FENERBAHÇE ÜNİVERSİTESİ
Bilgisayar Mühendisliği Bölümü
İstanbul, Türkiye

11 KOMUTLUK RISC-V İŞLEMCİ TASARIMI

COMP-202 Bilgisayar Mimarisi
2020-2021 Bahar Dönemi

Proje Teslim Sunumu

RECEP GEMALMAZ
200301501
recepgemalmaz@stu.fbu.edu.tr

ALP YILMAZ
190301017
alp.yilmaz@stu.fbu.edu.tr

ÖZET

Günümüzde kullanılan elektronik cihazları pazarda öne çıkaran iki önemli özelliği hızı ve maliyetidir. Elektronik cihazları maliyet açısından en çok zorlayan etken ise işlemcisidir ki bu aynı zamanda cihazın hızı üzerinde büyük etkisi olan bir elemandır. Günümüzde şirketler kendi işlemcisini kendisi üretiyor veya bu konuda deneyimli olan şirketlerin tasarlamış olduğu işlemcileri kullanıyor. İşlemci mimarisinde bir ortak standartın olmaması ise şirketleri farklı farklı amaca yönelik işlemci tasarımına yöneltiyor. İşlemci mimarisinin bu karmaşık ve çoklu yapısı yüzünden bir işlemciyi oluşturmak ve bunu kullanabilmek çok karmaşık bir işlem olarak görülüyor. Son yıllarda bu konuyu bir çözüme kavuşturmak ve ortak standartı olan bir işlemci tasarımı geliştirmek için RISC-V projesi başlatıldı.

Bu projenin amacı temel bir işlemcinin çalışma prensibini öğretmeyi amaçlamaktadır. FPGA kartları, sahada programlanabilen mantık blokları ve bloklar arası bağlardan oluşan sayısal tümleşik devrelerdir. Geniş programlama alanları ve amaca yönelik kullanımı nedeniyle bu projedeki olmazsa olmazlarımızdan biridir Açık kaynak kodlu donanım ISA (Instruction Set Architecture)'sı olarak geçen RISC-V ile yapılan çalışmalar günümüzde FPGA programlamada gittikçe yükselen bir trend haline gelmiştir.RISC-V'in basit işlemci mimarisinden biri olan FPGA'da kullanılmak için VHDL'de yazılmış birçok uygulama gerçeklenebilmektedir. Bizde bu uygulama ve yazılımlara projemizde yer verdik.

ABSTRACT

Two important features that make electronic devices used today stand out in the market are speed and cost. The most challenging factor for electronic devices in terms of cost is the processor, which is also an element that has a great impact on the speed of the device. Today, companies produce their own processors or use processors designed by companies experienced in this field. The lack of a common standard in the processor architecture leads companies to different processor designs for different purposes. Because of this complex and multiple structure of the processor architecture, creating and using a processor is seen as a very complex process. In recent years, the RISC-V project has been initiated to resolve this issue and develop a common standard processor design.

The aim of this project is to teach the working principle of a basic processor. FPGA boards are digital integrated circuits consisting of field programmable logic blocks and inter-block links. It is one of our essentials in this project due to its wide programming areas and its purposeful use. The work done with the open source hardware ISA (Instruction Set Architecture), RISC-V has become an increasing trend in FPGA programming today. Many applications written in VHDL can be implemented to be used in FPGA, one of its architecture. We have included these applications and software in our project.

---- 1-GİRİŞ

1.1 Projenin Tanımı Ve Amacı

- **Bu proje kapsamında Risc-v tabanlı bir işlemcinin System Verilog dili ile RTL tasarımı ve tasarlanan işlemci üzerinde makine dili ile yazılan çeşitli kod parçacıkları yazılacaktır.**
- **İşlemcinin her adımı farklı modüllerde yapılmıştır. Tek modül altında toplanmamıştır.**
- **Bu proje kapsamında bir RISC-V işlemcisinin ALU ve instruction decoder blokları temel SystemVerilog dili özellikleri kullanılarak tasarım ve doğrulama çalışmaları yapılacaktır.**
- **Proje sonunda basit bir işlemcideki RAM, Kontrol Ünitesi ve Saklayıcıların bir arada çalışıp, makine dilindeki kod parçacıklarını nasıl yürütebildiği gözlemlenecektir.**
- **Risc-v tabanlı bir işlemcinin ALU'sunun destekleyeceği 11 adet işlem vardır. Bu komutlar test yazılımları kullanarak test edilecektir.**
- **Bu projenin amacı temel bir işlemcinin çalışma prensibini öğretmeyi amaçlamaktadır.**

