

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



KIẾN TRÚC MÁY TÍNH (CO2007)

Báo cáo

Bài tập lớn Kiến trúc máy tính

GVHD: Võ Tấn Phương

Trần Thanh Bình

Sinh viên: Nguyễn Diệu Ái - 1910032

Võ Văn Đăng Khoa - 1910276

Nguyễn Vương Long-1911520

THÀNH PHỐ HỒ CHÍ MINH, 2020



Mục lục

1 Danh sách phân công nhiệm vụ	2
2 Cơ sở lý thuyết	2
2.1 Yêu cầu của đề	2
2.2 Giới thiệu về Merge Sort.....	2
3 Thực thi chương trình chính trên MIPS	6
3.1 Test case 1	6
3.2 Test case 2.....	6
3.3 Test case 3	7
3.4 Test case 4.....	7
3.5 Test case 5.....	8
3.6 Test case 6.....	8
3.7 Test case 7.....	9
3.8 Test case 8.....	9
3.9 Test case 9.....	10
3.10 Test case 10.....	10
4 Tính thời gian thực thi.....	11
4.1 Thống kê các loại lệnh	11
4.2 Tính thời gian	12
References	14



1 Danh sách phân công nhiệm vụ

STT	Họ và tên	MSSV	Nhiệm vụ	Đóng góp
1	Nguyễn Diệu Ái	1910032	Làm báo cáo và tính toán các yêu cầu	33,33%
2	Võ Văn Đăng Khoa	1910276	Lên ý tưởng viết code	33,33%
3	Nguyễn Vương Long	1911520	Hoàn thiện code	33,33%

2 Cơ sở lý thuyết

2.1 Yêu cầu của đề

Sắp xếp một chuỗi 20 số nguyên theo thứ tự tăng dần bằng giải thuật Merge Sort trên hợp ngữ Assembly MIPS. Xuất ra màn hình từng bước của quá trình.

Thống kê số lệnh, loại lệnh đã sử dụng.

Tính và trình bày cách tính thời gian chạy của chương trình trên máy tính MIPS có tần số 2GHz. Biết CPI là 1.

2.2 Giới thiệu về Merge Sort

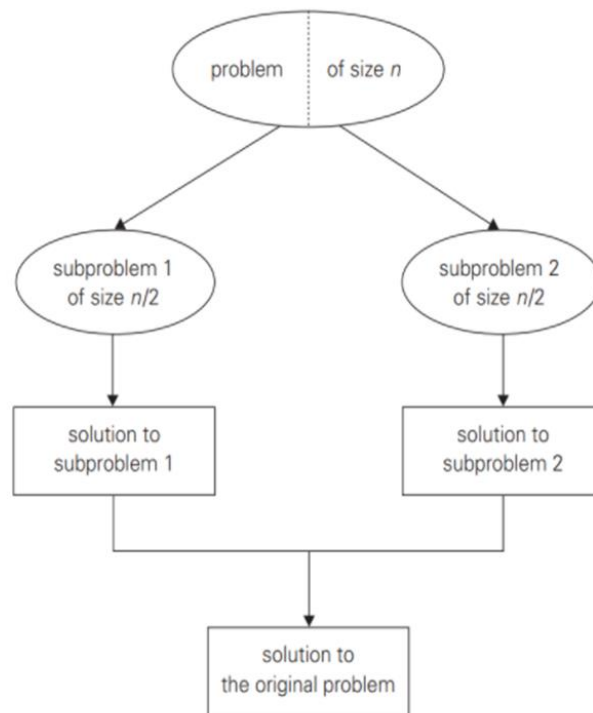
Thuật toán sắp xếp merge sort là một trong những thuật toán có độ phức tạp ở mức trung bình và cùng sử dụng phương pháp chia để trị.

Chia mảng lớn thành những mảng con nhỏ hơn bằng cách chia đôi mảng lớn và tiếp tục chia đôi các mảng con cho đến khi mảng con nhỏ nhất chỉ còn 1 phần tử.

So sánh 2 mảng con có cùng mảng cơ sở (khi chia đôi mảng lớn thành 2 mảng con thì mảng lớn đó gọi là mảng cơ sở của 2 mảng con đó).

Khi so sánh chúng vừa sắp xếp vừa ghép 2 mảng con đó lại thành mảng cơ sở, tiếp tục so sánh và ghép các mảng con lại đến khi còn lại mảng duy nhất, đó là mảng đã được sắp xếp.

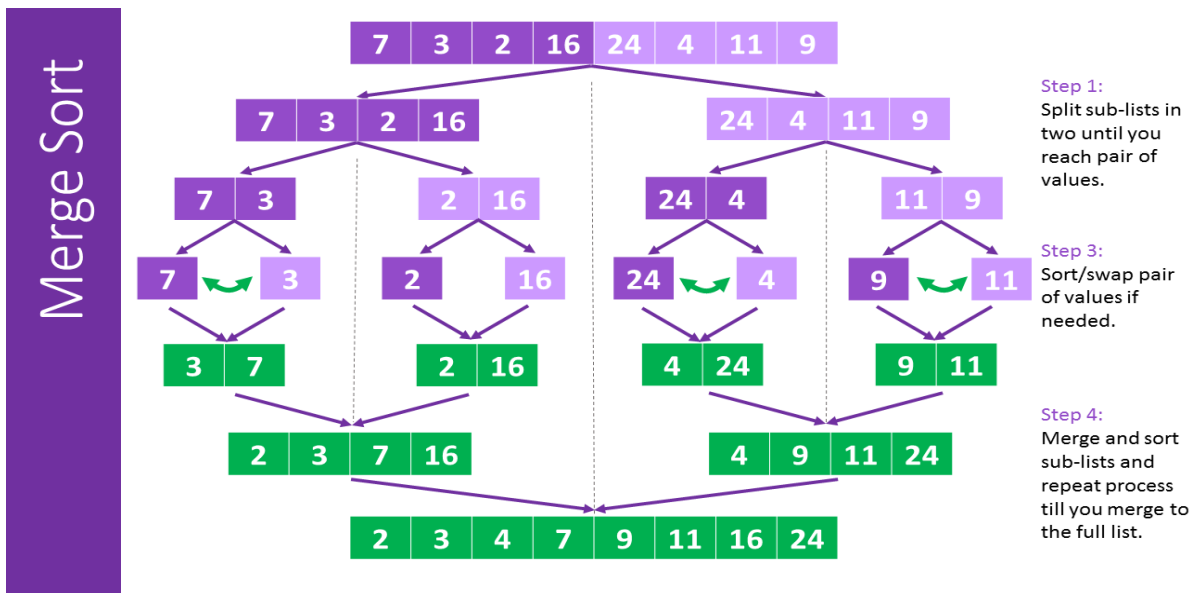
Hàm merge () được sử dụng để gộp hai nửa mảng. Hàm merge (arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành một mảng sắp xếp, các nửa mảng là arr [l...m] và arr [m+1...r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.



