BİLGİSAYAR MİMARİSİ

Tek Çevrimli (Single Cycle) MIPS Mimarisinin

VHDL İle Gerçeklenmesi

Proje Raporu

Hazırlayanlar

1306150077 – Kerem CANLI

1306150102 – Adem TÜRKOĞLU

1306150016 – Emre ERİNÇ

1306150056 – Muhammed Kemal PATIR

1306150062 – Fırat ÖNDER

Single – Cycle İşlemci

Single-cycle işlemci, komut setindeki her komutu tek bir clock cycle da tamamlama kapasitesine sahiptir. Bir işlemi bir clock cycle da tamamlamak demek, RF(register file), IR(instruction register),RAM ve ROM gibi büyük yapıların hepsini aynı clock cycle da kullanmak anlamına gelir. Her bir komut eşit zamanda tamamlanır.

Veri Yolu Elemanları

Program Counter

Bu blok genel anlamıyla 32-bitlik bir D-flip-floptur. Clock'un her yükselen kenarında çalıştırılacak olan bir sonraki komutun instruction memory deki adresini çıktı olarak verir. Önünde bulunan mutiplexerlar ise bir sonraki PC değerinin, PC+4 mü, branch adresi mi ya da jump adresi mi olacağını seçmek için kullanılmıştır.

```
entity PC is
    Port (
        clk : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        din : in  STD_LOGIC_VECTOR (31 downto 0);
        dout : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end PC;

architecture Behavioral of PC is

    begin
        process(clk,reset)
            begin
                if(reset='1') then
                    dout <= (others=>'0');
                elsif(clk'event and clk='1') then
                    dout<=din;
                end if;
        end process;

end Behavioral;
```

Program Counter VHDL Kodu

Instruction Memory

Bu blok programımızı yazdıktan sonra komutlarımızın kaydedildiği kısımdır. PC'den aldığı adreste ki komutu belirli bir kapı gecikmesi ile çıktı olarak verir, clock ile bağlantısı yoktur.

```
entity Instruction_Memory is
    Port (
        dir : in  STD_LOGIC_VECTOR (31 downto 0);
        instr : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end Instruction_Memory;

architecture Behavioral of Instruction_Memory is
    type mem is array(0 to 67) of std_logic_vector(7 downto 0);
    constant code : mem:=(

        -- Load here your software.



    --others=> x"00"
    );

    begin

        process(dir)
        begin
            instr(31 downto 24)<=code(conv_integer(dir));
            instr(23 downto 16)<=code(conv_integer(dir)+1);
            instr(15 downto 8)<= code(conv_integer(dir)+2);
            instr(7 downto 0)<=code(conv_integer(dir)+3);
        end process;

end Behavioral;
```

Instruction Memory VHDL Kodu

Register File

İçinde 32 tane 32-bitlik register bulunan bloktur. A1 ve A2, hangi registerların okunacağını belirler, ve bu registerların içindeki veriler RD1 ve RD2 den çıktı olarak alınır. Sadece registerlara yazma işlemi sequentialdır. we3 sinyali '1' olduğunda, A3'ün karşılık geldiği registera WD3'te ki 32-bitlik veriyi clockun yükselen kenarında yazar.

```
entity Register_File is
    Port (
        clk : in  STD_LOGIC;
        we3 : in  STD_LOGIC;
        A1 : in  STD_LOGIC_VECTOR (4 downto 0);
        A2 : in  STD_LOGIC_VECTOR (4 downto 0);
        A3 : in  STD_LOGIC_VECTOR (4 downto 0);
        RD1 : out  STD_LOGIC_VECTOR (31 downto 0);
        RD2 : out  STD_LOGIC_VECTOR (31 downto 0);
        WD3 : in  STD_LOGIC_VECTOR (31 downto 0)
        );
end Register_File;

architecture Behavioral of Register_File is

    type ram_type is array(0 to 31) of std_logic_vector(31 downto 0);
    signal ram : ram_type;
    begin

        process(clk)
            begin
                if(clk'event and clk='1') then
                    if (we3='1') then
                        ram(conv_integer(A3))<=WD3;
                    end if;
                end if;
        end process;

        process(a1,a2)
            begin
                if( conv_integer(A1)=0) then
                    rd1<=x"00000000";
                else RD1<=ram(conv_integer(A1));
                end if;
                if(conv_integer(A2)=0) then
                    rd2<=x"00000000";
                else RD2<=ram(conv_integer(A2));
                end if;
        end process;

end Behavioral;
```

Register File VHDL Kodu

ALU

ALU, func sinyaline göre işlemleri gerçekleştirir.