-----2-SİSTEM MİMARİSİ

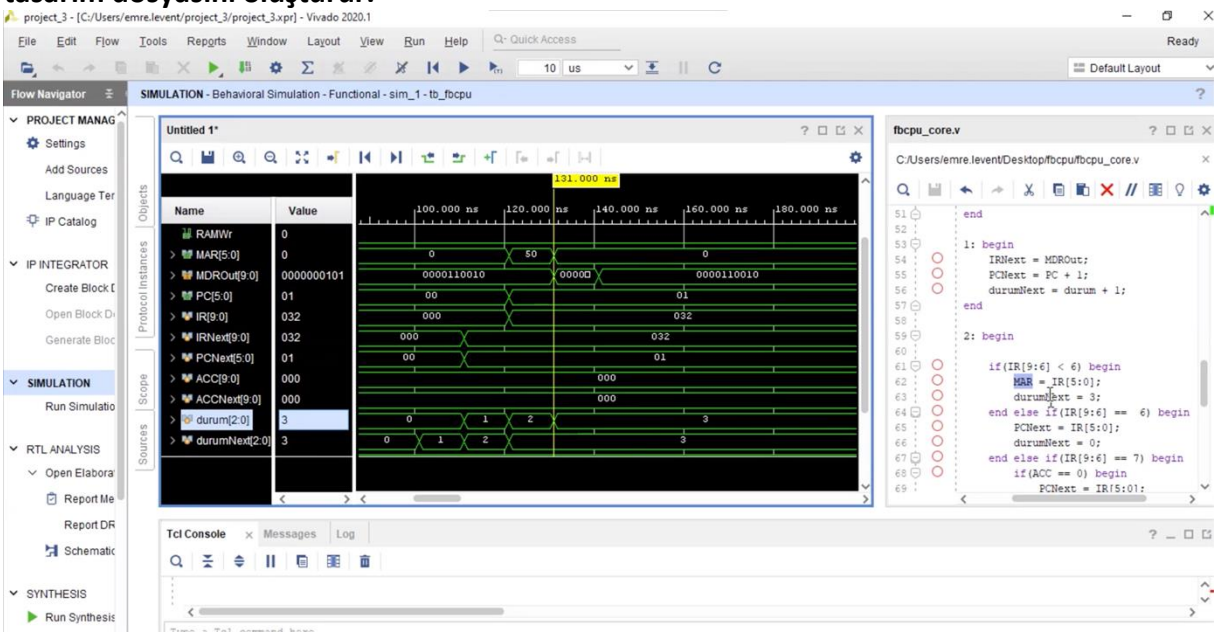
2.1-KULLANILAN ARAÇLAR

Proje kapsamında 1 araç kullanılmıştır.

- **Xilinx Vivado Design Suite**

2.1.1 Xilinx Vivado Design Suite:

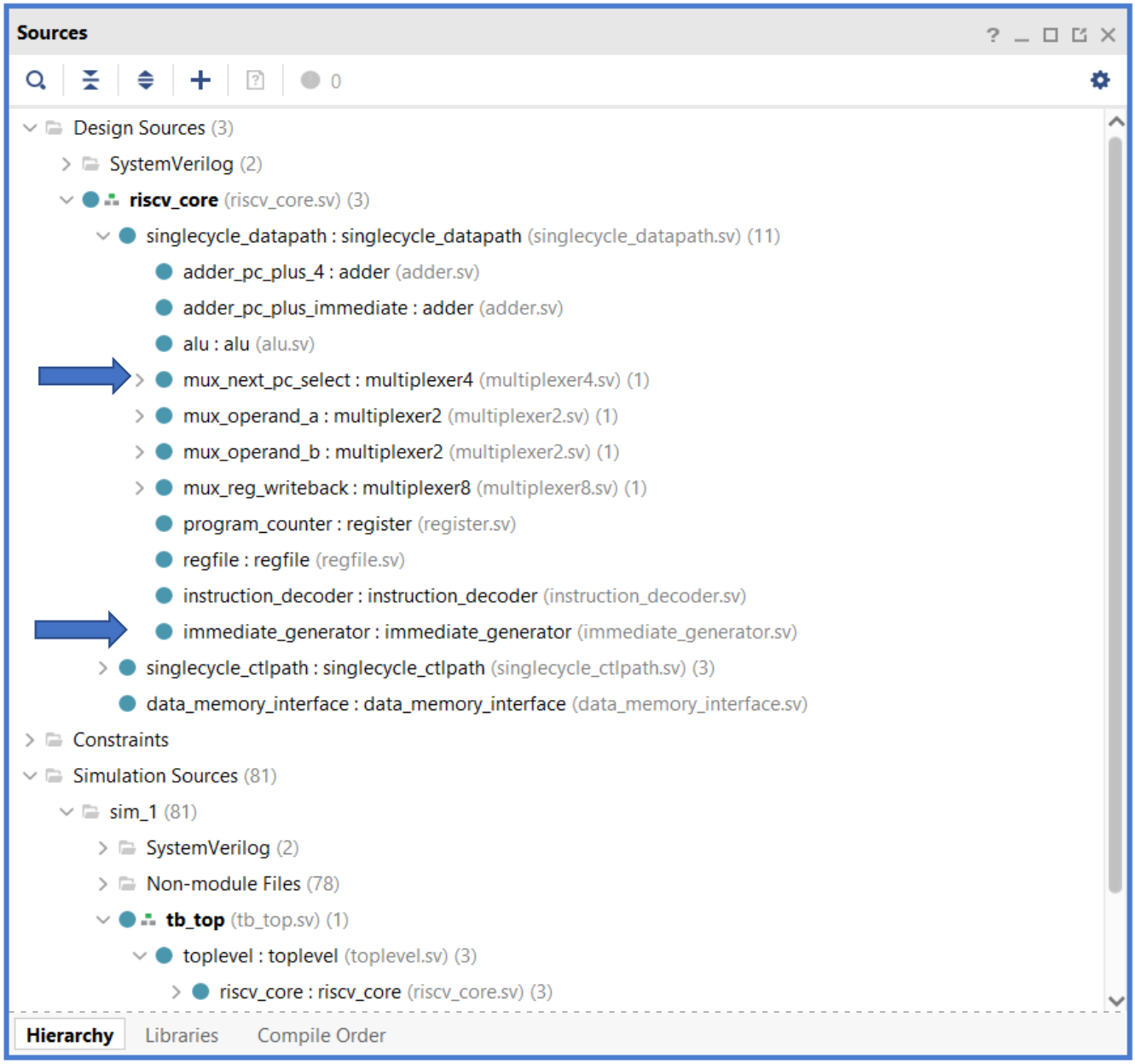
Xilinx Vivado Design Suite, FPGA geliştirme kartları üzerinde çalışmalar yapmak için gerekli olan tasarımı oluşturmak için kullanılmaktadır. Verilog, VHDL vb. donanım tasarım dillerini alarak, FPGA'e konfigüre edilebilecek (Xilinx firması FPGA'leri için .bit uzantılı dosyalar) tasarım dosyasını oluşturur.



