



BÁO CÁO BÀI TẬP LỚN KIẾN TRÚC MÁY TÍNH



Đề tài 2: **SỬ DỤNG HỢP NGỮ ASSEMBLY MIPS CỘNG TRỪ HAI SỐ THỰC**

GVHD: Trần Thanh Bình
Võ Tấn Phương

Thực hiện: Nhóm
Nguyễn Anh Kiệt - 1911456
Nguyễn Trung Phong - 1911841
Nguyễn Kim Lộc - 1911530

Mục lục

1	Lý thuyết	5
1.1	Số thực dạng chuẩn IEEE 754 dạng chính xác đơn	5
1.2	Giải thuật cộng, trừ 2 số thực	6
1.3	Thống kê số lệnh, loại lệnh chương trình của nhóm	8
1.3.1	Lệnh R-type	8
1.3.2	Lệnh I-type	8
1.3.3	Lệnh J-type	8
2	Hiện thực cộng trừ hai số thực	9
2.1	Hiện thực code MIPS	9
2.2	Kết quả các testcase	17
2.2.1	Testcase 1	17
2.2.2	Testcase 2	17
2.2.3	Testcase 3	18
2.2.4	Testcase 4	18
2.2.5	Testcase 5	18
2.2.6	Testcase 6	19
2.2.7	Testcase 7	19
2.2.8	Testcase 8	19
2.2.9	Testcase 9	20
2.2.10	Testcase 10	20
2.2.11	Testcase 11	20
2.2.12	Testcase 12	21
3	Thống kê lệnh và tính toán thời gian	22
3.0.1	Testcase 1: $1587.06 + 66.170$	22
3.0.2	Testcase 2: $-652.32 + -6769.658$	23
3.0.3	Testcase 3: $0 + -982.658$	24
3.0.4	Testcase 4: $6561.56 - 8484.556$	25
3.0.5	Testcase 5: $650.25 - -3261.075$	26
3.0.6	Testcase 6: $0 - -135.068$	27
3.0.7	Testcase 7: $-123.35 + 23.235$	28

3.0.8	Testcase 8: $1021.2376 + -565.575$	29
3.0.9	Testcase 9: $415.7232 + 0$	30
3.0.10	Testcase 10: $-12.343 - -2.346$	31
3.0.11	Testcase 11: $-0.325 - 1.851$	32
3.0.12	Testcase 12: $-654.1329 - 0$	33



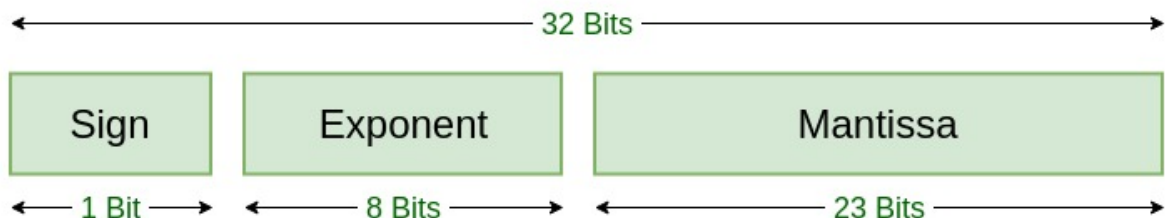
Danh sách các hình

1.1.1	Single Precision IEEE 754 Floating-Point Standard.	5
2.2.1	Testcase 1	17
2.2.2	Testcase 2	17
2.2.3	Testcase 3	18
2.2.4	Testcase 4	18
2.2.5	Testcase 5	19
2.2.6	Testcase 6	19
2.2.7	Testcase 7	19
2.2.8	Testcase 8	20
2.2.9	Testcase 9	20
2.2.10	Testcase 10	20
2.2.11	Testcase 11	21
2.2.12	Testcase 12	21
3.0.1	Thống kê lệnh testcase 1	22
3.0.2	Thống kê lệnh testcase 2	23
3.0.3	Thống kê lệnh testcase 3	24
3.0.4	Thống kê lệnh testcase 4	25
3.0.5	Thống kê lệnh testcase 5	26
3.0.6	Thống kê lệnh testcase 6	27
3.0.7	Thống kê lệnh testcase 7	28
3.0.8	Thống kê lệnh testcase 8	29
3.0.9	Thống kê lệnh testcase 9	30
3.0.10	Thống kê lệnh testcase 10	31
3.0.11	Thống kê lệnh testcase 11	32
3.0.12	Thống kê lệnh testcase 12	33

Phần 1 Lý thuyết

1.1 Số thực dạng chuẩn IEEE 754 dạng chính xác đơn

- Số thực được lưu trong máy tính dưới dạng 32 bit gồm:
 - + 1 bit dấu (*sign*).
 - + 8 bit xác định mũ (*exponent*).
 - + 23 bit phần định trị (*mantissa*).
- Bit dấu (*sign*): 0 biểu thị số dương, 1 biểu thị số âm.
- Bit xác định số mũ (*exponent*): 8 bits xác định giá trị mũ (số mũ + 127).
- Bit định trị (*mantissa*): số thực sẽ chuyển về dạng 1.m với m sẽ được chuyển sang nhị phân và lưu bằng 23 bits.



Single Precision IEEE 754 Floating-Point Standard

Hình 1.1.1: Single Precision IEEE 754 Floating-Point Standard.

1.2 Giải thuật cộng, trừ 2 số thực

1) Tách phần bit dấu của từng số:

- Lưu số vào 1 thanh ghi.
- and với $0x80000000$ để lấy bit đầu tiên.
- Các bit còn lại cho bằng 0.

2) Tách phần xác định mũ:

- Lưu số vào 1 thanh ghi.
- and với $0x7F800000$ để lấy 8 bits exponent.

3) Tách phần định trị:

- Lưu số vào 1 thanh ghi.
- and với $0x007FFFFFFF$ để lấy 23 bits cuối.
- or với $0x00800000$ để đặt bit 1 đứng trước 23bits cuối (bit thứ 24 từ cuối lên) để tượng trưng cho 1. của phần định trị ta gọi bit này là bit prefix.

4) Kiểm tra số mũ:

- Kiểm tra số mũ của hai số có bằng nhau không.
- Nếu không bằng nhau thì ta tăng mũ của số có mũ bé hơn, cứ tăng mũ của số nào lên 1 đơn vị thì phần định trị của số đó ta dịch sang phải 1 bit tương ứng. Tăng cho đến khi hai số có số mũ bằng nhau.
- Lấy số mũ đó là số mũ của kết quả.

5) Kiểm tra là phép cộng hay trừ, nếu là phép trừ, đảo bit dấu của số thứ hai bằng cách addu với $0x80000000$ rồi từ đây thực hiện phép cộng như thường.

6) Kiểm tra hai số có cùng dấu không: ta kiểm tra hai bit dấu của hai số có bằng nhau không:

- Nếu bằng:
 - + Cộng phần định trị của hai số rồi lưu vào thanh ghi.
 - + Lấy bit dấu của hai số là bit dấu của kết quả.
- Nếu không bằng (trái dấu):
 - + Kiểm tra xem số nào có phần định trị lớn hơn thì ta lấy phần định trị của số đó trừ phần định trị của số còn lại rồi lưu vào thanh ghi.
 - + Lấy bit dấu của kết quả là bit dấu của số có phần định trị lớn hơn. (Nếu phần định trị của hai số bằng nhau thì ta sẽ xuất kết quả bằng 0 luôn)

- 7) Điều chỉnh lại số mũ: sau khi thực hiện phép cộng ở trên thì ta kiểm tra lại thanh ghi lưu kết quả phần định trị:
- Nếu hai số cùng dấu:
 - + Kiểm tra bit đứng trước bit prefix có bằng 1 không.
 - + Nếu bằng 1 thì ta dịch phần định trị của kết quả sang phải 1 bit và tăng số mũ của kết quả 1 đơn vị.
 - Nếu hai số trái dấu:
 - + Kiểm tra bit prefix có bằng 0 hay không.
 - + Nếu bằng 0 thì dịch trái các bit cho đến khi bit tại vị trí prefix (bit thứ 24 từ cuối lên) bằng 1 thì ngừng và giảm số mũ đi một lượng bằng với số bit đã dịch.
- 8) Xuất kết quả: đặt lại bit prefix của thanh ghi lưu phần định trị của kết quả bằng 0 sau đó or với các thanh ghi lưu bit dấu và lưu phần xác định mũ của kết quả để thu được kết quả cuối cùng.

