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

Đề tài 6
Sắp xếp dãy số sử dụng thuật toán Merge Sort

GVHD: Võ Tấn Phương
Trần Thanh Bình

SV thực hiện: Lê Gia Huy – 1910202
Huỳnh Thành Đạt – 1910110
Nguyễn Ngô Thanh Trúc – 1910650

Tp. Hồ Chí Minh, Tháng 12/2020

Mục lục

1 Lý thuyết

Trong khoa học máy tính, sắp xếp trộn (merge sort) là một thuật toán sắp xếp để sắp xếp các danh sách (hoặc bất kỳ cấu trúc dữ liệu nào có thể truy cập tuần tự, v.d. luồng tập tin) theo một trật tự nào đó. Nó được xếp vào thể loại sắp xếp so sánh. Thuật toán này là một ví dụ tương đối điển hình của lối thuật toán chia để trị do John von Neumann đưa ra lần đầu năm 1945. Một thuật toán chi tiết được Goldstine và Neumann đưa ra năm 1948.

1.1 Ý tưởng thuật toán

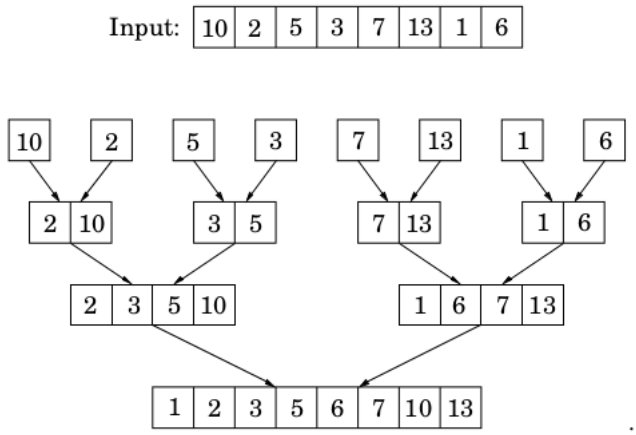
Giả sử có hai danh sách đã được sắp xếp a[1..m] và b[1..n]. Ta có thể trộn chúng lại thành một danh sách mới c[1..m+n] được sắp xếp theo cách sau: So sánh hai phần tử đứng đầu của hai danh sách, lấy phần tử nhỏ hơn cho vào danh sách mới. Tiếp tục như vậy cho tới khi một trong hai danh sách là rỗng. Khi một trong hai danh sách là rỗng ta lấy phần còn lại của danh sách kia cho vào cuối danh sách mới.

Ví dụ: Cho hai danh sách a=(1,3,7,9),b=(2,6), quá trình hòa nhập diễn ra như sau:

Danh sách a	Danh sách b	So sánh	Danh sách c
1,3,7,9	2,6	$1 < 2$	1
3,7,9	2,6	$2 < 3$	1,2
3,7,9	6	$3 < 6$	1,2,3
7,9	6	$6 < 7$	1,2, 6
7,9			1,2,3,6,7,9

1.2 Trộn 2 danh sách đã được sắp xến

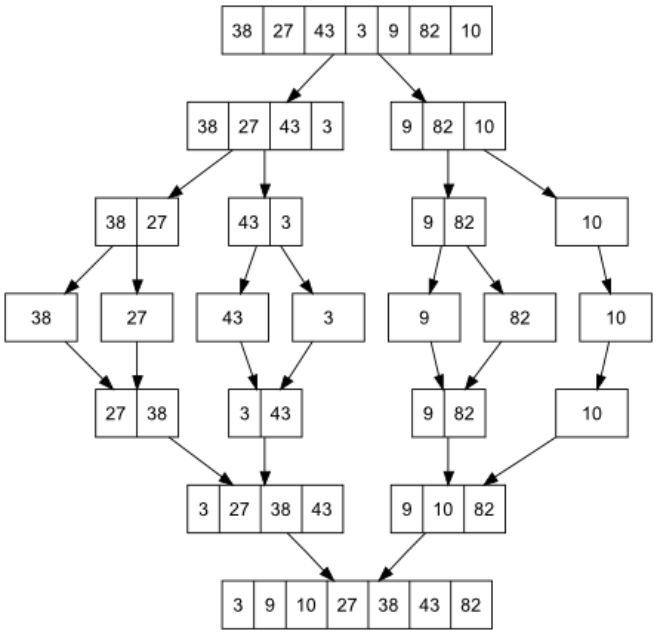
Giả sử trong danh sách a[1..n] có 2 danh sách con kề nhau $a[k_1..k_2]$ và $a[k_2+1..k_3]$ đã được sắp. Ta áp dụng cách trộn tương tự như trên để trộn hai danh sách con vào một danh sách tạm$T[k_1..k_3]$ rồi trả lại các giá trị của danh sách tạm T về danh sách A. Làm như vậy gọi là trộn tại chỗ.



Hình 1: *Trộn 2 mảng đã sắp xếp*

1.3 Merge Sort bằng giải thuật đệ quy

Với hàm đệ quy MergeSort, thủ tục đệ quy được gọi để chia mảng ban đầu thành 2 mảng con, tiếp tục chia mảng thành nửa cho đến khi mảng chỉ gồm 1 phần tử, mảng con với 1 phần tử được xem như đã sắp xếp xong. Tiến hành trộn 2 nửa mảng cho đến khi mảng ban đầu đã được trộn xong.



