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

KIẾN TRÚC MÁY TÍNH (CO2008)

Bài tập lớn
Kiến trúc tập lệnh
Đề 2

GVHD: NGUYỄN XUÂN MINH
SV thực hiện: TRẦN MINH DƯƠNG- 2310609
PHẠM CÔNG VÕ - 2313946
Lớp: L02
Nhóm: 21

Tp. Hồ Chí Minh, Tháng 5/2025

Lời Cảm Ơn

Đầu tiên, nhóm chúng em xin được gửi lời cảm ơn đến giảng viên là thầy Nguyễn Xuân Minh đã giúp nhóm thực hiện bài tập lớn này. Nhờ sự giúp đỡ tận tình của quý thầy, chúng em đã vượt qua những khúc mắc, khó khăn trong suốt quá trình thực hiện bài tập, từ đó hoàn thành đúng tiến độ của môn học và cho ra sản phẩm chất lượng. Ngoài ra, không thể không nhắc đến sự quan tâm giúp đỡ của các anh chị, các bạn sinh viên trong cộng đồng sinh viên trường Đại học Bách Khoa nói riêng và ĐHQG-HCM nói chung, những đóng góp tolớn của các anh, chị và các bạn đã giúp chúng em nắm chắc hơn cách sử dụng kiến trúc tập lệnh MIPS mà nhóm chỉ mới gần đây được tiếp cận trong quá trình theo học ở môi trường Đại học. Cuối cùng, nhóm chúng em xin gửi lời cảm ơn một lần nữa đến các tập thể, cá nhân đã giúp đỡ và truyền cảm hứng cho nhóm trong suốt quá trình thực hiện dự án bài tập lớn này.

Mục lục

Danh sách hình vẽ

1 Giới thiệu đề tài

1.1 Giới thiệu về MIPS

MIPS - Microprocessor without Interlocked Pipeline Stages - là kiến trúc bộ tập lệnh RISC phát triển bởi MIPS Technologies. Ban đầu kiến trúc MIPS là 32bit, và sau đó là phiên bản 64 bit. Nhiều sửa đổi của MIPS, bao gồm MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32 và MIPS64. Phiên bản hiện tại là MIPS32 và MIPS64.

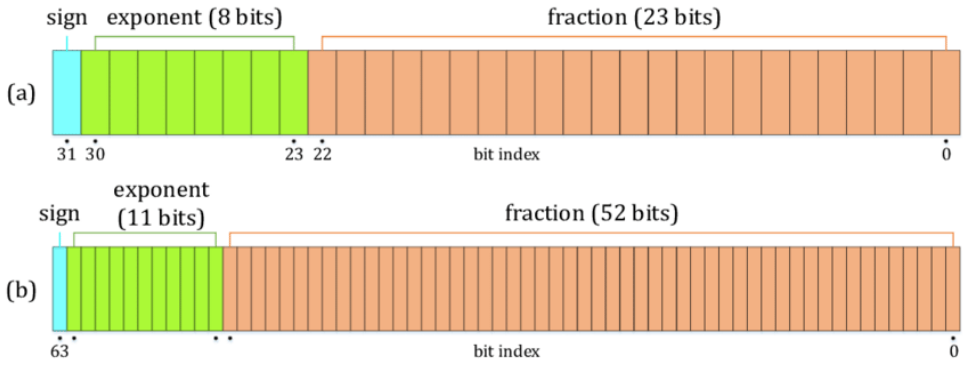
1.2 Đề bài 2:

Cộng 2 số thực chuẩn IEEE 754 chính xác đơn. Viết chương trình thực hiện phép cộng 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).

** Biểu diễn số thực trong máy tính theo chuẩn IEEE 754*

Máy tính chỉ có khả năng lưu trữ và xử lý các tín hiệu dưới dạng nhị phân. Do đó, để biểu diễn một số thực trong máy tính, trước tiên chúng ta cần chuyển đổi số đó sang dạng nhị phân. Tuy nhiên, không thể áp dụng cách biểu diễn như đối với số nguyên, mà cần tuân theo một chuẩn riêng biệt – phổ biến nhất là chuẩn **IEEE 754**. Chuẩn IEEE 754 quy định hai định dạng chính để biểu diễn số thực.

- **Độ chính xác đơn (single precision)**: sử dụng 32 bit, bao gồm 1 bit dấu, 8 bit cho phần mũ và 23 bit cho phần trị (phần lẻ).
- **Độ chính xác kép (double precision)**: sử dụng 64 bit, bao gồm 1 bit dấu, 11 bit cho phần mũ và 52 bit cho phần trị.



Hình 1: *IEEE Standard for Floating-Point Arithmetic (IEEE 754)*

Trước khi một số thực có thể được lưu trữ theo chuẩn IEEE 754, cần chuyển đổi số đó về **dạng chuẩn hóa**, tức là biểu diễn sao cho phần nguyên của số không chứa số 0 (ngoại trừ số 0 chính nó).

** Thống kê số lượng lệnh R, I, J trong chương trình MIPS*

Trong quá trình phát triển chương trình mô phỏng hợp ngữ MIPS, tổng số lệnh được sử dụng thuộc ba loại R, I và J được thống kê như sau:

Loại lệnh	Số lượng
R	20
I	36
J	4

Bảng 1: *Thống kê số lượng lệnh R, I, J sử dụng trong chương trình MIPS*

2 Giải thuật

2.1 Hướng tiếp cận tổng quát

Để thực hiện phép cộng (hoặc trừ) hai số thực biểu diễn theo chuẩn **IEEE 754**, ta cần tuần tự thực hiện các bước sau:

- *Bước 1: Chuẩn hóa các số*
  Chuyển hai số thực X và Y từ thanh ghi về dạng chuẩn hóa theo IEEE 754:
  $$X = 1.xxx\ldots x \times 2^{\text{exponent}_X - 127} \tag{1}$$
  $$Y = 1.yyy\ldots y \times 2^{\text{exponent}_Y - 127} \tag{2}$$
  Lưu ý: Dãy bit x hoặc y gồm 23 bit, tương ứng với phần **fraction (mantissa)** trong chuẩn IEEE 754.
- *Bước 2: Căn chỉnh số mũ*
  So sánh hai số mũ exponent_X và exponent_Y. Dịch phải phần mantissa của số có số mũ nhỏ hơn cho đến khi cả hai số có cùng số mũ (tức là cùng cơ số $2^{\text{exponent}-127}$).
- *Bước 3: Thực hiện phép cộng/trừ*
  Sau khi đã căn chỉnh số mũ, thực hiện phép cộng hoặc trừ trên phần mantissa của hai số.
- *Bước 4: Chuẩn hóa kết quả*
  Chuẩn hóa lại kết quả nếu cần thiết (ví dụ: nếu xảy ra tràn hoặc cần dịch trái để đưa về dạng chuẩn), sau đó mã hóa lại theo định dạng IEEE 754 để lưu trữ hoặc xuất kết quả.

2.2 Chi tiết giải thuật cộng/trừ số thực IEEE 754 sử dụng MIPS

2.2.1.Tách Các Trường Fraction, Exponent và Dấu (S) từ Thanh Ghi

Mục tiêu: Phân tách số thực 32-bit theo chuẩn IEEE 754 thành ba trường:

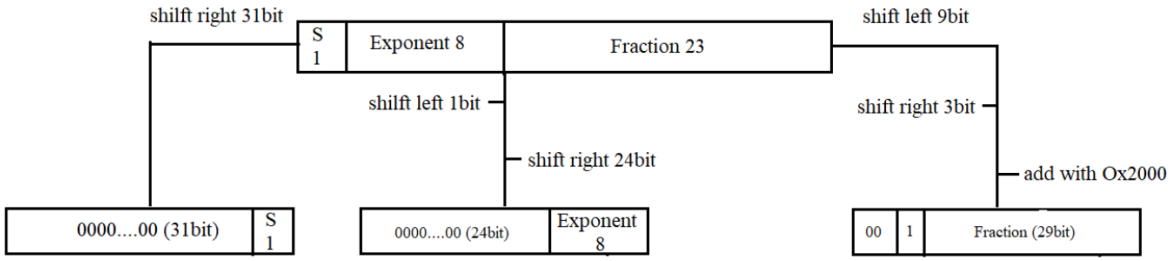
- **S (Sign)**: 1 bit (bit 31) – biểu thị dấu của số.
- **Exponent**: 8 bit (bit 30 đến 23) – mã hóa số mũ theo dạng bias (bias = 127).
- **Fraction (Mantissa)**: 23 bit (bit 22 đến 0) – phần trị không bao gồm bit ngầm định.

Quy trình thực hiện

1. **Tách các thành phần số thực từ thanh ghi:** Sau khi tải dữ liệu số thực vào thanh ghi, ta sử dụng các thao tác dịch bit để tách các trường thông tin. Các lệnh dịch bit như `Shift Left Logical (SLL)` và `Shift Right Logical (SRL)` sẽ được sử dụng để phân tách trường dấu, exponent, và fraction từ giá trị trong thanh ghi:
   - **Trường dấu** S**:** Dịch phải 31 bit để lấy giá trị của bit dấu S, xác định giá trị âm hoặc dương của số thực.

- **Exponent:** Tách phần exponent (8 bit) bằng cách dịch bit từ vị trí 23 đến 30.
- **Fraction (Mantissa):** Phần fraction được lấy từ các bit từ 0 đến 22, sau khi thực hiện dịch bit để tách chính xác phần trị của số thực.

2. **Chuẩn hóa fraction theo chuẩn IEEE 754:** Sau khi tách các thành phần, ta chuẩn hóa fraction theo chuẩn IEEE 754, bảo đảm không xảy ra hiện tượng tràn số trong các phép toán và tính toán chính xác phần trị của số thực.



Hình 2: *Mô tả cách lấy thông tin của một số thực trong thanh ghi*

Ghi chú: 2 bit phụ ở đầu fraction giúp tránh tràn số khi tính toán và hỗ trợ xét dấu sau phép cộng/trừ.

2.2.2. Thực hiện phép Cộng/Trừ Hai Thanh Ghi Fraction

Mục tiêu: Thực hiện phép cộng hoặc trừ giữa hai số thực đã tách và chuẩn hóa, đồng thời xử lý các trường hợp liên quan đến dấu và exponent.

Quy trình thực hiện

1. **So sánh Exponent:** Trước khi thực hiện phép cộng hoặc trừ, ta so sánh exponent của hai số thực. Số có exponent nhỏ hơn sẽ được dịch fraction sang phải để căn chỉnh với exponent của số có exponent lớn hơn. Việc dịch bit này sẽ đảm bảo phép toán chính xác trong hệ thống số thực.
2. **Xử lý Dấu:** Do phần fraction luôn là số dương, ta cần kiểm tra dấu của các thanh ghi. Nếu bit dấu $S = 1$, phần fraction của số đó sẽ được đổi dấu (chuyển sang dạng bù 2). Sau khi xử lý dấu, ta có thể tiếp tục thực hiện phép toán cộng hoặc trừ.
3. **Cộng hoặc Trừ Fraction:** Sau khi đã xử lý dấu và căn chỉnh exponent, ta thực hiện phép cộng hoặc trừ giữa hai phần fraction. Phép cộng được thực hiện như bình thường, còn nếu là phép trừ, ta cần điều chỉnh dấu của một trong các phần fraction.

2.2.3. Chuẩn Hóa Kết Quả về Dạng Chuẩn IEEE 754

Mục tiêu: Đưa kết quả về dạng chuẩn `1.xxxxxx` như định dạng IEEE 754 yêu cầu.

1. Xét dấu kết quả:
   - Nếu âm, đặt `S = 1` và lấy trị tuyệt đối.
   - Nếu dương, đặt `S = 0`.
2. So sánh với ngưỡng chuẩn IEEE 754:
   Để chuẩn hóa kết quả, ta so sánh giá trị với hai ngưỡng chuẩn $0x40000000$ và $0x20000000$:

- Nếu kết quả lớn hơn hoặc bằng $0x40000000$, ta thực hiện dịch phải (shift right) để giảm giá trị, đồng thời tăng exponent.
- Nếu kết quả nhỏ hơn $0x20000000$, ta thực hiện dịch trái (shift left) để tăng giá trị, đồng thời giảm exponent.

3. Tiếp tục cho đến khi phần trị nằm trong đoạn $[0x20000000, 0x40000000)$.
4. Chuyển đổi Fraction:
   Sau khi chuẩn hóa, phần fraction sẽ có dạng $1.xxx...xx$ với 30 bit. Điều này đảm bảo rằng phần trị của số thực đã được chuẩn hóa theo chuẩn IEEE 754.

2.2.4. Tổ hợp kết quả thành số thực IEEE 754

Mục tiêu: Tổ hợp các thành phần (dấu S, exponent và fraction) để tạo ra kết quả cuối cùng dưới dạng số thực chuẩn IEEE 754.

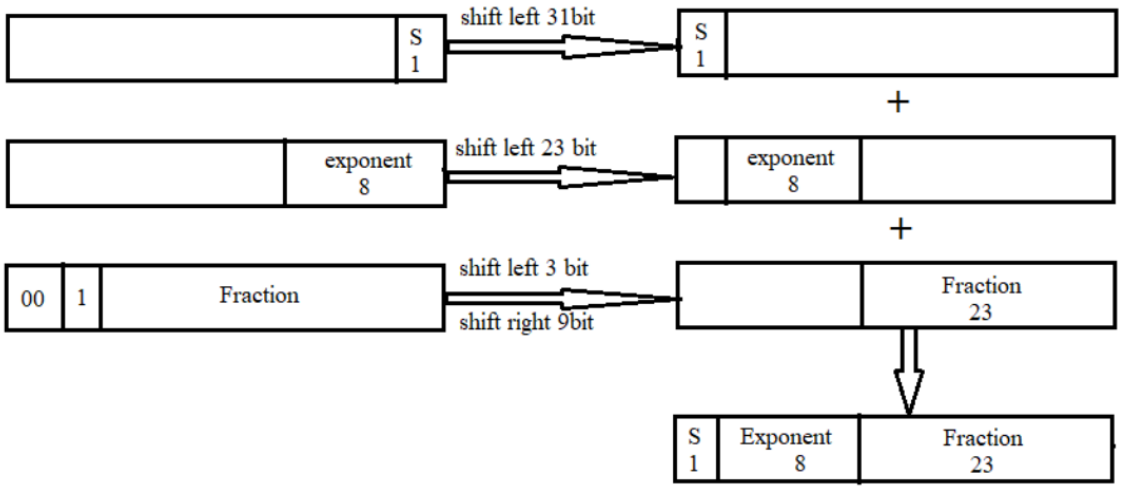
Quy trình thực hiện:

1. Dời các phần vào đúng vị trí:
   - `Dịch phần dấu` S `lên bit 31: S << 31`
   - `Dịch exponent vào các bit từ 23 đến 30: Exponent << 23`
   - `Dịch fraction vào các bit từ 0 đến 22.`

   Quá trình này giúp tái tạo lại số thực theo định dạng chuẩn IEEE 754.
2. Tổng hợp kết quả: Cuối cùng, kết quả sẽ được lưu vào thanh ghi, sẵn sàng cho các phép toán tiếp theo hoặc xuất ra bộ nhớ.

```
result = (S << 31) | (Exponent << 23) | (Fraction & 0x7FFFFF)
```

Hãy xem hình minh họa dưới đây:



Hình 3: *Mô tả cách lấy thông tin của một số thực trong thanh ghi*

Tổng kết quy trình

Giai đoạn	Mô tả
2.1	Tách các trường `S`, `Exponent`, `Fraction` từ số thực 32-bit.
2.2	So sánh số mũ, dịch fraction phù hợp, xử lý dấu và thực hiện phép cộng/trừ.
2.3	Chuẩn hóa kết quả về dạng chuẩn IEEE 754 bằng dịch bit và điều chỉnh exponent.
2.4	Tổ hợp các thành phần lại để tạo số thực IEEE 754 chuẩn 32-bit.

3 Chạy code với testcase

3.1 Tính thời gian chạy của chương trình

Thông số hệ thống và giả định:

- **Tần số xung nhịp (Clock Rate)**: 1 GHz ($= 10^9$ chu kỳ/giây)
- **Chu kỳ mỗi lệnh (CPI)**: 1
  (Giả định: mọi lệnh đều có cùng chi phí thực thi)

Dựa vào các giả định trên, thời gian thực thi của chương trình được tính theo công thức:

$$\text{CPU Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

Vì CPI = 1, nên công thức rút gọn còn:

$$\text{CPU Time} = \frac{\text{Instruction Count}}{10^9} \quad \text{(giây)}$$

Đối với mỗi trường hợp kiểm thử, cần thực hiện:

1. Thống kê số lượng và loại lệnh sử dụng (Instruction Types).
2. Tính toán thời gian thực thi dựa trên tổng số lệnh.
3. Trình bày kết quả kiểm thử gồm:
   - Mô tả chức năng kiểm thử.
   - Bảng thống kê loại lệnh và số lượng.
   - Thời gian thực thi tính được.

Nhận xét: Qua việc phân tích số lượng lệnh và tính toán thời gian thực thi, có thể thấy rằng hiệu năng chương trình phụ thuộc trực tiếp vào số lượng lệnh máy sinh ra. Với giả định CPI = 1 và tần số đồng hồ cố định, việc tối ưu mã nguồn để giảm tổng số lệnh sẽ giúp cải thiện đáng kể thời gian thực thi. Điều này cho thấy tầm quan trọng của việc lựa chọn thuật toán và cấu trúc dữ liệu phù hợp trong quá trình phát triển phần mềm.

Ta đánh giá hiệu quả của chương trình thông qua một số test cases sau đây:

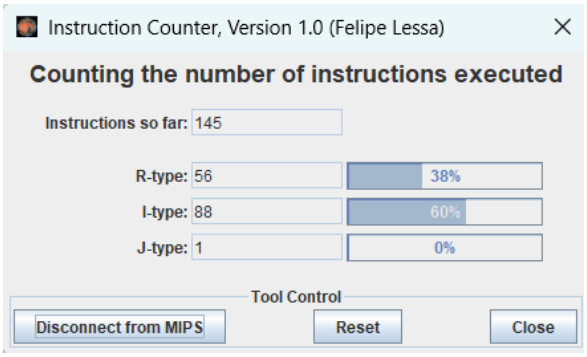
a	b	a+b	MIPS	R	I	J	IC	Time (ns)
3.14	4.13	7.27	7.27	56	88	1	145	145
3.57	2.86	6.43	6.43	47	89	1	137	137
3.57	-9.23	-5.66	-5.66	58	90	3	151	151
1.0	2.0	3.0	3.0	56	88	1	145	145
5.38	-5.38	0.0	0.0	41	82	2	125	125
5.38	-10.39	-5.01	-5.01	58	90	3	151	151
-321.59	555.55	233.96	233.96002	60	94	4	158	158
-55.36	123.32	67.96	67.96	56	86	2	144	144
-578.3612	12345.699	11767.3378	11767.339	56	86	2	144	144
8.12365	9.2587	17.38235	17.382349	47	89	1	137	137
25.032005	9.4558	34.487805	34.487804	47	89	1	137	137
-46.8129	1427.734	1380.9211	1380.9211	56	86	2	144	144
-1001.48145	-1125.919	-2127.40045	-2127.4004	57	89	1	147	147
-280.3314	-11.32654	-291.65794	-291.65793	46	88	1	135	135
92656.9	-20.369747	92636.53025	92636.53	46	86	2	134	134
3654157.0	-3651145.8	3011.2	3011.25	66	126	12	204	204
34.232624	-65.21548	-30.982856	-30.982857	60	94	4	158	158
321.32147	-1236.5479	-915.22643	-915.22644	58	90	3	151	151
680.77045	-206.09088	474.67957	474.67957	48	90	3	141	141
-538.9438	-746.4934	-1285.4372	-1285.4371	47	89	1	137	137
-312.01453	-1594.2965	-1906.31103	-1906.311	56	88	1	145	145
-245.30302	977.04926	731.74624	731.7463	56	86	2	144	144
616.9262	55.415	672.3412	672.3412	46	88	1	135	135
46.985474	-1309.58	-1262.594526	-1262.5945	56	86	2	144	144
-731.8893	-738.224	-1470.1133	-1470.1133	47	89	1	137	137

Bảng 2: *Bảng thống kê kết quả kiểm thử*

3.2 Test case 0:

```
a = 3.14;
b = 4.13;
a + b = 7.27
```

```
FLOAT 1 = 3.14
FLOAT 2 = 4.13
TONG 2 SO: 3.14 + 4.13 = 7.27
```



Hình 4: *Các kết quả kiểm thử cho Testcase 0*

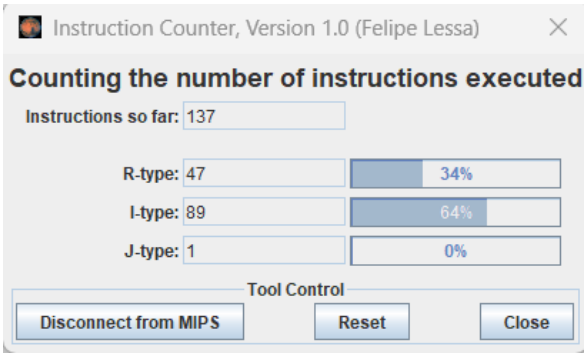
$$\text{CPU time} = \frac{145}{10^9} = 145ns.$$

3.3 Test case 1:

```
a = 3.57;
b = 2.86;
a + b = 6.43
```

$$\text{CPU time} = \frac{137}{10^9} = 137ns.$$

```
FLOAT 1 = 3.57
FLOAT 2 = 2.86
TONG 2 SO: 3.57 + 2.86 = 6.43
```



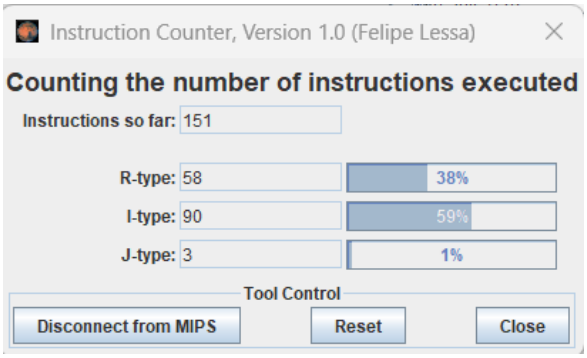
Hình 5: *Các kết quả kiểm thử cho Testcase 1*

3.4 Test case 2:

```
a = 3.57;
b = -9.23;
a + b = -5.66
```

$$\text{CPU time} = \frac{151}{10^9} = 151ns.$$

```
FLOAT 1 = 3.57
FLOAT 2 = -9.23
TONG 2 SO: 3.57 + -9.23 = -5.66
```



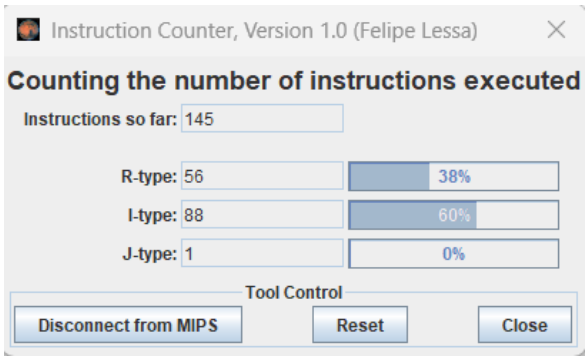
Hình 6: *Các kết quả kiểm thử cho Testcase 2*

3.5 Test case 3:

```
a = 1,0;
b = 2.0;
a + b = 3.0
```

$$\text{CPU time} = \frac{145}{10^9} = 145ns.$$

```
FLOAT 1 = 1.0
FLOAT 2 = 2.0
TONG 2 SO: 1.0 + 2.0 = 3.0
```



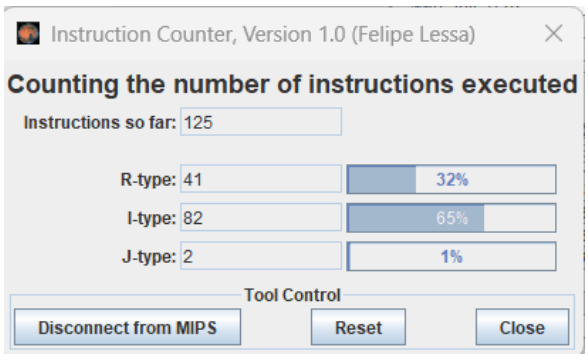
Hình 7: *Các kết quả kiểm thử cho Testcase 3*

3.6 Test case 4:

```
a = 5.38;
b = -5.38;
a + b = 0.0
```

$$\text{CPU time} = \frac{125}{10^9} = 125ns.$$

```
FLOAT 1 = 5.38
FLOAT 2 = -5.38
TONG 2 SO: 5.38 + -5.38 = 0.0
```



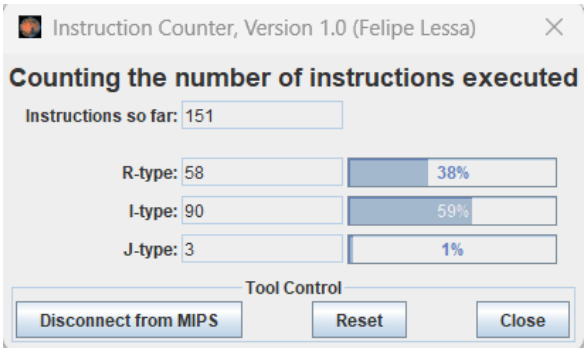
Hình 8: *Các kết quả kiểm thử cho Testcase 4*

3.7 Test case 5:

```
a = 5.38;
b = -10.39;
a + b = -5.01
```

$$\text{CPU time} = \frac{151}{10^9} = 151ns.$$

```
FLOAT 1 = 5.38
FLOAT 2 = -10.39
TONG 2 SO: 5.38 + -10.39 = -5.01
```



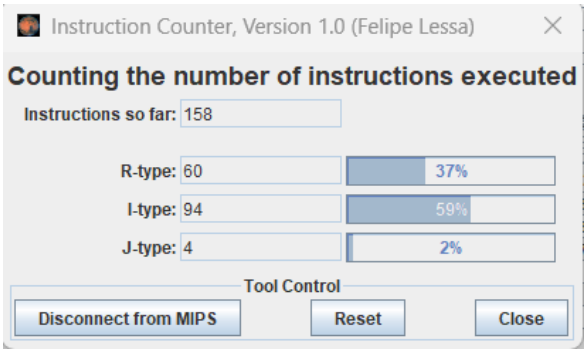
Hình 9: *Các kết quả kiểm thử cho Testcase 5*

3.8 Test case 6:

```
a = -321.59;
b = 555.55;
a + b = 233.96
```

$$\text{CPU time} = \frac{158}{10^9} = 158ns.$$

```
FLOAT 1 = -321.59
FLOAT 2 = 555.55
TONG 2 SO: -321.59 + 555.55 = 233.96002
```



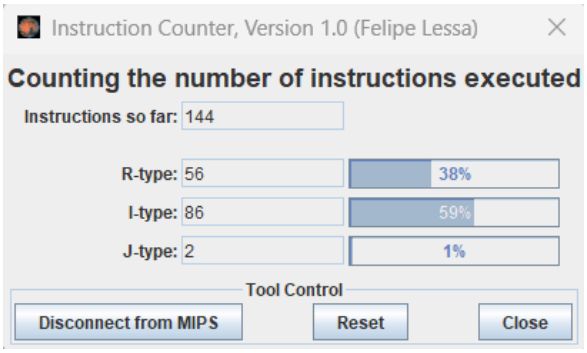
Hình 10: *Các kết quả kiểm thử cho Testcase 6*

3.9 Test case 7:

```
a = -55.36;
b = 123.32;
a + b = 67.96
```

$$\text{CPU time} = \frac{144}{10^9} = 144ns.$$

```
FLOAT 1 = -55.36
FLOAT 2 = 123.32
TONG 2 SO: -55.36 + 123.32 = 67.96
```



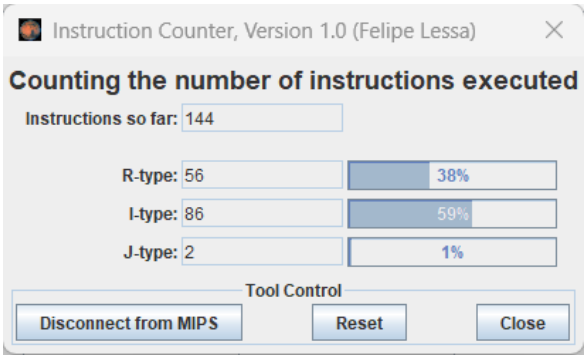
Hình 11: *Các kết quả kiểm thử cho Testcase 7*

3.10 Test case 8:

```
a = -578.3612;
b = 12345.699;
a + b = 11767.3378
```

$$\text{CPU time} = \frac{144}{10^9} = 144ns.$$

```
FLOAT 1 = -578.3612
FLOAT 2 = 12345.699
TONG 2 SO: -578.3612 + 12345.699 = 11767.339
```

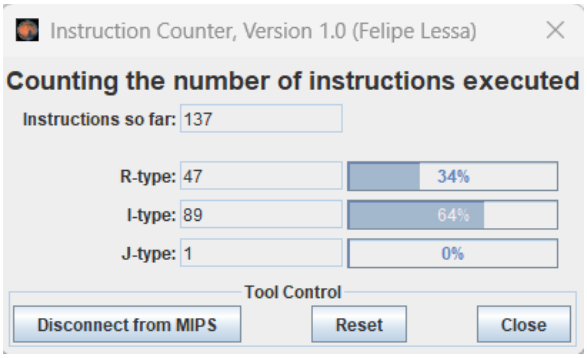


Hình 12: *Các kết quả kiểm thử cho Testcase 8*

3.11 Test case 9:

```
a = 8.12365;
b = 9.2587;
a + b = 17.38235
```

```
FLOAT 1 = 8.12365
FLOAT 2 = 9.2587
TONG 2 SO: 8.12365 + 9.2587 = 17.382349
```



Hình 13: *Các kết quả kiểm thử cho Testcase 9*

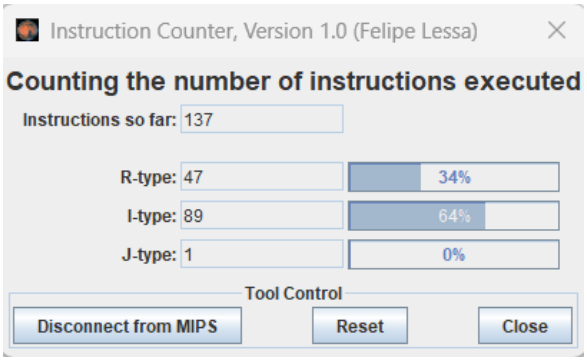
$$\text{CPU time} = \frac{137}{10^9} = 137ns.$$

3.12 Test case 10:

```
a = 25.032005;
b = 9.4558;
a + b = 34.487805
```

$$\text{CPU time} = \frac{137}{10^9} = 137ns.$$

```
FLOAT 1 = 25.032005
FLOAT 2 = 9.4558
TONG 2 SO: 25.032005 + 9.4558 = 34.487804
```



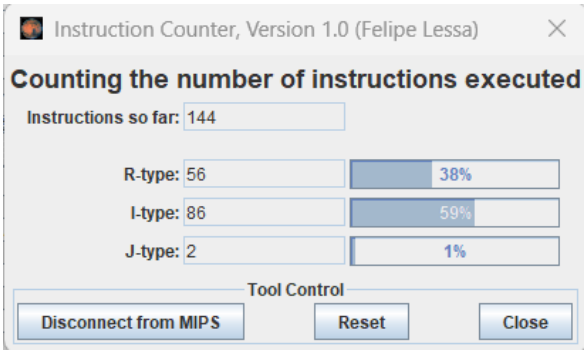
Hình 14: *Các kết quả kiểm thử cho Testcase 10*

3.13 Test case 11:

```
a = -46.8129;
b = 1427.734;
a + b = 1380.9211
```

$$\text{CPU time} = \frac{144}{10^9} = 144ns.$$

```
FLOAT 1 = -46.8129
FLOAT 2 = 1427.734
TONG 2 SO: -46.8129 + 1427.734 = 1380.9211
```



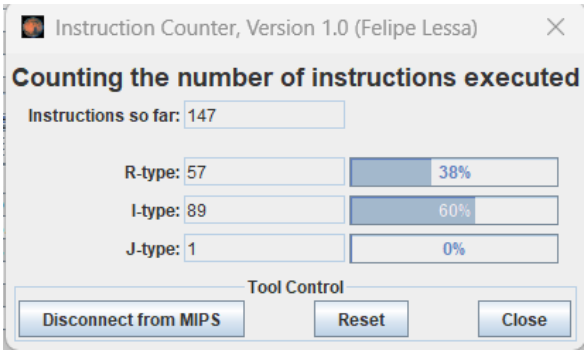
Hình 15: *Các kết quả kiểm thử cho Testcase 11*

3.14 Test case 12:

```
a = -1001.48145 ;
b = -1125.919;
a + b = -2127.40045
```

$$\text{CPU time} = \frac{147}{10^9} = 147ns.$$

```
FLOAT 1 = -1001.48145
FLOAT 2 = -1125.919
TONG 2 SO: -1001.48145 + -1125.919 = -2127.4004
```

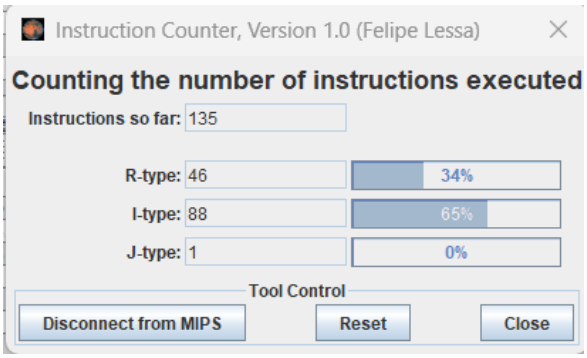


Hình 16: *Các kết quả kiểm thử cho Testcase 12*

3.15 Test case 13:

```
a = -280.3314;
b = -11.32654;
a + b = -291.65794
```

```
FLOAT 1 = -280.3314
FLOAT 2 = -11.32654
TONG 2 SO: -280.3314 + -11.32654 = -291.65793
```



Hình 17: *Các kết quả kiểm thử cho Testcase 13*

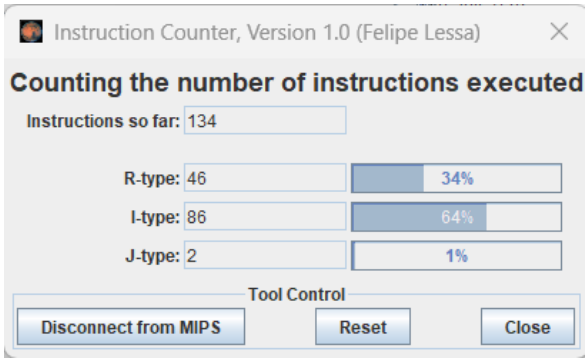
$$\text{CPU time} = \frac{135}{10^9} = 135ns.$$

3.16 Test case 14:

```
a = 92656.9 ;
b = -20.369747;
a + b = 92636.53025
```

$$\text{CPU time} = \frac{134}{10^9} = 134ns.$$

```
FLOAT 1 = 92656.9
FLOAT 2 = -20.369747
TONG 2 SO: 92656.9 + -20.369747 = 92636.53
```



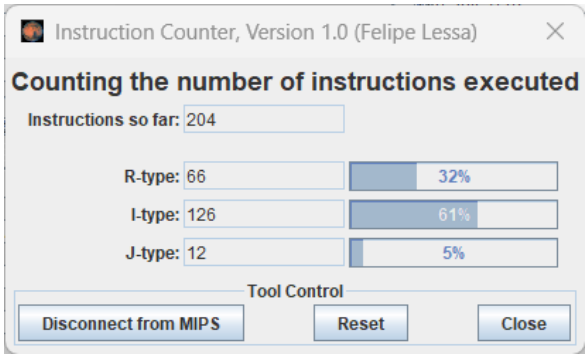
Hình 18: *Các kết quả kiểm thử cho Testcase 14*

3.17 Test case 15:

```
a = 3654157.0;
b = -3651145.8;
a + b = 3011.2
```

$$\text{CPU time} = \frac{204}{10^9} = 204ns.$$

```
FLOAT 1 = 3654157.0
FLOAT 2 = -3651145.8
TONG 2 SO: 3654157.0 + -3651145.8 = 3011.25
```



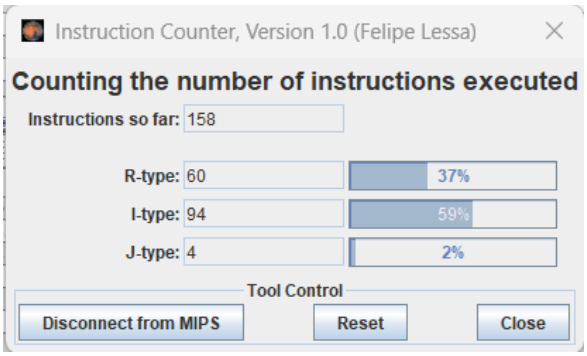
Hình 19: *Các kết quả kiểm thử cho Testcase 15*

3.18 Test case 16:

```
a = 34.232624;
b = -65.21548;
a + b = -30.982856
```

$$\text{CPU time} = \frac{158}{10^9} = 158ns.$$

```
FLOAT 1 = 34.232624
FLOAT 2 = -65.21548
TONG 2 SO: 34.232624 + -65.21548 = -30.982857
```



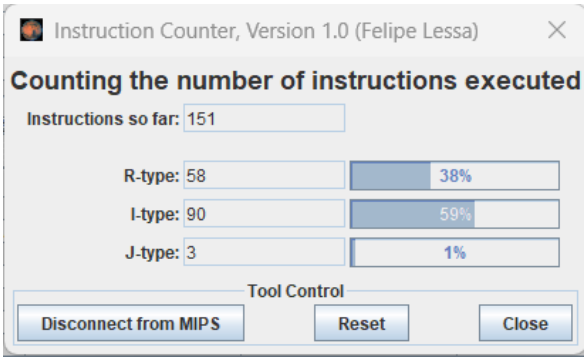
Hình 20: *Các kết quả kiểm thử cho Testcase 16*

3.19 Test case 17:

```
a = 321.32147;
b = -1236.5479;
a + b = -915.22643
```

$$\text{CPU time} = \frac{151}{10^9} = 151ns.$$

```
FLOAT 1 = 321.32147
FLOAT 2 = -1236.5479
TONG 2 SO: 321.32147 + -1236.5479 = -915.22644
```



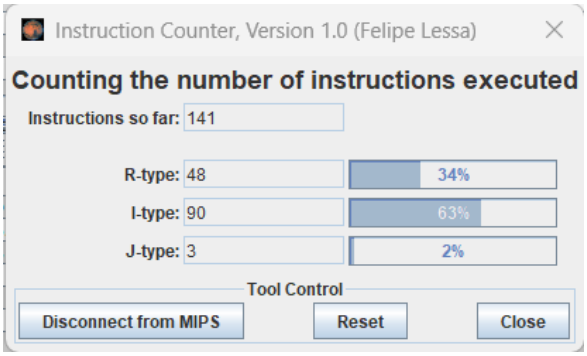
Hình 21: *Các kết quả kiểm thử cho Testcase 17*

3.20 Test case 18:

```
a = 680.77045;
b = -206.09088;
a + b = 474.67957
```

$$\text{CPU time} = \frac{141}{10^9} = 141ns.$$

```
FLOAT 1 = 680.77045
FLOAT 2 = -206.09088
TONG 2 SO: 680.77045 + -206.09088 = 474.67957
```



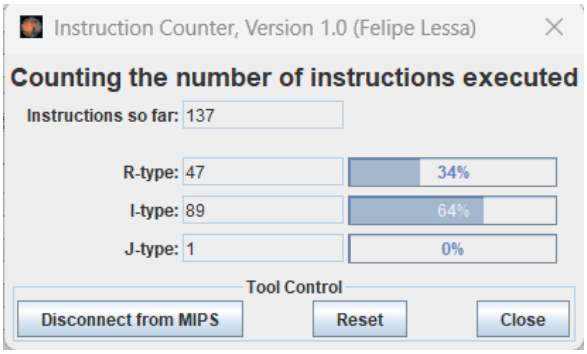
Hình 22: *Các kết quả kiểm thử cho Testcase 18*

3.21 Test case 19:

```
a = -538.9438;
b = -746.4934;
a + b = -1285.4372
```

$$\text{CPU time} = \frac{137}{10^9} = 137ns.$$

```
FLOAT 1 = -538.9438
FLOAT 2 = -746.4934
TONG 2 SO: -538.9438 + -746.4934 = -1285.4371
```

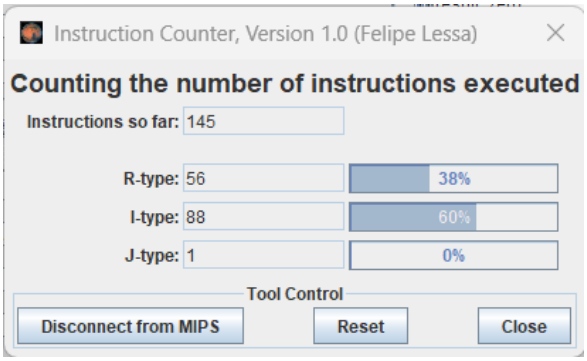


Hình 23: *Các kết quả kiểm thử cho Testcase 19*

3.22 Test case 20:

```
a = -312.01453;
b = -1594.2965;
a + b = -1906.31103
```

```
FLOAT 1 = -312.01453
FLOAT 2 = -1594.2965
TONG 2 SO: -312.01453 + -1594.2965 = -1906.311
```



Hình 24: *Các kết quả kiểm thử cho Testcase 20*

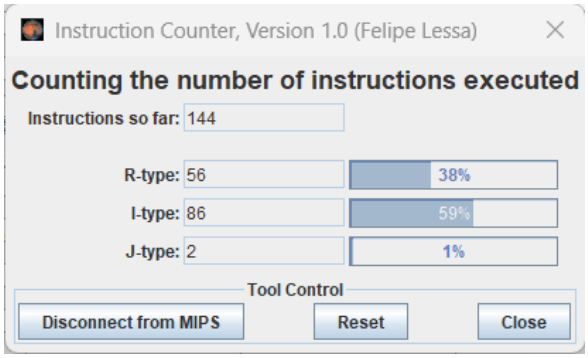
$$\text{CPU time} = \frac{145}{10^9} = 145ns.$$

3.23 Test case 21:

```
a = -245.30302;
b = 977.04926;
a + b = 731.74624
```

$$\text{CPU time} = \frac{144}{10^9} = 144ns.$$

```
FLOAT 1 = -245.30302
FLOAT 2 = 977.04926
TONG 2 SO: -245.30302 + 977.04926 = 731.7463
```



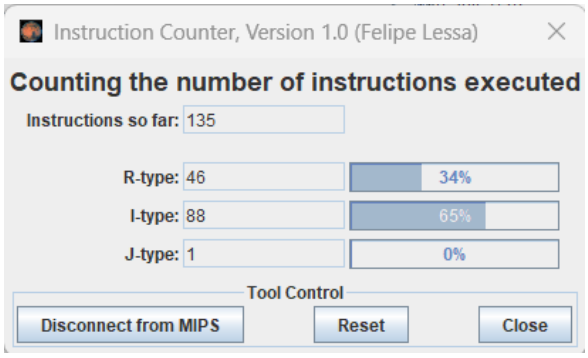
Hình 25: *Các kết quả kiểm thử cho Testcase 21*

3.24 Test case 22:

```
a = 616.9262;
b = 55.415;
a + b = 672.3412
```

$$\text{CPU time} = \frac{135}{10^9} = 135ns.$$

```
FLOAT 1 = 616.9262
FLOAT 2 = 55.415
TONG 2 SO: 616.9262 + 55.415 = 672.3412
```



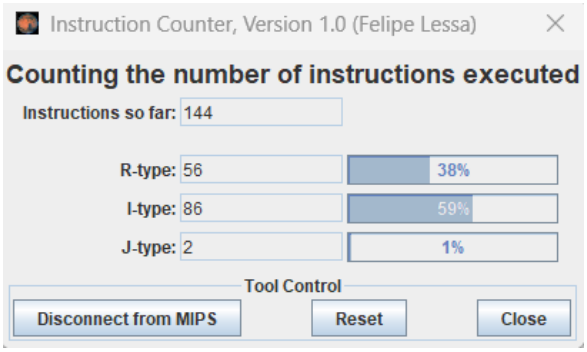
Hình 26: *Các kết quả kiểm thử cho Testcase 22*

3.25 Test case 23:

```
a = 46.985474;
b = -1309.58;
a + b = -1262.594526
```

$$\text{CPU time} = \frac{144}{10^9} = 144ns.$$

```
FLOAT 1 = 46.985474
FLOAT 2 = -1309.58
TONG 2 SO: 46.985474 + -1309.58 = -1262.5945
```



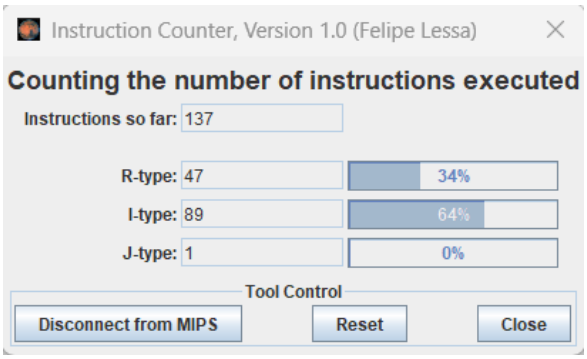
Hình 27: *Các kết quả kiểm thử cho Testcase 23*

3.26 Test case 24:

```
a = -731.8893;
b = -738.224;
a + b = -1470.1133
```

$$\text{CPU time} = \frac{137}{10^9} = 137ns.$$

```
FLOAT 1 = -731.8893
FLOAT 2 = -738.224
TONG 2 SO: -731.8893 + -738.224 = -1470.1133
```



Hình 28: *Các kết quả kiểm thử cho Testcase 24*

Tài liệu

[1] Phạm Quốc Cường. *Kiến trúc máy tính* (3st Edition, 2019), Companion Website: https://lib.hcmut.edu.vn/kien-truc-may-tinh