Universiteti i Prishtinës "Hasan Prishtina"

Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike

CPU 16-bit

Studenti	Arbresha Ejupi Blerina Durguti	ID	200714100073 200714100023
Grupi	31	Profesori	Valon Raca

Hyrje

-Detyra kërkon krijimin e një CPU apo procesori 16-bit e cila është e implementuar në Single-Cycle Datapath me anë të veglës Verilog. Punimi i detyrës është tërësisht analogji me atë të procesorit 32 bit, me disa modifikime, por në parim pra janë të njejta.

-Procesori 16-bit ka kuptimin se regjistrat, instruksionet, adresat janë 16 bit.

-Punimi i detyrës është bërë në mënyrë parciale, pra fillimisht janë implementuar pjesët përbërëse si:

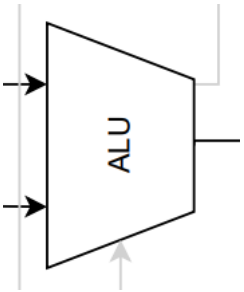
- Mux2ne1(Multiplekseri 2 në 1 (1bitësh))
- Mbledhësi
- Mux8ne1(Multiplekseri 8 në 1 apo Multiplekseri Kryesor)
- ALU1
- ALU16
- ALU Control

- Register file
- Instruction Memory
- Data Memory
- Datapath
- Control Unit

-Pastaj pjesa e fundit vjen ndërlidhja e këtyre elementeve në formimin e tërësishëm të CPU-së.

-Në kuadër të elementeve të specifikuara kemi nënelemente tjera, fillojmë me ALU-në.

ALU 16-bit



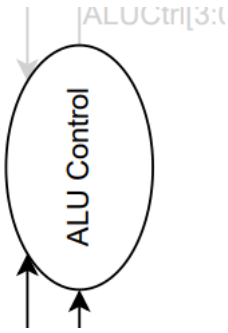
-Përgjegjëse për kryerjen e operacioneve si mbledhje, zbritje, and, or.

-Përkrah numra me gjatësi 16 bitëshe.

-Fillimisht është krijuar ALU-1 bit e cila i përkrah veprimet e lartëcekura, pastaj janë lidhur 16 ALU-1 bitëshe për kompletim të ALU-së 16-bitëshe. Brenda kësaj ALU kemi një multiplekser 8 në 1 i cili është përgjegjës për zgjedhjen e veprimit (pra njërit nga veprimet lartë).

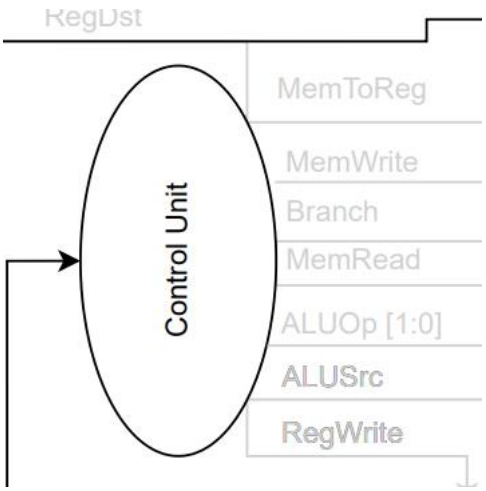
-Përpos kësaj ALU, kemi edhe ALU të cilat kryejnë vetëm mbledhje, sic është rasti i ALU tek pc e cila e rrit PC për 2, apo ALU tjetër e cila poashtu kryen mbledhje në rastin kur kemi branching condition, pra në rastin konkret mbledh vlerën imediate të shiftuar majtas me PC+2.

ALU control



-ALU control është përgjegjës që ALU-së 16 bitëshe me i dërgu bita të cilët e specifikojnë veprimin që ka me u kry në ALU. Pra, disa prej këtyre bitave ndahen për multiplekserin e specifikuar më lartë e disa prej tyre për invertim të njërës hyrje.

Control Unit



-Pjesë stateless, pjesë përgjegjëse për thuaja të gjitha veprimet brenda CPU-së.

