmıştır; pc_update sinyali aktif değil ise pc değeri aritmetik olarak 4 arttırılarak yeni değer olarak gösterilmiştir. Fetch ünitesi kontrol amaçlı iki farklı testbench örneği tasarlanmıştır. Basit testbench örneğinde pc sinyal değeri doğru çalıştığı görüldükten sonra diğer testbench örneğinde 64 satırlık, 32-bitlik komut hafızası oluşturulmuştur. Lab ile birlikte verilen hafıza değerleriyle yüklenmiştir ve gelen pc sinyalinin ilk 30 biti bu hafızanın adresi olarak verilmiştir. Hafızanın çıktısı da komut olarak belirlenip, testbenchte hafızaya yüklenilen değerlerle gözlemlenmiştir



Şekil 1. Simülasyon çıktısı



Şekil 2. RTL şeması

Şekil 2'te devrenin RTL şeması görülmektedir

2.1.3. Sonuçların Yorumu

Bu deney, problemde istenilen şekilde tasarlanmıştır ve şekil 1'de görüldüğü gibi istenilen sonuçlar elde edilmiştir. Simülasyon sonucu daha detaylı incelenirse pc değerinin pc_update sinyalinin aktif olmadığı zamanlarda 4 arttığını görebiliyoruz. Komut sinyalinin "fib20.mem" adlı dosyadaki değerleri barındıran komut hafızasından okuma yapabildiğini anladık.

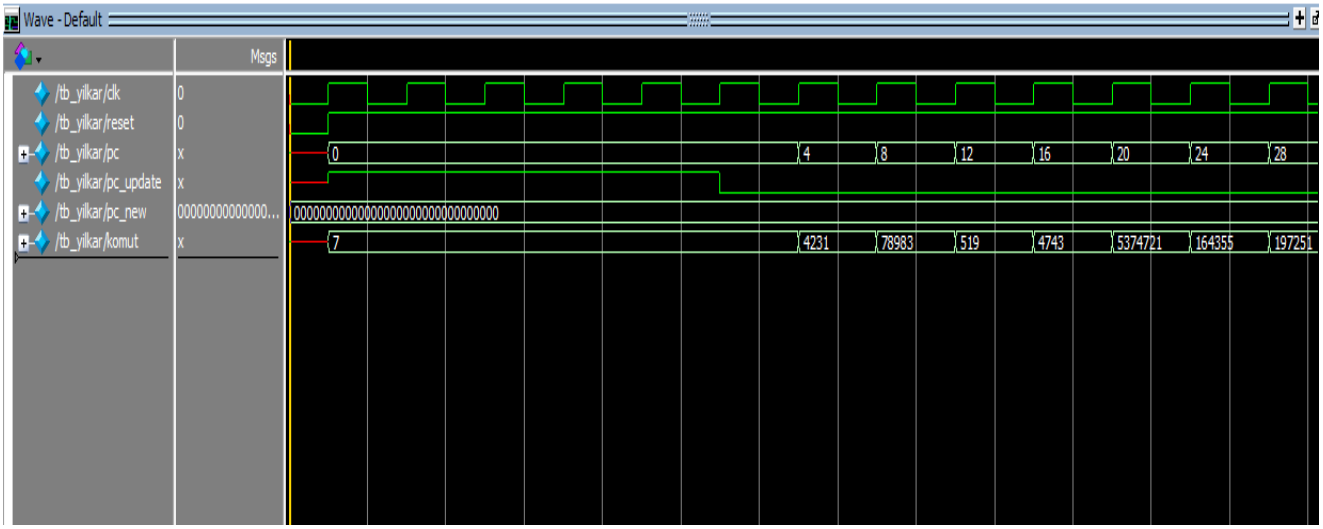
2.2. Problem II - Instruction Decode Ünitesi

2.2.1. Teorik Araştırma

Deneye başlamadan önce decode ünitesi hakkında bilgi sahibi olunmuştur. Devredeki bu ünitenin çalışma prensibi öğrenilmiştir.