Hình 2: *Minh hoạ thuật toán Merget Sort*

```
void merge(int* left, int* middle, int* right) {
    int size_1 = middle - left + 1;
    int* half = new int[size_1];
    for (int i = 0; i < size_1; i++) half[i] = left[i];
    int* p = left, * p1 = half, * ep1 = half + size_1, * p2 = middle + 1;
    while (p1 != ep1 && p2 != right + 1)
    {
        if (*p1 <= *p2)
        {
            *p = *p1;
            p++;
            p1++;
        }
        else
        {
            *p = *p2;
            p++;
            p2++;
        }
    }
    while (p1 != ep1)
    {
        *p = *p1;
        p++;
        p1++;
    }
    while (p2 != right + 1)
    {
        *p = *p2;
        p++;
        p2++;
    }
}
```

Hình 3: *Hàm trộn*

```cpp
void mergeSort(int* start, int* end) {
    if (end - start <= 0) return;
    int size = end - start + 1;
    int mid = (size / 2 == (double)size / 2) ? size / 2 - 1 : size / 2;
    mergeSort(start, start + mid);
    mergeSort(start + mid + 1, end);
    merge(start, start + mid, end);
}
```

Hình 4: *Hàm sắp xếp*

2 Hiện thực code trong MIPS

Để có thể hiện thực được thuật toán Merge Sort, ta sẽ cần hiện thực hai hàm phụ như đã được minh hoạ trong đoạn code C++ đó là hàm đệ quy MERGE_SORT để sắp xếp 2 mảng con tương ứng tăng dần và MERGE để trộn 2 mảng con đó lại với nhau thành 1 mảng tăng dần

Trong hàm MERGE_SORT, các công việc cần thực hiện là:

- Xác định kích thước của mảng được truyền vào tương ứng với phép toán end – start + 4, lưu giá trị đó vào thanh ghi $t0 (Kích thước ở đây là kích thước địa chỉ)
- Nếu end – start = 0 tương ứng với việc mảng chỉ có 1 phần tử thì ta không cần thực hiện việc sắp xếp mảng này, thoát hàm ngay

```
sub  $t0, $a2, $a1 #end - start
blez $t0, RETURN
addi $t0, $t0, 4 #size
```

- Ngược lại ta sẽ thực hiện thao tác đệ quy trên hai mảng con của mảng hiện tại, mảng con thứ nhất từ phần tử đầu đến phần tử chính giữa, mảng con thứ hai là phần còn lại của mảng hiện tại. Khi đó, phần tử chính giữa được xác định như sau: Index của phần tử ở giữa sẽ tương ứng với (end – start + 4) / 2 / 4 = size / 8. Tuy nhiên nếu mảng có số phần tử chẵn thì vị trí này sẽ lớn hơn vị trí ta mong muốn 1 phần tử (4 ô nhớ), vì vậy cần xác định lại chính xác vị trí ở giữ nếu số phần tử là số chẵn, sau đó lưu giá trị này vào thanh ghi $t3

```
addi $t1, $0, 8
divu $t0, $t1 #size / 8
mfhi $t2 #remainder
mflo $t3 #mid
bne $t2, 0, CONTINUE #t0 / 4 is not even
addi $t3, $t3, -1 #t0 /4 is even -> mid - 1
```

- Sau khi xác định chính xác vị trí index, ta sẽ cần xác định con trỏ đến vị trí đó. Vị trí này tương ứng với 4 nhân index của phần tử chính giữa. Lưu các giá trị cần tái sử dụng như: địa chỉ trả về (thanh ghi $ra), địa chỉ đầu (thanh ghi $a1), địa chỉ cuối (thanh ghi $a2), kích thước của mảng hiện tại (thanh ghi $t0)... vào stack

```
mul $t3, $t3, 4 #mid = mid * 4
addi $sp, $sp, -28
sw $ra, 0($sp)  #return address
sw $a1, 4($sp)  #start
sw $a2, 8($sp)  #end
sw $t0, 12($sp) #size
sw $t1, 16($sp) #8
sw $t2, 20($sp) #remainder
sw $t3, 24($sp) #mid_index * 4
```

- Sau khi đã lưu các giá trị cần tái sử dụng vào stack, ta sẽ cần thay đổi tham số $a2 (Con trỏ tới phần tử cuối của mảng truyền vào hàm MERGE_SORT) thành địa chỉ của phần tử chính giữa sau đó jump and link tới hàm MERGE_SORT của mảng con thứ nhất (từ phần tử đầu đến phần tử giữa của mảng)

```
add $a2, $a1, $t3
jal MERGE_SORT
```

- Sau khi kết thúc hàm đệ quy của mảng con thứ nhất, ta sẽ load cái giá trị đã lưu vào stack ra để tái sử dụng, đồng thời thay đổi các tham số $a1 tới phần tử sau phần tử chính giữa, $a2 tới phần tử cuối của mảng và jump and link tới hàm MERGE_SORT của mảng con còn lại:

```
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a2, 8($sp)
lw $t0, 12($sp)
lw $t1, 16($sp)
lw $t2, 20($sp)
lw $t3, 24($sp)
add $a1, $a1, $t3 #a1 = a1 + mid
addi $a1, $a1, 4
jal MERGE_SORT  #mergeSort(start + mid + 1, end);
```

- Sau khi kết thúc 2 lần đệ quy trên 2 mảng con tương ứng, ta sẽ được 2 mảng con, mỗi mảng đã được sắp xếp tăng dần. Thực hiện việc load dữ liệu đã lưu từ stack ra để tái sử dụng, đồng thời thay đổi các tham số tương ứng để chuẩn bị cho công việc việc trộn 2 mảng con đã được sắp xếp này lại. Ta thay đổi giá trị thanh ghi $a3 thành địa chỉ của phần tử cuối trong mảng, $a2 thành địa chỉ phần tử chính giữa trong mảng, sau đó jump and link tới hàm MERGE để thực hiện thao tác trộn 2 mảng con

```
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a2, 8($sp)
lw $t0, 12($sp)
lw $t1, 16($sp)
lw $t2, 20($sp)
lw $t3, 24($sp)
add $a3, $0, $a2 #end
add $a2, $a1, $t3 #start + mid

jal MERGE
```

Trong hàm MERGE, các công việc cần thực hiện là

- Đầu tiên cần thực hiện là xác định kích thước (Kích thước địa chỉ) của mảng con thứ nhất để thực hiện copy dữ liệu từ mảng con này qua mảng phụ temp đã được đề cập từ trước. Kích thước của mảng thứ nhất sẽ là middle – start + 4. Sau đó, ta copy dữ liệu từ mảng con thứ nhất sang mảng tạm và khởi tạo các con trỏ đến các mảng tương ứng:

  $t2 lưu địa chỉ của phần tử đầu tiên của mảng con thứ nhất ($a1)

  $t3 lưu địa chỉ của phần tử đầu tiên của mảng tạm (temp)

  $t5 lưu địa chỉ kết thúc của mảng tạm (temp + size)

  $t6 lưu địa chỉ bắt đầu của mảng con thứ hai (middle + 1)

  $t4 lưu địa chỉ kết thúc của mảng con thứ hai (right + 1)

```
MERGE:
        sub $t0, $a2, $a1 #middle - left
        addi $t0, $t0, 4 #size_1
        addi $t1, $0, 0 #i
        add $t2, $a1, $0 #left
        la $t3, temp #half
FOR_LOOP:
        lw $t4, 0($t2)
        sw $t4, 0($t3)
        addi $t1, $t1, 4
        addi $t2, $t2, 4
        addi $t3, $t3, 4
        slt $at, $t1, $t0
        bne $at, $zero, FOR_LOOP #i < size_1
        add $t2, $a1, $0 #p = left
        la $t3, temp #p1 = half
        add $t5, $t3, $t0 #ep1 = half + size_1
        add $t6, $a2, 4 #p2 = mid + 1
        addi $t4, $a3, 4 #right + 1
```

- Sau đó, ta tiến hành trộn 2 mảng con lại với nhau. Ý tưởng của thuật toán này là: so sánh 2 phần tử ứng với con trỏ đang trỏ tới mảng tạm (Tương ứng với mảng con thứ nhất đã được copy qua) và con trỏ đang trỏ tới mảng con thứ 2. Phần tử nào bé hơn sẽ được đưa vào mảng chính trước, với mỗi phần tử của mảng con tương ứng sau khi được đưa vào mảng chính thì ta sẽ dịch con trỏ tới phần tử đó qua phần tử kế tiếp (Tương ứng với việc bỏ phần tử đó ra khỏi mảng) để tiếp tục quá trình trộn cho đến khi hết 1 trong 2 mảng con. Phần còn lại của mảng con còn thừa lại sẽ được nhập vào sau mảng chính hiện tại do không còn phần tử nào bé hơn nó (các phần tử bé hơn đã được đưa vào mảng chính ở mảng con đã hết). Để đơn giản hoá code thì ta sẽ sử dụng 2 vòng lặp WHILE với điều kiện là con trỏ đầu khác con trỏ cuối, khi đó mảng con nào còn dư sẽ thực hiện vòng lặp để đưa các phần tử còn dư của mảng con vào mảng chính

```
WHILE_1:
        beq $t5, $t3, WHILE_2 #p1 != ep1
        beq $t4, $t6, WHILE_2 #p2 != right + 1
        lw $t7, 0($t3) #*p1
        lw $t8, 0($t6) #*p2
        slt $at, $t8, $t7
        bne $at, $zero, IF_2
IF_1:
        sw $t7, 0($t2) #*p = *p1
        addi $t2, $t2, 4#p++
        addi $t3, $t3, 4#p1++
        j WHILE_1
IF_2:
        sw $t8, 0($t2) #*p = *p2
        addi $t2, $t2, 4#p++
        addi $t6, $t6, 4#p2++
        j WHILE_1
WHILE_2:
        beq $t5, $t3, WHILE_3 #p1 != ep1
        lw $t7, 0($t3) #t7 = *p1
        sw $t7, 0($t2) #*p = *p1
        addi $t2, $t2, 4#p++
        addi $t3, $t3, 4#p1++
        j WHILE_2
WHILE_3:
        beq $t4, $t6, PRINT #p2 != right + 1
        lw $t8, 0($t6)#t8 = *p2
        sw $t8, 0($t2) #*p = *p2
        addi $t2, $t2, 4#p++
        addi $t6, $t6, 4#p2++
        j WHILE_3
```

- Sau khi trộn, ta đã được 1 mảng đã được sắp xếp tăng dần, ta tiến hành in mảng đó ra bằng vòng lặp để kiểm tra đồng thời jump register trở lại hàm MERGE_SORT

```
PRINT:
        sub $t9, $a3, $a1#end - start
        addi $t9, $t9, 4#end - start + 1
        addi $t1, $0, 0 #i
        add $t2, $a1, $0
```

```
FOR_PRINT:
        lw $a0, 0($t2)
        addi $v0, $0, 1
        syscall
        addi $t1, $t1, 4
        addi $t2, $t2, 4
        slt $at, $t1, $t9
        beq $at, $zero, PRINT_ENDL
        la $a0, comma
        addi $v0, $0, 4
        syscall
        j FOR_PRINT
PRINT_ENDL:
        lb $a0, endl
        addi $v0, $0, 11
        syscall
        jr $ra
```

- Sau khi jump register trở lại hàm MERGE_SORT, ta tiến hành pop các giá trị đã lưu trong stack ra, đặc biệt là giá trị thanh ghi $ra lưu địa chỉ trở về chương trình chính, sau đó phục hồi stack về trạng thái ban đầu. Sau đó tiến hành Jump register để trở về đoạn chương trình chính