-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,

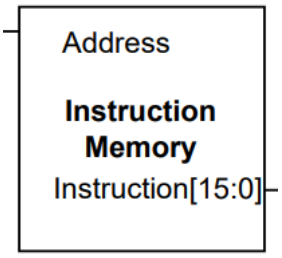
-MemWrite të shkruajmë në memorie,

-Branch për kërcime me bne,

-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 të formatit r,

-RegWrite për vlerën 1 lejon me shkrujt në regjistër edhe ALUsrc i cili është bit selektues tek multiplekseri 2 në 1 i cili shërben për të marrë vlerën imediate apo vlerën e read data 2, varësisht nga lloji i instruksionit I apo R.

Instruction memory



-Memorien e instruksioneve e poashtu edhe atë të datave e kemi të specifikuar si kërkesë deri në 128 bajt.

-Instruksionet të cilat i përkrah CPU-ja janë instruksionet e formatit I dhe formatit R.

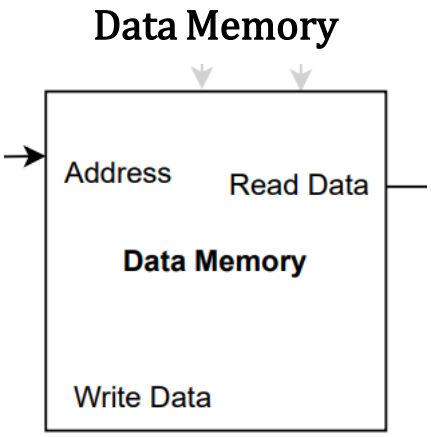
-Kuptohet se instruction memory është read only.

-Instruksionet e formatit R e kanë op code-in e specifikuar si 000 (pra op code 3 bitësh), kurse veprimet që kryhen pastaj variojnë në bazë të pjesës së fundit, pjesës së funct e cila është 4 bitësh. Formati i intruksionit të R-formatit:

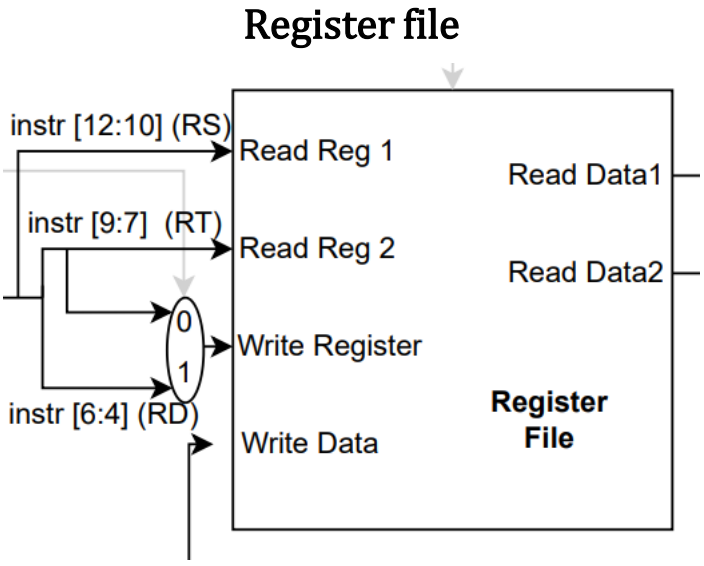
opcode-000 rs-xxx rt-xxx rd-xxx funct -xxxx

-Ku rs, rt, rd paraqesin regjistrat, nga numri i bitave të ndarë kuptojmë se gjithsej kemi 8 regjistra, rs paraqet regjistrin e parë, rt të dytin kurse rd paraqet regjistrin ku ruajmë rezultatin(tek R formati kemi gjithmonë write në register).

-Instruksionet e I formatit i dallojmë nga ato të R formatit vetëm se nuk kanë funct e as rd, por kanë vlerë imediate 7 bitëshe, dhe dallojnë nga njëra tjetra përmes op code-it.

