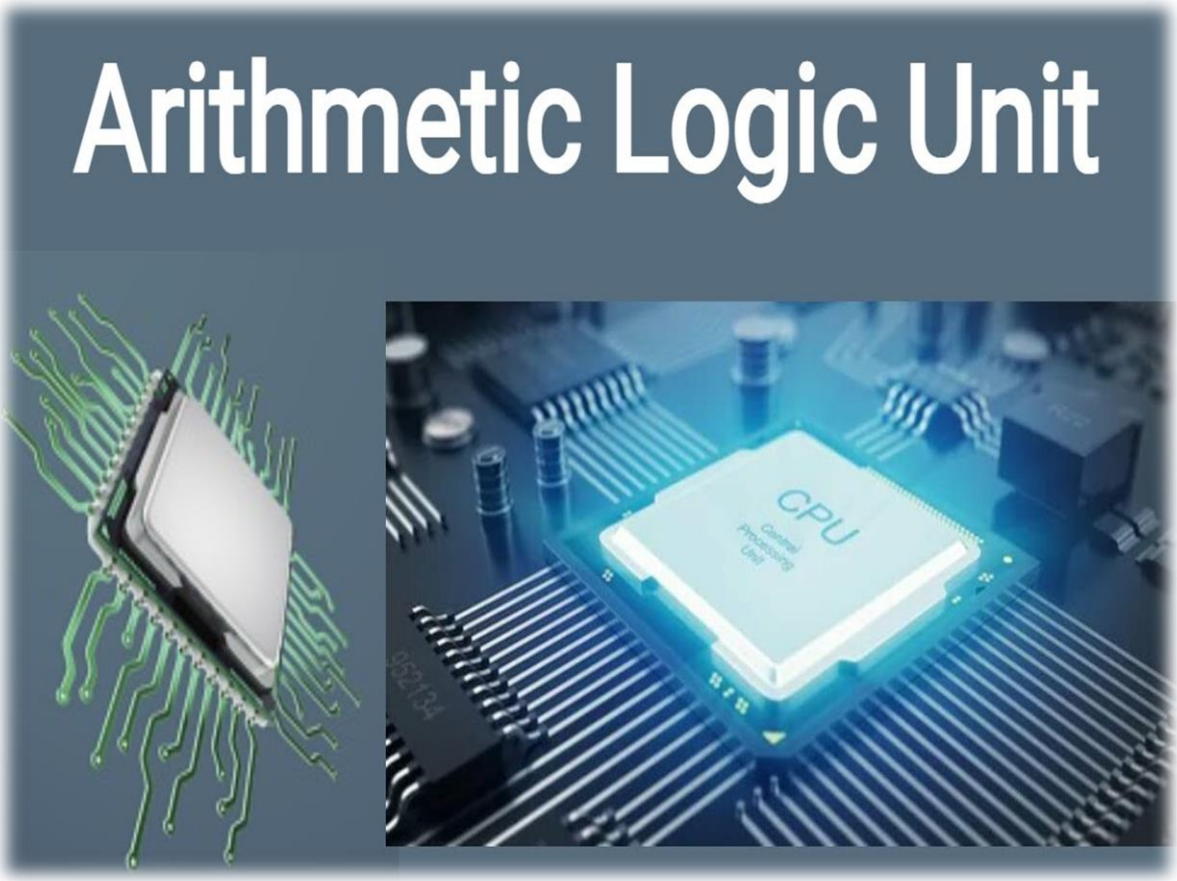
Unitatea Aritmetico-Logica
(ALU)



Poponet Tiberiu-Sergiu

Grupa 30231/2

Cuprins:

1.Introducere.....3

1.1Context.....3

1.2Specificatii.....3

1.3Obiective.....3

2.Studiu bibliografic.....4

3.Analiza si componente.....4

4.Design.....9

4.Implementare.....9

5.Testare si validare.....12

6.Concluzii.....14

7.Bibliografie.....15

Proiectarea unei unitati aritmetico-logice (UAL)

1. Introducere

1.1 Context

Unitatea Aritmetico-Logica (UAL) este una dintre principalele componente ale arhitecturii unui calculator și are un rol fundamental în efectuarea operațiilor aritmetice și logice asupra datelor. Această componentă este esențială pentru funcționarea corectă a unui procesor și pentru realizarea calculelor complexe într-un calculator modern. În continuare, vom explora în detaliu ce este UAL, cum funcționează și cum se încadrează în arhitectura generală a unui calculator.

1.2 Specificatii

Unitatea Aritmetico-Logica (ALU) este o componenta a Unitatii Centrale de Procesare (CPU), cunoscută și sub numele de procesor. Aceasta este componenta centrală a unui calculator sau a unui sistem informatic. CPU-ul îndeplinește un rol crucial în executarea instrucțiunilor și procesarea datelor. ALU este componenta CPU care efectuează operațiile aritmetice și logice. ALU realizează calculele și comparațiile necesare.

O Unitate Aritmetico-Logica poate fi proiectată să execute, în principiu, orice operație. Totuși, cu cât operațiile devin mai complexe UAL devine mai scumpă, ocupă mai mult loc și disipă mai multă căldură. Operațiile care sunt, în general, suportate de toate UAL sunt:

Operații logice: AND, OR, NOT, XOR, NOR, NAND, etc.

Operații de shift: shift stânga, shift dreapta, shift circular, etc.

Operații aritmetice: adunare, scădere, înmulțire (nu toate), împărțire (nu toate).

De asemenea, o UAL trebuie să fie în stare la orice moment să ruleze operaţia corespunzătoare la fiecare comandă din partea procesorului. Pentru aceasta fiecare operație are un cod al operației implementat în hardware, astfel încât la o instrucţiune add să intre în funcţiune modulul pentru adunare, la o instrucţiune and să intre în funcţiune poarta AND, etc. (vezi MIPS)

1.3 Obiective

1. **Înțelegerea Conceptelor de Arhitectură a Calculatorului**: Obiectivul de bază este dezvoltarea unei înțelegeri solide a conceptelor de bază legate de arhitectura calculatorului, în special a funcționării Unității Aritmetico-Logice.

2**. Proiectarea și Implementarea unei ALU Funcționale**: Obiectivul principal este proiectarea și implementarea unei Unități Aritmetico-Logice funcționale, capabilă să efectueze operații aritmetice și logice de bază.