1.3 Thống kê số lệnh, loại lệnh chương trình của nhóm

1.3.1 Lệnh R-type

STT	TÊN LỆNH	CHỨC NĂNG
1	mfc1 \$t1, \$f0	Gán giá trị thực từ thanh ghi \$f0 vào \$t1
2	add \$t0, \$v0, \$0	Tính tổng rồi lưu vào \$t0
3	and \$s2, \$t2, \$at	Tính phép “and” rồi nạp vào \$s2
4	addu \$s2, \$s2, \$at	Cộng không âm 2 thanh ghi rồi nạp vào \$s2
5	or \$s4, \$s4, \$at	Thực hiện phép “or” 2 thanh ghi
6	slt \$at, \$s5, \$s3	Kiểm tra điều kiện ($s5 < s3$)
7	subu \$s4, \$s4, \$s6	Phép trừ số không âm 2 thanh ghi rồi lưu vào \$s4
8	sub \$s3, \$s3, \$at	Lệnh trừ 2 thanh ghi rồi lưu vào \$s3
9	mtc1 \$t1, \$f12	Gán giá trị từ thanh ghi \$t1 vào \$f12

1.3.2 Lệnh I-type

STT	TÊN LỆNH	CHỨC NĂNG
1	la \$a0, sothunhat	Gán địa chỉ label sothunhat vào thanh ghi \$a0
2	li \$v0, 4	Nạp vào thanh ghi \$v0 giá trị 4
3	addiu \$t4, \$t1, 0	Cộng hằng số không âm
4	lui \$at, 0x8000	Nạp vào 16 bit cao thanh ghi \$at giá trị 0x8000
5	ori \$at, \$at, 0x0000	Nạp vào 16 bit thấp thanh ghi \$at giá trị 0x0000
6	srl \$s4, \$s4, 1	Dịch thanh ghi \$s4 sang phải 1 bit
7	sll \$s4, \$s4, 1	Dịch thanh ghi \$s4 sang trái 1 bit
8	beq \$s1, \$s2, adding	Lệnh rẽ nhánh
9	bne \$t0, \$0, subres	Lệnh rẽ nhánh

