***1. Phần giải thích bài làm***

***a) Xử lý đối với số thập phân:***

* Trước hết nhập 2 số đầu vào là $s0,$s1. Ta đi thực hiện việc lấy giá trị tuyệt đối của chúng (nếu số đó âm thì lấy 0 trừ đi số đó), và xét phần dấu của chúng lưu vào $t0,$t1 (bit 0 : âm, bit 1 : dương).
* Thực hiện hàm DIV, đầu vào là 2 giá trị tuyệt đối vừa tìm được ở trên, vì giá trị tuyệt đối nên dĩ nhiên tối đa chỉ có 31 bits (vì đầu vào là số có dấu 32bits). Ở hàm DIV thực hiện thuật toán phép chia tuần tự tối ưu (được trình bày ở textbook). Kết quả trả về là phần thương $v0, dư $v1
* Tiếp đó ta đi thực hiện nhãn Print\_RESULT\_DIV để thực hiện việc in đúng chuẩn đầu ra là phần nguyên và phần được in theo số nguyên có dấu. Ta xét dấu của thương = (dấu của số bị chia ) xor (dấu của số chia), dấu của phần dư = (dấu phần thương) xor (dấu của số chia) (cách xét dấu tham khảo ở cuốn CS422ComputerArchitectureComputerOrganizationAndDesign5thEdition2014, ở cuốn này dấu phần dư có thể là số âm (thay vì ta dùng cơ sở lý thuyết của Euclid thì dấu phần dư luôn dương <https://vi.wikipedia.org/wiki/Ph%C3%A9p_chia_c%C3%B3_d%C6%B0)>).

Vì là phép chia nên phần trị tuyệt đối của phần thương và phần dư chỉ có tối đa 31 bits)

* Tiếp theo ta thực hiện phép nhân là hàm Multiply, đầu vào là 2 thanh ghi là 2 số lấy trị tuyệt đối như đã nói ở trên đưa vào $a0,$a, đầu ra là là thanh ghi có giá trị là $v0=a,$v1=b, biểu diễn số z.

Số có dạng

(Phần này tham khảo thuật toán nhân tuần tự tối ưu trong textbook)

Sau đó ta thực hiện nhãn Print\_RESULT\_MUL để in kết quả phép nhân.

Thực tế ta chỉ thực hiện được phép nhân tối đa với 2 số trị tuyệt đối 31 bit là

đúng theo chuẩn của số z nêu ở trên, ta nhận xét rằng, a= 230-1 thì chỉ có tối đa 30 bits khi biểu diễn số dương. Cho nên thực tế số a sẽ có tối đa 30 bits và số b có tối đa 32 bits. Để có thể áp dụng phép chia ở trên trong lúc tính toán ta cần đưa mỗi số về dạng số nguyên không dấu 31 bits tức là (a,b)=(30 bits;32 bits)=(31 bits, 31 bits). Phần này dễ dàng thực hiện bằng các phép dịch bits.

Khi đó (a, b <= 231 - 1)

Sau khi chuyển đổi xong ta chưng minh được phép lặp sau.

DO\_Multiply:

Ta áp dụng phép chia phân tích được: a = 10x + y; b = 10u + v

(với x, y, u, v >= 0; y, v < 10)

Ta tính

Ta thấy về lý thuyết

Điều này chứng tỏ phần nguyên của trở thành phần 31 bits thấp

trong phép lặp. Còn phần dư sẽ là các ký số của kết quả nhân này, ta lưa vào mảng $s7. Đặt , Ta tiếp tục biểu diễn số z như vậy và

tiếp tục vòng lặp. Điều kiện dừng khi a = 0 and b = 0.

Dấu của phép nhân thì ta dễ thấy là bằng (dấu của số bị nhân) xor (dấu của số nhân). Và ta in mảng $s7 theo thứ tự ngược lại thì ta được kết quả phép nhân.

***b) Chuyển Hex sang Dec:***

Ở phần nhân định dạng số hệ 10, việc in kết quả 2 thành ghi 32 bits thành số nguyên 64 bits khá là phức tạp. Thì phần nhân và chia định dạng hệ 16, đòi hỏi quá trình đổi một chuỗi kí tự hợp lệ thành số nguyên lưu trong thanh ghi và ngược lại. Tuy nhiên, điều này khá đơn giản khi chúng ta thao tác trên từng bit dữ liệu. Cụ thể:

* Chuyển số hệ 10 sang hệ 16
* Chuyển cơ số 16 sang cơ số 10:

Để trình bày đẹp mắt và ngắn gọn code, thì ở phần hàm chuyển cơ số 16 sang cơ số 10 đỏi hỏi đầu vào chúng ta phải hợp lệ (kí tự chữ phải là in hoa).

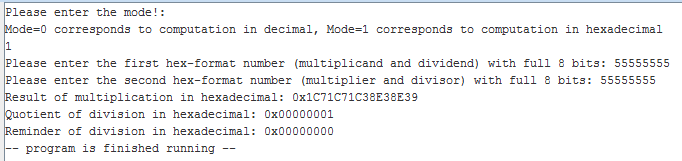
Một chữ số hệ 16 được biểu diễn bởi 4 bit (0->F tương ứng với 0000 -> 1111: 0 -> 15 ở dạng cơ số 10 ) . Mặt khác, ở dạng kí tự ‘0’ -> ‘9’ được quy ra 48 -> 57, ‘A’-> ‘F’ được qui ra 65->70 ở cơ số 10.