Şekil 1. Xilinx Vivado Design Suite

Rısc-v İŞLEMCİSİ tepe modülü ve test(Similasyon) dosyası

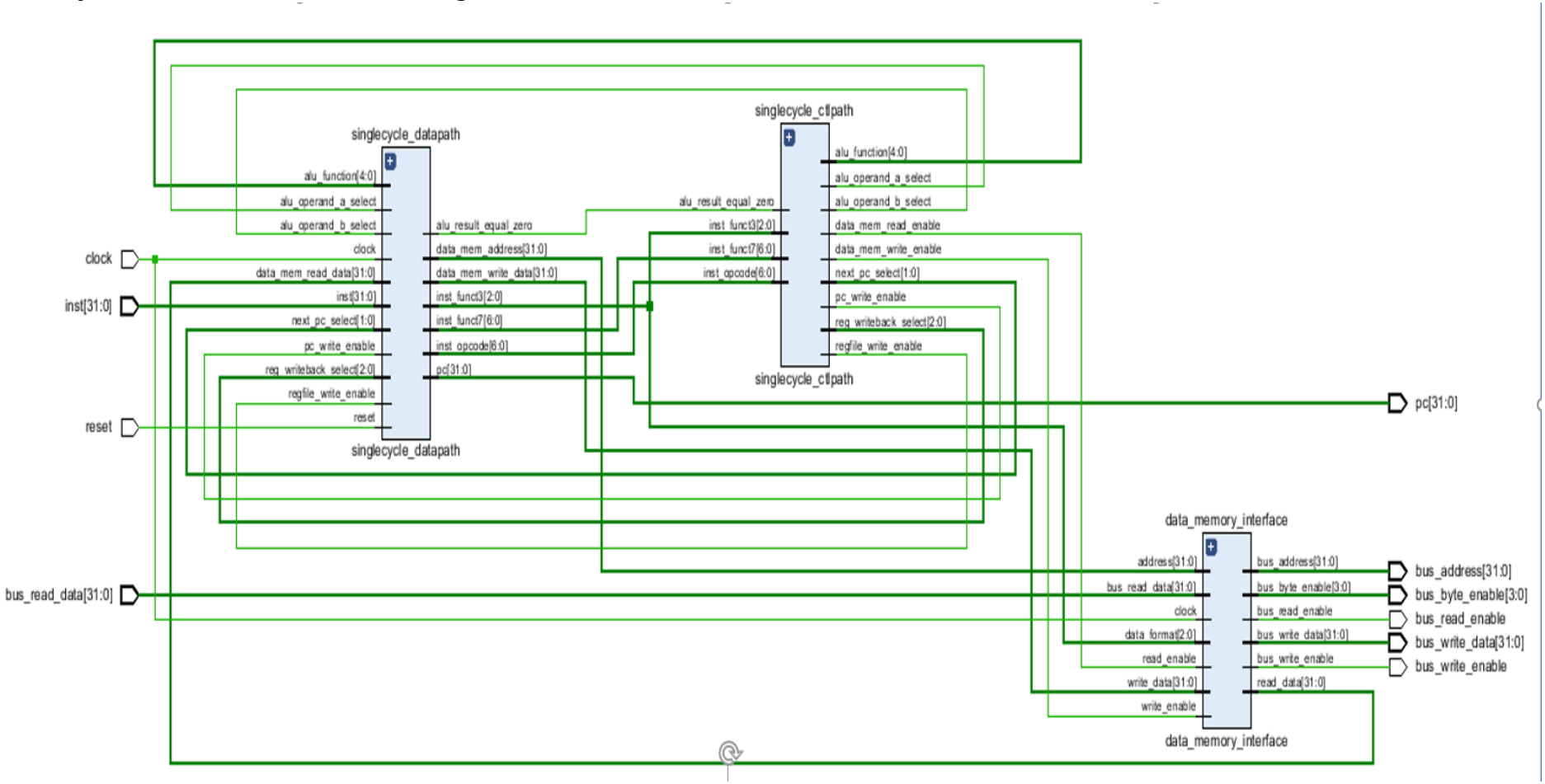
- **risc-v_core dosyası altındaki alt mödüller aşağıda gösterildiği şekildedir.**
- **Her modülün tasarımı system verilog(.sv) dili ile gerçekleştirilmiştir.**
- **Projenin kapsamında risc-v_core alt modülü olan alu ve instruction_decoder modüllerinin tasarımı yapılacaktır.**
- **tb_top modülü risc-v_core modülünüde içere bir similasyon dosyasıdır. Rısc-v işlemcisini test etmek için kullanılıcaktır.**



