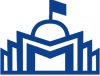
­­­



МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ

ХЭРЭГЛЭЭНИЙ ШИНЖЛЭХ УХААН, ИНЖЕНЕРЧЛЭЛИЙН СУРГУУЛЬ

ЭЛЕКТРОНИК, ХОЛБООНЫ ИНЖЕНЕРЧЛЭЛИЙН ТЭНХИМ

Махбалын Бүрэнзаяа

RISC процессорын HDL гүйцэтгэл

Компьютер зохион байгуулалт бие даалт

Улаанбаатар хот

2024 оны 2 сар

RISC процессорын HDL гүйцэтгэл

*Махбал Бүрэнзаяа*

*Электроник*

*Электроник холбооны инжинерчлэлмйн тэнхим*

*Мэдээллийн Технологи, Электроникийн Сургууль*

*bvrnee1009@gmail.com*

*Хураангуй* — *Энэхүү бие даалтын ажлаар нь MIPS 32 битийн процессорын core дизайныг HDL - ээр гүйцэтгэнэ. Нэг циклт 32 битийн процессорын core дизайныг гаргаж туршиж үзсэн. 5 pipeline stage - тэй болгож ажиллагааг сайжруулан, өгөгдөл болон удирдлагын хязгаарлалтуудыг шийдвэрлэн, гарсан үр дүнг харьцуулан туршилт хийж гүйцэтгэсэн.*

Түлхүүр үг — Reduced Instruction Set Computer (RISC), Microprocessor without Interlocked Pipelined Stages (MIPS), Instruction Set Design (ISD), Verilog.

# **УДИРТГАЛ**

Дижитал систем нь хоорондоо холбогдсон дижитал хэсгүүд болох тоолуур, буфер, логик гейт болон санах элементүүдийн цогц гэж болно. Микропроцессор, микроконтроллёр гэх мэт томоохон дижитал системыг хурдтай, үр ашиг өндөртэй байхаар загварчлах эсвэл өөрчлөлт сайжруулалт хийх нь тооцоолол, судалгаа их шаарддаг.

Орчин үед гар утас, таблет гэх төхөөрөмжүүд хүний амьдралд чухал хэрэгцээт зүйл болтлоо хөгжсөн байна. Энэ хэрэгцээг дагаад өндөр хурдтай, чадал бага зарцуулдаг, хэмжээ багатай гэх шаардлагатай тулгарна. Процессорын архитектур хэдий зэрэг комплекс болно түүнд шаардагдах транзисторын тоо нэмэгдэж улмаар хэмжээ, чадал зарцуулалт ихэсдэг [1]. Ийм учраас ембеддед төхөөрөмжүүдэд RISC архитектур тохиромжтой байдаг.

RISC – “Reduced Instruction Set Computer” нь харьцангуй цөөхөн инструкцтэй байдаг тул “Hardwired” буюу процессорын бүрэлдэхүүн хэсгүүдийг ISD – д зориулан загварчилдаг. Энэ нь инструкцийн гүйцэтгэлийг хурдасгахаас гадна удирлагын логикыг хялбарчилж өгдөг [3]. Энэхүү бие даалтын сэдвийн хүрээнд 32 битын MIPS буюу RISC архитуртай микропроцессорын core ыг HDL – ашиглан хийж гүйцэтгэнэ.

# **32 БИТ MIPS МИКРОПРОЦЕССОРЫН СИСТЕМ АРХИТЕКТУР**

MIPS – “Microprocessor without Interlocked Pipelined Stages” нь RISC архитуртай, pipeline ажиллагаатай микропроцессор юм. MIPS микропроцессор нь дараах үндсэн шинж чанаруудтай.

**Тогттсон инструкцийн хэмжээ** – бүх төрлийн инструкци адил хэмжээтэй байна. Ижил хэмжээтэй

инструкцийг декод хийх нь хэт комплекс дижитал систем шаардалгүйгээр, инстукцыг унших болон задлах хэсгүүдийг логик хэлхээг хялбаршуулдаг. Мөн MIPS микропроцессорт зориулагдсан машин хэлний хөрвүүлэгчын ажиллагааг хурдасгаж өгдөг.

**Регистр файл** – 32 битын MIPS микропроцессор нь нийт 32 ширхэг 32 битын General Purpose Register (GPR) – тэй байдаг. Олон регистертэй архитектур дээр регисетрээс регистер хооронд хандаж, өгөгдлийг боловсруулах боломжтой байдаг. Регистер хооронд хийгдэж буй инструкц нь гүйцэтгэл хурдан байдаг давуу талтай.

**Санах ойд өгөгдөл дамжуулах** “Load-Store Архитектур” – MIPS микропроцессор дээр санах ойд хандахын тулд зөвхөн load, store инстукцыг ашигладаг. Санах ой дахь өгөгдөлд хандах нь хамгийн удаан гүйцэтгэлтэй байдаг тул санах ой дахь өгөгдөлд шууд процесс хийх инструцийг хязгаарлаж өгдөг.

1. **Нэг циклт гүйцэтгэл.**

MIPS микропроцессорын инструкци нь ашиглаж буй дижитал элементүүдээрээ, гүйцэтгэх үүргээрээ ерөнхийдөө гурван төрөлд хуваагддаг.

* Санах ойд хандах - load word (lw), store word (sw)
* Арифметик логик - add, sub, and, or, болон slt гэх мэт
* Удирдлага салаалах - branch equal (beq) болон jump (j).

Инструкцууд нь төрлөөрөө хуваагдаж байгаа ч бүхийл инструкц хийгдэх эхний хоёр алхам адил байна.

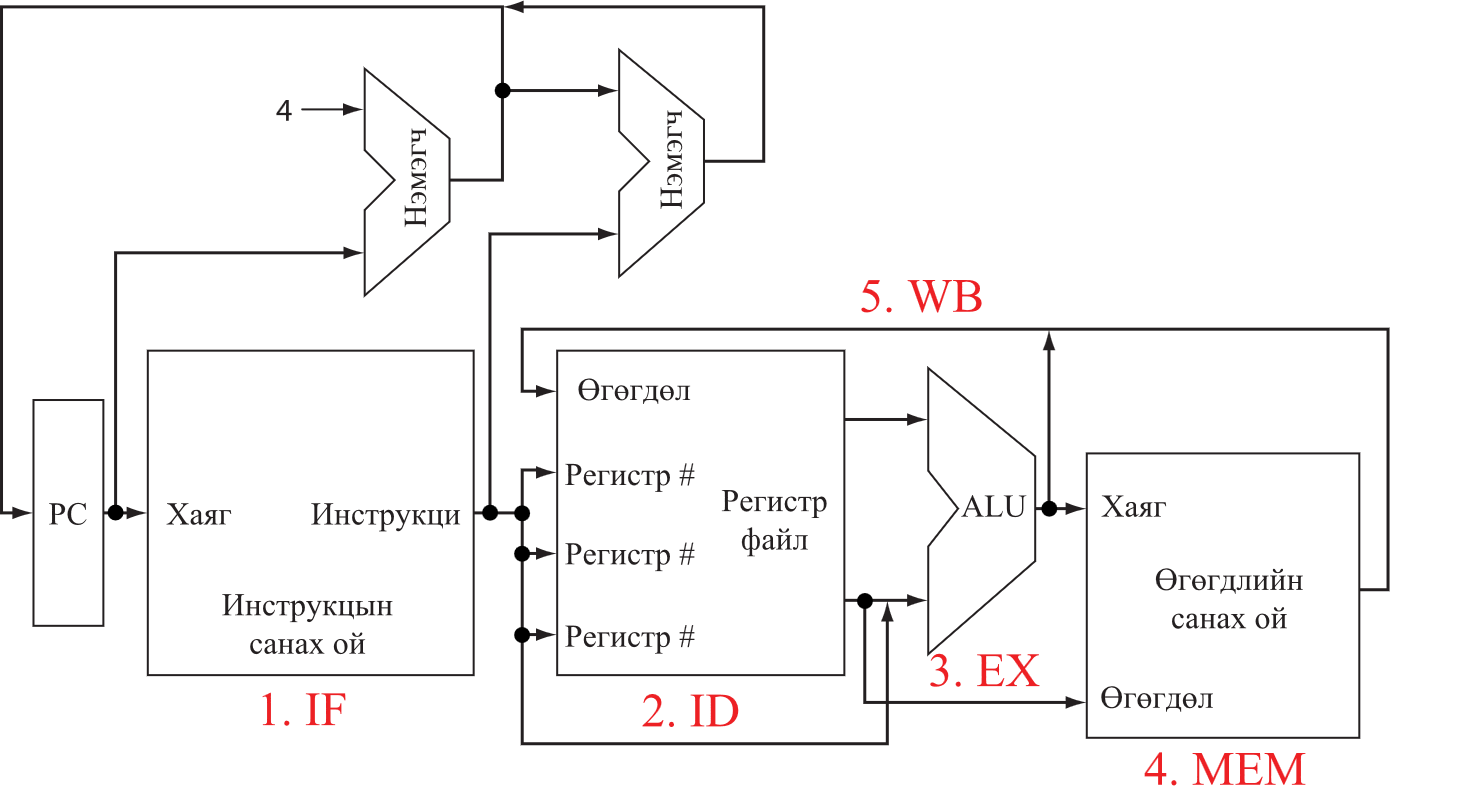
1. Програм тоолуурын “Program Counter (PC)” утгаар инструкцийн санах ой “Instruction Memory” – ын харгалзах үүрнээс инструкцийг уншиж авах.
2. Уншсан инструкцийг задалсаны дараагаар процесс хийгдэх нэг эсвэл хоёр регистрыг уншина.

Энэ хоёр алхамын дараагаар инструкцийн төрлөөс хамаарч үйлдэл хийгдэх хэдий ч jump инструкцаас бусад бүх инструкци arithmetic-logical unit (ALU) – г ашигладаг. Тодорхойлвол

* Санах ойд хандах инсрукцын хувьд санах ойн хаягийг бодох.
* Арифметик логик инсрукцын хувьд арифметик үйлдэл гүйцэтгэх.
* Удирдлага салаалах инсрукцын хувьд жиших үйлдэл хийнэ.

ALU ашигласаны дараагаар инструкцийн төрлөөс хамаарч хийгдэх үйлдэлүүд бүгд ялгаатай байна.

* Санах ойд хандах инсрукцын хувьд бодож гаргасан санах ойн хаягаас өгөгдлийн унших уу, бичих үү эсэх хийгдэнэ.
* Арифметик логик инсрукцын хувьд арифметик үйлдэл үйлдлийн хариуг заасан регисрлүү бичнэ.
* Удирдлага салаалах инсрукцын хувьд PC – ын утгийг өөрчилнө эсвэл 4 өөр нэмэгдсэн утгаар солино.



Зураг 1. MIPS ажиллагааны абстракт дүрслэл

Нэг циклт гүйцэтгэл тул инструкцийг уншиж аваад хариугаа эргүүлэн бичих хүртэл нэг клокын хугацаа зарцуулагдана. Энэ тохоилдолд голлох хэсгүүдйин үр дүнг завсран регистрт хадгалаж авч яваагүй учраас яг ч шатласан ажиллагаатай гэж ойлгож болохгүй. Гэхдээ эцсийн үр дүн адил байна.

**Нэг циклт MIPS микропроцессорын** **Datapath:**

Дээрх зураг нь микропроцессорын өгөгдөл болон хаягийн шугамын хэт хялбарчлан авсан байгаа. Жишээ нь хоёр хаягийн шугам PC шууд холбогдож болохгүй ийм учраас зөв сигналыг сонгох мультиплексор хэрэглэдэг.

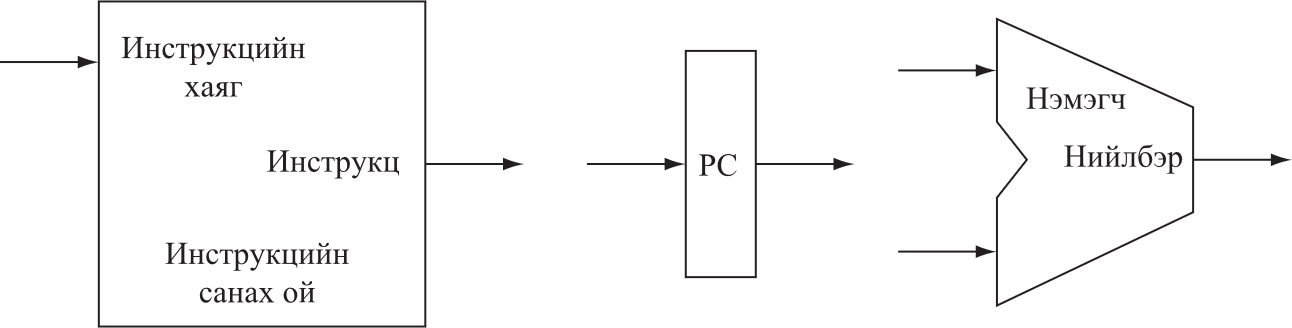
Мөн микропроцессорын элементүүдийн ажиллагаа инструкци тус бүрээс шалтгаалаад өөр байна. Үүнийг зохицуулахын тулд удирдлагын хэсэг “Control Unit” байх зайлшгүй шаардлагатай. Удирдлагын хэсгээс гарах удирдлагын сигнал нь хэрэгцээтэй хэсгүүдийг зөв агшигд идэвхжүүлснээр бүх хэсгүүд зохицож зөв ажиллана.



Зураг 2. Инстуркцийн формат

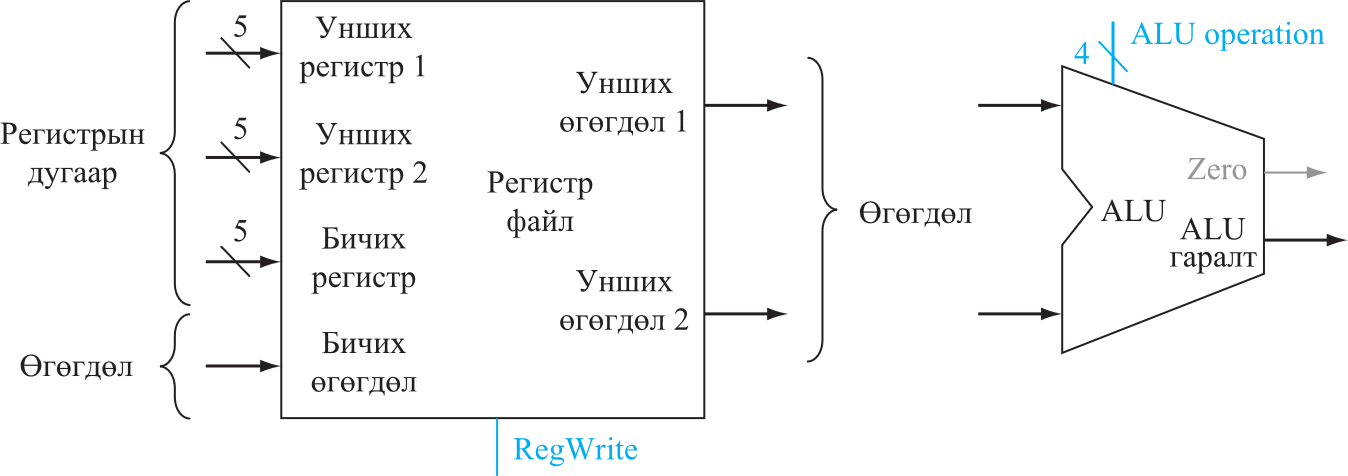
MIPS инструкцын форматаас хамааруулан логик хэлхээний дизайн гарна.

Хамгийн эхний хэсэгт хоёр санах элемент хэрэгтэй. Инструкцын санах ой нь нийт програм файлыг хадгалах бол программ тоолуур нь программ аль хэсэгтээ гүйцэтгэгдэж байгааг хадгална. Мөн программ тоолуурын утгийг нэмэгдүүлж дараагийн инструкцад хандах үүднээс нэг нэмэгч байрлана.



Зураг 3. Instruction memory, Program counter, Adder

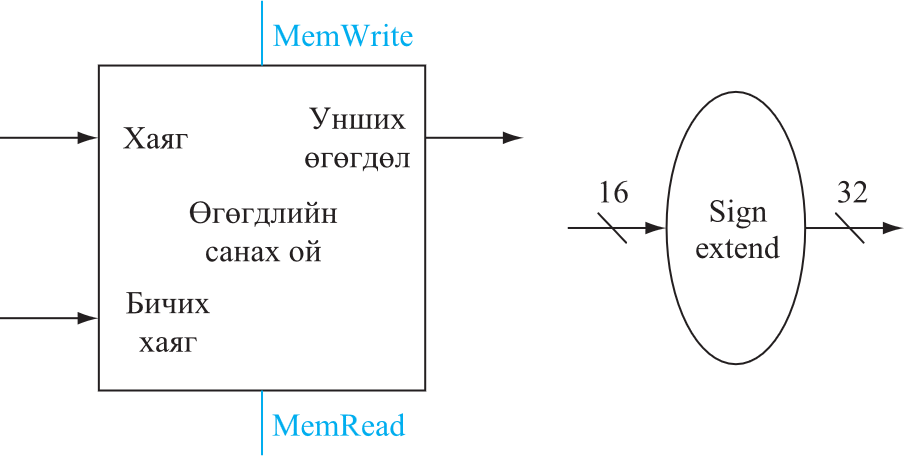
Үүний дараагаар инструкцийг задлаж гүйцэтгэнэ. MIPS инструкци нь регистр хооронд үйлдэл хийдэг. 32 ширхэг 32 битын регистрын бүрэлдэхүүнийг регистр файл гэж нэрлэдэг. 32 буюу 2 – ын таван зэргээр 32 регистрыг ялгаж болох учраас инструкцийн операндууд нь 5 бит байгааг харж болно. Мөн үйлдэл гүйцэтгэх ALU хэрэгтэй.