3. **Soluții Eficiente de Design**: Un alt obiectiv ar putea fi găsirea unor soluții eficiente de design pentru ALU, care să minimizeze consumul de energie și să optimizeze performanța.

4. **Documentarea**: documentarea despre proiectul în mod corespunzător astfel incat să poata fi comunicate clar deciziile de design luate și funcționalitatea ALU-ului.

5**. Testarea și Depanarea**: Un alt obiectiv poate fi să învatarea tesatrii și depanarii ALU-ul pentru a ne asigura că acesta funcționează corect, cat si identificarea și remedierea eventualelor erori.

2. Studiu Bibliografic

O ALU constă din trei tipuri de părți funcționale: registre de stocare, logica de operații și logica de secvențiere, ca prezentate în Fig. 1. Intrările și ieșirile ALU sunt conectat la alte unități funcționale ale CPU, cum ar fi memoria cache și unitatea de executie si control.

Unitatea logică aritmetică (ALU) este inima oricărui procesor. Un ALU efectuează trei tipuri de operații, de exemplu:

Operații aritmetice precum adunarea/scăderea etc.,

Operații logice precum AND, SAU etc.,

Operațiuni de mutare a datelor, cum ar fi shiftarea la stanga sau dreapta.

Sarcina principală a ALU este de a procesa datele stocate în memoria RAM a computerului. În plus, o unitate logică aritmetică este capabilă să producă semnale de control care direcționează un calculator să selecteze calea corectă pentru a efectua procesul de calcul necesar, în funcție de tipurile de date rezultate. Toate operațiunile implică circuite electronice, fiecare dintre ele fiind structurate în mii de elemente. Astfel de panouri sunt de obicei rapide și au o densitate mare.

În funcție de semnalele introduse, unitățile ALU execută diferite tipuri de operații cu două numere. Orice unitate logică aritmetică a calculatorului asigură implementarea a patru acțiuni de bază, schimburi, precum și transformări logice. Setul de operațiuni ALU este principala sa caracteristică.

Datele sunt încărcate înaintea operației în registre. Aceste registre sunt conectate prin căi de date la UAL. În urma realizarii operatiei, rezultatul este stocat intr-un alt registru.

3. Analiza si componente

Unitatea Aritmetico-Logica este formata din mai multe componente:

Unitatea aritmetica: responsabila pentru operațiile aritmetice precum adunarea, scăderea, înmulțirea și împărțirea, shiftare la dreapta si shiftare la stanga.

Unitatea logica: responsabila pentru operațiuni logice precum AND, OR, NOT, NAND și XOR.

Registre: Acestea sunt folosite pentru a stoca rezultatele de intrare, de ieșire și intermediare.

De asemenea, avem nevoie de un Control Unit, care pune laolalta toate operatiile si selecteaza operatia dorita.

Analiza operatiilor realízate:

Operatiile logice:

Operația "OR" (SAU):

Rezultat: Această operație returnează valoarea "adevărat" (1) dacă cel puțin una dintre intrările (A sau B) este "adevărată" (1). În caz contrar, rezultatul este "fals" (0).

Exemplu:

A = 1, B = 0 → A OR B = 1 ; A = 0, B = 0 → A OR B = 0

Operația "AND" (ȘI):

Rezultat: Această operație returnează valoarea "adevărat" (1) numai dacă ambele intrări (A și B) sunt "adevărate" (1). În caz contrar, rezultatul este "fals" (0).

Exemplu:

A = 1, B = 1 → A AND B = 1 ; A = 1, B = 0 → A AND B = 0

Operația "NOT" (NEGAȚIE):

Rezultat: Această operație neagă intrarea, adică dacă intrarea este "adevărată" (1), atunci rezultatul va fi "fals" (0), și invers. Este o operație unară, deoarece acționează doar asupra unei intrări.

Exemplu:

A = 1 → NOT A = 0 ; A = 0 → NOT A = 1

Operația "NAND" (ȘI NU):

Rezultat: Această operație realizează negarea și conjuncția simultan. Cu alte cuvinte, rezultatul este "fals" (0) doar atunci când ambele intrări sunt "adevărate" (1); în orice alt caz, rezultatul este "adevărat" (1).

Exemplu:

A = 1, B = 1 → A NAND B = 0 ; A = 1, B = 0 → A NAND B = 1 ;

Operația "XOR" (SAU EXCLUSIV):

Rezultat: Această operație returnează valoarea "adevărat" (1) dacă exact una dintre intrările (A sau B) este "adevărată" (1), dar nu ambele. În caz contrar, rezultatul este "fals" (0).

Exemplu:

A = 1, B = 0 → A XOR B = 1 ; A = 1, B = 1 → A XOR B = 0 ; A = 0, B = 0 → A XOR B = 0

Operatiile aritmetice:

Adunarea (Addition):

Descriere: Adunarea este o operație aritmetică de bază în care două sau mai multe numere sunt adunate pentru a obține un rezultat. Într-o ALU, adunarea se efectuează folosind bitwise (pe fiecare bit în parte, începând cu cel mai puțin semnificativ) și poate necesita și gestionarea depășirilor (overflow).

Funcționare:

Intrările (operanzii) pentru adunare sunt stocate în registre sau locații de memorie. Procesul începe prin adunarea primilor doi biți (cel mai puțin semnificativ) și calcularea unui rezultat și a unui eventual carry-out (dacă suma depășește capacitatea unui singur bit). Acest rezultat și carry-out-ul sunt apoi utilizați pentru a aduna următorii biți (inclusiv carry-ul anterior).

Procesul se repetă pentru toți biții până când toate cifrele au fost adunate.

Gestionarea Depășirilor (Overflow): În timpul adunării, poate apărea o depășire atunci când suma depășește capacitatea numărului de biți disponibili. O ALU trebuie să fie capabilă să detecteze și să gestioneze aceste situații, generând semnale de depășire pentru a indica că rezultatul nu este corect.

Scăderea (Subtraction):

Descriere: Scăderea este o altă operație aritmetică de bază, în care un număr este scăzut din altul pentru a obține un rezultat. Într-o ALU, scăderea se efectuează folosind operații de adunare și negare.

