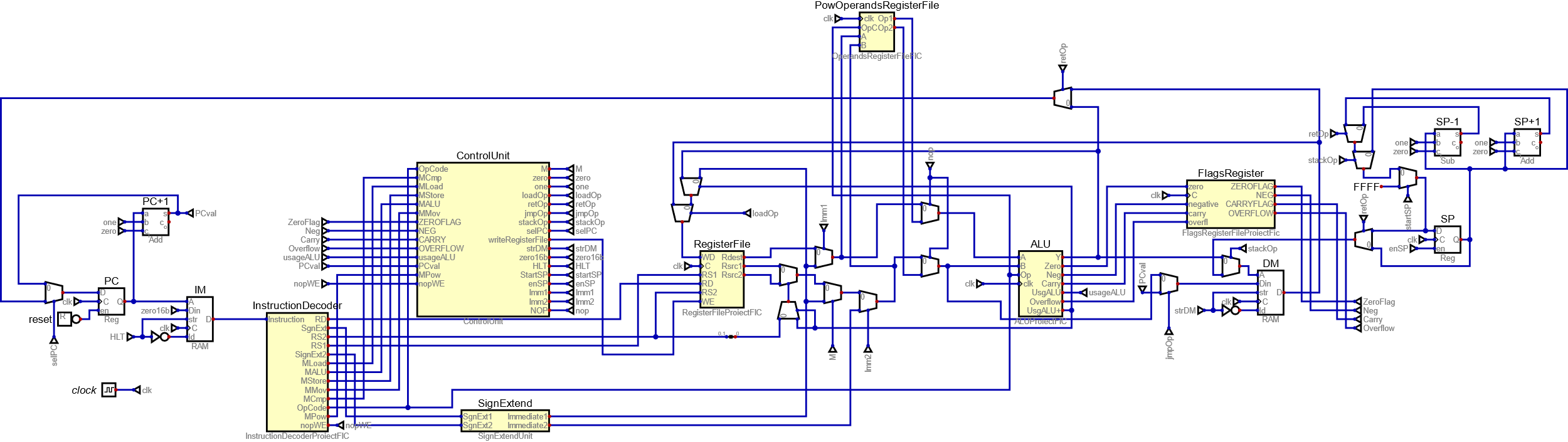
**DOCUMENTAȚIE HARDWARE**

**Procesorul**



Procesorul pe care l-am dezvoltat este bazat pe arhitectura MIPS single-cycle datapath. Acesta funcționează astfel:

* Instrucțiunea este fetch-uită din Instruction Memory de la adresa indicată de conținutul registrului Program Counter
* Instrucțiunea trece printr-un modul decodificator, care extrage informații necesare Control Unit-ului, dar și direcționează datele pe magistralele corespunzătoare
* Control Unit-ul generează semnale de control necesare multiplexării datelor înainte de a ajunge la ALU, scrierii în regiștrii etc.
* Dacă se lucrează cu valori immediate, acestea vor ajunge în Sign Extend Unit
* Dacă se lucrează cu regiștrii, adresele corespunzătoare vor fi trimise către Register File, care va pune la dispoziție conținutul curent al regiștrilor indicați
* Operanzii, împreună cu codul operației ajung la ALU, care este responsabil de efectuarea calculelor aritmetice și de operațiile logice
* ALU actualizează Flags Register-ul cu valorile corecte ale fanioanelor
* Dacă este cazul, rezultatul produs de ALU este stocat în Data Memory
* Dacă lucrăm cu stiva, registrul Stack Pointer va suferi și el modificări
* În final, după caz, rezultatul din ALU sau conținutul citit din Data Memory este scris înapoi în Register File, în registrul corespunzător destinației

**Program Counter**

A diagram of a computer circuit

Description automatically generated

Este un registru care conține adresa următoarei instrucțiuni ce trebuie citită din Instruction Memory. Intrarea sa este multiplexată, deoarece fie avem nevoie să incrementăm valoarea actuală, adică să citim instrucțiunea următoare, fie dorim să efectuăm un salt, caz în care noua adresă ce trebuie indicată de Program Counter se încarcă din ALU/Data Memory.

Semnalul de control selPC este activat când se efectuează instrucțiuni de salt.

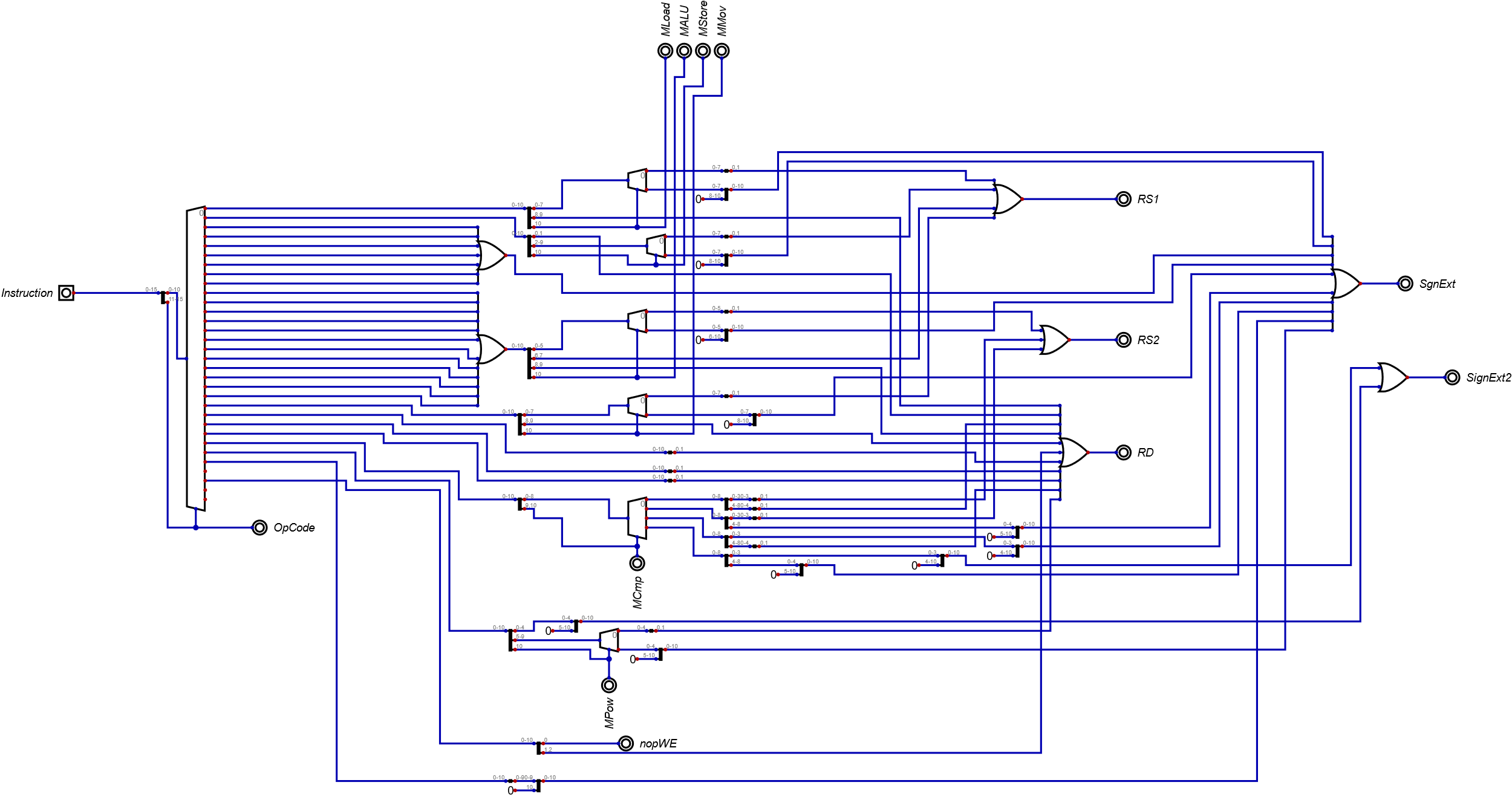
**Instruction Memory**

A diagram of a circuit

Description automatically generated

Memoria de instrucțiuni este implementată utilizând un modul de memorie RAM pe 16 biți, cu adrese pe 16 biți. Adresa este încărcată din Program Counter, iar semnalul de control HLT, generat la efectuarea unei instrucțiuni de halt, activează portul str al memoriei, lucru ce determină la ieșirea D starea în impedanță ridicată Z. Astfel, la întâlnirea lui HLT, procesorul nu trimite mai departe nicio instrucțiune.

**Instruction Decoder**



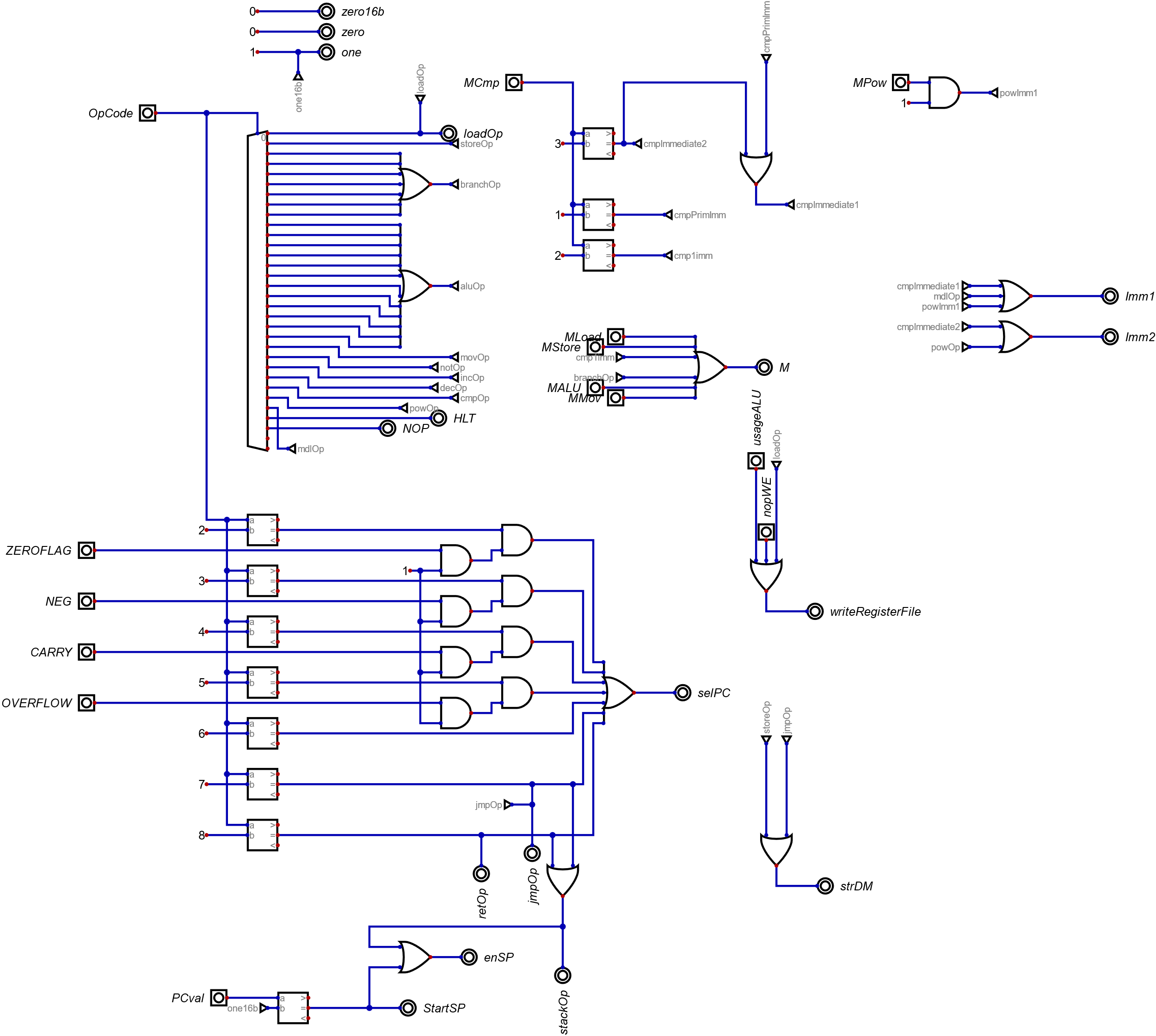
A diagram of a decoder

Description automatically generated

Acest modul preia instrucțiunea pe 16 biți furnizată de Instruction Memory și o decodifică. Cei mai semnificativi 5 biți sunt codul operației, restul de 11 biți trebuind interpretați în funcție de acesta. Din acest motiv, pe linia de selecție a demultiplexorului se află chiar codul operației. Fiecare operație este decodificată conform documentului “Formatele instrucțiunilor”.

Spre exemplu, instrucțiunea cu codul 0 este cea de load. Cei 11 biți sunt împărțiți într-un bit de mod (lucrul cu regiștrii/lucrul cu valori immediate), 2 biți de adresă a registrului destinație în care se va efectua load-ul și 8 biți care reprezintă fie adresa unui registru din care se extrage conținutul ce e interpretat ca adresă pentru Data Memory, fie o valoare immediate ce reprezintă tot adresa pentru Data Memory, lucru decis de bitul de mod. De aceea, cei 8 biți sunt demultiplexați în funcție de valoarea bitului de mod și apoi trimiși pe magistrala corespunzătoare. În mod analog se interpretează și restul instrucțiunilor.

**Control Unit**

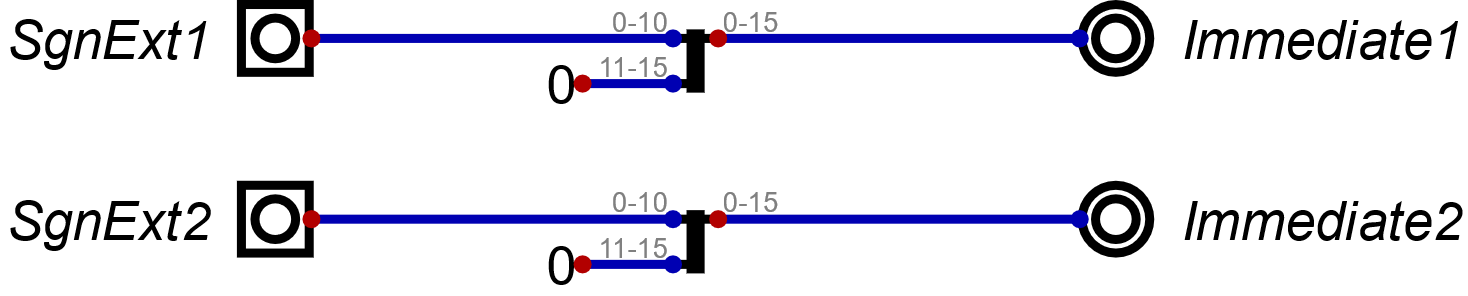


A computer screen shot of a computer

Description automatically generated

Unitatea de control este pur combinațională. În interiorul său se verifică anumite combinații de semnale de intrare și se generează valorile corespunzătoare semnalelor de ieșire, care sunt folosite mai departe în locurile unde este nevoie de o decizie condiționată.

**Sign Extend Unit**

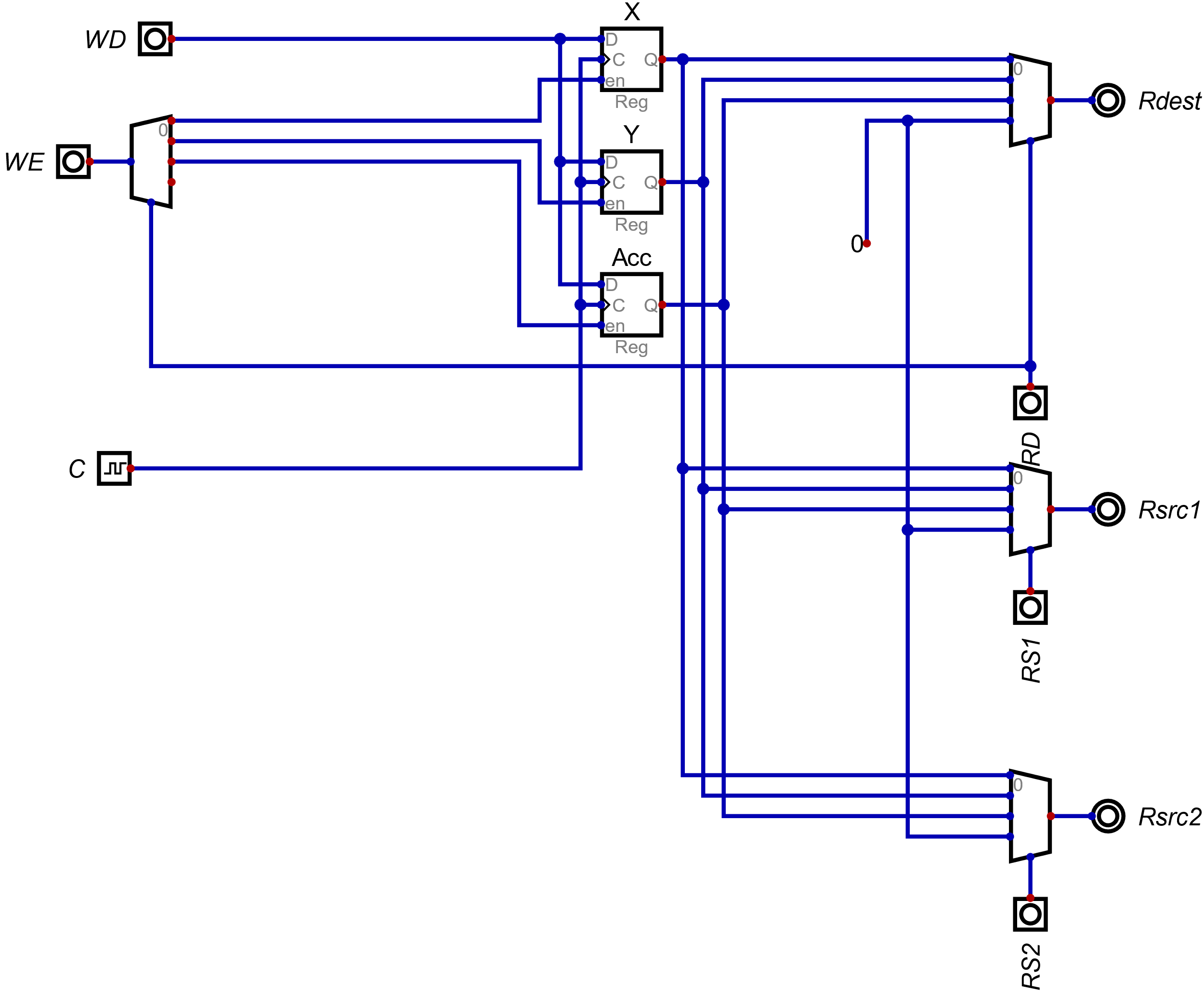


A close-up of a sign

Description automatically generated

Sign Extend Unit-ul are 2 intrări și 2 ieșiri deoarece avem câteva instrucțiuni ce necesită (în unele cazuri) ca ambii operanzi să fie dați de utilizator (CMP și POW). Ieșirile Sign Extend-ului sunt aduse la 16 biți și direcționate către multiplexoare aflate pe magistralele ce duc spre operanzii ALU și sunt alese în funcție de valorile biților de mod din interiorul instrucțiunilor.

**Register File**

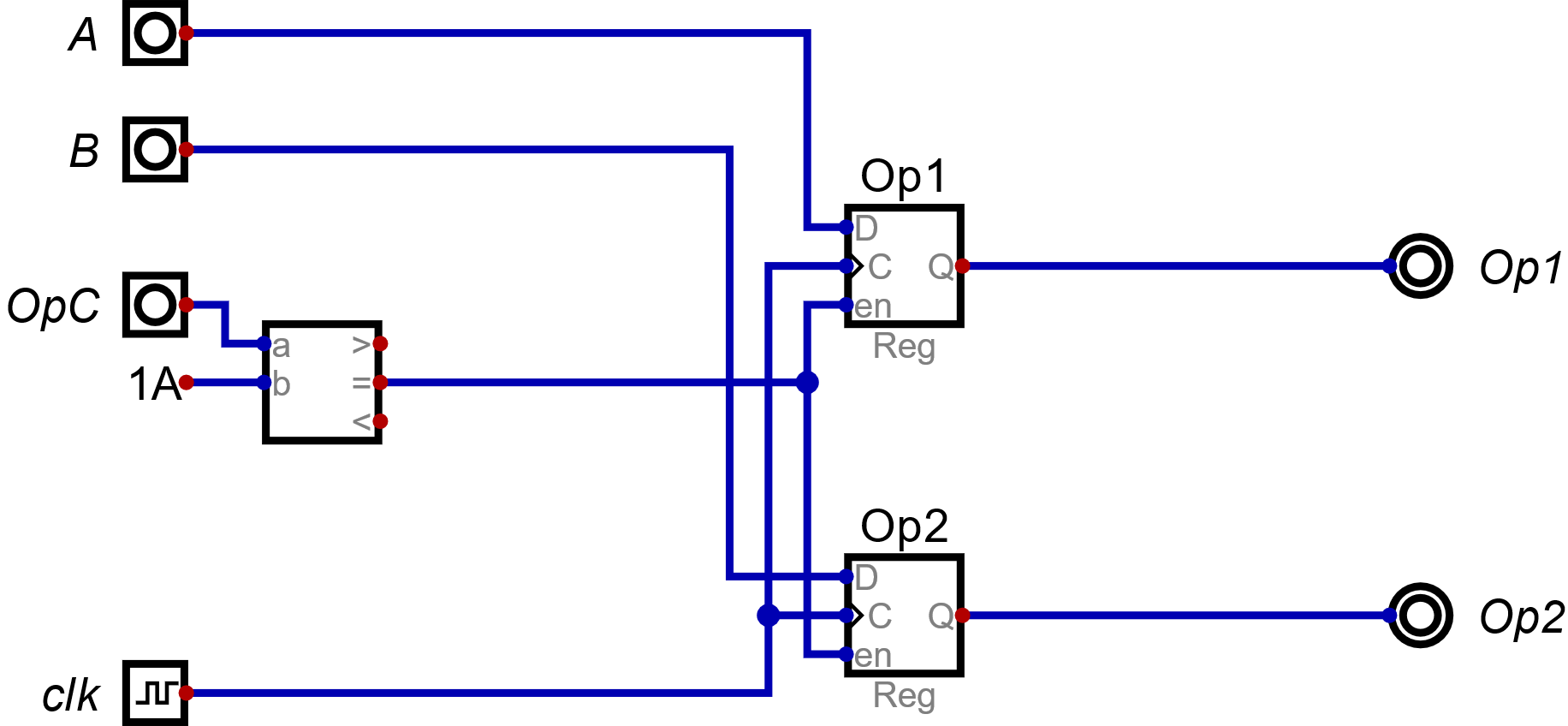


A diagram of a computer

Description automatically generated

Register File-ul conține cei 3 regiștrii cu care procesorul lucrează: X, Y și Acumulatorul. Intrarea WD reprezintă datele ce sunt pregătite să fie scrise în regiștrii. Când intrarea WE este activă, înseamnă că se va efectua o scriere în registrul selectat de intrarea RD (adresa registrului destinație). Ieșirile acestui modul reprezintă conținutul din momentul curent al regiștrilor selectați de intrările RD, RS1 și RS2.

**Operands Register File**

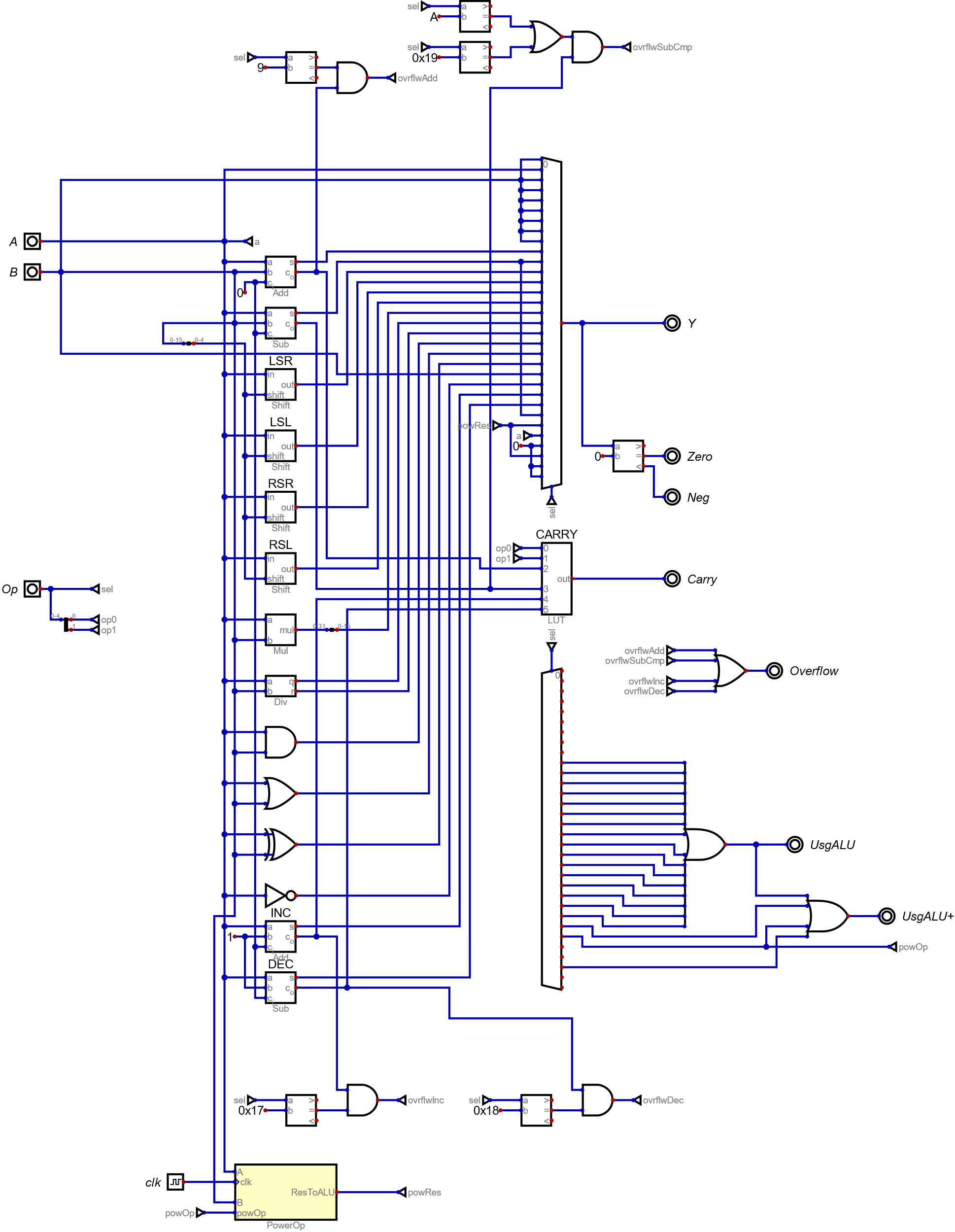


A diagram of a computer

Description automatically generated

Acest modul este folosit în cadrul operației de ridicare la putere. El reține valorile operanzilor trimiși către ALU, doar în cazul în care codul operației este cel al ridicării la putere. Acest lucru este necesar deoarece în interiorul ALU există un modul ce calculează rezultatul ridicării la putere și este construit astfel încât să aibă nevoie de valorile corecte până la finalizarea lui.

**ALU (Arithmetic-Logic Unit)**

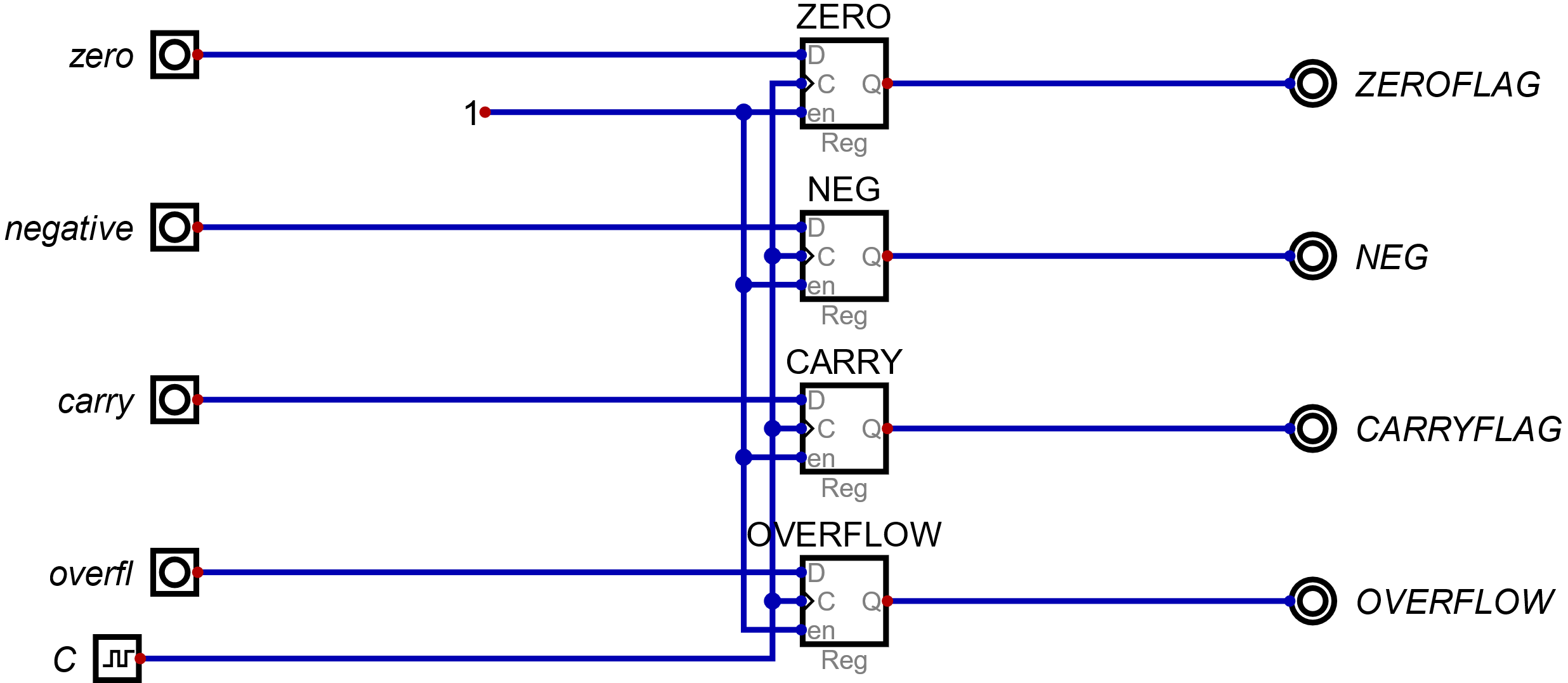


A diagram of a computer chip

Description automatically generated

ALU reprezintă “inima” procesorului. Aici au loc toate calculele aritmetice și logice. La intrare se află operanzii A și B cu care urmează să se realizeze calculele, în timp ce ieșirea este multiplexată în funcție de codul operației pentru a furniza rezultatul corect. De asemenea, ALU generează și 4 flag-uri: zero, negative, carry și overflow. Pe lângă acestea, mai sunt generate semnalele UsgALU, care marchează operațiile ce se încheie cu o scriere în registru, precum și UsgALU+, care pe lângă operațiile menționate, mai marchează și operații precum CMP și POW deoarece rezultatele lor sunt nevoite să treacă de anumite multiplexoare.

**Flags Register File**



A yellow rectangular with blue dots and black text

Description automatically generated

Acest Register File este menit să stocheze în orice moment de timp valoarea curentă a fiecărui fanion setat de către ALU, pentru a putea furniza starea corectă a acestora în cazul instrucțiunilor de salt condiționat.

**Data Memory**

A diagram of a device

Description automatically generated

Memoria de date este utilizată pentru a stoca/citi date din/în regiștrii și conține stiva, care începe cu ultima adresă disponibilă (0xFFFF) și “crește” în sens invers în memorie. Pe intrarea A (adresă) se va alege, în funcție de semnalul stackOp ce marchează efectuarea unei operații cu stiva, fie o adresă de la ALU (pentru load/store), fie o adresă de la Stack Pointer (pentru jump/return). Pe intrarea Din (date) vine fie valoarea corectă ce trebuie stocată, fie valoarea Program Counterului în cazul unei operații de jump. În memoria de date se scrie doar când semnalul de control strDM este activat de Control Unit.

**Stack Pointer**

A diagram of a computer

Description automatically generated

Registrul Stack Pointer reține adresa vârfului stivei. La început, este setat pe valoarea 0xFFFF, în continuare incrementându-se la o operație de return, respectiv decrementându-se la o operație de jump. Dacă nu are loc nicio operație cu stiva, conținutul registrului rămâne neschimbat.

În cazul unei operații de return, s-a observat că adresa din Stack Pointer se actualiza prea târziu, așa că având deja rezultatul corect la intrarea registrului, am înaintat valoarea aceasta prin intermediul unui multiplexor către portul de adresă a memoriei de date, astfel citind valoarea corectă stocată pe stivă.