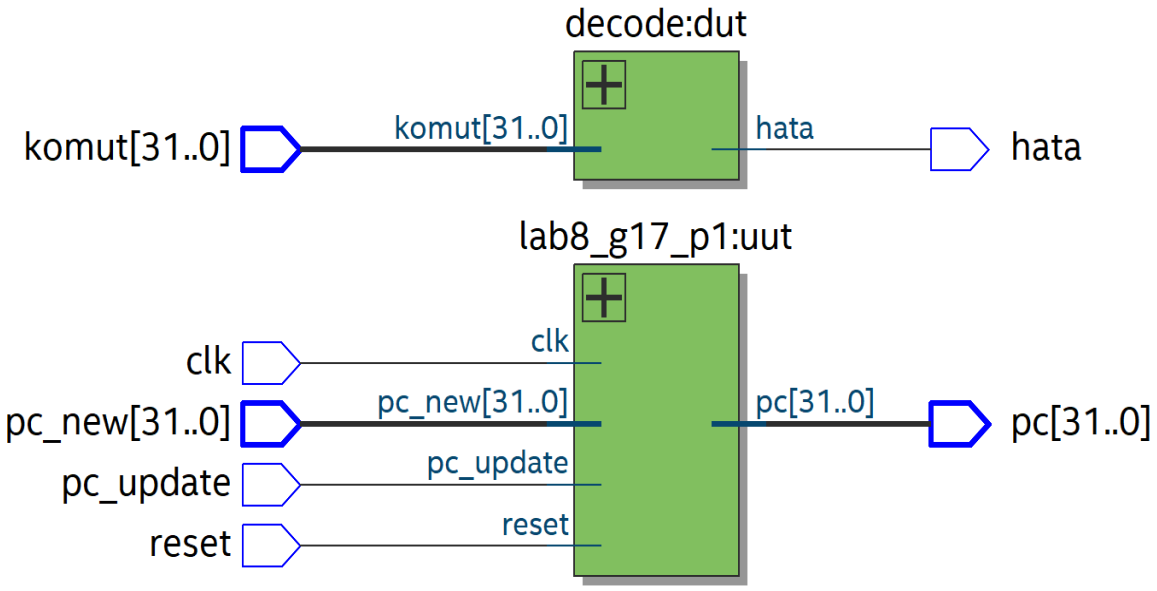
2.2.2. Deneyin Yapılışı

Bu deneyde öncelikle işlemcini ana bağlantılarını tutan devre tasarlanmıştır. Önceden tasarlanmış olan fetch ve decode üniteleri kullanılarak yeni bir modül oluşturulmuştur. Fetch ünitesindeki pc çıkışı ve decode ünitesindeki komut girişi tasarlanan yeni modüldeki pc çıkışı ve komut girişine bağlanmıştır. Lab 07 Problem 2'de oluşturulan devre Register File görevi görecek olup, komutlar için registerların değerlerini sağlamış ve tutacak hafıza birimi olmuştur.



Şekil 3. Simülasyon çıktısı

.



Şekil 4. RTL şeması

Şekil 4'da sıralayıcı devresinin RTL şeması görülmektedir.

2.2.3. Sonuçların Yorumu

Yapılan devre istenilen kriterlere göre tasarlanmıştır. Fetch ünitesindeki pc çıkışı ve decode ünitesindeki komut girişi tasarlanan yeni modüldeki pc çıkışı ve komut girişine başarıyla bağlanmıştır ve pc sinyali pc_update değerlerinin aktif olmadığı yerde değerini 4 arttırarak ilerlemiştir. Registerlar kullanarak komut sinyaline gerekli değerleri "fib20.mem" adlı dosyadan çektik. Komut sinyaline ait değerleri şekil 3'de gördük.

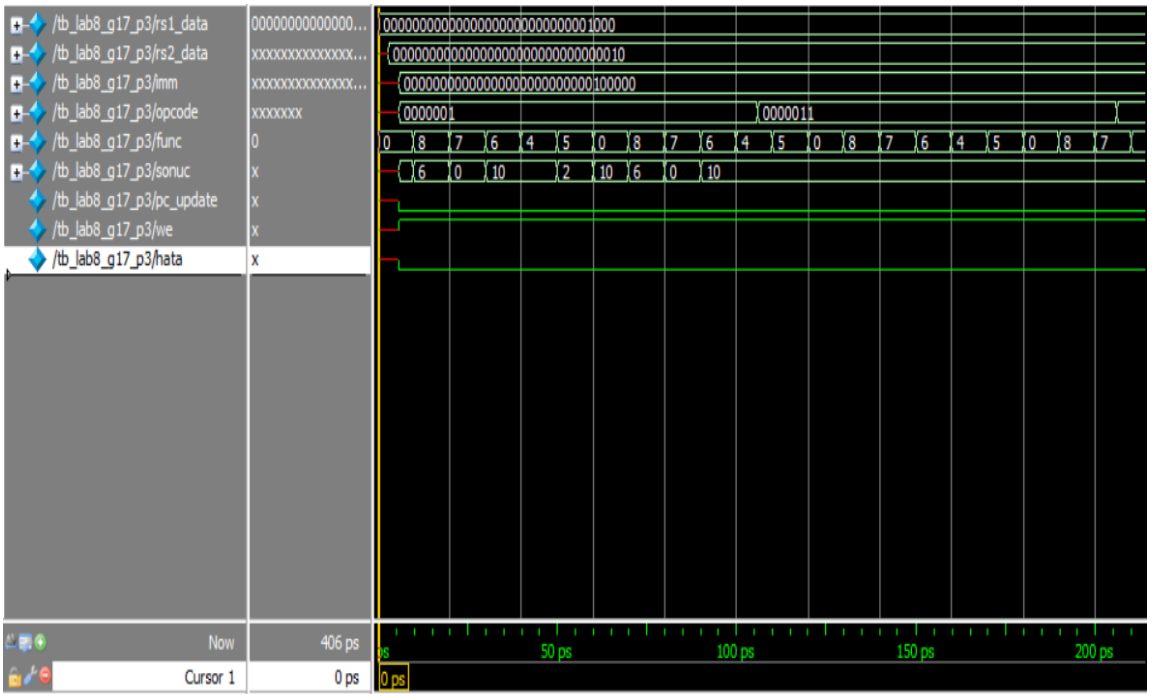
2.3. Problem III - Execute Ünitesi

2.3.1. Teorik Araştırma

Deneye başlamadan önce execute ünitesi hakkında araştırmalar yapılıp, devredeki bu ünitenin çalışma prensibi öğrenilmiştir.