ALU da yapılan işlemler sonucunda eğer sıfır elde edilirse, “zero” sinyali ‘1’ olur, diğer bütün durumlarda ‘0’ olarak kalır.

Veri yolundaki ALU'nun b girdisi önündeki bulunan multiplexer, b 'nin direk olarak register fileda ki RD2 mi, yoksa 32-bit e genişletilmiş immediate değer mi olacağını Alusrc sinyaline göre seçer. Veri yolunda görmüş olduğunuz AluSrc,RegDest, MemtoReg gibi sinyallerin durumu kontrol bloğu tarafından belirlenir.

```
entity ALU is
    Port (
        a : in  STD_LOGIC_VECTOR (31 downto 0);
        b : in  STD_LOGIC_VECTOR (31 downto 0);
        func : in  STD_LOGIC_VECTOR (2 downto 0);
        zero: out std_logic ;
        rslt : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end ALU;

architecture Behavioral of ALU is
    COMPONENT zero_extend
    PORT(
        inic : IN std_logic;
        extend : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    COMPONENT Mux_2to1_32b
    PORT(
        ctrl : IN std_logic;
        A : IN std_logic_vector(31 downto 0);
        B : IN std_logic_vector(31 downto 0);
        O : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    COMPONENT alu_sum_res
    PORT(
        ctrl : IN std_logic;
        a : IN std_logic_vector(31 downto 0);
        b : IN std_logic_vector(31 downto 0);
        sol : OUT std_logic_vector(31 downto 0);
        cout : OUT std_logic
        );
    END COMPONENT;

    COMPONENT mux_4_32b
    PORT(
        ctrl : IN std_logic_vector(1 downto 0);
        a : IN std_logic_vector(31 downto 0);
        b : IN std_logic_vector(31 downto 0);
        c : IN std_logic_vector(31 downto 0);
        d : IN std_logic_vector(31 downto 0);
        sal : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;


    signal  rslt_and,rslt_or,rslt_and_compl,rslt_mux_alu,
```

```
        rslt_or_compl,rslt_slt,bb,b_compl,rslt_sum_res : std_logic_vector (31 downto 0):=x"00000000";
    begin

        b_compl<=not(b);

        mux_b_b_compl: Mux_2to1_32b PORT MAP(
            ctrl => func(2),
            A => b,
            B => b_compl,
            O => bb
        );

        Inst_alu_sum_res: alu_sum_res PORT MAP(
            ctrl => func(2),
            a => a,
            b => bb,
            sol => rslt_sum_res
        );

        rslt_and<=a and bb;
        rslt_or<=a or bb;

        Inst_zero_extend: zero_extend PORT MAP(
            inic => rslt_sum_res(31),
            extend => rslt_slt
        );

        mux_rslt_alu: mux_4_32b PORT MAP(
            ctrl => func(1 downto 0),
            a => rslt_and,
            b => rslt_or,
            c => rslt_sum_res,
            d => rslt_slt,
            sal => rslt_mux_alu
        );


        zero<=  '1' when (rslt_mux_alu=x"00000000") else
                '0';
        rslt<=rslt_mux_alu;

end Behavioral;
```