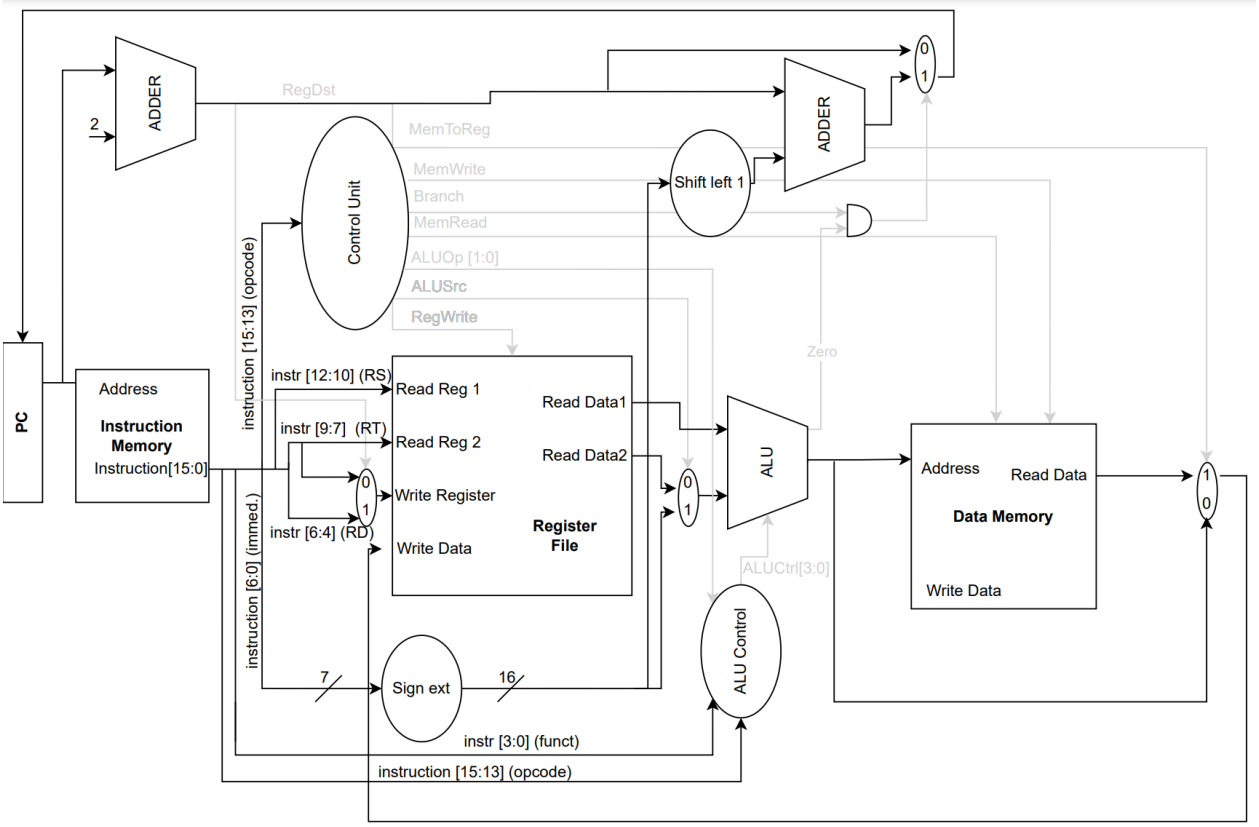


-E ngjajshme me instruction memory, përpos se këtu mundemi me shkrujt të dhëna. Shërben për instruksionet e formatit I, dy veprimet që lidhen me data memory janë sw dhe lw.



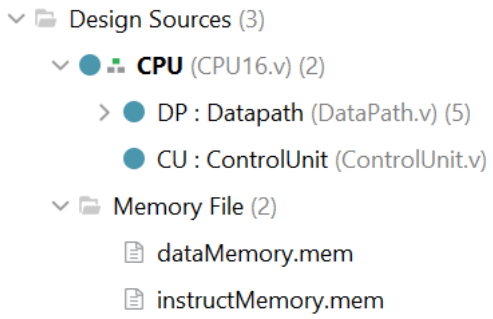
-Gjithsej 8 regjistra.

