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 (CO2007)

Báo cáo Bài tập lớn: Kiến trúc tập lệnh MIPS

GVHD: Võ Tấn Phương
Trần Thanh Bình
SV thực hiện: Nguyễn Hoàng Lâm – 1910298
Nguyễn Chí Trung – 1910643
Nguyễn Hoàng Trung – 1910644

Tp. Hồ Chí Minh, Tháng 12/2020

Mục lục

1 Mở đầu

1.1 Đề bài

Đề số 6:
Cho một chuỗi số nguyên 20 phần tử.
Sử dụng hợp ngữ assembly MIPS, viết thủ tục sắp xếp chuỗi đó theo thứ tự tăng dần theo giải thuật merge sort. Yêu cầu xuất ra từng bước trong quá trình demo.
Thống kê số lệnh, loại lệnh của chương trình.
Tính thời gian chạy trên của chương trình trên máy tính MIPS tần số 2 GHz.

1.2 Thuật toán

Thuật toán merge sort gồm các bước cơ bản sau:

- Chia mảng thành hai mảng con
- Thực hiện merge sort trên mảng con bên trái
- Thực hiện merge sort trên mảng con bên phải
- Hợp nhất (merge) hai mảng con lại

1.3 Cấu trúc bài nộp

Bài nộp bao gồm các file sau:

- Một file báo cáo (AssCAReport.pdf)
- Một file source code MIPS (MergeSort.asm)

2 Nội dung

2.1 Chức năng của các hàm

Đối với bài toán trên, ta hiện thực các hàm (các hàm đều không trả về giá trị):

- Hàm `main` (hàm không có tham số): hàm được gọi đầu tiên, thực hiện hai công việc:
  - Đọc giá trị nhập vào và lưu vào mảng số nguyên `arr`
  - Gọi hàm `mergeSort` lần đầu tiên
- Hàm `printArr` (hàm không có tham số): thực hiện in mảng số nguyên `arr`
- Hàm `mergeSort` (hàm 2 tham số: vị trí bắt đầu mảng, kích thước mảng): thực hiện các công việc:
  - Chia mảng thành hai mảng con (nếu thỏa mãn kích thước mảng lớn hơn 1), gọi đệ quy hàm `mergeSort` trên mỗi mảng con
  - Gọi hàm `merge`
- Hàm `merge` (hàm 4 tham số: vị trí bắt đầu mảng con bên trái, vị trí bắt đầu mảng con bên phải, kích thước mảng trái, kích thước mảng phải): thực hiện hợp nhất hai nửa mảng truyền vào, lưu tạm giá trị vào `tempArr`, sau đó lưu lại giá trị vào `arr`

2.2 Hiện thực trên MIPS

2.2.1 Khai báo mảng

Trong vùng `.data`, thực hiện khai báo hai mảng `arr` và `tempArr` gồm 20 phần tử:

```
.data
    # Declare two 20-integer arrays: arr and tempArr
    arr:        .word   0:20
    tempArr:    .word   0:20
```

Hai mảng này có thể được truy cập tới tại tất cả các hàm.

2.2.2 Hàm `main`

Các phần của hàm:

- Đọc dữ liệu từ console:

  ```
  # Read arr
  la $t0, arr
  addi $t1, $0, 0

  # Read arr for loop
  for_read_arr:
      # Terminating condition check
      addi $t9, $t1, -20
      bgez $t9, end_for_read_arr

      # Read input and save to pointer $t0
      addi $v0, $0, 5
      syscall
      sw $v0, 0($t0)

      # Update iterators
      addi $t0, $t0, 4
      addi $t1, $t1, 1
      j for_read_arr
  end_for_read_arr:
  # End read arr
  ```

  Ý tưởng:

  - Sử dụng `$t0` lưu địa chỉ phần tử đầu của mảng `arr`, `$t1` để lưu biến đếm từ 0
  - Thực hiện vòng lặp với điều kiện biến đếm nhỏ hơn 20, mỗi lần lặp đọc một số nguyên và lưu vào địa chỉ `$t0` đang giữ
  - Cuối mỗi lần lặp, gán địa chỉ của phần tử tiếp theo cho `$t0` (cộng thêm 4) và tăng biến đếm `$t1` thêm 1 đơn vị