ALU VHDL Kodu

Data Memory

Verilerimizi saklamak için kullandığımız bellek olarak düşünebilirsiniz. Bu blokta okuma işlemi combinational olarak çalışırken, yazma işlemi sequential olarak çalışmaktadır. MemWrite sinyali ‘1’ olduğunda ve clock un yükselen kenarında, ALU tarafından hesaplanmış adrese register file ın RD2 çıkışındaki değeri yazar. Eğer MemWrite ‘0’ ise read_data outputun da ALU tarafndan hesaplanmış adresin içindeki veriyi çıktı olarak verir. Veri yolunda data memory den sonraki multiplexer, register file da ki registera yazılacak 32-bitlik verinin data memory(lw komutu) den gelen veri mi, yoksa ALU’nun hesaplamış olduğu veri mi olduğunu seçer.

```
entity Datamemory is
    port (
        address: in STD_LOGIC_VECTOR (31 downto 0);
        write_data: in STD_LOGIC_VECTOR (31 downto 0);
        MemWrite: in STD_LOGIC;
        clk: in STD_LOGIC;
        read_data: out STD_LOGIC_VECTOR (31 downto 0)
    );
end Datamemory;


architecture behavioral of Datamemory is

    type mem_array is array(0 to 31) of STD_LOGIC_VECTOR (31 downto 0);

    signal data_mem: mem_array := (
        X"00000000", -- initialize data memory
        X"00000000", -- mem 1
        X"00000000",
        X"00000033",
         X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000", -- mem 10
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000", -- mem 20
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000",
        X"00000000", -- mem 30
        X"00000000");

    begin
```

```
    mem_process: process(address, write_data,clk)
        begin
            if clk = '0' and clk'event then
                read_data <= data_mem(conv_integer(address(6 downto 2)));
                if (MemWrite = '1') then
                    data_mem(conv_integer(address(6 downto 2))) <= write_data;
                end if;
            end if;
    end process mem_process;

end behavioral;
```

Data Memory VHDL Kodu

Sign Extend Unit

I-tip komutlardaki 16-bitlik immediate değerini, işaretlerine göre 32-bite genişleten bloktur.

```
entity SignExtend is
    Port (
        din : in  STD_LOGIC_VECTOR (15 downto 0);
        dout : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end SignExtend;

architecture Behavioral of SignExtend is
    --signal a : std_logic;
    begin
        dout<= x"0000"&din when (din(15)='0') else  x"ffff"&din;

end Behavioral;
```

Sign Extend VHDL Kodu

SL2

32-bite genişletilmiş immediate değerini 2-bit sola kaydırır.

```
entity SL2 is
    Port (
        din : in  STD_LOGIC_VECTOR (31 downto 0);
        dout : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end SL2;

architecture Behavioral of SL2 is

    begin
        dout<=din(29 downto 0)&"00"; --Multiply with 4

end Behavioral;
```

SL2 VHDL Kodu

Diğer Bazı Elemanlar

Contol Unit

Instruction memory'den aldığı instruction'ın opcode'una ve function code'una göre gerekli elemanlara gerekli çıktıları sağlar.

```
entity ControlUnit is
    Port (
        OpCode : in  STD_LOGIC_VECTOR (5 downto 0);
        Funct : in  STD_LOGIC_VECTOR (5 downto 0);
        MemtoReg : out  STD_LOGIC;
        MemWrite : out  STD_LOGIC;
        Branch : out  STD_LOGIC;
        AluSrc : out  STD_LOGIC;
        RegDst : out  STD_LOGIC;
        RegWrite : out  STD_LOGIC;
        jump : out std_logic;
        AluCtrl : out  STD_LOGIC_VECTOR (2 downto 0)
        );
end ControlUnit;

architecture Behavioral of ControlUnit is
    COMPONENT MainDecoder
    PORT(
        opcode : IN std_logic_vector(5 downto 0);
        RegWrite : OUT std_logic;
        RegDst : OUT std_logic;
        ALUSrc : OUT std_logic;
        Branch : OUT std_logic;
        MemWrite : OUT std_logic;
        MemtoReg : OUT std_logic;
        ALUOp : OUT std_logic_vector(1 downto 0);
        Jump : OUT std_logic
        );
    END COMPONENT;

    COMPONENT ALUdecoder
    PORT(
        ALUop : IN std_logic_vector(1 downto 0);
        funct : IN std_logic_vector(5 downto 0);
        ALUctrl : OUT std_logic_vector(2 downto 0)
        );
    END COMPONENT;

    signal opalu:std_logic_vector(1 downto 0);

    begin

    Inst_MainDecoder: MainDecoder PORT MAP(
            opcode => opcode,
            RegWrite => regwrite,
            RegDst => regdst,
            ALUSrc => alusrc ,
            Branch => branch,
            MemWrite => memwrite,
            MemtoReg => memtoreg,
```

```
            ALUOp => opalu,
            Jump => jump
        );

    Inst_ALUdecoder: ALUdecoder PORT MAP(
            ALUop => opalu,
            funct => funct,
            ALUctrl => aluctrl
        );

end Behavioral;
```