-Pamja totale e procesorit edhe me nënelementet tjera:



Dizajni

-File-t i kemi të ndarë në dy folder-a, i pari është Design Sources ku gjenden të gjithë modulet(file-t me ekstenzionin .v), dhe folder-i Memory File ku ruajmë file-t e jashtëm që i takojnë data memory edhe instruction memory (dyjat ruajnë 128 bajt madhësi të dhënash).



-Para spjegimit të moduleve është me rëndësi koncepti i modulit:

-Një modul është njësia bazike e dizajnimit në Verilog. Ajo mund të jetë e thjeshtë, apo të përmbaj module të ndërthurura në formimin e një moduli më kompleks. Mundet me qenë mbledhës, multiplekser apo diqka tjetër.

Modulet

-Modulet e përdorura në projekt:

mux2ne1

-modul i thjeshte, ka dy hyrje dhe nje bit selektues, te gjitha janë një bitëshe, ka një output e cila varet nga biti selektues, kur është zero zgjedh inputin e parë, përndryshe zgjedh inputin e dytë.

mux8ne1

-Ka 3 bita selektues, 8 hyrje të cilat janë 16 bitëshe dhe një dalje 16 bitëshe, logjika e selektimit e njëjtë, daljes i shoqërojmë:

```
assign Dalja=S[2] ? (S[1] ? (S[0] ? Hyrja4 : Hyrja7) : ( S[0] ? Hyrja6 : Hyrja2)) :
                    (S[1] ? (S[0] ? Hyrja3 : Hyrja1) : ( S[0] ? Hyrja5 : Hyrja0));
```

Mbledhesi_1bit

-Mbledhësi i plotë 1 bitësh merr si hyrje dy bita A dhe B, poashtu edhe bartjen, në dalje vendos shumën dhe vendos mbetjen e cila do shërbejë tek mbledhësi me ripple carry si bartje për mbledhësin tjetër.

-Qarku për shumën është xor mes A, B dhe bartjes kurse mbetja është A*B+A*CIN+B*CIN.

ControlUnit

-Merr si hyrje 3 bita të opcode, që i merr nga instruction memory, pastaj

përmes:

always @(OPCODE)

-Shqyrton kur kemi instruksione R apo I format dhe në varësi të instruksionit vendos bitat dalës përkatës të spjeguar lartë.

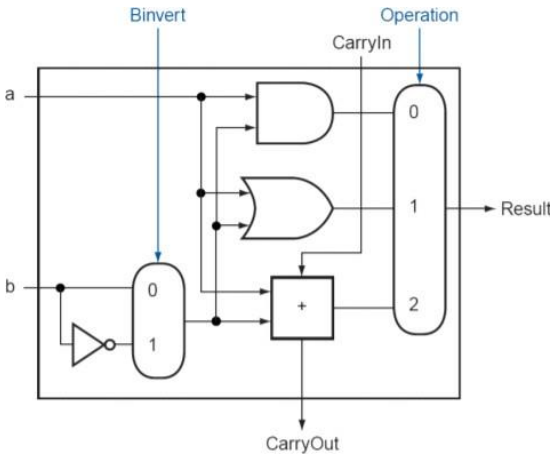
ALUControl

-Merr si hyrje bitat për operacionin e ALU-së (2 bita) që i merr nga njësia kontrolluese, pastaj inputi tjetër janë 4 bitat e funct të cilët vijnë nga instruction memory (këto shërbejnë vetëm nëse po flasim për operacione të R formatit).

-Si output jep 3 bita të cilët pastaj kalojnë në ALU-në 16 bitëshe, njëri bit shërben nëse jemi duke bërë zbritje, e inverton inputin e dytë, kurse dy të tjerët futen tek multiplekseri i ALU-së për të vendos cilin operacion me e kry.

