

# MỘT SỐ VÍ DỤ LẬP TRÌNH HỢP NGỮ LC-3

Trần Quốc Tiến Dũng

Bộ môn Điều khiển tự động, Khoa Điện – Điện tử

Trường Đại học Bách Khoa, ĐHQG TP.HCM

## 1. Nhắc lại về tập lệnh LC-3

Tập lệnh LC-3 (LC-3 ISA) bao gồm 15 lệnh. Mỗi lệnh có chiều dài là 16 bit bao gồm 4 bit đầu là mã lệnh (opcode) và 12 bit sau là các toán hạng (operands). Tùy thuộc vào lệnh mà có thể có 1 hoặc nhiều toán hạng.

Có 5 cách định vị địa chỉ trong LC-3:

- Tức thời (immediate)
- Thanh ghi (register)
- PC-relative
- Gián tiếp (indirect)
- Base + offset

Có 3 nhóm lệnh trong LC-3, đó là nhóm lệnh thực thi (NOT, ADD, AND), nhóm lệnh di chuyển dữ liệu (LD, ST, LDI, STI, LDR, STR, LEA) và nhóm lệnh điều khiển (BR, JMP/RET, JSR/JSRR, RTI, TRAP).

### a. Nhóm lệnh thực thi:

Chỉ bao gồm 3 lệnh là NOT, ADD, AND.

Lệnh\Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Lệnh hợp ngữ	Ghi chú
NOT	1	0	0	1	DR		SR			1	1	1	1	1	1		NOT Rx,Ry	Hai thanh ghi Rx, Ry có thể trùng nhau hoặc khác nhau đều được
ADD	0	0	0	1	DR		SR1			0	0	0	SR2			ADD Rx,Ry,Rz		
ADD	0	0	0	1	DR		SR			1	Imm5					ADD Rx,Ry,#imm5	Thành phần tức thời có tầm trị từ 16 đến +15.	
AND	0	1	0	1	DR		SR1			0	0	0	SR2			AND Rx,Ry,Rz		
AND	0	1	0	1	DR		SR			1	Imm5					AND Rx,Ry,#imm5	Thành phần tức thời có tầm trị từ 16 đến +15.	

Ví dụ 1: Trừ hai thanh ghi R0 và R1, lưu kết quả vào thanh ghi R0.

NOT R1,R1

ADD R1,R1,#1

ADD R0,R0,R1

Ví dụ 2: Thực hiện phép OR hai thanh ghi R0 và R1, lưu kết quả vào thanh ghi R0.

NOT R0,R0

NOT R1,R1

AND R0,R0,R1

NOT R0,R0

Ví dụ 3: Chuyển dữ liệu từ thanh ghi R0 sang thanh ghi R4

Cách 1: ADD R4,R0,#0

Cách 2: AND R4,R0,#-1

Ví dụ 4: Khởi tạo thanh ghi R2 với giá trị ban đầu là 10.

AND R2,R2,0

ADD R2,R2,#10

b. Nhóm lệnh di chuyển dữ liệu:

Có 2 loại di chuyển dữ liệu: lấy dữ liệu từ bộ nhớ nạp vào thanh ghi (LOAD) và lấy dữ liệu từ thanh ghi lưu vào bộ nhớ (STORE). Như vậy các lệnh di chuyển dữ liệu sẽ đi theo cặp. Tùy thuộc vào cách định vị địa chỉ sẽ có một lệnh riêng.

Mã lệnh:

<b>LD<sup>+</sup></b>	0010	DR	PCoffset9
<b>LDI<sup>+</sup></b>	1010	DR	PCoffset9
<b>LDR<sup>+</sup></b>	0110	DR	BaseR offset6
<b>ST</b>	0011	SR	PCoffset9
<b>STI</b>	1011	SR	PCoffset9
<b>STR</b>	0111	SR	BaseR offset6
<b>LEA<sup>+</sup></b>	1110	DR	PCoffset9

Cấu trúc lệnh:

Kiểu định vị địa chỉ	Lệnh LOAD tương ứng	Lệnh STORE tương ứng	Ghi chú
PC-relative	LD Rx, PCoffset9 $Rx \leftarrow M[PC + PCoffset9]$	ST Rx, PCoffset9 $M[PC + PCoffset9] \leftarrow Rx$	Tầm tham chiếu tối đa là 512 byte xung quanh giá trị PC.
Indirect	LDI Rx, PCoffset9 $Rx \leftarrow M[M[PC + PCoffset9]]$	STI Rx, PCoffset9 $M[M[PC + PCoffset9]] \leftarrow Rx$	Tầm tham chiếu là toàn bộ bộ nhớ 64K.
Base + offset	LDR Rx, Ry, offset6 $Rx \leftarrow M[Ry + offset6]$	STR Rx, Ry, offset6 $M[Ry + offset6] \leftarrow Rx$	

Lưu ý:

- Các phép cộng trong LC-3 đều là số bù 2, vì vậy trước khi cộng phải mở rộng dấu. Đặc biệt lưu ý các trường hợp giá trị PCoffset9 và offset6 là số âm.
- Sau pha lấy lệnh, giá trị PC đã được tăng thêm 1.

Lệnh LEA không có tác dụng di chuyển dữ liệu, tuy nhiên giúp sinh ra giá trị địa chỉ để tham chiếu đến.

Các bạn có thể tham khảo các ví dụ trong giáo trình.

### c. Nhóm lệnh điều khiển:

Trong chương trình môn học, các bạn chỉ cần chú ý đến 2 lệnh điều khiển chính là lệnh rẽ nhánh (BR) và lệnh TRAP. Các lệnh còn lại các bạn có thể xem thêm trong giáo trình.

