

## GEBZE TEKNİK ÜNİVERSİTESİ ELEKTRONİK MÜHENDİSLİĞİ

## ELM 235 LOJİK DEVRE TASARIM LABORATUVARI

Komut Ayrıştırma

LAB 0x6 Deney Raporu

Lojik Devreler ve Tasarım Laboratuvarı

Hazırlayanlar

- 1) 1801022024 M. Cemal Eryiğit
- 2) 1801022077 Burak Kamil Çiftci

## Bu labin amacı

- Komut parçakalama devresi tasarlama
- Tasarlanan devreleri gerçekleyip test edebilmek.

## Problem 1 - Komut ayırıcı

Bu problemde 32-bit gönderilen bir komutu, aşağıda verilen isterlere uygun olarak parçalayıp, gerekli çıkışları üreteceksiniz. Bütün devre kombinasyonel lojik olarak çalışacaktır. (clk ve reset pinleri kullanılmayacaktır). Gelen komutlar dört farklı tipte olabilir.

Sizin yapmanız gereken, öncelikle opcode bölümünü kontrol edeceksiniz. Tablo 6 te verilen değerlerden bir tanesi gelmiş ise ona göre Tablo 1,2,3 veya 4 e göre geri kalan bitleri ayıracaksınız. Eğer Tablo 6 te verilen değerlerden farklı bir değer gelirse hata biti 1 olacak (default olarak 0)

- Eğer R tipi bir komut geldiyse rs1, rs2 ve rd değerlerine direkt atama yapacaksınız, aluop çıkışına da fonksiyonda gösterilen formüle göre oluşturup atama yapacaksınız. (bit30, bit14, bit13, bit12). imm çıkışı 0 olacak.
- Eğer I tipi bir komut geldiyse rs1 ve rd değerlerine direkt atama yapacaksınız, aluop çıkışına da fonksiyon bitlerini soldan 0 ekleyerek atama yapacaksınız. (0, bit14, bit13, bit12). imm çıkışı imm12 olacak. diğer çıkışlar 0 olacak.
- Eğer U tipi bir komut geldiyse rs1 ve rd değerlerine direkt atama yapacaksınız. imm çıkışı imm20 olacak. diğer çıkışlar 0 olacak.
- Eğer B tipi bir komut geldiyse rs1, rs2 değerlerine direkt atama yapacaksınız, aluop çıkışına da fonksiyonda gösterilen formüle göre oluşturup atama yapacaksınız. (0, bit14, bit13, bit12). imm çıkışı LSB si 0 olacak şekilde 13 bitlik imm13 olacak.

Not: rs1 data ve rs2 data değerlerine her zaman 0 atayın.

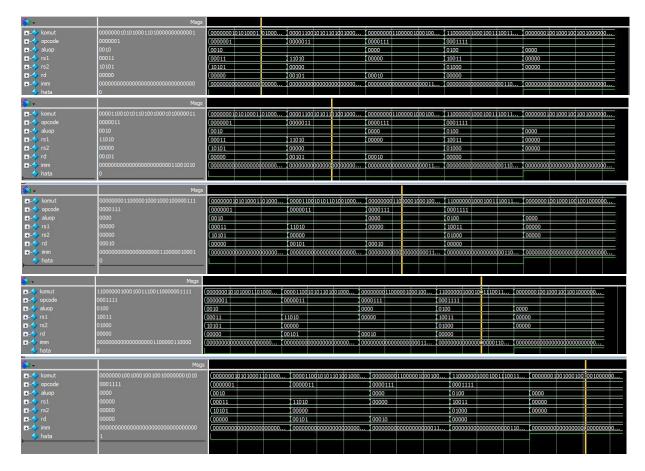
- A. Bu problemin testbench inde, her bir komut tipi için birkaç bit vektörü oluşturup test edin.
- B. Kapsamlı bir test vektörü oluşturun ve bu testvektörünü kullanarak test edin

```
module lab6_g24_p1 (
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
);
always_comb
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];
        imm=32'd0;
        rs1_data=32'b0;
        rs2_data=32'b0;
        opcode=komut[6:0];
        hata= 1'b0;
end
```

```
else if(komut[6:0] == 7'b0000011)
begin
       rs1 = komut[19:15];
       rd = komut[11:7];
       aluop[3] = 1'b0;
       aluop[2:0] = komut[14:12];
       imm[11:0] = komut[31:20];
       imm[31:12] = 0;
       rs2 = 5'b00000;
       rs2_data=32'b0;
       rs1_data=32'b0;
       opcode=komut[6:0];
       hata = 1'b0;
end
else if(komut[6:0] == 7'b0000111)
begin
       rd = komut[11:7];
       imm[19:0] = komut[31:12];
       imm[31:20]=12'b0;
       rs1_data=32'b0;
       rs2_data=32'b0;
       rs2 = 5'b00000;
       rs1 = 5'b00000;
       aluop = 4'b0000;
       opcode=komut[6:0];
       hata = 1'b0;
end
```

```
else if(komut[6:0] == 7'b0001111)
begin
       rs1 = komut[19:15];
       rs2 = komut[24:20];
       aluop[3] = 1'b0;
       aluop[2:0] = komut[14:12];
       imm[0] = 1'b0;
       imm[5:1] = komut[11:7];
       imm[12:6] = komut[31:25];
       imm[31:13] = 7'b0; imm[1] = 1'b0;
       rs1_data=32'b0;
       rs2_data=32'b0;
       rd = 5'b00000;
       opcode=komut[6:0];
       hata = 1'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
```

```
`timescale 1ns/1ps
                                                           //tb
module tb_lab6_g24_p1();
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;
lab6_g24_p1 dut0 ( .rs1(rs1), .rs2(rs2), .komut(komut), .rd(rd), .imm(imm) , .hata(hata), .opcode(opcode), .aluop(aluop));
initial begin
komut = 32'b000000101010001101000000000001; #600;
komut = 32'b0000110010101010010001010000011; #600;
komut = 32'b0000000110000010001000100000111; #600;
komut = 32'b11000000100010011100110000001111; #600;
komut = 32'b000000100100100100100000001010; #600;
#100;
/*int fd;
bit [31:0] mask;
fd = $fopen ("lab6_p1_testvector.txt", "r");
      while (!$feof(fd))
      begin
      mask = (31'b1 << 32);
komut = fd ^ (mask & 32'bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX); #600;
      end */
$stop;
end
endmodule
```



Şekil 1 Sırası ile R I U B ve hata=0 dalga gösterimleri

Bu sorunun A şıkkında 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. 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 2 testvektoru dalga formu

Bu sorunun B şıkkında ise bir test vektörü oluşturup oluşturduğumuz test vektörü ile test etmemiz beklenmektedir. Dosya okuma yapıp istenilen değerleri bir değişkene atayıp komut ayırma işlemi testbench dosyasında yazılmaya çalışılmıştır fakat yazdığımız kısımda shift olmamakla beraber sonsuz döngüye girmektedir. Bu kısım kodda yorum satırına alınmıştır. Şekil 2 de ise testvektörünün girdiği loopu kırarak dosyanın sonundaki komut gözlemlenmiştir.