1.3.3 Lệnh J-type

STT	TÊN LỆNH	CHỨC NĂNG
1	j addres	Nhảy không điều kiện

Phần 2

Hiện thực cộng trừ hai số thực

2.1 Hiện thực code MIPS

Code segment

```
.text
.globl input
input:
    # Yeu cau nguoi dung nhap so thu nhât
    la $a0, sothunhat
    li $v0, 4
    syscall

    # Luu so dau tien vao $t1
    li $v0, 6
    syscall
    mfc1 $t1, $f0

    # Yeu cau nguoi dung nhap so thu hai
    la $a0, sothuhai
    li $v0, 4
    syscall

    # Luu so thu hai vao $t2
    li $v0, 6
    syscall
    mfc1 $t2, $f0

    # Yeu cau nguoi dung chon phep cong hay tru
    la $a0, pheptinh
    li $v0, 4
    syscall
```

```
# Lưu biểu thức cần tính vào $t0
li $v0, 5
syscall
add $t0, $v0, $0

#####

# Lấy 1 bit sign
# Lấy bit sign của số thứ nhất, các bit còn lại bằng 0
#li $at, 0x80000000 # Tạo thanh ghi tạm có bit sign bằng
→ 1, các bit còn lại bằng 0
lui $at, 0x8000
ori $at, $at, 0x0000
and $s1, $t1, $at # Lấy bit sign

# Lấy bit sign của số thứ hai, các bit còn lại bằng 0
#li $at, 0x80000000 # Tạo thanh ghi tạm có bit sign bằng
→ 1, các bit còn lại bằng 0
lui $at, 0x8000
ori $at, $at, 0x0000
and $s2, $t2, $at # Lấy bit sign

# Kiểm tra $t0 để biết là phép cộng hay trừ
bne $t0, $0, subres # Nếu $t0 = 1 (khác 0) thực hiện phép
→ trừ
j addres # Nếu $t0 = 0 thực hiện phép cộng

# Thực hiện phép trừ
subres: # Đảo ngược bit sign của số thứ hai
#li $at, 0x80000000 # Tạo thanh ghi tạm có bit sign bằng
→ 1, các bit còn lại bằng 0
lui $at, 0x8000
ori $at, $at, 0x0000
addu $s2, $s2, $at # Đảo ngược bit sign của số thứ hai
```

```
# Thực hiện phép cộng
addres:
# Lay exponent va fraction của số đầu tiên
# Lay exponent của số đầu tiên lưu vào $s3
#li $at, 0x7F800000 # Tạo thanh ghi tạm có các bit
    → exponent bằng 1, các bit còn lại bằng 0
lui $at, 0x7F80
ori $at, $at, 0x0000
and $s3, $t1, $at # Lay exponent

# Lay fraction của số thứ nhất lưu vào $s4
#li $at, 0x007FFFFFFF # Tạo thanh ghi tạm có các bit
    → fraction bằng 1, các bit còn lại bằng 0
lui $at, 0x007F
ori $at, $at, 0xFFFF
and $s4, $t1, $at # Lay fraction

# Thêm bit prefix 1 vào trước fraction (dại diện cho 1.)
#li $at, 0x00800000
lui $at, 0x0080
ori $at, $at, 0x0000
or $s4, $s4, $at

# Lay exponent va fraction của số thứ hai
# Lay exponent của số thứ hai lưu vào $s5
#li $at, 0x7F800000 # Tạo thanh ghi tạm có các bit
    → exponent bằng 1, các bit còn lại bằng 0
lui $at, 0x7F80
ori $at, $at, 0x0000
and $s5, $t2, $at # Lay exponent

# Lay fraction của số thứ hai lưu vào $s6
#li $at, 0x007FFFFFFF # Tạo thanh ghi tạm có các bit
    → fraction bằng 1, các bit còn lại bằng 0
lui $at, 0x007F
ori $at, $at, 0xFFFF
and $s6, $t2, $at # Lay fraction
```

```
# Them bit prefix 1 vao truoc fraction (dai dien cho 1.)
#li $at, 0x00800000
lui $at, 0x0080
ori $at, $at, 0x0000
or $s6, $s6, $at

# Kiem tra exponent (so mu) cua hai so xem co bang nhau
↪ khong
expcheck:
# Neu so mu cua so thu hai nho hon so mu cua so thu nhât
slt $at, $s5, $s3
bne $at, $0, exp1 # Nhay toi exp1 de xu li

# Neu so mu cua so thu nhât nho hon so mu cua so thu hai
slt $at, $s3, $s5
bne $at, $0, exp2 # Nhay toi exp2 de xu li
j signcheck

exp1: # Tang exponent so thu hai len 1 don vi va dich phai
↪ fraction so thu hai 1 bit
#li $at, 0x00800000 # Tao thanh ghi co phan exponent bang
↪ 1
lui $at, 0x0080
ori $at, $at, 0x0000
addu $s5, $s5, $at # Cong exponent cho 1
srl $s6, $s6, 1 # Dich phai fraction cua 1 bit
j expcheck # Nhay toi expcheck de kiem tra lai

exp2: # Tang exponent so thu nhât len 1 don vi va dich phai
↪ fraction so thu nhât 1 bit
#li $at, 0x00800000 # Tao thanh ghi co phan exponent bang
↪ 1
lui $at, 0x0080
ori $at, $at, 0x0000
addu $s3, $s3, $at # Cong exponent cho 1
srl $s4, $s4, 1 # Dich phai fraction cua so 1 bit
j expcheck # Nhay toi expcheck de kiem tra lai
```

```
# Sau expcheck hai so co so mu bang nhau

# Kiem tra bit sign
signcheck:
    beq $s1, $s2, adding # Neu hai so cung dau, nhay toi
        → adding
    # Hai so trai dau
    beq $s4, $s6, triettieu # Neu fraction hai so bang nhau thi
        → triet tieu nhau, nhay den triettieu ( luu ket qua bang
        → 0 )
    slt $at, $s4, $s6
    beq $at, $0, subfirst # Tai day fraction so thu nhat lon
        → hon so thu hai, nhay toi subfirst ( lay $s4 - $s6 )
    j subsecond # Neu fraction so thu hai lon hon so thu nhat,
        → nhay toi subsecond ( lay $s6 - $s4 )

adding: # Cong hai so cung dau
    addu $s4, $s4, $s6 # Cong fraction cua hai so voi nhau luu
        → vao $s4
    addu $s0, $s1, $0 # Lay bit sign cua 2 so vao $s0
    j fix1 # Nhay toi fix1 de dieu chinh lai ket qua $s4

triettieu: # Hai so trai dau co fraction bang nhau, cho ket
        → qua bang 0
    and $t1, $t1, $0 # $t1 = 0 (ket qua bang 0)
    j output # Nhay toi output (in ket qua)

subfirst: # Lay fraction so thu nhat tru so thu hai, lay bit
        → sign cua so thu nhat
    subu $s4, $s4, $s6 # fraction so thu nhat - fraction so thu
        → hai ( $s4 - $s6 ) luu vao $s4
    addu $s0, $s1, $0 # Lay bit sign cua so thu nhat vao $s0
    j fix2 # Nhay toi fix2 de dieu chinh lai ket qua $s4

subsecond: # Lay fraction so thu hai tru so thu nhat, lay
        → bit sign cua so thu hai
    subu $s4, $s6, $s4 # fraction so thu hai - fraction so dau
        → ( $s6 - $s4 ) luu vao $s4
    addu $s0, $s2, $0 # Lay bit sign cua so thu hai vao $s0
    j fix2 # Nhay toi fix2 de dieu chinh lai ket qua $s4
```

```
fix1: # Điều chỉnh kết quả $s4 nếu hai số cùng dấu
# Kiểm tra bit dấu trước bit prefix (1.) đã cho lúc đầu có
→ bằng 1 hay không ( thay đổi từ 0 sang 1 )
# Nếu có thì dịch phải $s4 và tăng số mũ cho đến khi bit
→ tại vị trí đó bằng 0 thì nhảy tới saveres ( lưu kết quả
→ )
#li $at, 0x01000000 # Tạo thanh ghi có bit dấu trước bit
→ prefix bằng 1, các bit còn lại bằng 0
lui $at, 0x0100
ori $at, $at, 0x0000
and $s1, $s4, $at # Lấy bit dấu trước bit prefix của $s4
beq $s1, $0, saveres # Nếu bằng 0 thì nhảy tới saveres
# Dịch phải 1 bit
srl $s4, $s4, 1 # nếu bằng 1 thì dịch phải 1 bit
# Cộng exponent cho 1
#li $at, 0x00800000 # Tạo thanh ghi có phần exponent bằng
→ 1
lui $at, 0x0080
ori $at, $at, 0x0000
add $s3, $s3, $at # Cộng exponent của kết quả cho 1
j fix1 # Nhảy tới fix1 để kiểm tra lại

fix2: # Điều chỉnh kết quả $s4 nếu hai số khác dấu
# Kiểm tra bit prefix (1.) đã cho lúc đầu có bằng 0 hay
→ không ( thay đổi từ 1 sang 0 )
# Nếu có thì dịch trái $s4 và giảm số mũ cho đến khi bit
→ tại vị trí đó bằng 1 thì nhảy tới saveres ( lưu kết quả
→ )
#li $at, 0x00800000 # Tạo thanh ghi có bit prefix bằng 1,
→ các bit còn lại bằng 0
lui $at, 0x0080
ori $at, $at, 0x0000
and $s1, $s4, $at # Lấy bit prefix của $s4
bne $s1, $0, saveres # Nếu khác 0 ( = 1 ) thì nhảy tới
→ saveres
# Dịch trái 1 bit
sll $s4, $s4, 1 # Nếu bằng 0 thì dịch trái 1 bit
```

```
# Tru exponent cho 1
#li $at, 0x00800000 # Tao thanh ghi co phan exponent bang
→ 1
lui $at, 0x0080
ori $at, $at, 0x0000
sub $s3, $s3, $at # Tru exponent cua ket qua cho 1
j fix2 # Nhay toi fix2 dekiem tra lai

saveres: # Luu ket qua vao $t1
#li $at, 0x007FFFFFFF
lui $at, 0x007F
ori $at, $at, 0xFFFF
and $s4, $s4, $at # lay phan fraction cua ket qua, con lai
→ cho bang 0 ( cho bit prefix bang 0 )
addu $t1, $s0, $0 # Lay bit sign cua ket qua cho vao $t1
or $t1, $t1, $s3 # Lay exponent cua ket qua cho vao $t1
or $t1, $t1, $s4 # Lay fraction cua ket qua cho vao $t1
j output # Nhay toi output ( in ket qua )

output: # In ket qua
# Thong bao in ra ket qua
la $a0, ketqua
li $v0, 4
syscall

# Luu ket qua tu $t1 vao $f12 de in ra so thuc
mtc1 $t1, $f12

# In ra ket qua
li $v0, 2
syscall
```

```
asking: # Lua chon tiep tục tính toán hoặc kết thúc
        # Dữ liệu câu hỏi cho người dùng
        la $a0, cauhoi
        li $v0, 4
        syscall
        # Đọc kết quả được nhập vào
        li $v0, 5
        syscall
        # Nếu người dùng nhập 1 thì tiếp tục tính toán
        beq $v0, 1, input
        # Nếu người dùng nhập 0 thì kết thúc chương trình
        beq $v0, 0, done
        # Nếu người dùng nhập số khác, tiếp tục dữ liệu câu hỏi
        j asking

done:
        li $v0, 10
        syscall
```


2.2 Kết quả các testcase

2.2.1 Testcase 1

So hang thu nhat la: 1587.06

So hang thu hai la: 66.170

Phep tinh can thuc hien la (0: Cong, 1: Tru): 0

Ket qua la: 1653.23

Hình 2.2.1: Testcase 1

2.2.2 Testcase 2

So hang thu nhat la: -652.32

So hang thu hai la: -6769.658

Phep tinh can thuc hien la (0: Cong, 1: Tru): 0

Ket qua la: -7421.978

Hình 2.2.2: Testcase 2

2.2.3 Testcase 3

So hang thu nhat la: 0

So hang thu hai la: -982.658

Phep tinh can thuc hien la (0: Cong, 1: Tru): 0

Ket qua la: -982.658

Hình 2.2.3: Testcase 3

2.2.4 Testcase 4

So hang thu nhat la: 6561.56

So hang thu hai la: 8484.556

Phep tinh can thuc hien la (0: Cong, 1: Tru): 1

Ket qua la: -1922.9961

Hình 2.2.4: Testcase 4

2.2.5 Testcase 5

So hang thu nhat la: 650.25

So hang thu hai la: -3261.075

Phep tinh can thuc hien la (0: Cong, 1: Tru): 1

Ket qua la: 3911.325

Hình 2.2.5: Testcase 5

2.2.6 Testcase 6

So hang thu nhat la: 0

So hang thu hai la: -135.068

Phep tinh can thuc hien la (0: Cong, 1: Tru): 1

Ket qua la: 135.068

Hình 2.2.6: Testcase 6

2.2.7 Testcase 7

So hang thu nhat la: -123.35

So hang thu hai la: 23.235

Phep tinh can thuc hien la (0: Cong, 1: Tru): 0

Ket qua la: -100.115

Hình 2.2.7: Testcase 7

2.2.8 Testcase 8

So hang thu nhat la: 1021.2376

So hang thu hai la: -565.575

Phep tinh can thuc hien la (0: Cong, 1: Tru): 0

Ket qua la: 455.6626

Hình 2.2.8: Testcase 8

2.2.9 Testcase 9

```
So hang thu nhat la: 415.7232
So hang thu hai la: 0
```

```
Phep tinh can thuc hien la (0: Cong, 1: Tru): 0
Ket qua la: 415.7232
```

Hình 2.2.9: Testcase 9

2.2.10 Testcase 10

```
So hang thu nhat la: -12.343
So hang thu hai la: -2.346
```

```
Phep tinh can thuc hien la (0: Cong, 1: Tru): 1
Ket qua la: -9.997001
```

Hình 2.2.10: Testcase 10

2.2.11 Testcase 11

```
So hang thu nhat la: -0.325
So hang thu hai la: 1.851
```

```
Phep tinh can thuc hien la (0: Cong, 1: Tru): 1
Ket qua la: -2.1759999
```

Hình 2.2.11: Testcase 11

2.2.12 Testcase 12

```
So hang thu nhat la: -654.1329
So hang thu hai la: 0

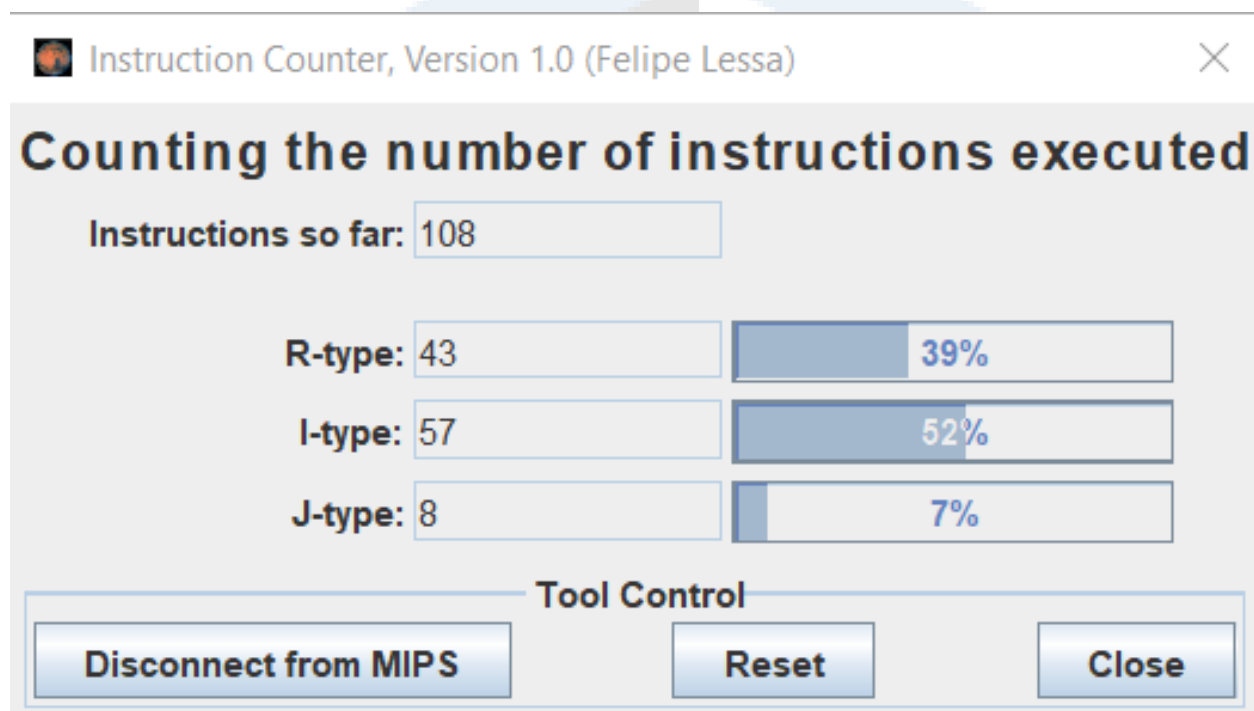
Phep tinh can thuc hien la (0: Cong, 1: Tru): 1
Ket qua la: -654.1329
```

Hình 2.2.12: Testcase 12

Phần 3

Thống kê lệnh và tính toán thời gian

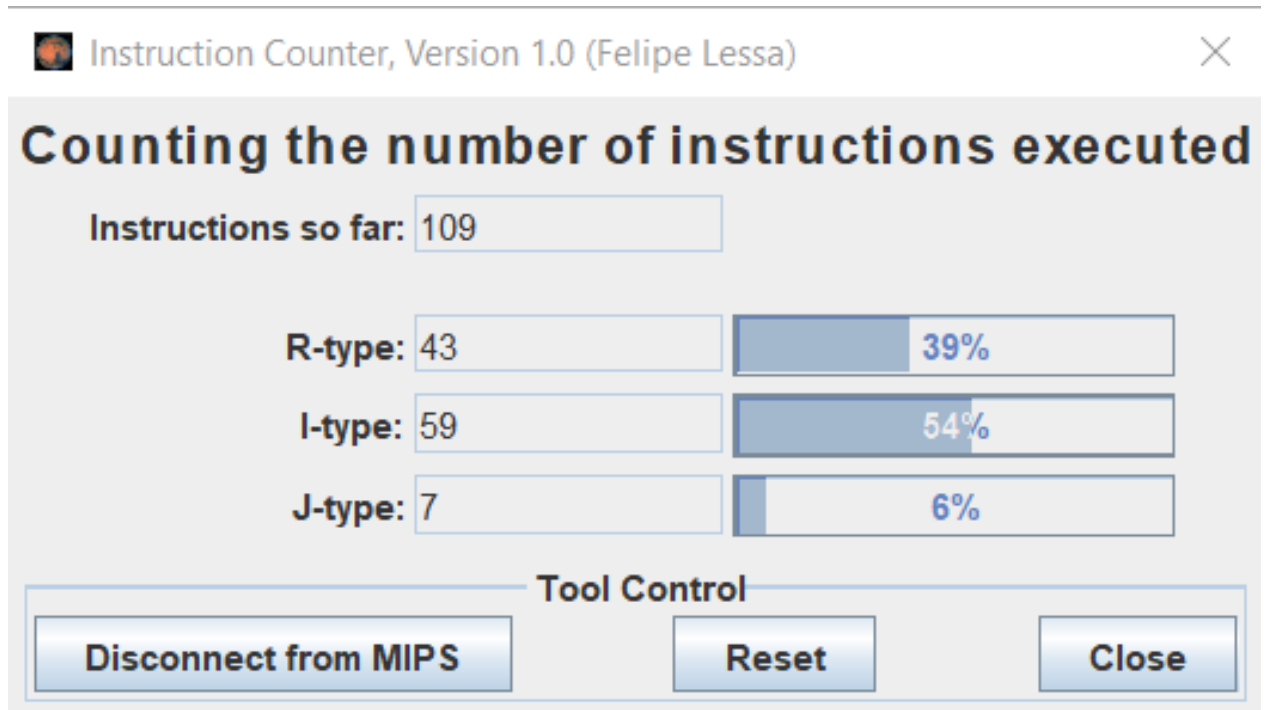
3.0.1 Testcase 1: 1587.06 + 66.170



Hình 3.0.1: Thống kê lệnh testcase 1

$$Time = CPI * IC * CycleTime = 1 * 108 * \frac{1}{2 * 10^9} = 5.4 * 10^{-8}(s)$$

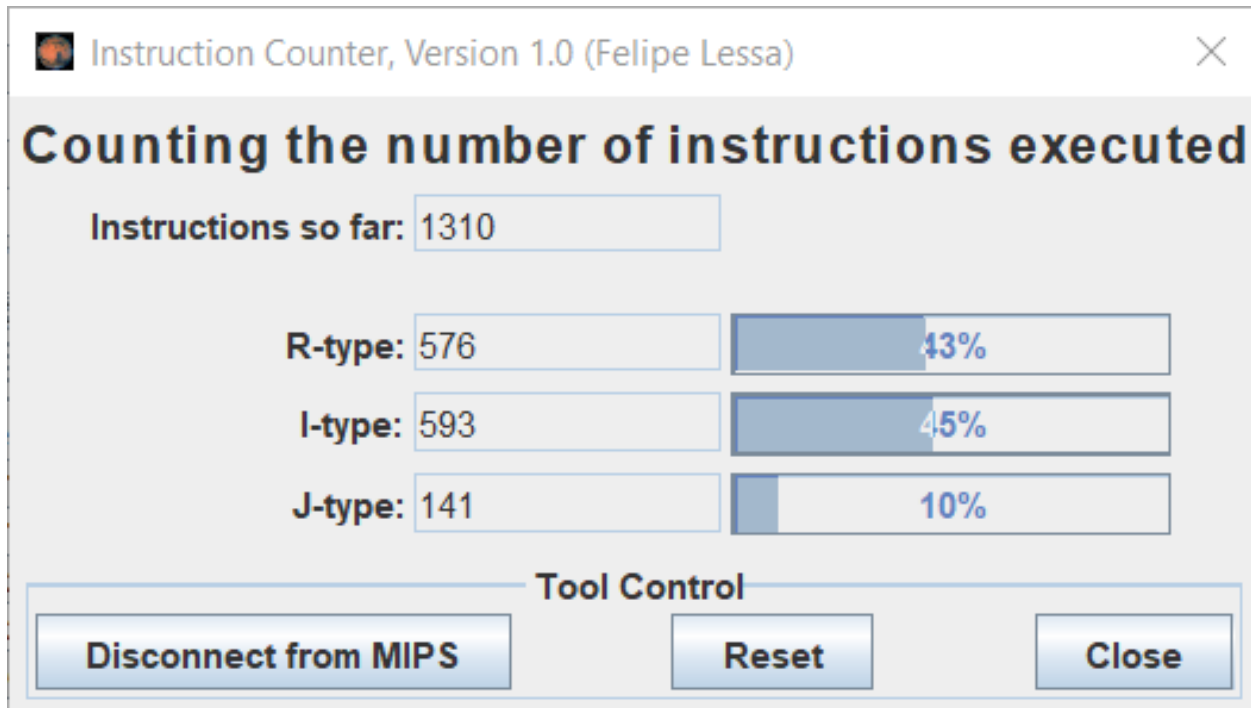
3.0.2 Testcase 2: -652.32 + -6769.658



Hình 3.0.2: Thống kê lệnh testcase 2

$$Time = CPI * IC * CycleTime = 1 * 109 * \frac{1}{2 * 10^9} = 5.45 * 10^{-8}(s)$$

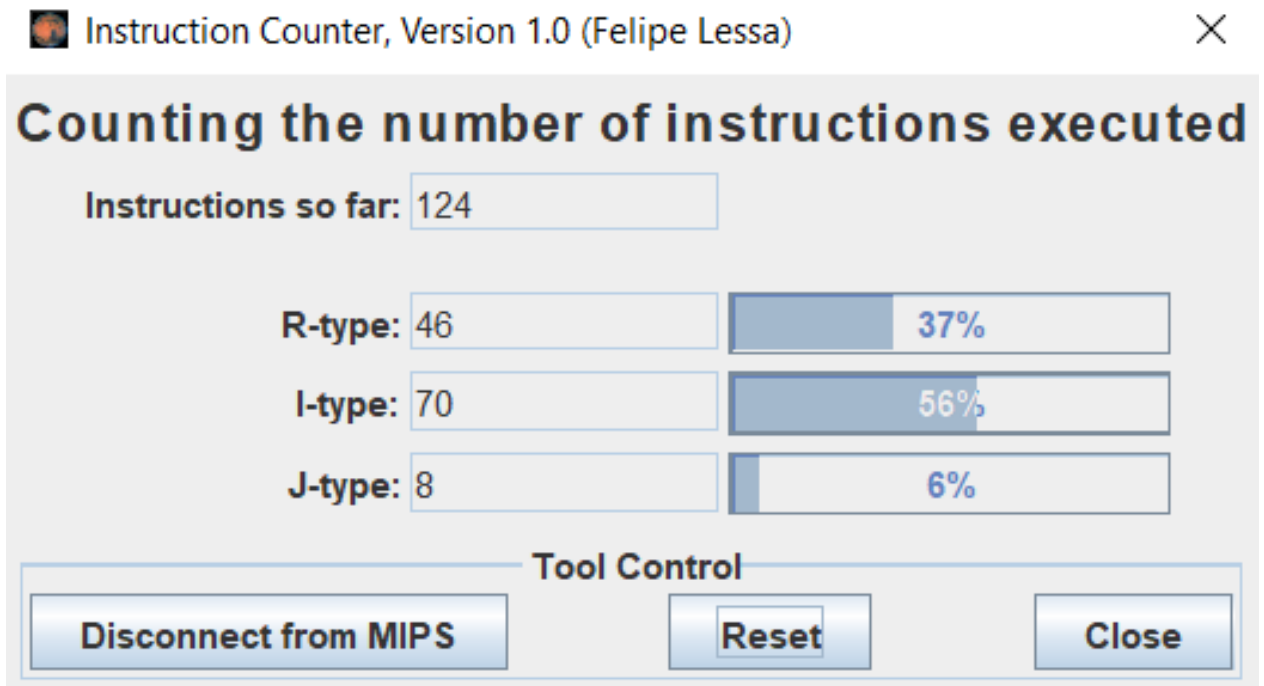
3.0.3 Testcase 3: 0 + -982.658



Hình 3.0.3: Thống kê lệnh testcase 3

$$Time = CPI * IC * CycleTime = 1 * 1310 * \frac{1}{2 * 10^9} = 6.55 * 10^{-7}(s)$$

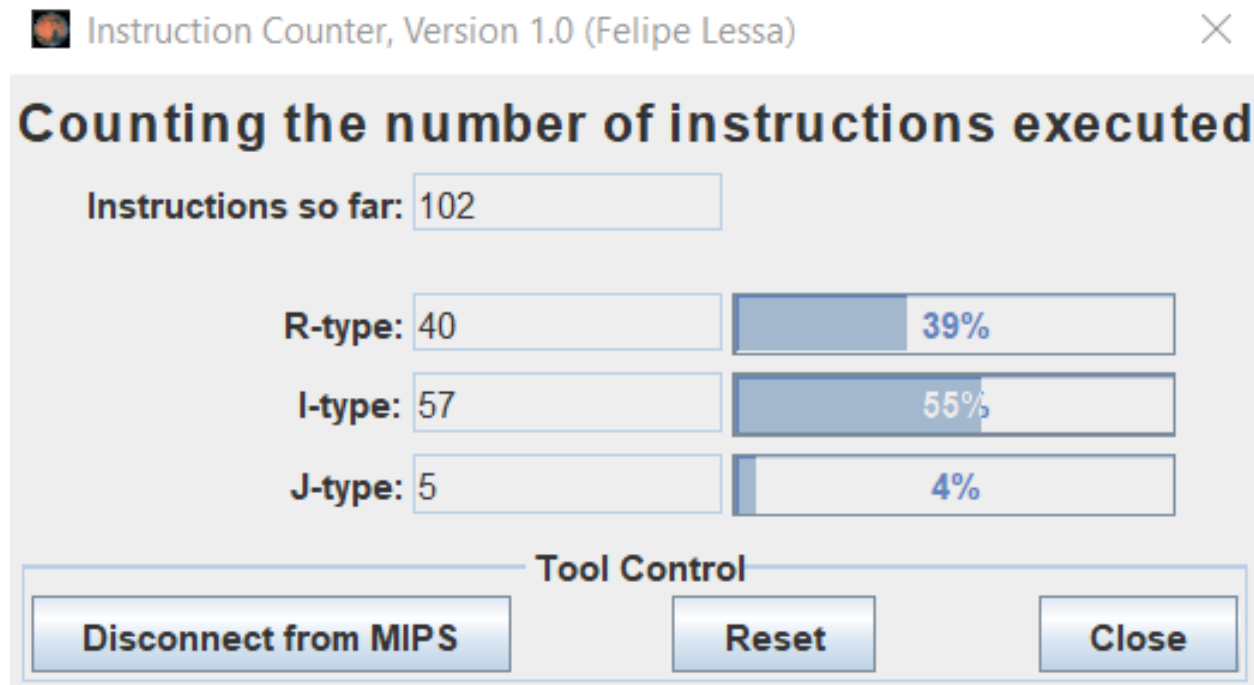
3.0.4 Testcase 4: 6561.56 - 8484.556



Hình 3.0.4: Thống kê lệnh testcase 4

$$Time = CPI * IC * CycleTime = 1 * 124 * \frac{1}{2 * 10^9} = 6.2 * 10^{-8}(s)$$

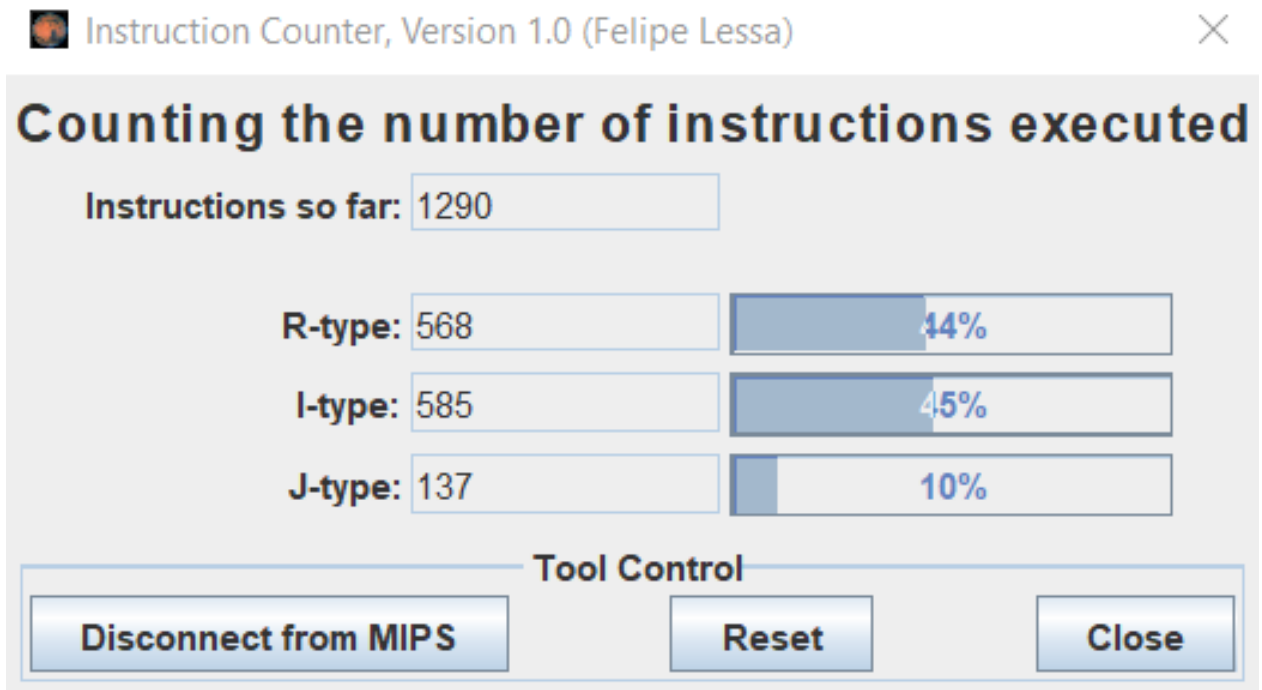
3.0.5 Testcase 5: 650.25 - -3261.075



Hình 3.0.5: Thống kê lệnh testcase 5

$$Time = CPI * IC * CycleTime = 1 * 102 * \frac{1}{2 * 10^9} = 5.1 * 10^{-8}(s)$$

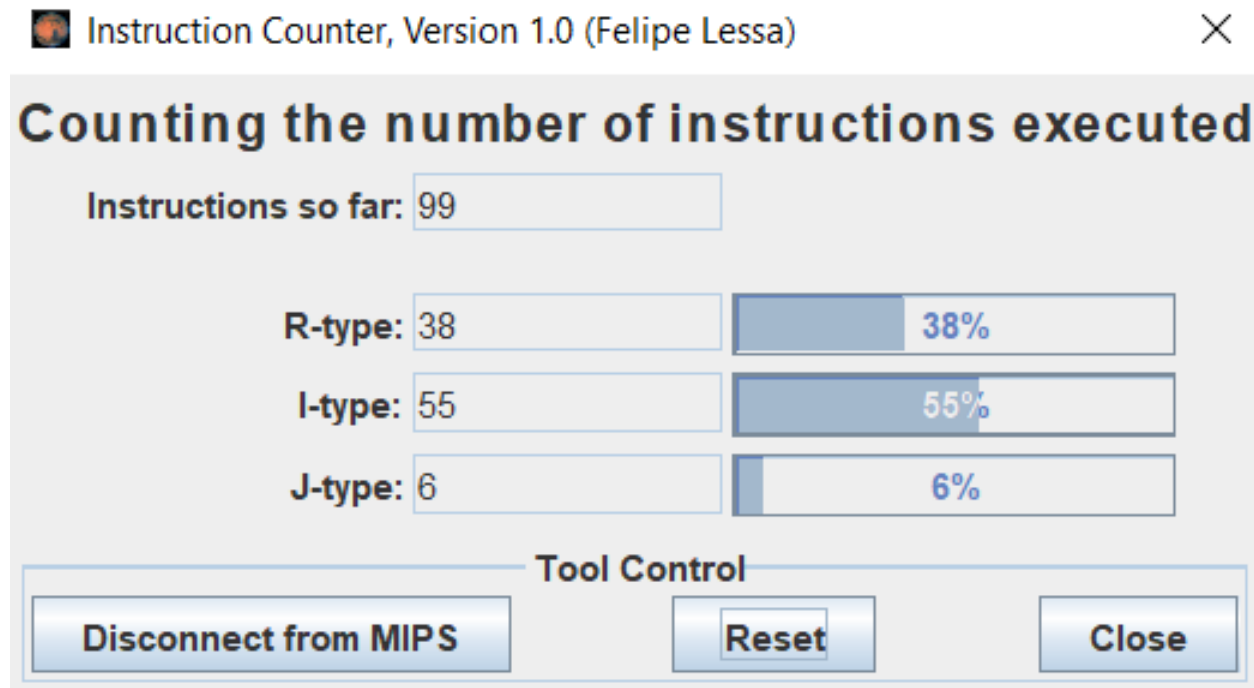
3.0.6 Testcase 6: 0 - -135.068



Hình 3.0.6: Thống kê lệnh testcase 6

$$Time = CPI * IC * CycleTime = 1 * 1290 * \frac{1}{2 * 10^9} = 6.45 * 10^{-7}(s)$$

