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 (CO2008) - Học kì 232

Đề 1:

Nhân 2 số nguyên 32bit

Giảng viên hướng dẫn: Nguyễn Xuân Minh
Sinh viên thực hiện: Lê Đức Cường
Nguyễn Hạo Duy

TP. HỒ CHÍ MINH, THÁNG 5/2024

Mục lục

1 Lời mở đầu

Bộ tính toán số học (ALU) là một thành phần thiết yếu trong kiến trúc máy tính, đóng vai trò thực hiện các phép toán số học và logic cho CPU. Nó hoạt động như một bộ xử lý chuyên dụng, nhận dữ liệu đầu vào từ các thanh ghi, thực hiện các phép toán theo hướng dẫn và lưu trữ kết quả vào các thanh ghi khác. Với đề tài được phân công *" Phép nhân hai số nguyên 32bit"*, được thực hiện thông qua một loạt các bước phức tạp bao gồm chuyển đổi số sang dạng nhị phân, thực hiện phép nhân từng bit và xử lý bit dấu. ALU đóng vai trò quan trọng trong việc thực hiện phép nhân nhanh chóng và chính xác.

Cùng với việc thực hiện đề tài này, thay mặt nhóm em xin gửi lời cảm ơn sâu sắc đến thầy Nguyễn Xuân Minh đã nhiệt tình chỉ dẫn các bước, hướng nghiên cứu, thực hiện cũng như yêu cầu cần có của đề tài trong suốt thời gian thực hiện đề tài. Trong quá trình thực hiện, dù đã rất cố gắng nhưng không tránh khỏi những sai sót và hạn chế nhất định. Vì vậy chúng em rất mong nhận được sự góp ý, bổ sung thêm của thầy để đề tài được hoàn thiện hơn.

2 Đề bài

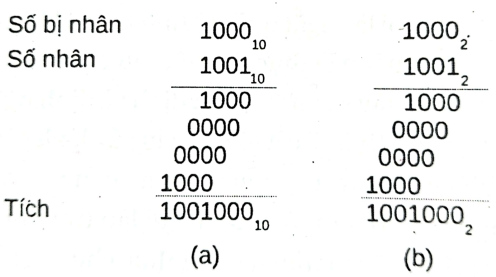
Viết chương trình hiện thực giải thuật nhân số nguyên trong textbook (hình 3.4 hoặc 3.5), áp dụng cho số có dấu. Dữ liệu đầu vào đọc từ file lưu trữ dạng nhị phân trên đĩa INT2.BIN (2 trị x 4 bytes = 8 bytes).

3 Yêu cầu

- ⋆ Chương trình viết và chạy trên MARS MIPS 4.5.
- ⋆ Trình bày giải pháp hiện thực, giải thuật
- ⋆ Thống kê số lệnh, loại lệnh (instruction type) sử dụng trong chương trình
- ⋆ Tính thời gian chạy của chương trình (CR=1GHz)
- ⋆ Kết quả kiểm thử
- ⋆ Code
    - ∘ Code style phải rõ ràng, có chú thích.
    - ∘ Phải có gọi hàm. Truyền tham số và trả kết quả khi gọi hàm theo quy ước của thanh ghi ($Ai chứa tham số, $Vi hoặc $fi chứa giá trị trả về). .
    - ∘ In thông tin ra màn hình để kiểm tra.
- ⋆ Nộp báo cáo
    - ∘ File báo cáo (không source code) định dạng .PDF (Bc_nhom037.pdf).
    - ∘ File mã nguồn (Mn_037.asm) và các file khác(nếu có).

4 Phép nhân số nguyên 32bit

Trong phép nhân, thừa số thứ nhất sẽ được gọi là số bị nhân (multipli-cand), thừa số thứ hai là số nhân (multiplier) và kết quả phép nhân là tích (product). Giải thuật nhân 2 số ở dạng thập phân sử dụng " giấy và bút" đã đươuc giới thiệu trong trường phổ thông là chọn từng ký số của số nhân từ phải sang trái, thực hiện phép nhân của ký số này với số bị nhân và dịch các giá trị tạm thời này sang trái ở mỗi phép tính.