Hãy xem pseudo-code dưới đây:

```
procedure mergesort(l,r: integer);
var i, j, k, m: integer;
begin
  if r>l then
    begin
      m:=(r+l)shr 1;
      mergesort(l,m); mergesort(m+1,r);
      for i:= m downto l do b[i]:=a[i];
      for j:=m+1 to r do b[r+m+1-j]:=a[j];
      for k:=l to r do
        if b[i] < b[j] then
          begin a[k]:=b[i]; i:=i+1 end
        else
          begin a[k]:=b[j]; j:=j-1 end;
      end;
    end;
end;
```

Hình ảnh dưới đây sẽ hiện thị toàn bộ sơ đồ tiến trình của thuật toán Merge Sort cho mảng {7,3,2,16,24,4,11,9}. Nếu nhìn kỹ hơn vào sơ đồ này, chúng ta có thể thấy mảng ban đầu được lặp lại hành động chia cho tới khi kích thước sau khi chia là 1. Khi đó, tiến trình gộp sẽ bắt đầu thực hiện gộp lại các mảng này cho tới khi hoàn thành và chỉ còn một mảng đã sắp xếp.



Cách hàm merge hoạt động khi gộp hai mảng con: với trường hợp khi 2 mảng con chỉ có một phần tử, ta chỉ việc xem phần tử nào nhỏ hơn và đẩy lên đầu, phần tử còn lại đặt phía sau. Do vậy, các mảng con cần gộp lại có tính chất luôn được sắp xếp tăng dần.

```

0
1 Giả sử ta có 2 mảng con lần lượt là:
2 arr1 = [1 9 10 10] , n1 = 4 // chiều dài của mảng con
3 arr2 = [3 5 7 9], n2 = 4
4 sort_arr = [] // Mảng lưu lại tiến trình gộp
5 Khởi tạo i = 0, j = 0 tương ứng là chỉ số bắt đầu của arr1 và arr2
6 Xét thấy arr1[i] < arr2[j] => chèn arr1[i] vào cuối mảng sort_arr, tăng i lên 1 đơn vị
7 => sort_arr = [1], i = 1
8 Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
9 => sort_arr = [1, 3], i = 1, j = 1
10 Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
11 => sort_arr = [1, 3, 5], i = 1, j = 2
12 Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
13 => sort_arr = [1, 3, 5, 7], i = 1, j = 3
14 Xét thấy arr1[i] = arr2[j] => chèn arr1[i] hoặc arr2[j] vào cuối mảng sort_arr
15 Giả sử tôi chọn arr1, tăng i lên 1 đơn vị
16 => sort_arr = [1, 3, 5, 7, 9], i = 2, j = 3
17 Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
18 => sort_arr = [1, 3, 5, 7, 9, 9], i = 1, j = 4
19 Do j >= n2, tiếp tục tăng i chừng nào i < n1 thì thêm phần tử ở arr1[i] vào sort_arr.
20 Sau cùng ta được mảng đã sắp xếp là sort_arr = [1, 3, 5, 7, 9, 9, 10, 10]
21

```

Minh họa thuật toán sắp xếp Merge Sort sử dụng C++:

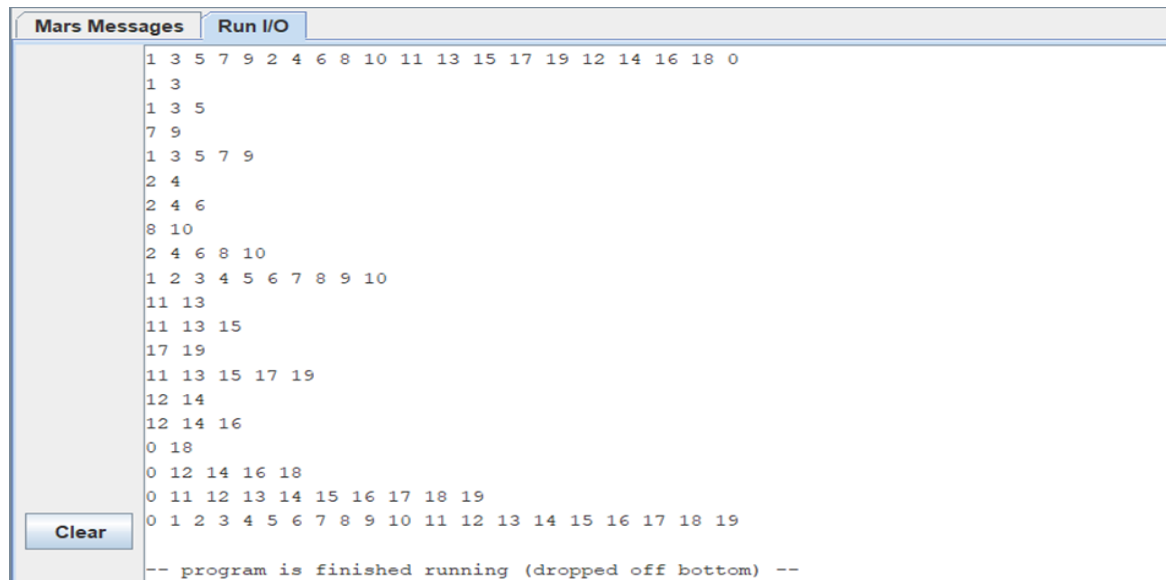
```
void merge_sort(vector<int> &v) {  
    merge_sort(v, 0, v.size() - 1);  
}  
  
void merge_sort(vector<int> &v, int l, int r) {  
    if (l ≥ r) return;  
    int m = l + (r - l) / 2;  
    merge_sort(v, l, m);  
    merge_sort(v, m + 1, r);  
    merge(v, l, m, r);  
}  
  
void merge(vector<int> &v, int l, int m, int r) {  
    vector<int> res(v.begin() + l, v.begin() + r + 1);  
    int i1 = l;  
    int i2 = m + 1;  
    int i = l;  
  
    while (i1 ≤ m && i2 ≤ r) {  
        int v1 = res[i1 - l];  
        int v2 = res[i2 - l];  
  
        if (v1 < v2) {  
            v[i++] = v1;  
            ++i1;  
        } else {  
            v[i++] = v2;  
            ++i2;  
        }  
    }  
  
    while (i1 ≤ m) v[i++] = res[i1++ - l];  
    while (i2 ≤ r) v[i++] = res[i2++ - l];  
}
```

3 Thực thi chương trình chính trên MIPS

3.1 Test case 1

Cho dãy số: {1 3 5 7 9 2 4 6 8 10 11 13 15 17 19 12 14 16 18 0}

Output:

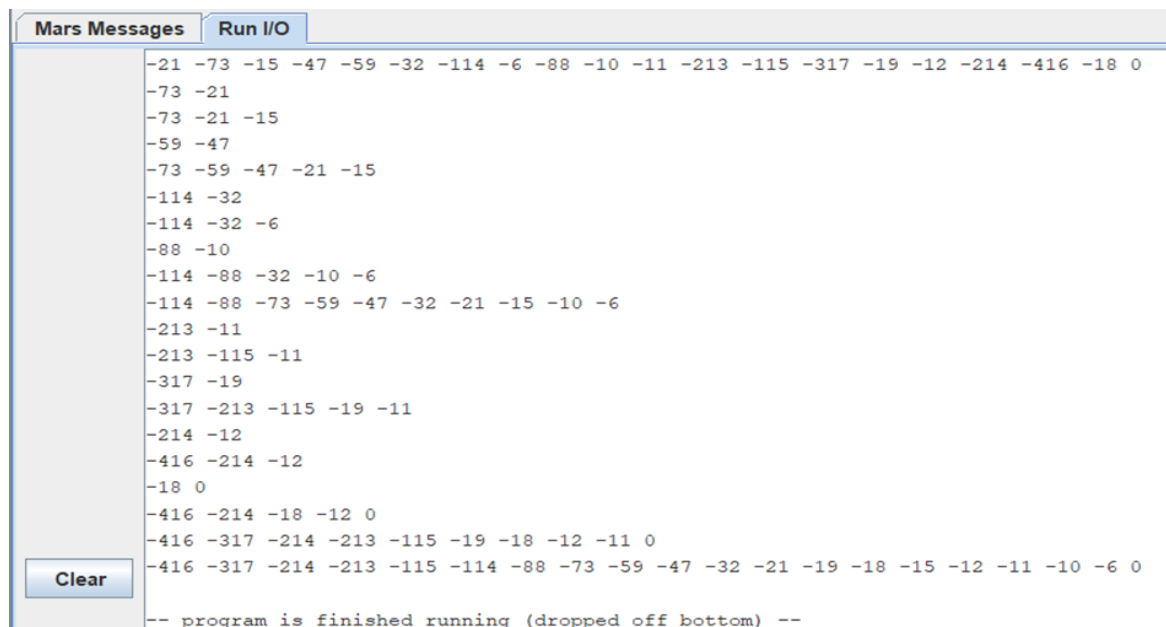


```
Mars Messages Run I/O
1 3 5 7 9 2 4 6 8 10 11 13 15 17 19 12 14 16 18 0
1 3
1 3 5
7 9
1 3 5 7 9
2 4
2 4 6
8 10
2 4 6 8 10
1 2 3 4 5 6 7 8 9 10
11 13
11 13 15
17 19
11 13 15 17 19
12 14
12 14 16
0 18
0 12 14 16 18
0 11 12 13 14 15 16 17 18 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
-- program is finished running (dropped off bottom) --
```

3.2 Test case 2

Cho dãy số: { -21 -73 -15 -47 -59 -32 -114 -6 -88 -10 -11 -213 -115 -317 -19
-12 -214 -416 -18 0}

Output:

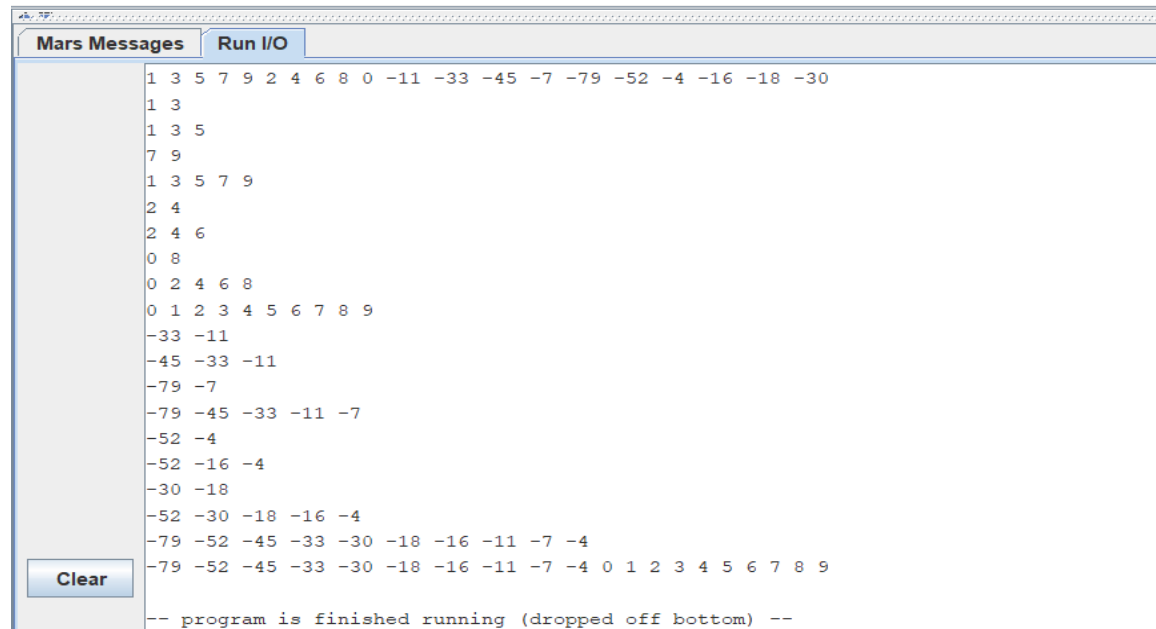


```
Mars Messages Run I/O
-21 -73 -15 -47 -59 -32 -114 -6 -88 -10 -11 -213 -115 -317 -19 -12 -214 -416 -18 0
-73 -21
-73 -21 -15
-59 -47
-73 -59 -47 -21 -15
-114 -32
-114 -32 -6
-88 -10
-114 -88 -32 -10 -6
-114 -88 -73 -59 -47 -32 -21 -15 -10 -6
-213 -11
-213 -115 -11
-317 -19
-317 -213 -115 -19 -11
-214 -12
-416 -214 -12
-18 0
-416 -214 -18 -12 0
-416 -317 -214 -213 -115 -19 -18 -12 -11 0
-416 -317 -214 -213 -115 -114 -88 -73 -59 -47 -32 -21 -19 -18 -15 -12 -11 -10 -6 0
-- program is finished running (dropped off bottom) --
```

3.3 Test case 3

Cho dãy số: {1 3 5 7 9 2 4 6 8 0 -11 -33 -45 -7 -79 -52 -4 -16 -18 -30}

Output:

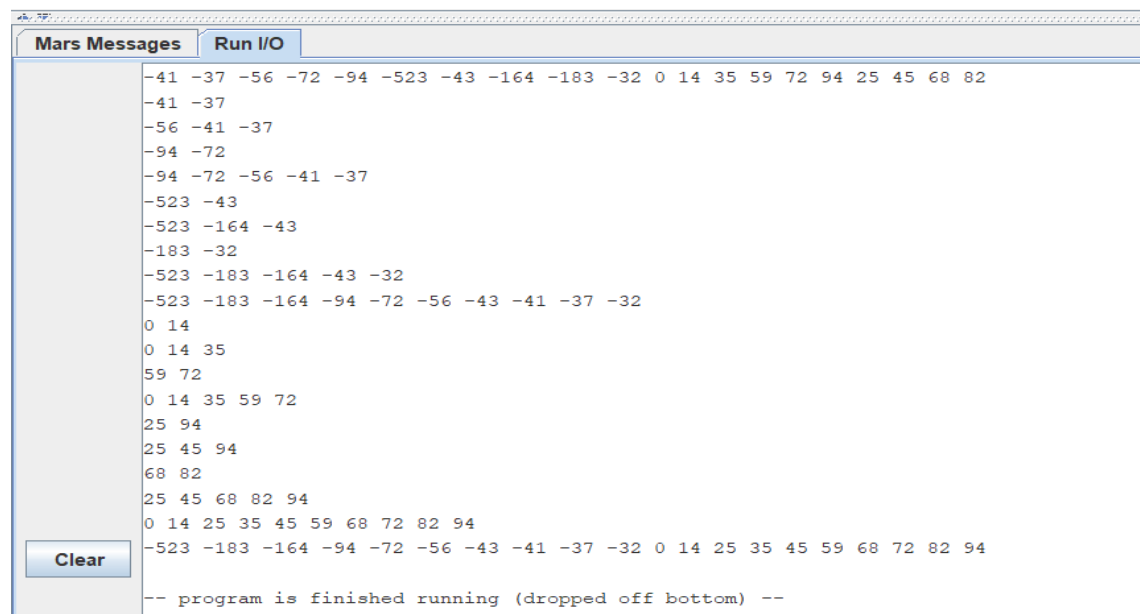


```
1 3 5 7 9 2 4 6 8 0 -11 -33 -45 -7 -79 -52 -4 -16 -18 -30
1 3
1 3 5
7 9
1 3 5 7 9
2 4
2 4 6
0 8
0 2 4 6 8
0 1 2 3 4 5 6 7 8 9
-33 -11
-45 -33 -11
-79 -7
-79 -45 -33 -11 -7
-52 -4
-52 -16 -4
-30 -18
-52 -30 -18 -16 -4
-79 -52 -45 -33 -30 -18 -16 -11 -7 -4
-79 -52 -45 -33 -30 -18 -16 -11 -7 -4 0 1 2 3 4 5 6 7 8 9
-- program is finished running (dropped off bottom) --
```

3.4 Test case 4

Cho dãy số: {-41 -37 -56 -72 -94 -523 -43 -164 -183 -32 0 14 35 59 72 94 25 45 68 82}

Output:

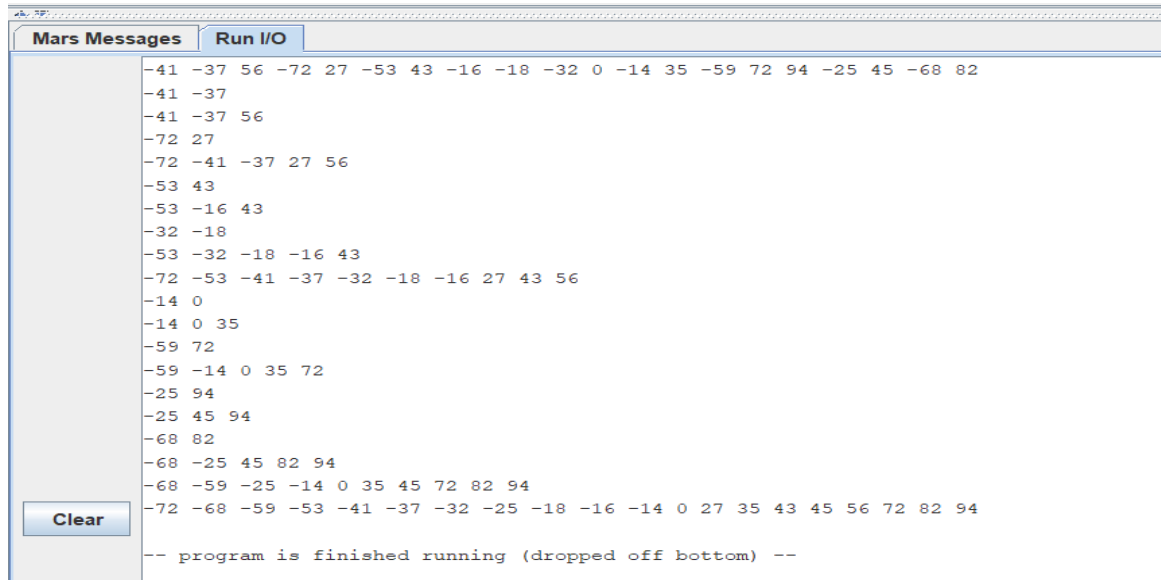


```
-41 -37 -56 -72 -94 -523 -43 -164 -183 -32 0 14 35 59 72 94 25 45 68 82
-41 -37
-56 -41 -37
-94 -72
-94 -72 -56 -41 -37
-523 -43
-523 -164 -43
-183 -32
-523 -183 -164 -43 -32
-523 -183 -164 -94 -72 -56 -43 -41 -37 -32
0 14
0 14 35
59 72
0 14 35 59 72
25 94
25 45 94
68 82
25 45 68 82 94
0 14 25 35 45 59 68 72 82 94
-523 -183 -164 -94 -72 -56 -43 -41 -37 -32 0 14 25 35 45 59 68 72 82 94
-- program is finished running (dropped off bottom) --
```


3.5 Test case 5

Cho dãy số: {-41 -37 56 -72 27 -53 43 -16 -18 -32 0 -14 35 -59 72 94 -25 45 -68 82}

Output:

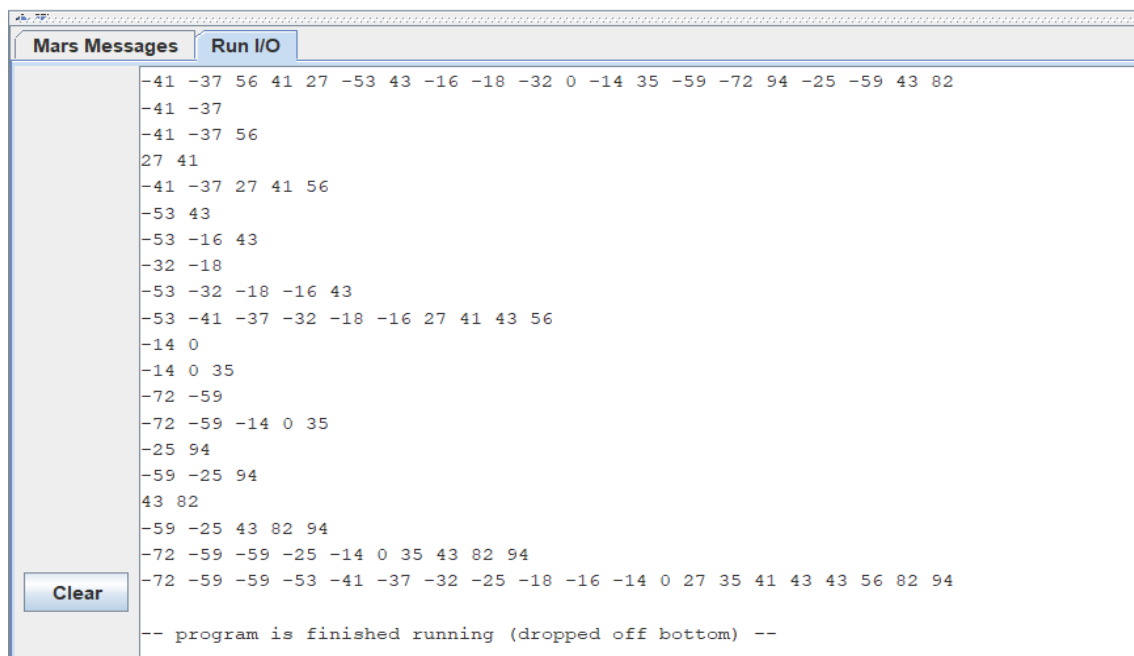


```
Mars Messages Run I/O
-41 -37 56 -72 27 -53 43 -16 -18 -32 0 -14 35 -59 72 94 -25 45 -68 82
-41 -37
-41 -37 56
-72 27
-72 -41 -37 27 56
-53 43
-53 -16 43
-32 -18
-53 -32 -18 -16 43
-72 -53 -41 -37 -32 -18 -16 27 43 56
-14 0
-14 0 35
-59 72
-59 -14 0 35 72
-25 94
-25 45 94
-68 82
-68 -25 45 82 94
-68 -59 -25 -14 0 35 45 72 82 94
-72 -68 -59 -53 -41 -37 -32 -25 -18 -16 -14 0 27 35 43 45 56 72 82 94
-- program is finished running (dropped off bottom) --
```

3.6 Test case 6

Cho dãy số: {-41 -37 56 41 27 -53 43 -16 -18 -32 0 -14 35 -59 -72 94 -25 -59 43 82}

Output:

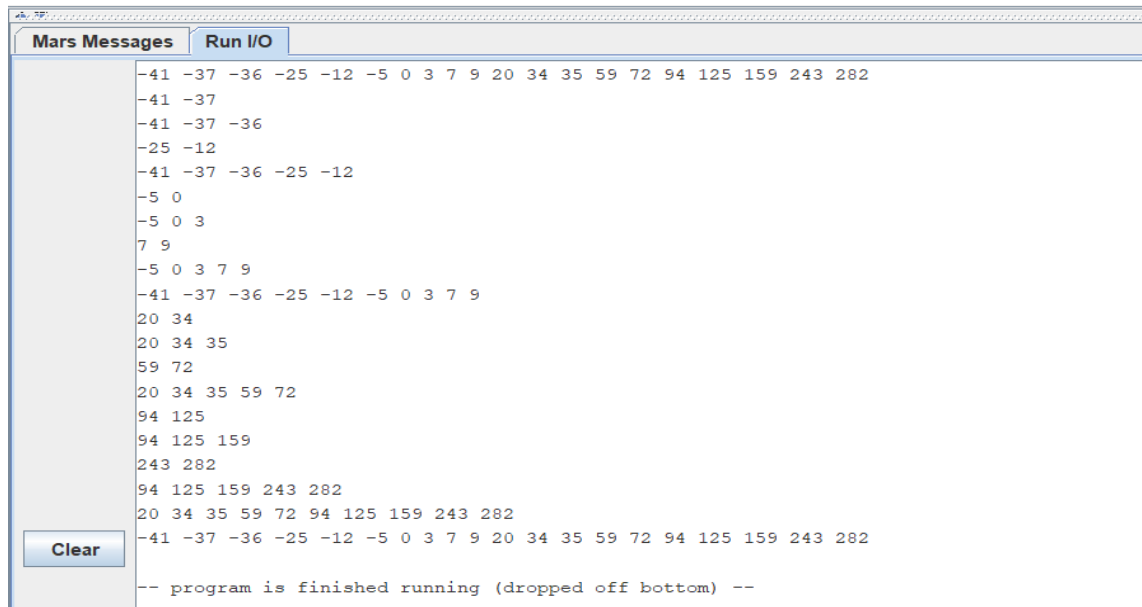


```
Mars Messages Run I/O
-41 -37 56 41 27 -53 43 -16 -18 -32 0 -14 35 -59 -72 94 -25 -59 43 82
-41 -37
-41 -37 56
27 41
-41 -37 27 41 56
-53 43
-53 -16 43
-32 -18
-53 -32 -18 -16 43
-53 -41 -37 -32 -18 -16 27 41 43 56
-14 0
-14 0 35
-72 -59
-72 -59 -14 0 35
-25 94
-59 -25 94
43 82
-59 -25 43 82 94
-72 -59 -59 -25 -14 0 35 43 82 94
-72 -59 -59 -53 -41 -37 -32 -25 -18 -16 -14 0 27 35 41 43 43 56 82 94
-- program is finished running (dropped off bottom) --
```

3.7 Test case 7

Cho dãy số: {-41 -37 -36 -25 -12 -5 0 3 7 9 20 34 35 59 72 94 125 159 243 282}

Output:

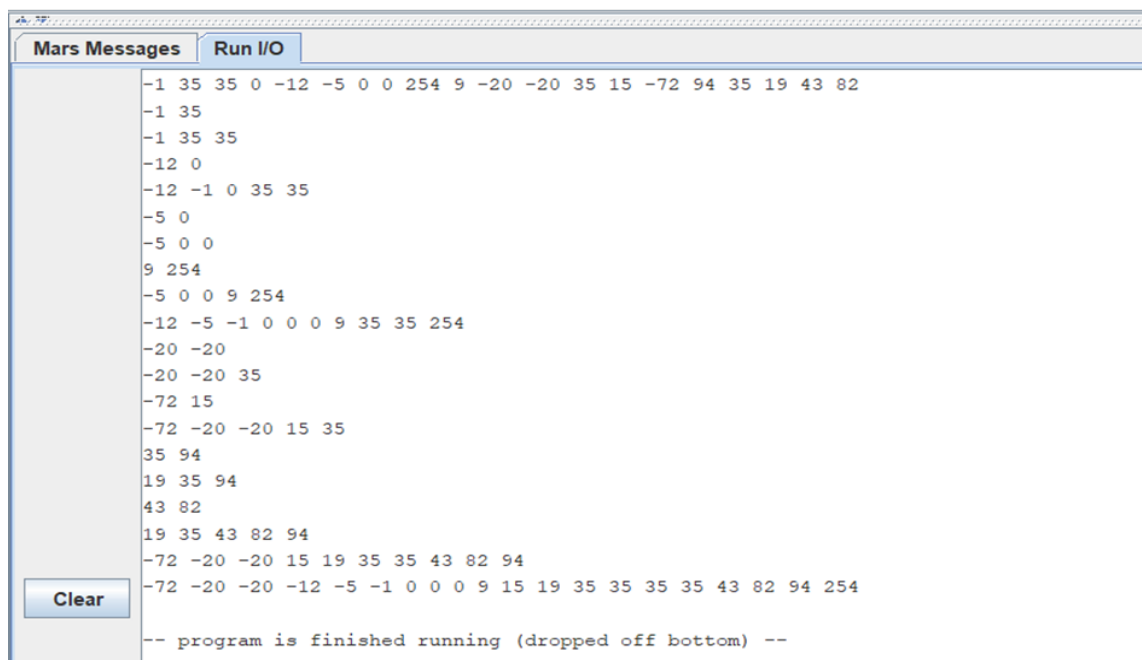


```
Mars Messages Run I/O
-41 -37 -36 -25 -12 -5 0 3 7 9 20 34 35 59 72 94 125 159 243 282
-41 -37
-41 -37 -36
-25 -12
-41 -37 -36 -25 -12
-5 0
-5 0 3
7 9
-5 0 3 7 9
-41 -37 -36 -25 -12 -5 0 3 7 9
20 34
20 34 35
59 72
20 34 35 59 72
94 125
94 125 159
243 282
94 125 159 243 282
20 34 35 59 72 94 125 159 243 282
-41 -37 -36 -25 -12 -5 0 3 7 9 20 34 35 59 72 94 125 159 243 282
-- program is finished running (dropped off bottom) --
```

3.8 Test case 8

Cho dãy số: {-1 35 35 0 -12 -5 0 0 254 9 -20 -20 35 15 -72 94 35 19 43 82}

Output:

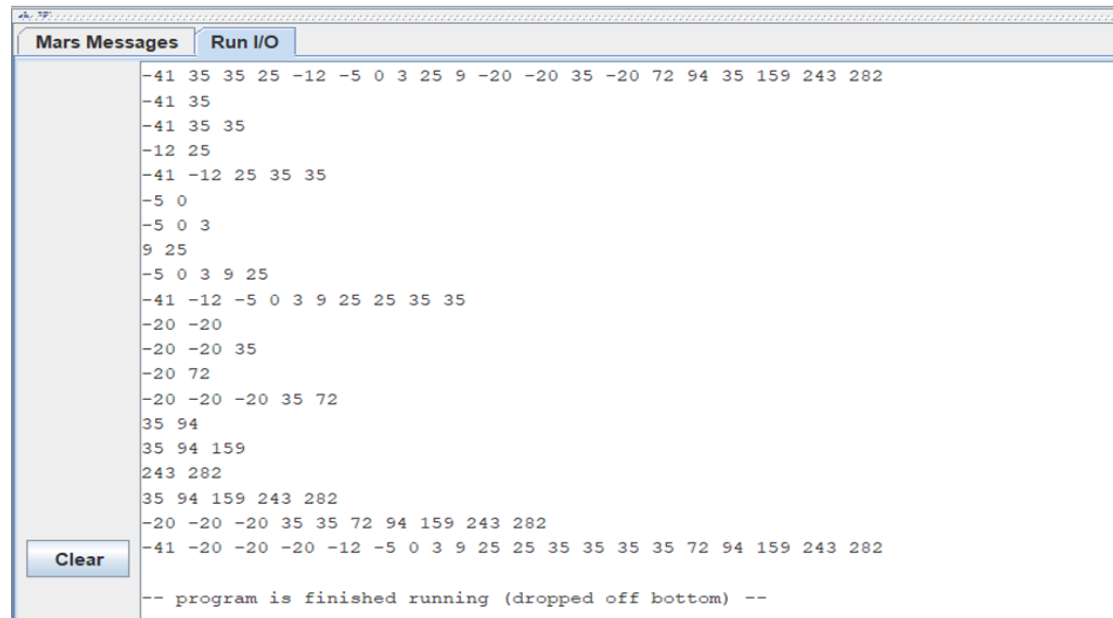


```
Mars Messages Run I/O
-1 35 35 0 -12 -5 0 0 254 9 -20 -20 35 15 -72 94 35 19 43 82
-1 35
-1 35 35
-12 0
-12 -1 0 35 35
-5 0
-5 0 0
9 254
-5 0 0 9 254
-12 -5 -1 0 0 9 35 35 254
-20 -20
-20 -20 35
-72 15
-72 -20 -20 15 35
35 94
19 35 94
43 82
19 35 43 82 94
-72 -20 -20 15 19 35 35 43 82 94
-72 -20 -20 -12 -5 -1 0 0 9 15 19 35 35 35 35 43 82 94 254
-- program is finished running (dropped off bottom) --
```

3.9 Test case 9

Cho dãy số: {-41 35 35 25 -12 -5 0 3 25 9 -20 -20 35 -20 72 94 35 159 243 282}

Output:



```

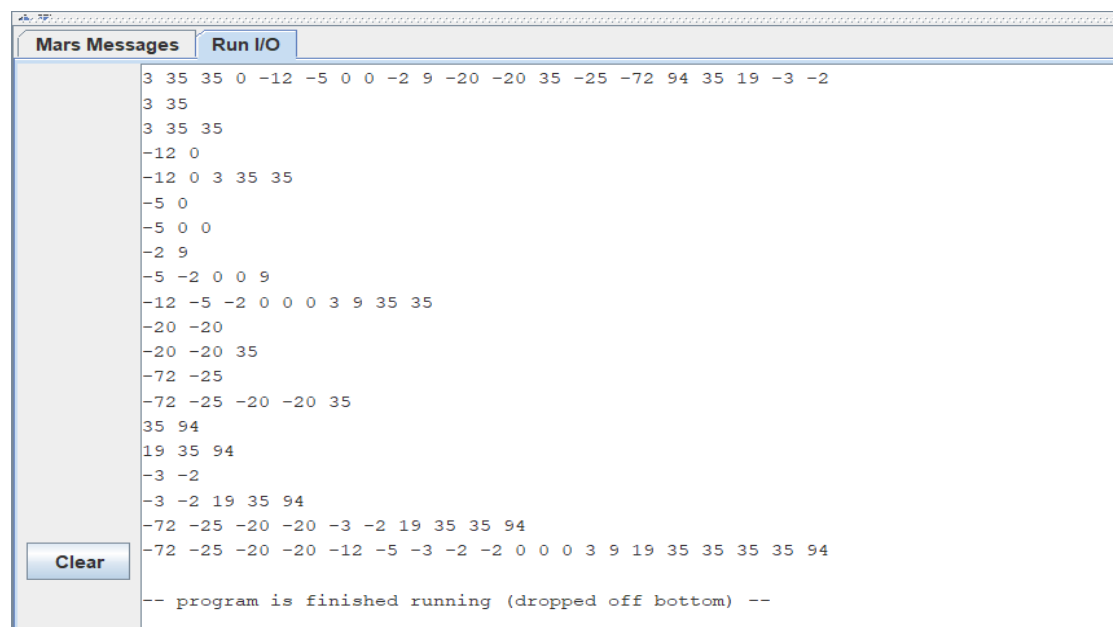
-41 35 35 25 -12 -5 0 3 25 9 -20 -20 35 -20 72 94 35 159 243 282
-41 35
-41 35 35
-12 25
-41 -12 25 35 35
-5 0
-5 0 3
9 25
-5 0 3 9 25
-41 -12 -5 0 3 9 25 25 35 35
-20 -20
-20 -20 35
-20 72
-20 -20 -20 35 72
35 94
35 94 159
243 282
35 94 159 243 282
-20 -20 -20 35 35 72 94 159 243 282
-41 -20 -20 -20 -12 -5 0 3 9 25 25 35 35 35 72 94 159 243 282

-- program is finished running (dropped off bottom) --
  
```

3.10 Test case 10

Cho dãy số: {3 35 35 0 -12 -5 0 0 -2 9 -20 -20 35 -25 -72 94 35 19 -3 -2}

Output:



```

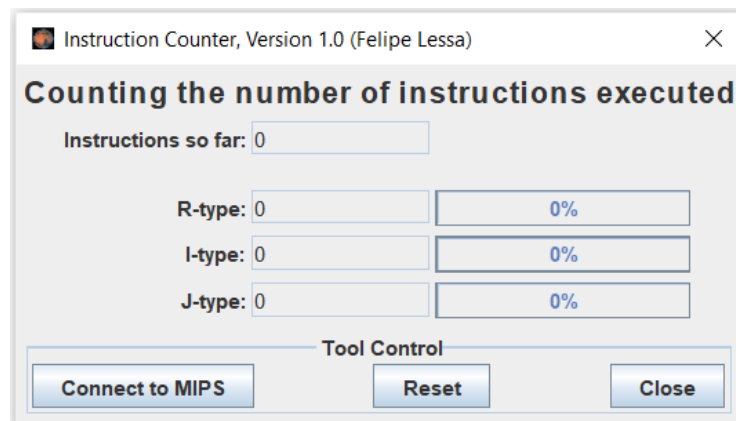
3 35 35 0 -12 -5 0 0 -2 9 -20 -20 35 -25 -72 94 35 19 -3 -2
3 35
3 35 35
-12 0
-12 0 3 35 35
-5 0
-5 0 0
-2 9
-5 -2 0 0 9
-12 -5 -2 0 0 0 3 9 35 35
-20 -20
-20 -20 35
-72 -25
-72 -25 -20 -20 35
35 94
19 35 94
-3 -2
-3 -2 19 35 94
-72 -25 -20 -20 -3 -2 19 35 35 94
-72 -25 -20 -20 -12 -5 -3 -2 -2 0 0 0 3 9 19 35 35 35 94

-- program is finished running (dropped off bottom) --
  
```

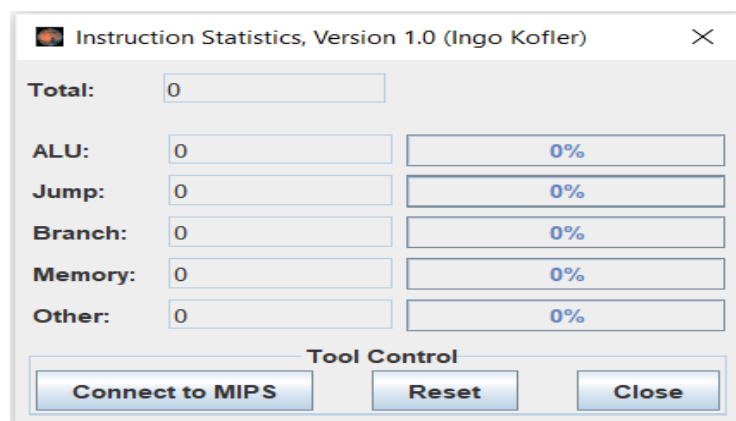
4 Tính thời gian thực thi

4.1 Thống kê các loại lệnh

Sử dụng công cụ Instruction Counter và Instruction Statistics có sẵn ở tab Tools của MARS để thống kê số lượng lệnh từng loại.

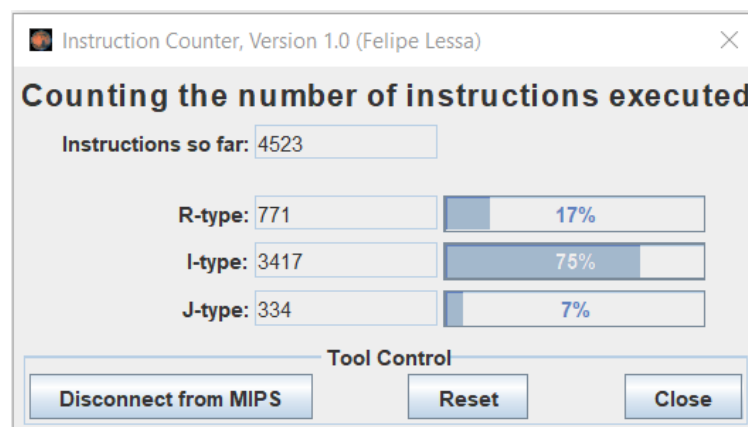


Instruction Type	Count	Percentage
R-type	0	0%
I-type	0	0%
J-type	0	0%

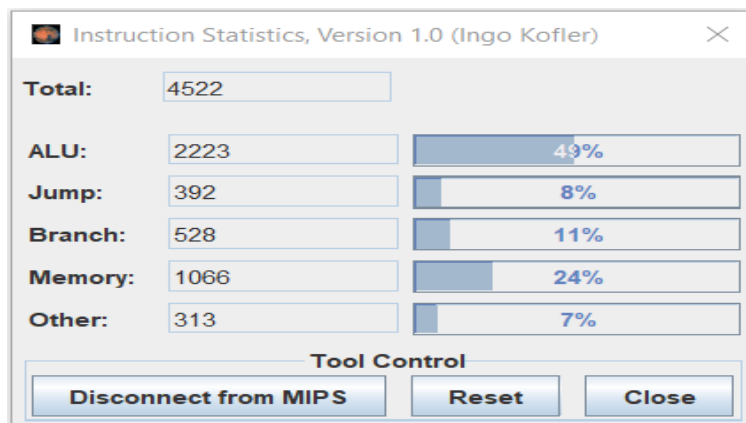


Instruction Category	Count	Percentage
ALU	0	0%
Jump	0	0%
Branch	0	0%
Memory	0	0%
Other	0	0%

Thu được số lượng từng loại lệnh như sau:



Instruction Type	Count	Percentage
R-type	771	17%
I-type	3417	75%
J-type	334	7%



Tổng số lệnh là 4523, trong đó:

- R – Type: 771, chiếm 17%.
- I – Type: 3417, chiếm 75%.
- J – Type: 334, chiếm 7%.

Số lệnh tính toán sử dụng ALU: 2223, chiếm 49%.

Số lệnh nhảy: 392, chiếm 8%.

Số lệnh rẽ nhánh: 528, chiếm 11%.

Số lệnh thao tác bộ nhớ: 1066, chiếm 24%.

Số lệnh loại khác: 313, chiếm 7%.

4.2 Tính thời gian

Thời gian CPU cho một chương trình bằng tổng số chu kỳ xung nhịp cần thiết để thực thi chương trình (CPU clock cycles) nhân với thời gian một chu kỳ xung nhịp (Clock cycle time).

$$CPU_{time} = CPU_{clock\ cycles} \times Clock\ cycle\ time = \frac{CPU_{clock\ cycles}}{Clock\ frequency} \quad (1)$$

Tính toán hiệu suất thông qua thời gian thực thi CPU không đề cập đến số lượng lệnh của chương trình. Tuy nhiên một chương trình khi được biên dịch sẽ bao gồm những lệnh và những lệnh này sẽ được máy thực thi. Do đó, thời gian thực thi một chương trình phụ thuộc vào số lượng lệnh của chương trình. Thời gian thực thi chương trình là tích của thời gian thực thi trung bình một lệnh và tổng số lệnh của chương trình. Nếu thời gian thực thi được đo dựa vào số chu kỳ thì số chu kỳ cần thiết cho một chương trình được tính như sau:

$$CPU_{clock\ cycles} = Instruction\ count \times Clock\ cycles\ per\ Instruction \quad (2)$$

Đại lượng số chu kỳ trên một lệnh (CPI) là số lượng chu kỳ trung bình cần thiết để hoàn thành một lệnh. Do các lệnh khác nhau sẽ thực thi những công việc khác nhau nên thời gian để hoàn thành các lệnh cũng khác nhau. Do đó, CPI được tính bằng giá trị trung bình của tất cả các lệnh trong chương trình.

Từ đó ta có công thức:

$$CPU_{time} = CPU_{clock\ cycles} \times Clock\ cycle\ time \quad (3)$$

$$\rightarrow CPU_{time} = Instruction\ count \times CPI \times Clock\ cycle\ time \quad (4)$$

$$\rightarrow CPU_{time} = \frac{Instruction\ count \times CPI}{clock\ rate} \quad (5)$$

Vì tất cả các loại lệnh đều có CPI là 1, ta gộp chung tất cả loại lệnh vào một lần tính. Thay số vào công thức trên, ta được:

$$CPU_{time} = \frac{Instruction\ count}{2 \times 10^9} = \frac{4523 \times 1}{2 \times 10^9} = 2.262 \times 10^{-6} s = 2262 ns \quad (6)$$

Vậy, thời gian thực thi chương trình này ở máy tính MIPS có tần số 2GHz là: 2262ns.

References

- [1] Phạm Quốc Cường, “Kiến trúc máy tính”, Nhà xuất bản Đại học quốc gia TP.HCM, Năm 2019, ISBN: 978-604-73-6708-5.
- [2] Nguyễn Văn Hiếu, “Merge Sort – Sắp xếp trộn”, xem 09/12/2020, <https://nguyenvanhieu.vn/thuat-toan-sap-xep-merge-sort/>.
- [3] MIPS Technologies, Inc., “MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set”, California June 9, 2003.
- [4] MIPS Technologies, Inc., “MIPS® Architecture for Programmers Volume IIA: The MIPS32® Instruction Set Manual”, December 15, 2016.
- [5] Wikibooks, “MIPS Assembly”, xem 10/12/2020, https://en.wikibooks.org/wiki/MIPS_Assembly