ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

BÁO CÁO

BÀI TẬP LỚN KIẾN TRÚC MÁY TÍNH

ĐỀ TÀI 3 PHẦN HỢP NGỮ

GVHD: Nguyễn Xuân Quang
SV thực hiện: Nguyễn Đức An – 2010102
Trần Chí Công – 2010170
Phan Hiền Mai – 2010405

Tp. Hồ Chí Minh, 01/11/2021

Contents

1 Đề tài

Không sử dụng các lệnh số thực, tính thương 2 số thực dấu chấm động chuẩn IEEE 754 độ chính xác đơn (IEEE 754 single-precision) a và b do người dùng nhập vào và hiển thị kết quả.

2 Mục tiêu

- Hiểu về hoạt động của máy tính, đặc biệt khi xử lý các lệnh số thực.
- Biết sử dụng và tối ưu chương trình hợp ngữ.
- Biết cách đánh giá hiệu năng của một chương trình máy tính.

3 Yêu cầu

- Sử dụng hợp ngữ MIPS (tập lệnh MIPS32) viết và chạy được chương trình trên phần mềm MARS MIPS
- Trong chương trình phải có thực hiện các thao tác gọi hàm (procedure call).
- Sử dụng các thanh ghi theo đúng quy ước về thanh ghi.
- Thống kê về số lệnh, loại lệnh, tính thời gian thực thi chương trình. Giả định bộ xử lí hoạt động ở tần số 2GHz

4 Cơ sở lý thuyết & Giải pháp

Để giải quyết được bài toán trên, đầu tiên, ta cần phải tìm hiểu sơ lược về cơ sở lí thuyết cần biết để có thể xử lí vấn đề, cần tìm hiểu về dấu chấm động là gì, hay biểu diễn dấu chấm động với độ chính xác đơn là như thế nào ? Bên cạnh đó, ta phải hiểu được bản chất của thuật toán tính thương 2 số thực, cách máy tính lưu trữ và xử lý số thực với độ chính xác đơn.

4.1 Single Precision trong biểu diễn Floating Point:

Dấu chấm động (Floating Point) được dùng để chỉ một hệ thống biểu diễn số mà trong đó sử dụng một chuỗi chữ số (hay bit) để biểu diễn một số thực (non-integral numbers), như kiểu float và double trong ngôn ngữ C, bao gồm cả những số rất nhỏ và những số rất lớn.

Ưu điểm của cách biểu diễn dấu phẩy động là nó cho phép biểu diễn một tầm giá trị rộng hơn nhiều so với cách biểu diễn dấu phẩy tĩnh.

Floating Point được định chuẩn bởi Tổ chức IEEE (754 – 1985) và được phát triển nhằm đáp ứng tiêu chuẩn trình bày thống nhất, dễ sử dụng và chuyển đổi giữa các bộ mã trong khoa học. Vì thế, nó ngày càng trở nên phổ biến và thông dụng hiện nay.

Hiện nay có 2 cách biểu diễn Floating Point:

- Độ chính xác đơn (Single Precision, 32-bit)
- Độ chính xác kép (Double Precision, 64-bit)

Tương ứng với 2 cách biểu diễn trên, Trong lập trình ngôn ngữ C, các số thực dạng float sẽ được định dạng theo kiểu độ chính xác đơn, còn các số dạng double sẽ được định dạng theo kiểu độ chính xác kép.

Theo yêu cầu của bài tập lớn lần này, chúng ta cần thực hiện với độ chính xác đơn (Single Precision), thế nên ta sẽ tìm hiểu sơ lược về cách biểu diễn này. Biểu diễn dấu chấm động theo chuẩn IEEE 754 với độ chính xác đơn:

1 bit	8 bits	23 bits
S	Exponent	Fraction

Số thực X có thể biểu diễn ở dạng Single Precision như sau:

$$X = (-1)^S \cdot (1 + Fraction) \cdot 2^{Exponent - Bias}$$

Trong đó :

- **S (1 bit)** là bit dấu (S = 1 là số âm , S = 0 là số dương).
- **Exponent (8 bits)** là mã excess – 127 của phần mũ.
  Exponent = Actual Exponent + Bias (Bias là độ lệch, trong độ chính xác đơn Bias = 127).
- **Fraction (23 bits)** là phần lẻ của phần định trị.

4.2 Tính thương 2 số thực dấu chấm động chuẩn IEEE 754 độ chính xác đơn (single-precision)

Phép chia 2 số thực X và Y

$$X = m_X \cdot 2^a$$

$$Y = m_Y \cdot 2^b$$

được biểu diễn như sau:

$$\frac{X}{Y} = \frac{m_X}{m_Y} \cdot 2^{a-b}$$

Thuật toán cơ bản cho việc tính thương 2 số thực bao gồm các bước cơ bản sau (lấy tổng quát là bài toán trên):

- Kiểm tra trường hợp Y = 0, nếu Y = 0 ngừng phép toán, nếu không thì tiếp tục tính các bước tiếp theo.
- Tính Actual exponent của kết quả bằng cách tính hiệu của a và b.
- Thực hiện phép chia (mX/mY) để quyết định dấu của kết quả.
- Đơn giản hóa và làm tròn kết quả nếu cần thiết.

Ví dụ: Cho 2 số X và Y, X = 91.34375 và Y = 0.14453125. Tính X/Y.

$$X = 91.34375_{10} = 1011011.01011_2 = 1.01101101011_2 \cdot 2^6$$

$$Y = 0.14453125_{10} = 0.00100101_2 = 1.00101 \cdot 2^{-3}$$

Biểu diễn IEEE 754 của X với độ chính xác đơn:

0	1000 0101	0110 1101 0110 0000 0000 000
S	Exponent	Fraction

Biểu diễn IEEE 754 của Y với độ chính xác đơn:

0	0111 1100	0010 1000 0000 0000 0000 000
S	Exponent	Fraction

$$\frac{X}{Y} = \frac{1.01101101011_2}{1.00101_2} \cdot 2^{6-(-3)} = \frac{1.01101101011_2}{1.00101_2} \cdot 2^9 = 1.001111_2 \cdot 2^9$$

$$Exponent = 127 + 9 = 136 = 10001000_2$$

Thuật toán chia hai số nhị phân:

```
                1.001111
          ┌─────────────
1.00101   │ 1.01101101011
        - 1 00101
          ──────
           1000101
         -  100101
          ───────
           1000000
         -  100101
          ───────
            110111
          -  100101
           ────────
             100101
           -  100101
            ────────
                   0
```

Output:

0	1000 1000	0011 1100 0000 0000 0000 000
S	Exponent	Fraction

Vậy $X/Y = (-1)^0 \cdot (1 + 0.234375) \cdot 2^9 = 632$

5 Giải thuật & Đếm số lệnh và tính thời gian thực thi chương trình

5.1 Ý tưởng

Dựa vào cơ sở lý thuyết trên, chúng ta sẽ tính toán lần lượt trên 3 thành phần của số thực : **Sign bit, Exponent và Fraction**.

Sau đó ta sẽ áp dụng công thức $X = (-1)^S \cdot (1 + Fraction) \cdot 2^{Exponent-Bias}$ (Bias = 127) với các thành phần sau khi xử lý để tính ra kết quả thương ở dạng thập phân.

Ngoài ra, nhóm sẽ xử lý một số ngoại lệ **(Exception)** khi tính toán với số thực.

5.2 Giải thuật

5.2.1 Quy ước chuẩn thanh ghi

- Số bị chia **(Dividend)** sẽ được lưu vào thanh ghi **$s0**
- Số chia **(Divisor)** sẽ được lưu vào thanh ghi **$s1**
- Thương **(Quotient)** sẽ được lưu vào thanh ghi **$s2**
- Các kết quả trung gian trong quá trình tính toán sẽ được lưu vào các thanh ghi tạm.

5.2.2 Xử lý ngoại lệ

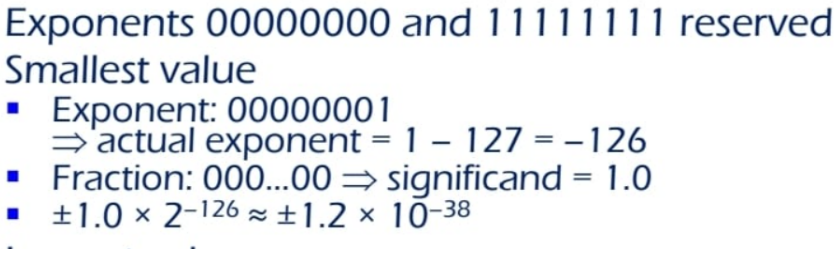
Theo quy ước của nhóm, có 4 ngoại lệ khi tính thương 2 số thực dấu chấm động chuẩn IEEE 754 độ chính xác đơn (IEEE 754 single-precision) :

- **Infinity**
  - **Nếu input đầu vào không hợp lệ (vượt quá 32-bit)** thì lập tức ném ra lỗi "Infinity"
  - Nếu **Dividend** != 0 && **Divisor** == 0 hoặc giá trị của thương tiến ra vô cùng thì cũng ném ra lỗi "Infinity" (implement theo chuẩn MIPS trên MARS của hàm **div.s**)

- Giải pháp : Ném ra lỗi **"Infinity"**

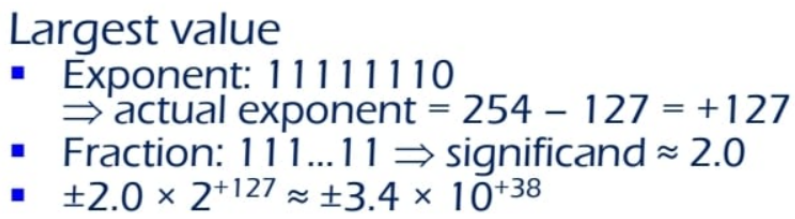
- **NAN**
  - Khi tính giá trị thương **Dividend/Divisor** nếu **(Dividend->0) && (Divisor->0)** hoặc **(Dividend->∞) && (Divisor->∞)** thì đây là lỗi NAN.
  - Giải pháp : Ném ra lỗi **"NAN"**

- **Underflow**



  - **Nếu input hợp lệ** và giá trị thương tính được nhỏ hơn cận dưới của vùng giá trị số thực tính với độ chính xác đơn (IEEE 754 single-precision) thì đây là lỗi Underflow
  - Giải pháp : Ném ra lỗi **"Underflow"**

- **Overflow**



  - **Nếu input hợp lệ** và giá trị thương tính được lớn hơn cận trên của vùng giá trị số thực tính với độ chính xác đơn (IEEE 754 single-precision) thì đây là lỗi Overflow
  - Giải pháp : Ném ra lỗi **"Overflow"**

5.2.3 Xử lý Sign Bit

Để lấy bit dấu, ta thực hiện các thao tác sau:

- Đầu tiên ta dịch phải 31 bit số bị chia và số chia để lấy bit dấu (bit thứ 31) sau đó lần lượt lưu vào hai thanh ghi tạm **$t0** và **$t1**.
- Kiểm tra xem bit dấu của số bị chia và số chia có giống nhau hay không ? Nếu 2 bit dấu khác nhau => Đây là phép chia hai số trái dấu nên dấu của thương là âm (Sign bit = 1), ngược lại Sign bit = 0.

5.2.4 Xử lý Exponent

Đến bước này, chúng ta sẽ tập trung tính toán dựa trên công thức sau:

$$\frac{X}{Y} = \frac{m_X}{m_Y} \cdot 2^{a-b}$$

Để xử lý Exponent, ta thực hiện các thao tác sau:

- Đầu tiên, ta sẽ tách Exponent của số bị chia và số chia ra (8 bits) lưu vào biến tạm.
- Tiếp theo, trừ Exponent của số bị chia cho exponent của số chia, kiểm tra xem mũ của thương có rơi vào các trường hợp ngoại lệ trên hay không ? Nếu có thì ném ra ngoại lệ, không thì lưu lại giá trị hiệu của exponent số bị chia và số chia để dùng cho tính kết quả thương sau này.

5.2.5 Xử lý Fraction và giải thuật chia

Để xử lý Fraction, ta thực hiện các thao tác sau:

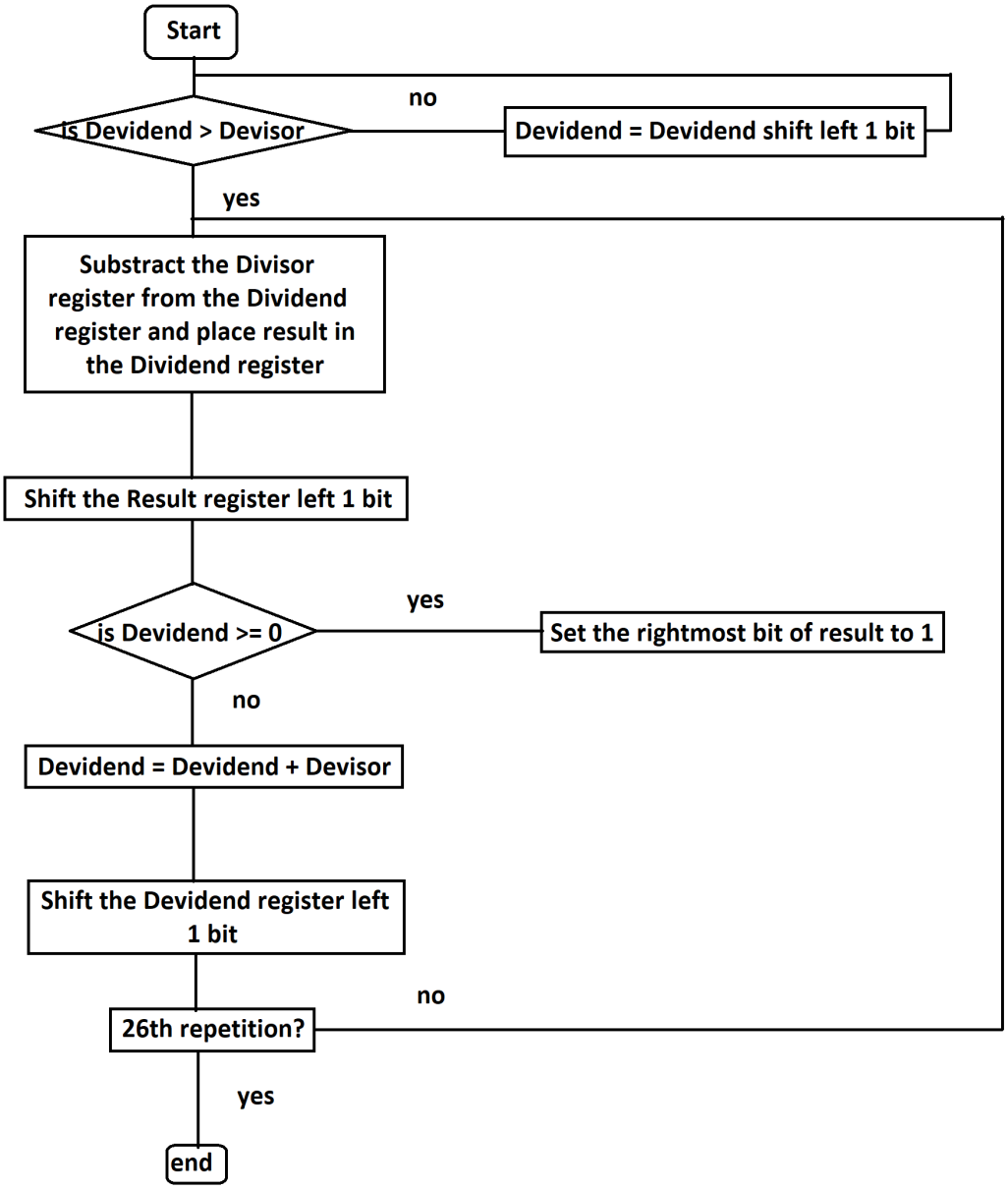
- Đầu tiên, ta sẽ tách Fraction của số bị chia và số chia ra (23 bits) lưu vào biến tạm.
- Để thực hiện phép chia, ta cần tính **Mantissa**[1] của số bị chia và số chia bằng cách thêm một bit 1 làm bit trọng số cao nhất trước phần Fraction.

Giải thuật chia **Mantissadivision** của số bị chia và số chia:

- Nếu số bị chia nhỏ hơn số chia ta tiến hành dịch trái dấu chấm động cho đến khi nào số bị chia >= số chia **(Dividend >= Divisor)**.

[1]Mantissa = 1 + Fraction

– Tiếp theo ta tiến hành chia theo giải thuật sau:



Cuối cùng là kết hợp ba phần Exponent, Fraction, Sign Bit đã xử lý lại để tính ra kết quả thương ở dạng thập phân.

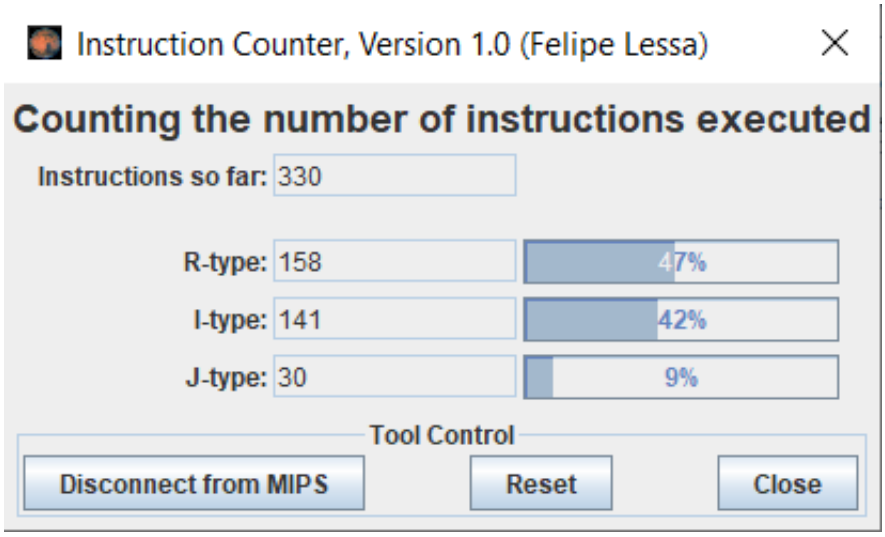
5.3 Đếm và thống kê số lệnh và tính thời gian thực thi của chương trình

Vì MIPS là tập lệnh có kiến trúc RISC nên mỗi lệnh đều có thể giải quyết được trong một chu kỳ, do vậy trong quá trình tính toán ta có thể xem CPI ≈ 1. Giả sử chương trình chạy trên máy tính có Clock Rate = 2 GHz.

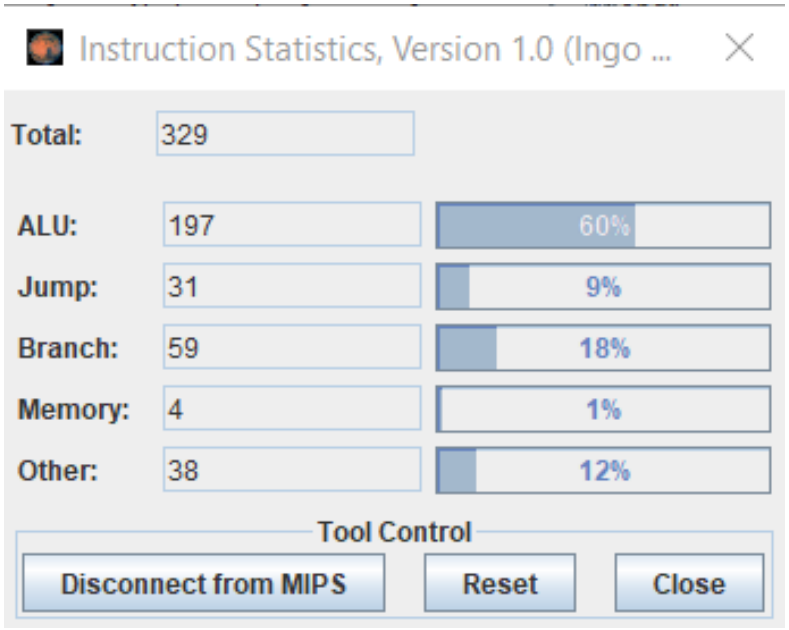
$$TimeExecuted = \frac{CPI \cdot InstructionsExecuted}{ClockRate}$$

Chúng ta sẽ đếm và thống kê số lệnh thực thi của một số testcases từ đơn giản đến phức tạp.

- **Testcase 1: 4.5 / 1.5**



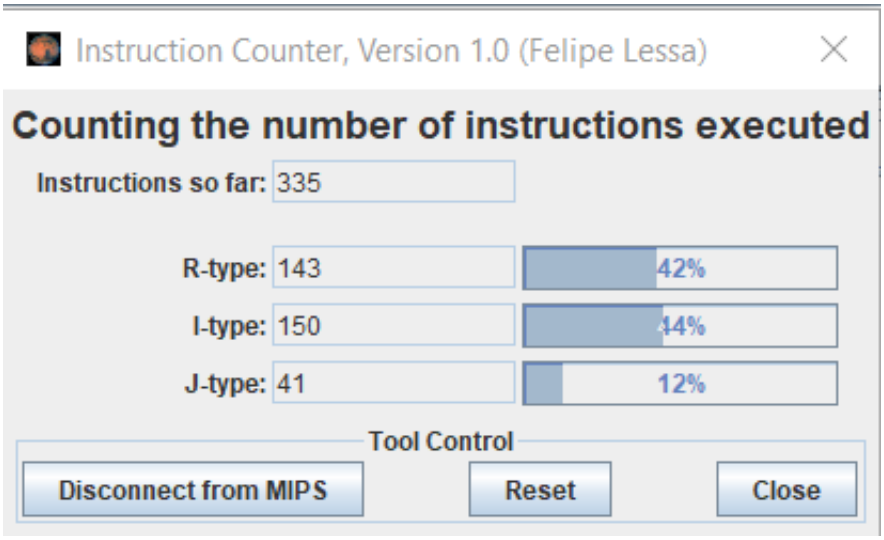
Instruction Counter



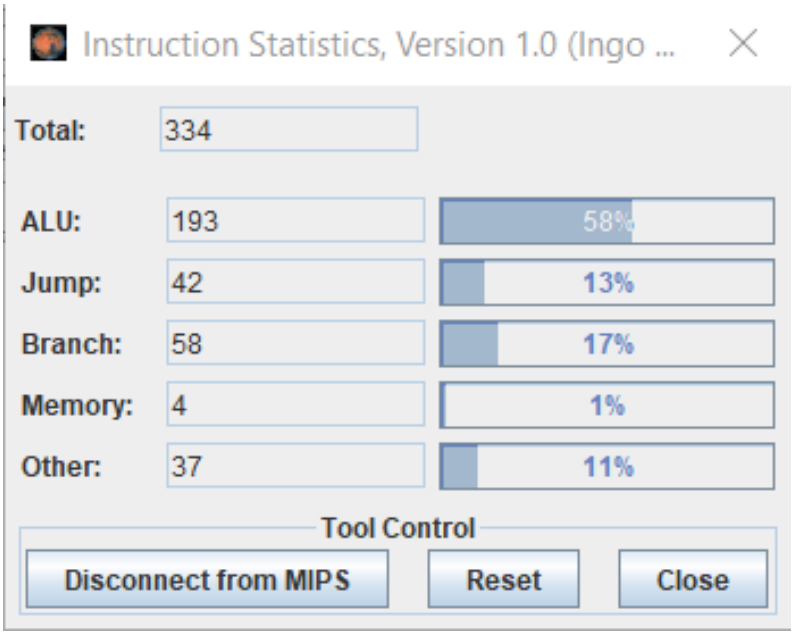
Instruction Statistics

Testcase 1 : Time Executed ≈ 165 ns

- **Testcase 2: 1.2345 / 2.3456**



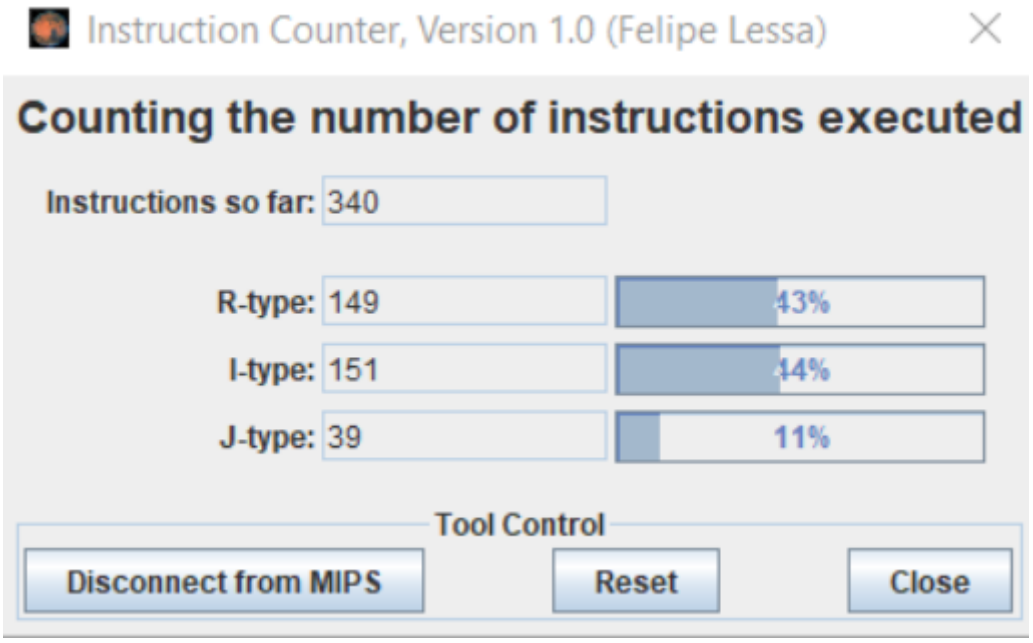
Instruction Counter



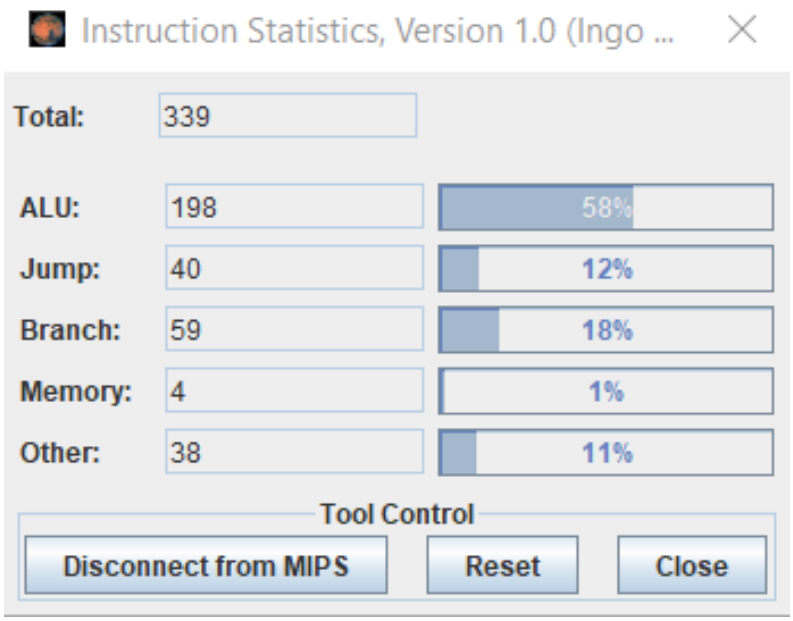
Instruction Statistics

Testcase 2 : Time Executed ≈ 167.5 ns

- **Testcase 3: -1.23456789 / 3.14**



Instruction Counter



Instruction Statistics

Testcase 3: Time Executed ≈ 170 ns

6 Kiểm thử chương trình

6.1 Trường hợp xuất hiện Exception

- **Infinity**

```
Input 2 float numbers:
1.5
0
Infinity
```

```
Input 2 float numbers:
99999999999999999999999999999999999999999999999999999999
0.00000000000000000000000000000000000000000000000000000001
Infinity
```

Input không hợp lệ nên ném ra lỗi Infinity

- **NAN**

```
Input 2 float numbers:
0.00000000000000000000000000000000000000000000000001
0.00000000000000000000000000000000000000000000000002
Error: NAN
```

- **Overflow**

```
Input 2 float numbers:
99999999999999999999999999
0.00000000000000000000000001
Error: Overflow
```

Input hợp lệ nhưng kết quả thương bị Overflow

- **Underflow**

```
Input 2 float numbers:
0.00000000000000000000000000000001
99999999999999999999999999999999999
Error: Underflow
```

Input hợp lệ nhưng kết quả thương bị Underflow

6.2 Trường hợp hai số dương

```
Input 2 float numbers:
1.23456
2.34567
Result: 0.52631444
```

6.3 Trường hợp hai số âm

```
Input 2 float numbers:
-1.23456
-2.34567
Result: 0.52631444
```

6.4 Trường hợp một dương một âm

```
Input 2 float numbers:
-1.23456
2.34567
Result: -0.52631444
```

7 Kết luận

Số thực dấu chấm động (Floating Point) cũng như tốc độ tính toán với số thực dấu chấm động là một thước đo quan trọng nhằm đánh giá khả năng của một máy tính trong nhiều ứng dụng khác nhau. Thông qua bài tập lớn lần này, chúng ta được tiếp cận và hiểu sâu hơn về Floating Point, hay nói riêng là phép tính thương hai số thực dấu chấm động bằng cả cách thủ công thông thường và cách một kiến trúc máy tính lưu trữ và tính toán phép chia hai số thực dấu chấm động theo chuẩn IEEE 754 với độ chính xác đơn (IEEE 754 single-precision).

8 Tài liệu tham khảo

[1] Phạm Quốc Cường, *Giáo trình Kiến trúc máy tính*, 2017.

[2] Trần Thanh Bình, *Slide Kiến trúc máy tính - Chương 3 - ĐH Bách Khoa TPHCM*.

[3] David A. Patterson, John L. Hennessy, *Computer Organization and Design*, 5th Edition, 2014.

[4] Mr. Arnab Chakraborty, *Floating Point Arithmetic on Division, Youtobe, Link:* `https://youtu.be/B3Sggj1HmR4`

[5] Jacob Schrum, *Floating Point Arithmetic 3: Division, Youtobe, Link:* `https://en.wikipedia.org/wiki/K-d_tree`