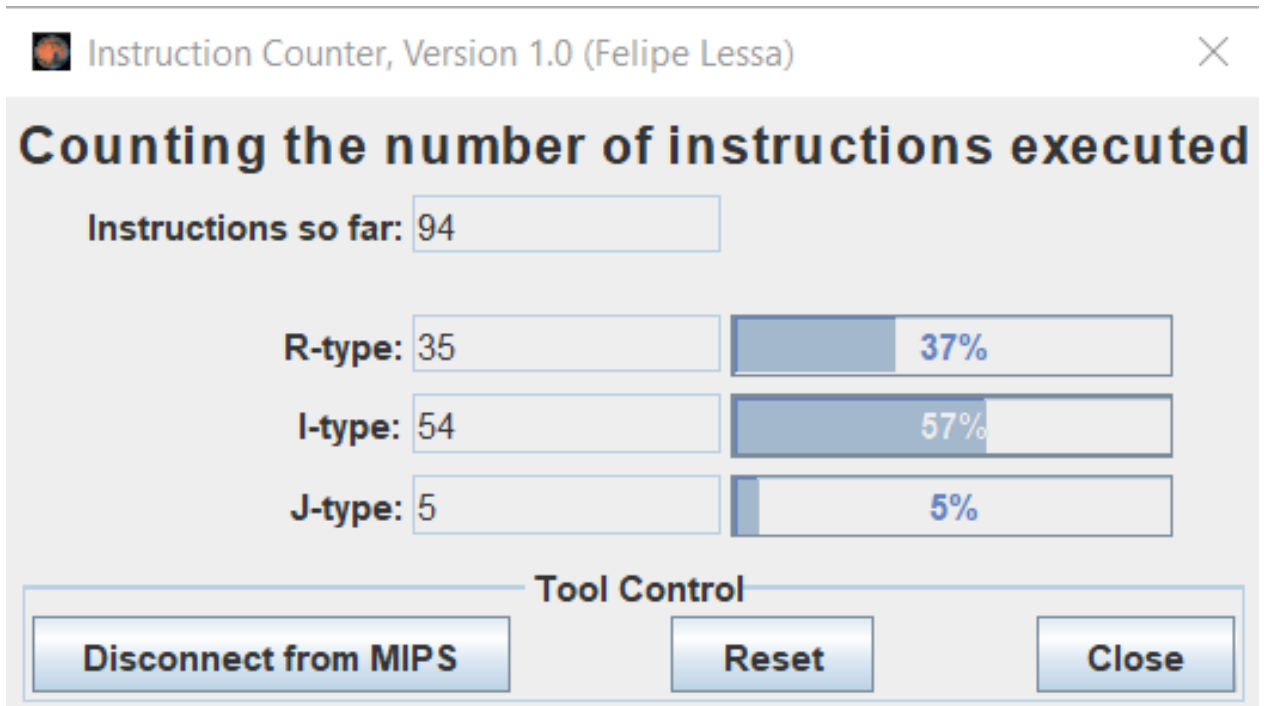
3.0.7 Testcase 7: -123.35 + 23.235



Hình 3.0.7: Thống kê lệnh testcase 7

$$Time = CPI * IC * CycleTime = 1 * 99 * \frac{1}{2 * 10^9} = 4.95 * 10^{-8}(s)$$

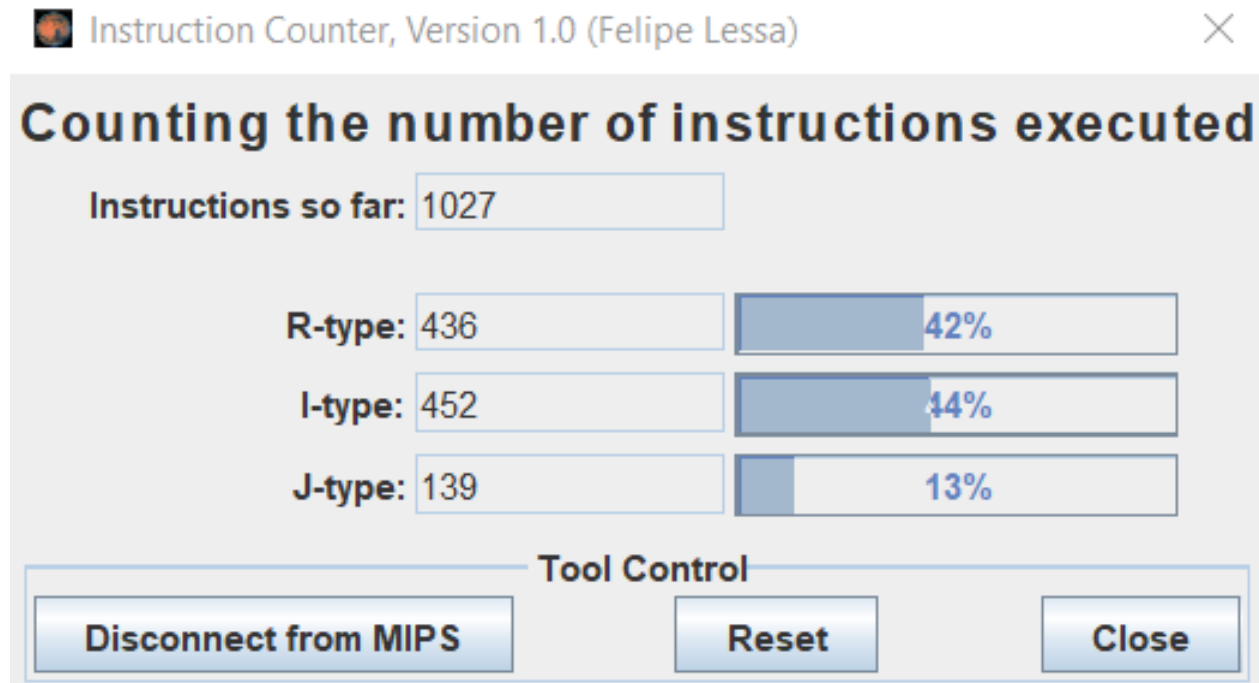
3.0.8 Testcase 8: 1021.2376 + -565.575



Hình 3.0.8: Thống kê lệnh testcase 8

$$Time = CPI * IC * CycleTime = 1 * 94 * \frac{1}{2 * 10^9} = 4.7 * 10^{-8}(s)$$

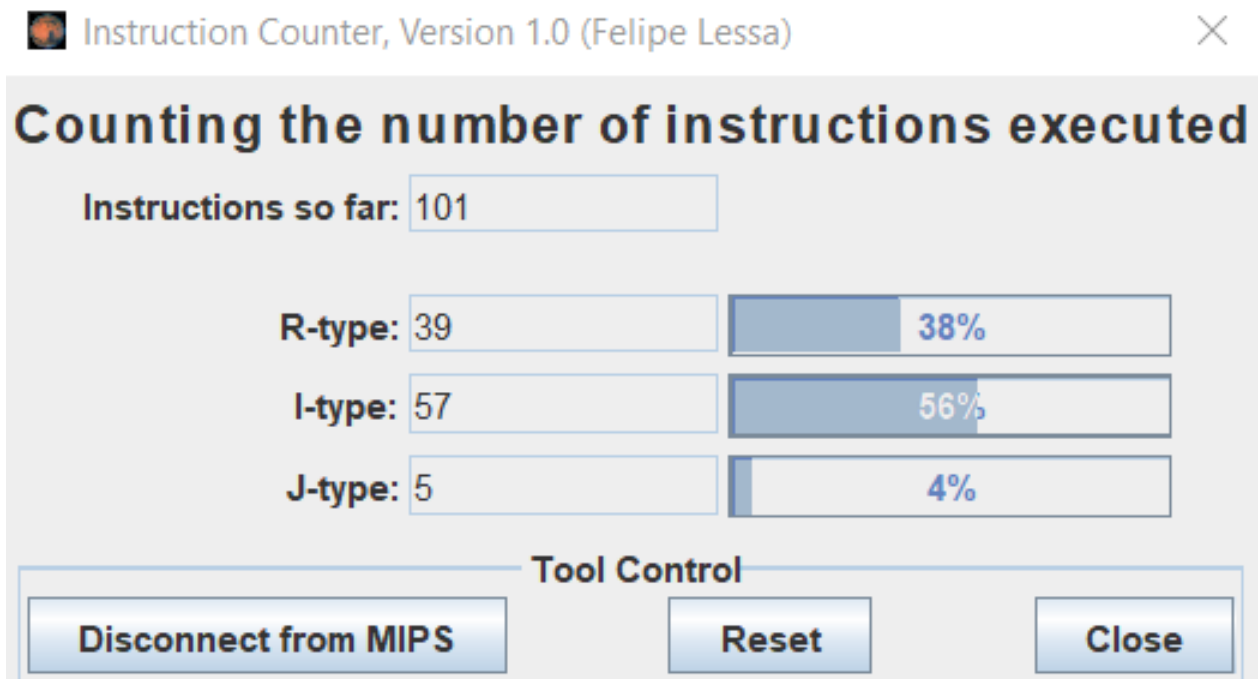
3.0.9 Testcase 9: 415.7232 + 0



Hình 3.0.9: Thống kê lệnh testcase 9

$$Time = CPI * IC * CycleTime = 1 * 1027 * \frac{1}{2 * 10^9} = 5.135 * 10^{-7}(s)$$

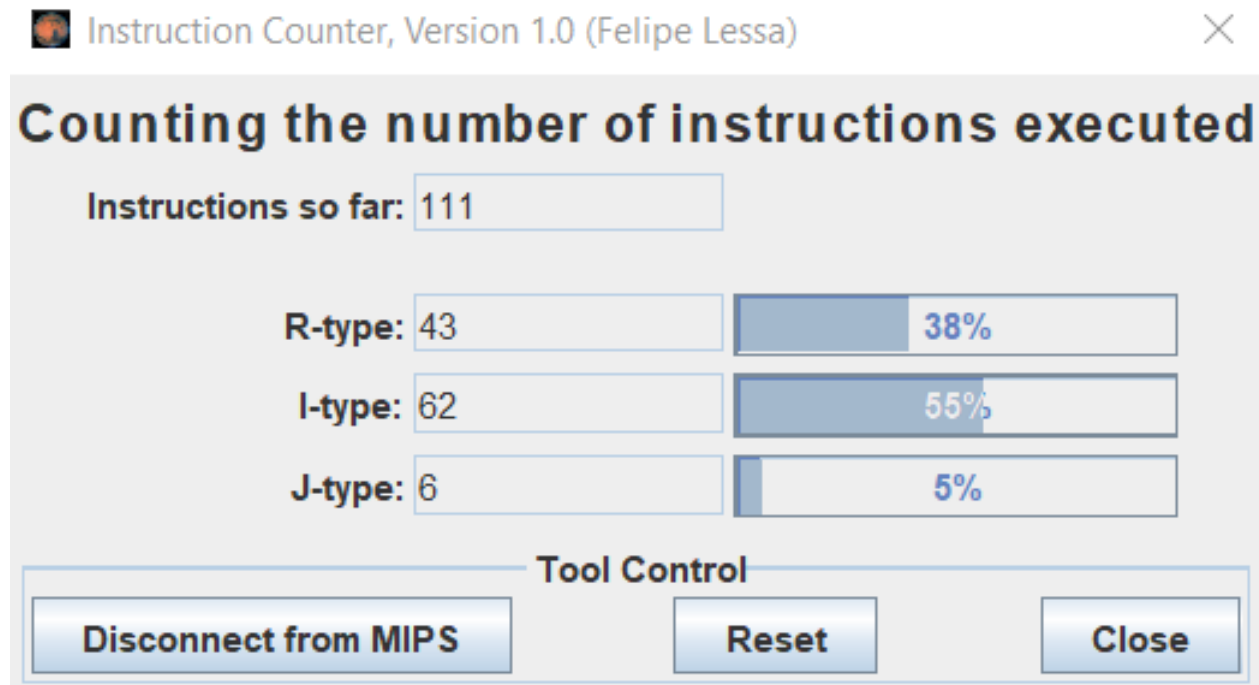
3.0.10 Testcase 10: -12.343 - -2.346



Hình 3.0.10: Thống kê lệnh testcase 10

$$Time = CPI * IC * CycleTime = 1 * 101 * \frac{1}{2 * 10^9} = 5.05 * 10^{-8}(s)$$

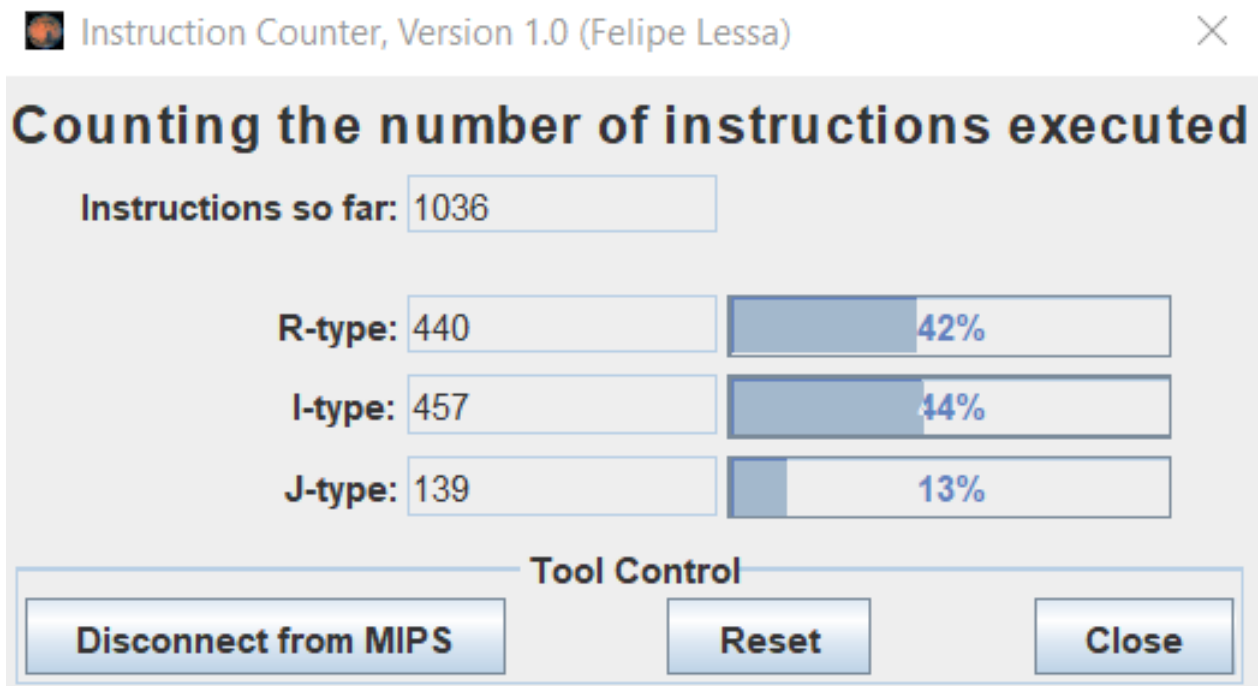
3.0.11 Testcase 11: -0.325 - 1.851



Hình 3.0.11: Thống kê lệnh testcase 11

$$Time = CPI * IC * CycleTime = 1 * 111 * \frac{1}{2 * 10^9} = 5.55 * 10^{-8}(s)$$

3.0.12 Testcase 12: -654.1329 - 0



Hình 3.0.12: Thống kê lệnh testcase 12

$$Time = CPI * IC * CycleTime = 1 * 1036 * \frac{1}{2 * 10^9} = 5.18 * 10^{-7}(s)$$