ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

BÀI TẬP LỚN KIẾN TRÚC MÁY TÍNH

NHÓM BTL067 - ĐỀ 4

NHÂN HAI SỐ THỰC CHUẨN IEEE-754 CHÍNH XÁC ĐƠN

GVHD: Nguyễn Xuân Minh
SV: Nguyễn Thiên Hào - 2310853
Phạm Quang Hiếu - 2310974

TP. HỒ CHÍ MINH, THÁNG 11/2024

Mục lục

MỞ ĐẦU

Chúng em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Xuân Minh vì đã định hướng, hướng dẫn và cung cấp những kiến thức quý giá để nhóm có thể hoàn thành tốt nhất nội dung bài báo cáo. Sự tận tình và hỗ trợ của thầy là nguồn động lực lớn lao giúp nhóm hoàn thiện đề tài một cách hiệu quả.

Bài báo cáo của chúng em sẽ đi từ giải thuật tổng quát cho bài toán nhân hai số thực cho đến hiện thực chi tiết và cụ thể, sau cùng là đánh giá lại chương trình. Nhóm đã cố gắng hết mình, nhưng việc tránh hoàn toàn những thiếu sót là điều không thể. Vì vậy, chúng em rất mong nhận được những ý kiến đóng góp quý báu từ thầy để có thể cải thiện bài báo cáo trở nên hoàn thiện hơn nữa.

Trân trọng cảm ơn thầy!

1 Đề tài

Đề 4. Nhân 2 số thực chuẩn IEEE 754 chính xác đơn.

Viết chương trình thực hiện phép nhân 2 số thực chuẩn IEEE 754 chính xác đơn mà không dùng các lệnh tính toán số thực của MIPS. Dữ liệu đầu vào đọc từ file lưu trữ dạng nhị phân trên đĩa `FLOAT2.BIN` (2 trị x 4 bytes = 8 bytes).

2 Cơ sở lý thuyết

Theo chuẩn IEEE754 cho số thực với độ chính xác đơn (single precision), một số thực sẽ chiếm 32 bit và bao gồm 3 trường:

- Trường dấu (sign): chiếm 1 bit đầu tiên. Ký hiệu là s.
- Trường mũ (exponent): chiếm 8 bit tiếp theo, là một số nguyên không dấu, ký hiệu là e.
- Trường thập phân (fraction): chiếm 23 bit cuối cùng, ký hiệu là fraction.

Khi đó, số thực chính xác đơn f có thể được biểu diễn theo công thức:

$$f = (-1)^s \times (1 + 0.\text{fraction}) \times 2^{e-\text{bias}}$$

Trong đó, bias = 127 (vì số mũ trong IEEE-754 có thể biểu diễn dải số từ 0 đến 255, và bias giúp làm cho số mũ này có thể sử dụng giá trị không âm).

3 Giải pháp hiện thực

3.1 Tổng quan giải thuật

Giả sử ta đã trích xuất được các trường của hai số thực chính xác đơn, f_1 và f_2, gồm các phần: dấu s_1, s_2, số mũ e_1, e_2 (chưa trừ bias), và chuỗi bit phần thập phân fraction_1, fraction_2. Ta sẽ tiến hành thao tác trên các bit thành phần trên để tính toán ra kết quả mà không dùng tới hàm nhân số thực của Mars 4.5

Công thức tổng quát của phép nhân hai số thực chính xác đơn là:

$$f_1 \times f_2 = (-1)^{s_1} \times (1 + 0.\text{fraction}_1) \times 2^{e_1-\text{bias}} \times (-1)^{s_2} \times (1 + 0.\text{fraction}_2) \times 2^{e_2-\text{bias}}$$

Biến đổi thành:

$$= (-1)^{s_1+s_2} \times (1 + 0.\text{fraction}_1) \times (1 + 0.\text{fraction}_2) \times 2^{e_1+e_2-2\times\text{bias}}$$

Kết quả tính toán thực hiện theo các bước sau:

1. Tính số mũ ban đầu của phép toán:

$$e = e_1 + e_2 - \text{bias}$$

2. Nhân phần bit Fraction (fraction): Ta nhân hai số trong hệ nhị phân:

$$i_1 = 1.\text{fraction}_1, \quad i_2 = 1.\text{fraction}_2$$

Tuy nhiên trong chương trình ta sẽ nhân dưới dạng bit thay vì thập phân như dạng trên. Vì vậy sau khi tách 23 bit Fraction, ta cần thêm 1 bit 1 tượng trưng cho phần nguyên bên ngoài

3. Chuẩn hóa: Sau khi tính toán, kết quả có thể phải chuẩn hóa, tức là dịch dấu phẩy và tăng mũ lên 1 đơn vị nếu cần. Trường hợp này xảy ra khi mà kết quả của việc nhân 2 fraction tạo ra 1 kết quả có phần nguyên lớn hơn 1.

   - Ví dụ 11.10101011011....

   Khi đó ta cần dịch dấu phẩy sang trái 1 lần để có dạng chuẩn hóa và đồng thời phải cộng 1 vào exponent.
   Sau khi chuẩn hóa, ta sẽ có giá trị r_{norm} và e_{norm}. Trong đó r_{norm} là phần sau dấu phẩy của kết quả, lấy đến 23 bit. e_{norm} là số Exponent đã được cộng vào sau bước chuẩn hóa hoặc sẽ là Exponent đã được giữ nguyên từ bước trước đó nếu không cần chuẩn hóa

4. Kiểm tra tràn số (Overflow/Underflow): Nếu $e_{\text{norm}} \geq 255$ hoặc $e_{\text{norm}} \leq 0$, chương trình sẽ dẫn đến ngoại lệ tương ứng.

5. Xác định bit dấu của phép toán: Dấu kết quả s sẽ bằng 0 nếu $s_1 = s_2$ và bằng 1 nếu $s_1 \neq s_2$. Ta sẽ Ex-or 2 bit dấu để có kết quả theo đúng dự tính

6. Kết quả xấp xỉ của phép nhân: Cuối cùng, kết quả xấp xỉ của phép nhân là một chuỗi 32 bit, với:

   - Trường dấu = s (1 bit) .
   - Trường mũ = e_{norm} (8 bit).
   - Trường thập phân = r_{norm} (23 bit) .

Tóm lại, quá trình tính toán thực hiện phép nhân số thực chính xác đơn theo chuẩn IEEE754 bao gồm các bước tính toán số mũ, nhân các phần thập phân, chuẩn hóa và làm tròn, kiểm tra tràn số, xác định dấu, và cuối cùng là lưu kết quả vào chuỗi 32 bit. Tóm tắt giải thuật:

1. **Cộng exponent (số mũ):** Cộng các Exponent và trừ đi *bias* để tính *exponent* mới.
2. **Nhân significand:** Tính tích của các *significand*.
3. **Chuẩn hóa:** Dịch phải kết quả và tăng *exponent* nếu cần chuẩn hóa.
4. **Kiểm tra overflow/underflow:** Xử lý ngoại lệ nếu xảy ra, nếu không tiếp tục.
5. **Làm tròn:** Rút gọn *significand* về số bit phù hợp.
6. **Xác nhận chuẩn hóa:** Lặp lại nếu kết quả chưa chuẩn hóa.
7. **Thiết lập dấu:** Dấu dương nếu các số cùng dấu, các trường hợp còn lại là dấu âm.
8. **Hoàn thành.**