Funcționare:

Intrarea principală (descazut) și intrarea secundară (scazator) sunt stocate în registre sau locații de memorie. Procesul începe prin negarea subtrahendului (prin operația NOT) pentru a obține complementul său fata de doi. Apoi, complementul fata de doi este adunat la descazut, similar cu adunarea obișnuită.

Gestionarea Depășirilor: Gestionarea depășirilor este la fel de importantă în cazul scăderii ca și în cazul adunării.

Înmulțirea (Multiplication):

Descriere: Înmulțirea este o operație aritmetică în care două sau mai multe numere (numite factori) sunt înmulțite pentru a obține un produs. Operația de înmulțire este mai complexă decât adunarea sau scăderea, deoarece implică multiple etape și se efectuează bit cu bit.

Funcționare:

Pentru înmulțire, se folosesc adunări repetate, unde un factor este adunat de mai multe ori cu celălalt factor, iar rezultatele intermediare sunt adunate pentru a obține produsul final.

În fiecare pas, un bit dintr-un factor este înmulțit cu toți biții din celălalt factor și suma este acumulată într-un rezultat intermediar.

Acest proces se repetă pentru fiecare bit din factori, de la cel mai puțin semnificativ bit (LSB) până la cel mai semnificativ bit (MSB).

Inmultirea va fi realizata si prin metoda shift and add.

Împărțirea (Division):

Descriere: Împărțirea este o operație aritmetică în care un număr (numit deimpartit) este împărțit la un alt număr (numit impartitor) pentru a obține un rezultat numit cât. Operația de împărțire este complexă și implică multiple etape.

Funcționare:

În fiecare pas, se încearcă să se potrivească cât mai mare parte din deimpartit în impartitor. Se începe de la cel mai semnificativ bit și se deplasează către LSB.

Se verifică dacă se poate face o scădere a unei părți din deimpartit folosind impartitorul.

Dacă se poate efectua o scădere, se marchează un bit în cat și se actualizează impartitorul pentru pasul următor.

Acest proces se repetă până când toți biții din deimpartit au fost procesați, iar rezultatul final este catul. De asemenea, putem adauga un registru in plus, pentru stocarea catului.

Shiftarea la Stânga (Left Shift):

Descriere: În shiftarea la stânga, biții unui cuvânt de date sunt deplasați spre poziții mai semnificative (mai la stânga). Acest lucru înseamnă că fiecare bit este înlocuit cu bitul de la poziția sa anterioară și bitul de la extremitatea stângă (MSB) este eliminat sau devine zero.

Funcționare:

Cu fiecare pas al shiftării la stânga, fiecare bit din cuvântul de date se deplasează spre stânga cu o poziție. Bitul cel mai din stânga (MSB) este pierdut în timpul acestei operații și este înlocuit cu un bit zero. Shiftarea la stânga poate fi efectuată cu un număr specificat de poziții sau de către un număr variabil de poziții, în funcție de implementare.

Shiftarea la Dreapta (Right Shift):

Descriere: În shiftarea la dreapta, biții unui cuvânt de date sunt deplasați spre poziții mai puțin semnificative (mai la dreapta). Această operație poate fi realizată în mai multe moduri, inclusiv shiftare aritmetică (cu semn) sau shiftare logică (fără semn).

Funcționare:

În shiftarea la dreapta, fiecare bit din cuvântul de date se deplasează spre dreapta cu o poziție. În cazul shiftării aritmetice la dreapta, bitul cel mai din dreapta (LSB) poate fi copiat în extremitatea dreaptă pentru a menține semnul valorii (în cazul numerelor cu semn, precum

cele în complementul pe doi). În cazul shiftării logice la dreapta, bitul cel mai din dreapta este pierdut și este înlocuit cu zero.

Componente:

Ripple Carry Adder

Intrari A,B pe 32b si iesiri S pe 32b si semnalul de carry Cout. Foloseste componenta FullAdder, care este un sumator complet pe 1 bit.

Ripple Borrow Subtractor

Intrari A,B pe 32b si iesiri D pe 32b si semnalul de borrow Bout (bitul de transport peste cel mai semnificativ bit). Foloseste componenta FullAdder pentru a realiza scaderea prin adunarea in complement fata de 2.

Inmultitor

AA și BB sunt intrările de tip vector logic cu lungimea de 16 biți, iar PP este ieșirea de tip vector logic cu lungimea de 32 de biți. Foloseste 4 inmultitoare cu intrari pe 8b si iesiri pe 16b. Rezultatele intermediare sunt convertite și adunate pentru a obține produsul final de 32 de biți.

Inmultitorul cu intrari pe 8 biti descompune operanzii in biti. Se calculează produsele parțiale (p0 până la p7) pentru fiecare bit al operandului B și fiecare bit al operandului A, prin aplicarea operatorului logic "AND" între biții corespunzători (p0(0) reprezintă rezultatul înmulțirii dintre b(0) și a(0)). Fiecare produs parțial este adunat cu produsul parțial corespunzător de pe nivelul anterior, utilizând instanțe ale unei componente FullAdder. La nivelul final, rezultatele adunărilor parțiale sunt stocate în vectorul p de 16 biți, care reprezintă rezultatul final al înmulțirii.

Impartitor

Memoria ROM

memorie ROM de 16 cuvinte de 32 de biți fiecare, adresată cu 4 biți. Valorile stocate în ROM sunt accesate pe baza adresei furnizate și sunt încărcate într-un registrul de ieșire pe un semnal de ceas, atunci când semnalul de activare (en) este 1. Memorarea conține o serie de valori fixe, reprezentând o listă de numere în format hexazecimal, și ieșirea (dout) furnizează valoarea corespunzătoare adresei actuale.