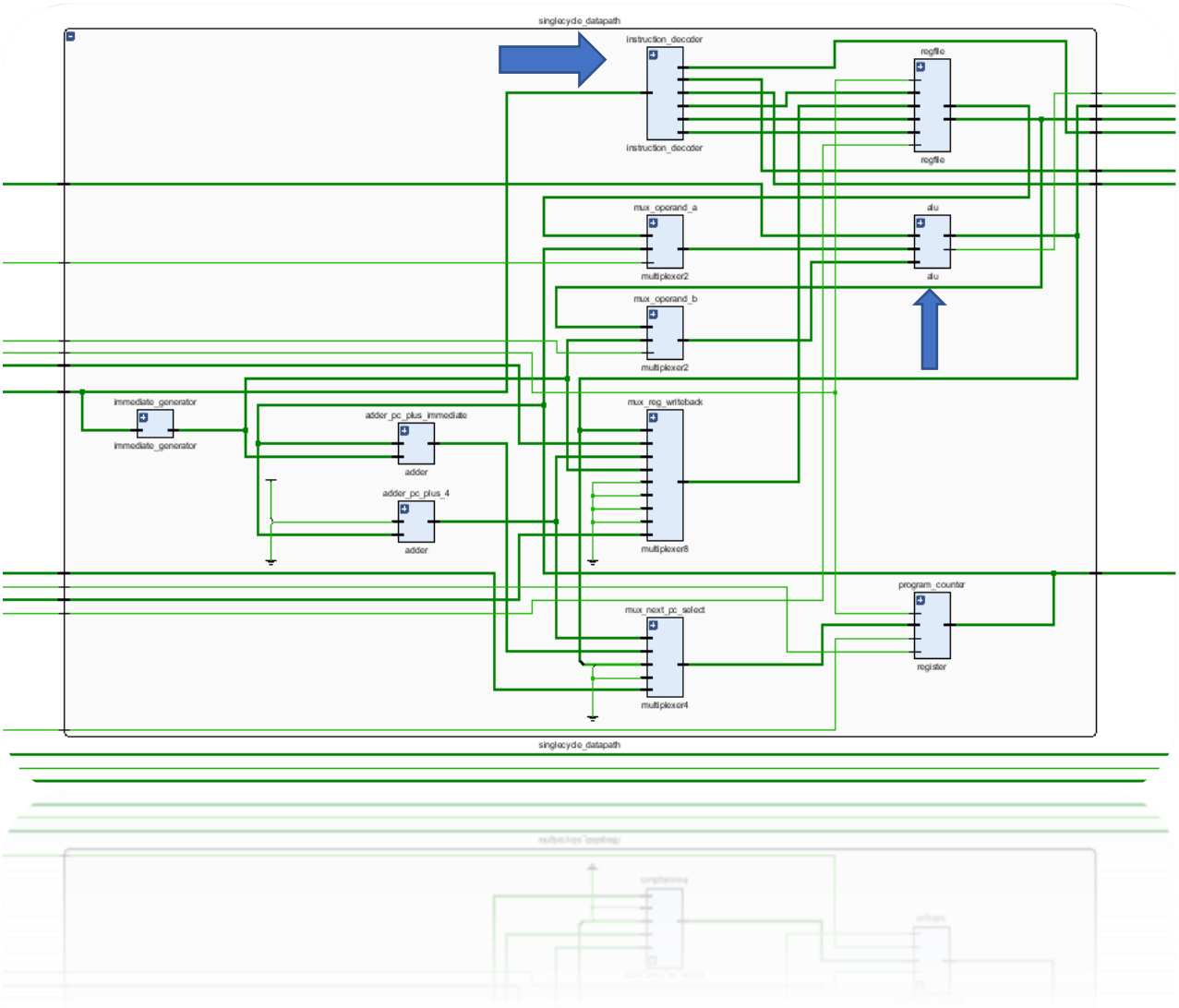
2.2 TASARIM

2.2.1

Risc-v işlemcisinin modüller arasındaki bağlantıları



2.2.2
singlecycle_datapath

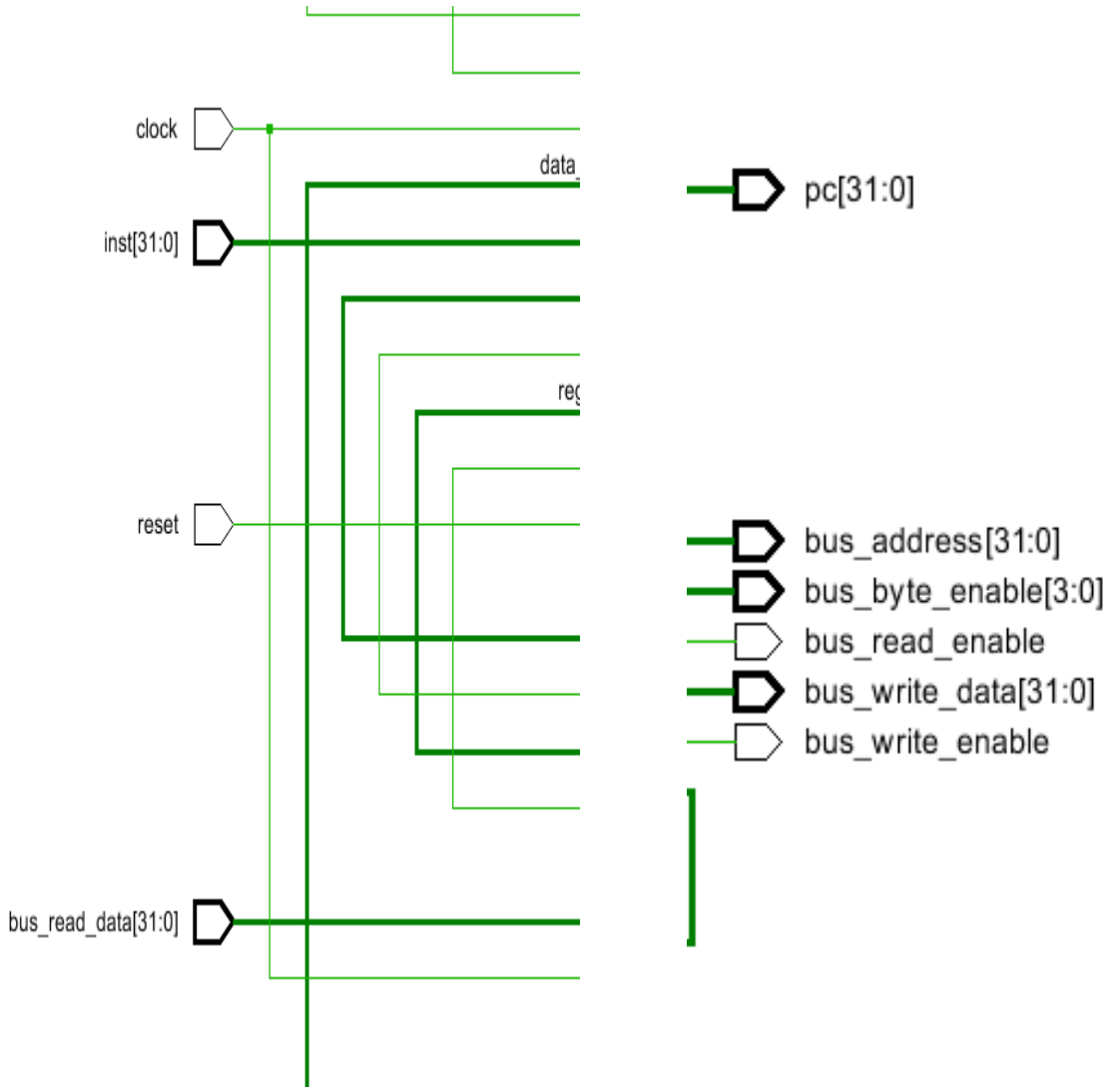


- **Risc-v işlemcisi single cycle tasarlanmıştır.**
- **Tasarımı yapılacak olan alu ve instruction_decoder modülleri singlecycle_datapath içerisindedir.**

2.2.3

Risc-v işlemcisinin giriş ve çıkışları

GİRİŞLER VE ÇIKIŞLAR



Risc-v işlemcisi 32 bitlik instruction girişi, clock, reset ve 32 bitlik bus_read_data girişleri vardır.

