**Bài làm thêm 2:**

a) Theo kiến trúc BIG ENDIAN, các byte trong một từ 32 bit được lưu trữ theo thứ tự từ cao đến thấp. Do đó, nội dung của vùng nhớ dữ liệu trong chương trình như sau:

0xCA002018

0xFF

0x2018

0xCA

0xFE

0xED

Các lệnh gây ra lỗi khi thực thi là:

- Lệnh lh $t2, 2($a0): Lệnh này truy cập vào địa chỉ int\_1 + 2, nhưng int\_1 chỉ có 4 byte, do đó địa chỉ này không hợp lệ.

- Lệnh lh $t3, 3($a0): Lệnh này cũng truy cập vào địa chỉ int\_1 + 3, nhưng int\_1 chỉ có 4 byte, do đó địa chỉ này cũng không hợp lệ.

- Lệnh lb $t4, 0($a0): Lệnh này truy cập vào địa chỉ int\_1 + 0, nhưng int\_1 có độ dài 4 byte, do đó chỉ có 4 byte đầu tiên của int\_1 là hợp lệ. Lệnh này sẽ đọc 4 byte đầu tiên của int\_1 và lưu vào t4.

- Lệnh lb $t5, 1($a0): Lệnh này cũng truy cập vào địa chỉ int\_1 + 1, nhưng chỉ có 3 byte còn lại của int\_1 là hợp lệ. Lệnh này sẽ đọc 3 byte còn lại của int\_1 và lưu vào t5.

Để cải thiện hiệu năng truy cập bộ nhớ, chúng ta có thể sắp xếp lại dữ liệu như sau:

char\_1: .byte 0xFF

int\_1: .word 0xCA002018

int\_2: .word 2018

char\_2: .byte 0xCA 0xFE 0xED

Với cách sắp xếp này, tất cả các dữ liệu đều được lưu trữ ở các địa chỉ bộ nhớ hợp lệ. Ngoài ra, các dữ liệu cùng loại cũng được lưu trữ gần nhau, giúp cho bộ xử lý có thể truy cập chúng nhanh hơn.

Cụ thể, các lệnh lw $t0, 0($a0) và lw $t1, 1($a0) sẽ đọc hai từ 32 bit đầu tiên của vùng nhớ dữ liệu. Lệnh lh $t2, 2($a0) sẽ đọc hai byte đầu tiên của int\_2. Lệnh lh $t3, 3($a0) sẽ đọc hai byte còn lại của int\_2. Lệnh lb $t4, 0($a0) sẽ đọc byte đầu tiên của char\_1. Lệnh lb $t5, 1($a0) sẽ đọc byte thứ hai của char\_1.

Với cách sắp xếp này, tất cả các lệnh đều truy cập vào các địa chỉ bộ nhớ hợp lệ và có thể được thực thi mà không gây ra lỗi.