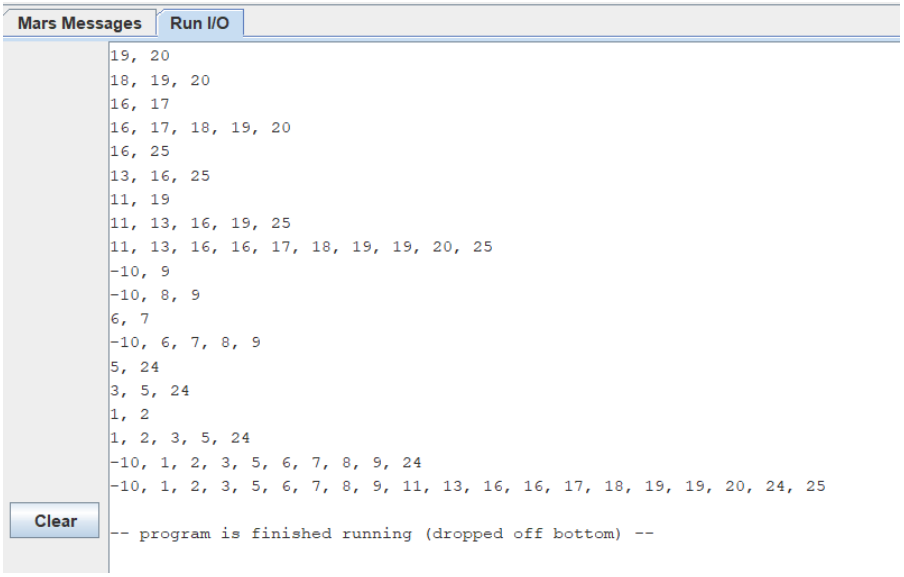
```
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a2, 8($sp)
lw $t0, 12($sp)
lw $t1, 16($sp)
lw $t2, 20($sp)
lw $t3, 24($sp)
addi $sp, $sp, 28
j RETURN


RETURN:
        jr $ra
```

- Sau khi trở về chương trình chính, công việc đến đây đã xong, mảng chính đã được sắp xếp nên lúc này ta kết thúc chương trình

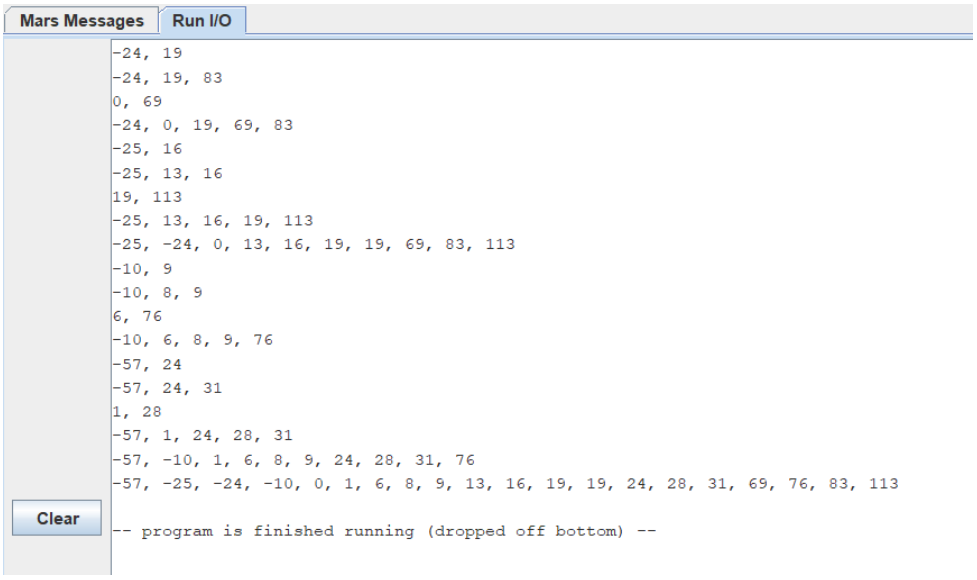
```
j EXIT
```

3 Một số test case và kết quả thực thi

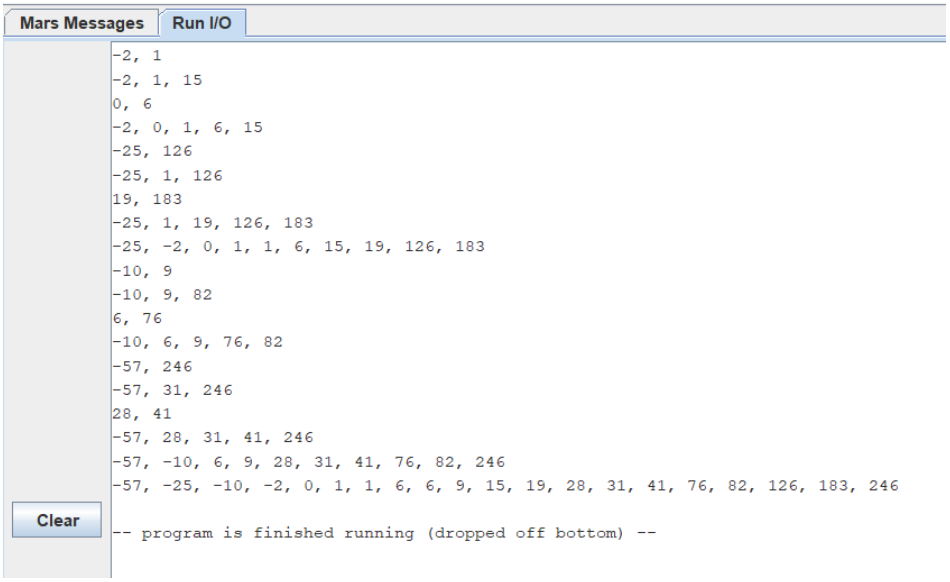
Test case 1: 20, 19, 18, 17, 16, 25, 16, 13, 19, 11, -10, 9, 8, 7, 6, 5, 24, 3, 2, 1



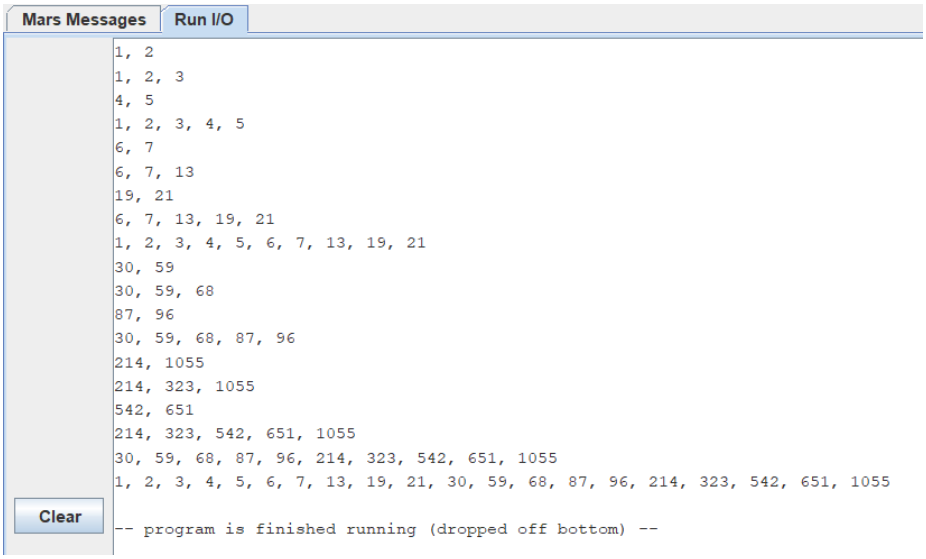
Test case 2: -24, 19, 83, 0, 69, -25, 16, 13, 19, 113, -10, 9, 8, 76, 6, -57, 24, 31, 28, 1



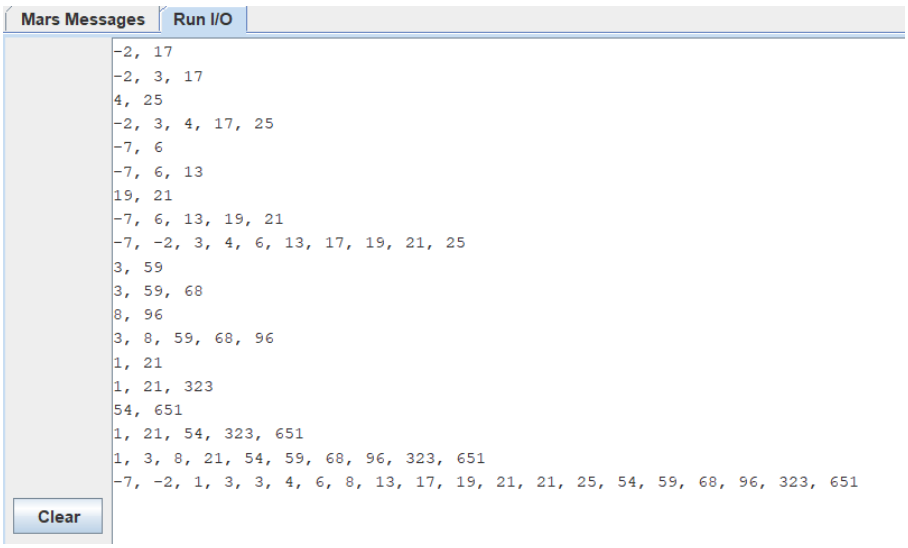
Test case 3: -2, 1, 15, 0, 6, -25, 126, 1, 19, 183, -10, 9, 82, 76, 6, -57, 246, 31, 28, 41



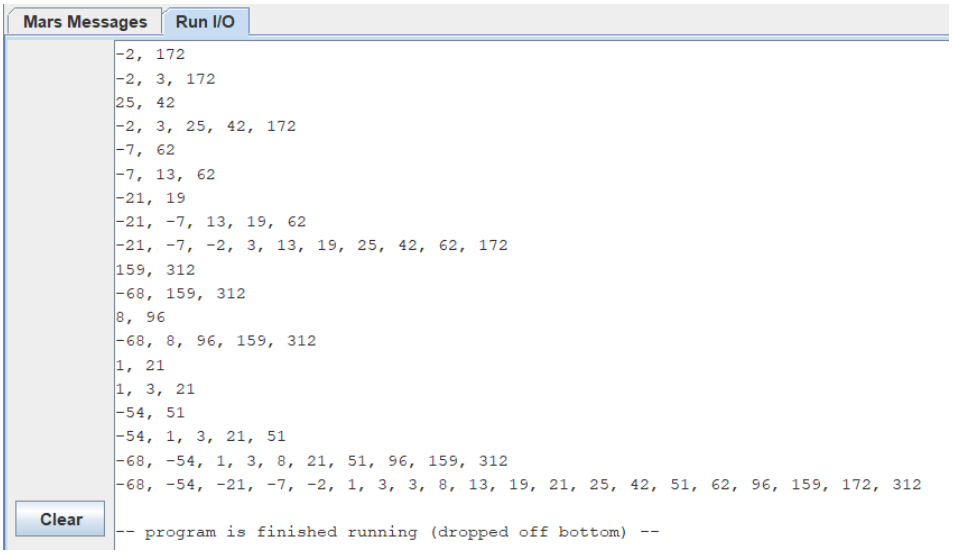
Test case 4: 1, 2, 3, 4, 5, 6, 7, 13, 19, 21, 30, 59, 68, 87, 96, 1055, 214, 323, 542, 651