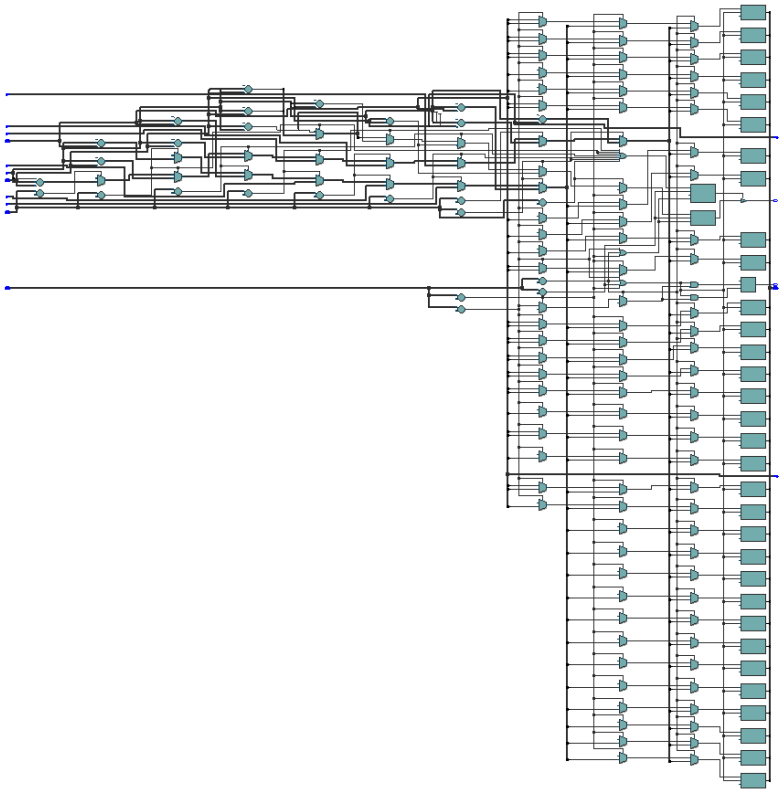
2.3.2. Deneyin Yapılışı

Bu deneyde öncelikle işlemcinin son parçası olan execute tasarlanmıştır. Öncelikle daha önceki deneyde tasarladığımız alu devresi kullanılarak execute devresi combinational olarak tasarlanmıştır. Func ve opcode giriş sinyallerine göre rs1_data, rs2_data ve imm sinyallerini seçen mux kullanılmıştır.



Şekil 5. Simülasyon çıktısı

.



Şekil 6. RTL şeması

Şekil 6'da devrenin RTL şeması görülmektedir.

2.3.3. Sonuçların Yorumu

Deney istenilen koşullara göre tasarlanmıştır. Devreyi tasarlamadan önce önceki deneylerde yapılan alu devresinden yardım alınmıştır. Şekil 5'de görüldüğü üzere , opcode değerlerini göz önüne alarak , girilen func sinyali değerlerine göre işlemler yaptırılmıştır. Rs1_data ve Rs2_data değerleri arasındaki işlem sonuçları kendimiz tarafından sonuç sinyalinde görülmüştür.

2.4. Problem IV - Tek Ritimli İşlemci Ünitesi

2.4.1. Teorik Araştırma

Deneye başlamadan önce kullanılacak olan devre modüllerinin nasıl bağlanılacağı araştırılmıştır.

2.4.2. Deneyin Yapılışı

Deney yapılışında, tasarlanan execute ünitesi ve instruction fetch ünitesi devre içerisinde çağırıp gerekli bağlantılar yapılmıştır lakin uzun süre uğraşılmasına rağmen bağlantı sonrası hatalarla karşılaşılmıştır ve gerekli simülasyon, rtl şema şekilleri elde edilememiştir.

2.4.3. Sonuçların Yorumu

Deneyde herhangi bir sonuca erişilemediği için sonuçların yorumu hakkında bilgi sahibi olunamamıştır.

3. Sonuçlar ve Genel Yorumlar

Bu laboratuvar çalışmasında, istenilen bir sorunun çözümüne ulaşmak için adım adım modüllerin nasıl bağlanılacağı hakkında bilgi sahibi olup, sonrasında bunun koda aktarımı tam anlamıyla anlaşılmıştır. Problemlerde bizden istenilenler anlaşılıp sırasıyla modüller tasarlanıp ve birbiriyle bağlanılmıştır. Deney yapılırken sorunlarla karşılaştık lakin aşama aşama sorunu tahlil ederek ilerlemek, problemlerde başarıya ulaşmamızda en büyük etkendi. Yazılan kodların simülasyon ekranında çıktısı, bize sorunu istenen çözümün doğruluğu veya yanlışlığı hakkında fikir verdi.

Genel itibariyle Tek Ritimli İşlemci devresini tasarlamak, bize tecrübe olarak geri döndü.