-Kemi bërë me case (ALUop)-operacionin e ALU-së, pra rasti kur ALUop është 00, dijmë se bëhet fjalë për lw apo sw, kur ai është 01, specifikon zbritje, pra kemi të bëjmë me bne, kur është 10 e specifikojmë pastaj sipas funksionit(pasi që është R format), dhe kur ALUop është 11 e kemi bërë për operacionet që janë të I formatit.

ALU_1bit



Veprimet që kryen dhe shoqërimi i daljeve:

```
wire JoB, mB, and_wire, or_wire, mb_wire, xor_wire, mod_wire, slti_wire, addi_wire;

assign JoB=~B;
mux2ne1 muxB(B, JoB, BInvert, mB);
assign and_wire=A&mB;
assign or_wire=A|mB;
Mbledhesi m1(A, mB, CIN, mb_teli, CarryOut);
assign xor_wire=A^mB;
assign mod_wire=A%mB;
assign slti_wire=A<mB ? 1:0;
Mbledhesi m2(mb_wire, 1'b1, CIN, addi_wire, CarryOut);
mux8ne1 MuxiKryesor(and_wire, or_wire, xor_wire, mb_wire, mod_wire, slti_wire, addi_wire, Less, Op, Result);
```

ALU_16bit

-Pamja e ALU-së lart, kjo ALU ka 16 ALU 1-bitëshe të lidhura.

```
wire [14:0] COUT;
//LIDH 16 ALU 1-biteshe
ALU1 ALU0(A[0], B[0], BNegate, BNegate, Result[15], Op, Result[0], COUT[0]);
ALU1 ALU1(A[1], B[1], COUT[0], BNegate, 0, Op, Result[1], COUT[1]);
ALU1 ALU2(A[2], B[2], COUT[1], BNegate, 0, Op, Result[2], COUT[2]);
ALU1 ALU3(A[3], B[3], COUT[2], BNegate, 0, Op, Result[3], COUT[3]);
ALU1 ALU4(A[4], B[4], COUT[3], BNegate, 0, Op, Result[4], COUT[4]);
ALU1 ALU5(A[5], B[5], COUT[4], BNegate, 0, Op, Result[5], COUT[5]);
ALU1 ALU6(A[6], B[6], COUT[5], BNegate, 0, Op, Result[6], COUT[6]);
ALU1 ALU7(A[7], B[7], COUT[6], BNegate, 0, Op, Result[7], COUT[7]);
ALU1 ALU8(A[8], B[8], COUT[7], BNegate, 0, Op, Result[8], COUT[8]);
ALU1 ALU9(A[9], B[9], COUT[8], BNegate, 0, Op, Result[9], COUT[9]);
ALU1 ALU10(A[10], B[10], COUT[9], BNegate, 0, Op, Result[10], COUT[10]);
ALU1 ALU11(A[11], B[11], COUT[10], BNegate, 0, Op, Result[11], COUT[11]);
ALU1 ALU12(A[12], B[12], COUT[11], BNegate, 0, Op, Result[12], COUT[12]);
ALU1 ALU13(A[13], B[13], COUT[12], BNegate, 0, Op, Result[13], COUT[13]);
ALU1 ALU14(A[14], B[14], COUT[13], BNegate, 0, Op, Result[14], COUT[14]);
ALU1 ALU15(A[15], B[15], COUT[14], BNegate, 0, Op, Result[15], CarryOut);
```

```
assign Zero = ~(Result[0] | Result[1] |
                Result[2] | Result[3] |
                Result[4] | Result[5] |
                Result[6] | Result[7] |
                Result[8] | Result[9] |
                Result[10] | Result[11] |
                Result[12] | Result[13] |
                Result[14] | Result[15]);

    assign Overflow = COUT[14] ^ CarryOut;
```

InstructionMemory

-Mer një hyrje 16 bitëshe nga PC, e cila paraqet adresën e instruksionit, dhe një output 16 bitësh që paraqet instruksionin e caktuar, këtu kemi lexu prej fajllit të jashtëm ku i kemi vendos instruksionet:

initial
$readmemb("instructMemory.mem", instrMem);

assign Instruction[15:8] = instrMem[PCAddress];
assign Instruction[7:0] = instrMem[PCAddress + 16'd1];

-Rreshti i fundit vlen pasi që adresat i kemi 8 bitëshe, kurse dalja 16 bit, i bie se duhet të marë dy rreshta të adresës nga memoria.

