Unitate de calcul în virgulă mobilă: extragerea radăcinii pătrate

Mureșan Alexandru-Dorian

Cuprins

1. Introducere

1.1 Context

Descriere generală: Unitatea de calcul în virgulă mobilă reprezintă o componentă esențială a procesorului (CPU) care este responsabilă pentru efectuarea operațiilor matematice care implică numere în format virgulă mobilă. Extracția rădăcinii pătrate dintr-un număr în virgulă mobilă este o operație matematică complexă și de mare importanță în domeniul calculului numeric, științific și graficii computerizate. Această operație necesită algoritmi și hardware specializați pentru a asigura acuratețe și eficiență.

1.2 Soluție propusă

Proiectarea și dezvoltarea unei unități de calcul în virgulă mobilă pentru extragerea rădăcinii pătrate reprezintă un proces complex care necesită expertiză în arhitectura procesoarelor și matematică numerică. O atenție deosebită la detaliile hardware este esențială pentru a obține un FPU dedicat eficient și precis pentru această operație matematică specifică.

1) Algoritmi Optimizați: Pentru a realiza extragerea rădăcinii pătrate într-un mod eficient, este esențial să dezvoltați sau să selectați algoritmi optimizați care să minimizeze numărul de operații necesare și să ofere precizie ridicată. Un astfel de algoritm poate implica utilizarea metodei Newton-Raphson sau a altor metode iterative care converg rapid către valoarea corectă a rădăcinii pătrate.
2) Arhitectura Hardware Dedicată: Pentru a implementa algoritmul de extragere a rădăcinii pătrate, este necesară dezvoltarea unei arhitecturi hardware dedicate în cadrul FPU. Aceasta ar putea include unități funcționale specializate, registre de control, și circuite pentru efectuarea operațiilor matematice.
3) Precision Hardware: Precizia este un aspect crucial în extragerea rădăcinii pătrate, mai ales când lucrăm cu numere în virgulă mobilă. Hardware-ul trebuie să fie capabil să gestioneze cu precizie virgula mobilă, să evite eroarea de trunchiere și să ofere rezultate corecte, cu cât mai multe cifre semnificative.
4) Control de Excepții: FPU trebuie să fie capabil să trateze excepțiile care pot apărea în timpul extragerii rădăcinii pătrate. De exemplu, în cazul unui număr negativ, poate fi necesar să se declanșeze o excepție și să se gestioneze corect această situație.
   Pentru indicarea diferitelor condiţii de excepţie, ca în cazul operaţiilor nedefinite de forma ∞/∞, $\infty-\infty$, $\infty * 0$, $0/\infty$, $0/0$, sau extragerea rădăcinii pătrate dintr-un număr negativ, s-a prevăzut un format special, care nu reprezintă un număr obişnuit, fiind numit NaN (Not a Number). Exponentul are valoarea maximă posibilă, iar mantisa este diferită de 0. Astfel, există o clasă întreagă de valori NaN.
5) Optimizare a Performanței: Într-un mediu de calcul modern, performanța este esențială. Dezvoltarea hardware-ului ar trebui să includă optimizări pentru a reduce latența și pentru a permite FPU să efectueze extrageri de rădăcini pătrate într-un timp cât mai scurt.
6) Integrare cu Arhitectura CPU: FPU trebuie să fie perfect integrată cu arhitectura generală a CPU-ului. Aceasta implică dezvoltarea unor mecanisme de interfațare adecvate pentru a permite transferul datelor între FPU și restul CPU-ului.
7) Testare și Validare: Odată ce hardware-ul este dezvoltat, acesta trebuie supus unor proceduri de testare și validare riguroase pentru a se asigura că funcționează corect și furnizează rezultate precise.

8) Documentare Tehnică: Un aspect important este crearea de documentație tehnică adecvată pentru a descrie arhitectura FPU, instrucțiunile suportate și modul de utilizare.

1.3 Plan de proiect

1.3.1 Studiul Bibliografic:

Efectuarea unui studiu aprofundat al literaturii pentru a identifica algoritmii eficienți și optimizați pentru extragerea rădăcinii pătrate în format virgulă mobilă.

Investigarea metodelor matematice care pot fi aplicate într-o unitate de calcul în virgulă mobilă pentru a obține rezultate precise.

1.3.2 Analiză:

Analiza cerințelor de performanță și precizie pentru operația de extragere a rădăcinii pătrate în diverse aplicații.

Evaluarea resurselor hardware disponibile pentru implementarea algoritmului într-o unitate de calcul în virgulă mobilă.

Reprezentarea numerelor în virgulă mobilă:

Primul aspect pe care trebuie să înțelegem este cum putem să reprezentăm numerele cu virgulă în formă binară. Ca și în cazul numerelor întregi, valorile fracționare se reprezintă pe un anumit număr de biți.

Principalele obiective ale unei reprezentări pe un anumit număr de biți sunt:

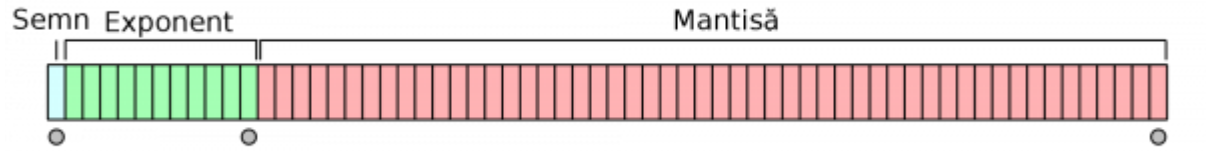
- posibilitatea de a reprezenta cât mai multe valori intre valoarea minimă și valoarea maximă
- o precizie cât mai bună a valorilor (numărul maxim de cifre după virgulă)

Pentru a reprezenta valorile fracționare vom folosi **Reprezentarea în virgulă mobilă** (**Floating Point Representation**). În această reprezentare, numerele au următoarea structură:

$$(-1)^{semn} * 1.mantisa * (baza)^{exponent}$$

După cum putem observa mai sus, valorile trebuie transformate astfel încât partea întreagă să fie 1. Această formă poartă numele de formă normală, iar operația de transformare în această formă poartă numele de **normalizare**.

În forma binară, valorile se reprezintă astfel:



Semnul este dat de primul bit din reprezentarea binară: 1 = negativ, 0 = pozitiv.
Mantisa dă partea fracționară a numărului în forma normală. Numărul de biți pe care mantisa este reprezentată dă precizia maximă a reprezentării.
Baza este de obicei 2, 10 sau 16 și este dată de standardul de reprezentare ales.
Exponentul dă valoarea la care este ridicată baza și numărul de biți pe care este reprezentat dă valorile valorile maxime și minime ce pot fi reprezentate.

Standarde de reprezentare în virgulă mobilă:

Cele mai folosite standarde de reprezentare în virgulă mobilă sunt cu precizie simplă (**Single Precision**) și cu precizie dublă (**Double Precision**).

Reprezentarea cu **precizie simplă** presupune folosirea a 32 de biți și corespunde valorilor float din limbajul C. În acest caz baza folosită este 2, exponentul are 8 biți, iar restul de 23 de biți corespund mantisei.

Reprezentarea cu **precizie dublă** presupune folosirea a 64 de biți și corespunde valorilor de tip double. În acest caz, baza este 2, exponentul are 11 biți, iar restul de 52 de biți corespund mantisei.

1.3.3 Descriere și Implementare:

Această secțiune a proiectului este crucială și necesită expertiză în dezvoltarea hardware-ului și în evaluarea matematică. Scopul final este de a oferi o unitate de calcul în virgulă mobilă dedicată și eficientă pentru extragerea rădăcinii pătrate, care să îndeplinească standardele de precizie și performanță cerute pentru aplicația respectivă.

1) Dezvoltarea sau Adaptarea Algoritmului:
   - În acest pas, se alege sau se dezvoltă un algoritm optimizat pentru extragerea rădăcinii pătrate în format virgulă mobilă. Algoritmul trebuie să fie adecvat pentru implementarea hardware și să aibă un nivel de precizie corespunzător.
   - Sunt luate în considerare aspecte matematice, cum ar fi metoda Newton-Raphson sau metode iterative, pentru a ajunge la o soluție eficientă. Adaptarea algoritmului pentru formatul în virgulă mobilă este esențială pentru a asigura precizia și performanța.
2) Proiectarea Unității de Calcul în Virgulă Mobilă (FPU):
   - Această etapă implică proiectarea hardware-ului pentru a implementa algoritmul selectat. Componentele hardware necesare includ unități funcționale pentru operațiile matematice, registre de control, unități de stocare temporară a datelor, și unități pentru gestionarea preciziei.
   - FPU trebuie să fie integrată în arhitectura generală a CPU-ului și să aibă interfețe corespunzătoare pentru a comunica cu restul sistemului.

3) Implementarea și Testarea Hardware-ului:
   - După proiectare, se trece la etapa de implementare, care presupune construcția fizică a hardware-ului. Acest lucru poate fi realizat fie pe o placă de dezvoltare, fie pe un cip (chip) dedicat.
   - După implementare, se efectuează teste hardware pentru a verifica corectitudinea funcționării. Aceste teste implică furnizarea datelor de intrare și verificarea rezultatelor obținute de hardware pentru a se asigura că algoritmul este implementat corect.
4) Evaluarea Performanței, Acurateții și Eficienței
   - După testarea hardware, se efectuează o evaluare detaliată a performanței, acurateții și eficienței soluției propuse. Se compară rezultatele obținute de hardware cu valorile teoretice corecte pentru a verifica precizia.
   - Se măsoară și se evaluează performanța în termeni de latență și putere consumată. Dacă este necesar, se fac optimizări pentru a îmbunătăți performanța sau eficiența.

2. Studiul bibliografic

2.1 Prezentare generală

În cadrul studiului bibliografic pentru proiectarea unei unități de calcul în virgulă mobilă (FPU) specializate pentru extragerea rădăcinii pătrate, este crucial să se analizeze în detaliu aspectele hardware care sunt relevante pentru această operație. Studiul bibliografic ar trebui să înceapă cu o prezentare generală a arhitecturii hardware, care include caracteristicile de bază ale unui FPU, cum ar fi tipul de virgulă mobilă (de exemplu, IEEE 754), capacitatea de calcul, precizia și performanța.

2.2 Utilizări

Trebuie să se exploreze utilizările potențiale ale unei unități de calcul în virgulă mobilă specializate pentru extragerea rădăcinii pătrate. Aceasta poate include aplicații din domenii precum grafica computerizată (pentru a accelera operațiile de transformare și rasterizare), simulări științifice (pentru calcule complexe în fizică sau inginerie) și în alte contexte în care extragerea rădăcinii pătrate este o operație frecventă.

2.3 Arhitectura setului de instrucțiuni

Studiul bibliografic trebuie să se concentreze pe arhitectura setului de instrucțiuni (ISA - Instruction Set Architecture) folosită pentru FPU. Acest lucru poate include detalii despre tipurile de instrucțiuni utilizate pentru operațiile în virgulă mobilă, modul în care instrucțiunile sunt codificate și cum sunt executate în hardware.

2.4 Date si memorie

Analiza ar trebui să acopere modul în care datele în virgulă mobilă sunt reprezentate și stocate în memoria fizică. Este important să se înțeleagă cum valorile în virgulă mobilă sunt normalizate, cum se realizează conversiile între formatul intern și reprezentarea externă și cum este gestionată precizia în timpul operațiilor.

2.5 Componente

Trebuie să se examineze componentele hardware ale FPU-ului, inclusiv registrele speciale, unitățile de calcul, registrele de stare și orice alte module care contribuie la operația FPU. De asemenea, este important să se studieze modul în care aceste componente lucrează împreună pentru a realiza extragerea rădăcinii pătrate.

2.6 Endiannes

Studiul bibliografic trebuie să abordeze problema endianness-ului, care se referă la modul în care datele cu mai multe octeți sunt stocate în memorie. Este important să se știe cum este gestionată endianness-ul în FPU și cum poate influența operațiile cu numere în virgulă mobilă.

2.7 Adresarea memoriei

Trebuie să se investigheze modul în care FPU accesează și adresează memoria pentru a citi sau scrie date. Acest aspect poate influența performanța și eficiența operațiilor de extragere a rădăcinii pătrate, în special atunci când se lucrează cu mari volume de date.

Un studiu bibliografic detaliat asupra acestor aspecte hardware este esențial pentru proiectarea unei unități de calcul în virgulă mobilă eficiente și precise pentru extragerea rădăcinii pătrate. Aceste cunoștințe vor servi ca fundație solidă pentru dezvoltarea hardware-ului specializat necesar pentru această operație matematică complexă.

3. Analiză

3.1 Setul de intrucțiuni

Setul de instrucțiuni ar trebui să includă următoarele tipuri de instrucțiuni, fiecare cu un opcode specific:

- Instrucțiuni aritmetice (AL): Pentru operații de bază în virgulă mobilă, cum ar fi adunarea, scăderea, înmulțirea și împărțirea în virgulă mobilă.
- Instrucțiuni de comparare (SC): Pentru a permite compararea numerelor în format în virgulă mobilă și setarea flag-urilor pentru a controla fluxul de program.
- Instrucțiuni de încărcare și stocare (I): Pentru a transfera date între registre și memorie.
- Instrucțiuni de salt (J): Pentru a permite bifurcări în program și sărituri la adrese specifice.

3.1.1 Exemple de Instrucțiuni:

Instrucțiuni Aritmetice (AL):

- ADD R1, R2, R3: Adună valorile din registrele R2 și R3 și stochează rezultatul în R1.
- SUB R4, R5, R6: Scade valoarea din registrul R5 din valoarea din registrul R6 și stochează rezultatul în R4.

- MUL R7, R8, R9: Înmulțește valorile din registrele R8 și R9 și stochează rezultatul în R7.
- DIV R10, R11, R12: Împarte valoarea din registrul R11 la valoarea din registrul R12 și stochează rezultatul în R10.

Instrucțiuni de Comparare (SC):

- CMP R13, R14, R15: Compară valoarea din R14 cu valoarea din R15 și setează flag-urile corespunzătoare în registrul R13 în funcție de rezultat.

Instrucțiuni de Încărcare și Stocare (I):

- LD R16, [R17]: Încarcă valoarea de la adresa de memorie stocată în R17 în registrul R16.
- ST [R18], R19: Stochează valoarea din registrul R19 la adresa de memorie specificată în R18.

Instrucțiuni de Salt (J):

- BRZ R20, ETICHETA: Saltă la eticheta specificată dacă registrul R20 conține zero.
- JMP ETICHETA2: Saltă necondiționat la eticheta2.

3.1.2 Instrucțiune Specială CISC (Complex Instruction Set Computer):

Într-un set de instrucțiuni CISC, puteți avea instrucțiuni complexe care efectuează mai multe operații într-o singură instrucțiune. Un exemplu ar putea fi:

- SQRTOPT R21, R22: Această instrucțiune ar putea să includă operații complexe necesare pentru extragerea rădăcinii pătrate din valoarea din R22 și să stocheze rezultatul în R21.

3.2 Câmp de condiție

3.2.1 Ce este campul de conditie?

După cum am prezentat anterior, fiecare instrucțiune va fi precedată de 4 biți care sunt numiți biți de condiție.

Rezultă 16 condiții posibile pentru fiecare instrucțiune. De asemenea, dacă toți cei patru biți sunt „0”, atunci instrucțiunea va fi executată, indiferent de steagurile din CPSR (Current Program Status Register).

De asemenea, merită menționat în acest moment că fiecare instrucțiune va afecta CPSR, în principal steagurile acestui registru. Acest lucru se face pentru a propaga efectele unei instrucțiuni la următoarea.

Acest aspect va fi prezentat mai detaliat la capitolul Proiectare, la prezentarea designului Field Register.

3.2.2 Codul conditiilor

Aici inca nu stiu ce sa pun

3.2.3 Utilizare si interpretare

În limbajul de asamblare, diferențierea între instrucțiunile condiționate și instrucțiunile normale (adică instrucțiuni care sunt întotdeauna executate) se realizează prin adăugarea sufixului indicat în tabel.

În limbajul mașină (a.k.a. reprezentarea binară a instrucțiunilor) diferențierea se face la pasul de decodare a instrucțiunii, când biții de condiție dictează ce flag-uri ale CPSR vor fi verificate.

4. Plan de proiectare

4.1 RLT Abstract

4.2 Codificarea instrucțiunilor

4.3 Codificarea funcțiilor

4.4 Codificarea operatiilor

4.5 Diagrame block

8. Bibliografie

- http://elf.cs.pub.ro/asm/wiki/laboratoare/laborator-12
- https://www.slideserve.com/yardley-hoover/2-unitatea-aritmetic-i-logic
- https://users.utcluj.ro/~baruch/book_ac/AC-Reprez-VM.pdf