4. Referanslar

[1] Ders slaytları

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

/Bir sonraki sayfadan itibaren kodlar başlamaktadır/

KODLAR

```
/*lab8_g17_p1.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab8 - Problem 1

* Fetch ünitesi

*/

module lab8_g17_p1(
input logic clk,reset,
output logic [31:0]pc,
input logic pc_update,
input logic [31:0]pc_new
);

 logic [31:0] pc_reg;

 always_ff @(negedge reset,posedge clk)
 begin

     pc <= pc_reg;

     if(!reset)
     begin
     pc_reg=0;
     end
     else if(clk)
     begin
        if(pc_update==1)
        begin
         pc_reg<=pc_new;
        end
        else
        begin
        pc_reg<=pc_reg+4;
        end
     end
 end
endmodule
```

```
/*tb_lab8_g17_p1.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab8 - Problem 1

* Fetch ünitesi testbench

*/

`timescale 1ns/1ps
module tb_lab8_g17_p1();
logic clk,reset;
logic [31:0]pc;
logic pc_update;
logic [31:0]pc_new;
logic [31:0] komut;

lab8_g17_p1 uut0(.clk(clk),.reset(reset), .pc(pc),
.pc_update(pc_update),.pc_new(pc_new));

logic [31:0] mem [255:0];

assign komut = mem[pc>>2];

initial begin
$readmemb("fib20.mem",mem);
 end

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

pc_update=1; #100;
pc_update=0; #400;
pc_update=1; #100;
pc_update=0; #10;

end

initial begin
pc_new=32'b0;

#1500;
$stop;
end

endmodule
```

```
/*yilkar.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab8 - Problem 2

* Decode ünitesi

*/

module yilkar(
 input logic clk,reset,
 input logic [31:0] komut,
 output logic [31:0]pc,
 output logic hata,
 input logic pc_update,
 input logic [31:0]pc_new

);

lab8_g17_p1 uut (.clk(clk), .reset(reset), .pc(pc), .pc_new(pc_new),
.pc_update(pc_update));
decode dut(.komut(komut), .hata(hata));

endmodule

module decode (
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;


    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;

    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;

    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;

    end

    else
    begin

    hata = 1;
    rs1 = 5'b0;
    rs2 = 5'b0;
    rd = 5'b0;
    aluop = 4'b0;
    imm = 32'b0;



    end

end

endmodule
```

```
/*tb_yilkar.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab8 - Problem 2

* Decode ünitesi testbench

*/

`timescale 1ns/1ps
module tb_yilkar();
logic clk,reset;
logic [31:0]pc;
logic pc_update;
logic [31:0]pc_new;
logic [31:0] komut;


yilkar uut0(.clk(clk),.reset(reset), .pc(pc),.komut(komut),
.pc_new(pc_new), .pc_update(pc_update));

logic [31:0] mem [255:0];


assign komut = mem[pc>>2];


initial begin
$readmemb("fib20.mem",mem);
 end


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

pc_update=1; #100;
pc_update=0; #400;
pc_update=1; #100;
pc_update=0; #10;

end

initial begin
pc_new=32'b0;#500;
pc_new=32'b1;
#1500;
$stop;
end

endmodule
```

```
/*lab8_g17_p3.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab8 - Problem 3

* Execute ünitesi

*/

module lab8_g17_p3 (
input logic [31:0] rs1_data,rs2_data,
input logic [31:0] imm,
input logic [6:0] opcode,
input logic [3:0] func,
output logic [31:0] sonuc,
output logic pc_update,
output logic we,
output logic hata
);

always_comb
case(opcode)

