



Universiteti i Prishtinës

Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike
Arkitektura e Kompjuterëve, 2023/2024
Prof. Valon Raça & Synim Selimi

Arbnora Dragaj, 220756100051
Rinesa Bejic, 22075610009
Rinesa Gajtani, 220756100050
Shqiponje Dani, 220756100073
Yllkë Berisha, 220756100100

Detyra e dytë

HYRJE

Përmes kësaj detyre ne po e krijojmë një CPU (procesor) 16-bitësh, e cila implementohet përmes Single-Cycle Datapath me anë të Verilog-ut. Krijimi i CPU 16-bitësh në parim është i ngjashëm me atë të CPU-së 32-bitëshe, por me disa modifikime (tek regjistrat e CPU-së 16-bitëshe, instruksionet, adresa, etj., janë 16-bitëshe). Për të ndërtuar CPU-në 16-bitëshe, fillimisht ishte e detyrueshme të krijojmë pjesët përbërëse të saj: ALU, ALU Control, Control Unit, Register File, Data Memory dhe Instruction Memory. Nëse lidhim ndermjet këtyre elementeve, formojmë Datapath-in, Control Unit-in, dhe fetch-in, këto tre pjesë krijojnë CPU-në.

ALU (16-bitëshe) – Për çfarë na shërben ALU 16-bitëshe? ALU 16-bitëshe na shërben për me i realizu të gjitha llogaritjet aritmetike.

ALU Control - Na specifikon cilin operacion me kry ALU-ja.

Control Unit - Të gjitha fushat e saj përpos ALUop e cila është 2 bit (që shërben për specifikimin e operacionit që do kryhet në ALU), pjesa tjetër ka vetëm një bit, ngase secila prej tyre është për të treguar vërtetësi psh:

- MemRead për vlerën 1 lejon të lexojmë në memorie,
- MemWrite të shkruajmë në memorie,
- Branch për kërcime me beq,
- MemtoReg i cili shërben si bit selektues tek multiplekseri 2 me 1 i cili zgjedh se a kemi të bëjme me lw apo me ndonjë operacion tjetër,
- RegWrite për vlerën 1 lejon me shkrujt në regjistër ,
- ALUsrc i cili është bit selektues tek multiplekseri 2 me 1 i cili shërben për të marrë vlerën imediate apo vlerën e read data 2, varësisht nga lloji i instruksionit i apo R.

Register File - qe do te thote secili regjistër mund të lexohet ose shkruhet duke e specifikuar numrin e atij regjistri.

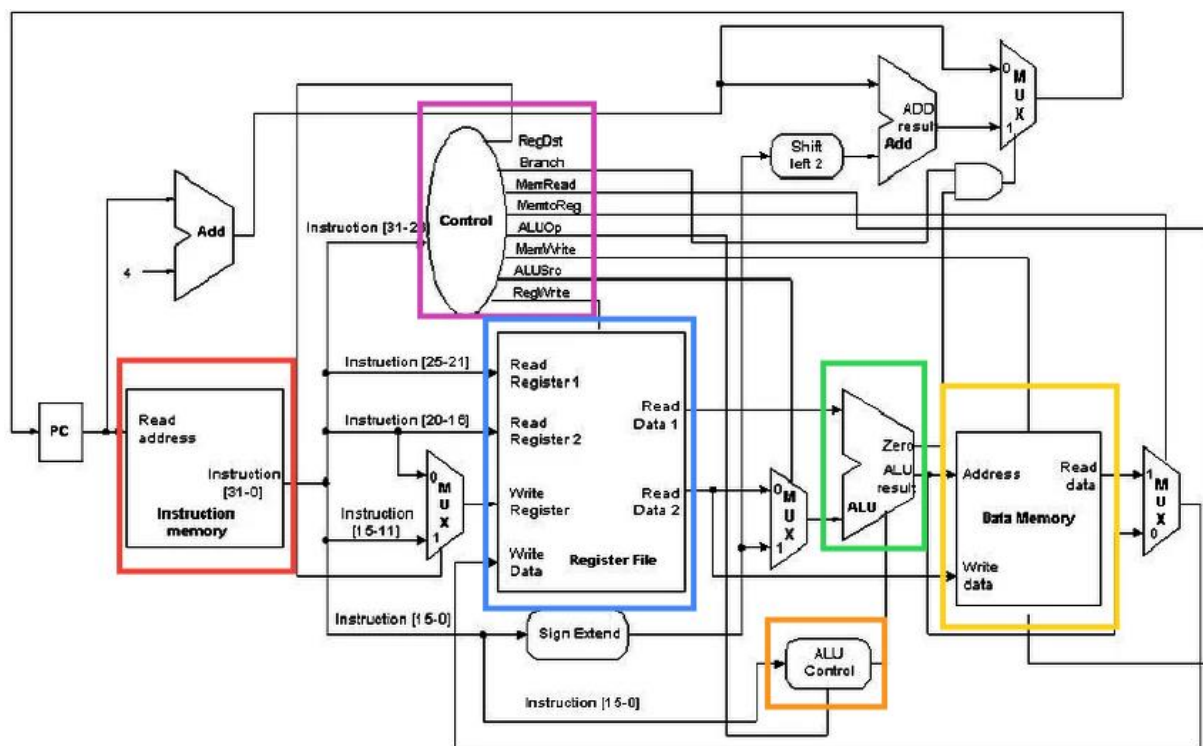
Instruction Memory – është një njësi memorike për të ruajtur udhëzimet e një programi dhe udhëzimet e furnizimit me të dhëna nga një adresë.

Data Memory – na mundeson me lexu dhe me shkrujt të dhëna ne memorie.

Multiplekseri – selektues (perzgjedhes).

Datapath - është hardueri që i kryen të gjithë operacionet e kërkuara.