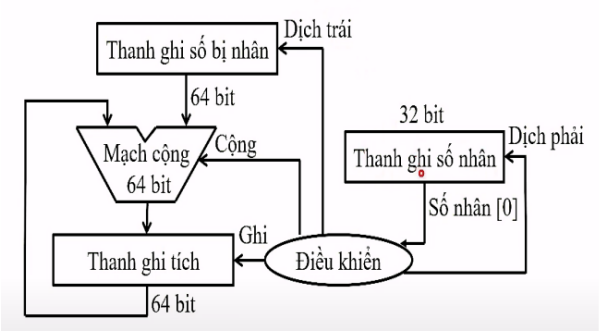


Hình 1: Ví dụ về phép nhân 2 số nguyên không dấu

Chọn từng ký số của số nhân (b) từ phải sang trái, thực hiện phép nhân của ký số này với số bị nhân (a) và dịch các giá trị tích tạm thời này sang trái ở mỗi phép tính. Đối với số nhị phân thì chỉ có 2 ký số là 0 và 1. Do đó, mỗi bước tính toán ở phép nhị phân chỉ có hai lựa chọn:

- Sao chép giá trị số bị nhân (a) và đặt vào vị trí thích hợp nếu ký số ở số nhân (b) đang xét là 1
- Đặt giá trị 0 vào vị trí thích hợp nếu ký số ở số nhân (b) đang xét là 0

4.1 Bộ nhân tuần tự



Hình 2: Kiến trúc bộ nhân tuần tự 32bit

Trong kiến trúc trên, số nhân được chứa trong thanh ghi 32 bit. Thanh ghi tích sẽ có kích thước 64 bit và được khởi tạo bằng 0. Bởi vì số nhân có kích thước 32bit nên số bị nhân cần dịch sẽ cần được dịch sang trái 32 lần. Do đó, cần dùng thanh ghi 64bit để chứa số bị nhân để đảm bảo rằng việc dịch trái không gây ra mất mát bất kì bit nào của số bị nhân. Ở thời điểm ban

đầu, thanh ghi 64bit này sẽ chứa 32bit của số bị nhân ở 32bit thấp và 32bit cao được gán bằng 0. Mỗi bước tính toán thanh ghi chứa số nhân sẽ được dịch sang phải 1 bit và bit cuối cùng sẽ được xử lý.

Lần lặp	Bước	Số nhân	Số bị nhân	Tích
0	Khởi tạo	0101	0000 0111	0000 0000
1	1a. m0 = 1 (cộng)	0101	0000 0111	*0000 0111*
	2. dịch	***0010***	***0000 1110***	0000 0111
2	1. m0 = 0	0010	0000 1110	0000 0111
	2. dịch	***0001***	***0001 1100***	0000 0111
3	1a. m0 = 1 (cộng)	0001	0001 1100	*0010 0011*
	2. dịch	***0000***	***0011 1000***	0010 0011
4	1. m0 = 0	0000	0011 1000	0010 0011
	2. dịch	***0000***	*0111 0000*	0010 0011

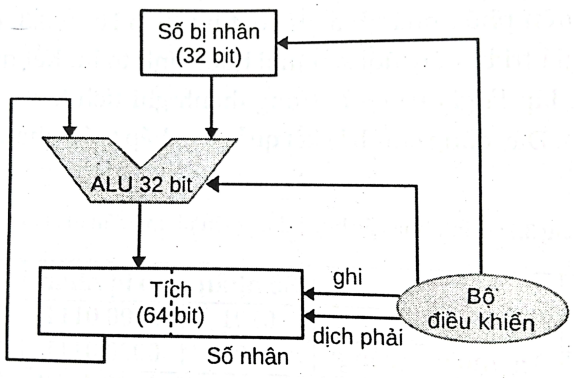
Hình 3: Minh họa các bước thực hiện phép nhân $0111_2 * 0101_2$

4.2 Bộ nhân tuần tự tối ưu

Kiến trúc bộ nhân tuần tự trong mục trên vẫn còn một số điểm chưa tối ưu như cần phải sử dụng thanh ghi 64bit để lưu trữ giá trị số bị nhân, cần sử dụng bộ công cụ ALU 64bit. Kiến trúc bộ nhân tuần tự trong hình 4 sẽ khắc phục những vấn đề này bằng cách chỉ sử dụng thanh ghi 32bit thay cho bộ cộng 64bit và không cần của một thanh ghi riêng biệt để lưu giá trị số nhân.

