# UNIVERSITETI I PRISHTINËS

# “HASAN PRISHTINA”


**Studentët:**Donika Rexhepi-ID:

Enis Berisha-ID:180714100034,

Eron Salihu-ID:180714100080

**Lënda:**Arkitektura e Kompjuterëve

**Projekti:**2

**Tema:**CPU 16bit në Verilog

Në këtë detyrë qëllimi kryesor ka qenë dizajnimi I një CPU 16 bitësh e cila në vetvete mbart disa operacione të thjeshta aritmetikore dhe logjike.CPU është 16 single-cycle.

CPU duhet të përkrah operacionet AND,OR,XOR,ADD,SUB,ADDI,SHIFT_LEFT,SHIFT_RIGHT,MULT,LË,SË e cila përkrah instruksione të formatit R dhe disa nga formati I.

## ALU 1-bit

Alu 1 bit ka 5 hyrje a,b,cin,bnegate,dhe 3 bitat që tregojnë zgjedhjen e operacionit dhe 2 dalje rezultati dhe cout.

Tek Alu 1 bit kryhen 5 operacione të caktuara siq janë AND,OR,XOR,ADD,SUB,ADDI.

AND kryhet me operacion të gatshëm gjithashtu dhe OR,XOR,kurse për ADD,SUB,ADDI përdoret një full Adder I gatshëm nga ligjeratat.Për ekzekutimin e zbritjes përdorim një mux I cili nëse është 1 atëher e kthen vlerën nga positive në negative.

Një mux 8_1 I marrë nga libri i ushtrimeve na ndihmon në zgjedhjen e operacioneve nga 3 bitët që I kemi caktuar në pozitat e duhura kurse pozitat të cilat nuk plotëshen I kemi mbushur me vlera default të a.

## ALU 16-bit

Për ekzekutimin e ALU 16 bit kemi 1 hyrje 4 bitëshe të cilën e merr nga ALU-Control ku në të cilën janë të zgjedhura most significant bit është bnegate kurse 3 bitat e tjerë përkujdesen për zgjedhjen e një ndër 8 operacionet.Pasi që disa operacione kryhen në nivele 1 bitëshe  thërrasim 1 modul I cili mbrenda vetes thërret 16 herë ALU-1bitësh dhe pastaj me anë të një case zgjedhim operacionin të cilin dëshirojmë të jetë ndër të cilat janë përfunduar AND,OR,XOR,ADD,ADDI,SUB,SHIFTLEFT,SHIFTRIGHT,MULT si dhe LË dhe SË të cilat kryejnë mbledhjen e zakonshme.

MULT është marrë I gatshëm nga libri I ushtrimeve I cili bën shumëzimin 8X8 bitësh ku rezultati në dalje maksimum mund të jetë 16 bitësh.

Gjithashtu ekzekutimin e ALU në bashkëpunim me ALU-CONTROL dhe CONTROL e kemi testuar veqmas para kompletimit të projektit e cila ka funksionuar në mënyrë perfekte duke zgjedhur dhe kryer operacionet e saja sic edhe është kërkuar nga detyra.

```
 assign op=vlera[2:0];
 assign bnegate=vlera[3];
 always @(bnegate)
 begin
    if(bnegate==1)
        cin=1'b1;
     else
        cin=1'b0;
end
froml_tol6 bashko(a,b,cin,bnegate,op,result,cout);
shumezuesi shumezo(a[7:0],b[7:0],shuma[15:0]);//mer
always @*
begin
case (op)
//shift left
3'b001 :
    REZULTATI={a[14:0],{1{1'b0}}};

//shift right
3'b110:
    REZULTATI={{1{1'b0}},a[15:1]};
```

Shihet se si vlera nje input I marrë nga alu control I jep vlerë bnegates dhe gjithashtu 3 bitave të shënuar si op pastaj me anë të të cilave zgjedhim operacionin.Gjithashtu shohim se cin merr vlerën e bnegate.

## Kontrolli(Controlli):

Kontrolli ka rol të thjeshtë ku vetëm merr 2 bitat e parë nga Instruction memory si hyrje dhe pastaj I aranzhon daljet e tij sipas specifikave që ka.Pasi që të githa operacionet në R format kanë opcode të njejtë atëher edhe daljet e tyre janë të njëjta sipas specifikave që na janë dhënë në projekt kurse dallojnë për addi dhe për lë dhe së opcode është I ndryshëm prandaj edhe daljet nga KONTROLLI janë të ndryshme sipas tabelës së marrë nga pdf në spjegimin e projektit.Kemi pasë edhe 1 model të ngjashëm të cilin e kemi

punuar në ushtrime.Kontrolli merr bitat e instruksionit[15:14] dmth dy bitat me të lartë në instruksion ose bitat e parë.

## ALU-CONTROL:

ALU-CONTROL përmban 2 hyrje ku njëra është ALUOp e marrë nga CONTROLI dhe hyrja tjetër është funksioni I marrë nga Instruksioni që del nga instruction memory lku funksioni përfshin 4 bitat e fundit nga instruksioni dmth instruksioni[4:0] dhe pastaj formon një variabël të re me bashkimin e të dyjave nëse op është 1 dhe tregon ekzaktësisht nga funksioni jep në dalje bitat të cilat nevoiten për të kryer atë operacion që cakton ai funksion për ta dërguar në ALU kurse nëse op është 0 për lë dhe për së ia jep bitat e nevojshëm për të kryer mbledhjen e zakonshme.

```
 reg[3:0] AluControl;
wire [5:0] AluControlHyrja;
 assign AluControlHyrja = {AluOp,Funksioni};
 always @(AluControlHyrja)
if(AluOp==1'b0)
AluControl=4'b0100;

else
begin
case (Funksioni)
// dhe logjike
5'b00001:
AluControl=4'b0000;

// ose logjike
5'b00010:
AluControl=4'b0010;
```

Këtu shihet formimi I një variable AluControlHyrja e formuar nga 1 bit I aluop dhe funksionit të cilën e kam parë dhe marrë në https://ëëë.fpga4student.com/ ku jam bazuar për disa nga punimet e tona.

--Instruction Memory Registre File dhe DataMemory si dhe 2 memory filet janë marrë të gatshme nga ushtrimet pasi edhe pas disa research të bëra kam vërejtur ngjashmërin e madhe që ka pasë përveq disa shtimeve në REGISTER FILE ku I kam treguar se nëse RS ose RT është 0 atëher vlera e atij regjistri është 0 në decimal.Gjithashtu kam pas

mundësin të vërej se mënyra e qasjes në file të memories ka qenë e ndryshme nga ajo e bërë në ushtrime.—

## DATA-PATH:

Në DATA-PATH janë bërë lidhjet e të gjithave sipas renditjes dhe rregullave ku për lidhjen e tyre jam bazuar në ushtrimet dhe fotografitë duke shkuar step by step kam mundur të bëj lidhjen efektive dhe të rregullt siq duhet bërë.

Në data-Path inpute janë të gjitha outputet e controllit pasi që controlli llogaritet si element I jashtëm nga data-pathi.

Së pari marrim pc të cilën e dërgojmë në instruction memory ku lexon vlerën në atë adresë pastaj instruction memory I jep vlera të gjithave opcodes,RS,RT,RD,immediate dhe function I pasuron me vlera sipas renditjes ku ato vlera pastaj disa shkojnë në Register file kurse funksioni dhe vlera immediate shkojnë në alu control.

Për arsye se I formati nuk futë në punë RD atëher para tij vendoset një mux ku nëse vlera e marrë RegDst është 1 atëher RD është regjistri I cili do marrë vlerën nga ërite data kurse nëse është 0 atëher RT do të marrë atë vlerë.

Në Register-File dërgohen vlerat e zakonshme dhe Regjistri RegËrite I cili tregon nëse shkruajmë në ëriteData apo jo e cila është 0 vetëm në rastin e storeËord.

```
memory DPIMemory(clk,pc,instruksioni);

 assign opcode=instruksioni[15:14];
 assign RS=instruksioni[13:11];
 assign RT=instruksioni[10:8];
 assign RD=instruksioni[7:5];
 assign Funksioni=instruksioni[4:0];
 assign Immidiate=instruksioni[7:0];



// mux_2_1 zgjedhjaRF(RegDst,RT,RD,daljaMux);
 assign daljaMux=RegDst?RD:RT;
 RegistreFile RFDP(RS,RT,daljaMux,WriteData,RegWrite,clk,RD1,RD2);
 assign immidiate_long= {{8{Immidiate[7]}}, Immidiate[7:0]};

// mux_2_1 zgjedhIM_RD(ALUSrc,RD2,Immidiate,VLERA);

 assign VLERA=ALUSrc?immidiate_long:RD2;

 aluControl opii(ALUOp,Funksioni,AluControl);
 ALU aluDP(AluControl,RD1,VLERA, cout,REZULTATI);
```

Shihet zgjerimi I vlerës immediate nga 8 bitë siq e ka në instruksion në 16 bit e cila njihet si sign-extend dhe pastaj kemi një mux 2-1 e cila varet nga ALUSrc mbi të cilën shikohet nëse do të mirret vlera immediate ose vlera nga RD2 dhe dërgohen ato vlera në ALU.

Kurse në datamemory vlera e parë që mirret është rezultati nga alu dhe vlera e dytë është RD2 që mirret direkt nga Register-File ku ka edhe 2 vlera të cilat mirren si inpute nga Controlli të cilat janë MemËrite dhe MemRead ku MemËrite shkruan në memorie e cila është 1 vetëm për storeËord kurse MemRead është 1 vetëm për loadËord.

Pastaj kemi një mux I cili në varësi nga MemToReg zgjedh nëse do të marri Rezultatin nga dalja e ALU apo daljen e Data Memoryt ku memtoreg është 1 vetëm për loadËord.

```
 dataMemory DPdataM(REZULTATI, RD2,MemWrite,MemRead,clk,ReadData);

//  mux_2_1 shkruarja(MemToReg,REZULTATI,ReadData,WriteData);
 assign WriteData=MemToReg?ReadData:REZULTATI;
```

## CPU:

CPU bën thirrjen e dy moduleve të DataPath-it si dhe të Controllit ku me bashkimin e tyre bëhet CPU komplet dhe ekzekutohet në mënyrë të duhur.

```
module CPU(
input clk
    );
    wire clk;
    wire pcFill=16'd10;
    wire RegDst;
    wire AluSrc;
    wire MemToReg;
    wire RegWrite;
    wire MemRead;
    wire MemWrite;
    wire AluOp;
    wire [1:0] opcode;
    DataPath CPU_DP( clk, pcFill, RegDst, AluSrc, MemToReg , RegWrite , MemRead , MemWrite,  AluOp, opcode);
    controlli CPU_c(opcode, RegDst, AluSrc, MemToReg , RegWrite , MemRead , MemWrite, AluOp);


endmodule
```

## TESTIMET:

Secila nga modulet veqmas përmban testimet e tyre të cilat janë të organizuara me emra të njëjtë si modulet përveq shtesës _tb për tregimin që është test_banch tek folder I simulation apo testimet.Për shkak që zhvillimi I detyrës ka qenë sekuencial testimet kanë qenë të nevojshme për secilën pjesë dhe kombinimin e disa pjesëve së bashkunë mënyrë të shikimit nëse detyra po vijon si duhet dhe kombinimi I pjesëve së bashku nuk paraqet problem.Pas kombinimit të disa pjesëve ka pasur shumë inpute dhe outpute të ndryshuara ngaqë të dhënat e marra tani po vinin nga modulet e tjera jo si fillestare në  bërjen e modulit në fillim.

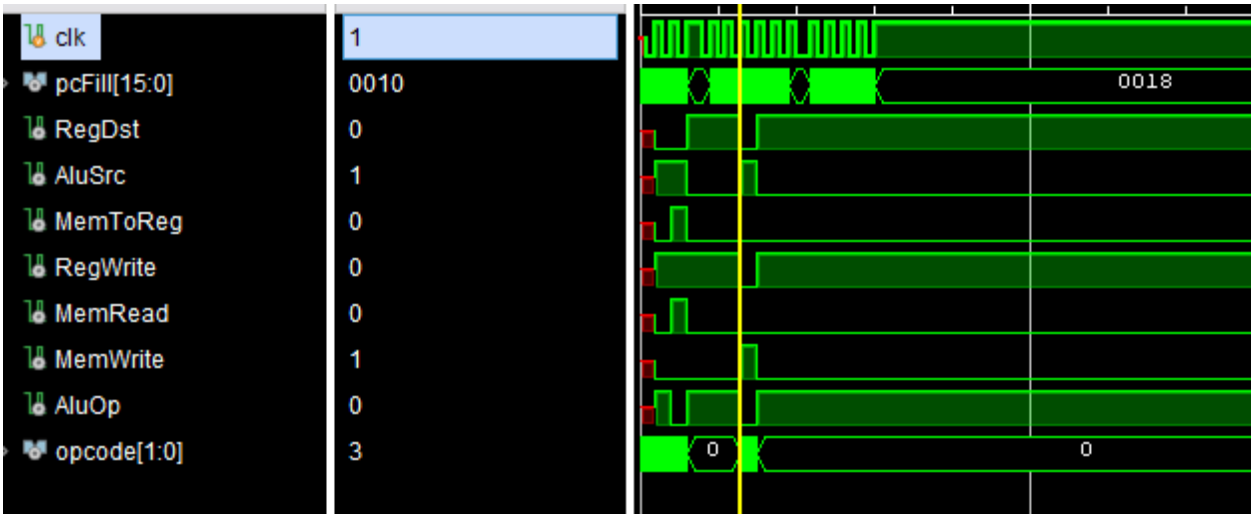
```
module ALU_tb(
);
    reg[2:0] op;

    reg [2:0]op;
    reg[15:0] a, b;
    reg cin, bnegate;
    wire[15:0] REZULTATI;
    wire cout;

    initial

    $monitor ("op=%b,a = %b, b = %b, cin = %b,bnegate=%b, cout = %b, REZULTATI=%d",
    op,a, b, cin,bnegate, cout, REZULTATI);
```

Screenshoti I nxjerrë më lart është një ndër testimet e para të ALU ku shihen edhe cin dhe bnegate si inpute të marra nga useri por pastaj me zhvillimin e datapathit skemi pasur nevojë më për këto të dyja ngaqë I kemi marrë të gatshme nga moduli paraprak I thirrur që është ALUCONTROLI

```
module ALUCONTROLLO(
    );

     reg[1:0] opcode;
    reg [4:0] Funksioni;
    reg[15:0] a, b;

    wire[15:0] REZULTATI;
    wire cout;

    initial

    $monitor ("opcode=%b,Funksioni=%b,a = %b, b = %b, cout = %b, REZULTATI=%d",
    opcode,Funksioni,a, b,cout, REZULTATI);
```

Këtu shihet kombinimi I ALU me ALUCONTROLIN ku ALU mbrenda vetes thërret ALUCONTROLIN në mënyrë që të shikohet nëse ALUCONTROL po dërgon bitat e duhur për kryerjen e operacioneve të nevojshme.

TESTIMI CPU:




Në këtë testim kam vërejtur rritjen e pcFill qdoherë që clk është në tehun pozitiv siq edhe kërkohet në detyrë gjithashtu në mënyrë që të kontrolloj mbarëvajtjen e programit kam kontrolluar nëse bitat që I kemi je psi bita dalës po dalin sipas kushteve të nevojshme duke u bazuar në operacionet që kemi cekur në instruction memory.

Rritja është bërë për 1 për arsye se instruction memory është e organizuar në atë mënyrë.Nëse 1 byte kishte qenë e shkruar në 1 rresht kurse byte tjetër në rreshtin tjetër atëher rritja është dashur të bëhet për 2.

Gjithashtu kam bërë testimin e datapathit veqmas në të cilën kam shfaqur outputet e ndryshme ku pastaj I kam lënë si komente për arsye testimi.

```
output opcode
//output RS,
//output RT,
//output RD,
//output daljaMux,
//output immidiate_long,
//output WriteData,
//output RD1,
//output RD2,
//output VLERA,
//output REZULTATI,
//output ReadData
//
```

```
$monitor ("clk=%b,pcFill=%b,RegDst = %b, ALUSrc = %b, MemTo
clk,pcFill,RegDst,ALUSrc,MemToReg,RegWrite,MemRead,MemWrite
 initial
    begin
     #10 clk=1'b0;
      #10 clk=1'b1;pcFill=16'd11;  RegDst=1'b0;ALUSrc=1'b1;
      #10 clk=1'b0;
      #10 clk=1'b1;pcFill=16'd12; RegDst=1'b0;ALUSrc=1'b1;M
      #10 clk=1'b0;
      #10 clk=1'b1;pcFill=16'd13;  RegDst=1'b1;ALUSrc=1'b0;
      #10 clk=1'b0;
      #10 clk=1'b1;pcFill=16'd14;  RegDst=1'b1;ALUSrc=1'b0;
```

Screeni I parë outputet të cilat kanë siguruar që datapathi po punon në mënyrën e duhur kurse screeni I dytë mënyra e pakët e testimit.

Vështirësi ka qenë mbartja e regjistrave pasi që nuk mbanin vlerën aktuale por ktheheshin në vlera null,ose të zbrazëta.

## Përfundimi:

E mira e këtij projekti ka qenë familjarizimi me Verilog si dhe dhënja e kuptimit të drejtimit tonë Inxhinieri Kompjuterike pasi që po kthehemi në gjuhën e parë dhe të ulët programuese në dizajnimin e një CPU në single cycle shpresoj që kjo të vazhdojë edhe në gjëra më të detajuara dhe të pasurohemi edhe me njohuri me bazë Inxhinierike e jo vetëm koncentrimi në gjuhët e larta duke marrë kështu epitetin e njëjtë me studentët që bëjnë pjesë në shkenca kompjuterike.

Referenca: https://ëëë.fpga4student.com/

https://electronics.stackexchange.com/