- **Şekilde gösterilen Risc-v işlemcisinin Bellekle olan ilişkileri için 32 bitlik bus_address,4 bitlik bus_byte_enable,1 bitlik bus_read_enable,32 bitlik bus_write_data,1 bitlik bus_write_enable olan vede 32 bitlik pc çıkışı vardır.**
- **Bir işlemci temelinde ram'e birşeyler yazıp ve okuyan bir yapıdır.**

2.3 Tasarım Gereksinimleri

2.3.1 instruction_decoder

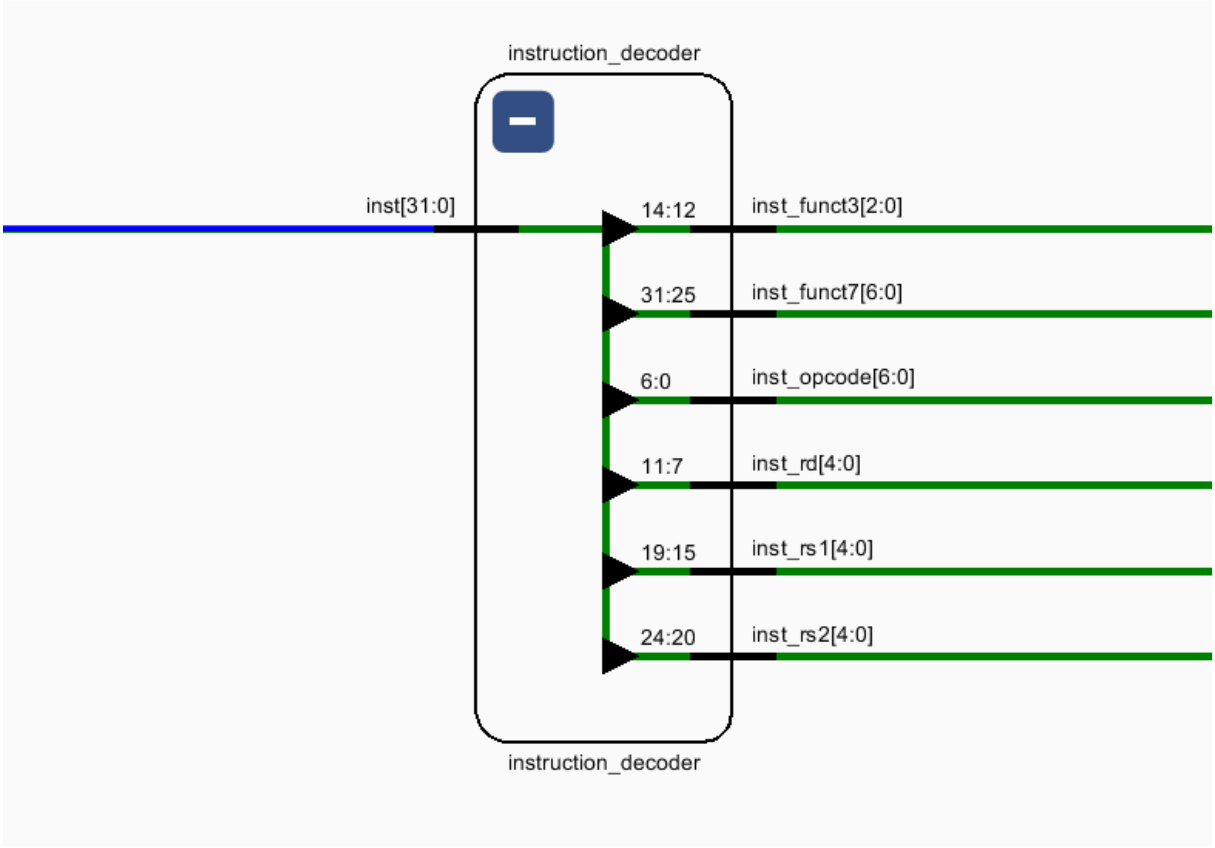
```
module instruction_decoder(
    input [31:0] inst,
    output [6:0] inst_opcode,
    output [2:0] inst_funct3,
    output [6:0] inst_funct7,
    output [4:0] inst_rd,
    output [4:0] inst_rs1,
    output [4:0] inst_rs2
);
```

- **Bu modülde giriş olarak 32 bitlik instruction word'u alınmaktadır.**
- **Çıkışta ise instruction'un parse edilmiş hali, yani decode edilmiş hali çıkış olarak verilmektedir.**
- **Opcode, instruction'un ilk 7 bitini yani [6:0]'ı temsil etmekte.**
- **Func3, instruction'un 14-12 bitleri arasını [14:12].**
- **Func7, instruction'un 31-25 bitleri arasını [31:25].**
- **Rd, instruction'un 11-7 bitleri arasını [11:7].**
- **RS1, instruction'un 19-15 bitleri arasını [19:15].**
- **RS2, instruction'un 24-20 bitleri arasını [24:20].**
- **Buna göre instruction sinyalini parçalayarak ilgili sinyallerin üzerlerine system verilog dilinde gerekli atama yapılmıştır.**

System Verilog dilinde instruction decoderin tasarımı aşağıdaki fotoğrafta görüldüğü gibi tasarlınmıştır, böylece instruction sinyalini parlayarak ilgili sinyallerin üzerine gerekli atamalar yapılmıştır.

```
`include "config.sv"
`include "constants.sv"

module instruction_decoder(
    input [31:0] inst,
    output [6:0] inst_opcode,
    output [2:0] inst_funct3,
    output [6:0] inst_funct7,
    output [4:0] inst_rd,
    output [4:0] inst_rs1,
    output [4:0] inst_rs2
);
    logic [6:0] opcode;
    logic[4:0]  rd;
    logic [2:0] funct3;
    logic [4:0] rs1;
    logic [4:0] rs2;
    logic [6:0] funct7;

    always_comb begin
        opcode = inst[6:0];
        rd= inst[11:7];
        funct3 = inst[14:12];
        rs1 = inst[19:15];
        rs2 = inst[24:20];
        funct7= inst[31:25];
    end
    assign inst_opcode = opcode;
    assign inst_rd = rd;
    assign inst_funct3 = funct3;
    assign inst_rs1 = rs1;
    assign inst_rs2 = rs2;
    assign inst_funct7 = funct7;

endmodule
```

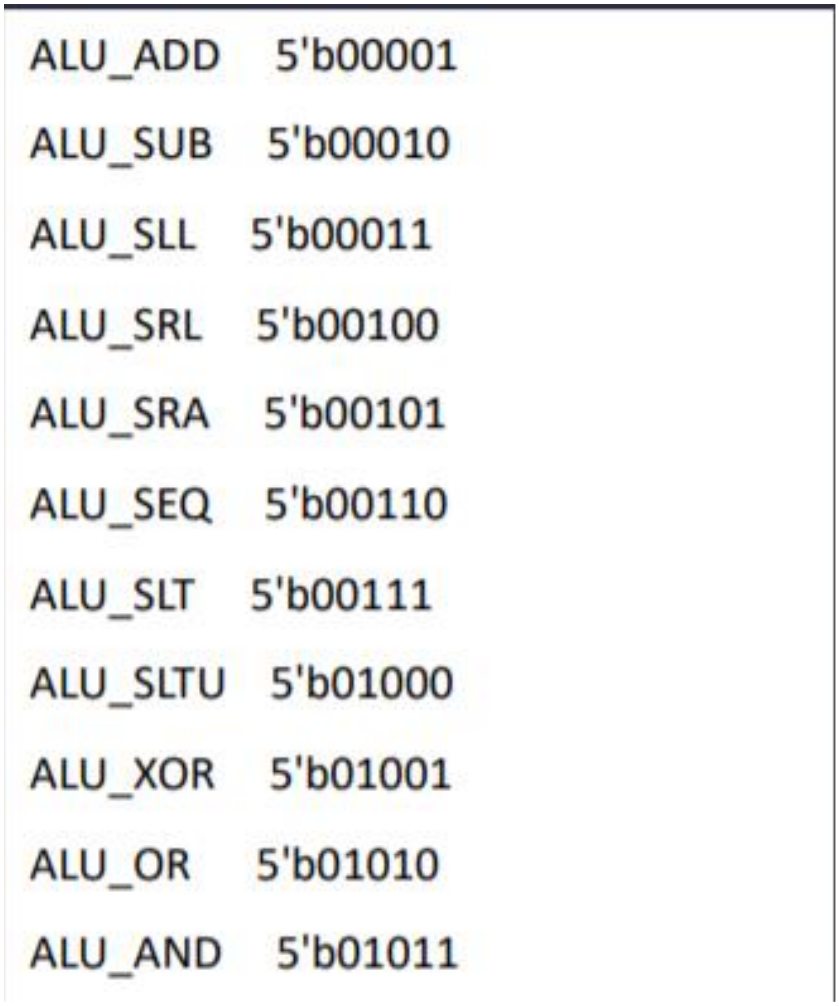
İnstraction decoder'a ait kod parçacığı vivado design studio programında çalıştırıldığı zaman
Şematik gösterim aşağıdaki şekilde olacaktır.