Unitatea de Control

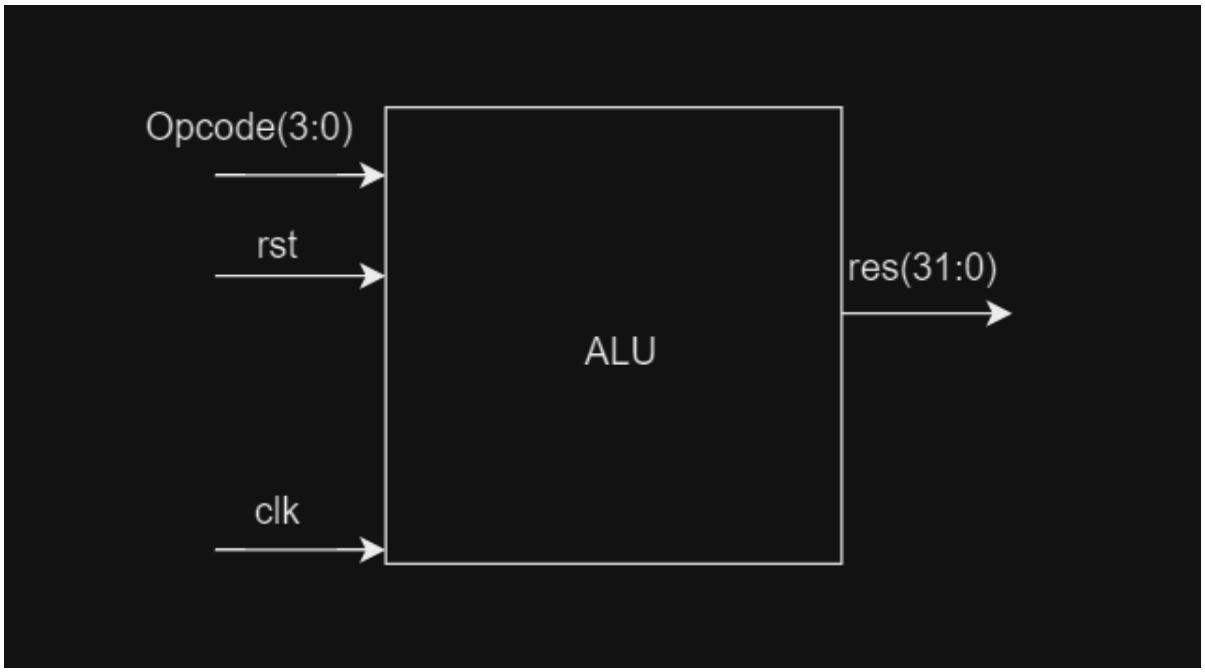
Realizează operații aritmetice și logice în funcție de un cod de operație (opcode) specificat. Modulele componente, precum adunarea cu transport (RippleCarryAdder), scăderea cu transport (RippleCarrySubtractor), și înmulțirea (Inmultitor32b) si impartirea (Impartire) sunt utilizate pentru a realiza operațiile de bază. Unitatea de control primește un opcode, semnale de ceas (clock), operand1 și operand2 (ambele de 32 de biți), și furnizează rezultatul corespunzător în semnalul de ieșire (result).

Operațiile suportate includ adunare, scădere, înmulțire, impartire (nefunctional), operații logice (AND, OR, NOT, XOR), deplasări (stânga și dreapta), incrementare, decrementare, și operația NAND.

ALU (componenta principala)

Componenta are ca intrari: opcode pe 4 biti, clk si rst (reset) si iesirea res pe 32 biti. Două instanțe ale componente RomMemory sunt utilizate pentru a citi două valori de 32 de biți din memorie (rom1 și rom2). Adresele de memorie sunt actualizate în fiecare ciclu de ceas. Componenta ControlUnit primește opcode-urile și valorile stocate în cele două locații de memorie (rom1 și rom2). Aceasta generează rezultatul operației specificate de opcode. Operațiile ALU sunt executate în cicluri de ceas. Adresele de memorie sunt actualizate, iar rezultatul operației este generat în fiecare ciclu de ceas.

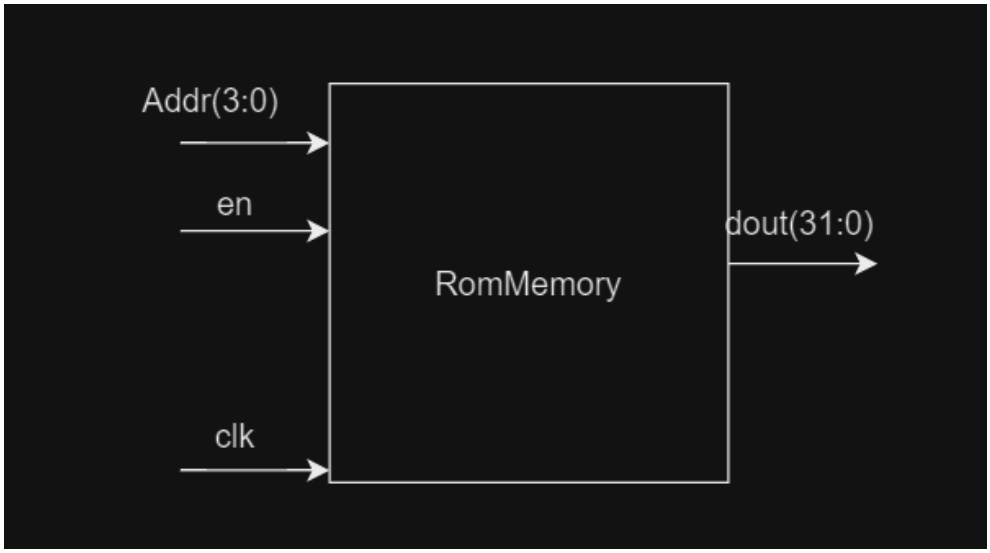
4. Design si Implementare



(ALU 32b Blackbox)

