Universiteti i Prishtinës "Hasan Prishtina"

Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike

CPU 16-bit

Studenti	Erlis Lushtaku Enes Hasani Donat Sinani	**Data e ushtrimit**	
Numri i ID-së	190714100055 190714100148 190714100149	**Data e dorëzimit**	26.05.2021
Grupi	14	**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 është bo implementimi i pjesëve përbërëse si:

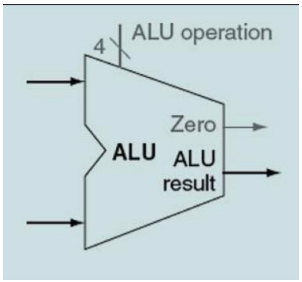
- ALU
- ALU control
- Control unit

- Register file
- Data memory
- Instruction memory

-Pastaj pjesa e fundit vjen ndërlidhja e këtyre elementeve në formimin e datapath-it, control unit-it dhe fetch-it, pastaj lidhja e këtyre 3 pjesëve tjera në formimin tërësishëm të CPU-së.

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

ALU 16-bit



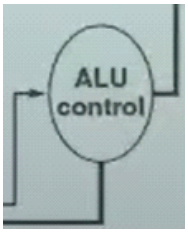
-Përgjegjëse për kryerjen e operacioneve si mbledhje, zbritje, and, or.

-Përkrah numra me gjatësi 16 bitëshe.

-Fillimisht është kriju 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 4 me 1 i cili është përgjegjës për zgjedhjen e veprimit(pra njërit nga veprimet lartë), për inputs edhe outputs do flasim tek pjesa e kodit.

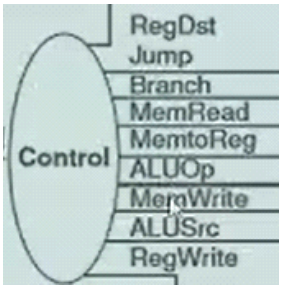
-Përpos kësaj ALU, kemi edhe ALU të cilat kryejnë vetëm mbledhje, siq është rasti i ALU tek pc e cila e inkrementon PC për 2, apo ALU tjetër e cila poashtu kryen mbledhje në rastin kur kemi branching condition, pra në rastin konkret mbledh vlerën imediate e shiftuar majtas me PC+2.

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 lart, disa prej tyre për invertim të njërës hyrje.

Control unit



-Pjesë stateless, pjesë përgjegjëse për thuaja të gjitha veprimeve 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,

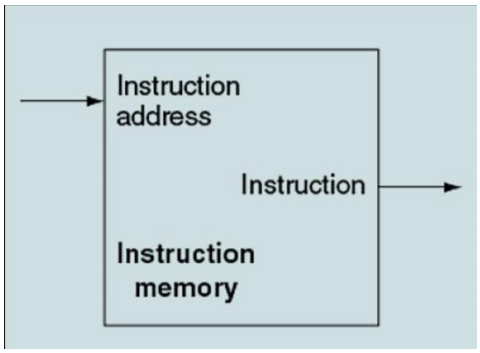
-MemWrite të shkruajmë në memorie,

-Branch për kërcime me 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 të formatit r,

-RegWrite për vlerën 1 lejon me shkrujt në regjistër edhe ALUsrc i cili është bit selektues tek multiplekseri 2 me 1 i cili shërben për të marë 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.

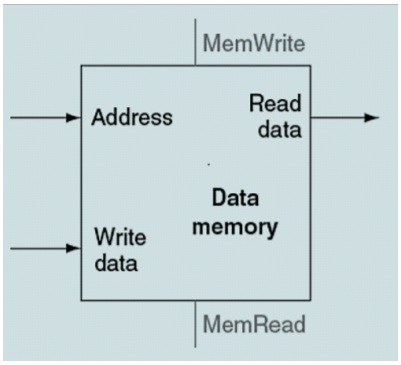
-Instruksionet e formatit r e kanë op code-in e specifikuar si 111(pra op code 3 bitësh), kurse veprimet që kryhen pastaj variojnë në bazë të pjesës së fundit, pjesës së funct e cila është 4 bitësh. Formati i intruksionit të r-formatit:

opcode-111 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).