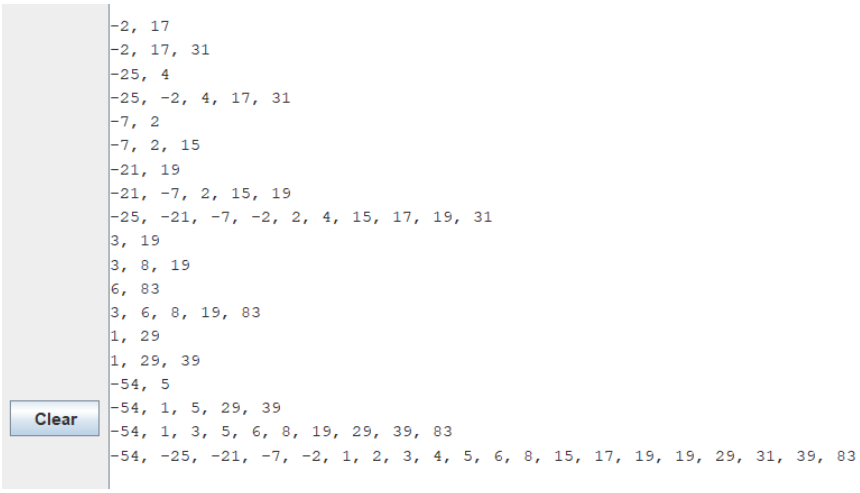
Test case 5: 17, -2, 3, 4, 25, 6, -7, 13, 19, 21, 3, 59, 68, 8, 96, 1, 21, 323, 54, 651



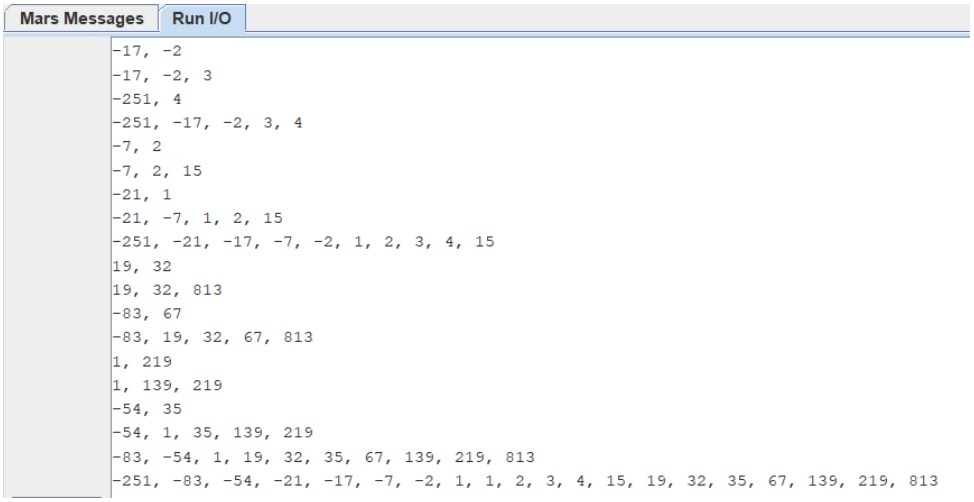
Test case 6: 172, -2, 3, 42, 25, 62, -7, 13, 19, -21, 312, 159, -68, 8, 96, 1, 21, 3, -54, 51



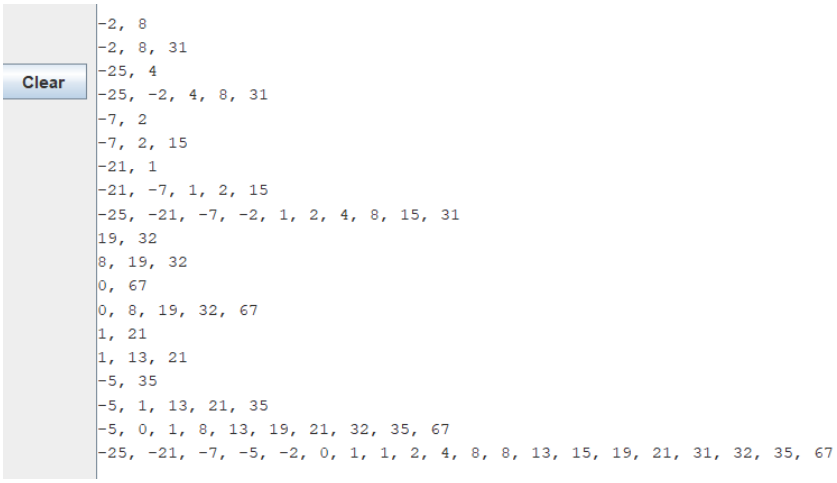
Test case 7: 17, -2, 31, 4, -25, 2, -7, 15, 19, -21, 3, 19, 8, 83, 6, 1, 29, 39, -54, 5