opcode==7'b0000001: begin //R
    hata = 0;
    pc_update = 0;
    we = 1;
        if (func==4'b0000) //ADD
        sonuc = $signed(rs1_data) + $signed(rs2_data);
        else if (func==4'b1000)
        sonuc = $signed(rs1_data) - $signed(rs2_data);
        else if (func==4'b0111)
        sonuc = $signed(rs1_data) & $signed(rs2_data);
        else if (func==4'b0110)
        sonuc = $signed(rs1_data)| $signed(rs2_data);
        else if (func==4'b0100)
        sonuc = $signed(rs1_data)^ $signed(rs2_data);
        else if (func==4'b0001)
        sonuc = $signed(rs1_data) << $signed(rs2_data);
        else if (func==4'b0101)
        sonuc = $signed(rs1_data) >> $signed(rs2_data);
        else if (func==4'b1101)
        sonuc = $signed(rs1_data) >>> $signed(rs2_data);
        else
        sonuc = 32'b0;
        end
```

```
opcode==7'b0000011: begin //I
        hata = 0;
        we = 1;
        pc_update = 0;
        if (func==4'b0000) //ADDI
        sonuc = $signed(rs1_data) + $signed(imm);
        else if (func==4'b0111)
        sonuc = $signed(rs1_data) & $signed(imm);
        else if (func==4'b0110)
        sonuc = $signed(rs1_data) | $signed(imm);
        else if (func==4'b0100)
        sonuc = $signed(rs1_data) ^ $signed(imm);
        else if (func==4'b0001)
        sonuc = $signed(rs1_data) << $signed(imm);
        else if (func==4'b0101)
        sonuc = $signed(rs1_data) >> $signed(imm);
        else
        sonuc = 32'b0;
        end
opcode==7'b0000111: begin
    hata = 0;
    we = 1;
    pc_update =0;
    sonuc = {12'b0,imm[19:0]};
    end
opcode==7'b0001111: begin //B
        hata = 0;
        we = 0;
        sonuc = {20'b0,imm[12:6] , imm[5:1]};
        if (func==4'b0011)//B
        pc_update = 1;
        else if(func==4'b0000)
        pc_update = ($signed(rs1_data)==$signed(rs2_data))?1:0;

        else if (func==4'b0001)
        pc_update = ($signed(rs1_data)!=$signed(rs2_data))?1:0;

        else if (func==4'b0100)
        pc_update = ($signed(rs1_data)<$signed(rs2_data))?1:0;

        else if (func==4'b0101)
        pc_update = ($signed(rs1_data)>=$signed(rs2_data))?1:0;

        else if (func==4'b0101)
        pc_update = ($unsigned(rs1_data)<$unsigned(rs2_data))?1:0;

        else if (func==4'b0111)
        pc_update = ($unsigned(rs1_data)>=$unsigned(rs2_data))?1:0;
        else
        pc_update = 1'bz;
    end
    endcase

endmodule
```

```
/*tb_lab8_g17_p3.sv

*Hazırlayanlar:

* Alperen Karataş - Ogün Uygar Yıldırım

* ELM235 2020 Bahar Lab8 - Problem 3

* Execute ünitesi testbench

*/

module tb_lab8_g17_p3 ();
logic [31:0] rs1_data,rs2_data;
logic [31:0] imm;
logic [6:0] opcode;
logic [3:0] func;
logic [31:0] sonuc;
logic pc_update;
logic we;
logic hata;

lab8_g17_p3
dut0(.rs1_data(rs1_data),.rs2_data(rs2_data),.imm(imm),.opcode(opcode),.
func(func),.sonuc(sonuc),.pc_update(pc_update),.we(we),.hata(hata));

initial begin
rs1_data = 32'd8;#3;
rs2_data = 32'd2;#3;
imm = 32'd32;
opcode = 7'b0000001; #100;
opcode = 7'b0000011; #100;
opcode = 7'b0000111; #100;
opcode = 7'b0001111; #100;
$stop; #1500;
end

always begin
func = 4'b0000; #10;
func = 4'b1000; #10;
func = 4'b0111; #10;
func = 4'b0110; #10;
func = 4'b0100; #10;
func = 4'b0101; #10;

end

endmodule
```