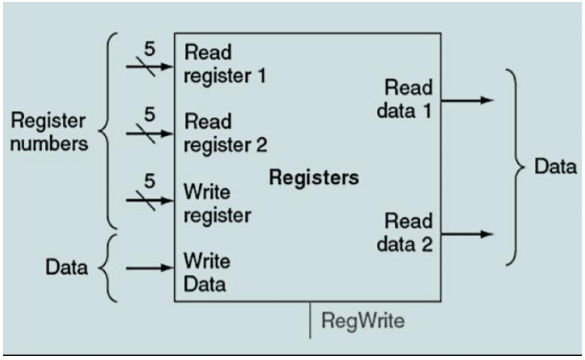
-Instruksionet e i formatit i dallojnë nga ato të r formatit vetëm se nuk kanë funct e as rd, por kanë vlerë imediate 7 bitëshe, dhe dallojnë nga njëra tjetra përmes op code-it.

Data Memory



-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.

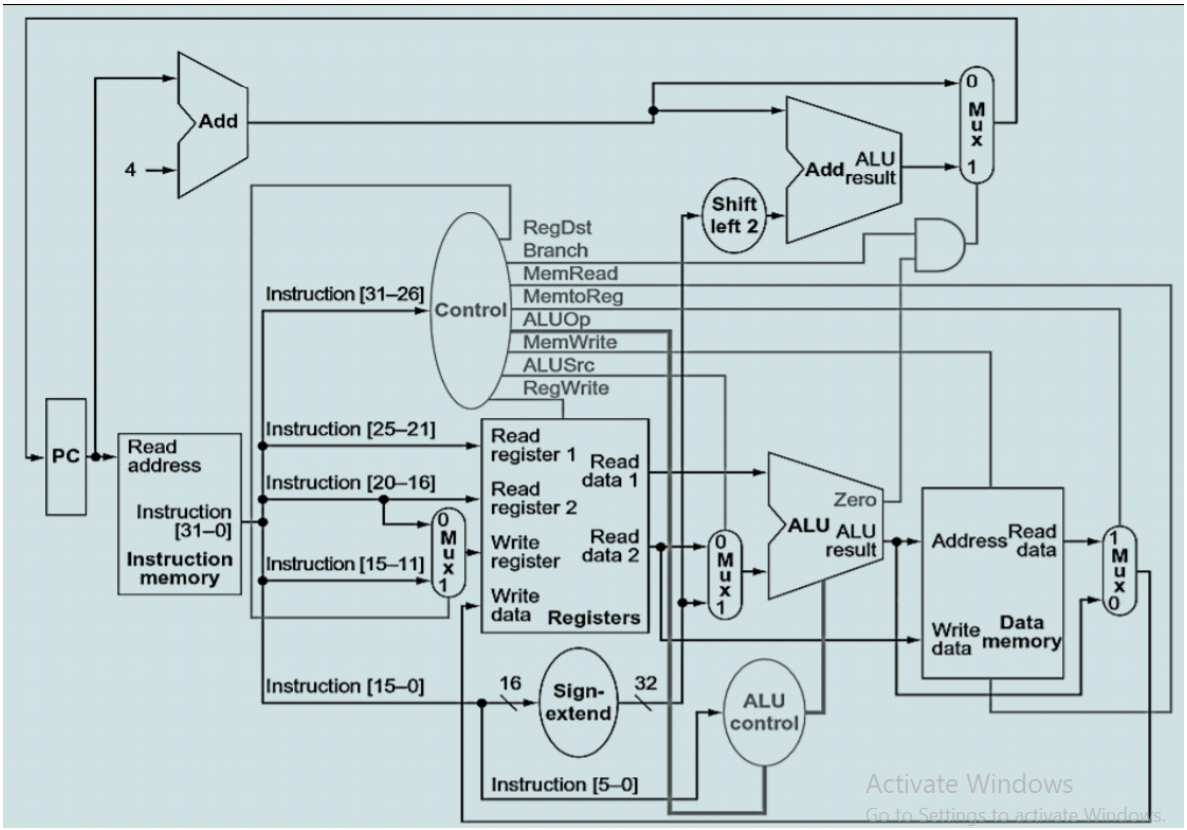
Register file



-Gjithsej 8 regjistra.

-Ka të implementuar dy multiplekserë(8 me 1 pasi kemi 8 regjistra) për zgjedhjen e rs dhe rt-së dhe një dekoder i cili vendos në cilin regjistër me shkrujt, pastaj në atë regjistër vendosen të dhënat që vijnë në writedata.

-Pamja totale e procesorit edhe me nënelementet tjera:



Dizajni

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

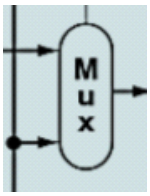
-Para spjegimit të moduleve është me rëndësi koncepti i modulit:

-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:

mux 2ne1



-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.

mux4ne1

-Logjika e njëjtë si mux2ne1, vetëm se këtu kemi 4 hyrje 1 bitëshe dhe një dalje 1 bitëshe, kurse biti selektues është 2 bitësh pasi me dy bita formojmë 4 permutacione, pra aq sa është e nevojshme per mux4në1.

