

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



BÁO CÁO BÀI TẬP

MÔN: KIẾN TRÚC HỆ THỐNG HIỆN ĐẠI

---

## BÀI TẬP MIPS

---

**GVHD: PGS.TS Trần Ngọc Thịnh**

**Khóa: 2021**

**Nhóm: 8**

Phan Lê Tuấn Anh - 1811442

Lê Đức Huy - 1810166

Nguyễn Công Minh - 1813083

Vũ Đức Mạnh - 1810320

Nguyễn Tiến Đạt - 1811884

Trần Khương Duy - 1811744

Đoàn Thanh Nam - 1813130

Tô Duy Hưng - 1810198

Đỗ Lê Quang Trung - 1811304

Lê Kha - 1810976

TP. HỒ CHÍ MINH, THÁNG 10/2021



## BẢNG PHÂN CÔNG CÔNG VIỆC

STT	Họ và tên	Phân công công việc	Hoàn thành công việc
1	Phan Lê Tuấn Anh	Làm bài 5	100%
2	Nguyễn Công Minh	Làm bài 6	100%
3	Nguyễn Tiến Đạt	Làm bài 5	100%
4	Đoàn Thanh Nam	Làm bài 7	100%
5	Đỗ Lê Quang Trung	Làm bài 3	100%
6	Lê Đức Huy	Làm bài 1	100%
7	Vũ Đức Mạnh	Làm bài 2	100%
8	Trần Khương Duy	Làm bài 7	100%
9	Tô Duy Hưng	Làm bài 4	100%
10	Lê Kha	Làm bài 6	100%



## MỤC LỤC

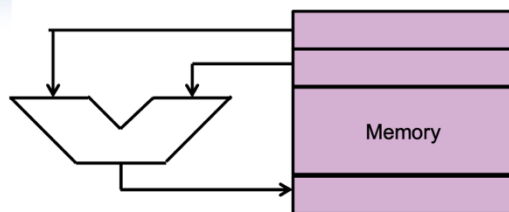
<b>Bài 1:</b> .....	<b>1</b>
<b>Bài 2:</b> .....	<b>2</b>
<b>Bài 3:</b> .....	<b>2</b>
<b>Bài 4:</b> .....	<b>2</b>
<b>Bài 5:</b> .....	<b>4</b>
<b>Bài 6:</b> .....	<b>4</b>
<b>Bài 7:</b> .....	<b>4</b>

### Bài 1:

Cho biết nếu sử dụng CPU có ISA kiểu Memory-Memory để tính biểu thức sau thì cần bao nhiêu lần truy cập bộ nhớ? Nếu sử dụng CPU có ISA kiểu General Purpose Register thì cần bao nhiêu lần truy cập bộ nhớ?

$$A = B + C + B * C;$$

\* CPU có ISA kiểu Memory – Memory:



$$A = B + C + B * C$$

ADD X, B, C  
MUL Y, B, C  
ADD A, X, Y

Kiểu Memory – Memory: tất cả các toán hạng đều phải truy cập bộ nhớ.

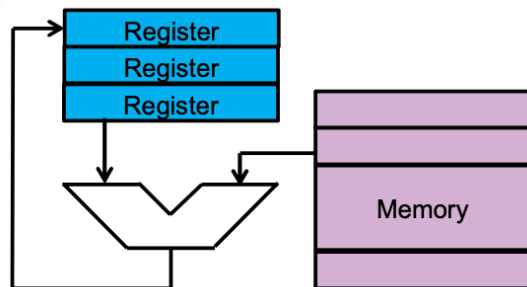
Nên lệnh **ADD X, B, C** có 3 lần truy cập bộ nhớ: truy cập B để lấy giá trị, truy cập C để lấy giá trị, truy cập X để ghi kết quả.

Tương tự lệnh **MUL Y, B, C** và **ADD A, X, Y** cũng đều có 3 lần truy cập bộ nhớ.

Số lần truy cập bộ nhớ =  $3 * 3 = 9$  lần

\* CPU có ISA kiểu General Purpose Register: chia thành 2 loại Register – Memory, Register – Register

a) Register – Memory

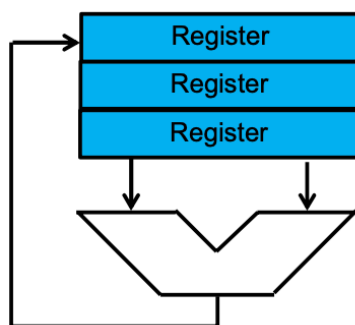


$$A = B + C + B * C$$

**LOAD R1, B**  
**ADD R2, R1, C**  
**MUL R3, R1, C**  
**ADD R4, R2, R3**  
**STORE R4, A**

Số lần truy cập bộ nhớ = Tổng số lệnh LOAD, STORE + 2 lần truy cập bộ nhớ để lấy giá trị C = 4 lần

b) Register – Register



$$A = B + C + B * C$$

**LOAD R1, B**  
**LOAD R2, C**  
**ADD R3, R1, R2**  
**MUL R4, R1, R2**  
**ADD R5, R3, R4**  
**STORE R5, A**

Số lần truy cập bộ nhớ = Tổng số lệnh LOAD, STORE = 3 lần

### Bài 2:

Trong tập lệnh MIPS không có lệnh nhảy nếu so sánh bằng hoặc khác Zero (mặc dù lệnh này được dùng rất phổ biến). Làm thế nào để hiện thực lệnh này hiệu quả?