- Gọi hàm `mergeSort`

  ```
  # mergeSort function calling

  # Load parameter (int* arr, int size)
  la $a0, arr
  addi $a1, $0, 20

  # Jump to mergeSort
  ```

```
jal mergeSort
# End mergeSort calling
```

Ý tưởng:

- Truyền vào hai tham số `$a0` và `$a1` lần lượt địa chỉ phần tử đầu của mảng và kích thước của mảng là 20
- Gọi hàm bằng lệnh `jal`, hàm `main` là hàm gốc nên khi gọi hàm không cần đưa `$ra` vào stack

- Kết thúc chương trình

```
# Program terminating
addi $v0, $0, 10
syscall
# End program terminating
```

Hàm `main` là hàm chính của chương trình, khi thực thi đến cuối hàm, ta kết thúc chương trình bằng `syscall` với `$v0` gắn bằng 10.

2.2.3 Hàm `printArr`

Các phần của hàm:

- In các phần tử của mảng `arr`

```
# Load arr
la $t0, arr
addi $t1, $0, 0

# Print arr for loop
for_print_arr:
    # Terminating condition check
    addi $t9, $t1, -20
    bgez $t9, end_for_print_arr

    # Print integer saved to pointer $t0
    addi $v0, $0, 1
    lw $a0, 0($t0)
    syscall

    # Print space character
    addi $a0, $0, ' '
    addi $v0, $0, 11
    syscall

    # Update iterators
    addi $t0, $t0, 4
    addi $t1, $t1, 1
    j for_print_arr
end_for_print_arr:

# Print endline character
addi $a0, $0, '\n'
addi $v0, $0, 11
syscall
```

Ý tưởng:

- Sử dụng `$t0` lưu địa chỉ phần tử đầu của mảng `arr`, `$t1` để lưu biến đếm từ 0
- Thực hiện vòng lặp với điều kiện biến đếm nhỏ hơn 20, mỗi lần lặp in ra số nguyên lưu tại địa chỉ `$t0` đang giữ, kèm một dấu khoảng cách
- Cuối mỗi lần lặp, gán địa chỉ của phần tử tiếp theo cho `$t0` (cộng thêm 4) và tăng biến đếm `$t1` thêm 1 đơn vị
- Kết thúc vòng lặp, in ra console một dấu xuống dòng

- Kết thúc hàm

```
# Return to previous function
jr $ra
# End read arr
```

Ý tưởng:

- Kết thúc hàm bằng lệnh `jr`
- Vì `printArr` là một hàm lá (hàm không gọi hàm khác) nên không cần lưu các giá trị hay `$ra` vào stack

2.2.4 Hàm `mergeSort`

Các phần của hàm:

- Load các tham số vào các thanh ghi và kiểm tra điều kiện

```
# Unload parameter (int* arr, int size)
add $s0, $a0, $0
add $s1, $a1, $0

# Terminate function if size <= 1
addi $t9, $s1, -1
blez $t9, end_merge_sort
```

Ý tưởng:

- Nếu kích thước mảng truyền vào không lớn hơn 1 thì kết thúc hàm (base case của đệ quy)

- Thực hiện đệ quy với mảng con bên trái

```
# Left half merge sort

# Save current parameters and return address to stack
addi $sp, $sp, -4
sw $ra, 0($sp)
addi $sp, $sp, -4
sw $s0, 0($sp)
addi $sp, $sp, -4
sw $s1, 0($sp)

# Compute new argument
add $a0, $s0, $0
addi $t2, $0, 2
div $s1, $t2
mflo $a1
```

```
# Half left merge sort
jal mergeSort

# Pop parameters from stack
lw $s1, 0($sp)
addi $sp, $sp, 4
lw $s0, 0($sp)
addi $sp, $sp, 4
lw $ra, 0($sp)
addi $sp, $sp, 4
# End left half merge sort
```

