**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

Logo

Description automatically generated**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO BÀI TẬP LAB X-86**

**MÔN HỌC: HỆ THỐNG MÁY TÍNH**

**C  ASSEMBLY**

Lớp: 21\_4

Giảng viên: Chung Thùy Linh

| Nhóm sinh viên thực hiện |

Trần Anh Kiệt – 21120093

Trần Nam Phương – 21120313

Nguyễn Thành Trung – 21120351

Nguyễn Huy Hoàng – 21120458

Nguyễn Ngọc Như Huyền – 21120475

Thành phố Hồ Chí Minh, tháng 4 năm 2023

* **C to Assembly Translation**

## Dereferencing Modes

## 

Hình 1: Mã Code C

## 

Hình 2: Mã Code Assembly

**Q3**: There are subtle differences between the assembly generated for each function. What are they, and why are they different?

Có hai sự khác biệt chính trong 2 đoạn mã assembly của deref\_one và deref\_two

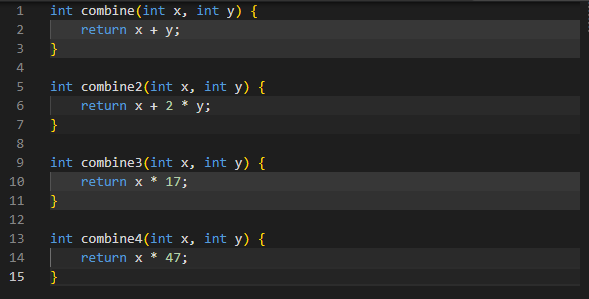
1. **Câu lệnh di chuyển dữ liệu:**

* deref\_one sử dụng câu lệnh **movb** vì kiểu dữ liệu của con trỏ là **char** có kích thước **1 byte.**
* deref\_two sử dụng câu lệnh **movl** vì kiểu dữ liệu của con trỏ là **int** có kích thước **4 byte.**
* Câu lệnh 1 có nghĩa: đưa giá trị 0 vào vùng nhớ có địa chỉ được lưu trong thanh ghi rdi (vùng nhớ được ptr trỏ tới).

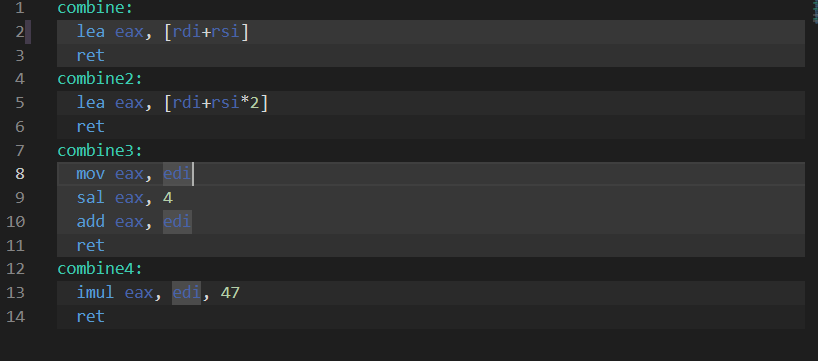
1. **Tính địa chỉ vùng nhớ ở câu lệnh 2:**

* deref\_one tính địa chỉ bằng cách: **ptr + index**. Vì mỗi phần tử kiểu **char** chiếm **1 byte** vùng nhớ nên không có giá trị scale.
* deref\_two tính địa chỉ bằng cách: **ptr + index \* 4**. Vì mỗi phần tử kiểu **int** chiếm **4 bytes** vùng nhớ nên có giá trị **scale = 4**.
* Câu lệnh 2 có nghĩa: đưa giá trị 0 vào vùng nhớ mới, được tính toán như trên.

## lea and Multiplication



Hình 3: Mã Code C



Hình 4: Mã Code Assembly

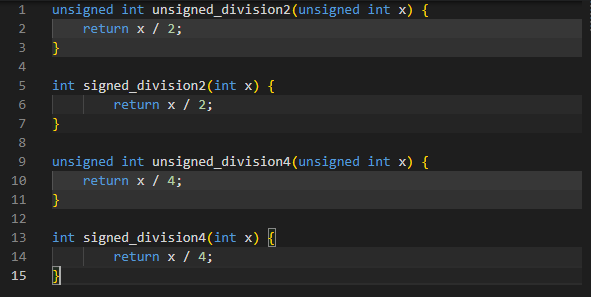
**Q4**: In combine1 and combine2, the compiler doesn't use add or multiply. How does it use lea instead?

* Lệnh **lea (Load Effective Address)** được sử dụng để thực hiện các phép tính toán mà không cần truy cập bộ nhớ, đồng thời nó cũng có thể thực hiện các phép tính toán kết hợp trong một lệnh duy nhất. Do đó, lệnh lea được sử dụng thay vì các lệnh add hay mul để giúp mã máy thực thi nhanh và hiệu quả hơn.
* Trong hàm combine1, thanh ghi %rdi chứa giá trị x và thanh ghi %rsi chứa giá trị y, lệnh lea dùng để cộng hai thanh ghi này và lưu kết quả vào thanh ghi %eax.
* Trong hàm combine2, thanh ghi %rdi chứa giá trị x và thanh ghi %rsi chứ giá trị y, lệnh lea sẽ nhân giá trị của %rsi với 2 rồi cộng với %rdi sau đó lưu kết quả vào thanh ghi eax.

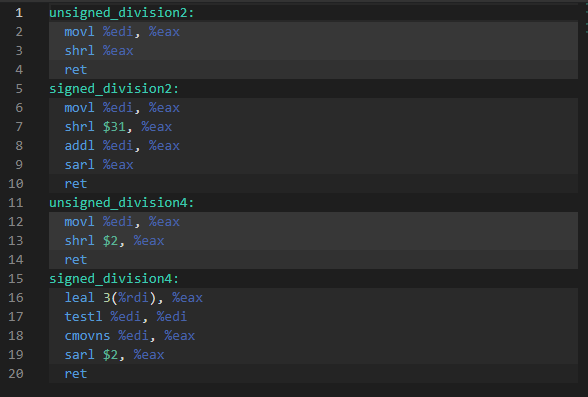
**Q5**: combine3 no longer fits the limited form of lea, but the compiler still avoids multiply. What does it use instead?

* Trong hàm combine3, trình biên dịch tránh sử dụng phép nhân bằng lệnh imull vì nó thực thi chậm hơn so với các lệnh có sẵn khác.
* Ta có 17 = 24 + 20, nên trình biên dịch có thể sử dụng kết hợp các phép cộng và dịch trái bit để nhân x với 17 mà không cần dùng tới imull thông qua các bước sau:
* Di chuyển x sang thanh ghi %eax bằng lệnh movl
* Dịch chuyển nội dung của %eax sang trái 4 bit (tương đương với phép nhân x với 16) bằng lệnh sall
* Thêm giá trị ban đầu của x vào kết quả của thao tác dịch chuyển bằng cách sử dụng lệnh addl (tương đương với x \* 16 + x = x \* 17)

## Division



Hình 5: Mã Code C

****

Hình 6: Mã Code Assembly

**Q6**: What does the assembly do to account for a fixup of 1 if the dividend is negative?

* Trình biên dịch sẽ cộng thêm 1 vào số bị chia trước khi thực hiện phép chia khi số bị chia là âm, chúng ta quan sát đoạn mã của hàm signed\_division2. Lúc này, x được lưu trữ trong thanh ghi edi. Ta chia đoạn code thành 2 phần như sau:
* Phần 1: mov và shr (dòng 6 và dòng 7). Trình biên dịch sử dụng lệnh mov để lưu giá trị của thanh ghi edi (giá trị x) vào thanh ghi eax. Khi sử dụng lệnh dịch phải luận lý 31 bit, eax (32 bit) lúc này sẽ mang bit dấu (bit trái nhất) của x. Như đã biết, khi chia lấy nguyên cho 2, nếu x là số không âm thì giữ nguyên giá trị, nếu x là số âm thì cần cộng thêm 1. Có như vậy, khi thực hiện phép dịch, kết quả sẽ làm tròn đến gần số 0. Việc lấy ra bit dấu của x sẽ thực hiện được điều này. Bit dấu là 0 đối với x không âm và là 1 đối với x âm. Vậy giá trị cần cộng thêm vào x trước khi dịch là giá trị bit dấu.
* Phần 2: add và sar (dòng 8 và dòng 9). Sử dụng lệnh add, ta có eax = eax + edi = bit dấu của x + x. Sau đó, sử dụng lệnh dịch phải số học sar để dịch eax sang phải 1 bit và giữ lại bit dấu. Kết quả thu được như đã nói ở cuối phần 1 là kết quả làm tròn gần số 0 nhất.