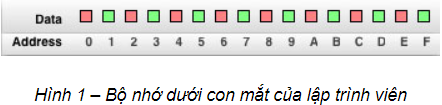
**Memory access granualarity**

Bộ nhớ:

­­­­

CPU không đọc (ghi) dữ liệu từ **memory** theo từng byte >> theo từng **block**

* Block

Kích thước có thể là 2, 4, 8, 16 hoặc 32 bytes >> **memory access granularity**

* Địa chỉ

Là vị trí bắt đầu của một block

+ có giá trị chia hết cho kích thước của 1 block >> **aligned address**

+ ngược lại >> **unaligned address**

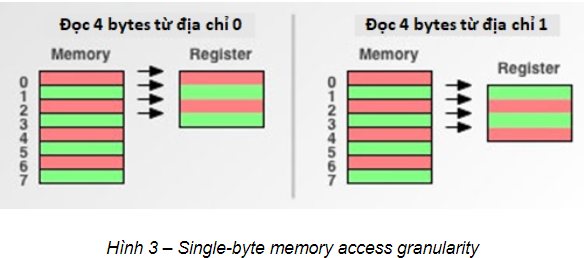
Ex: **MAG** = 4; các địa chỉ là 0, 4, 8 là **aligned** address; 1,3,2 là **unaligned** address

* **Nguyên tắc cơ bản của Data Alignment**

Minh họa:

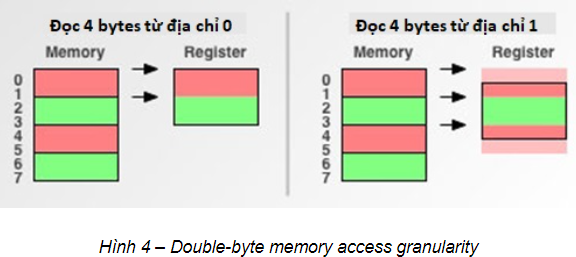
Nhiệm vụ: đầu tiên dọc 4 bytes từ địa chỉ 0 vào thanh ghi của CPU. Tiếp theo đọc 4 bytes từ địa chỉ 1 vào cùng thanh đó

* **MAG** = 1 bytes



Cả 2 TH (đọc từ 0 or 1) CPU đều phải mất **4 lần** truy cập vào memory

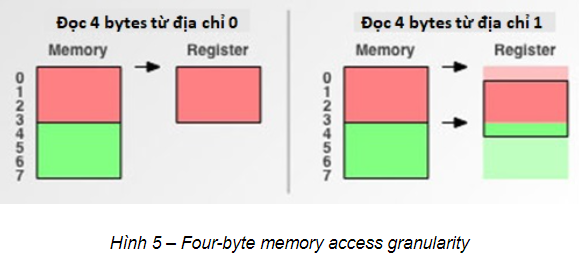
* **MAG** = 2 bytes



Khi đọc địa chỉ 0, CPU chỉ cần **2 lần** truy cập **memory** (bằng ½ so với single-byte)

Khi đọc địa chỉ 1 (**unaligned address**) >> CPU cần làm việc nhiều hơn. CPU phải truy cập **memory** đến **3 lần**: đọc 2 bytes từ 0, 2, 4. Sau đó thực hiện phép toán lấy ra 4 bytes cần thiết trong 6 bytes đã đọc ra >> hoạt động chậm hơn

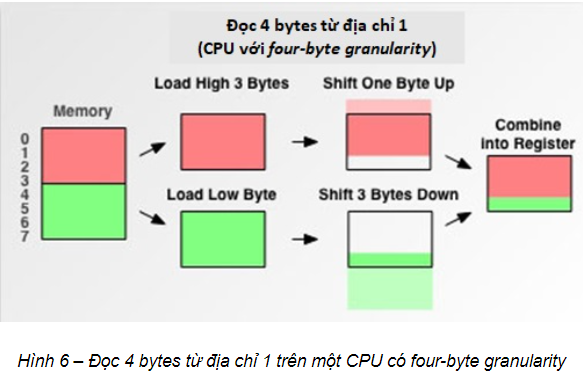
* **MAG** = 4



Từ địa chỉ 0 chỉ **1 lần** đọc

Từ địa chỉ 1 cần **2 lần** truy cập và thực hiện phép toán lấy ra 4 bytes cần thiết

* **CPU xử lý ntn khi đọc dữ liệu từ “unaligned address”**

****

CPU đọc một block (4 bytes) từ địa chỉ 0

>> loại bỏ các byte “không mong muốn”

>> đọc block thứ hai từ địa chỉ 4

>> loại bỏ các byte “không mong muốn”

>> cả hai phần được tổng hợp lại thành 4 bytes cần đọc để đặt vào trong thanh ghi

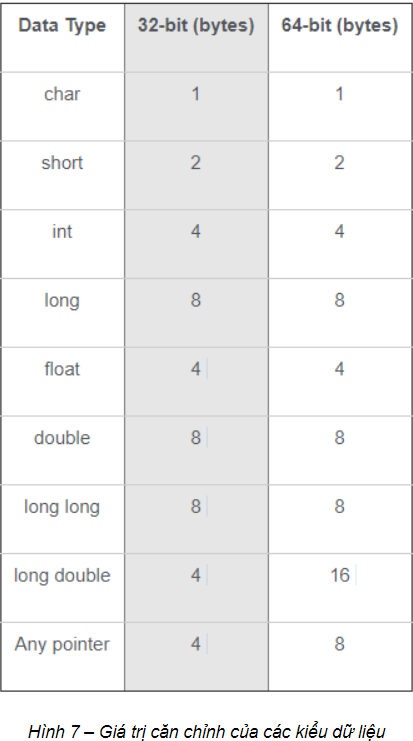
Có khá nhiều việc cần làm. Tuy nhiên một số CPU không hỗ trợ làm tất cả những công việc đó cho chúng ta.

* **Data structure Alignment trong C/C++**

Để tránh CPU phải thực hiện thêm các xử lý khi truy cập **unaligned** addresses thì trình biên dịch C/C++ tự động thực hiện alignment data trong quá trình biên dịch >> đảm bảo cho CPU luôn truy cập dữ liệu từ **aligned** addresses.

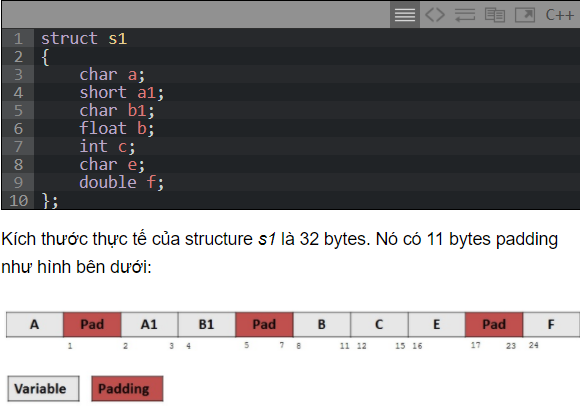
Các cấu trúc dữ liệu (**data structure**) thường có nhiều phần tử dữ liệu thuộc các kiểu dữ liệu khác nhau với các giá trị căn chỉnh (**alignment requirement**) khác nhau. Trình biên dịch sẽ cố gắng duy trì việc căn chỉnh của các phần tử dữ liệu bằng cách chèn các ô nhớ không sử dụng giữa các phần tử. Kỹ thuật này được gọi là “**Padding**”. Ngoài ra, trình biện dịch cũng thực hiện căn chỉnh toàn bộ structure bằng cách bổ sung **padding** vào cuối của structure, kỹ thuật này gọi là “**Tail Padding**”.

>> Mỗi phần tử của một mảng các structure được căn chỉnh hợp lý.



**Padding** sẽ được chèn vào trong structure khi một phần tử của structure được theo sau bởi một phần tử khác có giá trị căn chỉnh lớn hơn hoặc ở phần cuối của structure. Nguyên tắc:

* Khoảng cách từ một phần tử dữ liệu trong structure đến đầu của structure (tính theo byte) phải chia hết cho giá trị căn chỉnh của phần tử dữ liệu đó. Ví dụ: một phần tử dữ liệu có kiểu *int*trong structure thì khoảng cách từ phần tử đó đến đầu structure phải chia hết cho 4.
* Kích thước của toàn bộ structure phải đảm bảo chia hết cho giá trị căn chỉnh lớn nhất trong structure đó.



**Trong trường hợp struct rỗng (không chứa thuộc tính nào) thì kích thước của nó sẽ là 1**