Ý tưởng:

- Vì `mergeSort` gọi hàm, do đó phải đưa `$ra` và các giá trị có thể thay đổi khi hàm khác thực hiện vào stack
- Tính toán các tham số mới cho hàm `mergeSort` và thực hiện gọi đệ quy
- Pop stack và trả lại các giá trị và `$ra` như ban đầu

- Thực hiện đệ quy với mảng con bên phải
  Ý tưởng:
  - Tương tự như đệ quy đối với mảng con bên trái
- Gọi hàm `merge`

```
# Merge 2 halves of the array

# Save current parameters and return address to stack
addi $sp, $sp, -4
sw $ra, 0($sp)
addi $sp, $sp, -4
sw $s0, 0($sp)
addi $sp, $sp, -4
sw $s1, 0($sp)

# Compute arguments for merge: (int* left, int* right, int sizeLeft, int sizeRight)
add $a0, $s0, $0
addi $t2, $0, 2
div $s1, $t2
mflo $t0
addi $t9, $0, 4
mul $t1, $t0, $t9
add $a1, $a0, $t1
add $a2, $t0, $0
sub $a3, $s1, $a2

# Half right merge sort
jal merge

# Pop parameters from stack
lw $s1, 0($sp)
addi $sp, $sp, 4
lw $s0, 0($sp)
addi $sp, $sp, 4
lw $ra, 0($sp)
```

```
addi $sp, $sp, 4
# End merge 2 halves
```

Ý tưởng:

- Đẩy `$ra` và các giá trị có thể thay đổi vào stack
- Tính toán các tham số truyền vào và gọi hàm
- Pop stack, trả lại giá trị cho `$ra` và các giá trị khác

- Gọi hàm `printArr`

```
# Print arr

# Save current parameters and return address to stack
addi $sp, $sp, -4
sw $ra, 0($sp)
addi $sp, $sp, -4
sw $s0, 0($sp)
addi $sp, $sp, -4
sw $s1, 0($sp)

# Half right merge sort
jal printArr

# Pop parameters from stack
lw $s1, 0($sp)
addi $sp, $sp, 4
lw $s0, 0($sp)
addi $sp, $sp, 4
lw $ra, 0($sp)
addi $sp, $sp, 4
# End print arr
```

Ý tưởng:

- Đẩy `$ra` và các giá trị có thể thay đổi vào stack
- Gọi hàm `printArr`
- Pop stack, trả lại giá trị cho `$ra` và các giá trị khác

- Kết thúc hàm

```
# Return to previous function
jr $ra
```

Ý tưởng:

- Sử dụng lệnh `jr` để kết thúc hàm

2.2.5 Hàm `merge`

Các phần của hàm:

- Hợp nhất hai mảng con, lưu vào mảng `tempArr`

```
# Iterators initialize
addi $t0, $0, 0
addi $t1, $0, 0
la $s0, tempArr

#-------------------------------------------------------------
# Merging for_loop
for_merge:
    # Checking condition
    sub $t9, $t0, $a2
    bltz $t9, begin_for_merge
    sub $t9, $t1, $a3
    bltz $t9, begin_for_merge
    j end_for_merge
    begin_for_merge:

    # Address adding unit
    addi $t9, $0, 4
    mul $t2, $t0, $t9
    mul $t3, $t1, $t9
    add $t4, $a0, $t2
    add $t5, $a1, $t3
    lw $t6, 0($t4)
    lw $t7, 0($t5)

    # Branching cases
    if1:
        sub $t9, $t0, $a2
        bltz $t9, if2
        sw $t7, 0($s0)
        addi $t1, $t1, 1
        j end_branching

    if2:
        sub $t9, $t1, $a3
        bltz $t9, if3
        sw $t6, 0($s0)
        addi $t0, $t0, 1
        j end_branching

    if3:
        sub $t9, $t6, $t7
        bgtz $t9, if4
        sw $t6, 0($s0)
        addi $t0, $t0, 1
        j end_branching

    if4:
        sw $t7, 0($s0)
        addi $t1, $t1, 1
    end_branching:

    # Update iterators
    addi $s0, $s0, 4
    j for_merge
end_for_merge:
```

Ý tưởng:

- Sử dụng các thanh ghi `$t0` và `$t1` để làm các biến đếm từ 0 cho mỗi nửa mảng, thanh ghi `$s0` chứa địa chỉ phần tử đầu tiên của `tempArr`
- Nếu chưa duyệt hết hai mảng con, thực hiện vòng lặp, mỗi vòng lặp được rẽ nhánh thành 3 trường hợp:
  * Nếu đã duyệt hết mảng bên trái: gán giá trị của thanh ghi `$s0` đang trỏ tới bằng phần tử đang duyệt của mảng bên phải, tăng biến đếm `$t1` lên 1 đơn vị
  * Nếu đã duyệt hết mảng bên phải: gán giá trị của thanh ghi `$s0` đang trỏ tới bằng phần tử đang duyệt của mảng bên trái, tăng biến đếm `$t0` lên 1 đơn vị
  * Nếu đều chưa duyệt hết hai mảng: gán giá trị của thanh ghi `$s0` đang trỏ tới bằng phần tử có giá trị nhỏ hơn trong hai phần tử đang duyệt của hai mảng, tăng biến đếm tương ứng lên 1 đơn vị
- Trỏ `$s0` đến phần tử tiếp theo cuối mỗi lần lặp (cộng thêm 4)

- Lưu lại giá trị từ `tempArr` vào lại `arr` tại vị trí tương ứng

```
# Moving data to original array

# Iterators initialize
la $s0, tempArr
add $s1, $a0, $0
addi $t0, $0, 0
add $t1, $a2, $a3

# Moving data for loop
for_moving_data:
    # Checking condition
    sub $t9, $t0, $t1
    bgez $t9, end_for_moving_data

    # Moving data
    lw $t2, 0($s0)
    sw $t2, 0($s1)

    # Update iterators
    addi $s0, $s0, 4
    addi $s1, $s1, 4
    addi $t0, $t0, 1
    j for_moving_data
end_for_moving_data:
```

Ý tưởng:

- Sử dụng thanh ghi `$t0` làm biến đếm bắt đầu thì 0, `$s0`, `$s1` lần lượt chứa địa chỉ của phần tử đầu tiên trong mảng `tempArr` và mảng bên trái
- Nếu `$t0` bé hơn tổng kích thước hai mảng con trái và phải, thực hiện vòng lặp, mỗi lần lặp lưu giá trị của phần tử `$s0` trỏ đến vào địa chỉ `$s1` trỏ đến
- Sau mỗi lần lặp, tăng biến đếm `$t0` lên 1 đơn vị, `$s0` và `$s1` trỏ đến các phần tử tiếp theo (cộng thêm 4)

- Kết thúc hàm

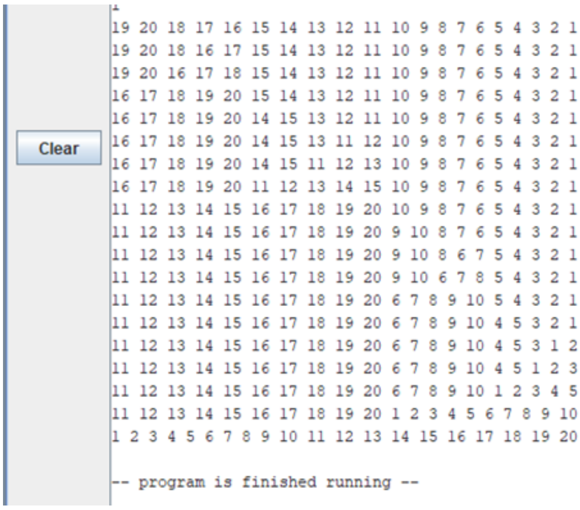
```
# Return to previous function
jr $ra
```

Ý tưởng:

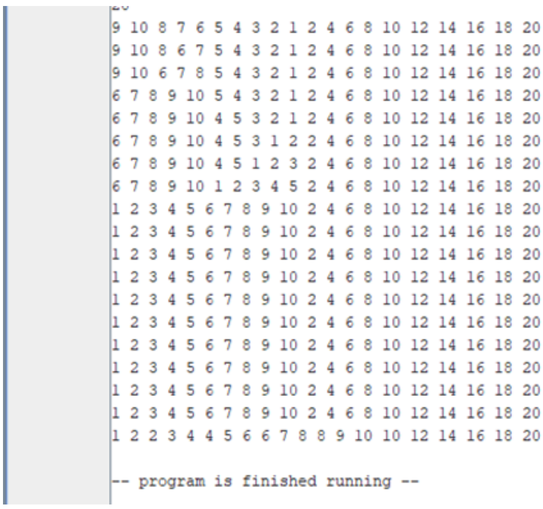
- Sử dụng lệnh `jr` để kết thúc hàm, trở về hàm trước đó
- Vì `merge` là hàm lá (hàm không gọi hàm khác), không cần các thao tác liên quan đến stack

2.3 Một số testcase

Testcase 1: Xét mảng 20 phần tử: [20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Kết quả in ra trong console qua từng bước:



Testcase 2: Xét mảng 20 phần tử: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
Kết quả in ra trong console qua từng bước:



Testcase 3: Xét mảng 20 phần tử: [20, -19, 18, -17, 16, -15, 14, -13, 12, -11, 10, -9, 8, -7, 6, -5, 4, -3, 2, -1]
Kết quả in ra trong console qua từng bước:

```
-19 20 18 -17 16 -15 14 -13 12 -11 10 -9 8 -7 6 -5 4 -3 2 -1
-19 20 18 -17 16 -15 14 -13 12 -11 10 -9 8 -7 6 -5 4 -3 2 -1
-19 20 -17 16 18 -15 14 -13 12 -11 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 16 18 20 -15 14 -13 12 -11 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 16 18 20 -15 14 -13 12 -11 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 16 18 20 -15 14 -13 -11 12 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 16 18 20 -15 14 -13 -11 12 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 16 18 20 -15 -13 -11 12 14 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 10 -9 8 -7 6 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 -9 10 8 -7 6 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 -9 10 8 -7 6 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 -9 10 -7 6 8 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 -9 -7 6 8 10 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 -9 -7 6 8 10 -5 4 -3 2 -1
-19 -17 -15 -13 -11 12 14 16 18 20 -9 -7 6 8 10 -5 4 -3 -1 2
-19 -17 -15 -13 -11 12 14 16 18 20 -9 -7 6 8 10 -5 4 -3 -1 2
-19 -17 -15 -13 -11 12 14 16 18 20 -9 -7 6 8 10 -5 -3 -1 2 4
-19 -17 -15 -13 -11 12 14 16 18 20 -9 -7 -5 -3 -1 2 4 6 8 10
-19 -17 -15 -13 -11 -9 -7 -5 -3 -1 2 4 6 8 10 12 14 16 18 20

-- program is finished running --
```

Testcase 4: Xét mảng 20 phần tử: [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
Kết quả in ra trong console qua từng bước:

```
0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
0 0 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0
0 0 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0
0 0 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 0
0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1
0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1
0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 1 1
0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1

-- program is finished running --
```

Testcase 5: Xét mảng 20 phần tử: [123, 234, 345, 135, 246, 357, 147, 258, 369, 1000, -123, -234, -345, -135, -246, -357, -147, -258, -369, -1000]
Kết quả in ra trong console qua từng bước:

```
-1000
123 234 345 135 246 357 147 258 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 234 345 135 246 357 147 258 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 234 135 246 345 357 147 258 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 234 246 345 357 147 258 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 234 246 345 147 357 258 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 234 246 345 147 357 258 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 234 246 345 147 258 357 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -123 -234 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -234 -123 -345 -135 -246 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -234 -123 -345 -246 -135 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -234 -123 -345 -246 -135 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -345 -246 -234 -135 -123 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -345 -246 -234 -135 -123 -357 -147 -258 -369 -1000
123 135 147 234 246 258 345 357 369 1000 -345 -246 -234 -135 -123 -357 -147 -258 -1000 -369
123 135 147 234 246 258 345 357 369 1000 -345 -246 -234 -135 -123 -357 -147 -1000 -369 -258
123 135 147 234 246 258 345 357 369 1000 -345 -246 -234 -135 -123 -1000 -369 -357 -258 -147
123 135 147 234 246 258 345 357 369 1000 -1000 -369 -357 -345 -258 -246 -234 -147 -135 -123
-1000 -369 -357 -345 -258 -246 -234 -147 -135 -123 123 135 147 234 246 258 345 357 369 1000

-- program is finished running --
```

Testcase 6: Xét mảng 20 phần tử: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Kết quả in ra trong console qua từng bước:

```
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

-- program is finished running --
```

Testcase 7: Xét mảng 20 phần tử: [5, 4, 6, 3, 7, 2, 8, 1, 9, 2, 8, 3, 7, 4, 6, 5, 55, 555, 5555, 5]
Kết quả in ra trong console qua từng bước:

```
5
4 5 6 3 7 2 8 1 9 2 8 3 7 4 6 5 55 555 5555 5
4 5 6 3 7 2 8 1 9 2 8 3 7 4 6 5 55 555 5555 5
4 5 3 6 7 2 8 1 9 2 8 3 7 4 6 5 55 555 5555 5
3 4 5 6 7 2 8 1 9 2 8 3 7 4 6 5 55 555 5555 5
3 4 5 6 7 2 8 1 9 2 8 3 7 4 6 5 55 555 5555 5
3 4 5 6 7 2 8 1 2 9 8 3 7 4 6 5 55 555 5555 5
3 4 5 6 7 2 8 1 2 9 8 3 7 4 6 5 55 555 5555 5
3 4 5 6 7 1 2 2 8 9 8 3 7 4 6 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 8 3 7 4 6 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 3 8 7 4 6 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 3 8 7 4 6 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 3 8 4 6 7 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 3 4 6 7 8 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 3 4 6 7 8 5 55 555 5555 5
1 2 2 3 4 5 6 7 8 9 3 4 6 7 8 5 55 555 5 5555
1 2 2 3 4 5 6 7 8 9 3 4 6 7 8 5 55 5 555 5555
1 2 2 3 4 5 6 7 8 9 3 4 6 7 8 5 5 55 555 5555
1 2 2 3 4 5 6 7 8 9 3 4 5 5 6 7 8 55 555 5555
1 2 2 3 3 4 4 5 5 5 6 6 7 7 8 8 9 55 555 5555

-- program is finished running --
```

Testcase 8: Xét mảng 20 phần tử: [20 1 19 2 18 3 17 4 16 5 15 6 14 7 13 8 12 9 11 10]
Kết quả in ra trong console qua từng bước:

```
1 20 19 2 18 3 17 4 16 5 15 6 14 7 13 8 12 9 11 10
1 20 19 2 18 3 17 4 16 5 15 6 14 7 13 8 12 9 11 10
1 20 2 18 19 3 17 4 16 5 15 6 14 7 13 8 12 9 11 10
1 2 18 19 20 3 17 4 16 5 15 6 14 7 13 8 12 9 11 10
1 2 18 19 20 3 17 4 16 5 15 6 14 7 13 8 12 9 11 10
1 2 18 19 20 3 17 4 5 16 15 6 14 7 13 8 12 9 11 10
1 2 18 19 20 3 17 4 5 16 15 6 14 7 13 8 12 9 11 10
1 2 18 19 20 3 4 5 16 17 15 6 14 7 13 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 15 6 14 7 13 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 6 15 14 7 13 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 6 15 14 7 13 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 6 15 7 13 14 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 6 7 13 14 15 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 6 7 13 14 15 8 12 9 11 10
1 2 3 4 5 16 17 18 19 20 6 7 13 14 15 8 12 9 10 11
1 2 3 4 5 16 17 18 19 20 6 7 13 14 15 8 12 9 10 11
1 2 3 4 5 16 17 18 19 20 6 7 13 14 15 8 9 10 11 12
1 2 3 4 5 16 17 18 19 20 6 7 8 9 10 11 12 13 14 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

-- program is finished running --
```

Testcase 9: Xét mảng 20 phần tử: [1, 6, 11, 16, 2, 7, 12, 17, 3, 8, 13, 18, 4, 9, 14, 19, 5, 10, 15, 20]
Kết quả in ra trong console qua từng bước:

```
20
1 6 11 16 2 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
1 6 11 2 16 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
1 6 2 11 16 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
1 2 6 11 16 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
1 2 6 11 16 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
1 2 6 11 16 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
1 2 6 11 16 7 12 3 8 17 13 18 4 9 14 19 5 10 15 20
1 2 6 11 16 3 7 8 12 17 13 18 4 9 14 19 5 10 15 20
1 2 3 6 7 8 11 12 16 17 13 18 4 9 14 19 5 10 15 20
1 2 3 6 7 8 11 12 16 17 13 18 4 9 14 19 5 10 15 20
1 2 3 6 7 8 11 12 16 17 13 18 4 9 14 19 5 10 15 20
1 2 3 6 7 8 11 12 16 17 13 18 4 9 14 19 5 10 15 20
1 2 3 6 7 8 11 12 16 17 4 9 13 14 18 19 5 10 15 20
1 2 3 6 7 8 11 12 16 17 4 9 13 14 18 5 19 10 15 20
1 2 3 6 7 8 11 12 16 17 4 9 13 14 18 5 19 10 15 20
1 2 3 6 7 8 11 12 16 17 4 9 13 14 18 5 19 10 15 20
1 2 3 6 7 8 11 12 16 17 4 9 13 14 18 5 10 15 19 20
1 2 3 6 7 8 11 12 16 17 4 5 9 10 13 14 15 18 19 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

-- program is finished running --
```

Testcase 10: Xét mảng 20 phần tử: [111 22 3 44 555 66 7 88 999 1000 999 88 7 66 555 44 3 22 111 0]
Kết quả in ra trong console qua từng bước:

```
22 111 3 44 555 66 7 88 999 1000 999 88 7 66 555 44 3 22 111 0
22 111 3 44 555 66 7 88 999 1000 999 88 7 66 555 44 3 22 111 0
22 111 3 44 555 66 7 88 999 1000 999 88 7 66 555 44 3 22 111 0
3 22 44 111 555 66 7 88 999 1000 999 88 7 66 555 44 3 22 111 0
3 22 44 111 555 7 66 88 999 1000 999 88 7 66 555 44 3 22 111 0
3 22 44 111 555 7 66 88 999 1000 999 88 7 66 555 44 3 22 111 0
3 22 44 111 555 7 66 88 999 1000 999 88 7 66 555 44 3 22 111 0
3 22 44 111 555 7 66 88 999 1000 999 88 7 66 555 44 3 22 111 0
3 7 22 44 66 88 111 555 999 1000 999 88 7 66 555 44 3 22 111 0
3 7 22 44 66 88 111 555 999 1000 88 999 7 66 555 44 3 22 111 0
3 7 22 44 66 88 111 555 999 1000 88 999 7 66 555 44 3 22 111 0
3 7 22 44 66 88 111 555 999 1000 88 999 7 66 555 44 3 22 111 0
3 7 22 44 66 88 111 555 999 1000 7 66 88 555 999 44 3 22 111 0
3 7 22 44 66 88 111 555 999 1000 7 66 88 555 999 3 44 22 111 0
3 7 22 44 66 88 111 555 999 1000 7 66 88 555 999 3 44 22 0 111
3 7 22 44 66 88 111 555 999 1000 7 66 88 555 999 3 44 0 22 111
3 7 22 44 66 88 111 555 999 1000 7 66 88 555 999 0 3 22 44 111
3 7 22 44 66 88 111 555 999 1000 0 3 7 22 44 66 88 111 555 999
0 3 3 7 7 22 22 44 44 66 66 88 88 111 111 555 555 999 999 1000

-- program is finished running --
```

2.4 Thống kê tổng số lượng lệnh và số lượng lệnh mỗi loại

Tổng số lượng lệnh thực thi tùy vào từng testcase, vì trong chương trình có cấu trúc rẽ nhánh, tùy vào từng điều kiện, chương trình sẽ thực thi những đoạn lệnh khác nhau.
Để thống kế số lượng lệnh, ta sử dụng tool **Instruction Counter** trong MARS.
Minh họa cho cách sử dụng, xét các testcase ở mục **2.4**, kết quả thu được là:

Số lượng lệnh	R-Type	I-Type	J-Type	Tổng cộng
Testcase 1	2087	5945	712	8744
Testcase 2	2087	5945	737	8769
Testcase 3	2098	5956	720	8774
Testcase 4	2098	5956	737	8791
Testcase 5	2092	5950	727	8769
Testcase 6	2079	5937	760	8776
Testcase 7	2100	5958	733	8791
Testcase 8	2098	5956	720	8774
Testcase 9	2102	5960	733	8795
Testcase 10	2101	5959	729	8789

2.5 Tính toán thời gian thực thi trên máy MIPS 2.0 GHz

Thời gian thực thi được tính bằng công thức:

$$\text{Time} = \frac{\text{Clock cycles}}{\text{Clock rate}} = \frac{\sum \text{CPI}_i \times n_i}{\text{Clock rate}}$$

Trong đó: Time là thời gian thực thi, tính bằng s
Clock rate là tần số, tính bằng Hz
Clock cycles là tổng số chu kì
CPI_i là số chu kì trên một lệnh ứng với loại lệnh thứ i
n_i là số lệnh ứng với lệnh thứ i

Với giả sử CPI của mỗi nhóm lệnh trên đều bằng 1, thời gian thực thi trở thành:

$$\text{Time} = \frac{\text{Clock cycles}}{\text{Clock rate}} = \frac{1 \times n}{\text{Clock rate}}$$

Vậy, thời gian thực thi của máy MIPS 2.0 GHz trên chương trình với mỗi testcase ở mục **2.4** là:

	Thời gian
Testcase 1	$\frac{1 \times 8744}{2 \times 10^9} = 4.372 \times 10^{-6}$ (s)
Testcase 2	$\frac{1 \times 8769}{2 \times 10^9} = 4.385 \times 10^{-6}$ (s)
Testcase 3	$\frac{1 \times 8774}{2 \times 10^9} = 4.387 \times 10^{-6}$ (s)
Testcase 4	$\frac{1 \times 8791}{2 \times 10^9} = 4.396 \times 10^{-6}$ (s)
Testcase 5	$\frac{1 \times 8769}{2 \times 10^9} = 4.385 \times 10^{-6}$ (s)
Testcase 6	$\frac{1 \times 8776}{2 \times 10^9} = 4.388 \times 10^{-6}$ (s)
Testcase 7	$\frac{1 \times 8791}{2 \times 10^9} = 4.396 \times 10^{-6}$ (s)
Testcase 8	$\frac{1 \times 8774}{2 \times 10^9} = 4.387 \times 10^{-6}$ (s)
Testcase 9	$\frac{1 \times 8795}{2 \times 10^9} = 4.398 \times 10^{-6}$ (s)
Testcase 10	$\frac{1 \times 8789}{2 \times 10^9} = 4.395 \times 10^{-6}$ (s)

3 Kết luận

Hợp ngữ MIPS là một ngôn ngữ lập trình cấp thấp thân thiện, dễ học và dễ viết. Với một chương trình cho trước, ta có thể chuyển đổi về MIPS, ngôn ngữ gần với ngôn ngữ máy, từ đó có thể hiểu rõ được quá trình hoạt động bên trong những chiếc máy tính.

Tài liệu

[1] David A.Patteson, John L.H. Computer Organization and Design: The Hardware/Software Interface, fifth edition. Morgan Kaufmann.

[2] Phạm Quốc Cường. Kiến trúc máy tính. NXB Đại học Quốc gia TP.Hồ Chí Minh.