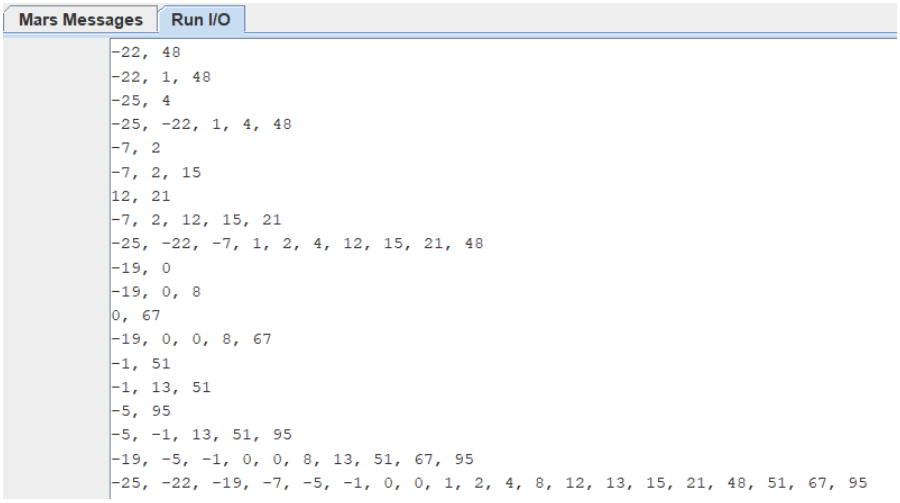
Test case 8: -17, -2, 3, 4, -251, 2, -7, 15, 1, -21, 32, 19, 813, -83, 67, 1, 219, 139, -54, 35



Test case 9: 8, -2, 31, 4, -25, 2, -7, 15, 1, -21, 32, 19, 8, 0, 67, 1, 21, 13, -5, 35



Test case 10: 48, -22, 1, 4, -25, 2, -7, 15, 12, 21, 0, -19, 8, 0, 67, -1, 51, 13, -5, 95

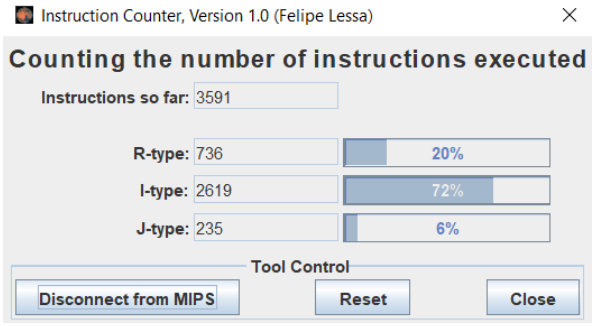


4 Thống kê các câu lệnh và thời gian chạy chương trình

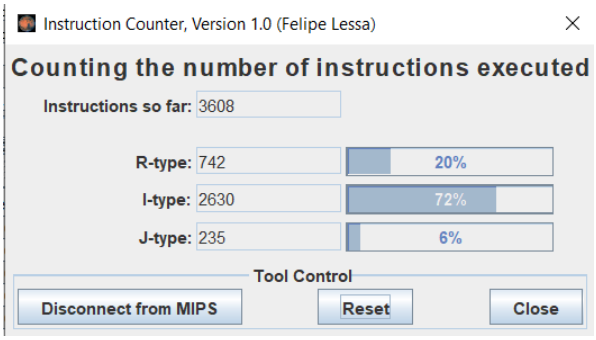
4.1 Thống kê các câu lệnh

Để thống kê các câu lệnh đã thực thi trong chương trình ta sử dụng công cụ hỗ trợ Instruction Counter trong MARS.

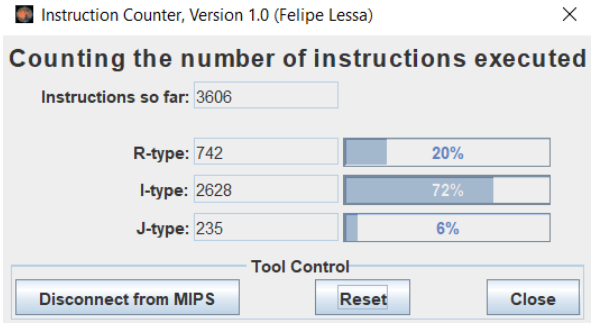
Dưới đây là kết quả thống kê các câu lệnh đã thực thi với từng test case ở trên



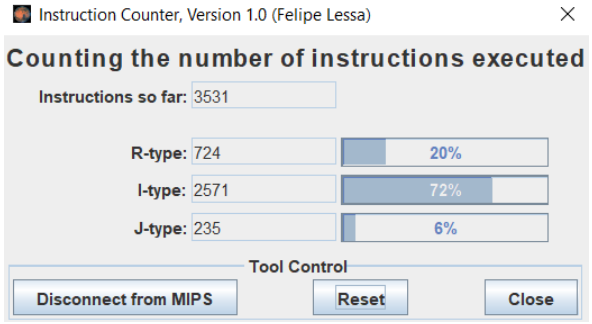
Hình 5: *Test case 1: 20, 19, 18, 17, 16, 25, 16, 13, 19, 11, -10, 9, 8, 7, 6, 5, 24, 3, 2, 1*



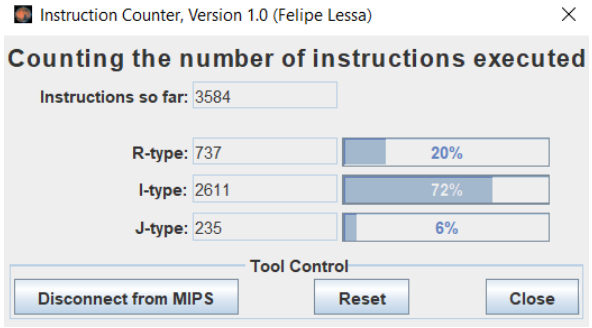
Hình 6: *Test case 2: -24, 19, 83, 0, 69, -25, 16, 13, 19, 113, -10, 9, 8, 76, 6, -57, 24, 31, 28, 1*



