Microprocesorul RISC-V

Canditat: Dan-Alexandru Bulzan

Coordonator științific: Ș.l.dr.ing Eugen-Horațiu Gurban

Sesiune: Iunie 2024

1 INTRODUCERE

1.1 SCOPUL ȘI MOTIVAȚIA LUCRĂRII

Implementarea personală a setului de instrucțiuni RISC-V, s-a născut din dorința de a realiza ceea ce poate fi considerat nimic mai puțin decât un apogeu al metodelor științifice din ultimul secol, și anume, procesorul.

Aceste dispozitive electronice reprezintă fundamentul tuturor științelor informatice, grație capacității computaționale intrinsece. Procesoarele, indiferent de gradul lor de specializare, au fost și rămân nucleul unei revoluții tehnologice pe care nici din pură ignoranță nu o putem omite, aceasta fiind prezentă până și în cele mai mundane aspecte ale vieții cotidiene. Scopul acestei lucrări este de a traversa universul digital, începând din rădăcinile sale analogice, ajungând într-un final la organizarea ierarhică a numeroaselor entități digitale în a căror întregime se constituie un sistem de calcul complet funcțional.

Adesea este usor sa ne pierdem în complexitățiile ascunse printre miile de porți logice, un veritabil microcosm digital, însă prin mijloacele abstractizării și modularizării, proiectarea unui procesor devine nimic mai mult decât o modelare regulată a unui sistem descriptibil de operațiile algebrei Booleane. Pe parcursul lucrării, se va prezenta de asemenea o simplă implementare didactică a modulului de memorie cache, un component digital de o importanță deosebită, precum si problematica care cere o astfel de soluție.

Implementarea va fi realizată în limbajul de descriere hardware VHDL, entitățile urmând să fie simulate prin intermediul Vivado, soluție de design și sinteză hardware oferită de Advanced Micro Devices.

DE CONTINUAT, ALTERAT

2 STUDIU BIBLIOGRAFIC

2.1 ARHITECTURA RISC

Înainte de realizarea unei analize asupra stadiului de dezvoltare și implementare al setului de instrucțiuni RISC-V, întelegerea locului pe care filozofia RISC o are în disciplina arhitecturii calculatoarelor, este de o importantă deosebită.

Acronimul RISC, face referintă la *reduced instruction set computer* sau calculator cu set de instrucțiuni reduse. Un microprocesor care implementează o astfel de filozofie, utilizează un set de instructiuni compact și puternic optimizat, garantând execuția rapidă a fiecărei instrucțiuni. Prin urmare, o caracteristică a acestei abordări, este faptul că microprocesorul va fi nevoit să execute un numar mai ridicat de instrucțiuni pentru a realiza aceleași operații efectuate de un calculator cu set de instrucțiuni complex, cunoscut și sub acronimul de *CISC*, printr-un număr observabil mai redus de instrucțiuni.

De-a lungul timpului, începând cu întemeierea arhitecturii RISC, au fost conceput mai multe seturi de instrucțiuni relevante, printre acestea enumerându-se următoarele: MIPS, ARM cât și setul care va reprezenta arhitectura procesorului implementat pe decursul acestei lucrari, RISC-V.

2.2 FAMILIA SETURILOR DE INSTRUCȚIUNI ARM

Seturile de instrucțiuni care aparțin familiei ARM sunt fără echivoc cele mai de succes dintre toate seturile aferente arhitecturii RISC. Acest succes este în mare parte datorat costurilor reduse de producție cât și eficienței computaționale ridicate. Dispozitivele dezvoltate în jurul microprocesoarelor ARM au un grade de utilitate ridicat, prezența acestora facându-se simțită într-o vastă gamă de domenii. Cele mai evidente utilizări sunt reprezentate de telefoanele mobile și computerele personale, însă arhitectura ARM a reușit să se etaleze până și în domeniul computerelor de înaltă performanță, prin intermediul supercomputerului Fugaku.

Arhitectura ARM s-a bucurat de decenii întregi de dezvoltare și prin urmare de vaste îmbunătățiri, ajungând la un grad înalt de maturitate, lucru care-i definește utilitate contemporană.

2.3 SETUL DE INSTRUCȚIUNI RISC-V

Setul de instrucțiuni RISC-V reprezintă una dintre cele mai noi adiții aduse mulțimii familiilor arhitecturii RISC. Acest ISA nu funcționează pe baza unei licențe de utilizare, fiind un standard deschis, este permisă folosirea sa tuturor entităților legale sau persoanelor care doresc implementarea unui microprocesor sau a unui sistem integrat bazându-se pe acest set.

2.4 IMPLEMENTĂRI RISC-V

Datorită proliferării lipsite de licență cât și împărțirii setului în extensii, se poate observa un constant flux de implementări, variind de la simple exemple didactice la sisteme cu module multicip complexe. Numeroase programe de studii care au ca scop dezvoltarea cunoștiintelor despre organizarea calculatoarelor, obișnuiesc să prezinte ca suport didactic implementări succinte ale unui nucleu RISC-V. Fiecare asemeni implementare prezintă ușoare diferențe arhitectural-organizatorice față de omologi săi. Aceste diferențe sunt produsul faptului că arhitectura RISC-V nu îngrădește utilizatorii săi într-o specifică topologie de organizare a modulelor care constituie în intregimea lor un microprocesor. Fiecare utilizator are astfel liber arbitru în definirea propriei organizări, atât timp cât respectă setul de instrucțiuni.

Se disting astfel două mari tipuri de microprocesoare RISC-V, ale căror implementări sunt disponibile spre analiză. Prima și cea mai comună este reprezentată de microprocesorul RISC-V SCP sau *single cycle processor*, cea de a doua purtând numele de *multi-cycle processor* sau pe scurt, MCP.

DE CONTINUAT

3 FUNDAMENTARE TEORETICĂ

3.1 GESTIONAREA COMPLEXITĂȚII

Cand vine vorba de modelara unui sistem computațional de o complexitate ridicată, este de preferat să avem anumite fundamente în implementare, pe care să ne putem baza fără echivoc. În lipsa acestor principii este adesea usor să ne pierdem în complexitatea sistemului, rezultând astfel posibile erori care-și vor face simțită prezența în produsul final.

3.1.1 ABSTRACTIZARE

Abstractizarea este opusul specificității. Din punct de vedere conceptual, actul de abstractizare, indiferent de suportul teoretic asupra căruia este aplicat, ajută la simplificarea unei probleme a cărei complexități ar fi de altfel prea greu de tratat. Prin abstractizare, detalile de la un anumit nivel logic al unui sistem, sunt redate sumar și considerate ca atare de catre nivelele logice superioare.

Acest lucru poate fi observat într-o multitudine de domenii, de la arhitectura calculatoarelor la studiul fiziologiei medicale. De exemplu, bazându-ne pe cel din urmă domeniu enumerat, modul de funcționare a unui organism viu poate fi privit din mai multe perspective de abstractizare, începând de la interacțiunile biochimice si biomecanice de la nivelul unei celule, trecând pe urmă la modul în care aceste celule interacționeaza între ele formând variate țesuturi, ajungând într-un final la nivelul de abstracție al țesuturilor care împreună formează organe, fiecare nivel implicându-l direct pe precedentul său.

3.1.2 MODULARITATE

Modularizarea definește modul în care un sistem computațional va fi divizat în numeroase parți de sine stătătoare, acum numite module, fiecare cu un rol și o interfață de utilizare concis definită. Aceste module permit astfel reutilizarea entităților pe care le definesc, ne mai fiind nevoie de irosirea unei perioade mari de timp cu diverse noi implementării care sunt congruente cu un modul deja existent. Modularizarea ne permite de asemenea înlocuirea unor părți ale sistemului nostru cu altele de o eficiență mai ridicată, cât timp acestea respectă aceeași interfață pentru a permite comunicarea cu modulele adiacente.

3.1.3 IERARHIZARE

Ierarhizarea implică ordonarea într-o arhitectură a modulelor anterior definite. Arhitectura, în cazul nostru, va fi reprezentată de modul de organizare a microprocesorului ce urmează a fi dezvoltat, microarhitectura acestuia. Organizarea ierarhică implică modularitatea dar vice-versa nu este mereu valabilă, modulele putând exista pe același nivel ierarhic, nefiind, prin urmare, subordonate unul altuia.

3.2 ABSTRACȚIA NUMERICĂ

Pentru a produce un rezultat de o oarecare utilitate, sistemele computaționale au nevoie de date. Aceste date sunt complet irelevante cât timp nu respectă un mod de reprezentare util sistemului. De asemenea, este importat de luat in considerare faptul ca datele hrănite pot avea semnificații diverse, complet obtuze una față de cealaltă.

Problema reprezentării datelor primește o importanță specială, deosebită chiar, dând naștere următoarei multitudini de întrebări, *care este este modul corect de reprezentare*; *cum asigurăm coerența datelor cu analizarea acestora de către sistemul de calcul*; *cum ne asigurăm ca datele indiferent formatului ligibil uman, nu sunt iligibile procesorului.*

Pentru a răspunde pe deplin, trebuie mai întâi să definim tipul datelor pe care microprocesorul le va accepta. Este rapid evident, din natura sistemului, că datele trebuie să fie numerice. Însă, nu la fel de evident este modul în care aceste numere vor fi reprezentate pentru a suporta toate operațiile admisibile de un motor logic-aritmetic.

Cea mai reprezentativă caracteristică a unui sistem de numerație este numărul de simboluri unice utilizate de acesta. Numărul de simboluri poartă numele de radix și este congruent cu conceptul de bază numerică. Valoarea minimă pe care radix-ul unui sistem de numerație o poate lua este *1*, corespunzând unui sistem cu un singur simbol, fiecare număr continând *n+1* simboluri față de precedentul sau *n*. Însă, trecând cu vederea această anomalie numerică, bazele care vor reprezenta suportul matematic al acestei lucrări sunt cea decimală, cea hexadecimală și cea binară. În Tabela 1 se pot observa bazele anterior menționate, însoțite de simbolurile aferente cât și de un exemplu reprezentativ.

Tabela 1: Intervalul de simboluri posibile, raportate la baza numerică

Radix	Valori	Exemplu
Unar	1	111
Binar	0, 1	1000
Decimal	[0, 9]	10
Hexadecimal	$[0, 9] \cup [A, F]$	B4

Un alt aspect important, strâns legat de radix, este numărul de simboluri s necesare pentru a reprezenta un număr oarecare n în baza r. Relația matematică care defineste acest aspect este redată prin formula 1.

$$s = log_r n \tag{1}$$

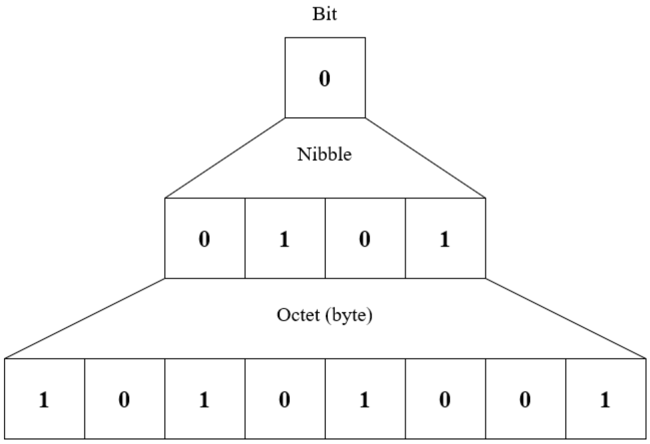
3.2.1 NUMERELE BINARE

Modul de funcționare a dispozitivelor digitale este constituit pe oscilațiile rapide ale semnalelor electrice, semnale a căror valori se identifică cu unul dintre membri faimosului cuplu binar, *0 și 1.* Prin urmare, datele vor avea o reprezentare care utilizează radixul binar.

Fiecare simbol dintr-un număr binar poarte numele de bit, un amalgam de 4 biți se numește nibble, iar o înșiruire de 2 nibble, echivalentă cu 8 biți, poartă numele de octet.

Conceptul de împărțire a unui numar binar în octeți ajuta reprezentarea acestora într-o bază numerică superioară, în special cea hexadecimală. Figura 1 prezintă clar părtile constituente a unui octet și relația dintre acestea.

Figura 1: Modul de împărțire a unui octet, părțile sale constituente



Octet-ul va reprezenta unitatea fundamentală și indivizibilă pentru microarhitectura microprocesorul dezvoltat prin această lucrare. Acesta, prin urmare, este cea mai mică entitate adresabilă cu care se va lucra. Un octet este limitat de numărul de date pe care le poate reprezenta, acestea fiind calculate prin exponentul 2^n, în cazul nostru, 2^8 sau 256 de valori.

Un lucru important de menționat este că valoarea maximă a unui număr pe n biți va fi mereu $2^n - 1$. Prin urmare, dacă dorim să reprezentăm o putere oarecare 2^n, vor fi necesari $n + 1$ biți de date. Tabelul 2 prezintă realația dintre magnitudinea binară (număul de biți folosiți în reprezentare) și cantitata de date reprezentate prin plaja de valori adiacentă, ignorând existenta numerelor negative.

Tabela 2: Plaja de valori asumând numere strict pozitive, de magnitudini binare diverse

Biți de date	Numărul datelor	Interval valori
8	256	$0 \leq n \leq 2^8 - 1$
16	65536	$0 \leq n \leq 2^{16} - 1$
32	2^{32}	$0 \leq n \leq 2^{32} - 1$
64	2^{64}	$0 \leq n \leq 2^{64} - 1$

3.2.2 OPERAȚIILE MATEMATICE ȘI NUMERELE BINARE

Pentru a utiliza în mod corect reprezentările în această bază, modul în care calculele matematice sunt efectuate asupra numerelor binare necesită clarificare.

Datele nu sunt folositoare doar prin existența lor. Pentru a dobândi utilitate, acestea sunt supuse aparatului matematic, prin care se calculează diverse valori, asumând un algoritm corect, care ne oferă informații despre problema pe care dorim să o rezolvăm.

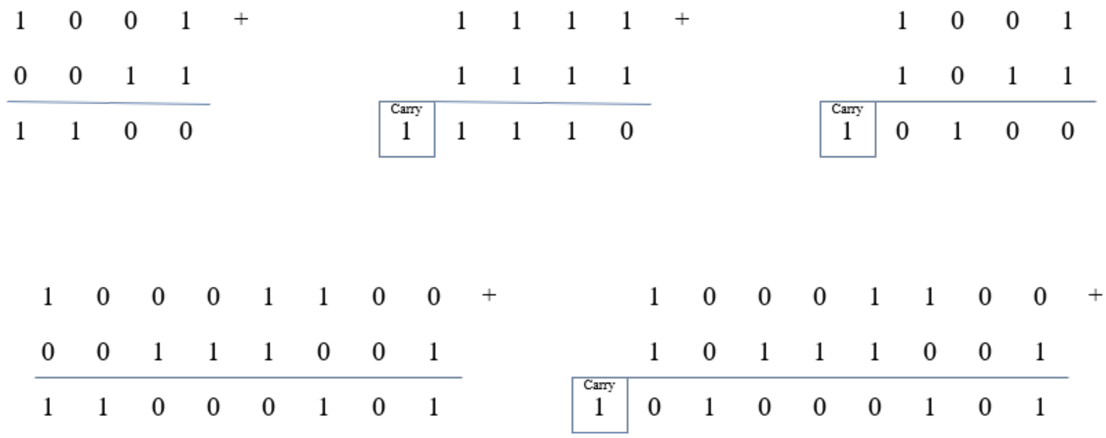
Cea mai elementară operație matematică care poate fi aplicată unui număr este adunarea. Însumarea numerelor este cu atât mai facilă cu cât numărul de simboluri folosite în reprezentarea acestora scade. Spre norocul lumii digitale, radix-ul utilizat permite efectuarea operațiilor matematice în cele mai simple metode. Există doar 4 operații fundamentale posibile, acestea fiind prezentate în Tabela 3.