LC-3 cung cấp 3 mã điều kiện (condition code) là N, Z, P. Sau mỗi lệnh có lưu dữ liệu vào thanh ghi DR, các mã điều kiện này sẽ thay đổi phụ thuộc vào kết quả trong thanh ghi DR vừa được lưu. Nếu kết quả là số dương thì cờ P được bật (P – positive), các cờ còn lại tắt. Tương tự cho trường hợp cờ Z (zero) và cờ N (negative).

Ví dụ 5: Sau lệnh AND R1, R1, #0 thì dữ liệu được ghi vào thanh ghi DR là 0, như vậy cờ Z được bật lên.

Ví dụ 6: Giả sử thanh ghi R1 đang chứa dữ liệu là #8, sau lệnh ADD R1, R1, #-10 thì dữ liệu trong thanh ghi DR là #-2 là số âm, như vậy cờ N sẽ được bật lên.

Ví dụ 7: Sau lệnh SR R2, #-10 thì các cờ điều kiện sẽ không thay đổi. Lý do là vì lệnh ST không lưu dữ liệu vào thanh ghi.

Trong LC-3, những lệnh có thủ tục lưu dữ liệu vào thanh ghi đó là NOT, AND, ADD, LD, LDR, LDI, LEA, những lệnh còn lại chắc chắn không làm thay đổi các mã điều kiện. Dựa vào các mã điều kiện này, chúng ta có thể thiết lập rẽ nhánh tùy vào yêu cầu của đề. Lệnh rẽ nhánh là lệnh BR (opcode 0000). Ba bit tiếp theo để xác định điều kiện (n, z, p), 9 bit còn lại xác định độ dời PCoffset. Nếu bit 11 (bit ứng với mã lệnh n) bằng 1, PC sẽ nhảy khi cờ N được bật lên. Tương tự khi bit 10 và bit 9 bằng 1. Có thể có nhiều hơn 1 bit trong 3 bit n, z, p bằng 1, khi đó điều kiện sẽ là “hoặc”.

Ví dụ 8: Mã lệnh 0000 011 000001000 có ý nghĩa là nếu cờ Z hoặc cờ P bằng 1 thì câu lệnh tiếp theo thực thi sẽ nằm ở địa chỉ PC + #8.

Ta có bảng lệnh hợp ngữ tương ứng:

n	z	p	Mã lệnh hợp ngữ	Ghi chú
0	0	0		Lệnh này không có ý nghĩa thực thi (tương ứng với lệnh NOP – No Operation)
0	0	1	BRp PCoffset9	
0	1	0	BRz PCoffset9	
1	0	0	BRn PCoffset9	
0	1	1	BRzp PCoffset9	
1	0	1	BRnp PCoffset9	
1	1	0	BRnz PCoffset9	
1	1	1	BRnzp PCoffset9	Lệnh này sẽ luôn nhảy, không quan trọng các cờ điều kiện. Khi lập trình hợp ngữ có thể dùng BR thay cho BRnzp.

Ví dụ 9: Viết chương trình bắt đầu từ x3010, kiểm tra nếu R0 dương thì nhảy đến câu lệnh ở địa chỉ x3020.

x3010: ADD R0, R0, #0

x3011: BRp x0E

Nhận xét: Bởi vì ta chưa biết các cờ điều kiện trước đó ứng với thanh ghi nào cả, vì vậy không thể đặt lệnh x3010: BRp x0F được. Phải gọi 1 lệnh để set các cờ điều kiện dựa vào thanh

ghi R0 trước, sau đó mới kiểm tra các cờ dùng lệnh BRp. Câu lệnh ở x3010 không có ý nghĩa thực thi, tuy nhiên có ý nghĩa set các cờ điều kiện theo thanh ghi mong muốn.

Ví dụ 10: So sánh hai thanh ghi R0 và R1, nếu thanh ghi R0 lớn hơn hoặc bằng R1 thì nhảy đến lệnh ở địa chỉ x3020. Chương trình bắt đầu từ x3050.

Phân tích: để so sánh ta sẽ phải dùng phép trừ hai thanh ghi.

x3050: NOT R1, R1

x3051: ADD R1, R1, #1

x3052: ADD R0, R1, R1

Phân tích: sau lệnh vừa rồi, thanh ghi kết quả R0 đó chính là hiệu của R0 và R1 trước đó, vì vậy ta không cần phải gọi lệnh như ví dụ 9 nữa. Tiếp theo sẽ là lệnh x3053: BRzp PCoffset9.

Để tính PC offset, ta cần tính độ dời bằng cách trừ thanh ghi đích cho thanh ghi PC. Chú ý là sau pha lấy lệnh thì PC được tăng thêm 1, như vậy  $PC = x3054$ .

Lúc này  $PCoffset = x3020 - x3054 = xFFCC$  (bỏ qua việc bị tràn số), rút gọn lại 9 bit (vì lệnh này chỉ có 9 bit cho phần offset) là x1CC:

x3053: BRzp x1CC

Một cách khác các bạn tính phép tính trên sẽ thu được  $PCoffset = x-34$ , như vậy có thể viết lệnh:

x3053: BRzp x-34

Từ câu lệnh rẽ nhánh này, ta có thể sử dụng phương pháp lặp để giảm bớt khối lượng lập trình. Có hai phương pháp để lặp, cách thứ nhất là dùng biến đếm (tương ứng với vòng lặp for mà các bạn đã được học trước đây), cách còn lại là dùng trị canh (tương ứng với vòng lặp while). Các bạn có thể tham khảo thêm trong giáo trình. Trong bài viết này trình bày hai ví dụ về hai cách lặp.

Ví dụ 11: Viết đoạn chương trình bắt đầu từ x3020 thực hiện nhân giá trị trong thanh ghi R0 với 8, sau đó lưu vào thanh ghi R1 và nhảy đến câu lệnh tại địa chỉ x3080.

Phân tích: đầu tiên ta sẽ khởi tạo thanh ghi R2 bằng 8, thanh ghi R1 bằng 0. Sau đó mỗi bước ta cộng vào thanh ghi R1 bằng với giá trị của R0 và giảm R2 đi 1 đơn vị. Đến khi R2 vừa bằng 0 thì nhảy đến câu lệnh ở x3080.

Địa chỉ	Lệnh hoặc dữ liệu	Giải thích
x3020	AND R1, R1, #0	Khởi tạo R1 = #0
x3021	ADD R2, R1, #8	Khởi tạo R2 = #8
x3022	ADD R1, R1, R0	
x3023	ADD R2, R2, #-1	
x3024	BRp x-3	Nếu R2 còn dương thì nhảy về câu lệnh ở x3022
x3025	BR x5A	Ngược lại, nhảy đến câu lệnh tại x3060

Ví dụ 12: Cho một chuỗi kí tự bắt đầu tại địa chỉ x3100, viết chương trình đếm số kí tự của chuỗi đó. Cho biết dấu hiệu nhận biết chuỗi kí tự là kí tự có mã ASCII bằng 0 và không tính kí tự này vào chiều dài của chuỗi. Đoạn chương trình viết bắt đầu tại x3020, sau khi đếm xong thì nhảy đến địa chỉ x3080.

Địa chỉ	Lệnh hoặc dữ liệu	Giải thích
x3020	LEA R1, x7F	Tạo địa chỉ đầu R1 = x3100
x3021	AND R2, R2, #0	Khởi tạo biến đếm R2 = #0
x3022	LDR R0, R1, #0	Lấy kí tự
x3023	BRz x5C	Nếu kí tự bằng 0, ngừng đếm và nhảy đến x3080
x3024	ADD R2, R2, #1	Nếu kí tự khác 0, tăng biến đếm thêm 1, tăng địa chỉ thêm 1 và nhảy về lại câu lệnh lấy kí tự
x3025	ADD R1, R1, #1	
x3026	BR x-5	

Lệnh cuối cùng là lệnh TRAP (opcode 1111). Mã lệnh này sẽ gọi các thủ tục có sẵn trong chương trình. Có 6 mã chương trình được hỗ trợ trong phần mềm Simulator.exe của LC-3.

Trap vector	Lệnh hợp ngữ	Lệnh hợp ngữ rút gọn	Chức năng
x20	TRAP x20	GETC	Đọc 1 kí tự từ bàn phím, kí tự này <i>không</i> hiện ra màn hình. Mã ASCII của kí tự vừa nhập được lưu vào thanh ghi R0.
x21	TRAP x21	OUT	Xuất kí tự từ thanh ghi R0 (mã ASCII) ra màn hình.
x22	TRAP x22	PUTS	Xuất mảng kí tự ra màn hình, địa chỉ bắt đầu của mảng là thanh ghi R0. Mỗi ô nhớ tương ứng với 1 kí tự. In đến khi xuất hiện kí tự bằng 0 trong mảng.
x23	TRAP x23	IN	Đọc 1 kí tự từ bàn phím, kí tự này <i>được</i> hiện ra màn hình. Mã ASCII của kí tự vừa nhập được lưu vào thanh ghi R0.
x24	TRAP x24	PUTSP	Xuất mảng kí tự ra màn hình, địa chỉ bắt đầu của mảng là thanh ghi R0. Mỗi ô nhớ tương ứng với 2 kí tự. Kí tự in đầu nằm ở bit [15:8], kí tự sau nằm ở bit [7:0]. In đến khi xuất hiện kí tự bằng 0 trong mảng.
x25	TRAP x25	HALT	Dừng chương trình

Ví dụ 13: Viết chương trình nhập 2 kí tự số (từ 0 đến 9) từ bàn phím, tính tổng 2 số và xuất ra màn hình.

Địa chỉ	Lệnh hoặc dữ liệu	Giải thích
x3000	IN	Nhập kí tự thứ nhất
x3001	LD R1, x13	Lấy hằng số chuyển từ mã ASCII sang số và lưu vào R1.
x3002	NOT R1, R1	Chuyển thành số âm
x3003	ADD R1, R1, #1	
x3004	ADD R2, R0, R1	Lưu kết quả số vừa nhập vào R2, để thanh ghi R0 nhận kí tự mới
x3005	IN	Nhập kí tự thứ hai
x3006	ADD R2, R0, R2	Cộng hai số vừa nhập với nhau.
x3007	ADD R2, R2, R1	
x3008	ADD R3, R2, #-10	Kiểm tra xem tổng có lớn hơn 10 hay không, nếu có thì nhảy đến câu lệnh x3011
x3009	BRn x7	

x300A	LD R0, xA	Xuất lần lượt hai ký tự
x300B	ADD R0, R0, #1	
x300C	OUT	
x300D	LD R0, x7	
x300E	ADD R0, R0, R3	
x300F	OUT	
x3010	BR x3	Nhảy đến ô x3014
x3011	LD R0, x3	Xuất 1 ký tự.
x3012	ADD R0, R0, R2	
x3013	OUT	
x3014	HALT	Dừng chương trình
x3015	x0030	Hằng số chuyển từ mã ASCII sang số (x30 = #48)

## 2. Lập trình hợp ngữ LC-3

### a. Lệnh hợp ngữ trong LC-3:

Một lệnh hợp ngữ trong LC-3 bao gồm 4 thành phần:

Nhãn	Mã lệnh	Toán hạng	; Chú thích
------	---------	-----------	-------------

Trong đó phần bắt buộc phải có là mã lệnh. Tùy vào lệnh là gì mà có thể có hoặc không có toán hạng. Hai phần “Nhãn” và “Chú thích” có thể có hoặc không. Trong đó chú thích sử dụng để giải thích ý nghĩa của một lệnh hoặc một đoạn lệnh. Giúp cho người đọc dễ dàng nắm bắt ý tưởng của người lập trình. Bắt buộc trước chú thích phải là dấu chấm phẩy (;).

#### Ví dụ 14: Câu lệnh

LAP ADD R1, R1, #-1 ; giảm R1 xuống 1 đơn vị

Trong đó:

- LAP là nhãn của lệnh
- ADD là mã lệnh
- R1, R1, #-1 là các toán hạng của lệnh tương ứng
- ; giảm R1 xuống 1 đơn vị là chú thích về ý nghĩa của lệnh

Ví dụ 15: Câu lệnh IN chỉ có mã lệnh, không có các thành phần còn lại.

### b. Nhãn:

Nhãn là các tên tượng trưng được dùng để xác định các ô nhớ được tham khảo tới trong chương trình. Trong hợp ngữ LC-3, một nhãn có thể được tạo từ một tới 20 ký số hay ký tự, và bắt đầu bằng một ký tự, như LAPLAI, KETTHUC, LAP100,....

Có hai lý do cần cho việc tham khảo một vị trí trong bộ nhớ, đó là

- Ô nhớ vị trí đó chứa đích của một lệnh rẽ nhánh.
- Ô nhớ vị trí đó chứa một giá trị cần được nạp hay lưu.

Ví dụ 16: Giả sử rằng các cờ điều kiện đã được set theo thanh ghi R0, viết đoạn chương trình tiếp theo kiểm tra nếu R0 lớn hơn 0 thì thực hiện cộng R1 thêm 1 đơn vị, nếu R0 bé hơn 0 thì cộng R2 thêm 1 đơn vị, nếu R0 bằng 0 thì cộng R3 thêm 1 đơn vị, sau đó dừng chương trình.

```

...
BRp      LON_HON_0
BRn      BE_HON_0
ADD      R3,R3,#1
BR       KET_THUC
LON_HON_0 ADD      R1,R1,#1
BR       KET_THUC
BE_HON_0 ADD      R2,R2,#1
KET_THUC HALT

```

Như vậy việc dùng nhãn sẽ giúp cho bộ hợp dịch tự động tính các giá trị offset cho các lệnh có chứa giá trị này (ví dụ lệnh BR, LD, LEA, ...). Người lập trình sẽ tránh được việc phải tự tính tay các độ dời, đặc biệt là tránh việc phải tính lại độ dời khi thêm các dòng lệnh vào giữa chương trình.

### c. Các mã giả:

Bộ hợp dịch LC-3 là một chương trình lấy đầu vào là chuỗi ký tự biểu diễn một chương trình được viết bằng hợp ngữ LC-3, và dịch nó ra thành một chương trình ở cấp kiến trúc tập lệnh (ISA) của LC-3.

Mã giả (pseudo-ops) giúp cho bộ dịch thực hiện nhiệm vụ này, còn được gọi bằng một tên khác là hướng dẫn dịch (assembler directives).

Bộ hợp dịch LC-3 gồm năm mã giả: .ORIG, .FILL, .BLKW, .STRINGZ, và .END. Lưu ý tất cả mã giả này đều có dấu chấm như là ký tự đầu tiên của nó.

Mã giả	Ví dụ	Chức năng
.ORIG	.ORIG x3000	Chỉ cho bộ hợp dịch biết vị trí ô nhớ bắt đầu của chương trình. Câu lệnh đầu tiên được dịch cũng sẽ đặt ở vị trí đó. Như ở ví dụ, câu lệnh đầu tiên được dịch sẽ được đặt tại vị trí x3000.
.FILL	.FILL x1234	Yêu cầu bộ hợp dịch để dành ô nhớ tiếp theo trong chương trình để lưu dữ liệu. Giá trị khởi tạo của ô nhớ này là toán hạng của mã giả. Như ở ví dụ, ô nhớ tiếp theo sẽ được khởi tạo giá trị x1234.
.BLKW	MANG .BLKW x10	Yêu cầu bộ hợp dịch để dành một lượng ô nhớ bằng với toán hạng của mã giả để lưu trữ dữ liệu. Như ở ví dụ, 16 ô nhớ tiếp theo sẽ được để dành để lưu dữ liệu. Nhãn của lệnh này (MANG) sẽ tương ứng với vị trí ô nhớ đầu tiên trong khối.
.STRINGZ	MANG_KI_TU .STRINGZ "Hello world!"	Yêu cầu bộ hợp dịch khởi tạo n+1 ô nhớ, trong đó n ô ứng với chiều dài của mảng và ô cuối cùng mang giá trị là x0000 biểu diễn giá trị cuối của mảng, có thể để làm trị canh cho các thuật toán lặp. Như ở ví dụ, bộ nhớ được yêu cầu để dành mảng có 13 ô nhớ. Nhãn của lệnh tương ứng với vị trí ô nhớ đầu tiên.
.END		Báo cho bộ hợp dịch biết vị trí cuối cùng của chương trình. Tất cả các lệnh sau mã này sẽ không được dịch.

Ví dụ 17: Viết chương trình hợp ngữ hoàn chỉnh thực hiện các yêu cầu sau:

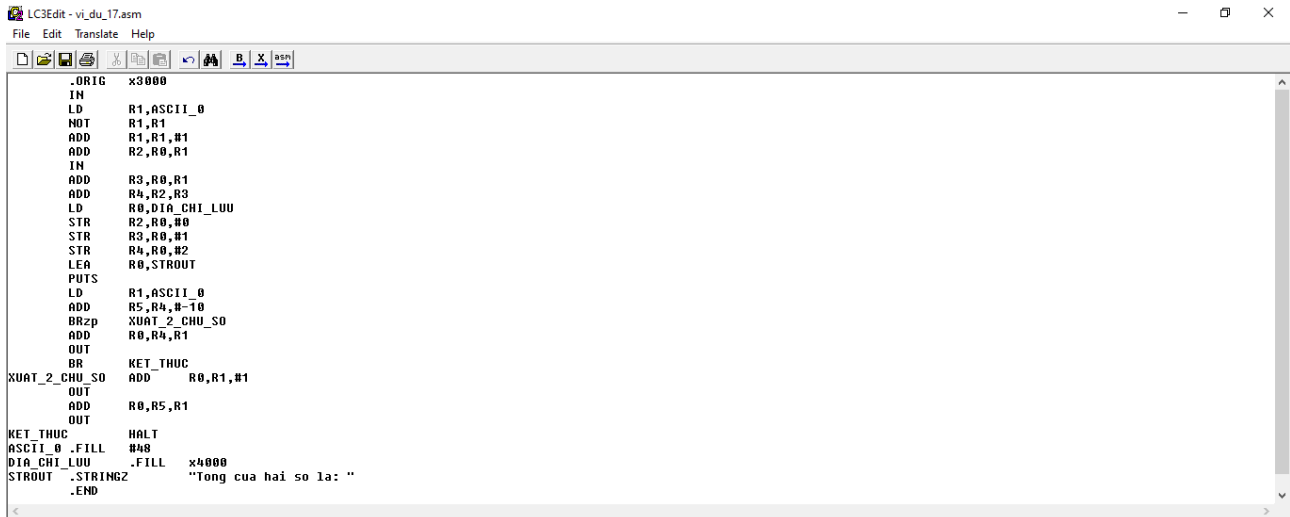
- Nhập lần lượt hai số (từ 0 đến 9) từ bàn phím.
- Tính tổng của hai số này và in ra màn hình theo cú pháp: “Tổng của hai số là: XX”  
Trong đó XX tương ứng với kết quả.
- Lưu kết quả này vào 3 ô nhớ tại địa chỉ x4000, x4001, x4002 theo thứ tự: số nhập thứ nhất, số nhập thứ hai, tổng của 2 số.

	.ORIG	x3000	
	IN		;nhập kí tự thứ nhất
	LD	R1,ASCII_0	;chuyển mã ascii thành số
	NOT	R1,R1	
	ADD	R1,R1,#1	
	ADD	R2,R0,R1	;lưu giá trị số vào R2
	IN		
	ADD	R3,R0,R1	;và R3
	ADD	R4,R2,R3	;tổng của 2 số
	LD	R0,DIA_CHI_LUU	;lấy địa chỉ lưu dữ liệu
	STR	R2,R0,#0	;lưu dưới dạng base + offset
	STR	R3,R0,#1	
	STR	R4,R0,#2	
	LEA	R0,STROUT	;lấy ô nhớ địa chỉ của chuỗi thông báo
	PUTS		;xuất chuỗi ra trước, xuất kết quả sau
	LD	R1,ASCII_0	
	ADD	R5,R4,#-10	;kiểm tra tổng có bao nhiêu chữ số
	BRzp	XUAT_2_CHU_SO	
	ADD	R0,R4,R1	;xuất 1 chữ số
	OUT		
	BR	KET_THUC	
XUAT_2_CHU_SO	ADD	R0,R1,#1	;chữ số đầu tiên chắc chắn là 1
	OUT		
	ADD	R0,R5,R1	;sau khi trừ 10 thì R5 chứa chữ số hàng đơn vị
	OUT		
KET_THUC	HALT		
ASCII_0	.FILL	#48	
DIA_CHI_LUU	.FILL	x4000	
STROUT	.STRINGZ	"Tổng của hai số là: "	
	.END		




#### d. Lập trình sử dụng LC3Edit.exe và mô phỏng dùng Simulate.exe

Sử dụng phần mềm LC3Edit.exe để lập trình lại ví dụ vừa rồi.



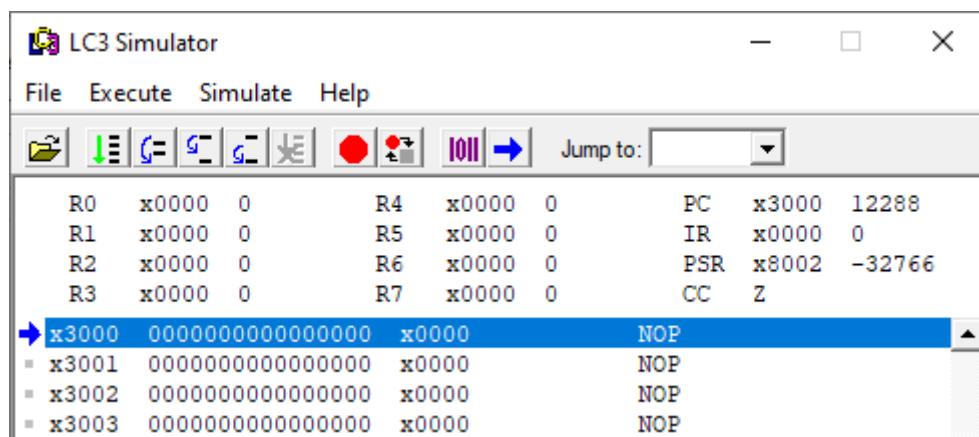
```
.ORIG x3000
IN
LD R1,ASCII_0
NOT R1,R1
ADD R1,R1,#1
ADD R2,R0,R1
IN
ADD R3,R0,R1
ADD R4,R2,R3
LD R0,DIA_CHI_LUU
STR R2,R0,#0
STR R3,R0,#1
STR R4,R0,#2
LEA R0,STROUT
PUTS
LD R1,ASCII_0
ADD R5,R4,#-10
BRzp XUAT_2_CHU_SO
ADD R0,R4,R1
OUT
KET_THUC
BR XUAT_2_CHU_SO
OUT
ADD R0,R5,R1
OUT
KET_THUC
ASCII_0 .FILL #48
DIA_CHI_LUU .FILL x4000
STROUT .STRINGZ "Tong cua hai so la: "
.END
```

Do chương trình được viết bằng hợp ngữ, phải lưu chương trình với tên file có đuôi là “.asm” và phải bấm vào nút  để biên dịch. Nếu không có lỗi, ta nhận được kết quả:



```
Assembling C:\Users\ryell\Desktop\lc3\vi du\vi_du_17.asm...
Starting Pass 1...
Pass 1 - 0 error(s)
Starting Pass 2...
Pass 2 - 0 error(s)
```



Sau đó, mở phần mềm Simulate.exe để chạy thử.

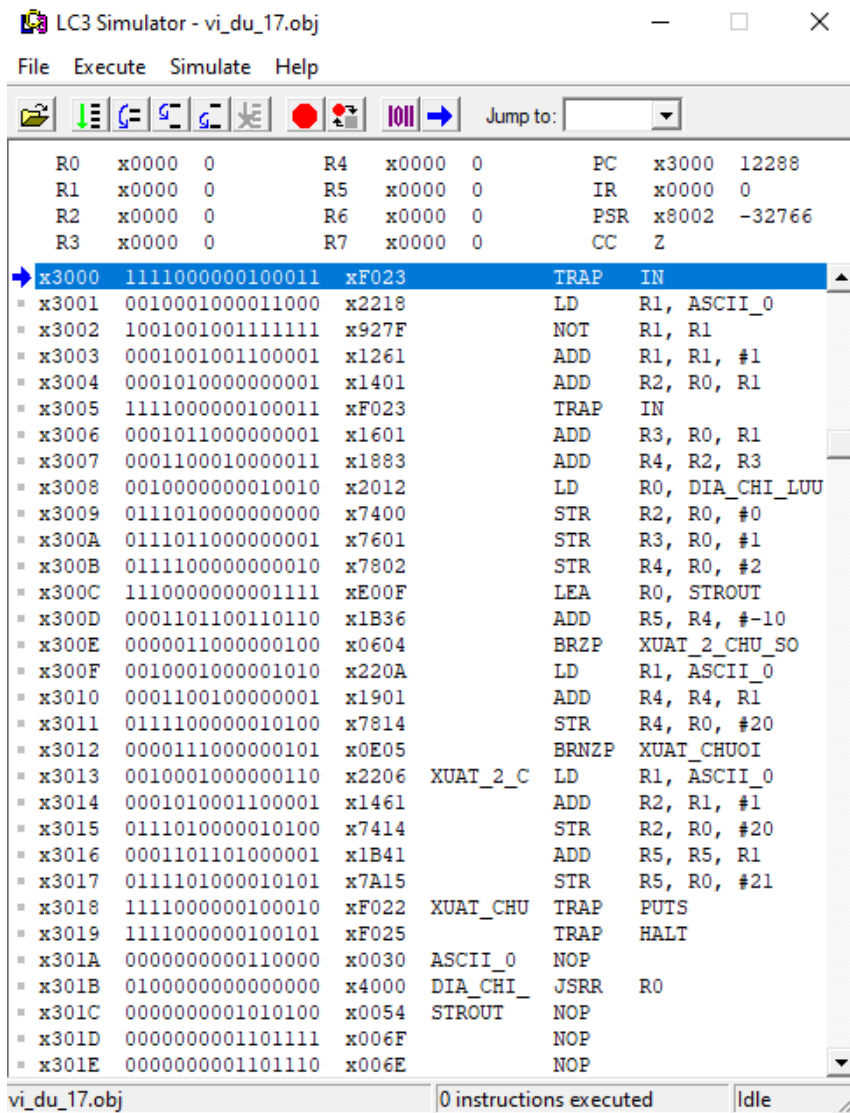


Register	Value
R0	x0000 0
R1	x0000 0
R2	x0000 0
R3	x0000 0
R4	x0000 0
R5	x0000 0
R6	x0000 0
R7	x0000 0
PC	x3000 12288
IR	x0000 0
PSR	x8002 -32766
CC	Z

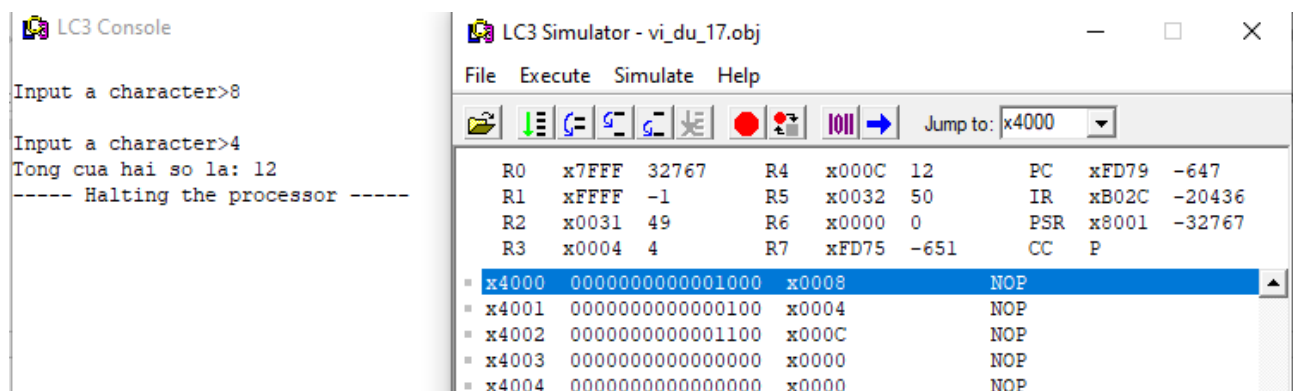
  

Address	Instruction
x3000	0000000000000000 x0000 NOP
x3001	0000000000000000 x0000 NOP
x3002	0000000000000000 x0000 NOP
x3003	0000000000000000 x0000 NOP

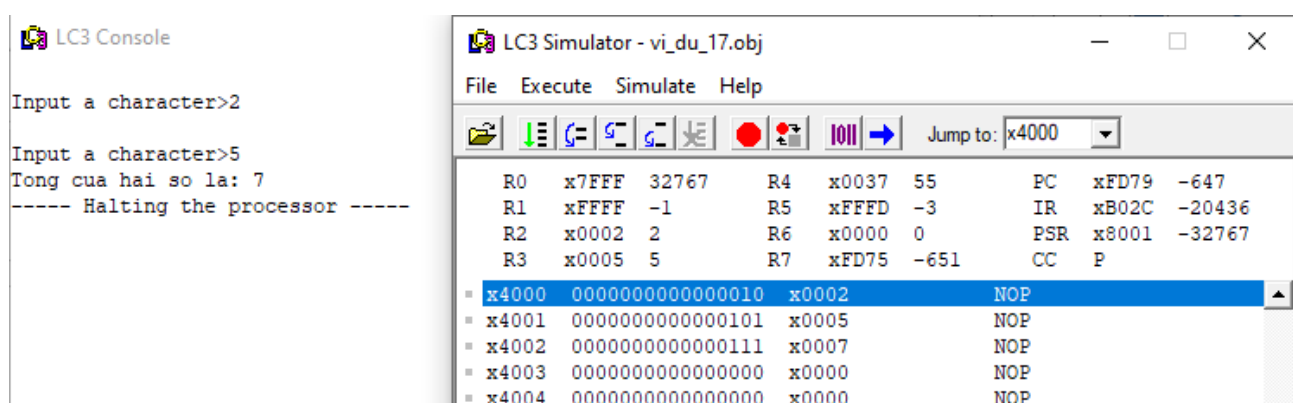
Mở File → Load Program (Ctrl + L) hoặc nút , chọn file object vừa được LC3Edit biên dịch, ta thu được kết quả như hình dưới. Bấm vào nút  để chạy chương trình. Sau khi chạy xong, để chạy lại một lần nữa ta phải Load Program lại, hoặc đơn giản hơn có thể chọn vào File → Reload Program. Ngoài ra có thể chọn chức năng “Jump to:” để xem giá trị tại một ô nhớ bất kì.



Một kết quả ví dụ khi tổng có 2 chữ số:



Một kết quả ví dụ khi tổng có 1 chữ số:



e. Một số ví dụ khác:

Trong phần này sẽ trình bày một số ví dụ đơn giản về lập trình hợp ngữ LC-3.

Ví dụ 18: Viết chương trình thực hiện nhập 2 số (từ 0 đến 9) từ bàn phím, tính hiệu của 2 số này và in ra màn hình.

Phân tích: Ta thấy rằng hiệu của 2 số 1 chữ số chắc chắn cũng chỉ có 1 chữ số. Tuy nhiên hiệu này có thể là số âm, vì vậy cần phải dùng câu lệnh rẽ nhánh. Khi gặp trường hợp số âm, ta in ra dấu “-“, sau đó chuyển kết quả thành số dương (bù 2) và in ra màn hình.

	.ORIG	x3000	
	IN		
	ADD	R1,R0,#0	;copy dữ liệu từ R0 sang R1, chừa lại vị trí R0 để nhập kí tự tiếp theo
	IN		
	NOT	R0,R0	
	ADD	R0,R0,#1	
	ADD	R1,R1,R0	;R1 := R1-R0
	BRzp	IN_CHU_SO	
	LD	R0,MINUS_SIGN	;nếu là số âm, chuyển thành số dương và in dấu “-“
	OUT		
	NOT	R1,R1	
	ADD	R1,R1,#1	
IN_CHU_SO	LD	R0,ASCII_0	
	ADD	R0,R0,R1	
	OUT		
	HALT		
MINUS_SIGN	.FILL	#45	;“-“
ASCII_0	.FILL	#48	;“0”
	.END		

Ví dụ 19: Viết chương trình thực hiện nhập 2 số (từ 0 đến 9) từ bàn phím, tính tích hai số này và in ra màn hình.

Phân tích: Có 2 thuật toán chính cần phải thực hiện trong bài này:

- Sử dụng lặp biết trước số lần để tính tích của hai số
- Sử dụng lặp không biết trước số lần để tách một số bất kì thành 2 chữ số hàng chục và hàng đơn vị

	.ORIG	x3000	
	IN		
	LD	R1,ASCII_0M	;lấy hàng số chuyển từ mã ASCII sang số
	ADD	R2,R0,R1	;copy dữ liệu số vừa nhập R2
	IN		
	ADD	R3,R0,R1	
	AND	R4,R4,#0	;tính tích R2 x R3

LAP	ADD	R2,R2,#-1	
	BRn	TACH_SO	
	ADD	R4,R4,R3	
	BR	LAP	
TACH_SO	AND	R2,R2,#0	;tách chữ số hàng chục và hàng đơn vị
LAP2	ADD	R4,R4,#-10	
	BRn	TIEP	
	ADD	R2,R2,#1	;R2 là chữ số hàng chục
	BR	LAP2	
TIEP	ADD	R3,R4,#10	;R3 là chữ số hàng đơn vị
	LD	R1,ASCII_0	
	ADD	R2,R2,#0	;kiểm tra R2 bằng 0 thì bỏ qua, không in
	BRz	XUAT_DON_VI	
	ADD	R0,R2,R1	
	OUT		
XUAT_DON_VI	ADD	R0,R3,R1	
	OUT		
	HALT		
ASCII_0	.FILL	#48	; "0"
ASCII_0M	.FILL	#-48	
	.END		

**Ví dụ 20:** So sánh hai số 1 chữ số được nhập từ bàn phím (lần lượt nhập vào là a và b)

- Nếu  $a > b$  thì in ra màn hình: "A lon hơn B"
- Nếu  $a < b$  thì in ra màn hình: "A be hơn B"
- Nếu  $a = b$  thì in ra màn hình "Hai so bang nhau"

**Phân tích:** Bài này không quá khó, chỉ cần lấy hiệu hai kí tự vừa được nhận vào và rẽ nhánh hai lần.

	.ORIG	x3000	
	IN		
	ADD	R1,R0,#0	;copy dữ liệu từ R0 sang R1
	IN		
	NOT	R0,R0	
	ADD	R0,R0,#1	
	ADD	R1,R1,R0	;R1 := R1-R0
	BRn	BE_HON	
	BRp	LON_HON	
	LEA	R0,STR_BANG	;2 so bang nhau
	BR	IN_CHUOI	
BE_HON	LEA	R0,STR_BE	
	BR	IN_CHUOI	
LON_HON	LEA	R0,STR_LON	
IN_CHUOI	PUTS		
	HALT		
STR_BE	.STRINGZ	"A be hơn B"	

STR_LON	.STRINGZ	"A lon hon B"
STR_BANG	.STRINGZ	"Hai so bang nhau"
	.END	

Ví dụ 21: Mở rộng ví dụ ở trên, in ra màn hình giá trị lớn nhất, giá trị bé nhất của 3 số được nhập từ bàn phím.

Phân tích: Sử dụng phép lặp biết trước số lần, có thể mở rộng thay số 3 bởi số lượng bất kì. Vì các phép so sánh là phép trừ nên có thể dùng trực tiếp mà không cần phải chuyển từ mã ASCII sang số.

Chức năng của các biến trong chương trình:

- R1: lưu số lần lặp còn lại
- R2: lưu giá trị nhỏ nhất
- R3: lưu giá trị lớn nhất

	.ORIG	x3000	
	LD	R1,SO_LAN_LAP	;số lần lặp
	LD	R2,INIT_MIN	;khởi tạo giá trị nhỏ nhất
	LD	R3,INIT_MAX	;khởi tạo giá trị lớn nhất
LAP	IN		
	NOT	R4,R0	;chuyển R0 thành số âm để tính hiệu
	ADD	R4,R4,#1	
	ADD	R5,R2,R4	;so sánh với giá trị nhỏ nhất
	BRnz	TIEP_TUC_1	
TIEP_TUC_1	ADD	R2,R0,#0	
	ADD	R5,R3,R4	
	BRzp	TIEP_TUC_2	
TIEP_TUC_2	ADD	R3,R0,#0	
	ADD	R1,R1,#-1	;giảm R1 đi 1 đơn vị, nếu bằng 0 thì bắt đầu in dữ liệu
	BRp	LAP	
	LEA	R0,STR_MIN	;in min và max
	PUTS		
	ADD	R0,R2,#0	
	OUT		
	LEA	R0,STR_MAX	
	PUTS		
	ADD	R0,R3,#0	
	OUT		
	HALT		
SO_LAN_LAP	.FILL	#3	
INIT_MIN	.FILL	#57	
INIT_MAX	.FILL	#48	
STR_MIN	.STRINGZ	"So be nhat la: "	
STR_MAX	.STRINGZ	"\nSo lon nhat la: "	
	.END		

Ví dụ 22: Nhập một chuỗi các số có một chữ số từ bàn phím, kết thúc khi nhập vào ký tự #. In ra màn hình tổng của dãy số vừa được nhập (giả sử rằng tổng của dãy này nhỏ hơn 100). Lưu các số vừa được nhập vào vào bộ nhớ từ địa chỉ x4000, kết thúc bằng giá trị “#”.

Cho biết mã ASCII của ký tự “#” là x23.

	.ORIG	x3000	
	LD	R1,ASCII_#M	
	LD	R2,ASCII_0M	
	LD	R3,DIA_CHI_DAU	
	AND	R4,R4,#0	;tổng của dãy
LAP	IN		
	ADD	R5,R0,R1	
	BRz	NGUNG_LAP	
	ADD	R5,R0,R2	
	STR	R5,R3,#0	;luu giá trị vào bộ nhớ
	ADD	R3,R3,#1	;tăng địa chỉ ô nhớ lên 1
	ADD	R4,R4,R5	;tính tổng
NGUNG_LAP	BR	LAP	
	STR	R0,R3,#0	;luu kí tự # vào ô nhớ, không cần thiết phải tăng địa chỉ nữa
	LEA	R0,STR_OUT	
	PUTS		
	AND	R2,R2,#0	;tách chữ số hàng chục và hàng đơn vị
LAP2	ADD	R4,R4,#-10	
	BRn	TIEP	
	ADD	R2,R2,#1	;R2 là chữ số hàng chục
	BR	LAP2	
TIEP	ADD	R3,R4,#10	;R3 là chữ số hàng đơn vị
	LD	R1,ASCII_0	
	ADD	R2,R2,#0	;kiểm tra R2 bằng 0 thì bỏ qua, không in
	BRz	XUAT_DON_VI	
	ADD	R0,R2,R1	
	OUT		
XUAT_DON_VI	ADD	R0,R3,R1	
	OUT		
	HALT		
ASCII_0	.FILL	#48	; "0"
ASCII_0M	.FILL	#-48	
ASCII_#M	.FILL	x-23	; "#"
DIA_CHI_DAU	.FILL	x4000	
STR_OUT	.STRINGZ	"Tong cua day so la: "	
	.END		

- HẾT -