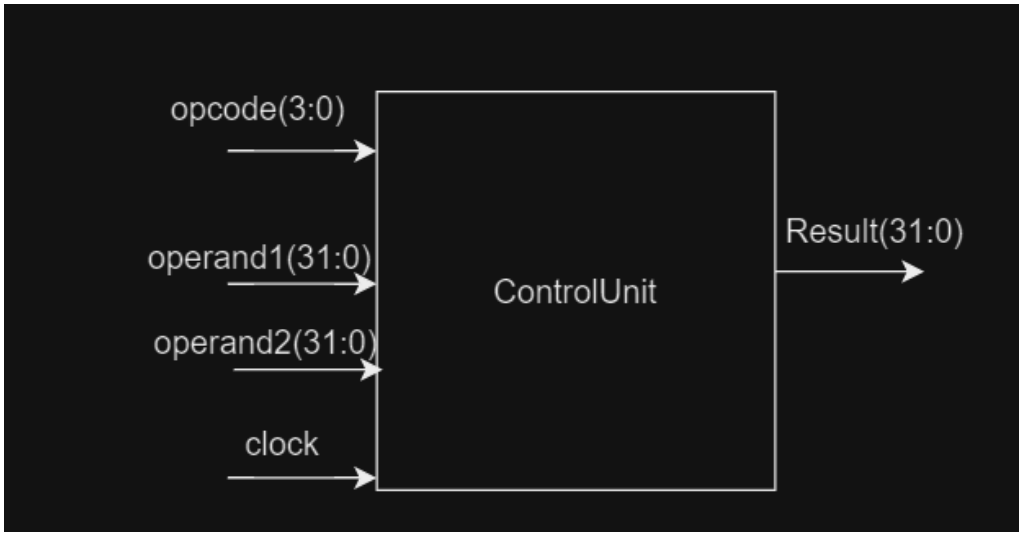
Figura descrie intrarile si iesirile pentru Unitatea Aritmetico-Logica a noastra. Avem ca intrari opcode-ul pe 4 biti, pentru a selecta una dintre cele 11 operatii descrise mai sus,precum si incrementare/decrementare (13 operatii in total) si iesirea res pe 32 de biti.

Pentru a stoca operanzii pe care ii vom folosi, avem nevoie de o memorie ROM de 16x32 biti.



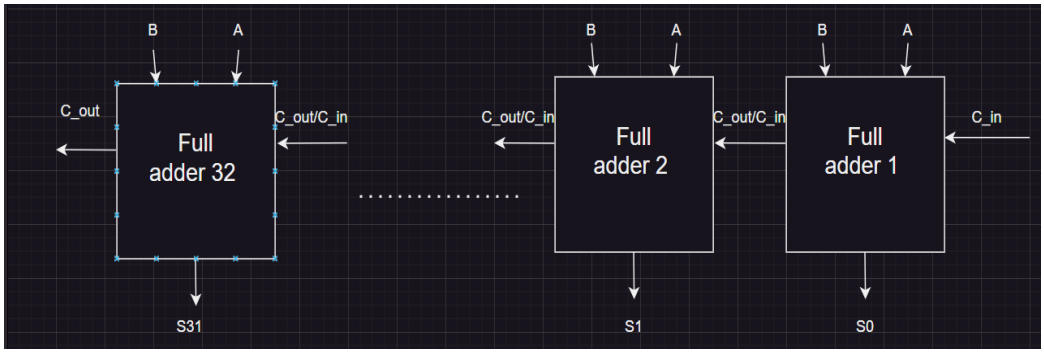
(ROM 16x32)

en functioneaza ca un enable, permitand extragerea informatiei pe 32 de biti de la adresa primita ca intrare pe iesirea dout.



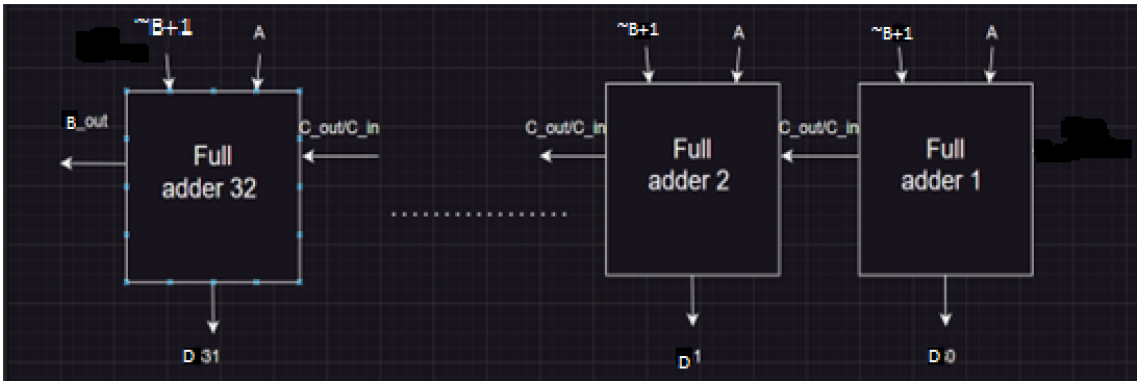
(Control Unit)

Unitatea de control realizeaza cu cei 2 operanzi primiti ca intrare operatia data de opcode.



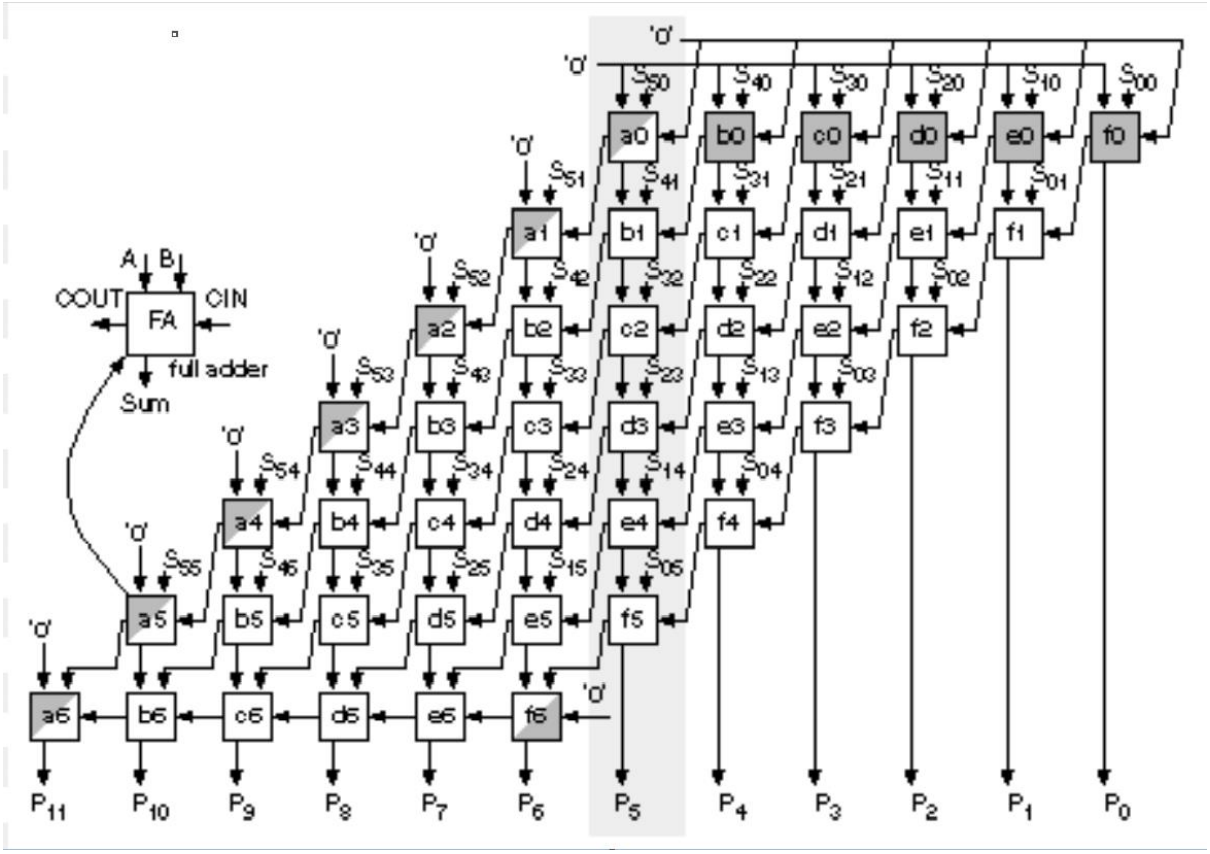
(Ripple Carry Adder)