2.3.2 arithmetic logic unit

```
module alu (
    input          [4:0]  alu_function,
    input signed   [31:0] operand_a,
    input signed   [31:0] operand_b,
    output logic   [31:0] result,
    output                result_equal_zero
);
```

- **İşlemcinin ALU'sunun destekleyeceği 11 adet işlem vardır.**
- **Bu işlemlerden hangisinin yapılacağı alu_function girişinden gelmektedir.**
- **İşlemlere göre a ve b sayıları, result isminde sonuç çıkışı ve sonuç eğer sıfır ise, ayrı bir çıkış olarak sonucun sıfır olması durumunda 1 olan bir çıktı vardır.**
- **Resimde ALU ünitesinin giriş ve çıkış sinyalleri gösterilmektedir.**



- **Yukarıdaki tabloda ALU'nun desteklediği işlemler ve operasyon kodları verilmektedir.**
- **ADD: A + B**
- **SUB: A - B**
- **SLL: A << B**
- **SLR: A >> B**
- **SRA: A >>> B**
- **SEQ: A == B**
- **SLT: A < B**
- **SLTU: $unsigned(A)< $unsigned(B)**
- **XOR: A ^ B**
- **OR: A | B**
- **AND: A & B**

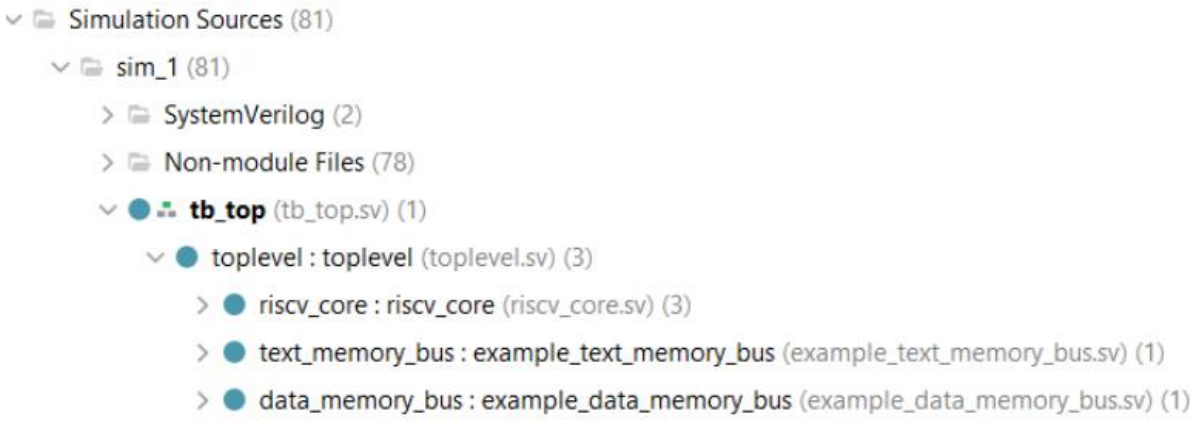
System Verilog dilinde Aritmetic Logic Unit tasarımı aşağıdaki fotoğrafta görüldüğü gibi tasarlınmıştır, ALU'nun desteklediği operasyonlar tasarım yapılırken karışıklık olmaması için enumeration tanımı yapılarak karışıklık oluşması engellenmiştir böylece Daha okunaklı bir kod yazımı sağlanmıştır.

```
module alu (
   input       [4:0]  alu_function,
   input signed [31:0] operand_a,
   input signed [31:0] operand_b,
   output logic [31:0] result,
   output              result_equal_zero
);
enum { ADD = 1 , SUB = 2 , SLL = 3 , SRL = 4 , SRA = 5 , SEQ = 6 , SLT = 7 , SLTU = 8
, XOR = 9 , OR = 10 , AND = 11 } INST_CODES;
   logic result_equal_zero_;
   assign result_equal_zero = result_equal_zero_;
   always_comb begin

      if (alu_function== ADD) begin
         result = operand_a + operand_b;
         if (result==0) begin
            result_equal_zero_=1;
         end else begin
            result_equal_zero_=0;
         end
      end else if(alu_function== SUB)begin
         result = operand_a - operand_b;
         if (result==0) begin
            result_equal_zero_=1;
         end else begin
            result_equal_zero_=0;
         end
      end else if(alu_function== SLL)begin
         result = operand_a << operand_b;
         if (result==0) begin
            result_equal_zero_=1;
         end else begin
            result_equal_zero_=0;
         end
      end else if(alu_function== SRL)begin //SRL
         result = operand_a >> operand_b;
         if (result==0) begin
            result_equal_zero_=1;
         end else begin
            result_equal_zero_=0;
         end

      end else if(alu_function== SRA)begin //SRA
         result = operand_a >>> operand_b;
         if (result==0) begin
```

```
            result_equal_zero_=1;
        end else begin
            result_equal_zero_=0;
        end

    end else if(alu_function== SEQ )begin //SEQ
        result = (operand_a == operand_b);
        if (result==0) begin
            result_equal_zero_=1;
        end else begin
            result_equal_zero_=0;
        end

    end else if(alu_function== SLT )begin //SLT
        result = operand_a < operand_b;
        if (result==0) begin
            result_equal_zero_=1;
        end else begin
            result_equal_zero_=0;
        end

    end else if(alu_function== SLTU)begin //SLTU
        result =  $unsigned(operand_a) < $unsigned(operand_b) ;
        if (result==0) begin
            result_equal_zero_=1;
        end else begin
            result_equal_zero_=0;
        end

    end else if(alu_function== XOR)begin //XOR
    result = operand_a ^ operand_b;
        if (result==0) begin
            result_equal_zero_=1;
        end else begin
            result_equal_zero_=0;
        end

    end else if(alu_function== OR)begin //OR
    result = operand_a | operand_b;
        if (result==0) begin
            result_equal_zero_=1;
        end else begin
            result_equal_zero_=0;
        end

    end else if(alu_function== AND)begin //AND
    result = operand_a & operand_b;
        if (result==0) begin
            result_equal_zero_=1;
        end else begin
             result_equal_zero_=0;
        end
    end
end
endmodul
```

