|  | **Universiteti i Prishtinës**  Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike  Arkitektura e Kompjuterëve, 2023/2024  Prof. Valon Raça & Synim Selimi | Aulona Livoreka, 220756100039  Artina Qorrolli, 220756100022  Art Kelmendi, 220758100017  Fatime Sadiku, 220756100023  Blerton Ismaili, 220757100075 |
| --- | --- | --- |

# Detyra e dytë: Dizajnimi i një CPU 16-bitëshe (Single-Cycle)

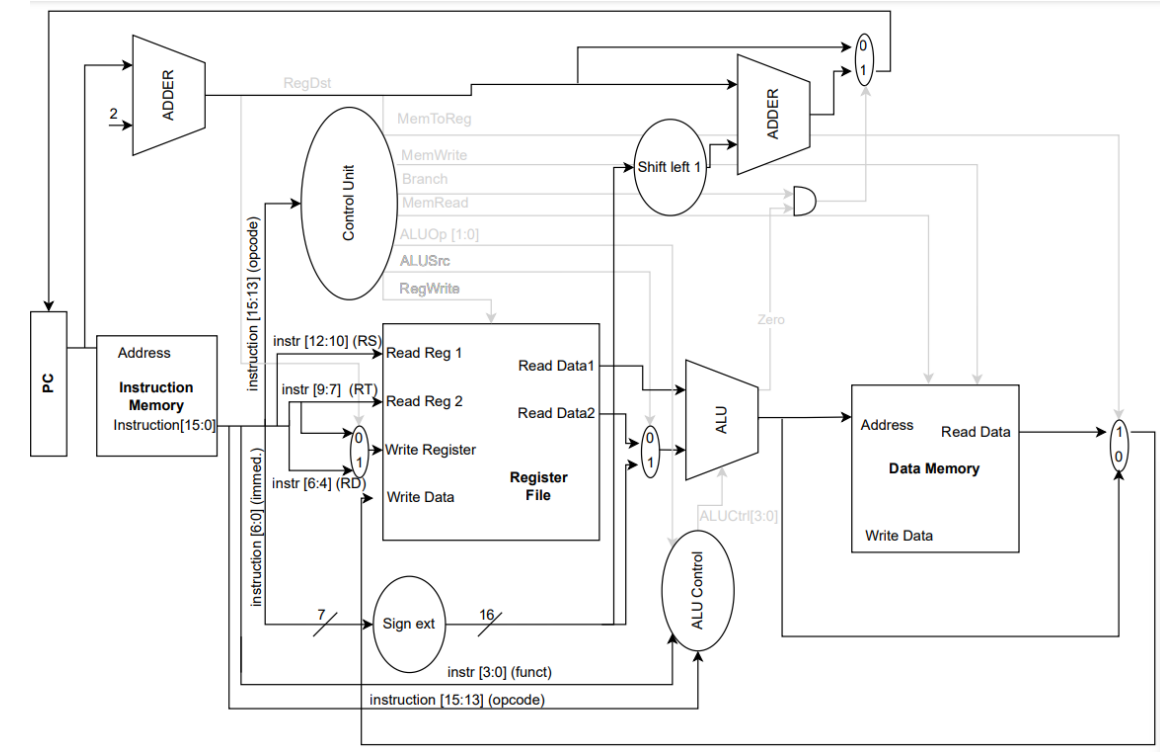
## 1. Hyrje

Kjo detyrë kërkon dizajnimin e një CPU 16-bitësh, i cili është ndërtuar si një sistemi single-cycle dhe mbart operacione të thjeshta aritmetikore dhe logjike. Procesori është projektuar për të përkrahur operacionet e definuara nga instruksionet në formatin R dhe disa nga formati I. Kjo punë është një përpjekje për të krijuar një procesor 16-bit të krahasueshëm me një procesor 32-bit, duke bërë disa modifikime të nevojshme për të përshtatur arkitekturën për një kapacitet më të vogël të të dhënave.

Procesi i dizajnit ka filluar me përdorimin e Verilog Hdl për të realizuar CPU-në, dhe më pas me simulimin e saj përmes mjedisit të Vivados. Gjatë këtij procesi, u morën parasysh elementë të ndryshëm siç janë ALU (Arithmetic Logic Unit), ALU control, Control unit, Register file, Data memory dhe Instruction memory. Për secilin prej këtyre komponentëve u zhvilluan modulet përkatëse, duke u përqendruar në krijimin e një strukture koherente dhe efikase.

Në kuadër të këtij projekti, u kushtua vëmendje e veçantë ndaj mënyrës së ndërlidhjes së këtyre komponentëve. Fillimisht, u ndërtuan pjesët përbërëse të CPU-së, dhe më pas u vijua me lidhjen e tyre për të formuar një datapath të plotë, një njësi kontrolli dhe një mekanizëm për fetch-in e të dhënave. Ky proces i ndërlidhjes ishte thelbësor për të siguruar që CPU-ja të funksionojë si një sistem i integruar dhe i qëndrueshëm.

Në këtë CPU 16-bit, kuptohet se regjistrat, instruksionet dhe adresat janë gjithashtu 16-bitëshe. Kjo bën të mundur një nivel më të lartë të optimizimit dhe efikasitetit, duke pasur parasysh që arkitektura e saj është më e thjeshtë krahasuar me një procesor 32-bit. Kjo detyrë ka qenë një sfidë në të kuptuarit e mënyrës se si elementet e ndryshme të një CPU mund të ndërveprojnë për të prodhuar një sistem të plotë dhe funksional.



**1.Figura e CPU-së 16-bitëshe**

## 2. Dizajni

Fajllat e projektit tonë janë organizuar në mënyrë sistematike në tre file kryesore për të lehtësuar menaxhimin dhe aksesin e tyre. File i parë, i quajtur “Design Sources”, përmban të gjitha modulet themelore të projektit tonë, të cilat janë krijuar në formatin e ekstensionit .v. Këtu gjenden file-t kyç që përbëjnë strukturën bazë të CPU-së dhe pjesëve të tjera të sistemit.

File i dytë, i quajtur “Memory, është dedikuar për ruajtjen e fajllave të jashtëm që kanë të bëjnë me DataMemory dhe InstructionMemory. Këto dy komponentë janë thelbësore për funksionimin e procesorit dhe secili prej tyre ka kapacitetin për të ruajtur 128 madhësi të të dhënash. Ky file është i rëndësishëm për të mbajtur të organizuar dhe të qartë të gjitha të dhënat që përdoren nga CPU për të kryer funksionet e tij.

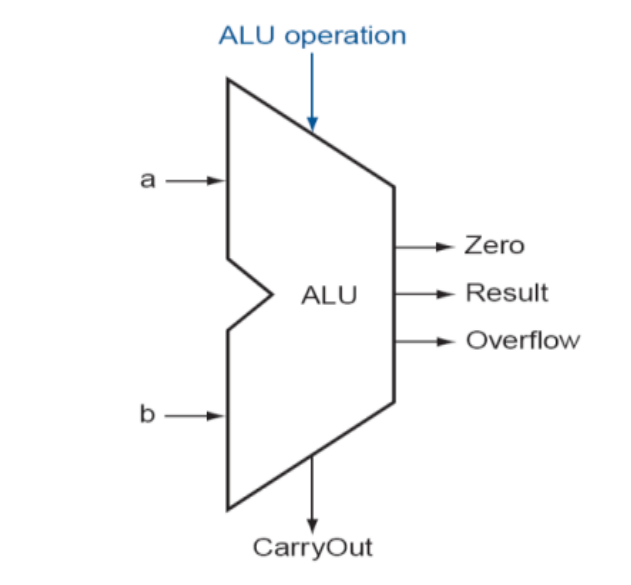
E fundit, por jo më pak e rëndësishme, është file “Simulation Sources”. Kjo pjesë e strukturës sonë të skedarëve është përkushtuar për testimin dhe verifikimin e moduleve të ndryshme dhe të pjesëve të CPU. Në këtë direktori, ne kryejmë simulime të ndryshme për të garantuar se çdo komponent funksionon siç pritet dhe për të identifikuar dhe zgjidhur çdo problem para se të integrohen në CPU. Kjo fazë e testimis është thelbësore për të siguruar që procesori ynë të jetë i besueshëm dhe efikas në punën e tij.

**ALU 16-bit**

Në qendër të një CPU qëndron ALU (Arithmetic Logic Unit), një komponent kyç që realizon veprimet aritmetike dhe logjike mbi operandët në përputhje me udhëzimet e dhëna nga programi kompjuterik. Kjo ALU mbështet përpunimin e numrave 16-bitësh. Duke filluar me një ALU 1-bitëshe, u zhvillua më tej një ALU 16-bitëshe duke lidhur 16 njësi të tilla ALU 1-bitëshe përmes metodës së quajtur RippleCarry. Ky bashkim mundëson kryerjen e operacioneve të ndryshme si mbledhja, zbritja, AND dhe OR.

Brenda ALU-së 16-bitëshe, gjendet një multiplekser 4 në 1, i cili është përgjegjës për zgjedhjen e operacionit specifik që do të kryhet. Detajet specifike të inputeve dhe outputeve do të shqyrtohen në pjesën e kodimit.

Përveç kësaj, ekzistojnë edhe ALU të tjera të specializuara, si për shembull një ALU që përdoret në PC (Program Counter) për të inkrementuar vlerën e PC-së me 2, ose një tjetër ALU që kryen mbledhje në raste të caktuara si kushtet e degëzimit, ku konkretisht ajo shton vlerën e menjëhershme të shiftuar majtas me PC+2.

****

**2.Figura e ALU**

**ALU 1-bitshe**

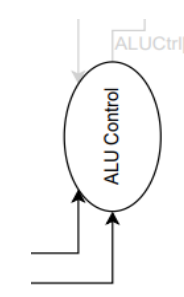
ALU-ja 1 bit ka pesë hyrje të cilat janë: A, B, Cin, Bnegate dhe një hyrje 3-bitëshe për zgjedhjen e operacionit, si dhe dy dalje që janë Rezultati dhe Cout. Në këtë ALU 1 bit realizohen pesë operacione të ndryshme si AND, OR, XOR, ADD, dhe SUB, si dhe operacioni i shtimit me një numër imediat (ADDI). Operacionet AND, OR dhe XOR janë të implementuara si operacione të gatshme, ndërsa për realizimin e ADD, SUB dhe ADDI përdoret një Adder i mësuar në ligjeratat dhe ushtrimet. Për realizimin e operacionit të zbritjes, ne përdorim një multiplekser që, kur është aktivizuar në 1, konverton vlerën në një të kundërt. Një multiplekser 8 në 1, i marrë nga libri i ushtrimeve, na ndihmon në zgjedhjen e operacioneve në bazë të tre bitëve të caktuar. Pozicionet që nuk janë të përcaktuara plotësisht janë mbushur me vlera të paracaktuara të a-s.

Në procesin e dizajnimit të ALU-së 1 bitëshe, ne kemi përfshirë operacione themelore si AND, ADD, OR, XOR dhe SUB, si dhe module të ndryshme të multiplekserëve, përfshirë multiplekserët 8 në 1 dhe 4 në 1.. Çdo modul, si XOR, është ndërtuar në mënyrë strukturale për të kryer operacionet logjike midis dy numrave 1 bitësh, ndërsa moduli si OR dhe AND, i ndërtuar në mënyrë behavioriste, kryen operacione mes dy numrave 1 bitësh.

Moduli ADD është i krijuar në mënyrë strukturale dhe bën operacionin e mbledhjes midis dy numrave 1 bitësh. Me anë të ADD kryhet edhe zbritja. Moduli AND është i krijuar në mënyrë strukturale dhe bën operacionin logjik AND midis dy numrave 1 bitësh.

**ALU Control**

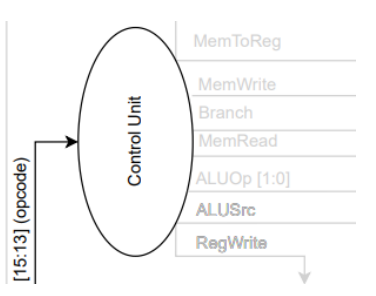
ALU control është përgjegjëse që ALU-së 16 bitëshe të cilët e specifikojnë veprimin që do të ekzekutohet në ALU. Pra tek ALUControl në hyrje kemi 2 bit nga fusha FUNCT e instruksionit dhe 2 bit nga ALUOp . ALU control është përgjegjës që ALU-së 16 bitëshe me i dërgu bita të cilët e specifikojnë veprimin 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.



**3.Figura e ALU Control**

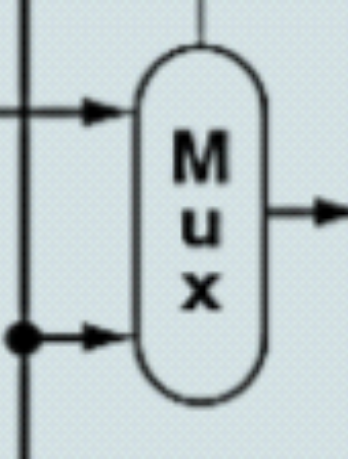
**Control Unit**

Pjesa e Control Unit në CPU është thelbësore për të menaxhuar dhe udhëzuar veprimet brenda procesorit. Ajo përbëhet kryesisht nga fusha me një bit, ka një hyrje 3-bitëshe OPCOD përveç ALUop që ka 2 bit, ku çdo bit ka një funksion specifik. Për shembull, MemRead me vlerën 1 lejon leximin në memorie, ndërsa MemWrite me vlerën 1 lejon shkrimin në memorie. Branch është përdorur për kërcime të kontrolluara nga beq, dhe MemtoReg funksionon si një bit selektues në multiplekserin 2 me 1, që përcakton nëse do të merret me lw apo ndonjë operacion tjetër të formatit R. RegWrite, kur është 1, lejon shkrimin në regjistër dhe ALUSrc është një bit selektues në një multiplekser 2 me 1 që zgjedh nëse do të përdoret vlera immediate apo read data 2, në varësi të llojit të instruksionit.

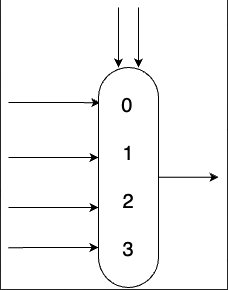


**4.Figura e Control Unit**

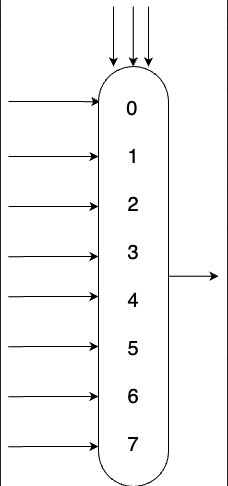
**Multiplekserët**

**Multiplekserët e përdorur në projekt:** 

**Mux 2ne1:** Ky modul është një projektim i thjeshtë që përfshin dy hyrje dhe një selektues, të gjitha me një bit. Moduli gjeneron një output që varet nga vlera e bitit të selektuesit. Nëse biti selektues është zero, moduli zgjedh dhe përcjell si output hyrjen e parë në të kundërtën përcjell hyrjen e dytë si output.

**Mux 4ne1:** Logjika e njëjtë si mux2ne1, vetëm se këtu kemi 4 hyrje 1 bitëshe dhe një dalje 16 bitëshe, kurse biti selektues është 2 bitësh pasi me tre bita formojmë 4 permutacione, pra aq sa është e nevojshme per mux4në1. -Gjendet brenda ALU-së, për me zgjedh operacionin. 

**Mux 8ne1:** Ka 3 bita selektues, 3 hyrje të cilat janë 1 bitëshe dhe një dalje 1 bitëshe, logjika e selektimit e njëjtë, daljes i shoqërojmë: assign Dalja = S[2] ? (S[1] ? (S[0] ? A7 : A6) : (S[0] ? A5 : A4)) : (S[1] ? (S[0] ? A3 : A2) : (S[0] ? A1 : A0));

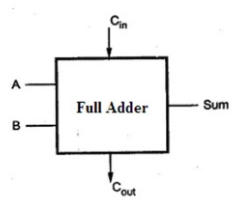


Koncepti i përshkruar është i ngjashëm me mux2ne1, por në këtë rast ne kemi një multiplexor 4 në 1 me katër hyrje 1-bitëshe dhe një dalje 1-bitëshe. Biti i selektimit këtu është 2-bitësh, pasi dy bita mundësojnë krijimin e katër permutacioneve, e cila përputhet me numrin e nevojshëm për një mux4ne1. Këtë mekanizëm e përdorim në ALU\_extra për të mundësuar thirrjen e bonuseve, veçanërisht SLL , SRA , dhe SLTI .

**5.Figura për Multiplekserët e përdorur**

**MBLEDHËSI**

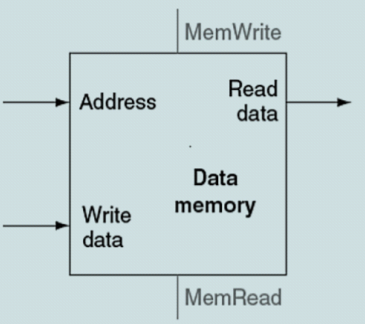
Ky modul është një realizim në Verilog i një mbledhësi 1-bitësh me hyrje dhe dalje. Kemi dy hyrje një bitëshe A dhe B që do të mbledhen, kemi dhe bitin e mbartjes hyrës CIN (Carry IN) që përdoret për të marë një bit mbartje nga një mbledhës i mëparshëm. Marrim rezultatin e mbledhjes së bitëve A dhe B dhe CIN. Në rast se mbledhja rezulton në një bit mbartës , COUT përdoret për të dërguar një bit mbartje te mbledhësi pasardhës, nëse ka një.



**6.Figura për Mbledhesin**

**ALU\_extra**

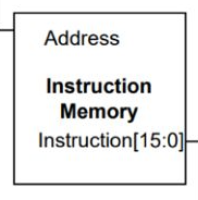
Enkapsulim i ALU\_extra 16 bitëshe, kryen funksione ekstra si SLTI, SLL, SRA si dhe ALU\_16. Të njëjtat hyrje, vetëm se kemi një dalje e cila është Result.

**DATA MEMORY** 

Data Memory në projektin tonë lejon leximin dhe shkrimin në fajllin e jashtëm "dataMemory.mem". Ky komponent ka si hyrje një adresë dhe të dhëna për shkrimin (WriteData), të dyja 16-bitëshe, dhe snjë sinjal (Clock), së bashku me dy sinjale kontrolluese një-bitëshe për leximin (MemRead) dhe shkrimin (MemWrite) në memorie, si dhe daljen e të dhënave të lexuara (ReadData). Clock dhe sinjalet MemRead dhe MemWrite vijnë nga Control Unit.

**7.Figura për Data Memory**

Adresa dhe WriteData, të cilat janë gjithashtu 16-bitëshe, vijnë nga ALU-ja kur duhet të bëhet shkrimi në memorie. Si output, Data Memory prodhon ReadData 16-bitëshe, e cila hyn në një multiplekser së bashku me adresën e ALU-së për operacione si sw apo lw, për t'u shkruar më pas në register file. Data Memory gjithashtu lexon nga një fajll i jashtëm përmes komandës "$readmemh" dhe shkruan në të me "$writememh", duke modifikuar përmbajtjen e fajllit si dhe të variablës së deklaruar brenda Verilogut. Nëse duhet të shkruhet një të dhënë 16-bitëshe, ajo ndahet dhe ruhet në dy rreshta të ndryshme të memories 8-bitëshe. Logjika e njëjtë aplikohet edhe kur merret dalja: ReadData formohet nga bashkimi i dy adresave të njëpasnjëshme në memorie.

**INSTRUCTION MEMORY**

Memoria e instruksioneve në CPU-në tonë është projektuar për të mbajtur një sasi të kufizuar prej deri në 128 bajt të dhënash. Ky kapacitet është i mjaftueshëm për të përmbajtur një varg të gjerë të instruksioneve që CPU-ja mund të përpunojë. Në veçanti, CPU-ja mbështet dy lloje të instruksioneve: ato në formatin I dhe ato në formatin R.

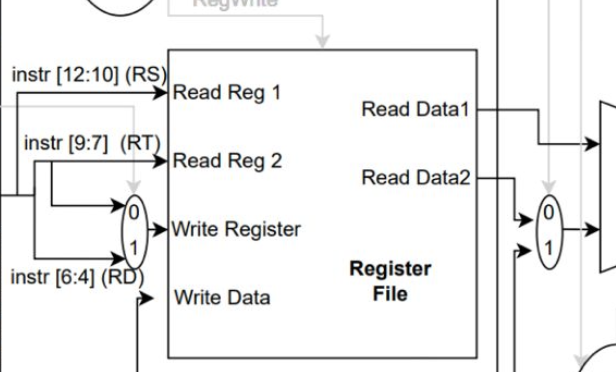
**8.Figura për Instruction Memory**

Memoria e instruksioneve funksionon si një depo e lexueshme vetëm (read-only), e cila ruajtjen e instruksioneve në mënyrë të qëndrueshme dhe të sigurt. Për instruksionet e formatit R, që do të thotë se ato përdorin një opcode 4-bitësh. Më pas, këto instruksione mund të ndryshojnë veprimet e tyre bazuar në pjesën e tyre të fundit, e quajtur 'funct', e cila është 2 bitësh. Struktura e një instruksioni të formatit R përfshin opcode-0XXX, rs-xx, rt-xx, rd-xx dhe funct-xx, ku rs, rt dhe rd janë regjistrat. Duke qenë se ne kemi përcaktuar tri fusha me dy bitë secila për regjistrat, kjo tregon se ne disponojmë gjithsej kater regjistra. Rs paraqet regjistrin e parë, rt të dytin, ndërsa rd është regjistri ku ruhet rezultati (në formatin R gjithmonë kemi shkrim në regjistrin).

Ndërsa, instruksionet në formatin I ndryshojnë nga ato të formatit R në atë se ato nuk përfshijnë 'funct' dhe 'rd', por në vend të tyre kanë një vlerë immediate 7-bitëshe. Këto instruksione gjithashtu ndryshojnë nga njëra-tjetra nëpërmjet opcode-it të tyre të veçantë. Ky dizajn i ndarë midis dy formateve të instruksioneve siguron një fleksibilitet të madh në përpunimin dhe ekzekutimin e urdhrave nga CPU.

**REGISTER FILE**

Për të mbuluar nevojat e instruksioneve të formatit R, ne do të përfshijmë një komponent memorik të njohur si register file në strukturën tonë. Ky komponent është një grupim i regjistrave të cilët janë të aftë për të lexuar dhe shkruar të dhëna. Në total, sistemi përfshin katër regjistra të ndryshëm. Për të menaxhuar zgjedhjen e regjistrave rs dhe rt, ne kemi integruar dy multiplekserë që përcakton në cilin regjistër do të bëhet shkrimi. Të dhënat që duhet të shkruhen në këtë regjistër merren nga linja e writedata. Ky sistem garanton një manaxhim efikas dhe të qartë të të dhënave brenda CPU-së për instruksionet e formatit R.

****

**9.Figura për Register File**

**Datapath**

Kjo pjesë e projektit trajton ndërlidhjen midis ALU, Register File, Data & Instruction Memory, ku si hyrje ka clockun dhe të gjithë bitat e njësisë së kontrollit. Dalja përfshin operacionin dhe lidhjen midis branch dhe zeros të ALU-së. Ndërlidhja kryhet përmes një sërë elementësh dhe procesesh të ndryshme. Për shembull, vlera imediate, e cila hyn në ALU, kalon fillimisht nëpërmjet një procesi të quajtur "sign extend". Gjithashtu, në raste të caktuara si branch, vlera imediate shiftohet për një pozicion. Në projekt janë përfshirë multiplekserë për zgjedhjen e të dhënave që do të lexohen dhe të dhënave që do të shkruhen në regjistrat. Shtohet plus dy për shtimin e vlerës së PC-së dhe multiplekserë të ndryshëm në funksion të nevojave të procesit, si ai në ALU që zgjedh vlerën që do të futet në PC.

Në datapath, janë krijuar lidhje të qarta dhe të rregullta për të gjitha pjesët, bazuar në ushtrimet dhe fotografitë e referencës. Kontrolli, i trajtuar si një element i jashtëm nga data-path, jep outputet që përdoren si inpute. Procesi fillon me vlerën e PC-së, e cila dërgohet në instruction memory për të lexuar vlerën nga adresa e caktuar. Instruction memory jep vlera të ndryshme të opcodes, RS, RT, RD, immediate dhe function, të cilat pastaj përdoren në Register file dhe ALU control. Në Register-File dërgohen vlerat standarde dhe Regjistri RegWrite, i cili tregon nëse do të bëhet shkrimi në të dhënat apo jo, që është zero vetëm në rastin e storeword.

**CPU**

Në strukturën e CPU-së, ne kemi integrime të dy moduleve kyçe: DataPath dhe ControlUnit. Bashkimi i tyre sjell në funksion një CPU të plotë, e cila funksionon në mënyrën e përshtatshme. Sinjali i orës (Clock) është hyrja e vetme e nevojshme për këtë proces.

Fillimisht, vendosim vlerën fillestare të Program Counter (PC) në 10, për shkak se adresat nga 0 deri në 9 janë të rezervuara. Kjo vlerë fillestare transmetohet në instruction memory, në datapath dhe në ControlUnit, duke kompleksuar kështu strukturën e CPU-së. Në lidhje me PC-në, atij i shoqërojmë BEQPC. Edhe pse në fillim duket sikur i caktojmë adresën vetëm për rastet e BEQ, në të vërtetë, kjo validohet dhe përfundon në datapath, ku PC përcaktohet si: assign PC = BEQPC. Ky proces siguron funksionimin e përshtatshëm dhe të koordinuar të të gjithë komponentëve të CPU-së.

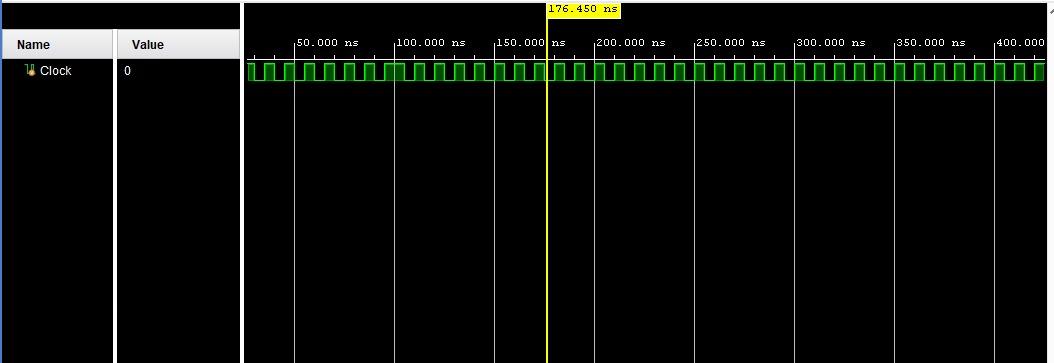
## 3. Ekzekutimi

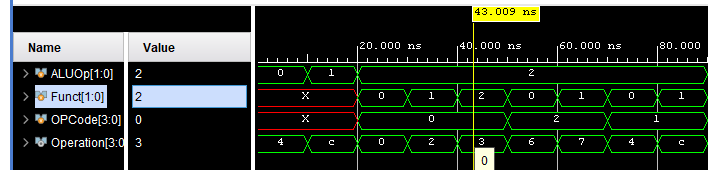
Çdo modul individual në projektin tonë ka testimet e veta të cilat janë të strukturuara dhe të emërtuara në mënyrë të njëjtë si moduli përkatës, me shtesën \_test për të treguar se bëhet fjalë për një 'test bench' brenda folderit të simulimeve. Duke qenë se zhvillimi i projektit u bë në mënyrë sekuenciale, testimet ishin jetike për çdo komponent dhe për kombinimin e tyre. Këto teste na ndihmuan të verifikojmë nëse secila pjesë dhe kombinimi i tyre funksiononin siç duhet. Në fazën e kombinimit të disa komponentëve, u shfaqën ndryshime të shumta në inputet dhe outputet, pasi tani të dhënat po vinin nga module të ndryshme, në krahasim me fazën fillestare të ndërtimit të çdo moduli. Ky proces ishte i domosdoshëm për të siguruar që bashkëveprimi i komponentëve të ndryshëm të funksiononte pa probleme.

**10.Figura e Hierarkise se file-ve për simulim**

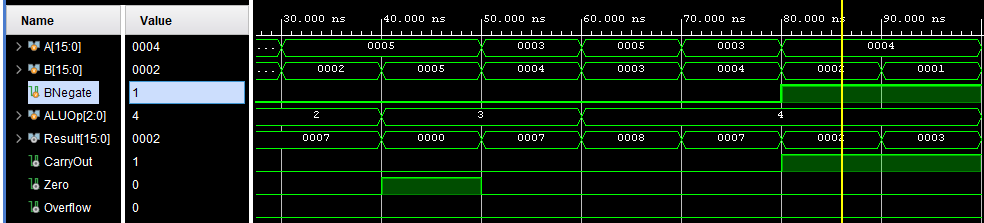
**TESTIMI CPU**

Gjatë testimit, kemi vënë re se Program Counter (PC) rritet në çdo cikël të orës pozitiv, siç kërkohet nga detyra. Për të vlerësuar progresin e programit, kemi verifikuar nëse bitat e daljes janë në përputhje me kushtet e caktuara, bazuar në operacionet e përcaktuara në instruction memory. Rritja e PC-së ndodh me një njësi, pasi instruction memory është strukturuar në atë mënyrë. Në rastin kur një byte është shkruar në një rresht dhe byte tjetër në rreshtin pasardhës, atëherë rritja e PC-së do të ishte e nevojshme të bëhej me dy njësi.

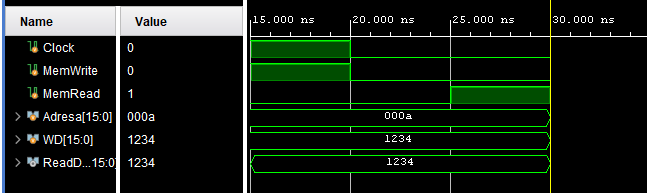


Testimi ALU\_Control

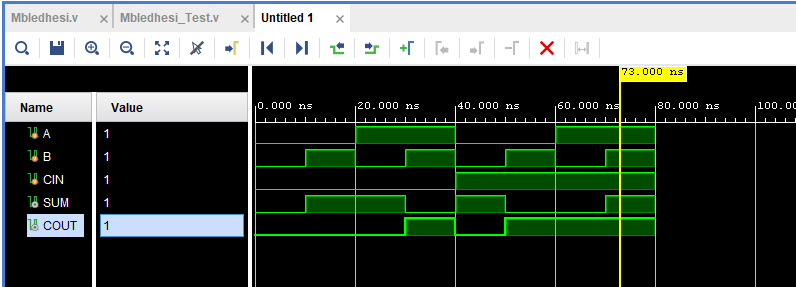
Testimi ALU16bit



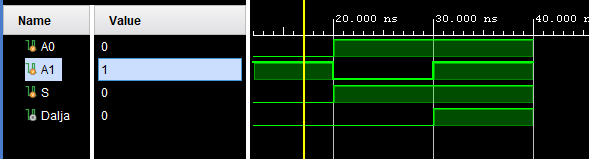
Testimi DataMemory



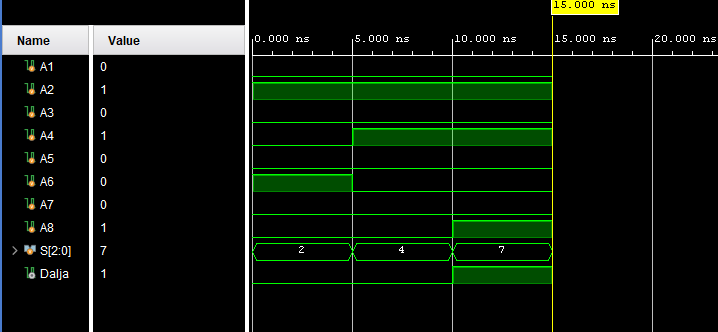
Testimi Mbledhesi



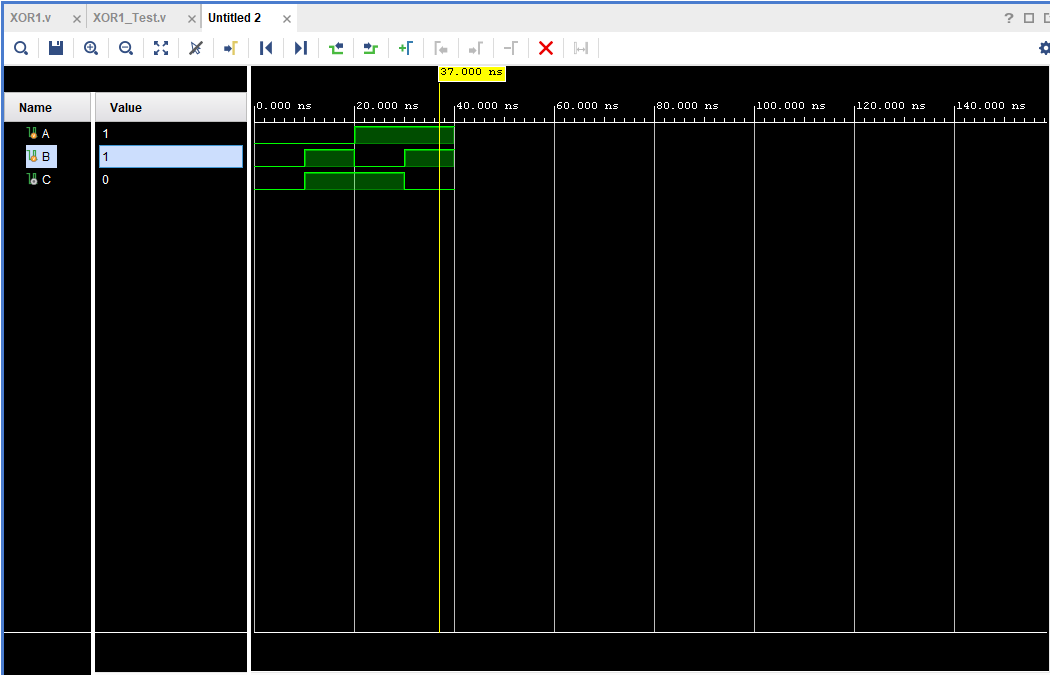
Testimi Mux2ne1



Testimi Mux8ne1



Testimi XOR



**4.Përfundimi**

Ky projekt ka qenë një eksperiencë e thellë dhe e detajuar që na ka ndihmuar të kuptojmë më mirë funksionimin dhe arkitekturën e një CPU-je, pavarësisht se modeli i punuar ishte më i thjeshtë se procesorët modernë. Burimet e ofruara nga profesori, si ushtrimet dhe materialet për një CPU 32-bitësh, ishin thelbësore për të arritur këtë kuptim. Projektin e karakterizon një thellim i njohurive në strukturën e CPU-së dhe mënyrën se si përbërësit e saj ndërveprojnë, si dhe një familjarizim i fortë me gjuhën programuese Verilog dhe mjedisin e simulimit Vivado. Kjo përvojë ka pasuruar njohuritë tona me një fokusim më të madh në aspektet inxhinierike dhe jo vetëm në gjuhët e programimit të larta.