Pentru adunare vom folosi un algoritm Ripple Carry Adder. Acesta se bazeaza pe cascadarea mai multor (in cazul nostru 32) sumatoare complete pe 1 bit (Full Adder). Dupa cum descrie si figura, fiecare Full Adder are ca intrari: 2 biti (A, B) de pe aceeasi pozitie, care se aduna, si un carry in (C_in). Fiecare adunare genereaza un rezultat (S0...S31) si un carry out, care va deveni carry in pentru sumatorul de pe pozitia urmatoare.



(Ripple Borrow Subtractor)

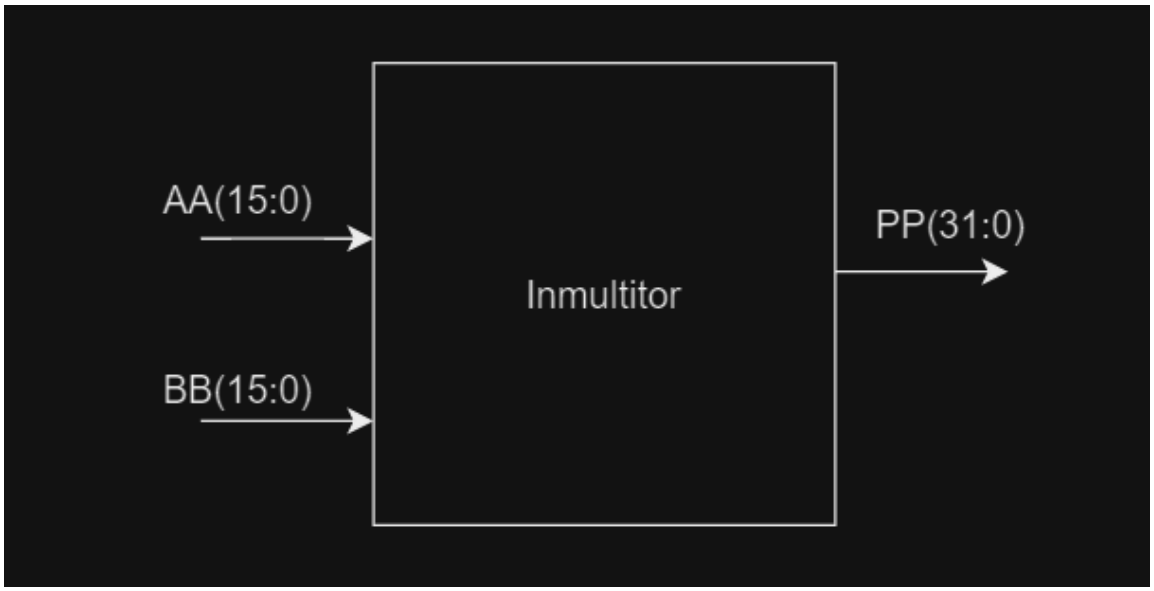
Scaderea se va face folosind aceeasi metoda, adunand primului operand, valoarea in complement fata de 2 a operandului doi.



(Multiplicator)

Pentru inmultire si impartire vom folosi algoritmii Shift and Add si Sfift and Subtract. Produsul va avea o lățime de 32 de biți si va primi 2 numere pe 16 biti. Imnultitor32 foloseste

4 componente Inmultitor (inmultitoare cu intrari pe 8b si iesiri de 16). Componentele Inmultitor sunt realizate prin maparea FullAdder-elor, ca in dfigura de mai sus, dar pe 8 biti.



(Inmultitor)

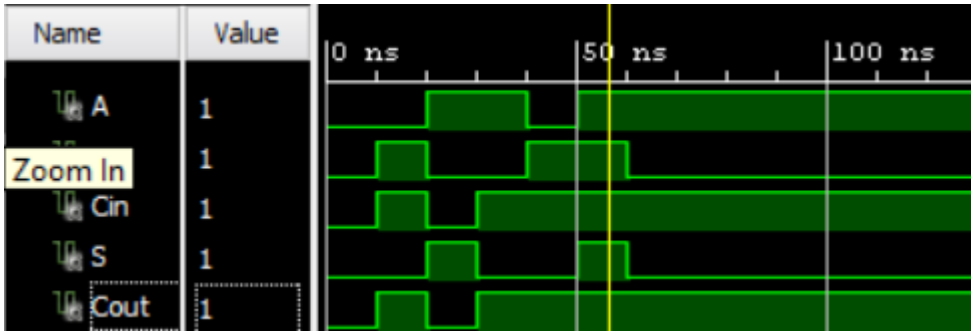
first: Reprezintă produsul celor mai puțin semnificativi octeți mLow și nLow (mLow_nLow). second: Reprezintă produsul celor mai semnificativi octeți mHigh și nLow (mHigh_nLow). third: Reprezintă produsul celor mai puțin semnificativi octeți mLow și nHigh (mLow_nHigh). fourth: Reprezintă produsul celor mai semnificativi octeți mHigh și nHigh (mHigh_nHigh2), deplasați la stânga cu 16 biți. Rezultatul final (răspunsul) se obține prin însumarea acestor valori în funcție de ponderile specificate în expresia first + (second+ third) * 256 (echivalent cu shiftarea la stanga cu 8 pozitii) + fourth. Valoarea rezultată pe 32 de biți este apoi atribuită lui PP.

Împărțirea se realizează prin shiftarea divizorului la stânga în fiecare iterație și scăderea acestuia din rest. Bitul corespunzător din rezultat este setat la '1' dacă restul este mai mare sau egal cu divizorul shiftat.

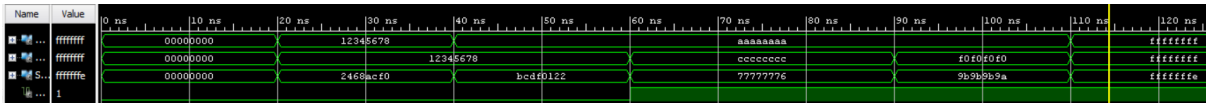
Algoritmi inmultire, impartire, adunare, scadere, shiftare la stanga, shiftare la dreapta, AND, OR, NOT, XOR, increment si decrement.

5. Testare si validare

Sumator complet pe un bit- waveform.

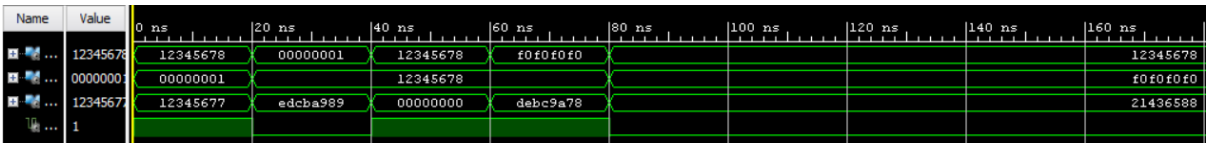


Ripple carry adder- waveform.



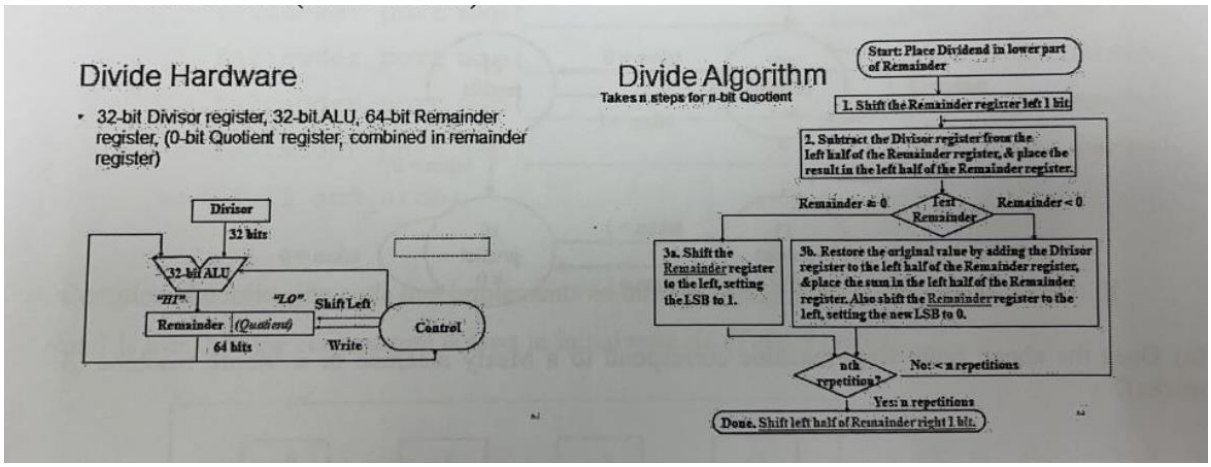
Functioneaza bine. Adunarea cu overflow functioneaza. Numere fara semn.

Ripple borrow subtract- waveform.

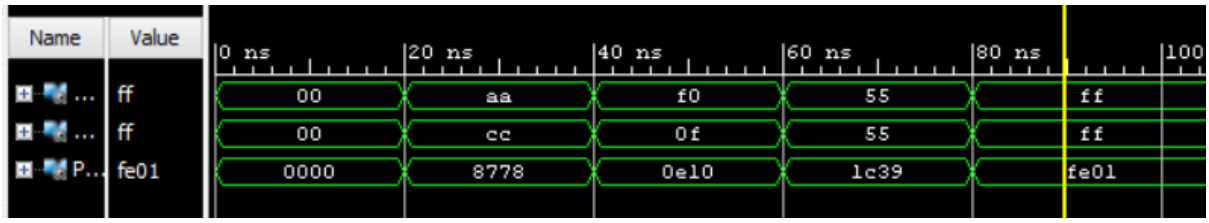


Functioneaza bine. Borrow-ul apare in mod asteptat. Numere fara semn. La scaderea unui numar mai mare dintr-un numar mai mic genereaza un numar negativ in complement fata de 2 (ex: x1- x12345678 = (ffff ffff)edcba989, care in complement fata de 2 este numarul negativ care apare la aceasta scadere).

Algoritm impartire- fara waveform.

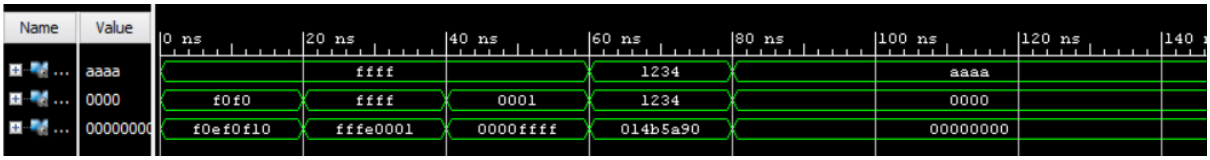


Inmultitor 16b- waveform.