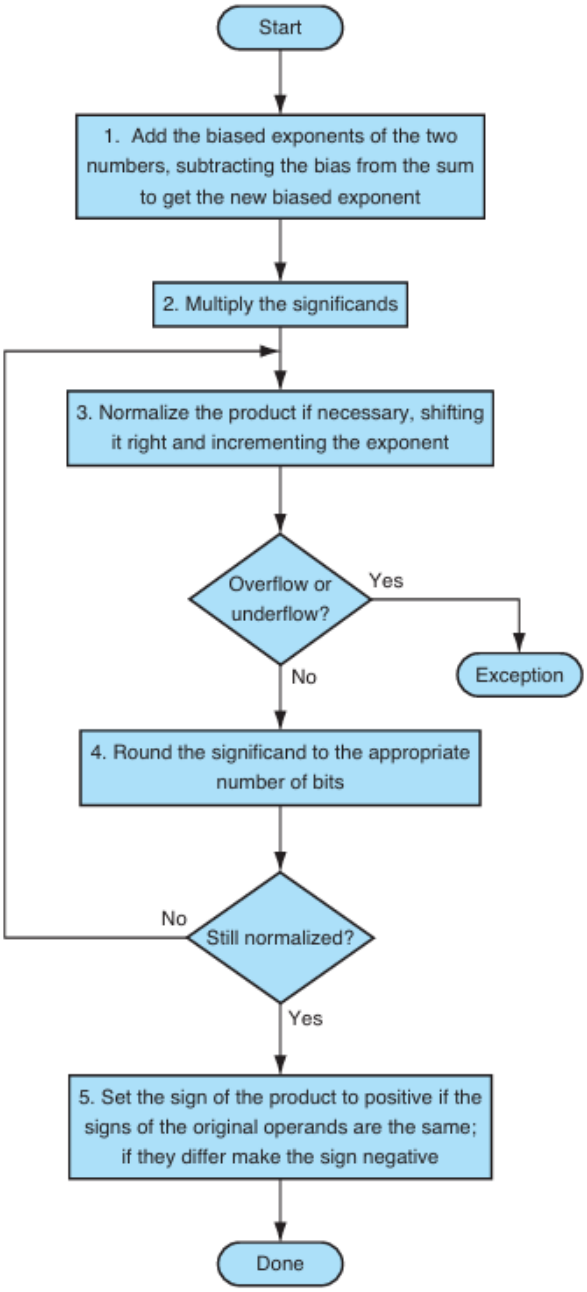

Figure 1: Lưu đồ của giải thuật nhân hai số thực chuẩn.[1]

[1]Nguồn: *Computer Organization and Design, Revised Fourth Edition, trang 258.*

3.2 Phân tích và hiện thực chi tiết

Mục tiêu: Chương trình thực hiện việc đọc hai số thực từ file, tách các thành phần (bit dấu, exponent, và fraction) của từng số thực, thực hiện các phép toán theo dạng IEEE-754, và ghép lại kết quả thành số thực mới. Sau đó, kết quả được in ra màn hình.

Các bước trong chương trình:

1. **Mở file và đọc hai số thực từ file:**
   - Mở file chứa các số thực dưới dạng nhị phân.
   - Đọc 4 byte (32-bit) cho mỗi số thực từ file vào bộ nhớ.
   - **Số thực bằng 0:** Nếu 1 trong 2 số thực đầu vào bằng 0, chương trình trả về kết quả là 0.
2. **Tách các số thực thành các thành phần:**
   - **Bit dấu :** Lấy bit đầu tiên của số thực.
   - **Bit Exponent:** Lấy 8 bit tiếp theo, điều chỉnh bias.
   - **Bit Fraction :** Lấy 23 bit còn lại và chuẩn hóa.
3. **Tính toán dấu :**
   - XOR giữa bit dấu của hai số thực để xác định dấu kết quả.
4. **Tính toán exponent :**
   - Cộng exponent của hai số thực và điều chỉnh bias.
   - Kiểm tra overflow hoặc underflow.
   - **Overflow:** Nếu exponent vượt quá giới hạn 255, thông báo overflow.
   - **Underflow:** Nếu exponent bé hơn 0, thông báo underflow.
5. **Tính toán fraction (đây cũng là bước phức tạp nhất trong toàn chương trình)**
   - **Thêm bit ẩn (1) vào Fraction:**
     - Để đảm bảo độ chính xác, ta thêm một bit ẩn (bit giá trị 1) vào phần Fraction của mỗi số thực. Điều này giúp chuẩn hóa phần lẻ của số, đảm bảo bit đầu tiên luôn là 1. Cụ thể, ta thực hiện thao tác bitwise OR với giá trị $0x800000$ (dạng nhị phân: 10000000000000000000000) để thêm bit 1 vào trước phần Fraction.
   - **Nhân hai phần Fraction:**
     - Sử dụng phép nhân hai số Fraction (không dấu) bằng lệnh `multu`, kết quả được lưu vào hai thanh ghi `HI` và `LO`. Phần cao (HI) chứa 16 bit đầu và phần thấp (LO) chứa 16 bit còn lại.
   - **Ghép kết quả nhân:**
     - Dịch phần cao của kết quả nhân (HI) sang trái 16 bit và phần thấp (LO) sang phải 16 bit để ghép lại thành một số 32 bit. Sau khi ghép, ta có phần Fraction 32 bit. Sau đó ta tiếp tục dịch phải 7 lần để có 25 bit. Sở dĩ ta xét 25 bit đầu trong 48 bit kết quả là do khi nhân 2 số có phần thập phân đến 23 số thì ta sẽ lấy phần thập phân lên 46 bit. Nghĩa là sẽ còn lại 2 bit phần nguyên, ta sẽ xét 2 bit phần nguyên này để quyết định xem nên lấy 23 bit Fraction kết quả thế nào

- **Chuẩn hóa Fraction:**
  - Sau khi có 25 bit trong thanh ghi, ta sẽ xét xem 2 bit cao nhất có biểu diễn cho giá trị 1 không, nếu đúng thì đã được chuẩn hóa.Nếu 2 bit này biểu diễn giá trị khác 1 (chẳng hạn 10.xxx hay 11.xxx thì 25 bit đang biểu diễn 10.xxx hay 11.xxx) ta phải sẽ phải chuẩn hóa.
  - Bước chuẩn hóa sẽ cộng 1 vào exponent và đồng thời bỏ bit cuối cùng trong 25 bit ta đã lưu, nghĩa là trong thanh ghi đang còn 24 bit.Để làm vậy ta sẽ dịch phải 1 bit. Sau đó ta lấy 23 bit đầu tiên trong thanh ghi , đó là kết quả Fraction ta mong muốn
  - Trong trường hợp ko cần chuẩn hóa, nghĩa là 2 bit cao nhất đang ở dạng 01 (nghĩa là 25 bit đang biểu diễn 01.xxxx) đây là dạng chuẩn, vì vậy ta sẽ bỏ qua bước chuẩn hóa và lấy 23 bit đầu tiên , là kết quả Fraction ta mong muốn
- **Kiểm tra Overflow và Underflow:**
  - Sau khi chuẩn hóa Fraction, ta kiểm tra xem exponent có vượt quá giới hạn (overflow) hoặc nhỏ hơn giới hạn cho phép (underflow). Nếu vượt quá, ta thông báo lỗi và nếu nhỏ hơn giới hạn, ta cũng xử lý tương tự.
- **Giữ lại 23 bit Fraction:**
  - Sau khi có kết quả ta sẽ lưu kết quả vào thanh ghi và tiếp tục tính toán

6. **Ghép lại thành số thực IEEE-754:**
   - Ghép các bit dấu, exponent và fraction lại để tạo số thực 32-bit.
7. **In kết quả ra màn hình:**
   - In kết quả cuối cùng dưới dạng số thực IEEE-754.
8. **Thoát chương trình.**

4 Tổng số lệnh, thời gian thực thi và kết quả kiểm thử

Loại lệnh	Số lượng
R-Type	23
I-Type	64
J-Type	5
Tổng cộng	**92**

Table 1: Tổng số lệnh trong chương trình chính

Ta sẽ kiểm thử độ chính xác của chương trình thông qua 15 testcases. Đồng thời, ở mỗi testcase, ta sẽ tính toán thời gian thực thi (T_{CPU}) được tính theo công thức:

$$T_{CPU} = \frac{IC \cdot CPI}{CR}$$

Với:

- IC: Tổng số lệnh.

- CPI: Chu kỳ thực thi trên mỗi lệnh (Với lệnh chuẩn CPI = 1).
- CR: Chu kỳ trên giây (CR = 1 GHz).

Bảng tổng kết

STT	Thừa số 1	Thừa số 2	Kết quả	R	I	J	Tổng lệnh	Thời gian (ns)
1	7.0	7.0	49.0	66	82	5	153	153
2	0.0	0.0	0.0	17	54	1	72	72
3	0.11	0.0	0.0	22	62	1	85	85
4	0.5	0.115	0.0575	63	77	5	145	145
5	177.1122	-2313.575	-409762.34	63	77	5	145	145
6	5439.4493	34275.2343	1.8643838E8	63	77	5	145	145
7	1E-38	1E-38	Underflow	31	61	3	95	95
8	4.5E20	12E13	5.4E34	66	82	5	153	153
9	-2.11E14	4.23E16	-8.925299E30	63	77	5	145	145
10	4.1E-5	3E21	1.22999995E17	63	77	5	145	145
11	0.3	-0.3	-0.09	63	77	5	145	145
12	0.12312	0.00012	1.47743995E-5	66	82	5	153	153
13	0.00001	0.00000001	9.999999E-14	63	77	5	145	145
14	200E-10	10E10	1999.9999	63	77	5	145	145
15	1E38	1E38	Overflow	30	59	3	92	92

Nhận xét: Thông qua bảng trên ta có thể thấy, thời gian chạy lớn nhất của chương trình là 153(ns) và trường hợp thường gặp nhất là 145(ns). Bên cạnh đó, những kết quả đưa của chương trình tương đối chính xác. Tuy nhiên, do sự giới hạn của số thực chuẩn chính xác đơn, một số trường hợp có số hạng đầu vào quá nhỏ (nhiều số lẻ) hoặc quá lớn sẽ gây ra tình trạng kết quả thiếu chính xác so với kết quả tính toán thực tế hoặc nặng hơn là overflow và underflow.

KẾT LUẬN

Trong bản báo cáo, nhóm đã trình bày về giải thuật cũng như là phân tích chi tiết khi hiện thực của bài toán nhân hai số thực chuẩn IEEE 745 chính xác đơn. Đồng thời, nhóm đã thống kế số lệnh thực thi, tính toán thời gian chạy và đánh giá kết quả kiểm định.

Thông qua bài tập lớn lần này, nhóm đã có cơ hội tìm hiểu về cách làm việc với số thực IEEE-754 và ứng dụng kiến thức đó để thiết kế một chương trình thực hiện các phép toán số học trên số thực 32-bit bằng MIPS, cụ thể là phép nhân hai số thực chuẩn, thông qua việc sử dụng công cụ MARS 4.5.

Trong quá trình thực hiện đề tài, nếu có một vài sai sót, nhóm em rất mong nhận được sự góp ý từ thầy để bài báo cáo được hoàn thiện hơn. Chúng em chân thành cảm ơn thầy!

TÀI LIỆU THAM KHẢO

1. Gill, D. (2013). *MIPS assembly language programming*. CreateSpace Independent Publishing Platform.

2. Patterson, D. A., & Hennessy, J. L. (2020). *Computer organization and design: The hardware/software interface* (5th ed.). Morgan Kaufmann.

3. Hyde, R. (2000). *The art of assembly language*. No Starch Press.

4. Kane, G., & Heinrich, J. (1992). *MIPS RISC architecture* (1st ed.). Prentice Hall.

5. Marschner, P. (2014). *The MARS simulator*. Retrieved from http://courses.missouristate.edu/kenvollmar/mars/