RegisterFile

-3 hyrje 3 bitëshe, që mbajnë adresat e regjistrave, në dalje dy dalje 16 bitëshe, register data 1 edhe 2.

-Vlerat e regjistrave i qasemi pak a shumë si vektorëve:

assign ReadRS = Registers[RS];
assign ReadRT = Registers[RT];

-Poashtu mundemi me shkrujt në register file në vartësi te bitit kontrollues që vjen nga control unit-i, RegWrite.

-Këtu kemi clock si input (clock-u tek elementet

stateful).**DataMemory**

-Edhe këtu kemi clock si input një bitësh, pastaj bitat kontrollues memread edhe memwrite nga control unit-i.

-Adresa 16 bitëshe input vjen nga ALU-ja, edhe writedata 16 bitëshe kur kemi me shkruajt në memorie.

-Si dalje kemi readData 16 bitëshe, e cila futet në mux me adresën e ALU-së, në rast të sw apo lw, për me u shkruajt pastaj në register file.

-Edhe datamemory lexon prej një fajlli të jashtëm:
initial $readmemh("dataMemory.mem", dMem);

-Por dallimi mes instruction memory është se këtu mundemi edhe me shkruajt të dhëna, andaj duhet që ta ndërrojmë edhe përmbajtjen e fajllit, jo vetëm variablës të deklaruar brenda verilogut, pra:
$readmemb("dataMemory.mem",dataMem);

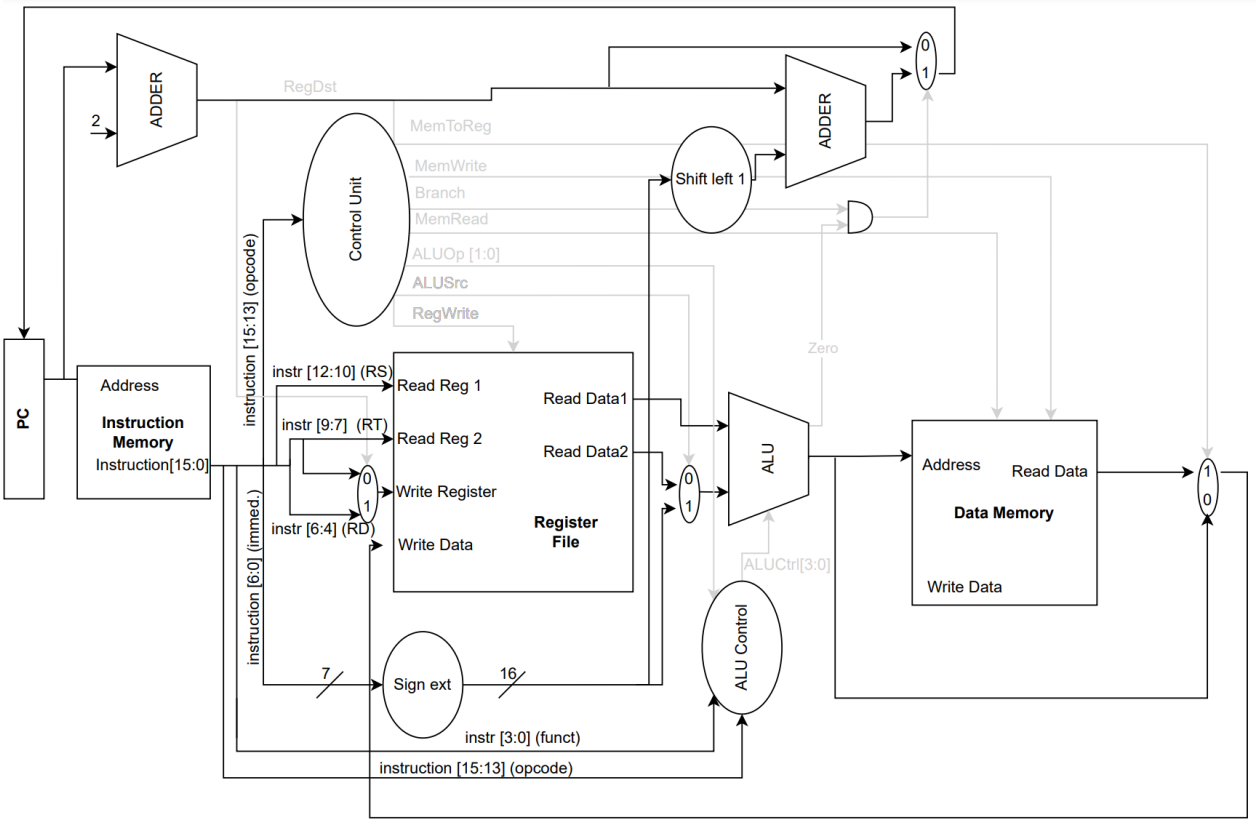
```
dataMem[Address + 16'd0] <= WriteData[15:8];
dataMem[Address + 16'd1] <= WriteData[7:0];
```