Control Unit VHDL Kodu

Multiplexer'lar

Girişine gelen verilerinden birini veya birkaçını kontrol sinyallerine göre çıkışa aktaran elemanlardır.

```
entity Mux_2to1_32b is
    Port (
        ctrl : in  STD_LOGIC;
        A : in  STD_LOGIC_VECTOR (31 downto 0);
        B : in  STD_LOGIC_VECTOR (31 downto 0);
        O : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end Mux_2to1_32b;

architecture Behavioral of Mux_2to1_32b is

    begin
        o <=    a   when ctrl='0' else b;

end Behavioral;
```

2 To 1 32 Bit Mux VHDL Kodu

```
entity mux_4_32b is
    Port (
        ctrl : in  STD_LOGIC_VECTOR (1 downto 0);
        a : in  STD_LOGIC_VECTOR (31 downto 0);
        b : in  STD_LOGIC_VECTOR (31 downto 0);
        c : in  STD_LOGIC_VECTOR (31 downto 0);
        d : in  STD_LOGIC_VECTOR (31 downto 0);
        sal : out  STD_LOGIC_VECTOR (31 downto 0)
        );
end mux_4_32b;

architecture Behavioral of mux_4_32b is

    begin
        sal<= a when (ctrl="00") else
              b when (ctrl="01") else
              c when (ctrl="10") else
              d;

end Behavioral;
```