Trả lời: Chúng ta có thể so sánh với thanh ghi \$0

Ví dụ:

1. So sánh thanh ghi \$t0 với thanh ghi \$0, nếu bằng thì sẽ branch đến nhãn LABEL1:

beq \$t0, \$0, LABEL1

2. So sánh thanh ghi \$t0 với thanh ghi \$0, nếu không bằng thì sẽ branch đến nhãn LABEL2:

bne \$t0, \$0, LABEL2

### Bài 3:

Trình bày code trong file bai3.asm

### Bài 4:

#### \* Lý thuyết nền tảng

Nhóm chúng em lựa chọn sắp xếp chuỗi số nguyên bằng giải thuật Quicksort, bởi vì đây là một giải thuật sắp xếp hiệu quả với độ phức tạp trung bình là  $O(n)$ . Giải thuật này được hiện thực trong ngôn ngữ C++ như sau:

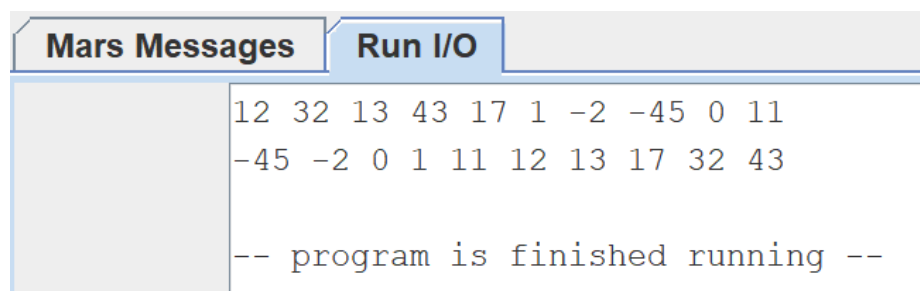
```
int partition(int *arr, int left, int right)
{
    int pivot = left; //chọn pivot là phần tử đầu tiên của mảng
    int i = left + 1, j = right;

    /*mục tiêu là sắp xếp pivot vào vị trí trong mảng sao cho bên trái là các phần tử nhỏ
    hơn pivot, bên phải là các phần tử lớn hơn pivot*/
    while (i <= j)
    {
        while (arr[i] < arr[pivot] && i <= j) i++;
        while (arr[j] >= arr[pivot] && i <= j) j--;
        if (i <= j)
        {
            swap(arr[i], arr[j]);
            i++; j--;
        }
    }
}
```

```
    }  
}  
swap(arr[left],arr[j]);  
//pivot đã được sắp xếp đúng vị trí của nó trong mảng  
return j;  
//trả về j là vị trí mới của pivot  
}  
void quicksort(int *arr, int left, int right)  
{  
    if (left < right) //nếu left = right tức là mảng chỉ có 1 phần tử, không sắp xếp  
    {  
        int j = partition(arr, left, right);  
        //thực hiện đệ quy cho 2 mảng bên trái và bên phải pivot  
        quicksort(arr, left, j-1);  
        quicksort(arr, j+1, right);  
    }  
}
```

**\* Kết quả:**

Kết quả nhóm thu được khi hiện thực trên MARS 4.5 được thể hiện qua Hình 1 và Hình 2



```
Mars Messages  Run I/O  
12 32 13 43 17 1 -2 -45 0 11  
-45 -2 0 1 11 12 13 17 32 43  
  
-- program is finished running --
```

Hình 1. Kết quả khi chạy đoạn mã hợp ngữ MIPS của nhóm

Mars Messages	Run I/O
	Ban dau: 12 32 13 43 17 1 -2 -45 0 11
	Sau do: 1 11 0 -45 -2 12 17 43 13 32
	Sau do: -45 -2 0 1 11 12 17 43 13 32
	Sau do: -45 -2 0 1 11 12 17 43 13 32
	Sau do: -45 -2 0 1 11 12 17 43 13 32
	Sau do: -45 -2 0 1 11 12 13 17 43 32
	Sau do: -45 -2 0 1 11 12 13 17 32 43
	-- program is finished running --

Hình 2. Kết quả chi tiết cách sắp xếp bằng Quicksort qua từng lần chọn pivot

Trình bày code trong file bai4.asm

**Bài 5:**

Trình bày code trong file bai5.asm

**Bài 6:**

Trình bày code trong file bai6.asm

**Bài 7:**

Trình bày code trong file bai7.asm