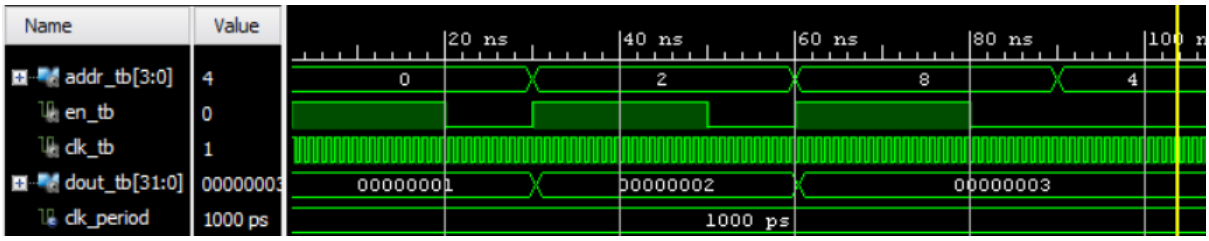
Comportament asteptat.

Inmultitor 32b -waveform.



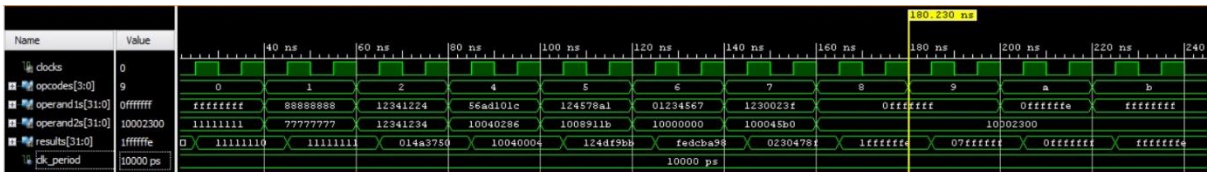
Comportament asteptat. Functioneaza pentru inmultirea cu 0. Nu poate genera overflow deoarece ia primii 16 biti din numerele pe care urmeaza sa le inmulteasca (cei mai putin semnificativi).

Memorie ROM- waveform



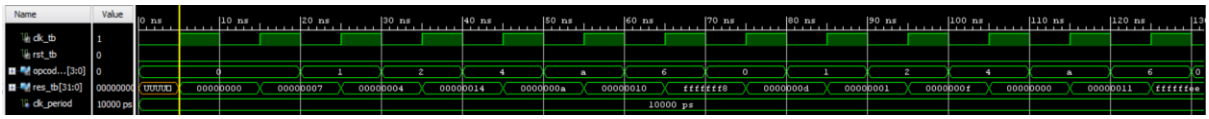
Functioneaza bine.

Control Unit- waveform



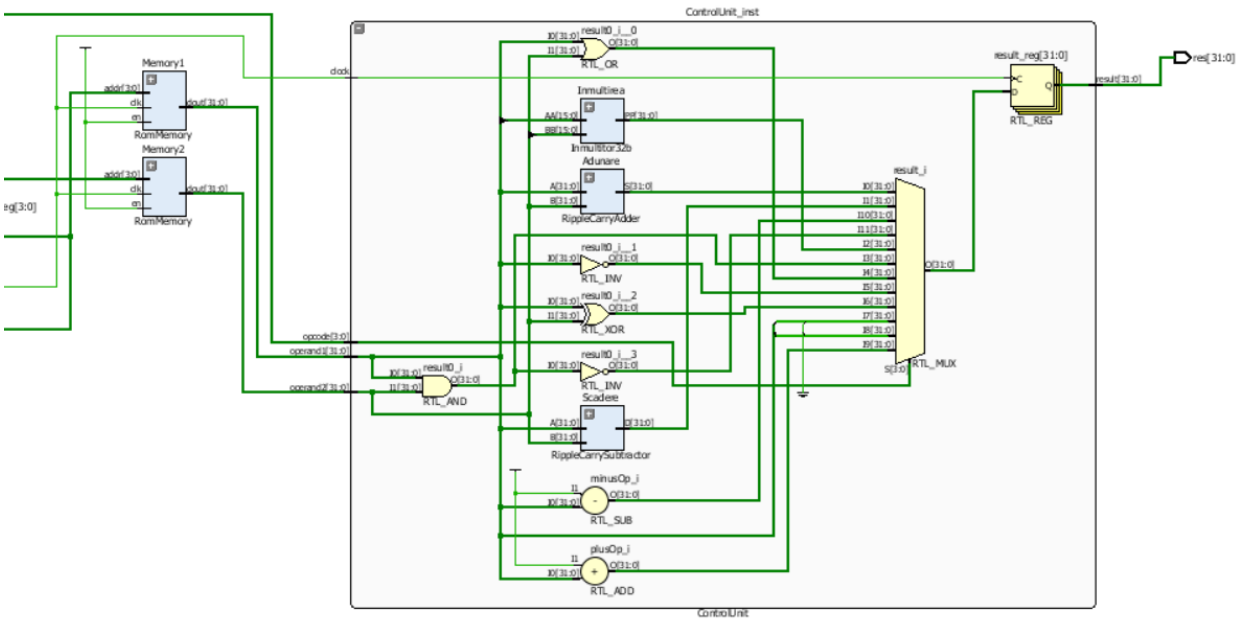
Functioneaza. Incorporeaza toate operatiile descrise inafara de impartire.

Unitate Aritmetico-Logica - waveform



Functioneaza. Pentru fiecare clock incrementeaza adresele numerelor cu care se performeaza operatiile. Cand adresa celui de al doilea operand ajunge la 16, operandul 1 reincepe de la 0 iar operandul 2 de la 1.

6. Concluzii



Proiectul functioneaza pe toate cazurile, mai putin impartire. In urma acestui proiect am inteles principiul de extragere al unui element dintr-o memorie, cat si metoda prin care se selecteaza o operatie aritnetica sau logica.

7. Bibliografie

1. https://www.spiceworks.com/tech/hardware/articles/what-is-alu/

2. https://www.studysmarter.co.uk/explanations/computer-science/computer-organisation-and-architecture/arithmetic-logic-unit/

3, https://www.baeldung.com/cs/arithmetic-logic-unit

4. https://www.javatpoint.com/what-is-alu

5. https://dl.acm.org/doi/pdf/10.5555/1074100.1074135

6. https://witscad.com/course/computer-architecture/chapter/arithmetic-logic-unit-design

7. http://www.csc.villanova.edu/~mdamian/Past/csc2400fa13/assign/ALU.html