4 To 1 32 Bit Mux VHDL Kodu

MIPS DataPath

Yukarıda verilen elemanları kullanarak MIPS DataPath'ini oluşturuyoruz ve simüle ederek istenen sonuçları alıyoruz.

```
entity MIPS is
    Port (
        clk : in  STD_LOGIC;
        reset : in std_logic;
        address : out  STD_LOGIC_VECTOR (31 downto 0)
         );
end MIPS;

architecture Behavioral of MIPS is

    COMPONENT PC
    PORT(
        clk : IN std_logic;
        reset : IN std_logic;
        din : IN std_logic_vector(31 downto 0);
        dout : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;



    COMPONENT Instruction_Memory
    PORT(
        dir : IN std_logic_vector(31 downto 0);
        instr : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    COMPONENT Alu_PCnext
    PORT(
        PC : IN std_logic_vector(31 downto 0);
        PCnext : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    COMPONENT ControlUnit
    PORT(
        OpCode : IN std_logic_vector(5 downto 0);
        Funct : IN std_logic_vector(5 downto 0);
        MemtoReg : OUT std_logic;
        MemWrite : OUT std_logic;
        Branch : OUT std_logic;
        AluSrc : OUT std_logic;
        RegDst : OUT std_logic;
        RegWrite : OUT std_logic;
        jump : OUT std_logic;
        AluCtrl : OUT std_logic_vector(2 downto 0)
        );
    END COMPONENT;
```

```
component Datamemory
port (
    address: in STD_LOGIC_VECTOR (31 downto 0);
    write_data: in STD_LOGIC_VECTOR (31 downto 0);
    MemWrite: in STD_LOGIC;
    clk: in STD_LOGIC;
    read_data: out STD_LOGIC_VECTOR (31 downto 0)
);
end component;

COMPONENT Register_File
PORT(
    clk : IN std_logic;
    we3 : IN std_logic;
    A1 : IN std_logic_vector(4 downto 0);
    A2 : IN std_logic_vector(4 downto 0);
    A3 : IN std_logic_vector(4 downto 0);
    WD3 : IN std_logic_vector(31 downto 0);
    RD1 : OUT std_logic_vector(31 downto 0);
    RD2 : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;

COMPONENT ALU
PORT(
    a : IN std_logic_vector(31 downto 0);
    b : IN std_logic_vector(31 downto 0);
    func : IN std_logic_vector(2 downto 0);
    zero : out std_logic;
    rslt : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;

COMPONENT SignExtend
PORT(
    din : IN std_logic_vector(15 downto 0);
    dout : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;

COMPONENT Mux_2to1_32b
PORT(
    ctrl : IN std_logic;
    A : IN std_logic_vector(31 downto 0);
    B : IN std_logic_vector(31 downto 0);
    O : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;

COMPONENT SL2
```

```
PORT(
    din : IN std_logic_vector(31 downto 0);
    dout : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;

COMPONENT ALU_sum
PORT(
    a : IN std_logic_vector(31 downto 0);
    b : IN std_logic_vector(31 downto 0);
    sal : OUT std_logic_vector(31 downto 0)
    );
END COMPONENT;


COMPONENT Mux_2to1_5bits
PORT(
    ctrl : IN std_logic;
    a : IN std_logic_vector(4 downto 0);
    b : IN std_logic_vector(4 downto 0);
    sal : OUT std_logic_vector(4 downto 0)
    );
END COMPONENT;

signal memtoreg,branch,alusrc,regdst,regwrite,jump,zero,memwrite : std_logic;
signal aluctrl : std_logic_vector(2 downto 0);

signal rst_pc : std_logic;
signal pc_in,pc_out,instr: std_logic_vector(31 downto 0);

alias code_op : std_logic_vector(5 downto 0) is instr(31 downto 26);
alias funct : std_logic_vector(5 downto 0) is instr(5 downto 0);

alias rs : std_logic_vector(4 downto 0) is instr(25 downto 21);
alias rt : std_logic_vector(4 downto 0) is instr(20 downto 16);
alias rd : std_logic_vector(4 downto 0) is instr(15 downto 11);
alias shamt : std_logic_vector(4 downto 0) is instr(10 downto 6);
alias inmd : std_logic_vector(15 downto 0) is instr(15 downto 0);
alias addr : std_logic_vector(25 downto 0) is instr(25 downto 0);

signal pc_out_next: std_logic_vector(31 downto 0);
signal sal_rt_o_rd : std_logic_vector(4 downto 0);

signal srca,srcb,rd2,alu_result,extsig_out,readdata : std_logic_vector(31 downto 0);
signal shift_out,pc_branch,result_mem : std_logic_vector(31 downto 0);

--signs for j

signal addr32,addr32_corri,addr32_pc_next,pc_next_j : std_logic_vector(31 downto 0);
signal pcsrc : std_logic;
```

```
begin

    Inst_ControlUnit: ControlUnit PORT MAP(
        OpCode => code_op,
        Funct => funct,
        MemtoReg => memtoreg,
        MemWrite => memwrite,
        Branch => branch,
        AluSrc => alusrc,
        RegDst => regdst,
        RegWrite => regwrite,
        jump => jump,
        AluCtrl => aluctrl
    );


    Inst_PC: PC PORT MAP(
        clk => clk,
        reset => reset,
        din => pc_next_j,
        dout => pc_out
    );

    Inst_Instruction_Memory: Instruction_Memory PORT MAP(
        dir => pc_out,
        instr =>instr
    );

    Memory: Datamemory PORT MAP(
    address => alu_result,
    write_data => rd2 ,
    MemWrite => memwrite,
    clk => clk,
    read_data => readdata

    );


    ALU_sum_4: ALU_sum PORT MAP(
        a => pc_out,
        b => x"00000004",
        sal => pc_out_next
    );



    Inst_Mux_rt_o_rd: Mux_2to1_5bits PORT MAP(
        ctrl => regdst,
        a => rt,
        b => rd,
```

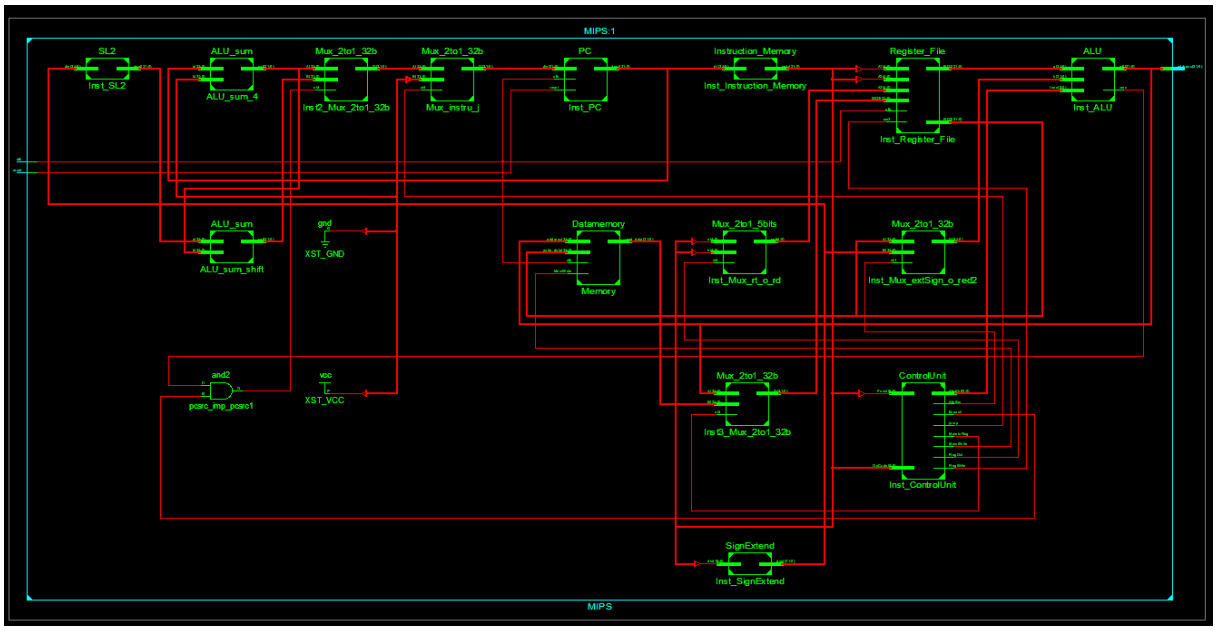
```
    sal => sal_rt_o_rd
);



Inst_Register_File: Register_File PORT MAP(
    clk => clk,
    we3 => regwrite,
    A1 => rs,
    A2 => rt,
    A3 => sal_rt_o_rd,
    RD1 => srca,
    RD2 => rd2,
    WD3 => result_mem
);

Inst_ALU: ALU PORT MAP(
    a => srca,
    b => srcb,
    func => aluctrl,
    zero=> zero,
    rslt => alu_result
);


Inst_SignExtend: SignExtend PORT MAP(
    din => inmd,
    dout => extsig_out
);

Inst_Mux_extSign_o_red2: Mux_2to1_32b PORT MAP(
    ctrl => alusrc,
    A => rd2,
    B => extsig_out,
    O => srcb
);

Inst_SL2: SL2 PORT MAP(
    din => extsig_out,
    dout => shift_out
);


ALU_sum_shift: ALU_sum PORT MAP(
    a => shift_out,
    b => pc_out_next,
    sal => pc_branch
);

pcsrc<=branch and zero;
```

```
        Inst2_Mux_2to1_32b: Mux_2to1_32b PORT MAP(
            ctrl => pcsrc,
            A => pc_out_next,
            B => pc_branch,
            O => pc_in
        );


        Inst3_Mux_2to1_32b: Mux_2to1_32b PORT MAP(
            ctrl => memtoreg,
            A => alu_result,
            B => readdata,
            O => result_mem
        );


        address<= alu_result;

        addr32_pc_next<= pc_out_next (31 downto 28) & addr &"00";

        Mux_instru_j: Mux_2to1_32b PORT MAP(
            ctrl => jump,
            A => pc_in,
            B => addr32_pc_next,
            O => pc_next_j
        );

end Behavioral;
```

MIPS Mimarisi VHDL Kodları

RTL Şema

Oluşturulan MIPS DataPath'inin Xilinx programındaki RTL Şeması



Simülasyon

Simülasyon için pc_in girişine 00000000000000000000000000000000 değerini verildi. Daha sonra clock tanımlaması yapıldı.

Leading Edge Value: 0

Trailing Edge Value: 1

Starting at Time Offset: 0

Cancel after Time Offset: <blank>

Duty Cycle (%): 50

Period: 1 us

Clock için bu değerler verildi.

Simülasyon çalıştırılmadan önce data_memory nin 3. kısmına 00000033 değeri verildi. Ayrıca çalıştırılması istenilen komutlardan önce $4 $5 $6 $7 registerlarına sırasıyla 4 5 6 7 değerleri yükleyen komutlar yazıldı. Peşine çalıştırılması istenilen komutlar eklendi. Çalıştırılması istenilen komutların en sonunda bulunan jump komutuna 00 yerine 04 verildi. Çünkü 0 1 2 ve 3. satırlarda registerlara yükleme yapan komutlar var. Daha sonra simülasyon çalıştırıldı.

Simülasyon çıktısı büyük olduğu için diğer sayfaya yan olarak eklendi ve proje ekleri içinde gönderildi.