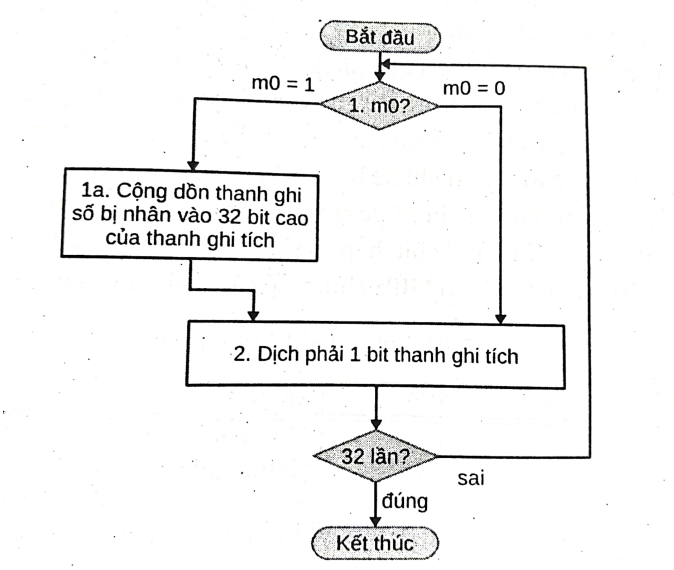
Kiến trúc bộ nhân tối ưu thực hiện việc dịch phải thanh ghi chứa tích và cộng dồn 32 bits số bị nhân vào 32 bits cao của thanh ghi tích. Ngoài ra, tại thời điểm ban đầu, giá trị của số nhân được lưu trữ vào trong 32 bits thấp của thanh ghi tích thay vì sử dụng một thanh ghi riêng biệt lưu trữ. Tại mỗi bước tính toán, bit cuối cùng của thanh ghi tích (hay chính xác hơn là bit thấp nhất của giá trị số nhân - bit m0) cũng sẽ được kiểm tra. Nếu giá trị m0 là 0 thì chỉ thực hiện dịch phải thanh ghi tích; ngược lại thực hiện việc cộng dồn thanh ghi số bị nhân vào 32 bits cao của thanh ghi tích. Sau 32 bước thực hiện thì giá trị chứa trong thanh ghi tích chính xác là kết quả của phép nhân.

Ở bài này, Nhóm sử dụng bộ nhân tuần tự tối ưu để giảm bớt số lượng thanh ghi.



Hình 4: Kiến trúc bộ nhân tuần tụ 32bit tối ưu

4.3 Hoạt động của bộ nhân tuần tự



Hình 5: Hoạt động của bộ nhân tuần tự trong hình 4

Lần lặp	Bước	Số bị nhân	Tích
0	Khởi tạo	0111	0000 0101
1	1a. m0 = 1 (cộng)	0111	***0111 0101***
	2. dịch	0111	**0011 1010**
2	1. m0 = 0	0111	0011 1010
	2. dịch	0111	**0001 1101**
3	1a. m0 = 1 (cộng)	0111	***1000 1101***
	2. dịch	0111	**0100 0110**
4	1. m0 = 0	0111	0100 0110
	2. dịch	0111	**0010 0011**

Hình 6: Minh họa thực hiện phép nhân $0111_2 * 0101_2$ sử dụng bộ nhân tuần tự tối ưu

5 Thực thi phép nhân 2 số nguyên trong MIPS

5.1 Giải pháp hiện thực

- Đọc 2 số nguyên từ file "INT2.BIN"
- In 2 số nguyên này ra màn hình
- Thực hiện phép nhân hai số nguyên (xem lại giải thuật ở phần đã được trình bày)
- In kết quả phép nhân ra màn hình

5.2 Hiện thực trong MIPS

Dữ liệu đầu vào là 2 giá trị lấy từ trong file `INT2.BIN` sau đó dữ liệu được upload lên 2 biến `dulieu1` và `dulieu2` trong phần `data`.

```
Diemtrave:
    bltz $s0, is_negative_s0  # Branch if s0 < 0 to label is_negative
    bltz $s1, is_negative_s1  # Branch if s1 < 0 to label is_negative
    j tieptuc
is_negative_s0:
    addi $t0, $zero, -1
    mul $s0, $s0, $t0
    li $s6, -1 # xét âm dương
    j Diemtrave
is_negative_s1:
    addi $t0, $zero, -1
    mul $s1, $s1, $t0
    li $s7, -1 # xét âm dương
    j Diemtrave
```

Hàm trên để xét giá trị trong `dulieu1` và `dulieu2` là âm hay dương. Nếu là âm thì sẽ nhân cho -1 để trở thành số dương mới đưa vào giải thuật nhân được.

Giải thuật nhân: kết quả phép nhân sẽ được chuyển vào 2 thanh ghi:

- Thanh ghi `$s3` lưu 32 bit thấp.
- Thanh ghi `$s4` lưu 32 bit cao của kết quả.

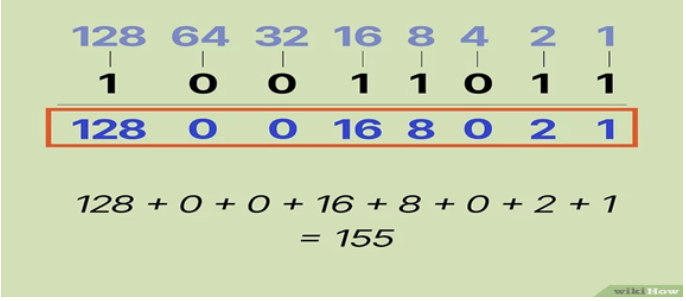
```
MyMult:
    move $s3, $0        # lw product
    move $s4, $0        # hw product
    beq $s1, $0, done
    beq $s0, $0, done
    move $s2, $0        # extend multiplicand to 64 bits
loop:
    andi $t0, $s0, 1    # LSB(multiplier)
    beq $t0, $0, next   # skip if zero
    addu $s3, $s3, $s1  # lw(product) += lw(multiplicand)
    sltu $t0, $s3, $s1  # catch carry-out(0 or 1)
    addu $s4, $s4, $t0  # hw(product) += carry
    addu $s4, $s4, $s2  # hw(product) += hw(multiplicand)
next:
    # shift multiplicand left
    srl $t0, $s1, 31    # copy bit from lw to hw
    sll $s1, $s1, 1
    sll $s2, $s2, 1
    addu $s2, $s2, $t0

    srl $s0, $s0, 1     # shift multiplier right
    bne $s0, $0, loop
done:
    jr $ra
```

Do kết quả phép nhân sẽ lớn hơn 32 bit nên không có thanh ghi số nguyên nào đủ để in ra kết quả nên chúng em bắt buộc phải chuyển kết quả về dạng số double (64 bit). Do 2 thanh ghi riêng biệt không có khả năng kết hợp chúng lại thành kết quả ở dạng double. Nên chúng em đã nghĩ ra cách chuyển tất cả dữ liệu trong 32 bit thấp (s3) từ dạng nhị phân về dạng double.

Cách hiện thực như sau:

Bước 1: Đầu tiên sẽ dùng cách nhị phân về dạng hệ thập phân như hình dưới đây



Hình 7: Ví dụ chuyển 1 số hệ 2 thành hệ 10

Bước 2: Tạo vòng lặp lặp 32 lần qua từ bit đầu tiên đến bit cuối cùng của thanh ghi `$s3` để lấy ra từng bit.

- Trong mỗi vòng lặp, sẽ lấy ra từng bit. Nếu tại đó là bit 1, thì sẽ lấy 2 mũ lên sau đó chuyển đổi về double.

Bước 3: Sau khi trải qua 32 lần lặp và cộng dồn nếu là bit 1, thì kết quả được lưu vào thanh ghi double `$f6`.

- Sau đó, thực hiện y chang cách trên cho 32 bit cao (`$s4`).
- Giá trị sẽ cộng dồn vào thanh ghi `$f6` cũ để ra được kết quả chính xác.

Hàm chuyển 32 bit thấp s3 thành dạng double và cộng dồn kết quả lưu vào $f6:

```
jump_1:   # tao gia tri cho 2 thanh ghi
    li $t0, 0
    li $t3, 1

loop_1:   # vòng lặp qua 32 bit của $s3
    beq $t0, 32, jump_2
    andi $t1, $s3, 1
    beq $t1, 1, nhan
    j tiep_1

nhan: # mỗi vòng lặp thực hiện chuyển 32 bit thành hệ double và cộng dồn giá trị vào $f6
    sllv $t4, $t3, $t0
    mtc1 $t4, $f0
    cvt.d.w $f0, $f0
    add.d $f6, $f6, $f0
    j tiep_1
```

```
tiep_1:  # dịch sang phải 1 bit cho thanh ghi $s3 và tiếp tục vòng lặp
    srl $s3, $s3, 1
    addi $t0, $t0, 1
    j loop_1

jump_2:  # điểm nhảy cuối cùng sau khi hoàn tất vòng lặp
```

Hàm chuyển 32 bit cao s4 thành dạng double và cộng dồn kết quả cũ lưu vào $f6:

```
jump_2:   # lay cac giá tri ra dung
    s.d $f6, buffer_product
    l.d $f6, buffer_product
    li $t5, 0
    l.d $f2, double_1  # thay doi_0
    l.d $f8, double_2  # 2
    l.d $f0, double_3  # 1
    l.d $f14, double_4

loop_2:   # vòng lặp qua 32 bit của $s4
    beq $t5, 32, print_2
    andi $t6, $s4, 1
    beq $t6, 1, nhan_2
    j tiep_2

nhan_2:
# mỗi vòng lặp thực hiện chuyển 32 bit trong thanh ghi $s4 thành hệ double
# cộng dồn giá trị vào $f6, $f6 lấy từ giá trị cũ
    mul.d $f4, $f0, $f2
    add.d $f6, $f6, $f4
    j tiep_2

tiep_2:   # dịch sang phải 1 bit cho thanh ghi $s4 để tiếp tục vòng lặp
    srl $s4, $s4, 1
    addi $t5, $t5, 1
    mul.d $f2, $f2, $f8
    j loop_2
```

6 Tổng kết

6.1 Thống kê số lệnh, loại lệnh

6.1.1 R - type

STT	Tên lệnh	Chức năng
1	sll	R[rd] = R[rt] « shamt
2	srl	R[rd] = R[rt] » shamt
3	add	R[rd] = R[rs] + R[rt]
4	jr	PC = R[rs]
5	xor	$R[rd] = R[rs] \oplus R[rt]$
6	or	$R[rd] = R[rs] \mid R[rt]$
7	and	$R[rd] = R[rs] \& R[rt]$
8	move	R[rd] = R[rs]
9	mul.d	F[fd],F[fd+1] = F[fs],F[fs+1] * F[ft],F[ft+1]

Bảng 1: Các lệnh loại R đã được sử dụng

6.1.2 I - type

STT	Tên lệnh	Chức năng
1	lw	R[rt]=M[R[rs]+SignExtImm]
2	sw	M[R[rs]+SignExtImm]=R[rt]
3	addi	R[rt]=R[rs]+SignExtImm
4	ble	if $t_0 \leq t_1$ then PC = Addr
5	bgtz	if $t_0 > 0$ then $PC = Addr$
6	beqz	if $t_0 == 0$ then $PC = Addr$
7	bne	if $t_0 \neq t_1$ then $PC = Addr$
8	li	R[rd] = immediate

Bảng 2: Các lệnh loại I đã được sử dụng

6.1.3 J - type

STT	Tên lệnh	Chức năng
1	j	PC = jumpAddr
2	jal	R[31]=PC+8; PC = jumpAddr

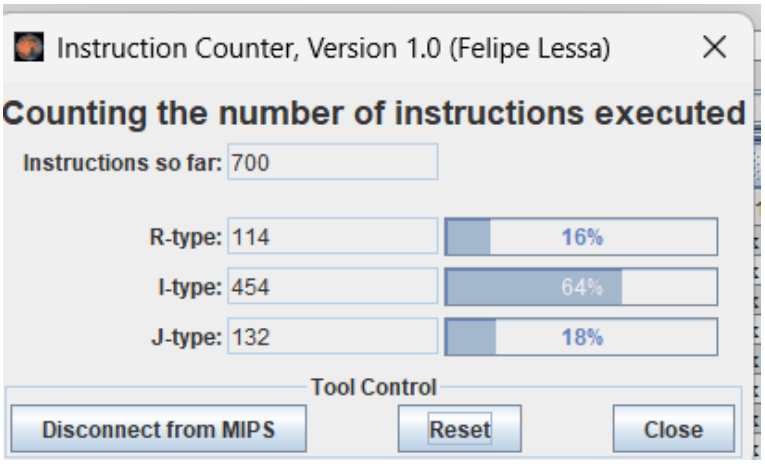
Bảng 3: Các lệnh loại J đã được sử dụng

6.2 Kiểm tra testcase

6.2.1 Testcase 1

```
Du lieu 1 = 554544554
Du lieu 2 = 0
0.0
-- program is finished running --
```

Hình 8: Trường hợp có 1 số 0

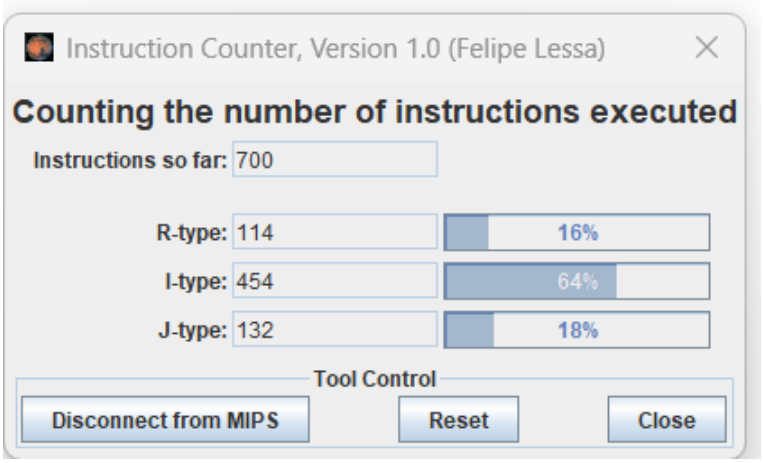


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{700*1}{10^9} = 700ns$

6.2.2 Testcase 2

```
Du lieu 1 = 0
Du lieu 2 = 0
0.0
-- program is finished running --
```

Hình 9: Trường hợp có 2 số 0

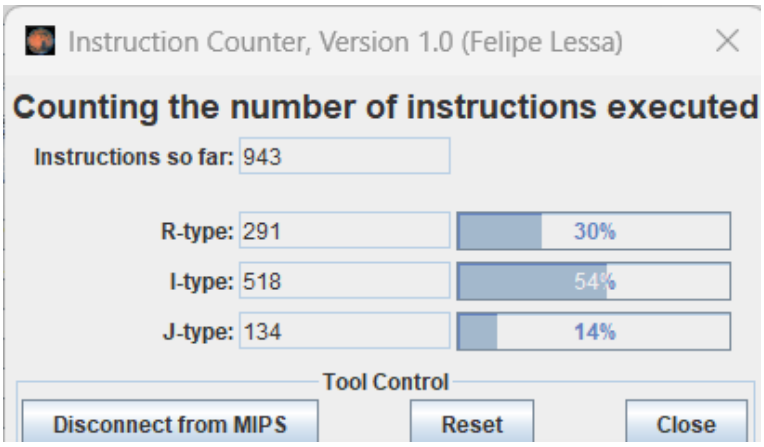


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{700*1}{10^9} = 700ns$

6.2.3 Testcase 3

```
Du lieu 1 = 445454
Du lieu 2 = -5
-2227270.0
-- program is finished running --
```

Hình 10: Trường hợp kết quả $<$ 32 bit (1 âm 1 dương)

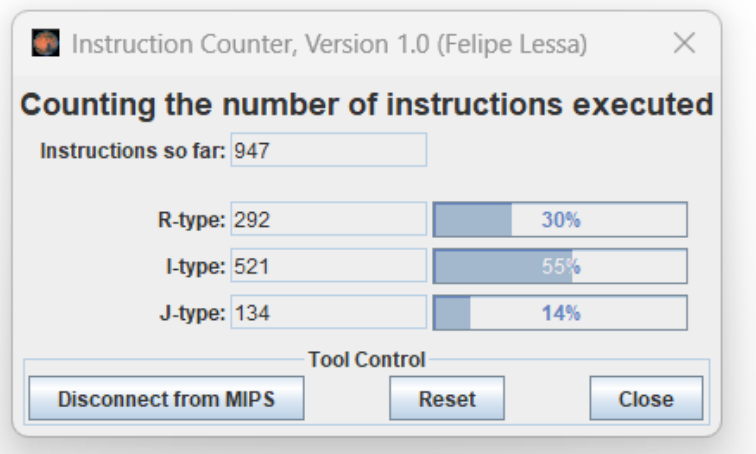


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{943*1}{10^9} = 943ns$

6.2.4 Testcase 4

```
Du lieu 1 = -445454
Du lieu 2 = -5
2227270.0
-- program is finished running --
```

Hình 11: Trường hợp kết quả $<$ 32 bit (2 âm)

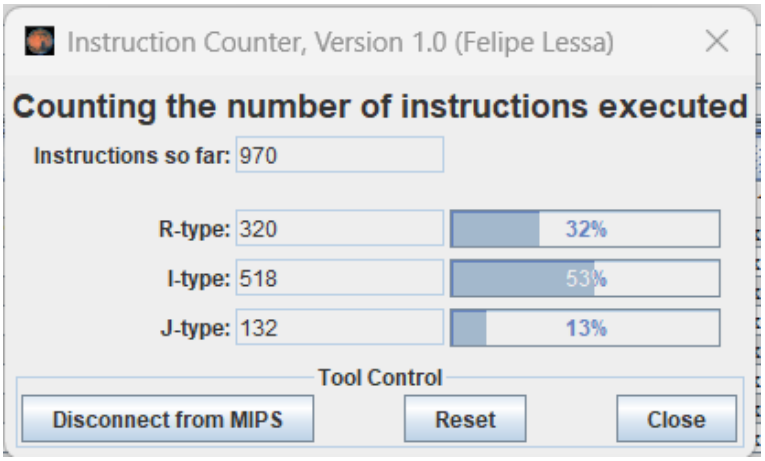


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{947*1}{10^9} = 947ns$

6.2.5 Testcase 5

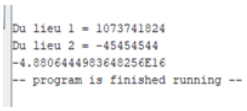
```
Du lieu 1 = 1515115
Du lieu 2 = 10
1.515115E7
```

Hình 12: Trường hợp kết quả $>$ 32 bit (2 dương)

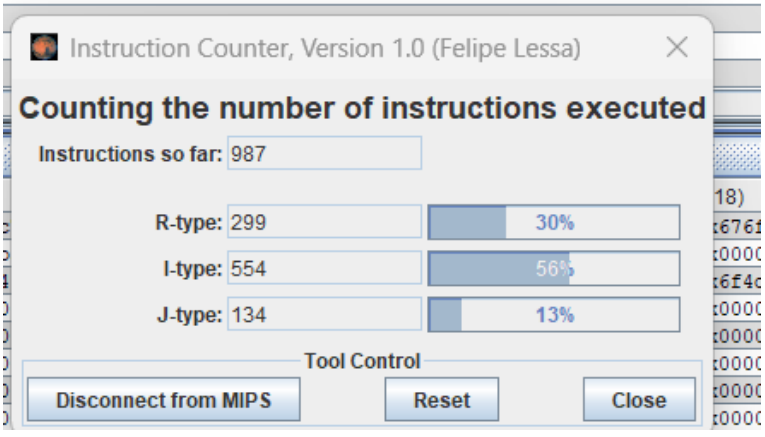


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{970*1}{10^9} = 970ns$

6.2.6 Testcase 6



Hình 13: Trường hợp kết quả > 32 bit (1 dương 1 âm)

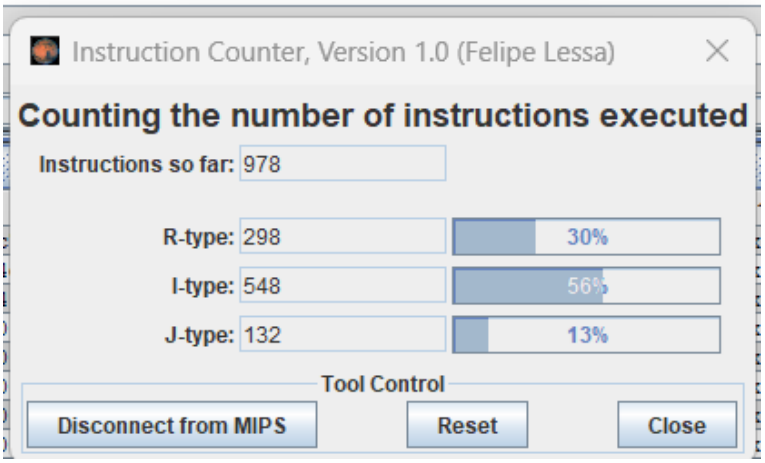


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{987*1}{10^9} = 987ns$

6.2.7 Testcase 7

```
Du lieu 1 = 1073741824
Du lieu 2 = 45454544
4.8806444983648256E16
-- program is finished running --
```

Hình 14: Trường hợp kết quả > 32 bit (1 dương 1 âm)

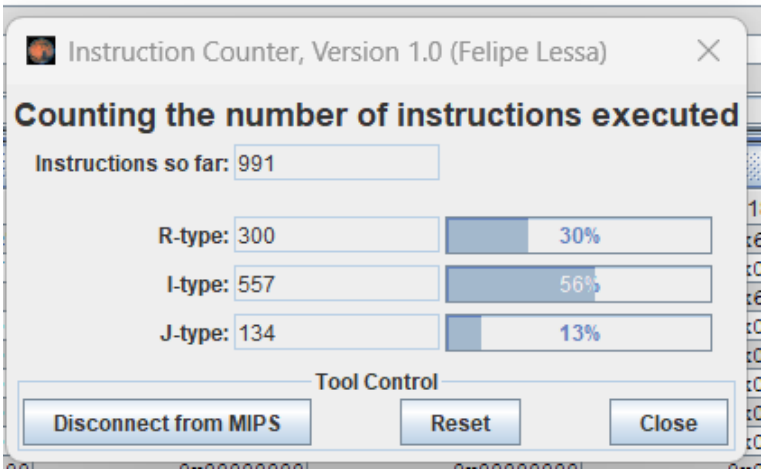


Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{978*1}{10^9} = 978ns$

6.2.8 Testcase 8

```
Du lieu 1 = -1073741824
Du lieu 2 = -100000000
1.073741824E17
-- program is finished running --
```

Hình 15: Trường hợp kết quả > 32 bit (2 âm)



Thời gian thực thi được tính là: $\frac{instructioncount*CPI}{Clockrate} = \frac{991*1}{10^9} = 991ns$

Tài liệu

[1] PGS.TS Phạm Quốc Cường, *Kiến trúc máy tính*, Nhà xuất bản Đại học Quốc gia TP Hồ Chí Minh, Năm 2021.

[2] David Patterson and John Hennessy, *Computer Organization and Design: the Hardware-SoftWare Interface*, Nhà xuất bản Morgan Kaufmann