Зураг 4. Register file, ALU

Регистр файлаас харвал регистрээс өгөгдлийг шууд уншиж RegWrite удирдлагын сигналаар бичих өгөгдөл дээр байгаа утгийг регистрт бичдэг гэсэн үг. Харин 4 битын ALU operation удирдлагын сигналаар ямар инструкци болоод ямар үйлдэл гүйцэтгэхийг удирдана. Мөн ALU нь Zero гаралттай байх ба хоёр оролт тэнцүү эсэхийг заана. Энэ нь удирдлага салаалах инструкцад хэрэглэгддэг.

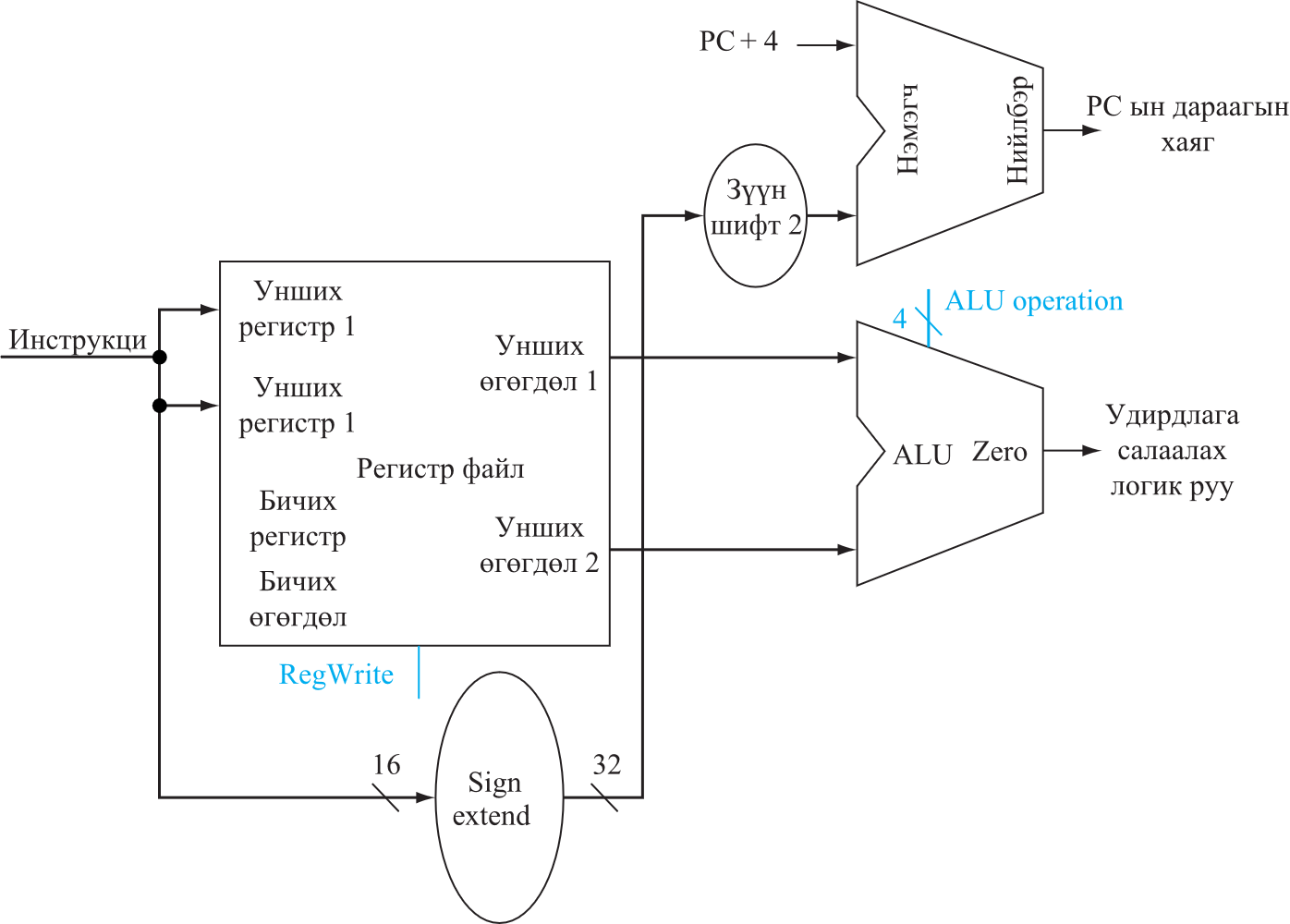
Санах ойгоос унших болон санах ойд өгөгдөл бичих инструкцуудыг хэрэгжүүлэхэд өгөгдлийг хадгалах санах элемент хэрэгтэй. Мөн инструкцын 16 бит хаягийн утгийг 32 бит хүргэх sign extension unit хэрэгтэй.



Зураг 5. Data memory, Sign extension unit

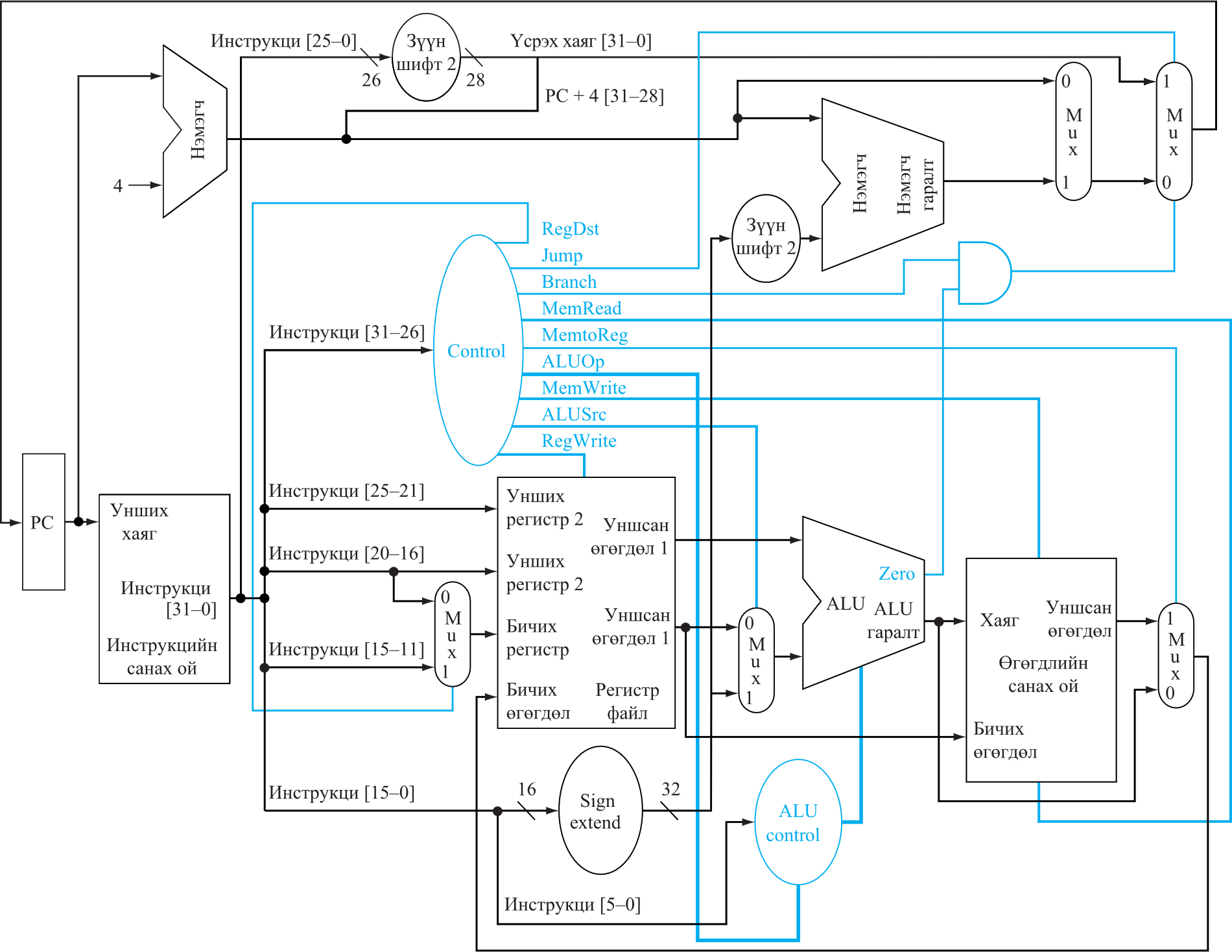
Удирдлага салаалах инструкцийн хувьд sign extend хийсэн хаягийн офсетыг зүүн 2 шифт хийх буюу 4 өөр

үржиж инструкцын хаягийг гаргаж авна. ALU ээс гарах Zero гаралтыг, удирдлагын сигналтай хамтад нь мултьплексорын сонгох оролт болгож өгснөөр программ тоолуурын дараагийн утгийг авна.



Зураг 6. Shift left 2

R болон I төрлийн инструкцийг дээрх хэсгүүдээр гүйцэтгэнэ. Үлдсэн нөхцөлгүй үсрэх Jump инструкцийг хэрэгжүүлэхэд нэг зүүн шифт хийх хэсэг, ахин нэг мултьплексор хэрэгтэй. J төрлийн инсирукци нь үсрэх 26 битын офсеттэй. Үүнийг зүүн 2 шифт буюу 4 өөр үржиж програм тоолуурын ахлах 4 биттэй нийлүүлэн мултьплексорын оролтод өгнө. Мултьплексорын сонгох оролтод нь удирдлагын хэсгээс гарсан J төрлийн инструкцад зориулагдсан Jump удирдлагын сигналыг өгсөн байна. Доорх зурагт нэг циклт MIPS микропроцессорын datapath – ыг харуулав.



Зураг 7. Single cycle MIPS datapath

Нэг циклт гүйцэтгэл нь алдаагүй зөв ажиллах хэдий ч үр ашиг багатай учраас хэрэглээнээс гарсан. Бүх инструкци тогтсон нэг клокд хийгдэж байгаа. Cycles Per Instruction (CPI) нь 1 гэсэн үг. Энэ нь хамгийн удаан хийгдэх load word (lw) инструкцаар тодорхойлогдоно. Учир нь инструкцын санах ой, регистр файл, ALU, өгөгдлийн санах ой, регистр файл гээд нийт 5 удаа микропроцессорын хэсгүүдийг ашигладаг. CPI нь 1 хэдий ч нэг кдокын урт нь удаан учир үр ашиг муутай гүйцэтгэл гэж хэлж болно.

Нэг клокд мултьплексорын хэсгүүд нэг л удаа ажиллах тул хөвөх цэгтэй тоо процесс хийх гэх мэт хэт комплекс инструкцыг хэрэгцүүлэхэд микропроцессорын зарим хэсгүүдээс бүр 2 байх шаардлагатай. Энэ нь эргээд клокын хугацаа мөн микропроцессорын өртөгийг ихэсгэнэ.

Нэг инструкцыг олон богино клокд хийж гүйцэтгэх аргаар “multicycle” энэ асуудлыг шийдэж болно. Цаашлаад пайпайн ажиллагаагаар үр ашгийг сайжруулдаг.

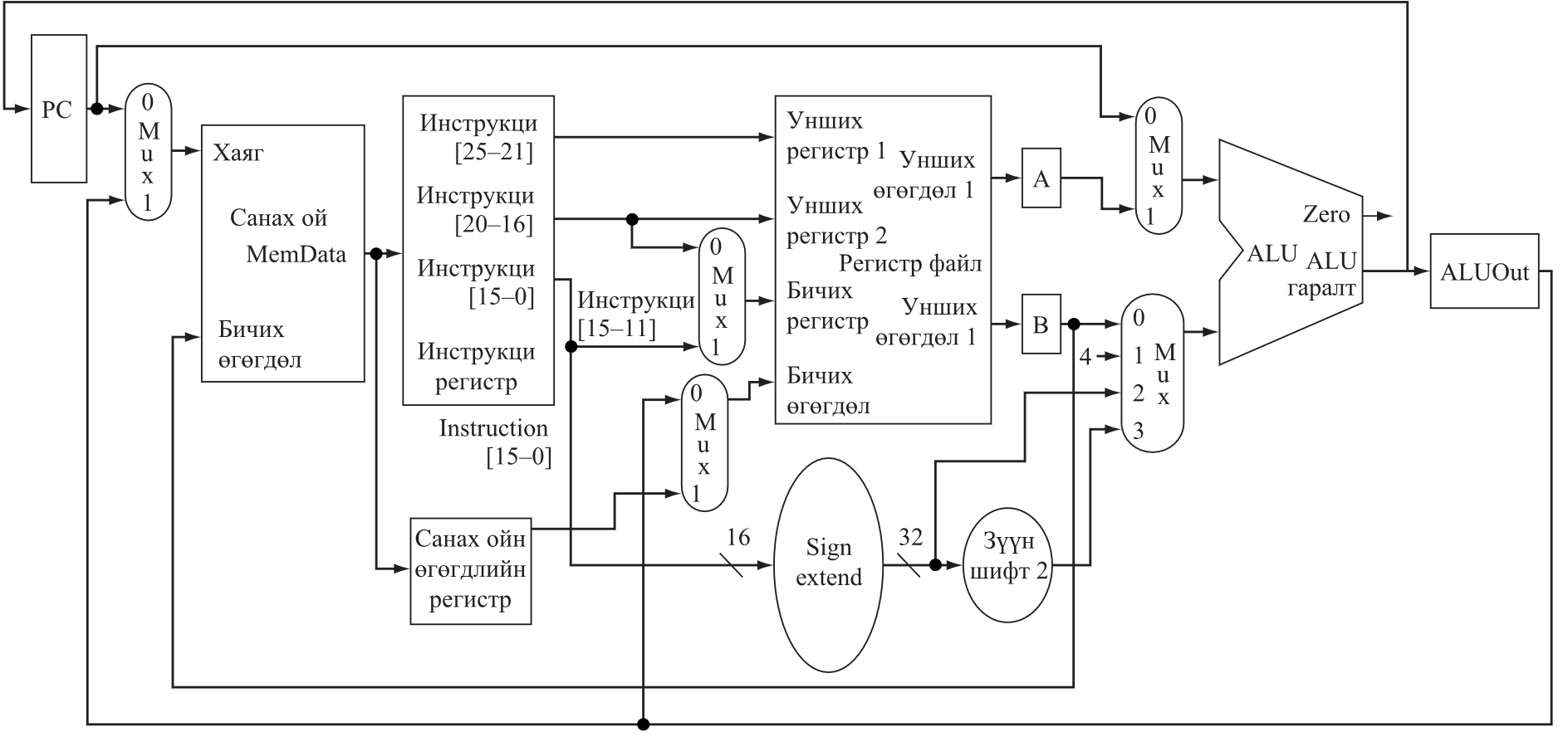
1. **Олон циклт гүйцэтгэл.**

Инструкцийг олон циклд гүйцэтгэхдээ шаталсан алхамуудад хувааж, нэг алхамыг нэг циклт гүйцэтгэнэ. MIPS микропроцессорын инструкцуудын хувьд инструкцийг гүйцэтгэх 5 алхамд хуваадаг. Олон циклт гүйцэтгэл нь микропроцессорын нэг хэсэг нэг инструкцад нэгээс олон утаа ажиллах боломж өгнө. Ингэснээр хэрэгцээт микропроцессорын хэсгүүдийн тоог бууруулхаас гадна инструкцыг ялгаатай клокд гүйцэтгэх боломжтой болно.

Нэг циклт гүйцэтгэлтэй микропроцессорын datapath тай харьцуулвал өгөгдөл болон инструкцийн санах ой нэг хэсэг. ALU болон 2 нэмэгчийн оронд нэг ALU дангаар ажиллана. Мөн хэд хэдэн регистр нэмэгдэх бөгөөд шат бүрийн гаралтыг инструкц хийгдэж дуустал түр хадгалах үүрэгтэй.

* Инструкцийн регистр “Instruction register (IR)” болон өгөгдлийн регистр “Memory data register (MDR)” нь санах ойгоос уншиж авсан өгөгдлийг тус бүр хадгалахад ашиглагдана. Хоёр утга нь нэг клок циклд цуг хэрэглэгдэх тул тусдаа регистрт хадгалагдах хэрэгтэй.
* A болон B регистр нь регистр файлаас гарсан 2 операндын утгийг хадгална.
* ALUOut регистр нь ALU ыг гаралтыг хадгална.