Pjeset per dizajnimin e CPU-se



ALU 16 – bitëshe: Me anë të metodës Ripple-carry e krijojmë ALU 16-biteshe duke i shtuar 16 ALU-ja 1 biteshe. Kjo ALU është përgjegjëse për kryerjen e operacioneve si AND-i, OR-i, XOR-i, ADD-i, ADDI, SUB etj. Meqe është 16 – biteshe përkrah numra me gjatësi 16 bitëshe. COUT i njëjës ALU është hyrje për ALU-në e ardhshme. Brenda kësaj ALU-je 1 bitëshe kemi një multiplekser mux2ne1 i cili na nevojitet për të përcaktuar se a na nevojitet B apo JoB, një

multiplekser mux8ne1 qe është përgjegjës për zgjedhjen e operacioneve(pra njërit nga operacioneve lartë) si dhe nje mbledhes.

ALU_with_extra_operation: Brenda kësaj ALU-je me operacione eksta janë këto operacion ALU16_Result, SLTI_Result, SLL_Result, SRA_Result ku secila nga këto është 16 bitëshe dhe si rezultat final e kemi marrë nga mux4ne1.

Mux4ne1 – përmban modulën mux4to1() si dhe ka hyrjet: hyrja0, hyrja1,hyrja2 dhe hyrja3, si dhe bitin perzgjedhes ALUOp 4 bitesh, ndersa si dalje: (dalja).

Mux2ne1 – përmban modulën mux2ne1() si dhe ka hyrjet: hyrja0, hyrja1, si dhe bitin perzgjedhes S ndersa si dalje: (dalja).

Mux8ne1 – përmban modulën mux8ne1(). Ka hyrjet: input0, input1, input2, input3, input4, input5, bitin perzgjedhes S si dhe daljen: dalja. Jane multiplekserat qe na ndihmojne për të zgjedhur operacionet e kerkuara.

ALU Control – Ne rastin tone alu control përmban hyrjet ALUOp dhe Function 2 biteshe , si dhe opcode 4 biteshe, ndersa si dalje ka ALUCtrl 4 biteshe. Ne ALU Control permes ALUOp përcaktojme se cfare operacione dëshirojmë të kryejmë. Pra ALU Control I specifikon ALU - së 16 biteshe bitat të cilët tregojnë veprimin që do te kryhet në ALU.

Control Unit - është pjesë përgjegjëse për të gjitha veprimet brenda CPU-së . Opcode deklarohet si hyrje, ndersa RegDst, MemToReg, MemWrite, Branch, MemRead, RegWrite, ALUOp , ALUSrc si dalje. ALUOp eshte 2 bitesh dhe sherben per me u informu se cili operacion do te kryhet. Nese MemRead ka vlerën 1 na lejon të lexojmë në memorie, nese MemWrite ka vleren 1 na lejon të shkruajmë në memorie, nese RegWrite ka vlerën 1 lejon qe te shkruajme në regjistër ,nese Branch ka vleren 1 na lejon kërcime me beq. MemtoReg na shërben si bit selektues tek multiplekseri mux2ne1 i cili zgjedh se a kemi të bëjme me lw apo me ndonjë operacion tjetër të formatit R si dhe ALUSrc i cili është bit selektues tek multiplekseri qe na shërben me marrë vlerën imediate apo vlerën e read data 2, varësisht nga lloji i instruksionit i apo r.

DataMemory - Data Memory është Read-Write (mundeson shkrim dhe lexim). Ka hyrjet Adresa dhe WriteData 16 biteshe, ka MemWrite, MemRead dhe Clock 1 biteshe ndersa si dalje ka ReadData 16 biteshe.

Me anë të \$readmemh për të inicializuar memorien nga fajlli dataMem.mem që përmban vlera hex të gjitha zero që tregojnë se është e shprazur.

Data Memory është e ngjajshme me instruction memory, përpos se këtu mundemi me shkrujt të dhëna. Shërben për instruksionet e formatit I. Dy veprimet që lidhen me data memory janë sw dhe lw.

InstructionMemory - Instruction Memory është vetëm Read-Only (vetëm për Instruksione). Ka një hyrje 16 biteshe PC Adresen si dhe një dalje 16 biteshe Instruksionin. InstructionMemory është në heksadecimal dhe është 128bajt, ku dhjetë bajtat e parë janë të rezervuar kurse nga bajti i 10 kemi shkruar instruksionet për ekzekutim nga memoria. E përdorim \$readmemh për të inicializuar memorien nga fajlli instrMem.mem që përmban vlera hex.

RegisterFile - Ka hyrjet RS,RT,RD 2 biteshe (për përcaktimin e regjistrave), ka WriteData 16 biteshe, si dhe RegWrite, Clock 1 biteshe. Register file ka dalja: ReadRS dhe ReadRD 16 biteshe. Numri i regjistrave do të jetë 4 (regjistri \$zero dhe 3 të tjerë për përdorim të përgjithshëm). Regjistrat do të jenë të gjerë 16 bit. Për të lexuar nga file InstructionMemory.mem përdorim \$readmemh (h pasi që file është në hex).

DataPath – Krijimi i një DataPath përfshin shumë pjesë të një procesori (CPU). Nëse e analizojmë dekodimin e instruksioneve atëherë opcode, rs, rt, rd e function dhe immediate janë pjesë të instruksioneve ku opcode na tregon se cili operacion ka me u kry. Në datapath marrin pjesë edhe multiplekseret që përdoren për të zgjedhur ndër dy opsione bazuar në një sinjal kontrolli (në këtë rast, **RegDst**, **ALUSrc**, **MemToReg**, **MemWrite**, etj.). Të gjitha që I kemi sqaruar më lartë (DataMemory, InstructionMemory, RegisterFile etj.) marrin pjesë në krijimin e DataPath. Pjesa fundit e kodit ka të bëjë me krahasimin dhe drejtimin e adresave në rastin e një operacioni BEQ. Përdoret një mbledhës dhe një MUX për të përcaktuar adresën e instruksionit të ardhshëm bazuar në kushtet e BEQ (Branch Equal).

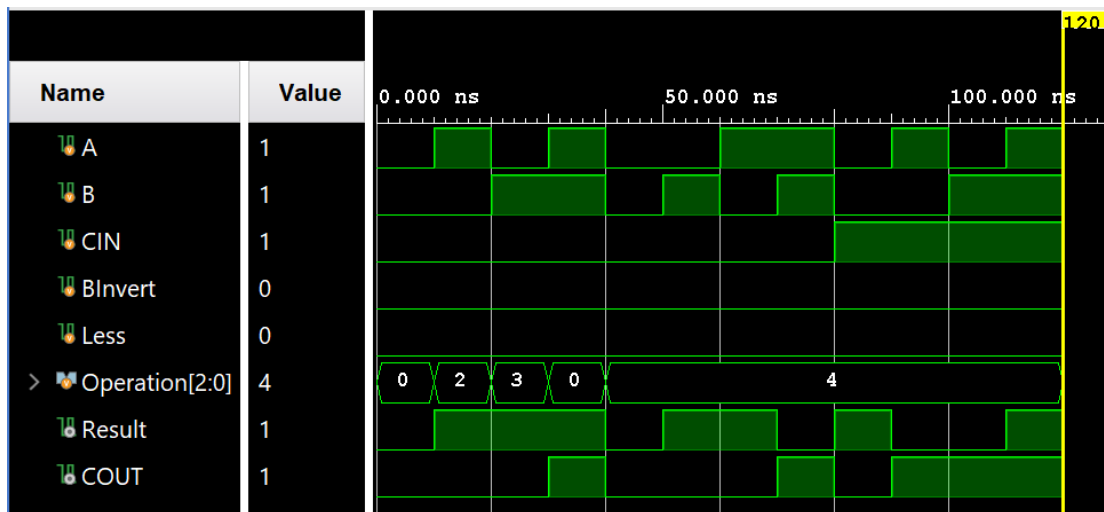
Si kërkesë e pjesës së detyrës bonus po ashtu kemi realizuar Instruksionet

- **SLTI (Set Less Than Immediate)**, **SLL (Shift Left Logic)** si dhe **SRA (Shift Right Arithmetic)**
 - këto instruksione janë krijuar nga jashtë dhe janë thirr brenda ALU_with_extra_operation.
- 1. **SLTI (Set Less Than Immediate):** Komanda SLTI e instruktore procesorin të vendosë vlerën e një regjistri në 1 nëse vlera e një regjistri tjetër është më e vogël ose e barabartë me një vlerë të caktuar (konstanta e përcaktuar nga Immediate), ndryshe vendos vlerën në 0.
- 2. **SLL (Shift Left Logic):** SLL realizohet duke bërë shtyrje logjike majtas për aq sa parashihet në shamt. Shtyrja logjike majtas bëhet duke e shtyr majtas dhe në vendet e zbrazta të mbetura e mbushim me zero.
- 3. **SRA (Shift Right Arithmetic):** SRA realizohet duke bërë shtyrje aritmetike djathtas për aq sa parashihet në shamt. Shtyrja aritmetike djathtas bëhet duke e shtyr djathtas dhe në vendet e zbrazta të mbetura e mbushim sa ka qenë most significant bit.

Testimi

ALU1 bitëshe Test

```
23 module ALU1_Test();
24
25 reg A, B, CIN, BInvert, Less;
26 reg [2:0] Operation;
27 wire Result, COUT;
28
29 initial
30 $monitor("A=%b, B=%b, CIN=%b, BInvert=%b, Less=%b, Operation=%b, Result=%b, COUT=%b",
31 A, B, CIN, BInvert, Less, Operation, Result, COUT);
32
33 initial
34 begin
35
36 #0 A=1'b0;B=1'b0; CIN=1'b0; BInvert=1'b0; Less =1'b0 ;Operation=3'b000; // and
37 #10 A=1'b1;B=1'b0; CIN=1'b0; BInvert=1'b0; Less =1'b0 ;Operation=3'b010; // or
38 #10 A=1'b0;B=1'b1; CIN=1'b0; BInvert=1'b0; Less =1'b0; Operation=3'b011; // xor
39 #10 A=1'b1;B=1'b1; CIN=1'b0; BInvert=1'b0; Less =1'b0;Operation=3'b000; //and
40
41
42 //ADD
43 #10 A=1'b0;B=1'b0; CIN=1'b0; BInvert=1'b0; Less =1'b0; Operation=3'b100;
44 #10 A=1'b0;B=1'b1; CIN=1'b0; BInvert=1'b0; Less =1'b0; Operation=3'b100;
45 #10 A=1'b1;B=1'b0; CIN=1'b0; BInvert=1'b0; Less =1'b0; Operation=3'b100;
46 #10 A=1'b1;B=1'b1; CIN=1'b0; BInvert=1'b0; Less =1'b0; Operation=3'b100;
47
48 #10 A=1'b0;B=1'b0; CIN=1'b1; BInvert=1'b0; Less =1'b0; Operation=3'b100;
49 #10 A=1'b1;B=1'b0; CIN=1'b1; BInvert=1'b0; Less =1'b0; Operation=3'b100;
50 #10 A=1'b0;B=1'b1; CIN=1'b1; BInvert=1'b0; Less =1'b0; Operation=3'b100;
51 #10 A=1'b1;B=1'b1; CIN=1'b1; BInvert=1'b0; Less =1'b0; Operation=3'b100;
52
53 #10 $stop;
54
55 end
56 ALU1 ALU1_Test(A,B,CIN,BInvert,Less,Operation,Result,COUT);
57 endmodule
```



```
..
A=0, B=0, CIN=0, BInvert=0, Less=0, Operation=000, Result=0, COUT=0
A=1, B=0, CIN=0, BInvert=0, Less=0, Operation=010, Result=1, COUT=0
A=0, B=1, CIN=0, BInvert=0, Less=0, Operation=011, Result=1, COUT=0
A=1, B=1, CIN=0, BInvert=0, Less=0, Operation=000, Result=1, COUT=1
A=0, B=0, CIN=0, BInvert=0, Less=0, Operation=100, Result=0, COUT=0
A=0, B=1, CIN=0, BInvert=0, Less=0, Operation=100, Result=1, COUT=0
A=1, B=0, CIN=0, BInvert=0, Less=0, Operation=100, Result=1, COUT=0
A=1, B=1, CIN=0, BInvert=0, Less=0, Operation=100, Result=0, COUT=1
A=0, B=0, CIN=1, BInvert=0, Less=0, Operation=100, Result=1, COUT=0
A=1, B=0, CIN=1, BInvert=0, Less=0, Operation=100, Result=0, COUT=1
A=0, B=1, CIN=1, BInvert=0, Less=0, Operation=100, Result=0, COUT=1
A=1, B=1, CIN=1, BInvert=0, Less=0, Operation=100, Result=1, COUT=1
```

Instruction Memory Test

```

module InstructionMemory(
input wire[15:0] PCAddress,
output wire[15:0] Instruction);

reg[7:0] instrMem[127:0]; // [7:0] 8 bit me 128 reshta

initial $readmemh("InstructionMemory.mem", instrMem);

assign Instruction = {instrMem[PCAddress] , instrMem[PCAddress + 1]};

endmodule

```

```

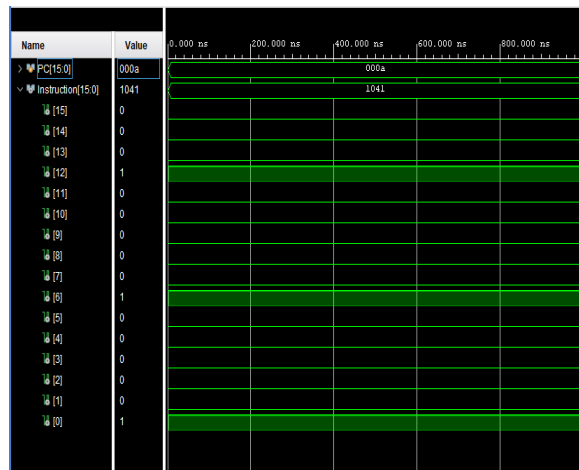
10 : 00
11 : 10 // sub $r1, $zero, $zero
12 : 41 // 0001 0000 0100 0001
13 : C6 // lw $r2, 4($r1)
14 : 04 // 1100 0110 0000 0100
15 : 0B // xor $r1, $r2, $r3
16 : 42 // 0000 1011 0100 0010
17 : F8 // beq $r2, $zero, kercimi
18 : 01 // 1111 1000 0000 0001
19 : 06 // and $r3, $r1, $r2
20 : C0 // 0000 0110 1100 0000
21 : DB // kercimi: sw $r3, 4($r2)
22 : 04 // 1101 1011 0000 0100
23 : BE // slti $r2, $r3, 2
24 : 02 // 1011 1110 0000 0010
25 : 28 // sll $r1, $r2, 3
26 : 4C // 0010 1000 0100 1100
27 : 24 // sra $r2, $r1, 9
28 : A5 // 0010 0100 1010 0101
29 : 00
30 : 00

```

```

: # run 1000ns
: PC = 0000000000001010; Instruction = 0001000001000001;
:

```



Data Memory Test

```
module DataMemory_Test();
reg Clock, MemWrite, MemRead;
reg[15:0] Adresa;
reg[15:0] WriteData;
wire[15:0] ReadData;

initial
$monitor("Clock=%b; MemWrite=%b; MemRead=%b; Adresa=%d; WriteData=%h; ReadData=%b",Clock, MemWrite,MemRead, Adresa,WriteData,ReadData);

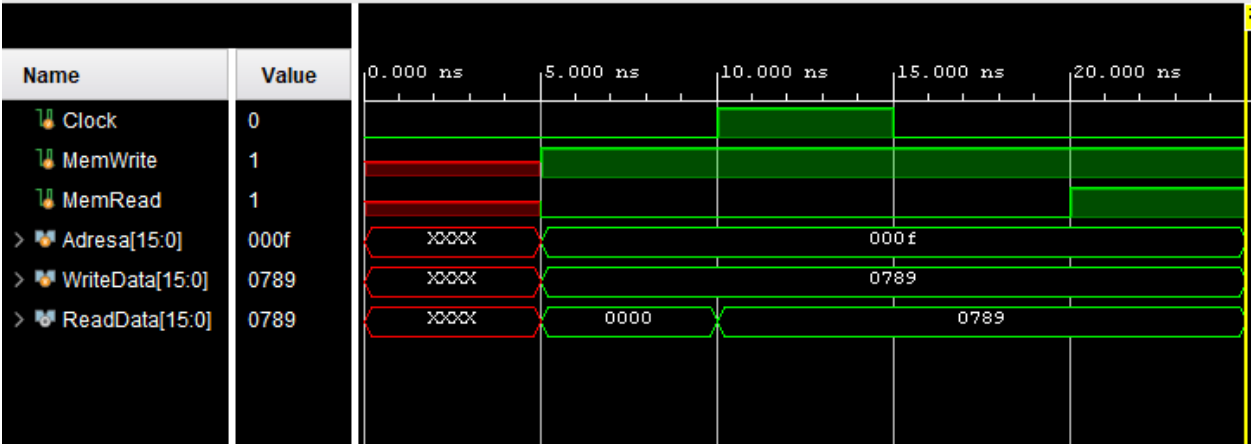
initial
begin
#0 Clock=1'b0;
#5 MemWrite=1'b1; Adresa=16'd15; WriteData=16'h789; MemRead=1'b0;
#5 Clock=1'b1;
#5 Clock=1'b0; MemWrite=1'b1;
#5 MemRead=1'b1; Adresa=16'd15;
#5 $stop;
end

DataMemory DM(Adresa, WriteData, MemWrite, MemRead, Clock, ReadData);
endmodule
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

DataMemory.mem

1	00
2	00
3	00
4	00
5	00
6	00
7	00
8	00
9	00
10	00
11	00
12	00
13	00
14	00
15	00
16	07
17	89
18	00



```
# run 1000ns
Clock=0; MemWrite=x; MemRead=x; Adresa=      x; WriteData=xxxx; ReadData=xxxxxxxxxxxxxxxx
Clock=0; MemWrite=1; MemRead=0; Adresa=    15; WriteData=0789; ReadData=0000000000000000
Clock=1; MemWrite=1; MemRead=0; Adresa=    15; WriteData=0789; ReadData=0000011110001001
Clock=0; MemWrite=1; MemRead=0; Adresa=    15; WriteData=0789; ReadData=0000011110001001
Clock=0; MemWrite=1; MemRead=1; Adresa=    15; WriteData=0789; ReadData=0000011110001001
```

Register File

```

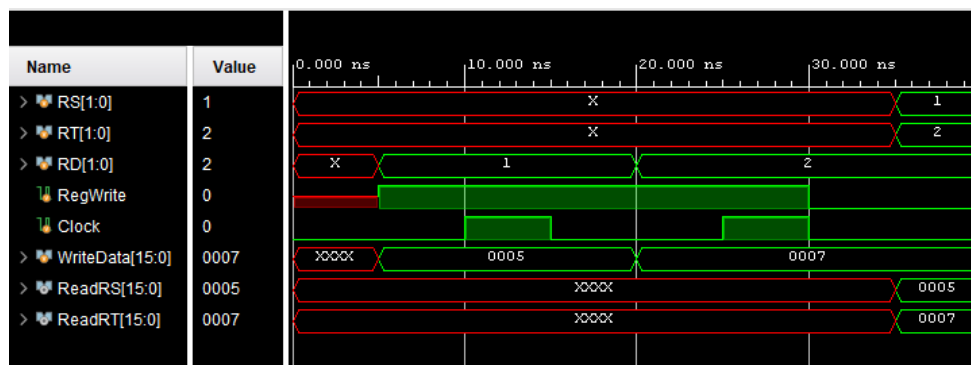
module RegisterFile_Test();
reg[1:0] RS, RT, RD;
reg RegWrite, Clock;
reg[15:0] WriteData;
wire[15:0] ReadRS, ReadRT;

initial
$monitor("RS=%d; RT=%d; RD=%d; RegWrite=%b; Clock=%b; WriteData=%d; ReadRS=%b;ReadRT=%b",
RS, RT, RD,RegWrite, Clock,WriteData,ReadRS, ReadRT );

initial
begin
#0 Clock=1'b0;
#5 RD=2'd1; WriteData = 16'd5; RegWrite=1'b1;
#5 Clock=1'b1;
#5 Clock=1'b0;RegWrite= 1'b1;
#5 RD=2'd2; WriteData = 16'd7; RegWrite=1'b1;
#5 Clock=1'b1;
#5 Clock=1'b0; RegWrite=1'b0;
#5 RS=2'd1; RT=2'd2;
#5 $stop;
end

RegisterFile RF(RS, RT, RD, WriteData, ReadRS, ReadRT, RegWrite, Clock);
endmodule

```



```

# run 1000ns
RS=x; RT=x; RD=x; RegWrite=x; Clock=0; WriteData=  x; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=x; RT=x; RD=1; RegWrite=1; Clock=0; WriteData=  5; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=x; RT=x; RD=1; RegWrite=1; Clock=1; WriteData=  5; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=x; RT=x; RD=1; RegWrite=1; Clock=0; WriteData=  5; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=x; RT=x; RD=2; RegWrite=1; Clock=0; WriteData=  7; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=x; RT=x; RD=2; RegWrite=1; Clock=1; WriteData=  7; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=x; RT=x; RD=2; RegWrite=0; Clock=0; WriteData=  7; ReadRS=xxxxxxxxxxxxxxxx;ReadRT=xxxxxxxxxxxxxxxx
RS=1; RT=2; RD=2; RegWrite=0; Clock=0; WriteData=  7; ReadRS=0000000000000101;ReadRT=0000000000000111

```


ALU Control

```

module ALUControl_Test();
  reg [1:0] ALUOp;
  reg [1:0] Funct;
  reg [3:0] Opcode;
  wire [3:0] ALUCtrl;

  initial
    $monitor("ALUOp = %b; Funct = %b; Opcode = %b; ALUCtrl = %b", ALUOp, Funct, Opcode, ALUCtrl);

  initial
    begin
      #0 ALUOp = 2'b00;
      #5 ALUOp = 2'b01;

      #5 ALUOp = 2'b10; Opcode = 4'b0000; Funct = 2'b00;
      #5 ALUOp = 2'b10; Opcode = 4'b0000; Funct = 2'b01;
      #5 ALUOp = 2'b10; Opcode = 4'b0000; Funct = 2'b10;

      #5 ALUOp = 2'b10; Opcode = 4'b0001; Funct = 2'b00;
      #5 ALUOp = 2'b10; Opcode = 4'b0001; Funct = 2'b01;

      #5 ALUOp = 2'b10; Opcode = 4'b0010; Funct = 2'b00;
      #5 ALUOp = 2'b10; Opcode = 4'b0010; Funct = 2'b01;

      #5 ALUOp = 2'b11; Opcode = 4'b1001; Funct = 2'bXX;
      #5 ALUOp = 2'b11; Opcode = 4'b1010;
      #5 ALUOp = 2'b11; Opcode = 4'b1011;

      #5 $stop;
    end

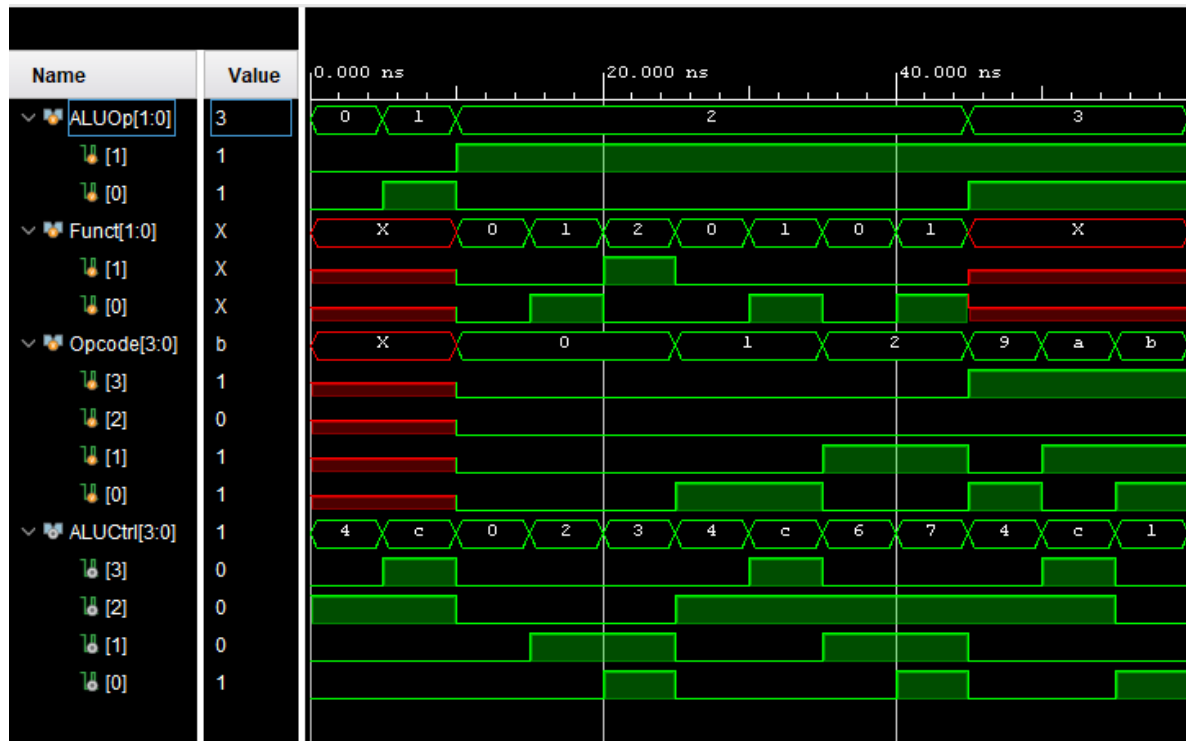
  ALUControl aluCon (
    .ALUOp(ALUOp),
    .Funct(Funct),
    .Opcode(Opcode),
    .ALUCtrl(ALUCtrl)
  );
endmodule

```

```

# run 1000ns
ALUOp = 00; Funct = xx; Opcode = xxxx; ALUCtrl = 0100
ALUOp = 01; Funct = xx; Opcode = xxxx; ALUCtrl = 1100
ALUOp = 10; Funct = 00; Opcode = 0000; ALUCtrl = 0000
ALUOp = 10; Funct = 01; Opcode = 0000; ALUCtrl = 0010
ALUOp = 10; Funct = 10; Opcode = 0000; ALUCtrl = 0011
ALUOp = 10; Funct = 00; Opcode = 0001; ALUCtrl = 0100
ALUOp = 10; Funct = 01; Opcode = 0001; ALUCtrl = 1100
ALUOp = 10; Funct = 00; Opcode = 0010; ALUCtrl = 0110
ALUOp = 10; Funct = 01; Opcode = 0010; ALUCtrl = 0111
ALUOp = 11; Funct = xx; Opcode = 1001; ALUCtrl = 0100
ALUOp = 11; Funct = xx; Opcode = 1010; ALUCtrl = 1100
ALUOp = 11; Funct = xx; Opcode = 1011; ALUCtrl = 0001

```



Control Unit

```

)
    )
        *bedmktce (bedmktce)
        *ytnztc (ytnztc)
        *ytnob (ytnob)
        *newpseq (newpseq)
        *pseusp (pseusp)
        *newmktce (newmktce)
        *newtoqed (newtoqed)
        *bedrac (bedrac)
        *obcoqe (obcoqe)

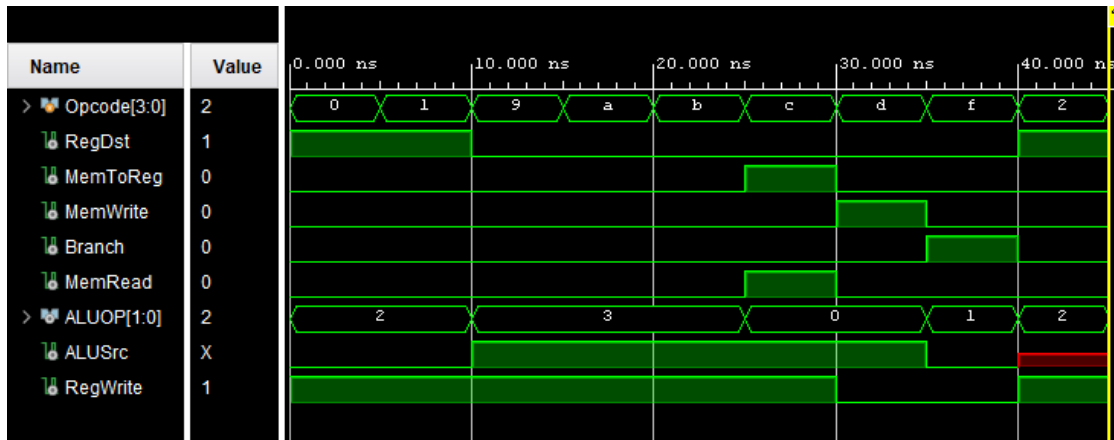
    couctotuttc cn (
        \ \ couctotuttc cn ( obcoqe bedrac newtoqed newmktce pseusp newpseq ytnob ytnztc bedmktce )
    )
    euq
    #2 zefob:

    #2 obcoqe = #,p0010:
    #2 obcoqe = #,p1111:
    #2 obcoqe = #,p1101:
    #2 obcoqe = #,p1100:
    #2 obcoqe = #,p1011:
    #2 obcoqe = #,p1010:
    #2 obcoqe = #,p1001:
    #2 obcoqe = #,p0001:
    #0 obcoqe = #,p0000:
)
    peditu
)
    tutctet

    obcoqe bedrac ytnztc newtoqed bedmktce newpseq newmktce ytnob pseusp:
    zewutfoi ( obcoqe = sp: bedrac = sp: ytnztc = sp: newtoqed = sp: bedmktce = sp: newpseq = sp: newmktce = sp: ytnob = sp: pseusp = sp: )
)
    tutctet

    atle bedmktce:
    atle ytnztc:
    atle [7:0] ytnob:
    atle newpseq:
    atle pseusp:
    atle newmktce:
    atle newtoqed:
    atle bedrac:
    led [3:0] obcoqe:
)
    moqnte couctotuttc leat():

```



```
# run 1000ns
```

```

Opcode = 0000; RegDst = 1; ALUSrc = 0; MemToReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; ALUOp = 10; Branch = 0
Opcode = 0001; RegDst = 1; ALUSrc = 0; MemToReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; ALUOp = 10; Branch = 0
Opcode = 1001; RegDst = 0; ALUSrc = 1; MemToReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; ALUOp = 11; Branch = 0
Opcode = 1010; RegDst = 0; ALUSrc = 1; MemToReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; ALUOp = 11; Branch = 0
Opcode = 1011; RegDst = 0; ALUSrc = 1; MemToReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; ALUOp = 11; Branch = 0
Opcode = 1100; RegDst = 0; ALUSrc = 1; MemToReg = 1; RegWrite = 1; MemRead = 1; MemWrite = 0; ALUOp = 00; Branch = 0
Opcode = 1101; RegDst = 0; ALUSrc = 1; MemToReg = 0; RegWrite = 0; MemRead = 0; MemWrite = 1; ALUOp = 00; Branch = 0
Opcode = 1111; RegDst = 0; ALUSrc = 0; MemToReg = 0; RegWrite = 0; MemRead = 0; MemWrite = 0; ALUOp = 01; Branch = 1
Opcode = 0010; RegDst = 1; ALUSrc = x; MemToReg = 0; RegWrite = 1; MemRead = 0; MemWrite = 0; ALUOp = 10; Branch = 0

```

SLL

```

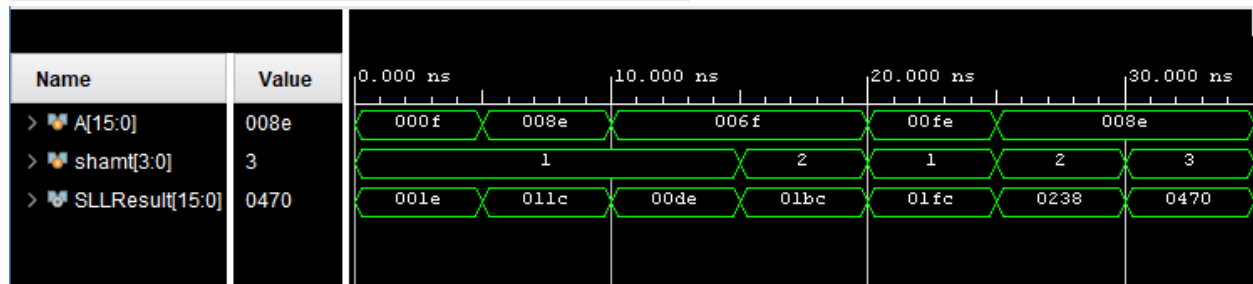
module SLL_Test();

    reg [15:0] A;
    reg [3:0] shamt;
    wire [15:0] SLLResult;

    initial
    $monitor("A=%b; shamt=%b; SLLResult=%b", A, shamt, SLLResult);
    initial
    begin
        #0 A = 16'b00001111; shamt = 4'd1;
        #5 A = 16'b10001110; shamt = 4'd1;
        #5 A = 16'b01101111; shamt = 4'd1;
        #5 A = 16'b01101111; shamt = 4'd2;
        #5 A = 16'b11111110; shamt = 4'd1;
        #5 A = 16'b10001110; shamt = 4'd2;
        #5 A = 16'b10001110; shamt = 4'd3;
        #5 $stop;
    end

    SLL sll(
        .A(A),
        .shamt(shamt),
        .SLLResult(SLLResult)
    );
endmodule

```



```

# run 1000ns
A=00000000000001111; shamt=0001; SLLResult=00000000000011110
A=00000000010001110; shamt=0001; SLLResult=0000000100011100
A=00000000001101111; shamt=0001; SLLResult=00000000011011110
A=00000000001101111; shamt=0010; SLLResult=00000000110111100
A=00000000011111110; shamt=0001; SLLResult=00000000111111100
A=00000000010001110; shamt=0010; SLLResult=00000001000111000
A=00000000010001110; shamt=0011; SLLResult=00000010001110000

```

SRA

```

module SRA_Test();

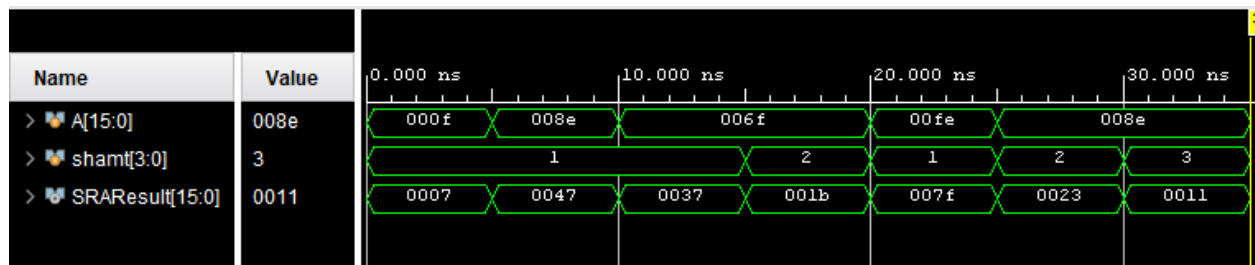
  reg [15:0] A;
  reg [3:0] shamt;
  wire [15:0] SRAResult;

  initial
    $monitor("A=%b; shamt=%b; SRAResult=%b", A, shamt, SRAResult);

  initial
  begin
    #0 A = 16'b00001111; shamt = 4'd1;
    #5 A = 16'b10001110; shamt = 4'd1;
    #5 A = 16'b01101111; shamt = 4'd1;
    #5 A = 16'b01101111; shamt = 4'd2;
    #5 A = 16'b11111110; shamt = 4'd1;
    #5 A = 16'b10001110; shamt = 4'd2;
    #5 A = 16'b10001110; shamt = 4'd3;
    #5 $stop;
  end

  SRA sra(
    .A(A),
    .shamt(shamt),
    .SRAResult(SRAResult)
  );
endmodule

```



```

# run 1000ns
A=000000000000001111; shamt=0001; SRAResult=00000000000000111
A=00000000010001110; shamt=0001; SRAResult=00000000001000111
A=00000000001101111; shamt=0001; SRAResult=00000000000110111
A=00000000001101111; shamt=0010; SRAResult=00000000000011011
A=00000000011111110; shamt=0001; SRAResult=00000000001111111
A=00000000010001110; shamt=0010; SRAResult=00000000000100011
A=00000000010001110; shamt=0011; SRAResult=0000000000010001

```

SLTI

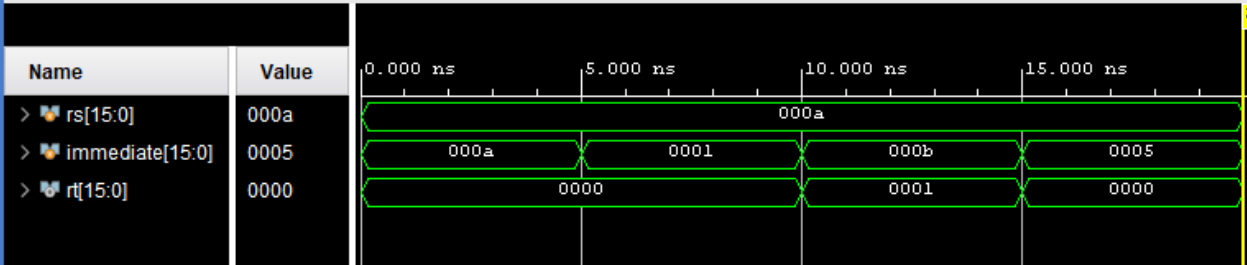
```
module SLTI_Test();
  reg[15:0] rs; // A
  reg[15:0] immediate; // mB
  wire[15:0] rt; // slti

  initial
    $monitor("A(rs)=%b; imediate=%b; rt=%b", rs, immediate, rt);

  initial
  begin
    #0 rs = 16'd10 ; immediate = 16'd10;
    #5 rs = 16'd10 ; immediate = 16'd1;
    #5 rs = 16'd10 ; immediate = 16'd11;
    #5 rs = 16'd10 ; immediate = 16'd5;

    #5 $stop;
  end

  SLTI slti(
    .rs(rs),
    .immediate(immediate),
    .rt(rt)
  );
endmodule
```




```
# run 1000ns
A(rs)=00000000000001010; imediate=00000000000001010; rt=00000000000000000
A(rs)=00000000000001010; imediate=00000000000000001; rt=00000000000000000
A(rs)=00000000000001010; imediate=00000000000001011; rt=00000000000000001
A(rs)=00000000000001010; imediate=0000000000000101; rt=00000000000000000
```

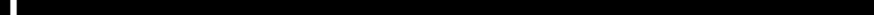
CPU

```

module CPU_Test();                                # run 1000ns
reg Clock;                                         Clock=0
                                                    Clock=1
initial                                             Clock=0
$monitor("Clock=%b",Clock);                       Clock=1
                                                    Clock=0
initial                                             Clock=1
begin                                               Clock=0
#0 Clock = 1'b0;                                    Clock=1
#100 Clock=1'b0;                                   Clock=0
end                                                  Clock=1
                                                    Clock=0
always                                              Clock=1
begin                                               Clock=0
#5 Clock=~Clock;                                    Clock=1
end                                                  Clock=0
                                                    Clock=1
CPU cpu(Clock);                                    Clock=0
                                                    Clock=1
endmodule                                           Clock=0
                                                    Clock=1
                                                    Clock=0
                                                    Clock=1
                                                    Clock=0
                                                    Clock=1
                                                    Clock=0
                                                    Clock=1
                                                    Clock=0
                                                    Clock=1

```

Name	Value
 Clock	1



Konkluzioni

Arkitektura e CPU-së është një aspekt i rëndësishëm i dizajnit të harduerit, duke përcaktuar funksionimin dhe aftësitë e një kompjuteri. Procesori është përgjegjës për ekzekutimin e programeve dhe ndikon drejtpërdrejtë performancën e sistemit. Duke pasur përvojën me detyrën e dizajnit të një procesori duke u bazuar në parimin e Single Cycle, kemi kuptuar në thellësi sfidat dhe shprehshmëritë praktike të një dizajni të tillë. Ky proces na ka ofruar një perspektivë të thelluar mbi kompleksitetin e implementimit të një arkitekture të tillë dhe rëndësinë e zgjedhjeve dizajni në krijimin e një CPU efikase dhe të përputhur me kërkesat e programeve të ndryshme.