Tabela 3: Operațiile fundamentale de însumare a numerelor binare

Operație	Sumă	Carry
$0+0$	0	0
$0+1$	1	0
$1+0$	1	0
$1+1$	0	1

Dintre toate aceste operații, cea căreia îi vom oferi o importanță ridicată este $1+1=0$ *carry* 1. Aceasta ne obligă să adunăm o unitate bițiilor de pe poziții superioare. Practic, acestă sumă, odată ce este generalizată, ne spune că $2^n+2^n=2\cdot 2^n=2^{n+1}$, o trivialitate matematică. Diverse exemple de adunare ale numerelor binare pot fi consultate în Figura 2.

Figura 2: Exemple de adunare a numerelor binare



3.2.3 REPREZENTAREA NUMERELOR NEGATIVE

Modul în care numerele positive sunt adunate fiind acum clarificat, următoara operație matematică tratată este scăderea. Aceasta poate fi vizualizată ca adunarea unui numar a la inversul aditiv al altui număr b. Această operație cere prin urmare un mod de reprezentare al numerelor binare negative, soluție care vine prin trei metode.

Primul mod de reprezentare este cel prin semn-magnitudine. Acesta este intuitiv, fiind similar cu reprezentarea numerelor decimale cu semn. Bit-ul cel mai semnificativ devine acum bit-ul de semn, 1 reprezentând un număr negativă, iar 0 unul pozitiv. Tabelul 4 prezintă aplicarea acestei reprezentări asupra numerelor pe 8 biți.

Tabela 4: Reprezentarea prin semn-magnitudine

Valoare binară	Semn magnitudine	Fără semn
00000000	0	0
00000001	1	1
00000010	2	2
...	...	...
01111110	126	126
01111111	127	127
10000000	-0	128
10000001	-1	129
10000010	-2	130
...	...	...
11111101	-125	253
11111110	-126	254
11111111	-127	255

Se pot astfel distinge următoarele lucruri:

- Există două reprezentări posibile pentru 0, și anume ± 0.
- Deși se acopera tot 255 de valori numerice posibile (256 cu cel de al doilea 0), plaja de valori *signed* s-a distribuit egal numerelor negative și celor pozitive. Astfel, numerele *unsigned* semn-magnitudine sunt cuprinse în intervalul $[-2^{n-1}+1, 2^{n-1}-1]$ unde $n > 0$

O altă medotă de reprezentare este prin complementul de 1. Conform acesteia, se inversează biții numărului binar pozitiv (biții cu valoarea 1 vor deveni 0 și viceversa) rezultând astfel inversul său aditiv. Spre exemplu, $00000001' = 11111110$; $01010101' = 10101010$ iar, în cazul lui 0, $00000000' = 11111111$. Tabela 5 conține reprezentările numerelor de 8 biți.

Tabela 5: Reprezentarea prin complement de 1

Valoare binară	Semn magnitudine	Fără semn
00000000	0	0
00000001	1	1
00000010	2	2
...	...	...
01111110	126	126
01111111	127	127
10000000	-127	128
10000001	-126	129
10000010	-125	130
...	...	...
11111101	-2	253
11111110	-1	254
11111111	-0	255

La fel ca în cazul reprezentării prin semn-magnitudine, 0 dorește să respecte principiul superpoziției, diferența pricipală însă, precum se distinge din compararea Tabelei 5 cu Tabela 4 este faptul ca valorile negative sunt eșalonate invers.

Problema acestor reprezentări este cu atât mai vizibilă când se efectuează adunarea a două numere binare, rezultatul fiind mereu 111...111. Soluția vine prin complementul de 2, complement format prin adăugarea unei unități reprezentării complementare de 1. Efectul însumării unitare este cel mai bine explicitat de Figura 3.

Figura 3: PLACEHOLDER

$$-2^{n-1}+1 \longleftarrow\!\!\!-\!\!\!-\!\!\!-\; 0 \;-\!\!\!-\!\!\!-\!\!\!\longrightarrow 2^{n-1}-1$$