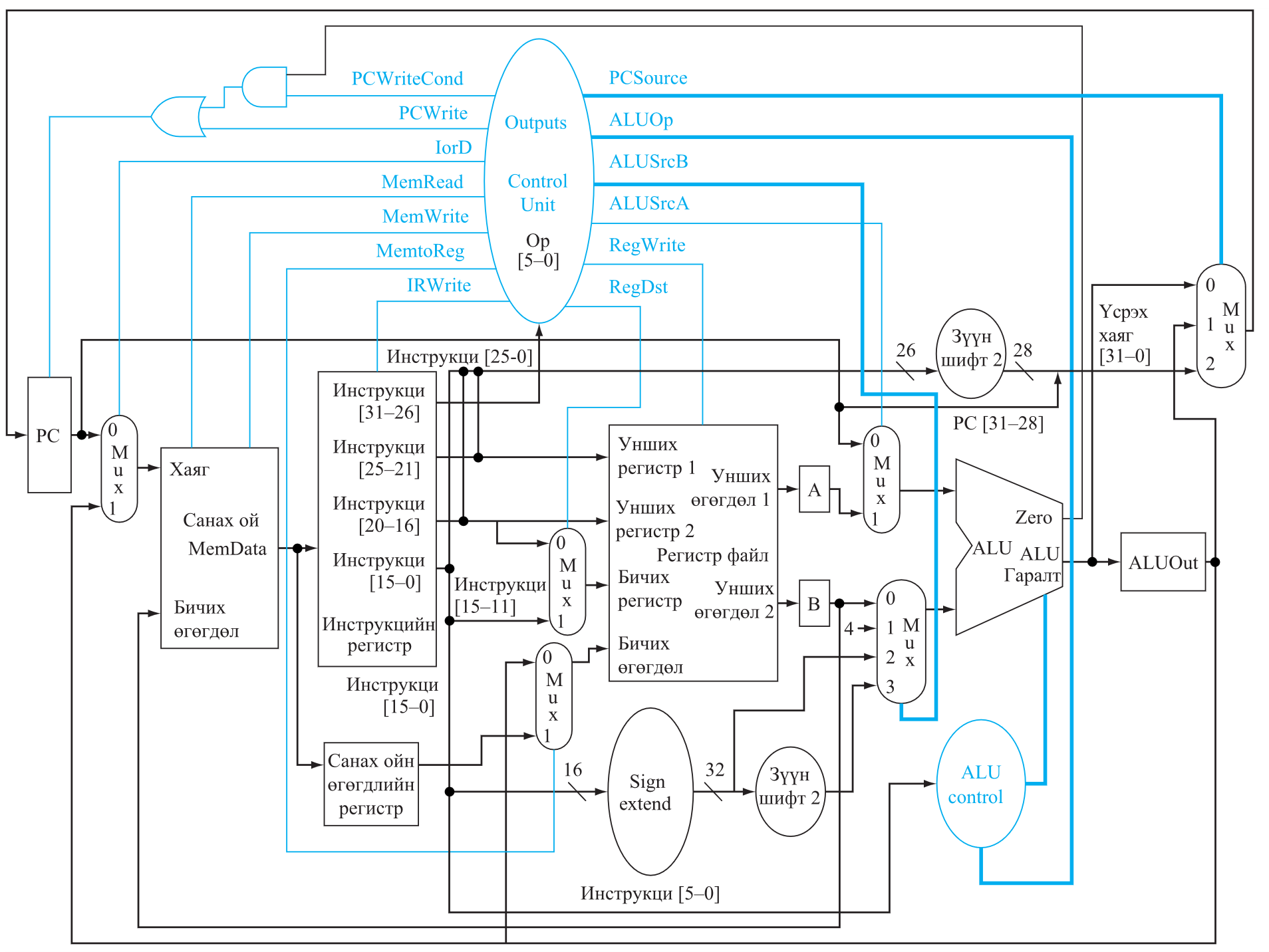
Зөвхөн IR регистр нь инструкци эхлэхээс хийгдэж дуусах хүртэл мэдээллээ хадгалсаар байх ёстой үүнийг удирдах ахин нэг удирдлагийн сигнал хэрэгтэй. ALU болон Adder, хоёр санах ойн нэгжүүд тус бүр нийлж байгаа тул түүнд харгалзах мультиплексорын оролт ихэснэ. Үүнийг дагаад зарим сонгох оролтын сигналын бит нэмэгдэнэ.



Зураг 8. IR, MDR, A, B, ALUOut

### Регистрын арифметик гүйцэтгэл, программ тоолуурын 4 өөр нэмэгдсэн утга, удирдлага салаалах

хаяг, нөхцөлгүй үсрэг хаяг гээд бүх үйлдлийг ALU – аар гүйцэтгүйлж байгааг харж болно.



Зураг 9. Удирдлагийн хэсгийг нэмсэн datapath

Дээрх зурагт удирдлагийн сигналуудыг харуулсан байна. Нэг циклт гүйцэтгэлтэй харьцуулахад шаардагдах нэгж хэсгүүд багассан ч удирдлагын сигналууд нэмэгдсэн байгааг харж болно.

MIPS инструкцуудын төрлөөс мөн микропроцессорын нэгж хэсгүүдийн ажиллагаанаас хамаарч инструкцийн гүйцэтгэлийг 5 шатанд хувааж байгаа. Энэ нь энэхүү олон циклт микропроцессор нь хамгийн удаан хийгдэх load word (lw) инструкцийг 5 клок циклд хийж дуусгана гэсэн үг.

**Instruction fetch** “Инструкцийг унших”.Инструкцийг санах ойгоос унших, программ тоолуурын утгийг 4 өөр нэмэгдүүлэх.

IR <= Memory[PC];

PC <= PC + 4;

**Instruction decode** “Инструкцийг задлах” болон registr fetch. Ямар ч төрлийн инструкци байсан хоёр source регистрыг харгалзах A болон B завсрын регистрт уншина. Үүний зэрэгцээ Инструкцын бага 16 битээр үсрэх хаягийг бодож гүйцэтгэнэ.

A <= Reg[IR[25:21]];

B <= Reg[IR[20:16]];

ALUOut <= PC + (sign-extend (IR[15-0]) << 2);

**Execution (EX)** “Арифметик үйлдэл хийх”, санах ойн хаягийг бодуулах, удирдлага салаалах инструкцийг гүйцэтгэж дуусгах. Инструкцын төрлөөс нь хамаарч ялгаатай гүйцэтгэл хийгдэнэ.

Санах ойд харьцангуй: Санах ойн хаягийг бодно.

ALUOut <= A + sign-extend (IR[15:0]);

Арифтетик – логик: Хоёр source регистрт байсан өгөгдлийн хооронд арифметик үйлдэлийг гүйцэтгэнэ.

ALUOut <= A op B;

Удирдлага салаалах: Хэрэв branch инструкцын нөхцөл биелвэл програм тоолуурын утгийг өөрчилнө.

if (A == B) PC <= ALUOut;

Нөхцөлгүй үсрэх: Үсрэх хаягаар програм тоолуурын утгийг солино.

PC <= {PC [31:28], (IR[25:0]],2'b00)};

**Memory access** санах ойд хандах, R – төрлийн инструкци гүйцэтгэгдэж дуусах.

Санах ойд харьцангуй: Бодсон санах ойн хаягнаас өгөгдлийг MDR регистр рүү уншина. Харин бодсон санах ойн хаяг руу B регистр дээрх source операндын утгийг бичнэ.

MDR <= Memory [ALUOut];

Memory [ALUOut] <= B;

Арифтетик – логик: ALU ын гаралтыг Destination регистр рүү бичнэ.

Reg[IR[15:11]] <= ALUOut;

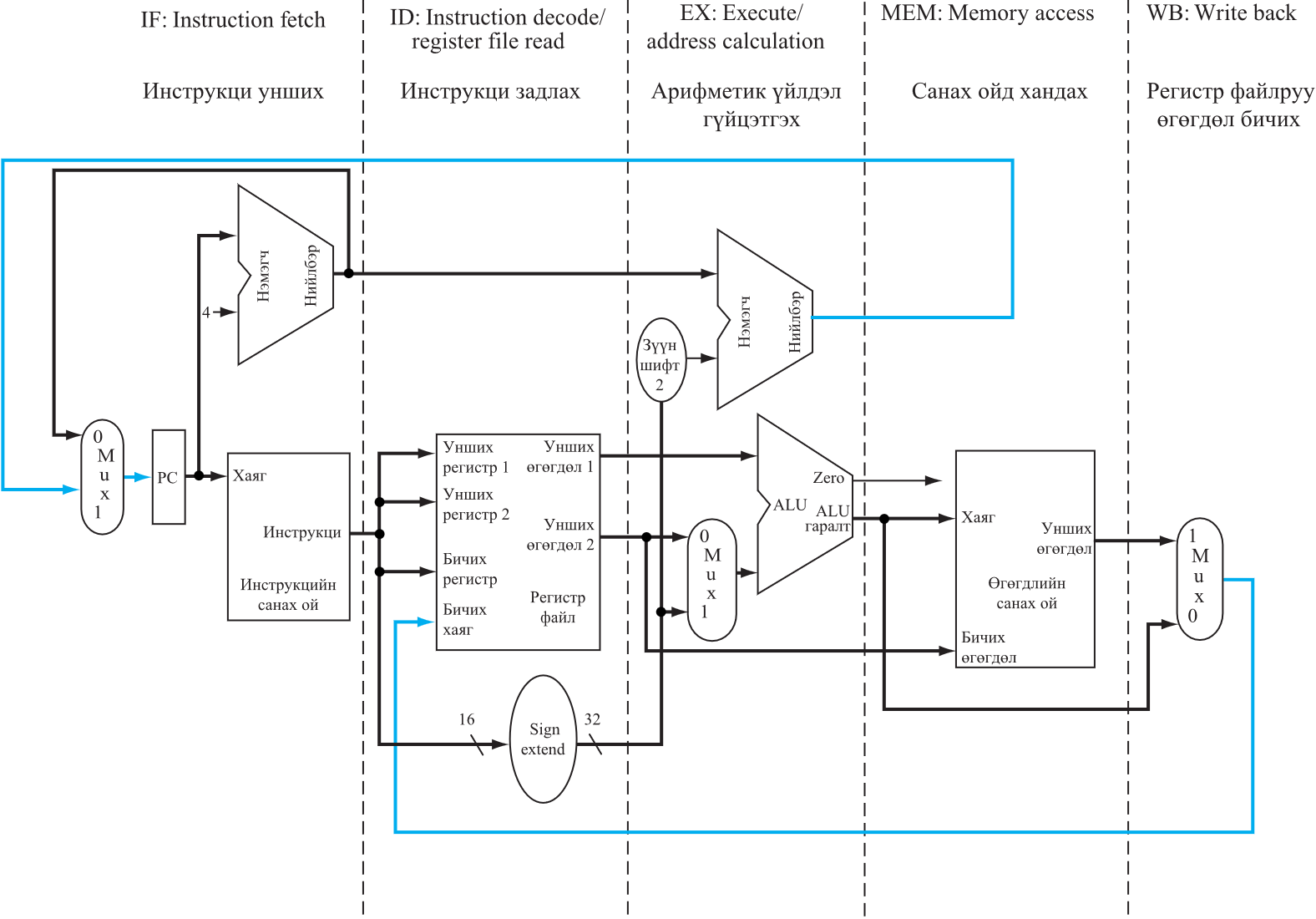
**Memory read completion** санах ойгоос регистр рүү өгөгдөл унших инструкци дуусах. MDR регистр дээрх өгөгдлийг destination регистр рүү бичнэ.

Reg[IR[20:16]] <= MDR;

1. **Пайплайн.**

Пайплайн гүйцэтгэл нь микропроцессорын ялгаатай хэсгүүдээр олон инструкцын харгалзах шатуудыг нэгэн зэрэг гүйцэтгэх арга юм. Жишээлбэл эхний инструкци нь инструкцыг задлах шат буюу 2 дугаар шатан дээрээ явж байна гэж үзье яг энэ мөчид дараагийн инструкцыг уншиж авч байна гэсэн үг. Микропроцессорын бүх хэсэг тасралтгүй ажиллах бөгөөд энэ нь микропроцессорын хурдыг нэмэгдүүлж үр ашгийг сайжруулна.

Пайплайн ажиллахын тулд datapath – ыг олон шатуудад хуваана. 32 бит MIPS микропроцессорын хувьд 5 шатад хуваагдана.

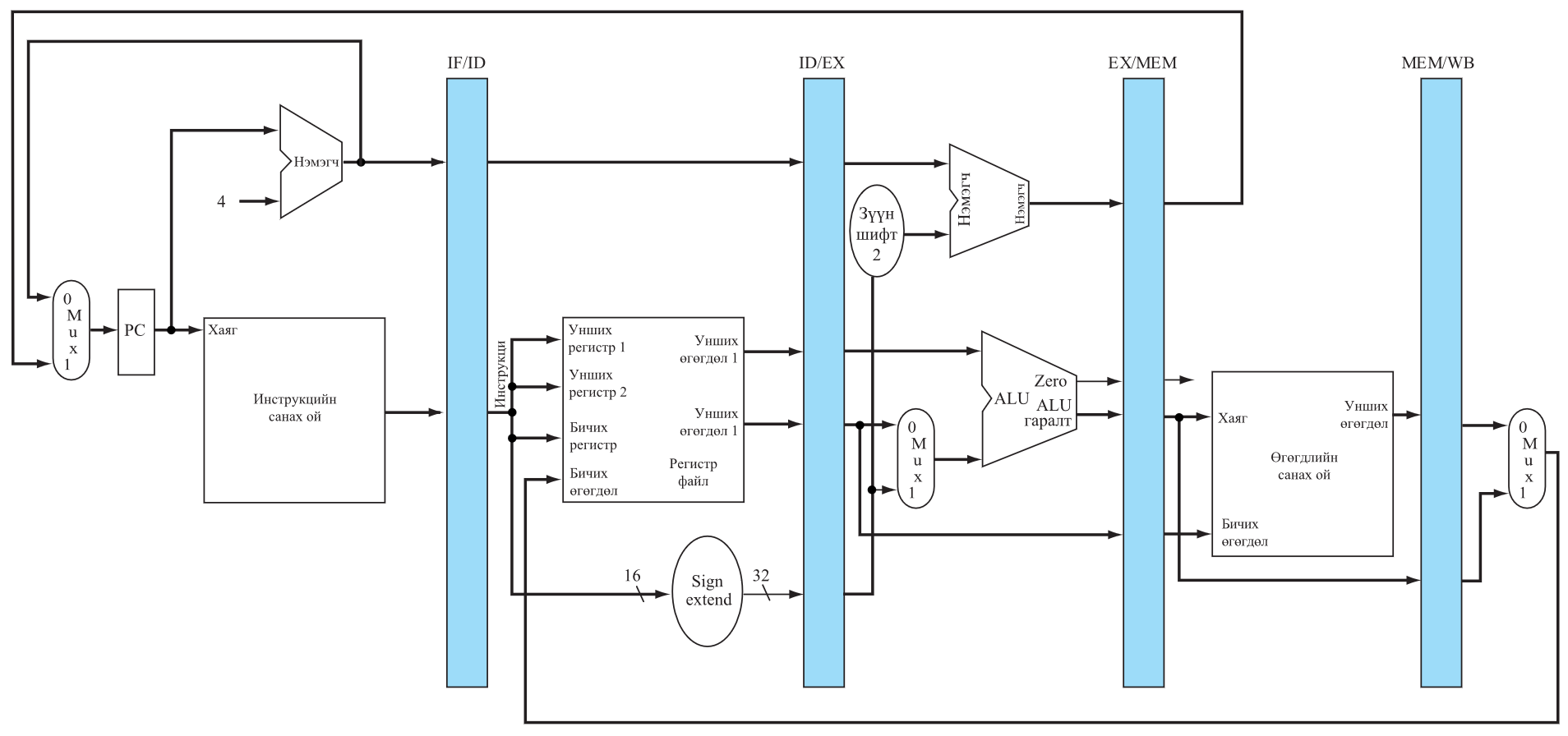


Зураг 10. Pipeline stage

Дээрх зурагт жишээ болгож нэг циклт гүйцэтгэлийн datapath ыг 5 шатанд хувааж харуулсан байна.

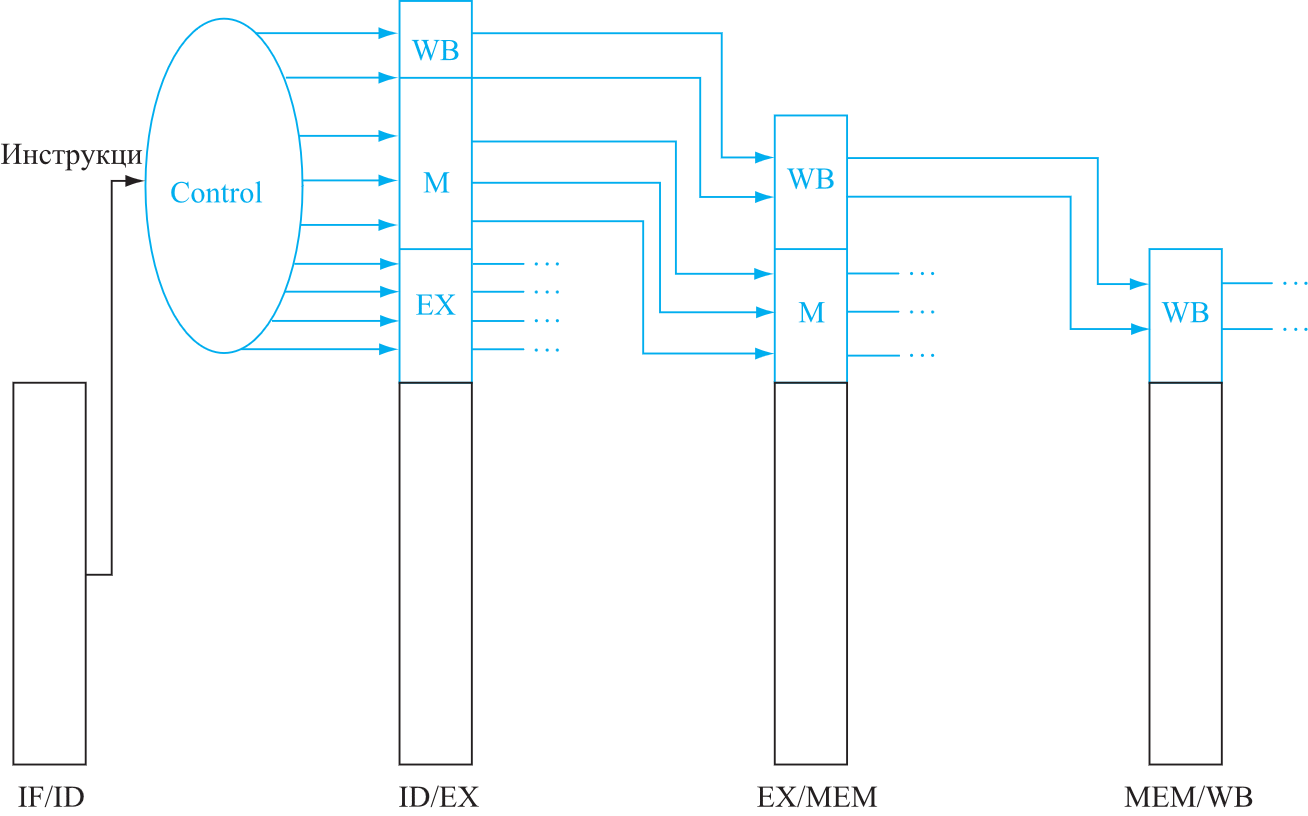
Уншсан инструкцийг 5 клок циклын турш шат бүрийн

завсар хадгалж явах завсрыг регистр хэрэгтэй.



Зураг 11. IF/ID, ID/EX, EX/MEM, MEM/WB

Инструкци бүрт удирдлагийн сигнал өөр байх учир үүнийг мөн адил шат тус бүрт дамжуулах нэмэлт регистр хэрэгтэй.



Зураг 12. Сүүлийн 3 шатад хэрэглэгдэх удирдлагийн сигнал

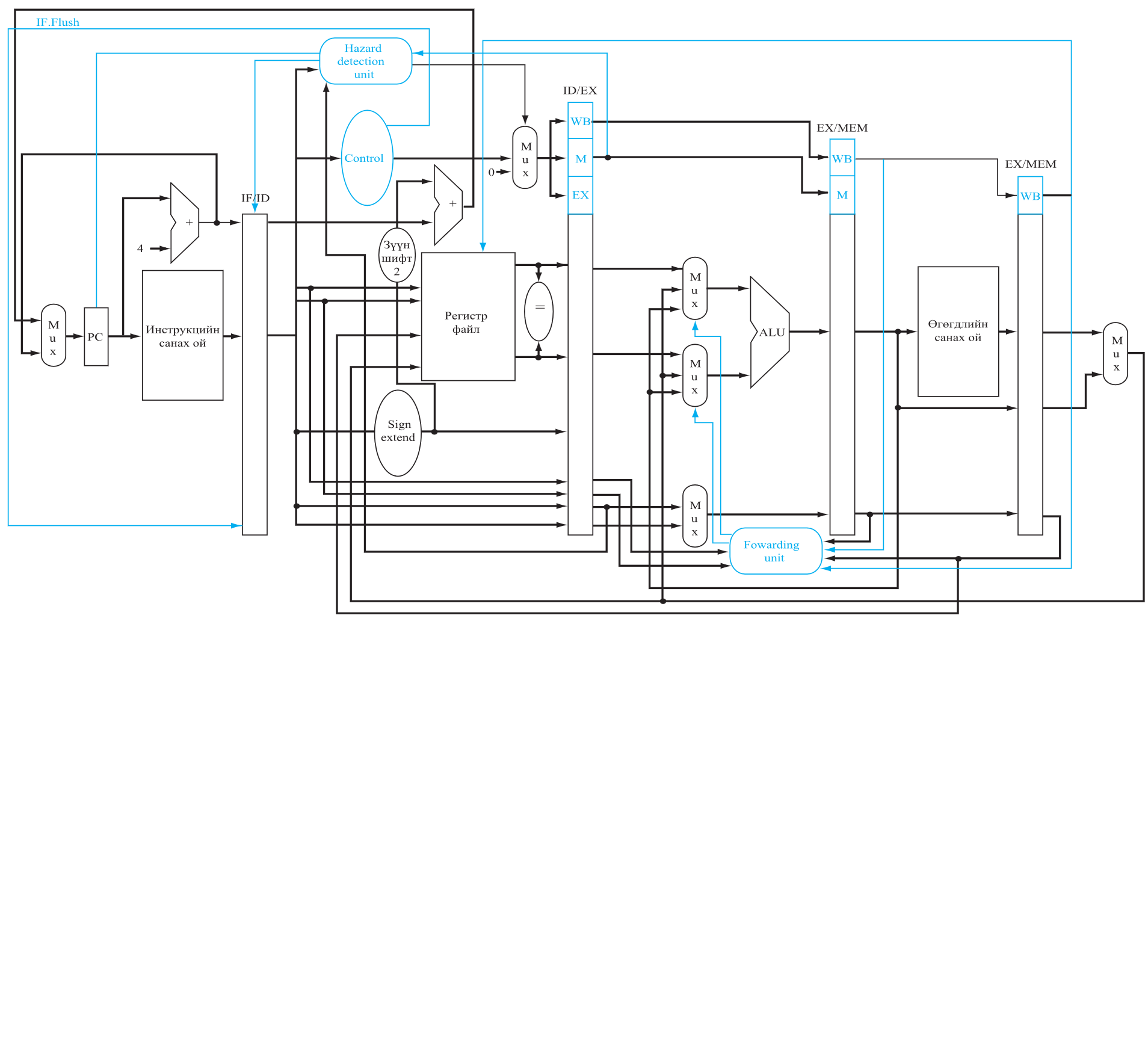
**Hazard detection, forwarding unit.**

Өмнөх инструкцийн үр дүн гарч дуусаагүйгээс үүдэн арын инструкци буруу үр дүн гаргах эсвэл буруу инструкци уншиж авахыг пайдлайн гүйцэтгэлийн hazard гэж хэлж болно. Hazard – аас үүдэн пайплайн ажиллагаа зогсооход хүрдэг. Дээрх datapath – тай холбоотойгоор өгөгдлийн hazard болон удирдлагийн hazard гэсэн 2 төрлийн hazard үүснэ.

* WB шат дээр үр дүнг регистр файлруу бичиж байгаагаас “Data hazard” үүснэ.
* MEM шат дээрээс ирэх удирдлага салаалах хаяг болон програм тоолуурын 4 өөр нэмэгдэх утгийн аль нэгийг сонгохоос “Branch hazard” үүснэ.

Өгөгдлийн хамааралтай үүссэн hazard – ыг forwarding unit ашиглан өгөгдөлийг хэрэгцээтэй болсон үед нь клок алдалгүй урьдчилан авч ажилладаг. Харин зайлшгүй нэг клок алдах ганц нөхцөл нь lw инструкцтай өгөгдлийн хамааралтай арын инструкцыг задалсаны дараагаар 1 клок “stall” хийж ажиллана.

Удирдлага салаалах инструкцаас үүдэлтэй hazard – ыг “branch prediction” буюу нөхцөл биелээгүй гэж үзэн арын инструкцыг уншиж ажиллуулах аргаар шийдэж болно. Олон хийгдэх давталт дээр ашиглагдсан удирдлага салаалах инструкци давталтын үед энгийн пайплайн ажиллагаагаар ажиллах бол давталтаас гарах буюу нөхцөл биелэх үед клок алдна. Нөхцөл биелэх үед аль хэдийн буруу инструкци уншиж авсан байх бөгөөд үүний “flush” хийх аргаар шийддэг.



Зураг 13. Hazard detection unit, Forwarding unit

# **HDL – ДЭЭРХ ГҮЙЦЭТГЭЛ**

Доорх бүх гүйцэтгэлийг “Computer Organization and Design by David A. Patterson and John L. Hennessy , 3rd edition” номны Datapath – ыг харж хэрэгжүүлсэн. Микропроцессорын бүх хэсгүүд “Verilog” хэлээр бичигдсэн. Голлох нэгжүүд болох

* PC, Instruction memory, Adder
* Register file, ALU
* Data memory
* Sign extention unit, Left shift 2

нэгжүүд нь бараг бүх гүйцэтгэлд өөрчлөгдөлгүй хэрэглэгдсэн учраас хэрэгжүүлэлтийг нэгтгэж орууллаа. Харин гүйцэтгэл бүрийн удирдлагийн хэсэг мөн завсрын санах элемент, нэмэлт нэгжүүдийн хэрэгжүүлсэн логикыг тус бүрд нь орууллаа.

**Instruction memory** инструкцын санах ой. Инструкци нь бүгд 32 бит буюу 4 байтын хэмжээтэй гээд санах ойн хэмжээг 256 инструкци багтахаар үүсгэсэн.



Зураг 14. Instruction memory

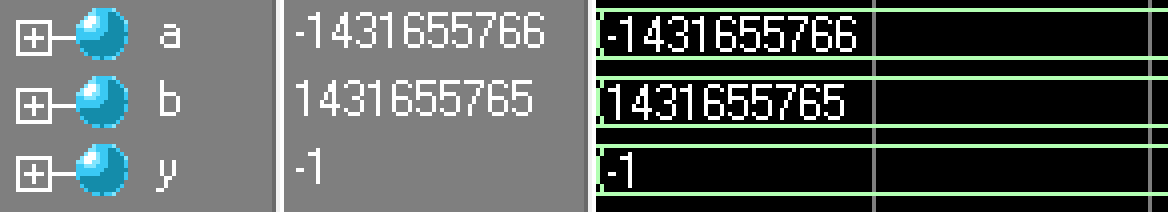
QtSpim симулятоыг ашиглан MIPS ассембле кодыг Hex кодоор илэрхийлсэн текст болгож хөрвүүлсэн. Хөрвүүлсэн текстээ Verilog хэлний $readmemh() функц ашишлан hex кодыг үүсгэсэн санах ойдоо уншиж ашигласан.

**Program counter** програм тоолуурыг 4 өөр нэмэгдүүлэхэд PC <= PC + 4; .



Зураг 15. PC test

**Adder** нэмэгч. 32 битын нэмэгчийг давталт ашиглан 32 ширхэг RCA – ыг generate хийх аргаар гүйцэтгэсэн.



Зураг 16. Adder test

**Register file** регистр файл. Давталт ашиглан 32 ширхэг регистр анхний утга 0 байхаар зарлаж өгсөн. register[i] <= 32'd0;

**Data memory** өгөгдлийн санах ойн хувьд дээрх аргаар 128 санах ойн үүр гаргаж өгсөн.

**ALU** арифметик – логик хэсэг.

|  |  |
| --- | --- |
| ALU удирдлага | Функц |
| 0000 | Логик AND |
| 0001 | Логик OR |
| 0010 | Нэмэх |
| 0110 | Хасах |
| 0111 | SLT |
| 1100 | NOR |

Хүснэгт 1. ALU control

|  |  |  |  |
| --- | --- | --- | --- |
| INST | Opcode | Инструкцийн үйлдэл | Функц |
| ADD | 000000 | rd = rs + rt | 100000 |
| SUB | 000000 | rd = rs - rt | 100010 |
| AND | 000000 | rd = rs & rt | 100100 |
| OR | 000000 | rd = rs | rt | 100101 |
| NOR | 000000 | rd = ~ (rs | rt) | 100111 |
| SLT | 000000 | rd = (rs < rt) | 101010 |
| SLL | 000000 | rd = rt << sa | 000000 |
| SRL | 000000 | rd = rt >> sa | 000010 |
| LW | 100011 | rt <= memory[rs+offset] | xxxxxx |
| SW | 101011 | mem[base+offset] <= rt | xxxxxx |
| BEQ | 000100 | if(rs =rt) then branch | xxxxxx |
| ADDI | 001000 | rt = rs + immediate | xxxxxx |
| ANDI | 001100 | rt = rs & immediate | xxxxxx |
| XORI | 001110 | rt = rs ^ immediate | xxxxxx |
| ORI | 001101 | rt = rs | immediate | xxxxxx |
| LUI | 001111 | rt = imm<<16 | xxxxxx |
| J | 000010 | pc = pc\_upp|(target<<2) | xxxxxx |

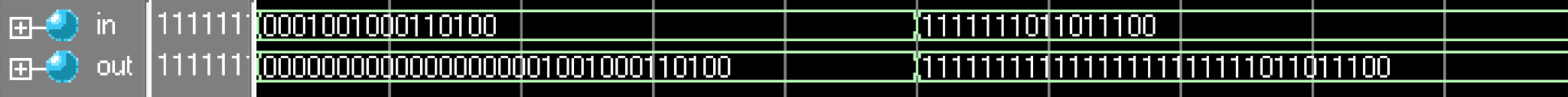
Хүснэгт 2. Инструкцын

R – төрлийн инструкцүүдийн хувьд opcode нь ижил байдаг тул ALU удирдлага бага 6 бит буюу функцээр нь ялгаж өгнө.

**Sign extention unit.** Ахлах битын утгаас нь хамаарч

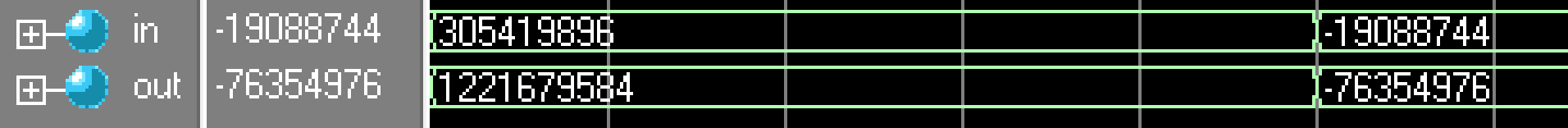
тухайн битээр дүүргэх энгийн логикоор хийж гүйцэтгэсэн.

assign out = { {16{in[15]}} , in };



Зураг 17. Sign extend test

**Shift left 2** зүүн тийш 2 шифт хийх буюу 4 т үржүүлэх хэсэг.



Зураг 18. Shift left 2 test

1. **Нэг циклт гүйцэтгэл.**

Нэг циклт гүйцэтгэлийн хувьд дээрх хэсгүүд дээр нэмээд удирдлагийн хэсэг нэмж өгөөд datapath – ын дагуу холбоход хангалттай. Удирдлагийн хэсгээс гарах шаардлагатай сигналууд нь нийт 9 байсан. ALU control сигналаас бусад нь

|  |  |  |
| --- | --- | --- |
| Сигнал | Идэвжсэн үед | Идэвжээгүй үед |
| RegDst | Dest Reg нь rt (20:16) | Dest Reg нь rd (15:11) |
| RegWrite | … | Регистрээс санах ойруу бичнэ |
| ALUSrc | ALU - ын 2 дахь операнд нь регстрээс | ALU - ын 2 дахь операнд нь инструкцын бага 16 битээс |
| PCSrc | 4 өөр нэмэгдсэн утга | Удирдлага салаалах хаяг |
| MemRead | ... | Заасан хаяг дээрх өгөгдөл санах ойн гаралтанд гарна |
| MemWrite | ... | Заасан хаяг дээрх өгөгдөл шинэчлэгдэнэ |
| MemtoReg | ALU - ын гаралтаас регистр рүү бичнэ | Санах ойн гаралтаас регистр рүү бичнэ |
| Jump | … | Үсрэх хаяг |

Хүснэгт 3. Control signal

1. **Олон циклт гүйцэтгэл.**

Олон циклт гүйцэтгэлийн хувьд үр дүн хадгалах регистр үүд хэрэгтэй байдаг. Үүнийг Verilog дээр хялбар шийдэх арга нь гаралтын сигналыг wire гэж зарлалгүйгээр reg гэж зарлаж өгхөд хангалттай.

|  |  |  |
| --- | --- | --- |
| Сигнал | Идэвжсэн үед | Идэвжээгүй үед |
| RegDst | Dest Reg нь rt (20:16) | Dest Reg нь rб (15:11) |
| RegWrite | … | Регистрээс санах ойруу бичнэ |
| ALUSrcA | ALU - ын эхний операнд нь PC | ALU - ын эхний операнд нь A регистр |
| MemRead | ... | Заасан хаяг дээрх өгөгдөл санах ойн гаралтанд гарна |
| MemWrite | ... | Заасан хаяг дээрх өгөгдөл шинэчлэгдэнэ |
| MemtoReg | ALU - ын гаралтаас регистр рүү бичнэ | Санах ойн гаралтаас регистр рүү бичнэ |
| IorD | PC санах ойд хягийг өгнө | ALU - ын гаралтаас санах ойн хаягийг өгнө |
| IRWrite | … | Санах ойн гаралт IR - ь бичигдэнэ |
| PCWrite | … | PC - ын утга өөрчлөгдөнө. |
| PCWriteCond | … | Zero гаралт 1 бол PC - ын утга өөрчлөгдөнө. |

Хүснэгт 4. Нэг битын удирдлагын сигнал

|  |  |  |
| --- | --- | --- |
| Сигнал | 2 - тын утга | Идэвжээгүй үед |
| ALUSrcB | 00 | Операнд нь B регистр |
| 01 | Операнд нь 4 |
| 10 | Операнд нь sign-extend (IR[15-0]) |
| 11 | Операнд нь sign-extend (IR[15-0]) << 2 |
| PCSource | 00 | ALU (PC + 4) |
| 01 | Удирдлага салаалах хаяг |
| 10 | Үсрэх хаяг |

Хүснэгт 5. 2 Битын удирдлагын сигнал

Дээрх сигналаар удирдлагийн хэсгийн дизайныг гаргаад зураг 9 д үзүүлсэний дагуу datapath – ыг холбоосон.

1. **Пайплайн.**

Завсрын регистрүүдийг гүйцэтгэхдээ праллель оролт гаралттай “PIPO” регистр байдлаар хийсэн.

Пайдлайн ажиллагаа дээр нэмэгдэж буй хоёр гол нэгж нь forwarding unit, hazard detecton unit. Forwarding unit – ыг загварчлахын тулд анхаарах зүйл нь өгөгдлийн хамаарлийг илрүүлдэг байх. Өгөгдлийн хамаарал үүссэн байх нөхцолүүд нь

**if (**EX/MEM.RegWrite **and** (EX/MEM.RegisterRd 0) **and** (EX/MEM.RegisterRd = ID/EX.RegisterRs)**)**

{

ForwardA = 10

}

**if (**EX/MEM.RegWrite **and** (EX/MEM.RegisterRd 0) **and** (EX/MEM.RegisterRd = ID/EX.RegisterRt)**)**

{

ForwardB = 10

}

**if (**MEM/WB.RegWrite **and** (MEM/WB.RegisterRd 0) **and** (MEM/WB.RegisterRd = ID/EX.RegisterRs)**)**

{

ForwardA = 01

}

**if (**MEM/WB.RegWrite **and** (MEM/WB.RegisterRd 0) **and** (MEM/WB.RegisterRd = ID/EX.RegisterRt)**)**

{

ForwardB = 01

}

|  |  |  |
| --- | --- | --- |
| MUX удирдлага | Source | Тайлбар |
| ForwardA = 00 | ID/EX | Эхний операнд регистр файлаас |
| ForwardA = 10 | EX/MEM | Эхний операнд ALU - ын гаралтаас |
| ForwardA = 01 | MEM/WB | Эхний операнд өгөгдлийн санах ойгоос |
| ForwardB = 00 | ID/EX | Хоёр дахь операнд регистр файлаас |
| ForwardB = 10 | EX/MEM | Хоёр дахь операнд ALU - ын гаралтаас |
| ForwardB = 01 | MEM/WB | Хоёр дахь операнд өгөгдлийн санах ойгоос |

Хүснэгт 6. Forwarding mux control value

LW-ийн Destitation register , Дараагийн инструкцын Source register-тэй адил үед заавалчгүй нэг клок юу ч хийхгүй байх шаардалга үүснэ. Үүнийг stall хийх байдлаар шийдвэрлэнэ.

**if (**ID/EX.MemRead **and** ((ID/EX.RegisterRt = IF/ID.RegisterRs) **or** (ID/EX.RegisterRt = IF/ID.RegisterRt))**)**

{

stall the pipeline

}

# **ТУРШИЛТ, СИМУЛЯЦИ**

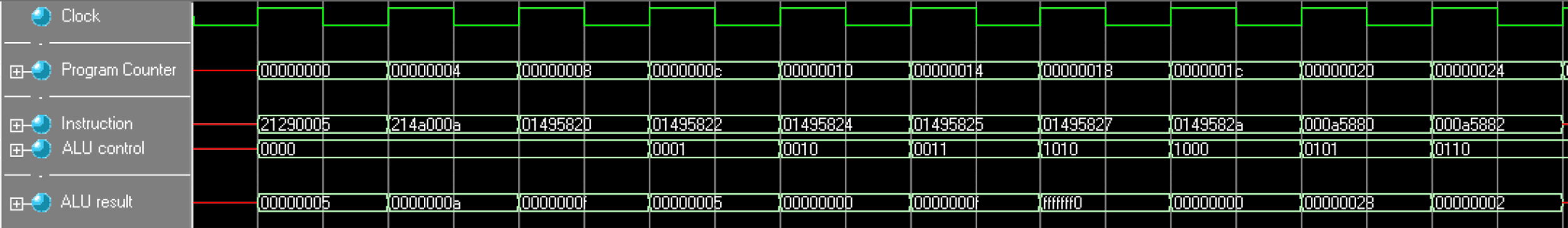
Эхлээд R төрлийн инструкцуудын ажиллагааг шалгаж үзсэн. Ингэхдээ уридчилан уншсан регистр дээрх өгөгдөл дээр ALU – аар процесс хийж гаралтыг харьцуулж харах замаар шалгасан.

|  |  |
| --- | --- |
| MIPS инструкци туршилт 1 | QtSpim ашиглан үүсгэсэн hex код |
| **addi** $t1, 5  **addi** $t2, 10  **add** $t3, $t2, $t1  **sub** $t3, $t2, $t1  **and** $t3, $t2, $t1  **or** $t3, $t2, $t1  **nor** $t3, $t2, $t1  **slt** $t3, $t2, $t1  **sll** $t3, $t2, 2  **srl** $t3, $t2, 2 | 21290005  214a000a  01495820  01495822  01495824  01495825  01495827  0149582a  000a5880  000a5882 |

Хүснэгт 7. R төрлийн инструкцын цуваа

|  |  |
| --- | --- |
| Инструкци | Тооцоолсон ALU гаралт |
| **addi**  **addi**  **add**  **sub**  **and**  **or**  **nor**  **slt**  **sll**  **srl** | 0000 0000 0000 0000 0000 0000 0000 0101  0000 0000 0000 0000 0000 0000 0000 1010  0000 0000 0000 0000 0000 0000 0000 1111  0000 0000 0000 0000 0000 0000 0000 0101  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 1111  1111 1111 1111 1111 1111 1111 1111 0000  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0010 1000  0000 0000 0000 0000 0000 0000 0000 0010 |

Хүснэгт 8. Инструкцын цувааны тооцоолсон утга



Хүснэгт 9. Туршилт 1

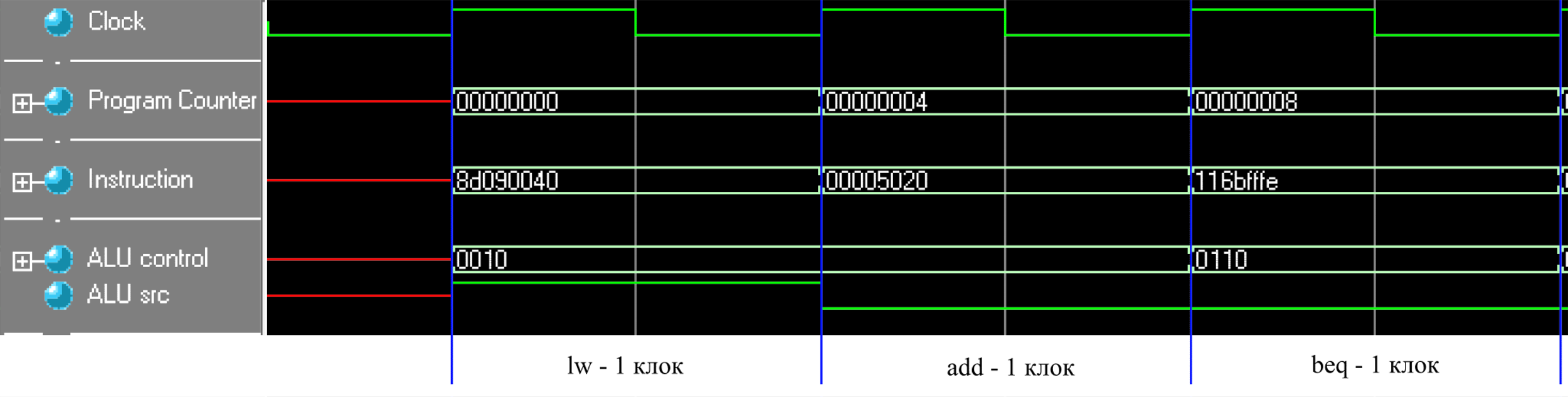
Туршилтын үр дүн тооцоолсон утгатай таарсан. Туршилт 1 – ын зурагт клок, програм тоолуур, уншсан инструкци, ALU удирдлага болон ALU гаралтыг харуулсан байгаа. 32 битын бүх сигналыг hex утгаар харуулсан байна.

Хоёр дахь туршилтаар нэг циклт болон олон циклт гүйцэтгэлийн ялгааг гаргах гэж үзсэн. Санах ойруу хандах нэг, регистр дээрх өгөгдөл дээр процесс хийх нэг, удирдлага салаалах нэг гээд гурван төрлийн инструкцын цувааг ажиллуулан шалгаж үзлээ.

|  |  |
| --- | --- |
| MIPS инструкци туршилт 2 | QtSpim ашиглан үүсгэсэн hex код |
| **loop:**  **lw** $t1, 64($t0)  **add** $t2, $0, $0  **beq** $t3, $t3, **loop** | 8d090040  00005020  116bfffe |

Хүснэгт 10. Төрөл бүрийн инструкцын цуваа

1. **Нэг циклт гүйцэтгэл.**



Зураг 19. Туршилт 2 –Нэг циклт гүйцэтгэл

1. **Олон циклт гүйцэтгэл.**



Зураг 20. Туршилт 2 – Олон циклт гүйцэтгэл

Нэг циклт гүйцэтгэлийн хувьд нэг клокын хугацаа хамгийн удаан хийгдэх инструкцаар тодорхойлогдож байсан. Харин олон циклт гүйцэтгэлийн хувьд нэг клокын урт нь хамгийн удаан хийж гүйцэтгэгдэх шат болох “memory access” аар тодорхойлогдоно. lw болон sw оос бусад инструкци санах ойд ханддагүй учраас клокыг хувааж өгснөөр дээрх удаан шатаар дамжиж өнгрөлгүй дуусах боломж өгдөг. Дээрх жишээнээс харвал регистр дээр үйлдэл хийх инструкци 4 клок, удирдлага салаалах инструкци 3 клокд хийгдэж байгаа нь микропроцессорын хурдыг бага ч гэсэн сайжруулж байгаа хэрэг.

1. **Пайплайн.**

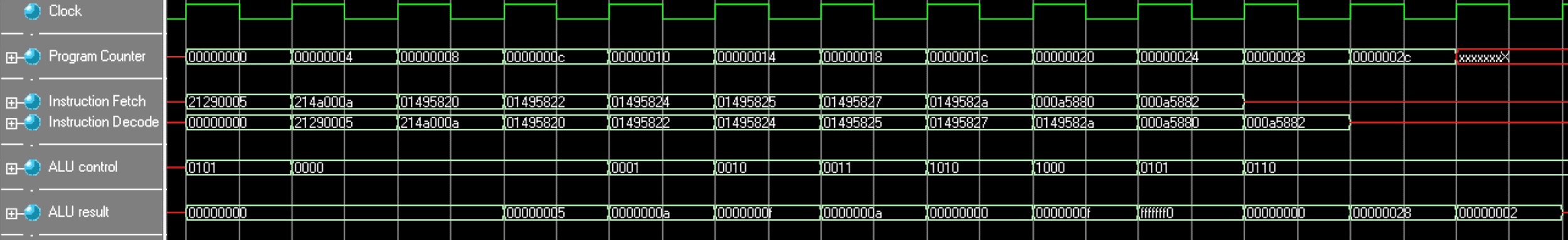
Гурав дахь туршилтаар пайплайн ажиллагааг шалгахын тулд өгөгдлийн хамааралгүй R төрлийн инструкцын цувааг ажиллуулж үзсэн.

|  |  |
| --- | --- |
| MIPS инструкци туршилт 3 | QtSpim ашиглан үүсгэсэн hex код |
| **addi** $t1, 5  **addi** $t2, 10  **add** $t3, $t2, $t1  **sub** $t3, $t2, $t1  **and** $t3, $t2, $t1  **or** $t3, $t2, $t1  **nor** $t3, $t2, $t1  **slt** $t3, $t2, $t1  **sll** $t3, $t2, 2  **srl** $t3, $t2, 2 | 21290005  214a000a  01495820  01495822  01495824  01495825  01495827  0149582a  000a5880  000a5882 |

Хүснэгт 11. R төрлийн инструкцын цуваа

|  |  |
| --- | --- |
| Инструкци | Тооцоолсон ALU гаралт |
| **addi**  **addi**  **add**  **sub**  **and**  **or**  **nor**  **slt**  **sll**  **srl** | 0000 0000 0000 0000 0000 0000 0000 0101  0000 0000 0000 0000 0000 0000 0000 1010  0000 0000 0000 0000 0000 0000 0000 1111  0000 0000 0000 0000 0000 0000 0000 0101  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 1111  1111 1111 1111 1111 1111 1111 1111 0000  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0010 1000  0000 0000 0000 0000 0000 0000 0000 0010 |

Хүснэгт 12. Инструкцын цувааны тооцоолсон утга



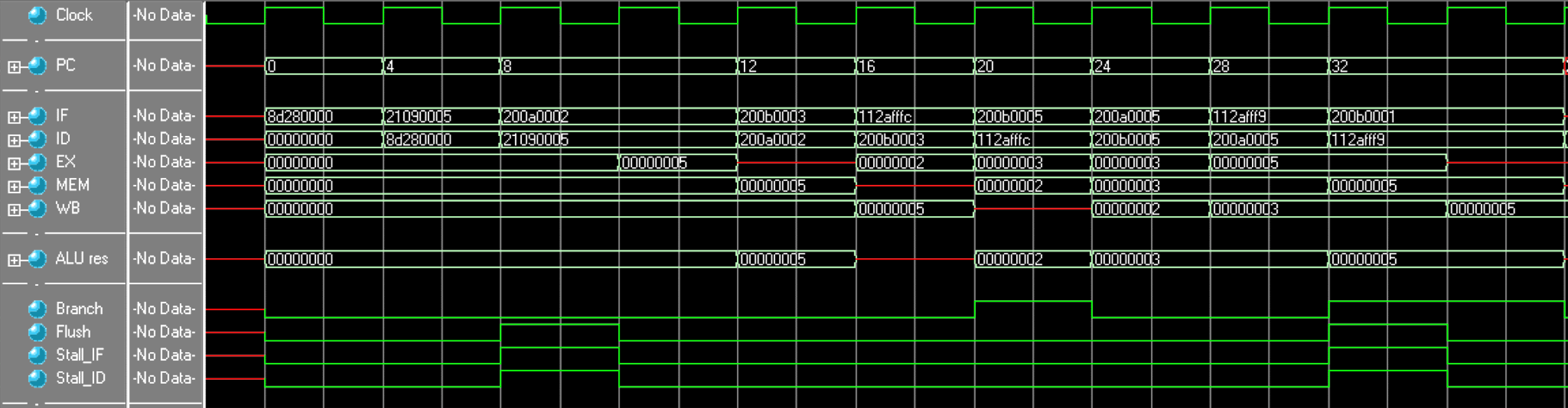
Зураг 21. Туршилт 3

Туршилтын үр дүн тооцоолсон утгатай таарсан. Туршилт 1 – ээс ялгаатай нь нэг клокын урт 5 дахин богино байна.

Дөрөв дэх туршилтаар пайплайн ажиллагаанд гарах hazard – уудыг зохицуудж байгааг шалгах үүднээс өгөгдөл болон удирдлагийн hazard үүсгэх инструкцын цувааг ажиллуулж үзсэн.

|  |  |
| --- | --- |
| MIPS инструкци туршилт 4 | QtSpim ашиглан үүсгэсэн hex код |
| loop:  ;data hazard  **lw** $t0, 0($t1)  **addi** $t1, $t0, 5  **addi** $t2, $zero, 2  **addi** $t3, $zero, 3  ;control hazard  **beq** $t1, $t2, loop  **addi** $t3, $zero, 5  **addi** $t2, $zero, 5  ;control hazard  **beq** $t1, $t2, loop  **addi** $t3, $zero, 1  **addi** $t2, $zero, 1  **addi** $t1, $zero, 1 | 8d280000  21090005  200a0002  200b0003  112afffc  200b0005  200a0005  112afff9  200b0001  200a0001  20090001 |

Хүснэгт 13. Hazard тай инструкцын цуваа



Зураг 22. Туршилт 4

lw инструкцтэй өгөгдлийн хамааралтай инструкци дээр нэг клок stall хийж байна. Мөн удирдлага салаалах инструкцийн нөхцөл биелсэн үед нэг клок stall хийж байна.

# **ДҮГНЭЛТ**

32 бит MIPS микропроцессорын core – ыг “Verilog” HDL ашиглан хийж гүйцэтгэсэн. Олон циклт болон пайплайн гүйцэтгэлтэй болгон өөрчилж ажиллагааг сайжруулсан. Тус бүр туршилт буюу инструкцын цувааг ажиллуулан үр дүнг шалгаж, хооронд нь харьцуулаж үзсэн.

# **НОМ ЗҮЙ**

1. Galani Tina G., Riya Saini and R.D.Daruwala “Design and Implementation of 32 – bit RISC Processor using Xilinx”.
2. David A. Patterson, John L. Hennessy “Computer Organization and Design,” the hardware / software interface, third edition, pp. 369– 436, 2005.
3. Dilip Kumar, K. P. “Design of High performance MIPS-32 Pipeline Processor”. Mohali, India, April, 2012.
4. Charles Price, “MIPS IV Instruction Set” Revision 3.2, September, 1995.0
5. K. Elissa, “FPGA prototyping by Verilog examples”.
6. Shobhit Shrivastav, S. K. (n.d.). Qualitative Analysis of 32 Bit MIPS Pipelined Processor. Delhi Technological University, May, 2020.

# **ХАВСРАЛТ**

1. **Нэг циклт гүйцэтгэл.**

`timescale 1ns**/**1ns

**module** adder **(**a**,**b**,**y**);**

**parameter** n**=**32**;**

**input** **[**n**-**1**:**0**]** a**,**b**;**

**wire** **[**n**:**0**]** w**;**

**output** **[**n**-**1**:**0**]** y**;**

**assign** w**[**0**]=**0**;**

**genvar** i**;**

**generate**

**for(**i**=**0**;**i**<=**n**-**1**;**i**=**i**+**1**)** **begin:**adding

FA FA\_inst**(.**a**(**a**[**i**]),.**b**(**b**[**i**]),** **.**cin**(**w**[**i**]),** **.**cout**(**w**[**i**+**1**]),** **.**s**(**y**[**i**]));**

**end**

**endgenerate**

**endmodule**

**module** FA**(input** a**,** **input** b**,** **input** cin**,** **output** cout**,** **output** s**);**

**assign** **{**cout**,**s**}=**a**+**b**+**cin**;**

**endmodule**

`timescale 1ns**/**1ns

**module** alu32**(** **input** **[**31**:**0**]** a**,**

**input** **[**31**:**0**]** b**,**

**input** **[**3**:**0**]** f**,**

**input** **[**4**:**0**]** shamt**,**

**output** **reg** **[**31**:**0**]** y**,**

**output** **reg** zero**);**

**always** **@** **(\*)** **begin**

**case** **(**f**)**

4'b0000**:** y **=** a **+** b**;** // ADD

4'b0001**:** y **=** a **-** b**;** // SUB

4'b0010**:** y **=** a **&** b**;** // AND

4'b0011**:** y **=** a **|** b**;** // OR

4'b0100**:** y **=** a **^** b**;** // XOR

4'b0101**:** y **=** b **<<** shamt**;** // SLL

4'b0110**:** y **=** b **>>** shamt**;** // SRL

4'b0111**:** y **=** $signed**(**$signed**(**b**)** **>>>** shamt**);** // SRA

4'b1000**:** y **=** $signed**(**a**)** **<** $signed**(**b**)** **?** 1 **:** 0**;** // SLT

4'b1001**:** y **=** a **<** b **?** 1 **:** 0**;** // SLTU

4'b1010**:** y **=** **~** **(**a **|** b**);** // NOR

4'b1011**:** y **=** b **<<** a**;** // SLLV

4'b1100**:** y **=** b **>>** a**;** // SRLV

4'b1101**:** y **=** $signed**(**$signed**(**b**)** **>>>** a**);** // SRAV

4'b1110**:** y **=** **{**b**[**15**:**0**],** 16'b0**};** // LUI

**endcase**

zero **=** **(**y**==**8'b0**);**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** Controlunit**(input** **[**5**:**0**]** Opcode**,**

**input** **[**5**:**0**]** Func**,**

**input** Zero**,**

**output** **reg** MemtoReg**,**

**output** **reg** MemWrite**,**

**output** **reg** ALUSrc**,**

**output** **reg** RegDst**,**

**output** **reg** RegWrite**,**

**output** **reg** Jump**,**

**output** PCSrc**,**

**output** **reg** **[**3**:**0**]** ALUControl

**);**

**reg** **[**7**:**0**]** temp**;**

**reg** Branch**,**B**;**

**always** **@(\*)** **begin**

**case** **(**Opcode**)**

6'b000000**:** **begin** // R-type

temp **<=** 8'b11000000**;**

**case** **(**Func**)**

6'b100000**:** ALUControl **<=** 4'b0000**;** // ADD

6'b100010**:** ALUControl **<=** 4'b0001**;** // SUB

6'b100100**:** ALUControl **<=** 4'b0010**;** // AND

6'b100101**:** ALUControl **<=** 4'b0011**;** // OR

6'b100110**:** ALUControl **<=** 4'b0100**;** // XOR

6'b100111**:** ALUControl **<=** 4'b1010**;** // NOR

6'b101010**:** ALUControl **<=** 4'b1000**;** // SLT

6'b000000**:** ALUControl **<=** 4'b0101**;** // SLL

6'b000010**:** ALUControl **<=** 4'b0110**;** // SRL

**endcase**

**end**

6'b100011**:** **begin** // LW

temp **<=** 8'b10100100**;**

ALUControl **<=** 4'b0000**;**

**end**

6'b101011**:** **begin** // SW

temp **<=** 8'b00101000**;**

ALUControl **<=** 4'b0000**;**

**end**

6'b000100**:** **begin** // BEQ

temp **<=** 8'b00010000**;**

ALUControl **<=** 4'b0001**;**

**end**

6'b001000**:** **begin** // ADDI

temp **<=** 8'b10100000**;**

ALUControl **<=** 4'b0000**;**

**end**

6'b001100**:** **begin** // ANDI

temp **<=** 8'b10100000**;**

ALUControl **<=** 4'b0010**;**

**end**

6'b001101**:** **begin** // ORI

temp **<=** 8'b10100000**;**

ALUControl **<=** 4'b0011**;**

**end**

6'b001110**:** **begin** // XORI

temp **<=** 8'b10100000**;**

ALUControl **<=** 4'b0100**;**

**end**

6'b000010**:** **begin** // J

temp **<=** 8'b00000010**;**

ALUControl **<=** 4'b0010**;**

**end**

**default:** temp **<=** 12'bxxxxxxxxxxxx**;**

**endcase**

**{**RegWrite**,**RegDst**,**ALUSrc**,**Branch**,**MemWrite**,**MemtoReg**,**Jump**,**B**}** **=** temp**;**

**end**

**assign** PCSrc **=** Branch **&** **(**Zero **^** B**);**

**endmodule**

`timescale 1ns**/**1ns

**module** decoder4**(** **input** **[**3**:**0**]** a**,**

**output** **reg** **[**15**:**0**]** y

**);**

**always** **@(\*)** **begin**

**if(**a **==** 4'b0000**)**

y **<=** 16'b0000000000000001**;**

**else** **if(**a **==** 4'b0001**)**

y **<=** 16'b0000000000000010**;**

**else** **if(**a **==** 4'b0010**)**

y **<=** 16'b0000000000000100**;**

**else** **if(**a **==** 4'b0011**)**

y **<=** 16'b0000000000001000**;**

**else** **if(**a **==** 4'b0100**)**

y **<=** 16'b0000000000010000**;**

**else** **if(**a **==** 4'b0101**)**

y **<=** 16'b0000000000100000**;**

**else** **if(**a **==** 4'b0110**)**

y **<=** 16'b0000000001000000**;**

**else** **if(**a **==** 4'b0111**)**

y **<=** 16'b0000000010000000**;**

**else** **if(**a **==** 4'b1000**)**

y **<=** 16'b0000000100000000**;**

**else** **if(**a **==** 4'b1001**)**

y **<=** 16'b0000001000000000**;**

**else** **if(**a **==** 4'b1010**)**

y **<=** 16'b0000010000000000**;**

**else** **if(**a **==** 4'b1011**)**

y **<=** 16'b0000100000000000**;**

**else** **if(**a **==** 4'b1100**)**

y **<=** 16'b0001000000000000**;**

**else** **if(**a **==** 4'b1101**)**

y **<=** 16'b0010000000000000**;**

**else** **if(**a **==** 4'b111**)**

y **<=** 16'b0100000000000000**;**

**else** **if(**a **==** 4'b1111**)**

y **<=** 16'b1000000000000000**;**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** mux2 **(**d0**,**d1**,**s**,**y**);**

**parameter** n**=**32**;**

**input** **[**n**-**1**:**0**]** d0**;**

**input** **[**n**-**1**:**0**]** d1**;**

**input** s**;**

**output** **[**n**-**1**:**0**]** y**;**

**assign** y **=** s **?** d1 **:** d0**;**

**endmodule**

`timescale 1ns**/**1ns

**module** mux4 **(**d0**,**d1**,**d2**,**d3**,**s**,**y**);**

**parameter** n**=**32**;**

**input** **[**n**-**1**:**0**]** d0**,**d1**,**d2**,**d3**;**

**input** **[**1**:**0**]** s**;**

**output** **reg** **[**n**-**1**:**0**]** y**;**

**always** **@\*** **begin**

**case(**s**)**

2'b00**:** y**<=**d0**;**

2'b01**:** y**<=**d1**;**

2'b10**:** y**<=**d2**;**

2'b11**:** y**<=**d3**;**

**endcase**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** ram**(**clk**,**we**,**adr**,**din**,**dout**);**

**parameter** depth **=**128**;**

**parameter** bits **=** 32**;**

**parameter** width **=** 32**;**

**input** clk**,** we**;**

**input** **[**bits**-**1**:**0**]** adr**;**

**input** **[**width**-**1**:**0**]** din**;**

**output** **[**width**-**1**:**0**]** dout**;**

**reg** **[**width**-**1**:**0**]** Dmem **[**depth**-**1**:**0**];**

**assign** dout **=** Dmem**[**adr**];**

**always** **@** **(posedge** clk**)** **begin**

**if** **(**we**)**

Dmem**[**adr**]** **<=** din**;**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** registerfile32 **(input** clk**,**

**input** we**,**

**input** reset**,**

**input** **[**4**:**0**]** ra1**,**

**input** **[**4**:**0**]** ra2**,**

**input** **[**4**:**0**]** wa**,**

**input** **[**31**:**0**]** wd**,**

**output** **[**31**:**0**]** rd1**,**

**output** **[**31**:**0**]** rd2**);**

**reg** **[**31**:**0**]** register **[**31**:**0**];**

**assign** rd1 **=** register**[**ra1**];**

**assign** rd2 **=** register**[**ra2**];**

**integer** i**;**

**initial** **begin**

**for** **(**i**=**1**;** i**<**32**;** i**=**i**+**1**)** **begin**

register**[**i**]** **<=** 32'd0**;**

**end**

**end**

**always** **@(posedge** clk**)**

**begin**

register**[**0**]=**0**;**

**if(**reset**)** **for(**i **=** 0**;** i **<** 32**;** i **=** i **+** 1**)** register**[**i**]** **=** 32'd0**;**

**else** **if** **(**we**)**

**if(**wa **!=** 0**)** register**[**wa**]=** wd**;**

**end**

`timescale 1ns**/**1ns

**module** rom**(**adr**,**dout**);**

**parameter** depth **=**256**;**

**parameter** bits **=** 32**;**

**parameter** width **=** 32**;**

**input** **[**bits**-**1**:**0**]** adr**;**

**output** **[**width**-**1**:**0**]** dout**;**

**reg** **[**width**-**1**:**0**]** Imem**[**depth**-**1**:**0**];**

**initial**

$readmemh**(**"Prog\_2.txt"**,** Imem**);**

**assign** dout **=** Imem**[**adr**/**4**];**

**endmodule**

`timescale 1ns**/**1ns

**module** signext**(** **input** **[**15**:**0**]** a**,**

**output** **[**31**:**0**]** y

**);**

**assign** y **=** **{** **{**16**{**a**[**15**]}}** **,** a **};**

**endmodule**

`timescale 1ns **/** 1ns

**module** slt2 **(**

**input** **[**31**:**0**]** a**,**

**output** **[**31**:**0**]** y

**);**

**assign** y **=** a **<<** 2**;**

**endmodule**

`include "adder.v"

`include "alu32.v"

`include "flopr\_param.v"

`include "mux2.v"

`include "mux4.v"

`include "regfile32.v"

`include "signext.v"

`include "sl2.v"

`timescale 1ns**/**1ns

**module** Datapath**(input** clk**,**

**input** reset**,**

**input** RegDst**,**

**input** RegWrite**,**

**input** ALUSrc**,**

**input** Jump**,**

**input** MemtoReg**,**

**input** PCSrc**,**

**input** **[**3**:**0**]** ALUControl**,**

**input** **[**31**:**0**]** ReadData**,**

**input** **[**31**:**0**]** Instr**,**

**output** **[**31**:**0**]** PC**,**

**output** ZeroFlag**,**

**output** **[**31**:**0**]** WriteData**,**

**output** **[**31**:**0**]** ALUResult**);**

**wire** **[**31**:**0**]** PCNext**,** PCplus4**,** PCbeforeBranch**,** PCBranch**;**

**wire** **[**31**:**0**]** extendedimm**,** extendedimmafter**,** MUXresult**,** dataone**,** aluop2**;**

**wire** **[**4**:**0**]** writereg**;**

// PC

flopr\_param **#(**32**)** PCregister**(**clk**,**reset**,** PC**,**PCNext**);**

adder **#(**32**)** pcadd4**(**PC**,** 32'd4 **,**PCplus4**);**

slt2 shifteradd2**(**extendedimm**,**extendedimmafter**);**

adder **#(**32**)** pcaddsigned**(**extendedimmafter**,**PCplus4**,**PCbeforeBranch**);**

mux2 **#(**32**)** branchmux**(**PCplus4 **,** PCbeforeBranch**,** PCSrc**,** PCBranch**);**

mux2 **#(**32**)** jumpmux**(**PCBranch**,** **{**PCplus4**[**31**:**28**],**Instr**[**25**:**0**],**2'b00 **},** Jump**,**PCNext**);**

// Register File

registerfile32 RF**(**clk**,**RegWrite**,** reset**,** Instr**[**25**:**21**],** Instr**[**20**:**16**],** writereg**,** MUXresult**,** dataone**,**WriteData**);**

mux2 **#(**5**)** writeopmux**(**Instr**[**20**:**16**],**Instr**[**15**:**11**],**RegDst**,** writereg**);**

mux2 **#(**32**)** resultmux**(**ALUResult**,** ReadData**,** MemtoReg**,**MUXresult**);**

// ALU

alu32 alucomp**(**dataone**,** aluop2**,** ALUControl**,** Instr**[**10**:**6**],** ALUResult**,** ZeroFlag**);**

signext immextention**(**Instr**[**15**:**0**],**extendedimm**);**

mux2 **#(**32**)** aluop2sel**(**WriteData**,**extendedimm**,** ALUSrc**,** aluop2**);**

**endmodule**

**B. Олон циклт гүйцэтгэл**

**module** alucontrol**(**AluOp**,**FnField**,**AluCtrl**);**

**input** **[**1**:**0**]** AluOp**;**

**input** **[**5**:**0**]** FnField**;** //for R-type instruction

**output** **reg** **[**3**:**0**]** AluCtrl**;**

**always@(**AluOp **or** FnField**)begin**

**casex({**AluOp**,**FnField**})**

8'b00\_xxxxxx**:**AluCtrl**=**4'b0010**;** //lw / sw

8'b01\_xxxxxx**:**AluCtrl**=**4'b0110**;** //beq

8'b1x\_xx0000**:**AluCtrl**=**4'b0010**;** //add

8'b1x\_xx0010**:**AluCtrl**=**4'b0110**;** //sub

8'b1x\_xx0100**:**AluCtrl**=**4'b0000**;** //and

8'b1x\_xx0101**:**AluCtrl**=**4'b0001**;** //or

8'b1x\_xx1010**:**AluCtrl**=**4'b0111**;** //slt

**endcase**

**end**

**endmodule**

**module** control **(**clk**,** reset**,**Op**,** Zero**,** IorD**,** MemRead**,** MemWrite**,** MemtoReg**,** IRWrite**,**

PCSource**,** ALUSrcB**,** ALUSrcA**,** RegWrite**,** RegDst**,** PCSel**,** ALUOp**);**

**input** clk**;**

**input** reset**;**

**input** **[**5**:**0**]** Op**;**

**input** Zero**;**

**output** **reg** IorD**;**

**output** **reg** MemWrite**;**

**output** **reg** MemRead**;**

**output** **reg** MemtoReg**;**

**output** **reg** IRWrite**;**

**output** **reg** PCSource**;**

**output** **reg** RegDst**;**

**output** **reg** RegWrite**;**

**output** **reg** ALUSrcA**;**

**output** **reg** **[**1**:**0**]** ALUSrcB**;**

**output** PCSel**;**

**output** **reg** **[**1**:**0**]** ALUOp**;**

**reg** PCWrite**;**

**reg** PCWriteCond**;**

**assign**

PCSel **=** **(**PCWrite **|** **(**PCWriteCond **&** Zero**));**

//states

**parameter** FETCH **=** 4'b0000**;**

**parameter** DECODE **=** 4'b0001**;**

**parameter** MEMADRCOMP **=** 4'b0010**;**

**parameter** MEMACCESSL **=** 4'b0011**;**//L1

**parameter** MEMREADEND **=** 4'b0100**;**//L2

**parameter** MEMACCESSS **=** 4'b0101**;**//S

**parameter** EXECUTION **=** 4'b0110**;**

**parameter** RTYPEEND **=** 4'b0111**;**

**parameter** BEQ **=** 4'b1000**;**

**reg** **[**3**:**0**]** state**;**

**reg** **[**3**:**0**]** nextstate**;**

**always@(posedge** clk**)**

**if** **(**reset**)**

state **<=** FETCH**;**

**else**

state **<=** nextstate**;**

**always@(**state **or** Op**)** **begin**

**case** **(**state**)**

FETCH**:** nextstate **=** DECODE**;**

DECODE**:** **case(**Op**)**

//OpCode

6'b100011**:** nextstate **=** MEMADRCOMP**;**//lw

6'b101011**:** nextstate **=** MEMADRCOMP**;**//sw

6'b000000**:** nextstate **=** EXECUTION**;**//r

6'b000100**:** nextstate **=** BEQ**;**//beq

**default:** nextstate **=** FETCH**;**

**endcase**

MEMADRCOMP**:** **case(**Op**)**

6'b100011**:** nextstate **=** MEMACCESSL**;**//lw

6'b101011**:** nextstate **=** MEMACCESSS**;**//sw

**default:** nextstate **=** FETCH**;**

**endcase**

MEMACCESSL**:** nextstate **=** MEMREADEND**;**

MEMREADEND**:** nextstate **=** FETCH**;**

MEMACCESSS**:** nextstate **=** FETCH**;**

EXECUTION**:** nextstate **=** RTYPEEND**;**

RTYPEEND**:** nextstate **=** FETCH**;**

BEQ**:** nextstate **=** FETCH**;**

**default:** nextstate **=** FETCH**;**

**endcase**

**end**

**always@(**state**)** **begin**

IorD**=**1'b0**;** MemRead**=**1'b0**;** MemWrite**=**1'b0**;** MemtoReg**=**1'b0**;** IRWrite**=**1'b0**;** PCSource**=**1'b0**;**

ALUSrcB**=**2'b00**;** ALUSrcA**=**1'b0**;** RegWrite**=**1'b0**;** RegDst**=**1'b0**;** PCWrite**=**1'b0**;** PCWriteCond**=**1'b0**;** ALUOp**=**2'b00**;**

**case** **(**state**)**

FETCH**:**

**begin**

MemRead **=** 1'b1**;**

IRWrite **=** 1'b1**;**

ALUSrcB **=** 2'b01**;**

PCWrite **=** 1'b1**;**

**end**

DECODE**:**

ALUSrcB **=** 2'b11**;**

MEMADRCOMP**:**

**begin**

ALUSrcA **=** 1'b1**;**

ALUSrcB **=** 2'b10**;**

**end**

MEMACCESSL**:**

**begin**

MemRead **=** 1'b1**;**

IorD **=** 1'b1**;**

**end**

MEMREADEND**:**

**begin**

RegWrite **=** 1'b1**;**

MemtoReg **=** 1'b1**;**

RegDst **=** 1'b0**;**

**end**

MEMACCESSS**:**

**begin**

MemWrite **=** 1'b1**;**

IorD **=** 1'b1**;**

**end**

EXECUTION**:**

**begin**

ALUSrcA **=** 1'b1**;**

ALUOp **=** 2'b10**;**

**end**

RTYPEEND**:**

**begin**

RegDst **=** 1'b1**;**

RegWrite **=** 1'b1**;**

**end**

BEQ**:**

**begin**

ALUSrcA **=** 1'b1**;**

ALUOp **=** 2'b01**;**

PCWriteCond **=** 1'b1**;**

PCSource **=** 2'b01**;**

**end**

**endcase**

**end**

**endmodule**

**module** datapath**(**clk**,** reset**,** IorD**,** MemRead**,** MemWrite**,** MemtoReg**,** IRWrite**,**

PCSource**,** ALUSrcB**,** ALUSrcA**,** RegWrite**,** RegDst**,** PCSel**,** ALUCtrl**,** Op**,** Zero**,** Function**);**

**parameter** PCSTART **=** 128**;** //sanah oi ehleh

**input** clk**;**

**input** reset**;**

**input** IorD**;**

**input** MemWrite**,**MemRead**,**MemtoReg**;**

**input** IRWrite**;**

**input** PCSource**;**

**input** RegDst**,**RegWrite**;**

**input** ALUSrcA**;**

**input** **[**1**:**0**]** ALUSrcB**;**

**input** PCSel**;**

**input** **[**3**:**0**]** ALUCtrl**;**

**output** **[**5**:**0**]** Op**;**

**output** Zero**;**

**output** **[**5**:**0**]** Function**;**

**reg** **[**7**:**0**]**PC**;**

**reg** **[**31**:**0**]** ALUOut**;**

**reg** **[**31**:**0**]** ALUResult**;**

**wire** **[**31**:**0**]** OpA**;**

**reg** **[**31**:**0**]** OpB**;**

**reg** **[**31**:**0**]**A**;**

**reg** **[**31**:**0**]**B**;**

**wire** **[**7**:**0**]** address**;**

**wire** **[**31**:**0**]** MemData**;**

**reg[**31**:**0**]**mem**[**255**:**0**];**

**reg** **[**31**:**0**]**Instruction**;**

**reg** **[**31**:**0**]**mdr**;**

**wire** **[**31**:**0**]** da**;**//read data 1

**wire** **[**31**:**0**]** db**;**//read data 2

**reg[**31**:**0**]**registers**[**31**:**0**];**

**assign** Function**=**Instruction**[**5**:**0**];**

**assign** Op**=**Instruction**[**31**:**26**];**

//data and instruction memory

**assign** address**=(**IorD**)?**ALUOut**:**PC**;**

**initial**

$readmemh**(**"mem.dat"**,** mem**);**

**always** **@(posedge** clk**)** **begin**

**if(**MemWrite**)**

mem**[**address**]<=**B**;**

**end**

**assign**

MemData **=(**MemRead**)?** mem**[**address**]:**32'bx**;**

//PC logic

**always@** **(posedge** clk**)begin**

**if(**reset**)**

PC**<=**PCSTART**;**

**else**

**if(**PCSel**)begin**

**case** **(**PCSource**)**

1'b0**:** PC**<=**ALUResult**;**

1'b1**:** PC**<=**ALUOut**;**

**endcase**

**end**

**end**

//instruction register

**always** **@(posedge** clk**)** **begin**

**if** **(**IRWrite**)**

Instruction **<=** MemData**;**

**end**

//memory data register

**always** **@(posedge** clk**)** **begin**

mdr **<=** MemData**;**

**end**

//register file

//$r0 is always 0

**assign** da **=** **(**Instruction**[**25**:**21**]!=**0**)** **?** registers**[**Instruction**[**25**:**21**]]** **:** 0**;**

**assign** db **=** **(**Instruction**[**20**:**16**]!=**0**)** **?** registers**[**Instruction**[**20**:**16**]]** **:** 0**;**

**always** **@(posedge** clk**)** **begin**

**if** **(**RegWrite**)begin**

**if** **(**RegDst**)**

registers**[**Instruction**[**15**:**11**]]<=(**MemtoReg**)?**mdr**:**ALUOut**;**

**else**

registers**[**Instruction**[**20**:**16**]]<=(**MemtoReg**)?**mdr**:**ALUOut**;**

**end**

**end**

//A and B registers

**always** **@(posedge** clk**)** **begin**

A**<=**da**;**

**end**

**always@(posedge** clk**)** **begin**

B**<=**db**;**

**end**

//ALU

**assign** OpA**=(**ALUSrcA**)?**A**:**PC**;**

**always@(**ALUSrcB **or** B **or** Instruction**[**15**:**0**])begin**

**casex(**ALUSrcB**)**

2'b00**:**OpB**=**B**;**

2'b01**:**OpB**=**1**;**

2'b1x**:**OpB**={{(**16**){**Instruction**[**15**]}},**Instruction**[**15**:**0**]};**

**endcase**

**end**

**assign** Zero **=** **(**ALUResult**==**0**);** //Zero == 1 when ALUResult is 0 (for branch)

**always** **@(**ALUCtrl **or** OpA **or** OpB**)** **begin**

**case(**ALUCtrl**)**

4'b0000**:**ALUResult **=** OpA **&** OpB**;**

4'b0001**:**ALUResult **=** OpA **|** OpB**;**

4'b0010**:**ALUResult **=** OpA **+** OpB**;**

4'b0110**:**ALUResult **=** OpA **-** OpB**;**

4'b0111**:**ALUResult **=** OpA **<** OpB**?**1**:**0**;**

4'b1100**:**ALUResult **=** **~(**OpA **|** OpB**);**

**endcase**

**end**

//ALUOut register

**always@(posedge** clk**)** **begin**

ALUOut**<=**ALUResult**;**

**end**

**endmodule**

1. **Пайплайн.**

`timescale 1ns**/**1ns

**module** forwardingunit**(** **input** **[**4**:**0**]** Rs\_EX**,**

**input** **[**4**:**0**]** Rt\_EX**,**

**input** **[**4**:**0**]** Rs\_ID**,**

**input** **[**4**:**0**]** Rt\_ID**,**

**input** **[**4**:**0**]** writereg\_M**,**

**input** **[**4**:**0**]** writereg\_WB**,**

**input** RegWrite\_M**,**

**input** RegWrite\_WB**,**

**output** **reg[**1**:**0**]** ForwardAE**,**

**output** **reg[**1**:**0**]** ForwardBE**,**

**output** **reg** ForwardAD**,**

**output** **reg** ForwardBD **);**

**always** **@(\*)**

**begin**

// EX

**if** **(**RegWrite\_M

**&&** **(**writereg\_M **!=** 0**)**

**&&** **(**writereg\_M **==** Rs\_EX**))**

ForwardAE **=** 2'b10**;**

// MEM

**else** **if** **(**RegWrite\_WB

**&&** **(**writereg\_WB **!=** 0**)**

**&&** **(**writereg\_WB **==** Rs\_EX**))**

ForwardAE **=** 2'b01**;**

// NO

**else**

ForwardAE **=** 2'b00**;**

// EX

**if** **(**RegWrite\_M

**&&** **(**writereg\_M **!=** 0**)**

**&&** **(**writereg\_M **==** Rt\_EX**))**

ForwardBE **=** 2'b10**;**

// MEM

**else** **if** **(**RegWrite\_WB

**&&** **(**writereg\_WB **!=** 0**)**

**&&** **(**writereg\_WB **==** Rt\_EX**))**

ForwardBE **=** 2'b01**;**

// ID/EX

**else**

ForwardBE **=** 2'b00**;**

ForwardAD **=** **(**writereg\_M **!=**0**)** **&&** **(**Rs\_ID **==** writereg\_M**)** **&&** RegWrite\_M**;**

ForwardBD **=** **(**writereg\_M **!=**0**)** **&&** **(**Rt\_ID **==** writereg\_M**)** **&&** RegWrite\_M**;**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** hazardunit**(** **input** **[**4**:**0**]** Rt\_EX**,**

**input** **[**4**:**0**]** Rs\_D**,**

**input** **[**4**:**0**]** Rt\_D**,**

**input** **[**4**:**0**]** writereg\_M**,**

**input** **[**4**:**0**]** writereg\_EX**,**

**input** MemtoReg\_E**,**

**input** MemtoReg\_M**,**

**input** RegWrite\_EX**,**

**input** Branch\_ID**,**

**input** Jump\_ID**,**

**output** **reg** stall\_IF\_ID**,**

**output** **reg** stall\_ID\_EX**,**

**output** **reg** flush\_EX\_Mem**);**

**reg** lwstall**,** branchstall**;**

**always** **@(\*)** **begin**

lwstall**=** **((**Rs\_D **==** Rt\_EX**)** **||** **(**Rt\_D **==** Rt\_EX**))** **&&** MemtoReg\_E**;**

branchstall **=**Branch\_ID **&**

**(**RegWrite\_EX **&**

**(**writereg\_EX **==** Rs\_D **|** writereg\_EX **==** Rt\_D**)** **|**

MemtoReg\_M **&**

**(**writereg\_M **==** Rs\_D **|** writereg\_M **==** Rt\_D**));**

stall\_ID\_EX **=** lwstall **||** branchstall **||** Jump\_ID**;**

stall\_IF\_ID **=** lwstall **||** branchstall **||** Jump\_ID**;**

flush\_EX\_Mem **=** lwstall **||** branchstall **||** Jump\_ID**;**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** IF\_ID**(input** clk**,**

**input** rst**,**

**input** stall**,**

**input** **[**31**:**0**]**PCplus4\_IF**,**

**output** **reg** **[**31**:**0**]**PCplus4\_ID**,**

**input** **[**31**:**0**]**Instr\_IF**,**

**output** **reg** **[**31**:**0**]**Instr\_ID**);**

**always@(posedge** clk**)**

**begin**

**if** **(**rst**)** **begin**

PCplus4\_ID **<=** 0**;**

Instr\_ID **<=** 0**;**

**end**

**else** **if(**stall**)** **begin**

PCplus4\_ID **<=** PCplus4\_ID**;**

Instr\_ID **<=** Instr\_ID**;**

**end**

**else** **begin**

PCplus4\_ID **<=** PCplus4\_IF**;**

Instr\_ID **<=** Instr\_IF**;**

**end**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** ID\_EX**(input** clk**,**

**input** rst**,**

**input[**31**:**0**]** dataone\_ID**,**

**output** **reg** **[**31**:**0**]** dataone\_Ex**,**

**input[**31**:**0**]** WriteData\_ID**,**

**output** **reg** **[**31**:**0**]** WriteData\_Ex**,**

**input[**31**:**0**]** extendedimm\_ID**,**

**output** **reg** **[**31**:**0**]** extendedimm\_Ex**,**

**input** **[**31**:**0**]** Instr\_ID**,**

**output** **reg** **[**31**:**0**]** Instr\_Ex**,**

**input** RegWrite\_ID**,**

**output** **reg** RegWrite\_Ex**,**

**input** MemtoReg\_ID**,**

**output** **reg** MemtoReg\_Ex**,**

**input** MemWrite\_ID**,**

**output** **reg** MemWrite\_Ex**,**

**input** **[**3**:**0**]**ALUControl\_ID**,**

**output** **reg** **[**3**:**0**]**ALUControl\_Ex**,**

**input** ALUSrc\_ID**,**

**output** **reg** ALUSrc\_Ex**,**

**input** RegDst\_ID**,**

**output** **reg** RegDst\_Ex**);**

**always@(posedge** clk**)**

**begin**

**if** **(**rst**)** **begin**

dataone\_Ex **<=** 0**;**

WriteData\_Ex **<=** 0**;**

extendedimm\_Ex **<=** 0**;**

Instr\_Ex **<=** 0**;**

RegWrite\_Ex **<=** 0**;**

MemtoReg\_Ex **<=** 0**;**

MemWrite\_Ex **<=** 0**;**

ALUControl\_Ex **<=** 0**;**

ALUSrc\_Ex **<=** 0**;**

RegDst\_Ex **<=** 0**;**

**end**

**else** **begin**

dataone\_Ex **<=** dataone\_ID**;**

WriteData\_Ex **<=** WriteData\_ID**;**

extendedimm\_Ex **<=** extendedimm\_ID**;**

Instr\_Ex **<=** Instr\_ID**;**

RegWrite\_Ex **<=** RegWrite\_ID**;**

MemtoReg\_Ex **<=** MemtoReg\_ID**;**

MemWrite\_Ex **<=** MemWrite\_ID**;**

ALUControl\_Ex **<=** ALUControl\_ID**;**

ALUSrc\_Ex **<=** ALUSrc\_ID**;**

RegDst\_Ex **<=** RegDst\_ID**;**

**end**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** EX\_M**(input** clk**,**

**input** rst**,**

**input** **[**31**:**0**]** ALUResult\_Ex**,**

**output** **reg** **[**31**:**0**]** ALUResult\_M**,**

**input** **[**31**:**0**]** WriteData\_Ex**,**

**output** **reg** **[**31**:**0**]** WriteData\_M**,**

**input** **[**4**:**0**]** writereg\_Ex**,**

**output** **reg** **[**4**:**0**]** writereg\_M**,**

**input** RegWrite\_Ex**,**

**output** **reg** RegWrite\_M**,**

**input** MemtoReg\_Ex**,**

**output** **reg** MemtoReg\_M**,**

**input** MemWrite\_Ex**,**

**output** **reg** MemWrite\_M**);**

**always@(posedge** clk**)**

**begin**

**if** **(**rst**)** **begin**

ALUResult\_M **<=** 0**;**

WriteData\_M **<=** 0**;**

writereg\_M **<=** 0**;**

RegWrite\_M **<=** 0**;**

MemtoReg\_M **<=** 0**;**

MemWrite\_M **<=** 0**;**

**end**

**else** **begin**

ALUResult\_M **<=** ALUResult\_Ex**;**

WriteData\_M **<=** WriteData\_Ex**;**

writereg\_M **<=** writereg\_Ex**;**

RegWrite\_M **<=** RegWrite\_Ex**;**

MemtoReg\_M **<=** MemtoReg\_Ex**;**

MemWrite\_M **<=** MemWrite\_Ex**;**

**end**

**end**

**endmodule**

`timescale 1ns**/**1ns

**module** M\_WB**(input** clk**,**

**input** rst**,**

**input** **[**31**:**0**]** ReadData\_M**,**

**output** **reg** **[**31**:**0**]** ReadData\_WB**,**

**input** **[**31**:**0**]** ALUResult\_M**,**

**output** **reg** **[**31**:**0**]**ALUResult\_WB**,**

**input** **[**4**:**0**]**writereg\_M**,**

**output** **reg** **[**4**:**0**]**writereg\_WB**,**

**input** RegWrite\_M**,**

**output** **reg** RegWrite\_WB**,**

**input** MemtoReg\_M**,**

**output** **reg** MemtoReg\_WB**);**

**always@(posedge** clk **)**

**begin**

**if** **(**rst**)** **begin**

ReadData\_WB **<=** 0**;**

ALUResult\_WB **<=** 0**;**

writereg\_WB **<=** 0**;**

RegWrite\_WB **<=** 0**;**

MemtoReg\_WB **<=** 0**;**

**end**

**else** **begin**

ReadData\_WB **<=** ReadData\_M**;**

ALUResult\_WB **<=** ALUResult\_M**;**

writereg\_WB **<=** writereg\_M**;**

RegWrite\_WB **<=** RegWrite\_M**;**

MemtoReg\_WB **<=** MemtoReg\_M**;**

**end**

**end**

**endmodule**

`include "adder.v"

`include "alu32.v"

`include "flopr\_param.v"

`include "mux2.v"

`include "mux3.v"

`include "regfile32.v"

`include "signext.v"

`include "sl2.v"

`include "EX\_M.v"

`include "ID\_EX.v"

`include "IF\_ID.v"

`include "M\_WB.v"

`include "forwardingunit.v"

`include "hazardunit.v"

`timescale 1ns**/**1ns

**module** Datapath**(input** clk**,**

**input** reset**,**

**input** RegDst\_ID**,**

**input** RegWrite\_ID**,**

**input** ALUSrc\_ID**,**

**input** B**,**

**input** Jump\_ID**,**

**input** MemtoReg\_ID**,**

**input** MemWrite\_ID**,**

**input** Branch\_ID**,**

**input** **[**3**:**0**]** ALUControl\_ID**,**

**input** **[**31**:**0**]** ReadData\_M**,**

**input** **[**31**:**0**]** Instr\_IF**,**

**output** MemWrite\_M**,**

**output** **[**31**:**0**]** Instr\_ID**,**

**output** **[**31**:**0**]** PC\_IF**,**

**output** **[**31**:**0**]** WriteData\_M**,**

**output** **[**31**:**0**]** ALUResult\_M**);**

**wire** **[**31**:**0**]** PCNEXT\_IF**,** PCplus4\_IF**,** PCplus4\_ID**;**

**wire** **[**31**:**0**]** PCBranch\_ID**,**PCbeforeBranch**;**

**wire** **[**31**:**0**]** extendedimm\_ID**,** extendedimm\_Ex**,** extendedimmafter**;**

**wire** **[**31**:**0**]** dataone\_ID **,**dataone\_Ex**;**

**wire** **[**31**:**0**]** WriteData\_ID**,** WriteData\_Ex**;**

**wire** **[**31**:**0**]** ALUResult\_Ex**,** ALUResult\_WB**,** ALUResult\_Mem**;**

**wire** **[**31**:**0**]** MUXresult\_WB**,** aluop2**,** SrcA\_EX**,** SrcB\_EX**;**

**wire** **[**31**:**0**]** ReadData\_WB**;**

**wire** **[**4**:**0**]** writereg\_Ex**,** writereg\_M**,** writereg\_WB **;**

**wire** ZeroFlag\_Ex**;**

**wire** **[**31**:**0**]** Instr\_Ex**;**

**wire** RegWrite\_Ex**,** RegWrite\_M**,** RegWrite\_WB**;**

**wire** MemtoReg\_Ex**,** MemtoReg\_M**,** MemtoReg\_WB**;**

**wire** MemWrite\_Ex**;**

//wire MemWrite\_M;

**wire** **[**3**:**0**]** ALUControl\_Ex**;**

**wire** ALUSrc\_Ex**;**

**wire** RegDst\_Ex**;**

**wire** **[**1**:**0**]** ForwardAE**,**ForwardBE**;**

**wire** ForwardAD**,** ForwardBD**;**

**wire** Flush\_Ex**,** Stall\_IF**,** Stall\_ID**;**

**wire** BranchMUXselect**,** Equal\_ID**;**

**wire** **[**31**:**0**]** equalone**,**equaltwo**;**

// Fetch

flopr\_param **#(**32**)** PCregister**(**clk**,** reset**,!**Stall\_IF **,**PC\_IF**,** PCNEXT\_IF**);**

adder **#(**32**)** pcadd4**(**PC\_IF**,** 32'd4 **,** PCplus4\_IF**);**

**assign** BranchMUXselect **=** **((**B **^** Equal\_ID**)** **&** Branch\_ID**);**

mux2 **#(**32**)** branchmux**(**PCplus4\_IF **,** PCBranch\_ID**,** BranchMUXselect **,** PCNEXT\_IF**);**

// IF\_ID

IF\_ID Fetch\_Decode\_Buffer**(**clk**,**reset **|** BranchMUXselect **|** Jump\_ID **,**Stall\_ID**,**PCplus4\_IF**,**PCplus4\_ID**,**Instr\_IF**,**Instr\_ID**);**

// Decode

signext immextention**(**Instr\_ID**[**15**:**0**],**extendedimm\_ID**);**

slt2 shifteradd2**(**extendedimm\_ID**,**extendedimmafter**);**

registerfile32 RF**(**clk**,**RegWrite\_WB**,** reset**,** Instr\_ID**[**25**:**21**],** Instr\_ID**[**20**:**16**],** writereg\_WB**,** MUXresult\_WB**,** dataone\_ID**,**WriteData\_ID**);**

mux2 **#(**32**)** equalonemux**(**dataone\_ID**,**ALUResult\_Mem**,**ForwardAD**,**equalone**);**

mux2 **#(**32**)** equaltwomux**(**WriteData\_ID**,**ALUResult\_Mem**,**ForwardBD**,**equaltwo**);**

**assign** Equal\_ID **=** **(**equalone**==**equaltwo**);**

adder **#(**32**)** pcaddsigned**(**extendedimmafter**,** PCplus4\_ID**,** PCBranch\_ID**);**

// ID\_EX

ID\_EX Decode\_Execute\_Buffer**(**clk**,** reset **,** dataone\_ID**,** dataone\_Ex**,**WriteData\_ID**,**WriteData\_Ex**,** extendedimm\_ID**,**extendedimm\_Ex**,** Instr\_ID**,**Instr\_Ex**,** RegWrite\_ID**,** RegWrite\_Ex**,**

MemtoReg\_ID**,** MemtoReg\_Ex**,** MemWrite\_ID**,**MemWrite\_Ex**,** ALUControl\_ID**,** ALUControl\_Ex**,** ALUSrc\_ID**,** ALUSrc\_Ex**,** RegDst\_ID**,** RegDst\_Ex**);**

// Execute

mux3 forwardmuxA **(**dataone\_Ex**,** MUXresult\_WB**,** ALUResult\_Mem**,** ForwardAE**,** SrcA\_EX**);**

mux3 forwardmuxB **(**WriteData\_Ex**,** MUXresult\_WB**,** ALUResult\_Mem**,** ForwardBE**,** aluop2**);**

alu32 alucomp**(**SrcA\_EX**,** SrcB\_EX**,** ALUControl\_Ex**,** Instr\_Ex**[**10**:**6**],** ALUResult\_Ex**,** ZeroFlag\_Ex**);**

mux2 **#(**32**)** aluop2sel**(**aluop2**,**extendedimm\_Ex**,** ALUSrc\_Ex**,** SrcB\_EX**);**

mux2 **#(**5**)** writeopmux**(**Instr\_Ex**[**20**:**16**],**Instr\_Ex**[**15**:**11**],**RegDst\_Ex**,** writereg\_Ex**);**

// EX\_M

EX\_M Execute\_Memory\_Buffer**(**clk**,** reset**,** ALUResult\_Ex**,** ALUResult\_Mem**,** aluop2**,** WriteData\_M**,** writereg\_Ex**,** writereg\_M**,**

RegWrite\_Ex**,** RegWrite\_M**,** MemtoReg\_Ex**,** MemtoReg\_M**,** MemWrite\_Ex**,** MemWrite\_M **);**

**assign** ALUResult\_M **=** ALUResult\_Mem**;**

// Memory

// Forwarding Unit

forwardingunit Forward\_Unit**(** Instr\_Ex **[**25**:**21**],** Instr\_Ex **[**20**:**16**],** Instr\_ID **[**25**:**21**],** Instr\_ID **[**20**:**16**],** writereg\_M**,** writereg\_WB**,** RegWrite\_M**,** RegWrite\_WB**,** ForwardAE**,** ForwardBE**,** ForwardAD**,** ForwardBD**);**

hazardunit hazard\_unit**(**Instr\_Ex **[**20**:**16**],** Instr\_ID **[**25**:**21**],** Instr\_ID **[**20**:**16**],** writereg\_M**,**writereg\_Ex**,**MemtoReg\_Ex**,**MemtoReg\_M**,**RegWrite\_Ex**,**Branch\_ID**,**Jump\_ID**,**

Stall\_IF**,**Stall\_ID**,**Flush\_Ex **);**

// M\_WB

M\_WB Memory\_WriteBack\_Buffer**(**clk**,**reset**,** ReadData\_M**,** ReadData\_WB**,** ALUResult\_M**,** ALUResult\_WB**,** writereg\_M**,** writereg\_WB**,**

RegWrite\_M**,** RegWrite\_WB**,** MemtoReg\_M**,** MemtoReg\_WB**);**

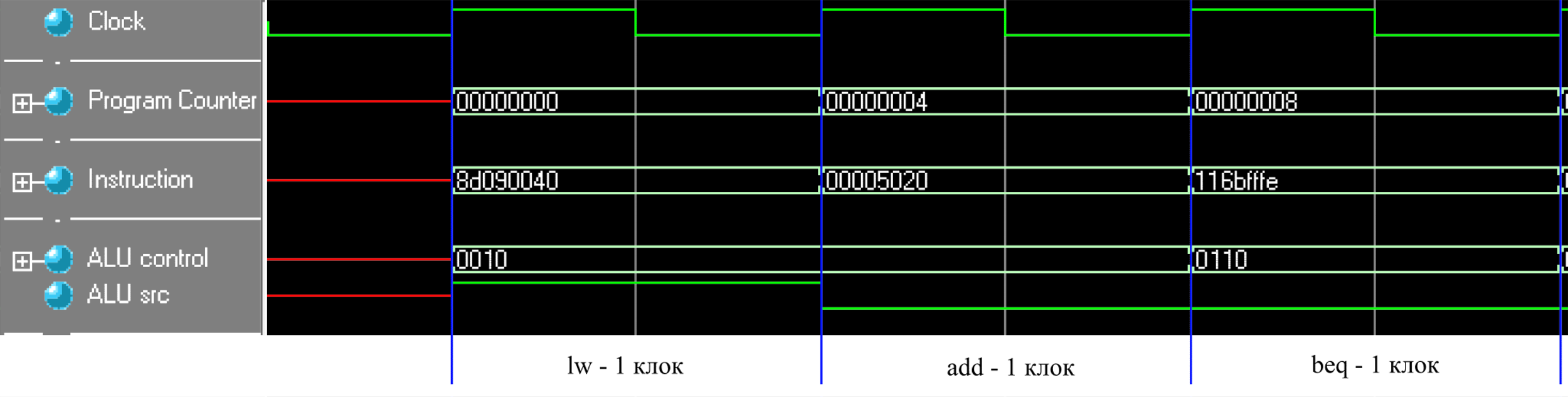
// WriteBack

mux2 **#(**32**)** resultmux**(**ALUResult\_WB**,** ReadData\_WB**,** MemtoReg\_WB**,** MUXresult\_WB**);**

**endmodule**

**Нэг циклт болон олон циклт гүйцэтгэл**

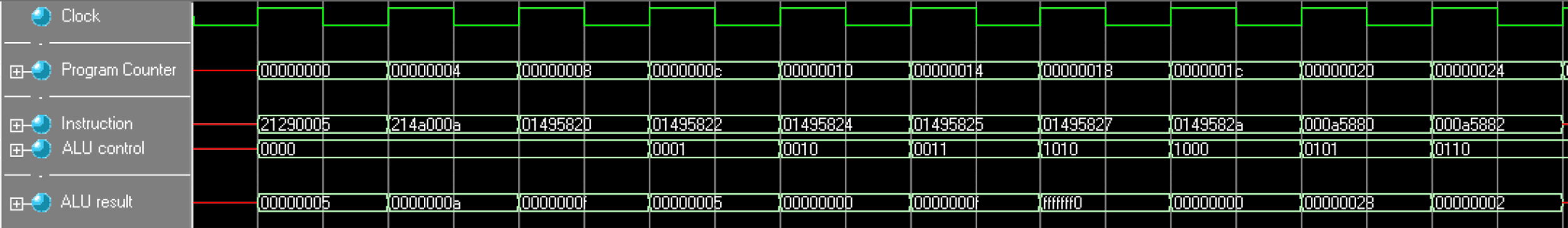
|  |  |
| --- | --- |
| **loop:**  **lw** $t1, 64($t0)  **add** $t2, $0, $0  **beq** $t3, $t3, **loop** | 8d090040  00005020  116bfffe |

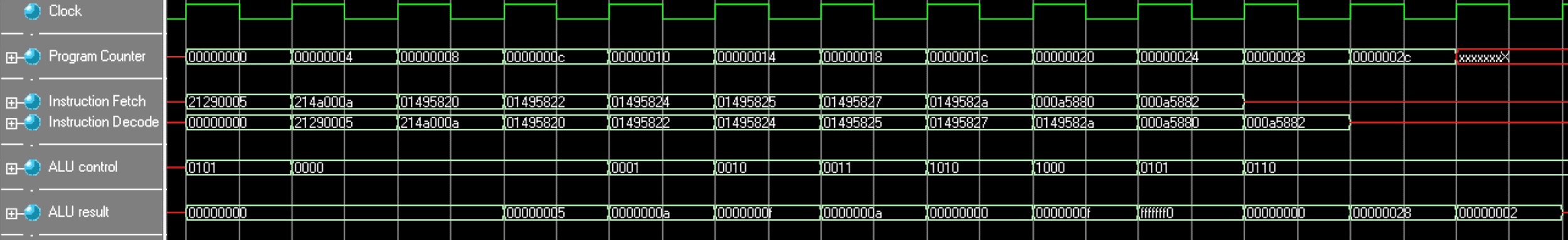




**Нэг циклт болон пайплайн гүйцэтгэл**

|  |  |
| --- | --- |
| **addi** $t1, 5  **addi** $t2, 10  **add** $t3, $t2, $t1  **sub** $t3, $t2, $t1  **and** $t3, $t2, $t1  **or** $t3, $t2, $t1  **nor** $t3, $t2, $t1  **slt** $t3, $t2, $t1  **sll** $t3, $t2, 2  **srl** $t3, $t2, 2 | 21290005  214a000a  01495820  01495822  01495824  01495825  01495827  0149582a  000a5880  000a5882 |





**Пайплайн гүйцэтгэл – hazard**

|  |  |
| --- | --- |
| loop:  ;data hazard  **lw** $t0, 0($t1)  **addi** $t1, $t0, 5  **addi** $t2, $zero, 2  **addi** $t3, $zero, 3  ;control hazard  **beq** $t1, $t2, loop  **addi** $t3, $zero, 5  **addi** $t2, $zero, 5  ;control hazard  **beq** $t1, $t2, loop  **addi** $t3, $zero, 1  **addi** $t2, $zero, 1  **addi** $t1, $zero, 1 | 8d280000  21090005  200a0002  200b0003  112afffc  200b0005  200a0005  112afff9  200b0001  200a0001  20090001 |