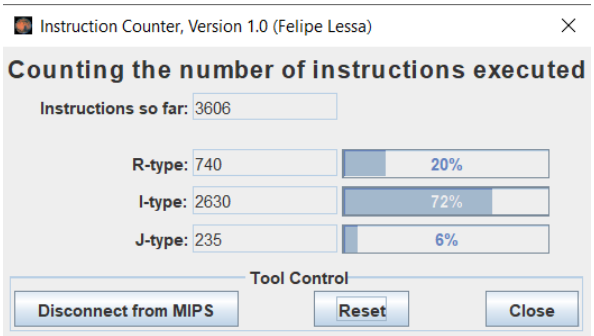
Hình 7: *Test case 3: -2, 1, 15, 0, 6, -25, 126, 1, 19, 183, -10, 9, 82, 76, 6, -57, 246, 31, 28, 41*



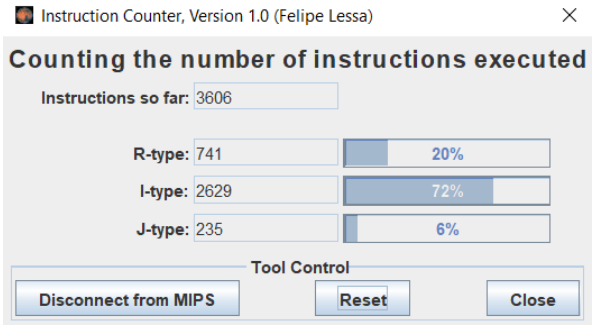
Hình 8: *Test case 4: 1, 2, 3, 4, 5, 6, 7, 13, 19, 21, 30, 59, 68, 87, 96, 1055, 214, 323, 542, 651*



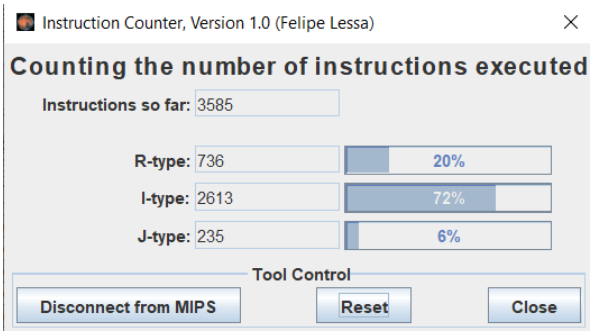
Hình 9: *Test case 5: 17, -2, 3, 4, 25, 6, -7, 13, 19, 21, 3, 59, 68, 8, 96, 1, 21, 323, 54, 651*



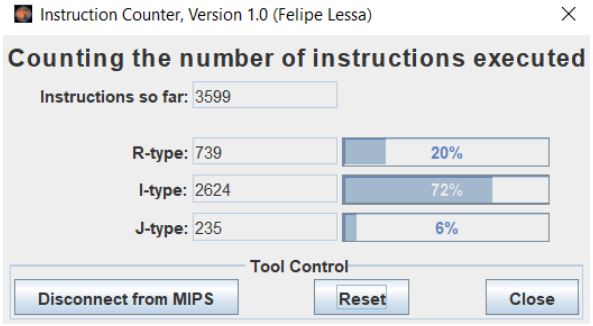
Hình 10: *Test case 6: 172, -2, 3, 42, 25, 62, -7, 13, 19, -21, 312, 159, -68, 8, 96, 1, 21, 3, -54, 51*



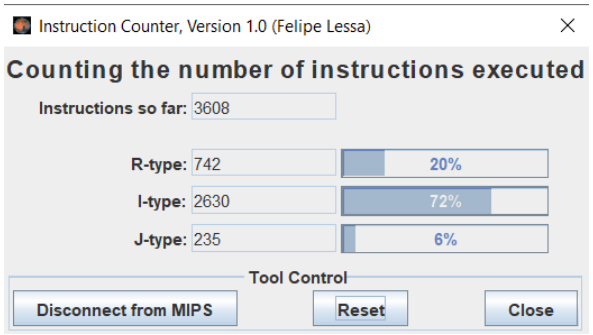
Hình 11: *Test case 7: 17, -2, 31, 4, -25, 2, -7, 15, 19, -21, 3, 19, 8, 83, 6, 1, 29, 39, -54, 5*



Hình 12: *Test case 8: -17, -2, 3, 4, -251, 2, -7, 15, 1, -21, 32, 19, 813, -83, 67, 1, 219, 139, -54, 35*



Hình 13: *Test case 9: 8, -2, 31, 4, -25, 2, -7, 15, 1, -21, 32, 19, 8, 0, 67, 1, 21, 13, -5, 35*



Hình 14: *Test case 10: 48, -22, 1, 4, -25, 2, -7, 15, 12, 21, 0, -19, 8, 0, 67, -1, 51, 13, -5, 95*

4.2 Tính thời gian thực thi chương trình

Tthời gian thực thi của chương trình được tính bằng $t = (CPI * n)/f$, với n là số lệnh đã được thực thi trong chương trình và f là tần số của máy.
Mỗi lệnh trong chương trình có CPI = 1. Vì vậy áp dụng công thức trên ta tìm được thời gian thực thi của chương trình:

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Số Lệnh	3591	3608	3606	3531	3584	3606	3606	3585	3599	3608
Thời gian(ns)	1.7955	1.804	1.803	1.7655	1.792	1.803	1.803	1.7925	1.7995	1.804

Tài liệu

[1] Phạm Quốc Cường (20019).*Kiến trúc máy tínhi*, NXB: Đại học quốc gia TP.HCM

[2] Nguyễn Đức Nghĩa (2020). *Cấu trúc dữ liệu và thuật toán*, NXB: Bách Khoa Hà Nội

[3] https://www.wikipedia.org/