Hay nói cách khác, để chuyển số dạng hex sang dec thì chúng ta chỉ cần xét từng bytes của string chứ hex. Nếu bytes đó có giá trị từ 48->57 thì ta trừ cho 48, còn nếu nằm trong 65->70 thì ta trừ cho 55. Sau đó dịch trái 4 bit ở kết quả rồi cộng trực tiếp kết quả đã tính lên vào cho đến khi đến cuối string ‘\0’.

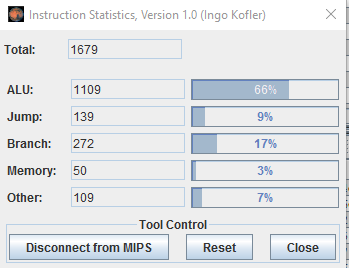
Như vậy ta đã có được số hệ 10 từ hệ 16

***2. Phần thống kê lệnh***

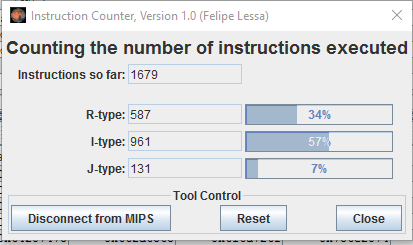
Ứng với input:



***a) Thống kê số lệnh***

******

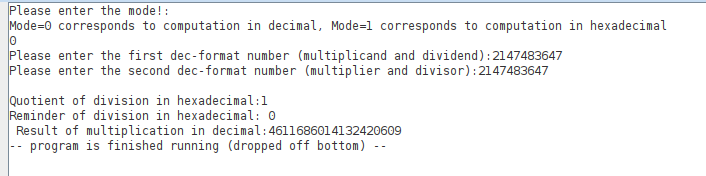
***b) Thống kê loại lệnh***

******

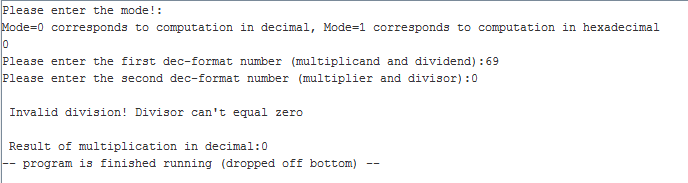
***3. Phần tính toán***

***4. Phần Test***

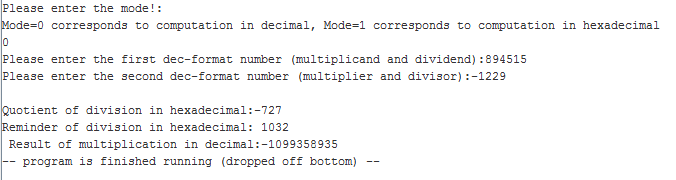
Test 1:



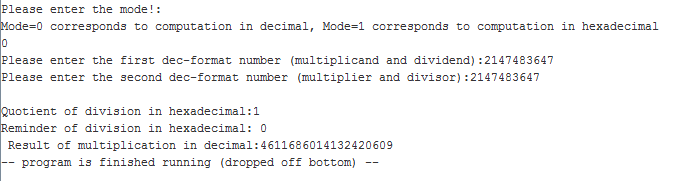
Test 2:



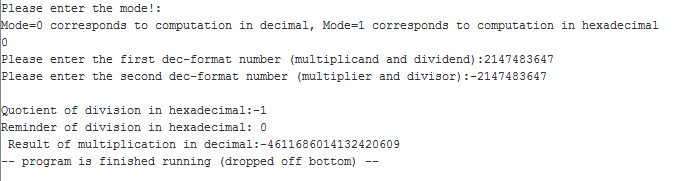
Test 3:



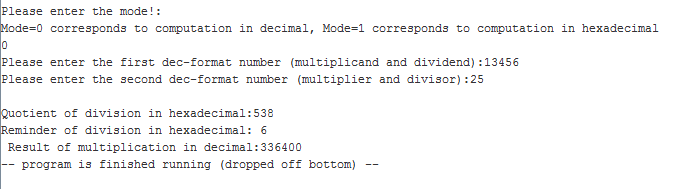
Test 4:



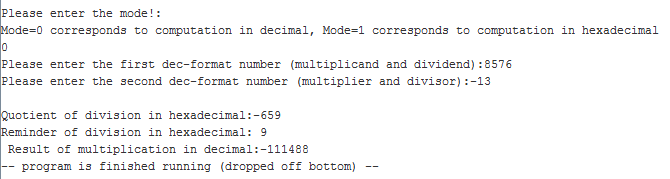
Test 5:



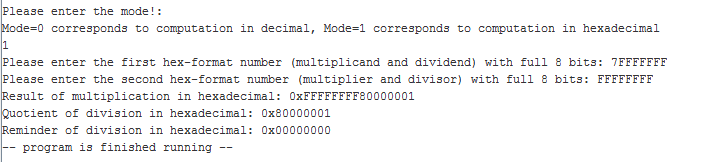
Test 6:



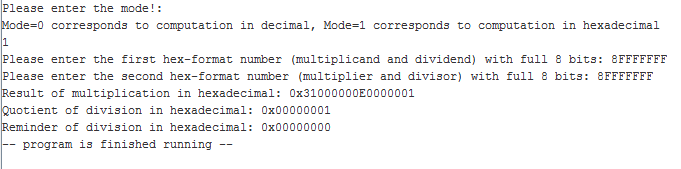
Test 7:



Test 8:



Test 9:



Test 10:

