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

Kiến trúc Máy tính

Báo cáo bài tập lớn

Nhân 2 số thực chuẩn IEEE 754 chính xác đơn

Giảng viên hướng dẫn: Huỳnh Phúc Nghị

Sinh viên: Nguyễn Hoàng Long 2311906

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 11 2025

Contents

1 Mở đầu

Báo cáo này trình bày giải pháp hiện thực phép nhân hai số thực chuẩn IEEE 754 chính xác đơn trên kiến trúc MIPS mà không sử dụng các lệnh tính toán số thực sẵn có của MIPS. Chương trình đọc dữ liệu đầu vào từ file nhị phân "FLOAT2.BIN", thực hiện phép nhân và in kết quả ra màn hình.

2 Giải pháp hiện thực

Giải pháp dựa trên việc phân tích cấu trúc của số thực chuẩn IEEE 754 chính xác đơn (32-bit):

- Dấu (Sign): 1 bit (bit cao nhất)
- Số mũ (Exponent): 8 bit (tiếp theo sau bit dấu)
- Phần lẻ (Mantissa): 23 bit (phần còn lại)

Chương trình thực hiện các bước sau:

- Đọc dữ liệu từ file: Đọc hai số thực từ file "FLOAT2.BIN" vào bộ nhớ.
- Phân tích số thực: Tách riêng dấu, số mũ và phần lẻ của mỗi số.
- Điều chỉnh số mũ: Cộng hai số mũ lại với nhau và trừ đi 254 (do mỗi số mũ ban đầu đã được cộng thêm 127 để biểu diễn theo chuẩn IEEE 754). Thêm 127 vào kết quả để chuẩn hóa.
- Nhân phần lẻ: Thực hiện phép nhân hai phần lẻ bằng cách sử dụng các lệnh nhân nguyên số của MIPS (mult, mfhi, mflo).
- Chuẩn hóa: Kiểm tra bit cao nhất của kết quả nhân. Nếu bit này là 1, dịch kết quả sang phải 1 bit và tăng số mũ lên 1. Nếu cần thiết, chương trình sẽ dịch trái kết quả cho đến khi bit cao nhất là 1 (chuẩn hóa).
- Kiểm tra Overflow/Underflow: Kiểm tra xem số mũ có nằm trong khoảng cho phép hay không (0 đến 255). Nếu vượt quá, chương trình sẽ kết thúc.
- Ghép kết quả: Ghép dấu, số mũ và phần lẻ đã được tính toán lại thành một số thực IEEE 754.
- In kết quả: In kết quả ra màn hình.

3 Giải thuật

3.1 Định nghĩa số thực dấu chấm động IEEE 754

Cấu trúc của số thực 32 bit (độ chính xác đơn) bao gồm:

- 1 bit dấu (sign): biểu diễn dương hoặc âm với 0 là dương và ngược lại.
- 8 bit mũ (exponent): biểu diễn dưới dạng bias-127.
- 23 bit trị số (mantissa): xxx biểu diễn phần trị của số với dạng chuẩn hóa $1.xxx$

3.2 Hiện thực code

- Bên trong hàm main, qua các hàm syscall đọc file ghi, lưu các số thực vào các biến đã khởi tạo sẵn để chứa dữ liệu hai số thực, đóng file bin.

```
#mo file
addi $v0, $zero, 13            #syscall open file
#la $a0, input_file            #tai dia chi input_file vao $a0
lui $a0, 4097
ori $a0, $a0, 0
addi $a1, $zero, 0             #tai flags la 0
addi $a2, $zero, 0             #tai mode la 0
syscall
addu $s0, $v0, $zero           #di chuyen $v0 den $s0

#tai so thuc dau tien
addi $v0, $zero, 14            #syscall read file
addu $a0, $s0, $zero           #di chuyen $s0 den $a0
#la $a1, first_num             #tai dia chi first_num vao $a1
lui $a1, 4097
ori $a1, $a1, 12
addi $a2, $zero, 4             #tai 4 bytes
syscall

#tai so thuc thu hai
addi $v0, $zero, 14            #syscall read file
addu $a0, $s0, $zero           #di chuyen $s0 den $a0
#la $a1, second_num            #tai dia chi second_num vao $a1
lui $a1, 4097
ori $a1, $a1, 16
addi $a2, $zero, 4             #tai 4 bytes
syscall
```

Với các số thực lưu trong `first_num` và `second_num`

- Kế đến, gọi vào hai biến vừa lưu ở trên và đưa lần lượt hai số thực vào hai thanh ghi `$t0` và `$t1`, đồng thời tạo bước nhảy đến dòng `check_zero` với mục đích nếu một trong hai bằng 0 thì lập tức trả về kết quả 0.

- Thực hiện tách 3 phần của số thực ra như đã được trình bày ở trên bao gồm sign, exponent và mantissa. Các thanh ghi chứa dữ liệu này của số đầu tiên lần lượt là `$s0`, `$s1` và `$s2`. Tương tự với số thực còn lại và các thanh ghi lần lượt từ `$s3` đến `$s5`.

```
#tach so thuc thu nhat (s0 la sign, s1 la exponent, s2 la mantissa)
#andi $s0, $t0, 0x80000000     #tach bit dau cua so dau tien
lui $at, 0x8000
ori $at, $at, 0
and $s0, $t0, $at
srl $s0, $s0, 31               #giu  1 bit dau
srl  $s1, $t0, 23              #tach phan exponent
andi $s1, $s1, 0xFF            #giu 8 bit
#andi $s2, $t0, 0x007FFFFF     #tach phan mantissa
lui $at, 0x007F
ori $at, $at, 0xFFFF
and $s2, $t0, $at
#ori $s2, $s2, 0x00800000      #them so 1 ngam dinh
lui $at, 0x0080
ori $at, $at, 0
or $s2, $s2, $at
```

- Sau đó gọi hàm thực thi nhân hai số thực chuẩn IEEE với độ chính xác đơn để thực thi, rồi lưu vào biến `result` để in ra kết quả trên màn hình.

Hàm `multiple_floats`:

- Thực hiện xor hai thanh ghi `$s0` và `$s3$` để có được dấu của kết quả.
- Tính phần mũ bằng cách cộng hai phần của hai số với nhau và trừ đi bias là 127.
- Tiếp theo là phần nhân hai mantissa với đoạn code sau:

```
#tinh phan mantissa
#nhan phan mantissa
mult $s2, $s5
mfhi $t3                    #lay phan tren
mflo $t5                    #lay phan duoi
sll $t3, $t3, 16            #dich trai phan tren
srl $t5, $t5, 16            #dich phai phan duoi
or $t3, $t3, $t5            #gop phan tren va duoi
j normalize_mul
```

Tách hai trị số thu được sau khi nhân từ thanh ghi `HI` và`LO`. Sau đó gộp cả hai phần lại cho ra kết quả phần mantissa khi đã dịch các bit về đúng vị trí.

Các hàm hỗ trợ:

- Hàm `normalize_mul` thực hiện chuẩn hóa phép nhân bằng cách kiểm tra bit cao nhất của trị số. hàm sé gọi `left` để dịch trái đến khi bit cao nhất là 1.
- Hàm `check_under_and_over` để kiểm tra kết quả có bị Underflow hoặc Overflow hay không (nếu số mũ của nó dưới 0 và số mũ vượt quá 255, lần lượt) và trả về giá trị 0 hoặc là vô cùng, có bước nhảy đến `underflow` và `overflow` cho mỗi trường hợp.
- Hàm `pack_result` thực hiện gộp các thanh ghi chứa các phần sign, exponent và mantissa của kết quả để trả về giá trị 32 bit (sau khi đã kiểm tra qua các điều kiện khác) rồi đưa vào biến `result` sẵn sàng để cho ra kết quả.

Sơ đồ giải thuật:

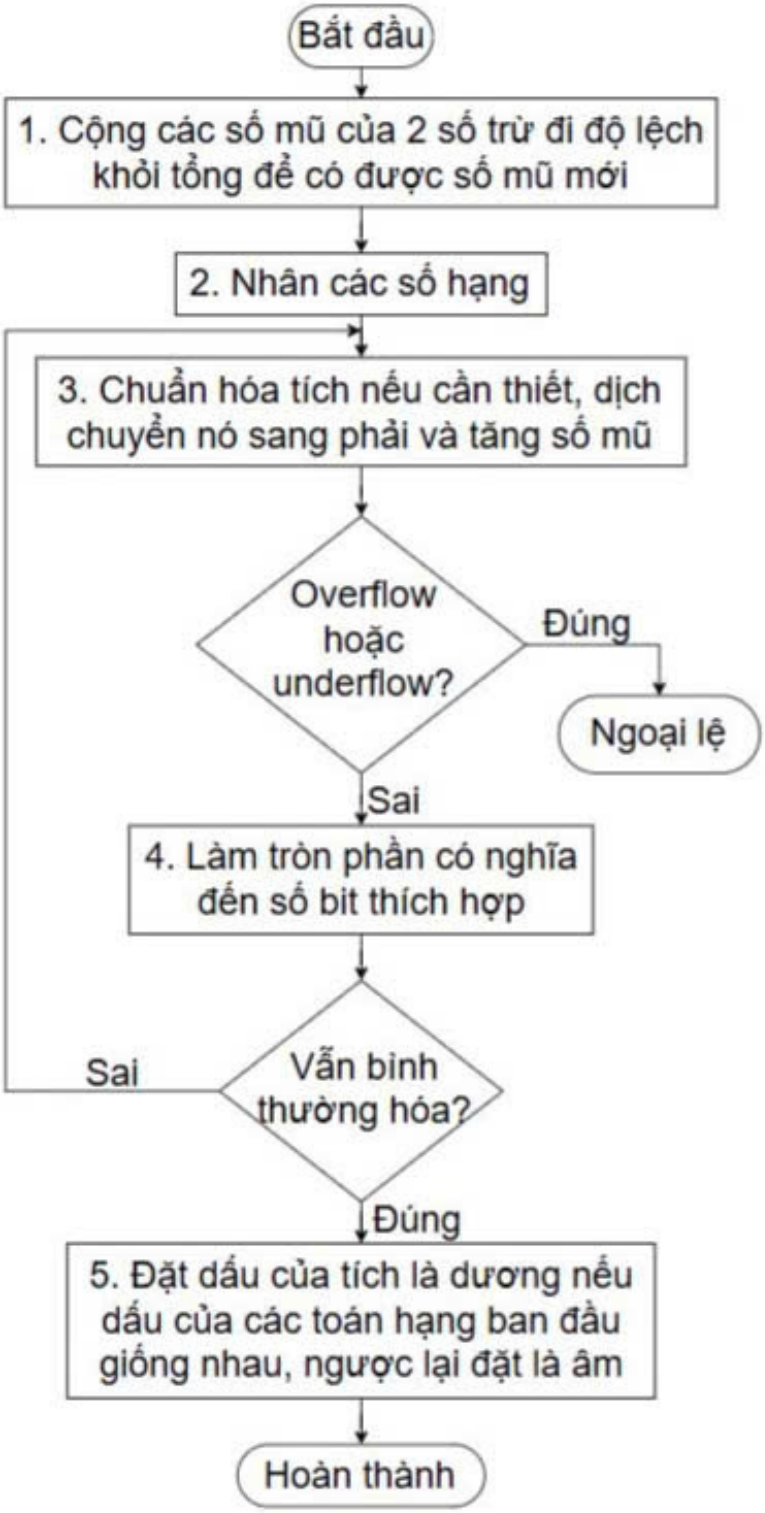


Figure 3.1: Sơ đồ ngắn gọn giải thuật đưa ra

4 Thống kê

Phân loại lệnh

- R-type (Register-to-register): Các lệnh thực hiện phép toán giữa các thanh ghi. Ví dụ: add, sub, and, or, xor, sll, srl, mult, mfhi, mflo.
- I-type (Immediate): Các lệnh sử dụng một giá trị immediate (hằng số) cùng với một thanh ghi. Ví dụ: addi, ori, lui, lw, beq, j.
- J-type (Jump): Các lệnh nhảy đến một địa chỉ khác trong chương trình. Ví dụ: j, jal.
- syscall: Lệnh hệ thống để tương tác với hệ điều hành (mở file, đọc file, ghi file, in ra màn hình, kết thúc chương trình).

Tùy theo dữ liệu của các test case mà chương trình chạy ra các lệnh khác nhau. Ở đây, chúng ta bỏ qua các trường hợp số thực vô cùng và NaN (Not a Number) bởi vì file FLOAT2.BIN được tạo ra dựa trên hai số thực được nhập vào cho trước.
Ta xét các trường hợp sau:

1. Nhân hai số dương thông thường.
2. Nhân hai số âm thông thường.
3. Nhân một số âm và một số dương.
4. Nhân 0 với một số.
5. Nhân một số với 0.
6. Nhân số dương rất nhỏ với số dương rất lớn.
7. Nhân số dương với số âm rất lớn.
8. Nhân số nhỏ với số cực kỳ nhỏ.

Ta có thống kê sau:

1. Nhân hai số dương thông thường.(3.14, 2.71)
    - R-type: 40
    - I-type: 53
    - J-type: 5
2. Nhân hai số âm thông thường.(-17.09, -5.23)
    - R-type: 44
    - I-type: 54
    - J-type: 5
3. Nhân một số âm và một số dương.(-7.63, 4.2)
    - R-type: 40
    - I-type: 53
    - J-type: 5
4. Nhân 0 với một số.(0, 6.21)
    - R-type: 10
    - I-type: 20
    - J-type: 1
5. Nhân một số với 0.(-10.4, 0)
    - R-type: 10
    - I-type: 23
    - J-type: 1
6. Nhân số dương rất nhỏ với số dương rất lớn.(1e-38, 1e+38)
    - R-type: 40
    - I-type: 53
    - J-type: 5
7. Nhân số dương với số âm rất lớn.(3.14, -1e+38)

- R-type: 44
- I-type: 54
- J-type: 5

8. Nhân số nhỏ với số cực kỳ nhỏ.(1e-40, 1e-40)

- R-type: 39
- I-type: 51
- J-type: 4

5 Thời gian chạy của chương trình

Với mỗi testcase được đưa ra ở mục trước, ta tính được thời gian chạy cho mỗi chương trình là khác nhau, ở đây ta xét đến testcase nhân hai số âm với nhau (với số lượng lệnh nhiều nhất ở trên).

5.1 Single Clock Cycle

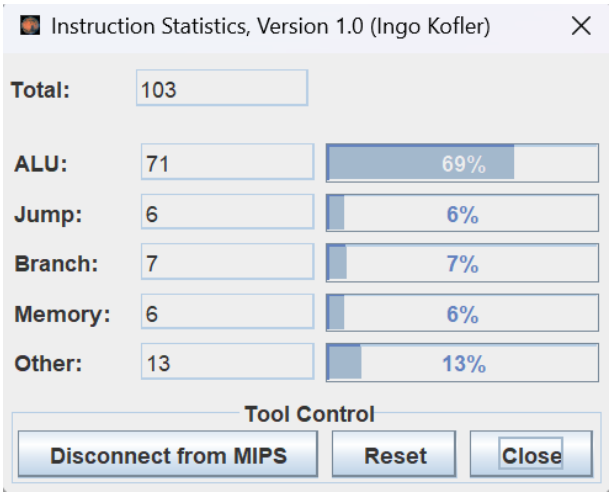
Sử dụng công cụ Instruction Statistics trong MARS. Ta được bảng sau:

ALU	71
Jump	6
Branch	7
Memory	6
Other	13

Với Clock Rate giả sử được cho là 1 GHz, ta có công thức:
$CPUtime = IC * ClockCycleTimes$

- IC tổng là 103
- $ClockCycleTime = 1/ClockRate = 1$ (ns)
- Mỗi lệnh thực hiện 5 chu kì nhỏ.

Ta có được $CPUtime = 103 * 5 * (1ns) = 515$ (ns)



5.2 Multi Cycle

Từ thống kê trên, ta có:

- CPI của Multi Cycle của hệ thống là:
  ALU chiếm 69%
  Jump chiếm 6%
  Branch chiếm 7%
  Memory chiếm 6%
  Other chiếm 12%

- Với ALU cần 4 chu kì, Jump cần 2 chu kì, Branch cần 3 chu kì, Other cần 4 chu kì (trung bình), Memory cần 5 chu kì.
  Ta có $AverageClockCycleTimes = 0.69*4+0.06*2+0.07*3+0.06*5+0.12*4 = 3.87$

- Với Clock Rate là 1 GHz, suy ra Clock Cycle time là 1 ns

Ta có được $CPUtime = IC * ClockCycleTimes = 103 * 3.87 * (1ns) = 398.61$ (ns)

5.3 Pipeline

Bộ xử lý Pipeline chia quá trình thực thi lệnh thành 5 bước, mỗi bước thực thi trong trong một chu kỳ. Ta có lệnh đầu tiên trong 5 chu kì, các lệnh sau hoàn thành trong 1 chu kì với Clock Cycle Time như ở trên Multi Cycle là 1 ns.
$CPUtime = (5 + 103 - 1) * 1 * (1ns) = 107$ (ns)

5.4 Tổng hợp

Dưới đây là bảng thời gian cho các testcase

	Single Cycle	Multi Cycle	Pipeline
(3.14, 2.71)	490 ns	377.3 ns	102 ns
(-17.09, -5.23)	515 ns	398.61 ns	107 ns
(-7.63, 4.2)	490 ns	377.3 ns	102 ns
(0, 6.21)	155 ns	122 ns	35 ns
(-10.4, 0)	170 ns	153 ns	38 ns
(1e-38, 1e+38)	490 ns	377.3 ns	102 ns
(3.14, -1e+38)	515 ns	398.61 ns	107 ns
(1e-40, 1e-40)	470 ns	366 ns	98 ns

Table 5.1: Thống kê cho các testcase

6 Kết quả kiểm thử

Dưới đây là kết quả của các testcase khi được thực thi

1. 3.14 x 2.71

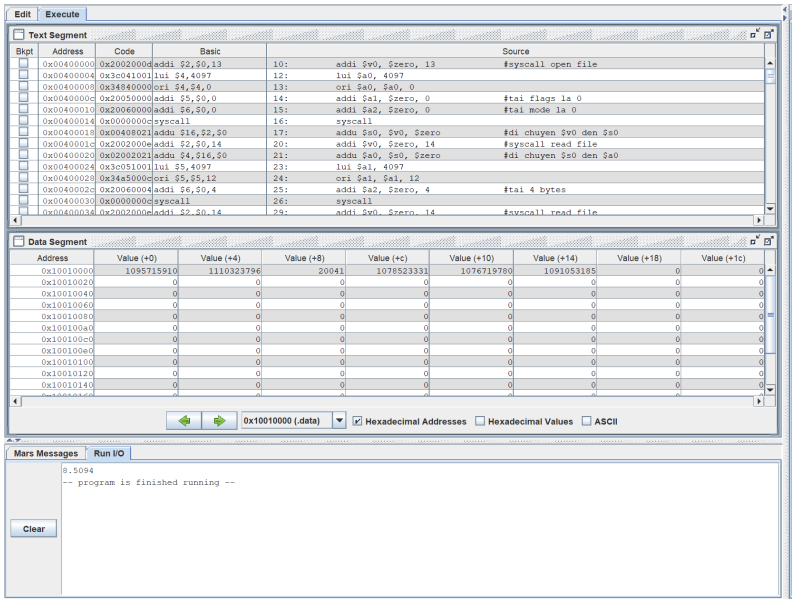


Figure 6.1: Nhân hai số dương thông thường.

2. -17.09 x -5.23

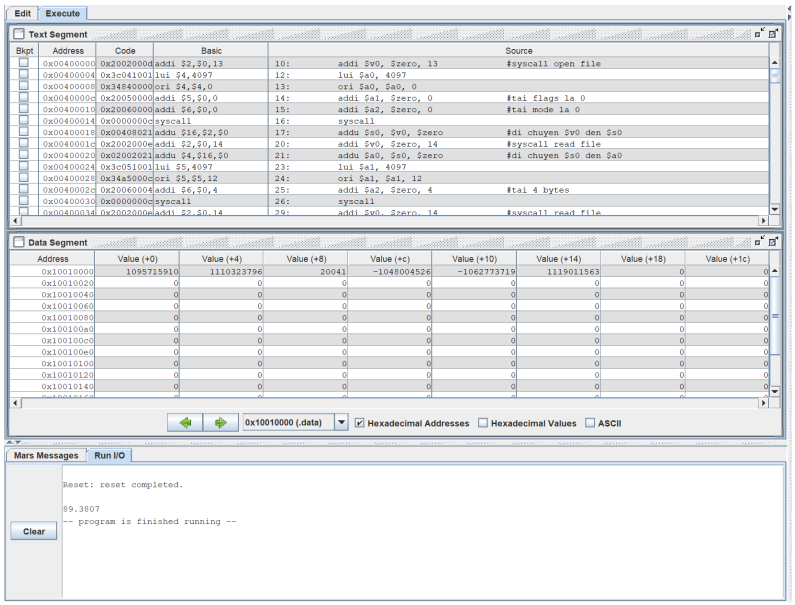


Figure 6.2: Nhân hai số âm thông thường.

3. -7.63 x 4.2

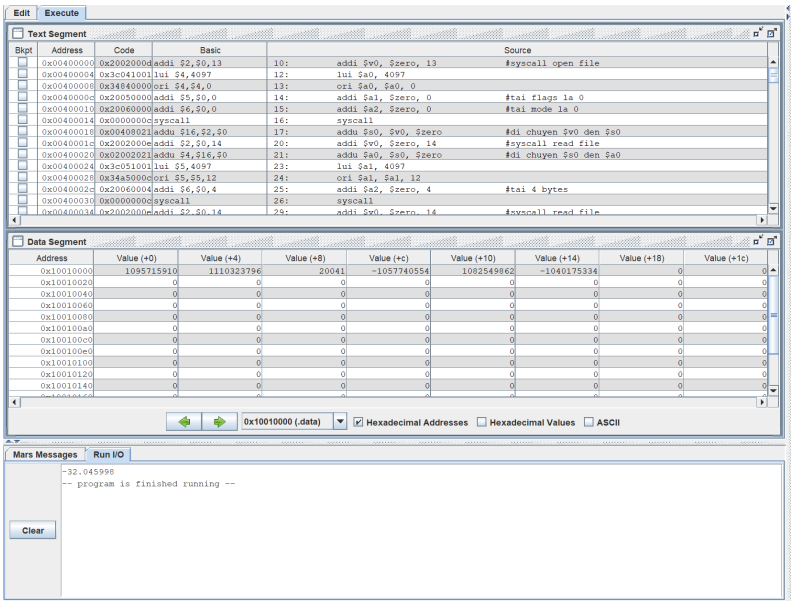


Figure 6.3: Nhân một số âm và một số dương.

4. 0 x 6.21

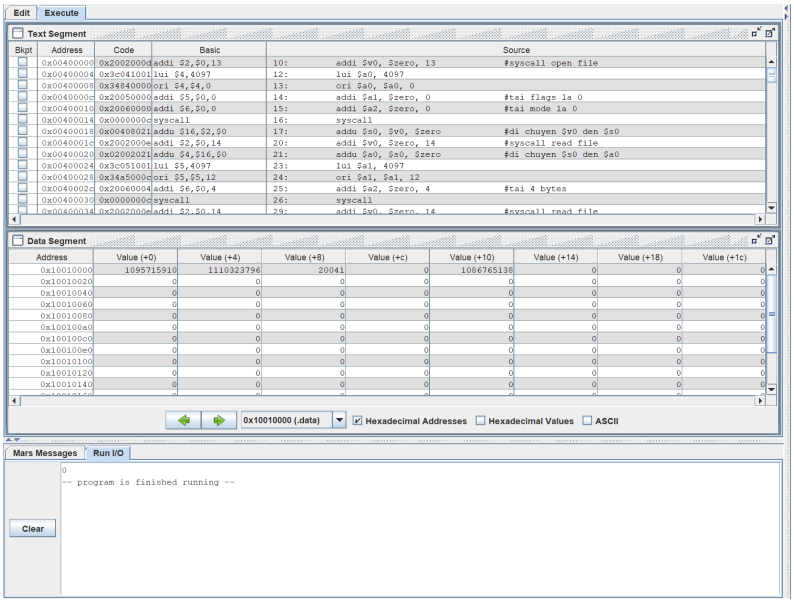


Figure 6.4: Nhân 0 với một số.

5. -10.4 x 0

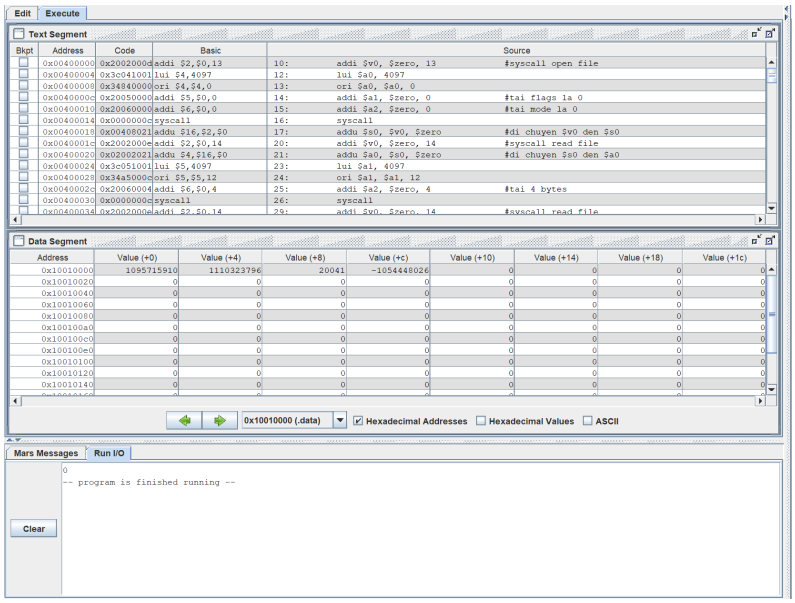


Figure 6.5: Nhân một số với 0.

6. 1e-38 x 1e+38

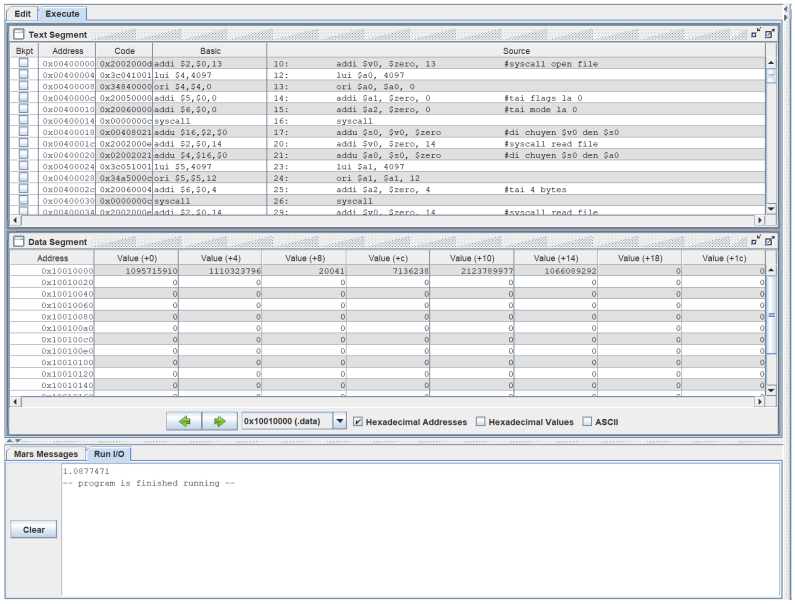


Figure 6.6: Nhân số dương rất nhỏ với số dương rất lớn.

7. 3.14 x -1e+38

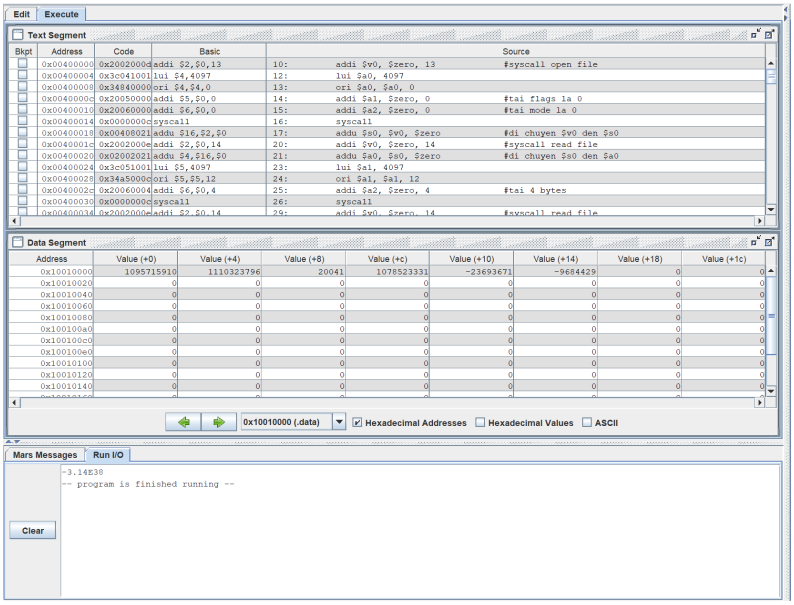


Figure 6.7: Nhân số dương với số âm rất lớn.

8. 1e-40 x 1e-40

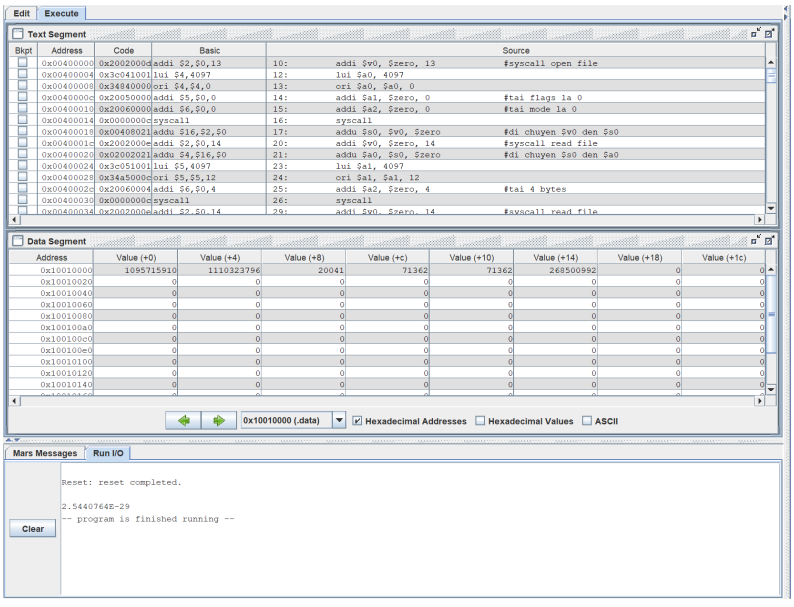


Figure 6.8: Nhân hai số cực kỳ nhỏ.

7 Đánh giá kết quả

7.1 Kết quả

Sau khi triển khai chương trình thực hiện các phép tính trên hai số thực theo chuẩn IEEE 754 (Single Precision) trong ngôn ngữ Assembly MIPS, nhóm thu được các kết quả sau:

- Phép toán nhân hoạt động chính xác theo chuẩn IEEE 754.
- Chương trình hiển thị kết quả đúng trên màn hình theo yêu cầu đề bài.
- Đã xử lí các trường hợp Underflow và Overflow, đồng thời dữ liệu đầu vào là 0.

Bên cạnh đó, bài làm còn có những điểm yếu:

- Thời gian và khả năng sai khi thực thi của một số trường hợp ngoại lệ còn cao do yêu cầu kiểm tra và xử lý lỗi nhiều bước.
- Cần thêm những biện pháp tối ưu giải thuật để hoạt động ít lỗi hơn.
- Chưa sử dụng tối ưu được các thanh ghi được cho.
- Nhóm đã cố gắng sử dụng các lệnh cơ bản cũng như tách các lệnh `pseudo` cho chương trình, mặc dù vẫn còn các lỗi nhỏ trong cách trình bày dòng lệnh.

7.2 Tổng kết

Chương trình đã hoàn thành mục tiêu đề ra là thực hiện nhân hai số thực theo chuẩn IEEE và xử lí theo thuật toán đề ra, tận dụng tốt các lệnh MIPS trong ứng dụng MARS, xử lí số không dùng các lệnh số thực và cho ra kết quả đúng như mong đợi theo như cách tiếp cận được đề ra. Nhóm xin chân thành cảm ơn.