---- 3. KULLANILAN YAZILIMLAR

- **3.1.1 Test Yazılım**
- **Test yazılımı işlemcininde içinde bulunduğu risc-v_core modülünü içermektedir.**
- **İşlemciyi test edecek yazılımlar Simulation Sources dosyasının içerisinde olup tb_top modülünde bulunmaktadır.**



3.1.2 Test Yazılımı Sonucu

```
PC: 00400210, Inst: df428293, Addr: 80000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxxxxxx, WrEn: 0, Wr-BE: 0001
PC: 00400214, Inst: 00028103, Addr: 80000000, Rd-Dt: 0ff000ff, Rd-En 1, Wr-Dt: 00000000, WrEn: 0, Wr-BE: 0001
PC: 00400218, Inst: 00200113, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: ffff0000, WrEn: 0, Wr-BE: 0100
PC: 0040021c, Inst: 00200e93, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00020000, WrEn: 0, Wr-BE: 0100
PC: 00400220, Inst: 01200e13, Addr: 00000012, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxx0000, WrEn: 0, Wr-BE: 0100
PC: 00400224, Inst: 03d11463, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000200, WrEn: 0, Wr-BE: 0110
PC: 00400228, Inst: 7fc00297, Addr: 80000228, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000012, WrEn: 0, Wr-BE: 0001
PC: 0040022c, Inst: dd828293, Addr: 80000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxxxxxx, WrEn: 0, Wr-BE: 0001
PC: 00400230, Inst: 00028103, Addr: 80000000, Rd-Dt: 0ff000ff, Rd-En 1, Wr-Dt: 00000000, WrEn: 0, Wr-BE: 0001
PC: 00400234, Inst: 00000013, Addr: 00000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000000, WrEn: 0, Wr-BE: 0001
PC: 00400238, Inst: 00200113, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: ffff0000, WrEn: 0, Wr-BE: 0100
PC: 0040023c, Inst: 00200e93, Addr: 00000002, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00020000, WrEn: 0, Wr-BE: 0100
PC: 00400240, Inst: 01300e13, Addr: 00000013, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xx000000, WrEn: 0, Wr-BE: 1000
PC: 00400244, Inst: 01d11463, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000200, WrEn: 0, Wr-BE: 0110
PC: 00400248, Inst: 01c01a63, Addr: 00000000, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000013, WrEn: 0, Wr-BE: 0011
PC: 0040025c, Inst: ff000513, Addr: fffffff0, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: xxxxxxxx, WrEn: 0, Wr-BE: 0001
PC: 00400260, Inst: 00100593, Addr: 00000001, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000000, WrEn: 0, Wr-BE: 0010
PC: 00400264, Inst: 00b52023, Addr: fffffff0, Rd-Dt: xxxxxxxx, Rd-En 0, Wr-Dt: 00000001, WrEn: 1, Wr-BE: 1111
Pass
$finish called at time : 2165 ns : File "C:/Users/recep/OneDrive/Masaüstü/PC proje/riscvVivado.srcs/sim_1/new/tb_top.sv" Line 31
```

- **Projenin doğru çalıştığının test edilmesi için başlangıç test kodları ile test edilmektedir.**
- **Test kodunda daha önceden hazırlanmış olan bir CPU test uygulamasının makine diline döndürülmüş halini, işlemciye besleyip sonucunu kontrol eden bir uygulama bulunmaktadır.**
- **Tasarım simülasyonu başlatılıp play tuşuna basıldığında, en fazla 10000 cycle bekleyip, sonuç hesaplanmış ise aşağıdaki şekilde görülebilen pass çıktısını vermektedir. Aksi takdirde fail çıktısı verecek.**

----4. SONUÇLAR

- **Basit bir işlemcideki RAM, Kontrol Ünitesi ve Saklayıcıların bir arada çalışıp, makine dilindeki kod parçacıklarını nasıl yürütebildiği gözlemledik.**
- **Geliştirilen Risc-v işlemcisi gerekli durum koşullarını sağladığında 11 adet komutu yerine getirdiğini gözlemledik.**
- **Xilinx Vivado Design Suite tasarım aracını kullanarak bir işlemcinin nasıl dananım tasarımı yapıldığını öğrenmiş olduk.**
- **Bellek ve işlemci arasındaki veri alışverişinde neden ihtiyaç duyulduğunu öğrenmiş olduk.**
- **Bir işlemcinin temelinde Ram'e yazıp , okuyan bir yapı olduğu ve ISA'da belirtilen komutları yerine getirdiğini öğrenmiş olduk.**

----5.PROJE EKİBİ

RECEP GEMALMAZ (200301501)
16.100.2000 tarihinde Kadıköy'de dünyaya geldim. Özel Sınav lisesinden mezun oldum. Şu anda Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde lisans eğitimimi tamamlamaktayım.
Alp Yılmaz(190301017)
Celal Aras Anadolu Lisesinden mezun olup eğitim hayatıma Fenerbahçe Üniversitesinde devam etmekteyim.

---6.REFERANS DOSYALAR

- **Youtube Link:**
- **https://www.youtube.com/watch?v=5uValY-HsZg**
- **Githup Link:**