-Gjendet brenda ALU-së, për me zgjedh operacionin.

mux8ne1

-Ka 3 bita selektues, 3 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] ? Hyrja0 : (S[2] ? Hyrja2 : Hyrja1)) : (Hyrja0);
```

Mbledhesi_1bit

-Mbledhësi i plotë 1 bitësh mer 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

-Mer si hyrje 3 bita të opcode, që i mer 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

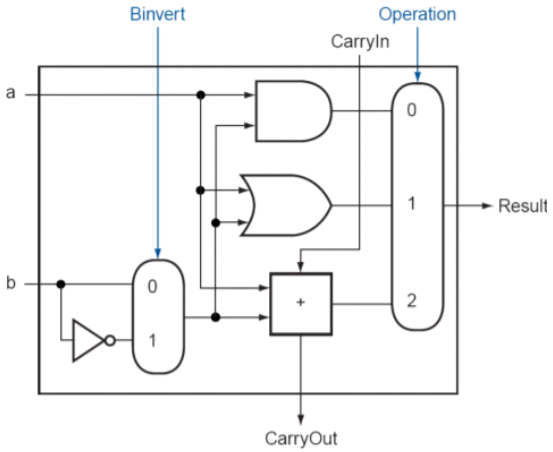
-Mer si inpute bitat për operacionin e ALU-së(2 bita) që i mer nga njësia kontrolluese, pastaj inputi tjetër janë 4 bitat e funct të cilët vijnë nga instruction memory(këto shërbejnë vetëm nëse po flasim për operacione të r formatit).

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

-Kemi bo 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 beq, kur është 10 e specifikojmë pastaj sipas funksionit(pasi që është r format), dhe kur ALUop është 11 e kemi bo për slli.

-Në qdo kushtëzim të përmendur më lartë, ne bëjmë assign në dalje(ia bashkangjesim daljes bitat e caktuar), psh për rastin e parë 00, pasi që lw dhe sw janë i format pjesa e funct nuk na intereson, e dijmë se kemi të bëjmë me add, nuk e invertojmë daljen e dytë dhe dy bitat e multiplekserit janë përkatës për mbledhje, pra bitat total në dalje për rastin 1 janë:010. Në mënyrë analoge përcaktohen edhe për tjerat.

ALU_1bit



-Hyrjet 1 bitëshe A, B dhe CIN, tek multiplekseri është 4 me 1, edhe veprimi xor i shtuar, operation e mer nga ALU control, e poashtu edhe për invertim të hyrjes B.

Veprimet që kryen dhe shoqërimi i daljeve:

assign Dhe = A & mB;

assign Ose = A | mB;

assign XOR = ~Dhe & Ose;

Mbledhesi_1bit Adder(A, mB, CIN, Mbledhesi, COUT);

ALU_16bit

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

-Mer 2 hyrje A, B 16 bitëshe, edhe për secilin bit e lidh nga një ALU-1 bit, psh:

ALU_1bit ALU0(A[0], B[0], BNegate, BNegate, ALUOp, Result[0], COUT[0]);

ALU_1bit ALU1(A[1], B[1], COUT[0], BNegate, ALUOp, Result[1], COUT[1]);

-Vazhdon deri tek 16.

-Ka një dalje zero 1 bitëshe që shërben me tregu nëse dy inputa janë të barabartë ose jo duke i zbritur, e kjo ndihmon tek beq për kërcim.

ALU_with_extra_OP

-Enkapsulim i ALU-së 16 bitëshe, kryen funksione ekstra si mul, ror, slli.

-Të njëjtat hyrje, vetëm se kemi një dalje ekstra për shumëzim e cila është 32 bitëshe:

output [31:0] MulResult

-Kodi:

Mul MULTIPLICATOR(A,B,MulResult);

ROR _ror(A,B,Ror_);

SLLI _slli(A,B,Slli_);

mux8ne1 MUX(Sum,Ror_,Slli_,ALUOp,Result);

-Mux8ne1 ngase nuk ka mjaftu mux4ne1 për krejt rastet.

Mul

-Kryen shumëzim, dy hyrje 16 bitëshe, dalja mundet me qenë deri 32 bitëshe, kodi:

assign Mul = A*B;

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 $readmemh("instrMemory.mem", iMem);

assign Instruction = {iMem[PC], iMem[PC+1]};

-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 shkrujt në memorie.

-Si output kemi readData 16 bitëshe, e cila futet në mux me adresën e ALU-së, në rast të sw apo lw, për me u shkrujt 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 shkrujt të dhëna, andaj duhet që ta ndërrojmë edhe përmbajtjen e fajllit, jo vetëm variablës të deklaruar brenda verilogut, pra:
$writememh("datamemory.mem", sinlgeAdresses);

singleAdresses[Adresa] <= WriteData[15:8];

singleAdresses[Adresa+1] <= WriteData[7:0];

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

assign ReadData = {singleAdresses[Adresa],singleAdresses[Adresa+1]};

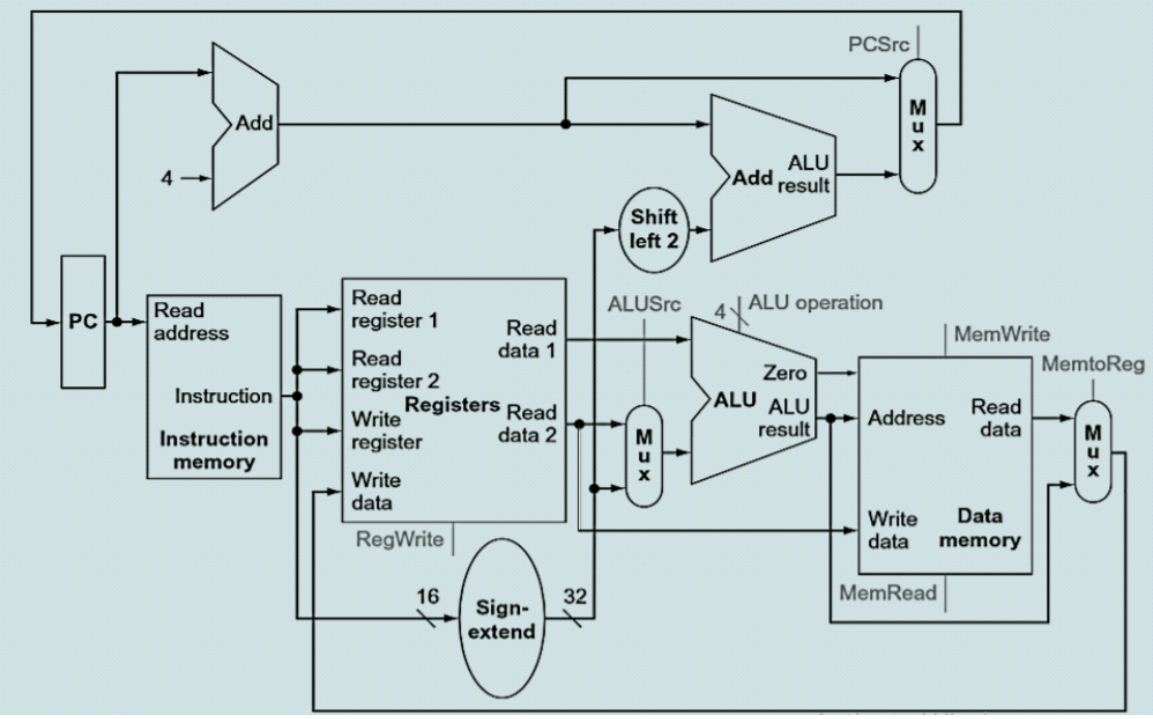
16bit_Adder

-Shërben tek pjesa e inkrementimit të PC-së për 2 e poashtu edhe për rastin kur mbledhim PC+2 me vlerën imediate të shiftuar për 2 në rastin e beq.

DataPath

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

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



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

assign Immediate16 = {{9{Immediate[6]}}, Immediate[6:0]};

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

assign ImmediateShifted = {Immediate16[13:0], 2'b00};

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

-Një adder 16 bitësh i cili na shton vlerën e PC-së:

Adder_16bit(PC, 16'd2, PC, PcCarryOut);

-Multiplekseri tek ALU para PC i cili zgjedh vlerën e cila do futet në PC, varësisht se a kemi branch apo jo:

mux2ne1_16bit(PC, ImmediateShiftedPlusPC, Branch & Zero, BEQPC);

CPU

-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.

-Poashtu PC ia shoqërojmë BEQPC, edhe pse në dukje të parë sikurse i shoqërojmë adresën vetëm për beq, por kjo validohet tek datapath:

assign PC = BEQPC;

Ekzekutimi