-DataMem paraqesin adresat në memorie që janë 8 bitëshe, pra nëse dëshirojmë me shkruajt një të dhënë 16 bitëshe ajo duhet të ruhet në dy rreshta të memories, e njëjta logjikë vlen edhe kur shoqërojmë daljen:

```
assign ReadData[15:8] = dataMem[Address + 16'd0];
assign ReadData[7:0] = dataMem[Address + 16'd1];
```

DataPath

-Pjesa më me rëndësi, kjo pjesë përfshin pjesët përpos njësisë së kontrollit, e register files.



-Si hyrje ka clock-un dhe të gjithë bitat e njësisë së kontrollit, dalje e ka operacionin e dhënë mes të branch dhe zeros së ALU-së.

-Këtu bëhet ndërlidhja e elementeve të krijuara më herët, p.sh. vlera imediate e cila futet tek ALU dhe paraprakisht kalon tek sign extend:

assign Zgjerimi = {{9{instruction[6]}}, instruction[6:0]};

-Apo vlera imediate në rastin e branch e cila shiftohet për 1:

assign shifter2beq = {{8{instruction[6]}}, instruction[6:0], 1'b00};

-Pastaj janë multiplekserët për zgjedhjen e read data 2, pastaj për zgjedhjen se cka shkruhet në regjistër.

-Pjesa e fundit e cila paraqet ndërlidhjen mes të gjitha elementeve.

-Clock i vetmi input.

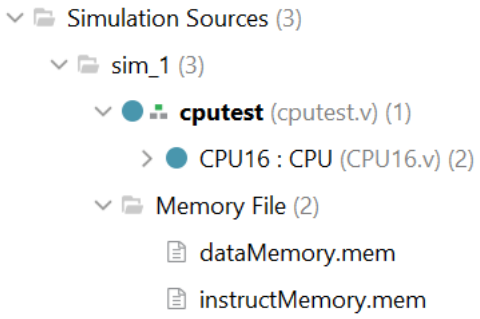
assign pc_initial = 16'd10;

-Fillimisht vendosim vlerën e PC-së në 10, ngase adresat prej 0-9 të rezervuara.

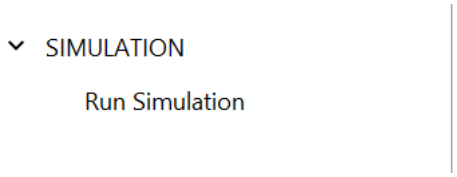
-I bartim parametrat tek instruction memory, datapath dhe UI control, me këto pjesë kompletohet pra CPU.

Ekzekutimi

-Për ta ekzekutuar programin, klikojmë në folder-in Simulation sources:



Pastaj:



Dhe rezultati: