TỰA ĐỀ TÀI :

THIẾT KẾ

THIẾT BỊ THỰC TẬP
VI XỬ LÍ 8085

Giáo viên hướng dẫn : NGUYỄN VIỆT HÙNG
NGUYỄN THANH BÌNH
Sinh viên thực hiện : NGUYỄN THẾ KỲ SƯƠNG
Lớp : 95KDD

TP. HCM , 2/2000

Bộ Giáo Dục Đào Tạo
Đại Học Quốc Gia TP.HCM
Trường Đại Học Sư Phạm Kỹ Thuật
Khoa Điện – Điện Tử
Bộ Môn Điện Tử

Cộng Hòa Xã Hội Chủ Nghĩa Việt Nam
Độc lập – Tự do – Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ và tên : NGUYỄN THẾ KỲ SƯƠNG
Khoá : 1995 – 2000 Lớp : 95KDD Ngành : Điện tử

1. Đề tài : **THIẾT KẾ THIẾT BỊ THỰC TẬP VI XỬ LÍ 8085**

2. Phần thuyết minh : Thiết kế chương trình monitor
3. Bản vẽ , bảng biểu : Các bản vẽ bảng biểu cần thiết .
4. Giáo viên hướng dẫn : Thầy NGUYỄN VIỆT HÙNG
   Thầy NGUYỄN THANH BÌNH
5. Ngày nhận đề tài :
6. Ngày nộp đề tài : 28/2/2000

Cán bộ hướng dẫn

Thông qua bộ môn
Ngày tháng năm 2000

Chủ nhiêm bộ môn

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

NHẬN XÉT GIÁO VIÊN DUYỆT

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

LỜI MỞ ĐẦU

Ngày nay, kỹ thuật vi xử lí đã trở nên quen thuộc với hầu hết mọi người, được ứng dụng rộng rãi trong nhiều lĩnh vực, đặc biệt trong lĩnh vực điều khiển tự động. Do đó nhu cầu nghiên cứu để sử dụng vi xử lí là hết sức cần thiết đối với hầu hết các sinh viên ngành điện tử .

Đề tài :" Thiết Kế Thiết Bị Thực Tập Vi Xử Lí" nhằm giải quyết phần nào nhu cầu nghiên cứu của bản thân, cũng như nhu cầu thực tập, nghiên cứu của sinh viên và những ai yêu thích vi xử lí.

Thời gian, khả năng và công tác in ấn là những yếu tố chính gây ra những sai sót và khiếm khuyết trong đồ án này. Tôi rất mong và ghi nhận những đóng góp của qúi **thầy cô** và **các bạn sinh viên.**

Thủ Đức , 21/2/2000
Sinh viên thực hiện
NGUYỄN THẾ KỲ SƯƠNG

LỜI CẢM TẠ

Sau bảy tuần làm viêc, tập đồ án đã được hoàn tất, đó là dấu hiệu cuối cùng để báo hiệu sự kết thúc của 5 năm đại học.

Tôi cảm ơn **cha mẹ** tôi rât nhiều. Chắc chắn, tôi sẽ không đạt được gì nếu không có sự hy sinh giáo dưỡng của cha mẹ. Chính sự hy sinh đó đã là nguồn động viên và thúc đẩy tôi trên con đường học tập, nghiên cứu.

Tôi chân thành cảm ơn thầy **NGUYỄN VIỆT HÙNG** và thầy **NGUYỄN THANH BÌNH**, là những người đã trực tiếp hướng dẫn tôi thực hiện đề tài này .

Tôi chân thành cảm ơn các THẦY CÔ khoa Điện và khoa sư phạm đã cung cấp những kiến thức qúy báu trong suốt thời gian học tập.

Tôi chân thành cảm ơn các BẠN đã giúp đỡ tôi trong học tập cũng như lúc làm đề tài.

Thủ Đức , 21/2/2000

Sinh viên thực hiện

NGUYỄN THẾ KỲ SƯƠNG

MỤC LỤC

LIỆT KÊ BẢNG

Trang

LIỆT KÊ HÌNH

PHẦN B
NỘI DUNG

Chương 1 :
DẪN NHẬP

CHƯƠNG 1 : DẪN NHẬP

1.1 ĐẶT VẤN ĐỀ:

Công nghệ điện tử và tin học ngày nay phát triển rất mạnh mẽ. Đặc biệt, sự ra đời của các bộ vi xử lí, vi điều khiển có tốc độ ngày càng cao như:

Vi xử lí 4040 (4 bit) là một vi xử lí thuộc thế hệ đầu tiên do Intel sản xuất.

8080 và 8085 của Intel, Z80 của Zilog, 6800 và 6809 của Motorola. Đây là các vi xử lí 8 bit tiêu biểu cho thế hệ thứ hai.

8086/80186/80286 của Intel, 68000/68010 của Motorola. Đây là các vi xử lí 16 bit thuộc thế hệ thứ ba.

Ngày nay có các vi xử lí có tốc độ rất cao như 80386/80486 (32 bit) và Pentium (64 bit) của Intel…

Các bộ vi xử lí, vi điều khiển không những được ứng dụng rộng rãi trong hệ điều khiển của các nhà máy, mà còn được sử dụng trong dân dụng như: Tivi, đầu máy và các loại đồ chơi…

Trước nhu cầu tìm hiểu về vi xử lí, của chính bản thân và của những người yêu thích vi xử lí, nhóm đã bắt tay vào thực hiện hiện đề tài ***"Thiết kế thiết bị thực tập vi xử lí 8085"***

1.2 TẦM QUAN TRỌNG CỦA VẤN ĐỀ:

Nhu cầu tìm hiểu về vi xử lí rất lớn. Nhưng nhà trường vẫn chưa có thiết bị thực tập về vi xử lí có thể đáp ứng đầy đủ những yêu cầu thực tập nghiên cứu của sinh viên. Đề tài "***Thiết kế thiết bị thực tập vi xử lí 8085***" với mong muốn thiết kế được một thiết bị thực tập có thể đáp ứng hầu hết các nhu cầu của người sử dụng. Ngoài ra cũng là tài liệu hữu ích cho những ai muốn thiết kế riêng cho mình một hệ thống ưu việt hơn.

1.3 GIỚI HẠN VẤN ĐỀ:

Các thiết bị thực tập vi xử lí do các hãng nước ngoài chế tạo cũng rất đa dạng, tiện lợi, nhưng chúng có chung những đặc điểm cơ bản của một bộ vi xử lí.

Đây là lần đầu tiên tìm hiểu về vi xử lí trong điều kiện:

+ Ở trường đại học, nhóm thực hiện đề tài chỉ được học 15 tiết chuyên đề vi xử lí.
+ Thời gian thực hiện chính thức chỉ có 49 ngày.
+ Kinh nghiệm thực tế không nhiều.
+ Đề tài được chia làm hai phần: phần cứng và phần mềm, được báo cáo trong hai cuốn khác nhau. Phần báo cáo này chỉ trình bày về phần mềm. Vì vậy người thực hiện chỉ thiết kế phần mềm có những đặc điểm chính như sau:
- 36 phím: gồm các phím số và phím chức năng.
+ Các phím số từ 0 tới F
+ Các phím chức năng gồm:

- ADD : Phím nhận địa chỉ bộ nhớ. Các led hiển thị địa chỉ và dữ liệu tại địa chỉ đó.
- BREAK: Chạy chương trình từng đoạn, các điểm dừng được đặt bằng phím set. (Không có trong phạm vi đề tài này)
- COPY : Chép đoạn dữ liệu từ nơi này sang nơi khác trong vùng địa chỉ của Ram, cho phép vùng đích và vùng gốc trùng nhau.
- CTRL, SHIFT : Phần cứng 8279.
- DEL : Xóa một đoạn dữ liệu hay một ô dữ liệu.
- DOWN : Giảm địa chỉ hiện hành.
- INS : Cho phép chèn một ô dữ liệu hay một đoạn dữ liệu, có sửa địa chỉ.
- INTR : Ngắt cứng.
- PAUSE : Dừng chương trình đang thực hiện (Không có trong phạm vi đề tài này)
- PC và GO: Dùng để đặt địa chỉ và chạy chương trình tại địa chỉ đó.
- REG: Xem và nạp nội dung thanh ghi.
- RESET : Reset lại hệ thống. (phần cứng)
- SET : Đặt điểm dừng. (Không có trong phạm vi đề tài này)
- SRCH : Tìm kiếm dữ liệu trong vùng nhớ.
- STEP : Chạy từng lệnh.
- UP : Tăng địa chỉ hiện hành và xác định dữ liệu vào địa chỉ hiện hành.
- USER1 và USER2 : Dành cho người sử dụng.

1.4 MỤC ĐÍCH NGHIÊN CỨU:

Qua việc thực hiện đề tài ***"Thiết kế thiết bị thực tập vi xử lí 8085"*** là một cách để người thực hiện đề tài nghiên cứu kĩ tập lệnh, cấu trúc của vi xử lí.

Sản phẩm có thể ứng dụng trong giảng dạy, thực tập và thí nghiệm về phần mềm cũng như phần cứng của vi xử lí.

Chương 2 :
CƠ SỞ LÝ LUẬN

Chương 3 :
GIỚI THIỆU TỔNG QUÁT VỀ THIẾT BỊ

CHƯƠNG 3 : GIỚI THIỆU TỔNG QUÁT VỀ THIẾT BỊ THỰC TẬP

Thiết bị sử dụng vi xử lí 8085 do Intel chế tạo, có vỏ hai hàng chân gồm 40 chân. Được cắm trên Socket 40 chân. Tốc độ được quyết định bằng thạch anh cung cấp xung clock cho vi xử lí.

Bộ nhớ gồm : 2EPROM 8K

3 RAM 8K

Trong 3 RAM trên có một socket có thể cắm cả EPROM. 3 RAM có địa chỉ từ 4000 → 9FFFH.

EPROM thứ nhất dùng để lưu trữ chương trình Monitor điều khiển toàn bộ hoạt động của thiết bị, có địa chỉ từ 000H → 1FFFH.

EPROM thứ hai dùng để lưu trữ những chương trình tiện ích, những chương trình này phục vụ bài thí nghiệm, có địa chỉ từ 2000 → 3FFFH.

Các IC ngoại vi bao gồm :

- 8279 dùng để quét bàn phím và hiển thị, được giải mã theo địa chỉ, có địa chỉ từ điều khiển là A001H và địa chỉ từ dữ liệu là A00H.
- Gồm hai con 8255 vào ra song song có mã I/O từ 00H → 07H và từ 08H → 0FH.
- Gồm hai con 8253, trong đó một con dùng để tạo xung clock cho 8251 có mã I/O từ 10H → 17H, con còn lại có mã từ 18H → 1FH
- 8251 dùng để giao tiếp nối tiếp có mã I/O từ 28 → 2FH
- 8259 dùng để điều khiển ngắt ưu tiên có mã I/O từ 20H → 27H
- ADC 0809 có mã I/O từ 30H → 37H
- DAC 0808 có mã I/O từ 38H → 3FH

Bàn phím gồm 36 phím, ngoài các phím số, phím chức năng, phím ngắt cứng còn có phím Shift và phím Control dùng để mở rộng bàn phím.

Có 8 led hiển thị, 4 led bên trái dùng để hiển thị địa chỉ, các thông báo. Bốn led bên phải dùng để hiển thị dữ liệu, hiển thị số khi ấn các phím số.

Chương 4 :
XÂY DỰNG CHƯƠNG TRÌNH MONITOR

CHƯƠNG 4 : XÂY DỰNG CHƯƠNG TRÌNH MONITOR

4.1 GIỚI THỆU

Trong báo cáo về phần cứng, phần cứng đã được đề cập chi tiết. Trong phần này, chỉ liệt kê các IC chính của hệ thống :

+ Vi xử lí : 8085A
+ Bộ nhớ : hai ROM 2764, hai RAM 6264 và một đế cắm có thể dùng RAM hoặc ROM.
+ Giao tiếp 8255, 8251.
+ Định thời 8253.
+ Quét phím và hiển thị 8279
+ Các bộ chuyển đổi ADC 0809, DAC…

Các IC trên được kết nối trực tiếp hoặc gián tiếp qua các IC phụ (chốt, đệm, giải mã…) sao cho đảm bảo đúng yêu cầu về điện và chức năng của mỗi thiết bị.

Đối với các hệ thống vi xử lí, để hệ thống hoạt động được thì điều kiện cần là phải có một phần cứng đúng và điều kiện đủ là phải có một phần mềm chính xác, phần cứng và phần mềm chúng có mối quan hệ hữu cơ với nhau, có thể thay thế nhau ở một vài chức năng nào đó. vì vậy tùy theo phần cứng cụ thể ta có cách lập trình khác nhau.

4.2 MỘT VÀI YÊU CẦU CHÍNH ĐỐI VỚI CHƯƠNG TRÌNH MONITOR

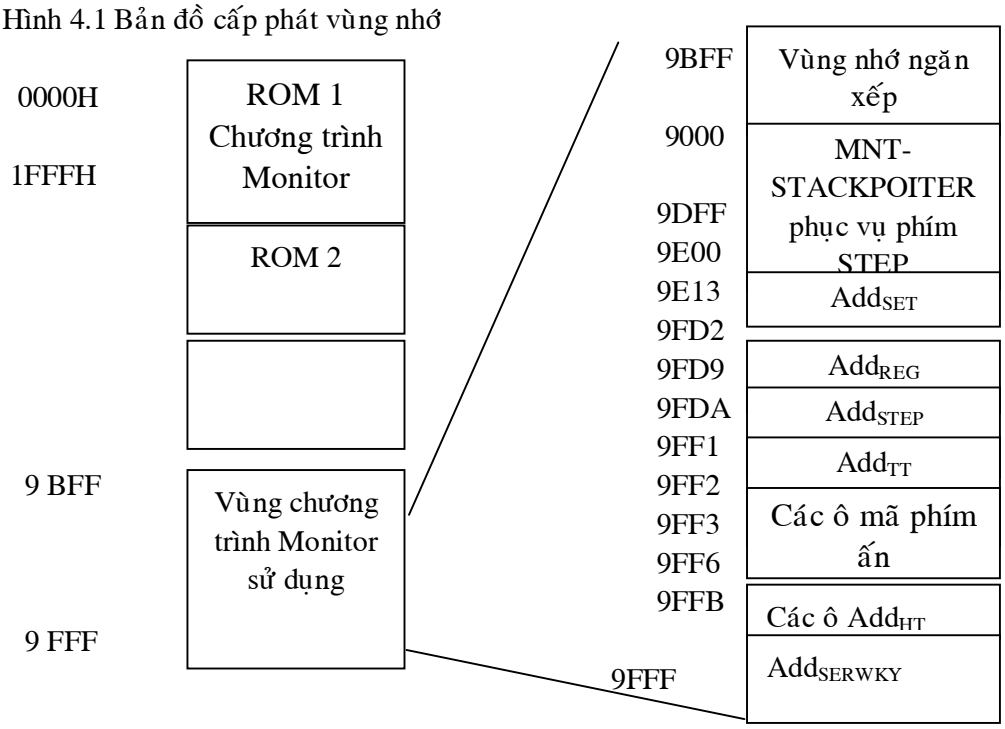
Xuất phát từ việc giới hạn đề tài ta đặt ra những yêu cầu cụ thể như sau:
+ Chương trình phải thực hiện chức năng của các phím sao cho người sử dụng có thể nạp chương trình và chạy được chương trình đó.

Các chương trình con phục vụ cho chương trình Monitor được trình bày theo qui ước:
+ Chương trình con nào xuất hiện đầu tiên sẽ được trình bày trước.
+ Các nhãn được sử dụng cho địa chỉ ô nhớ.
+ Chỉ trình bày giải thuật và các chương trình quan trọng.
+ Chú thích được sử dụng khi cần thiết.

4.3 CẤP PHÁT VÙNG NHỚ

+ ROM 1 : 8 Kbyte lưu trữ tất cả những chương trình có liên quan đến chương trình Monitor.
+ ROM 2 : lưu trữ chương trình phục vụ các bài thí nghiệm (sau này).
+ RAM : Dành cho người sử dụng và 1 Kbyte chót của vùng nhớ để dùng khởi tạo ngăn xếp và các ô nhớ phục vụ chương trình Monitor.

Hình 4.1 Bản đồ cấp phát vùng nhớ



4.4 XÂY DỰNG CHƯƠNG TRÌNH MONITOR

Chương trình Monitor là chương trình Monitor chính, lệnh đầu tiên của chương trình này phải được bắt đầu tại ô nhớ có địa chỉ 000H.

- Khởi tạo ngăn xếp

Ngăn xếp là một tập các ô nhớ trong bộ nhớ RAM. Các ô nhớ này được sử dụng để lưu trữ các thông tin nhị phân một cách tạm thời trong suốt quá trình thi hành một chương trình. Thông tin trao đổi với ngăn xếp có tính LIFO (Last in first Out).

Khởi tạo ngăn xếp là một hoạt động định nghĩa địa chỉ đáy của vùng ngăn xếp, phụ thuộc vào người lập trình.

Căn cứ vào bản đồ cấp phát vùng nhớ RAM trên hình 4.1, có thể khởi tạo ngăn xếp bắt đầu tại địa chỉ 9CFFH.

- Khởi tạo ngoại vi

Khởi tạo ngoại vi là một thủ tục qui định cách thức hoạt động của từng thiết bị ngoại vi đang được sử dụng trong hệ thống.

Nếu không được khởi tạo, các thanh ghi điều khiển (Control Register) của và thanh ghi dữ liệu (Data Register) của ngoại vi đó sẽ ở giá trị ngẫu nhiên, nhưng thiết bị ngoại vi lại hiểu các giá trị này như là các giá trị khởi tạo. Dẫn đến thiết bị ngoại vi có khả năng không làm việc hoặc có làm việc nhưng không đúng yêu cầu.

Để ổn định cách thức hoạt động của ngoại vi, vấn đề khởi tạo ngoại vi phải được thực hiện.

Như đã đề cập, phần cứng của thiết bị này được thiết kế dự trù cho cả việc phát triển đề tài sau này. Hay nói cách khác, phần mềm trong phạm đề tài này vẫn chưa khai thác hết khả năng của phần cứng.

Nhằm nâng cao tính ổn định của hệ thống, ngoại vi nào không được sử dụng cũng sẽ được khởi tạo, ở một cách thức hoạt động cụ thể sau này để tránh tình trạng thả nổi ngoại vi. Chú ý những ngoại vi nào không thay đổi mode hoạt động thì không cần khởi tạo lại.

- Khởi tạo ngắt

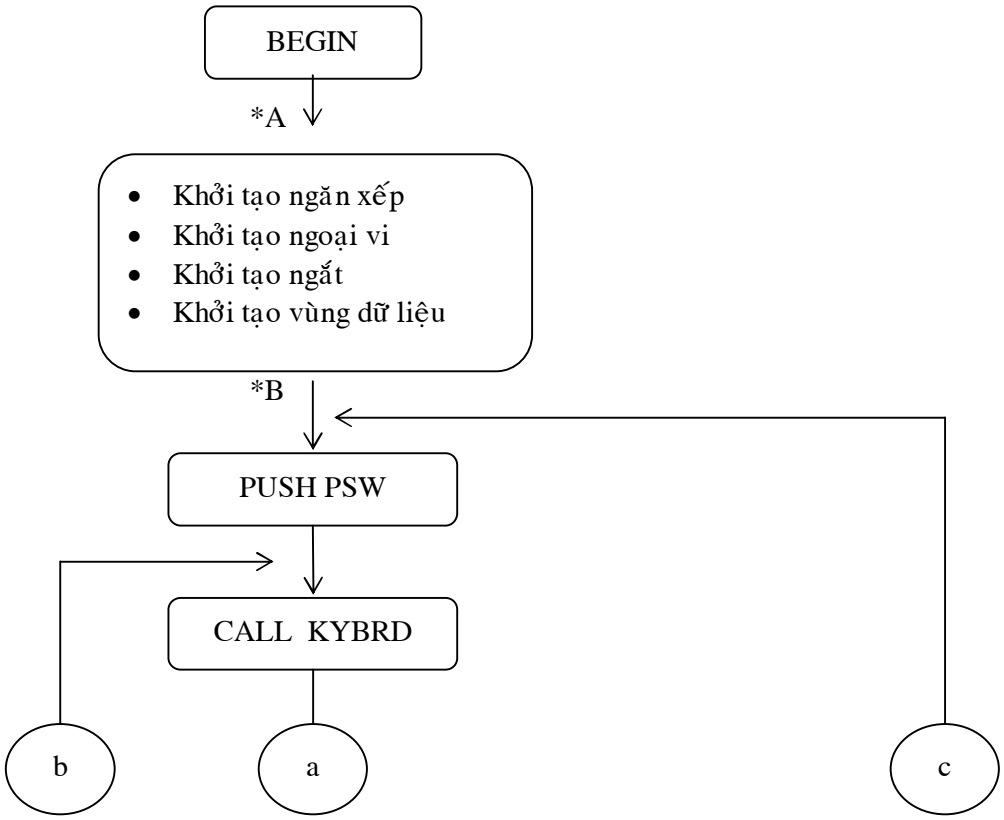
Ngắt là một quá trình thông tin bất đồng bộ với vi xử lí, được kích bởi một ngoại vi bên ngoài.

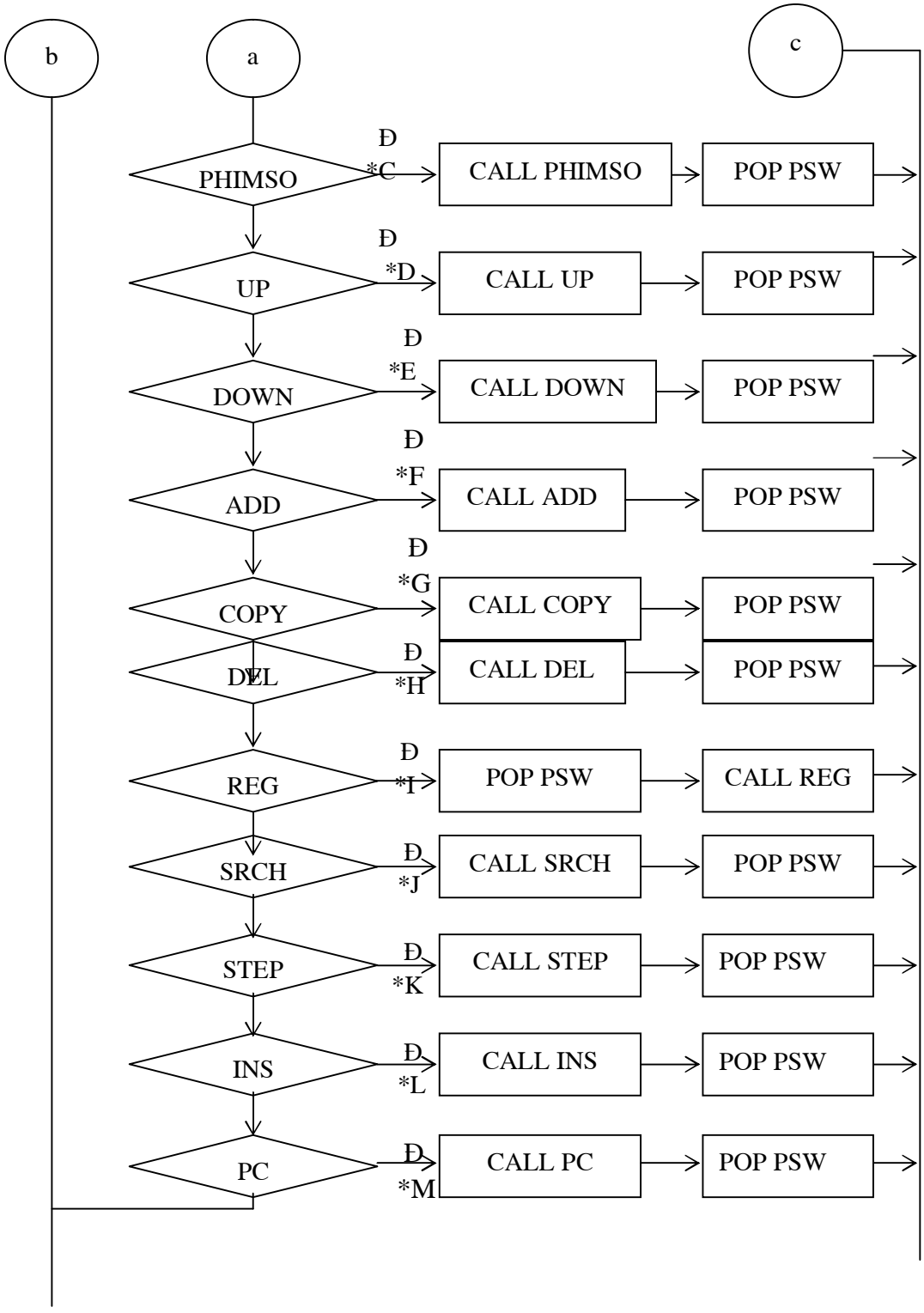
Trong phạm vi đề tài này các vấn đề phần mềm chỉ liên quan tới ngắt Trap, và ngắt RST 6.5 còn các chân ngắt khác chưa sử dụng đến sẽ bị che đi.

- Khởi tạo một số vùng dữ liệu

Do tính chất của phần mềm, đòi hỏi một số vùng dữ liệu phải được khởi tạo một giá trị cụ thể nào đó thì thiết bị mới hoạt động khi mới bật công tắc.

Hình 4. 2 : Lưu đồ chương trình chính MNT




b
a
c
PHIMSO
Đ
*C
CALL PHIMSO
POP PSW
UP
Đ
*D
CALL UP
POP PSW
DOWN
Đ
*E
CALL DOWN
POP PSW
ADD
Đ
*F
CALL ADD
POP PSW
COPY
Đ
*G
CALL COPY
POP PSW
DEL
Đ
*H
CALL DEL
POP PSW
REG
Đ
*I
POP PSW
CALL REG
SRCH
Đ
*J
CALL SRCH
POP PSW
STEP
Đ
*K
CALL STEP
POP PSW
INS
Đ
*L
CALL INS
POP PSW
PC
Đ
*M
CALL PC
POP PSW

```
ORG    0000H
       JMP        START
       ORG        0080
       ; Khởi tạo ngăn xếp
START: LXI        SP, 9CFFH
       ; Khởi tạo 8279
       MVI        A, 3EH      ; lập trình xung clock
       STA        Add_ct79    ; bằng 100 KHz
       MVI        A, 10H      ; Hiển thị 8 kí tự, ghi phải
       STA        Add_CT79    ; Bàn phím quét có lập mã khóa ngoài hai
                              ; phím
       MVI        A, C3H      ; xóa FIFO và xóa hiển thị
       STA        Add_CT79
       MVI        A, 40H      ; Đọc FIFO, không tự tăng, hàng đầu tiên
       STA        Add_CT79
       MVI        A, 90H      ; Ghi vào RAM hiển thị, tự tăng
       STA        Add_CT79    ; Bắt đầu tại Led sát lề phải
       ; Khởi tạo ngoại vi 8255
       MVI        A, 8BH      ; Mode 0, I/0 đơn giản
       OUT        03H
       OUT        0BH
       ; Khởi tạo ngoại vi 8253
       MVI        A, 35H      ; Bộ đếm 0 mode 2, gửi 2 byte
       OUT        13H         ;  đếm BCD
       MVI        A, B5H      ; Bộ đếm 2 mode 2, 2 byte,
       OUT        13H         ; BCD
       MVI        A, 75H      ; Bộ đếm 1; Mode 2; gửi 2 byte
       OUT        13H         ; đếm BCD
       ; Khởi tạo 8253 tạo xung 300 Hz cấp cho TXC và RXC của 8251
       MVI        A, 99H      ; Bộ đếm 0 chia 10000
       OUT        10H
       MVI        A, 99H
       OUT        10H
       MVI        A, 99   ; Bộ đếm 1 chia 10000
       OUT        11H
       MVI        A, 99H
       OUT        11H
       MVI        A, 99H      ; Tạo xung 5Hz đưa ra
       OUT        12H         ; sử dụng
       ; Khởi tạo ngoại vi 8251
```

```
        MVI     A, 5DH      ; Bất đồng bộ, hệ số nhân tốc độ bằng 1, cho
        OUT     23H         ; phép dùng parity, không kiểm tra parity
                            ; chẵn, kí tự dài 18 bit, 1 bit stop
        MVI     A, 10H      ; Cấm phát, cấm nhận, reset tất cả các cờ
lỗi,
        OUT     23H         ; ngăn chặn reset mode bên trong,
                            ; Khởi tạo 8259
        MVI     A, 12H      ; tác động sườn dương, khoảng cách
        OUT     20H         ; các vectơ ngắt 8 byte, có 1 mạch
                            ; PIC, bỏ ICW4
        MVI     A, 40 H     ; chọn kiểu ngắt 40h đến
        OUT     27H         ; 47H
        MVI     A, 00H      ; Đầu vào IRi không nối
        OUT     27H         ; mạch thợ
        ; Khởi tạo ngắt
        MVI     A, E5H      ; Che RST 7.5, RST 5.5 không
        SIM                 ; che RST 5.5
        ; Khởi tạo một số ô nhớ
```

```
MVI     A,10H
STA     Add_TT
MVI     A, 00H
STA     Add_qp+0
STA     Add_qp+1
STA     Add_qp+2
STA     Add_qp+3
STA     Add_HT+4
STA     Add_HT+5
STA     Add_HT+6
STA     Add_HT+7
```

```
        STA     Add_ST 14
        CALL    HELLO
*A :    PUSH    PSW
*B :    CALL    KYBRD
        CPI     10H
        JM      *C
        CPI     'UP'
        JZ      *D
        CPI     'DOWN'
        JZ      *E
```

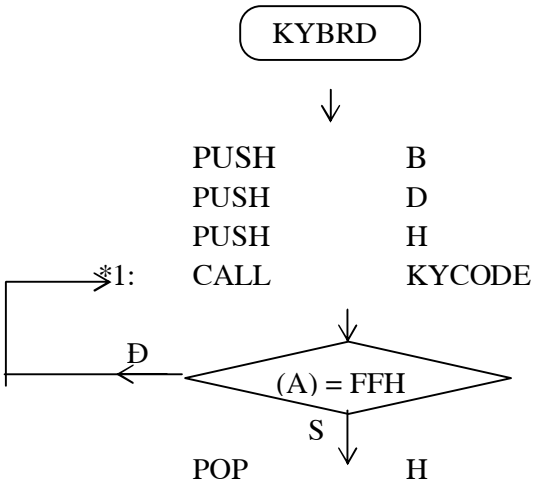
```
        CPI     'ADD'
        JZ      *F
        CPI     COPY
        JZ      *G
        CPI     'DEL'
        JZ      *H
        CPI     'REG'
        JZ      *I
        CPI     'SRCH'
        JZ      *J
        CPI     'STEP'
        JZ      *K
        CPI     'INS'
        JZ      *L
        CPI     ''PC'
        JZ      *M
        JMP     *B
*C :    CALL    PHIMSO
        POP     PSW
        JMP     *A
*D :    CALL    UP
        POP     PSW
        JMP     *A
*E :    CALL    DOWN
        POP     PSW
        JMP     *A
*F :    CALL    ADD
        POP     PSW
        JMP     *A
*G :    CALL    COPY
        POP     PSW
        JMP     *A
*H :    CALL    DEL
        POP     PSW
        JMP     *A
*I :    POP     PSW
        CALL    REG
        JMP     *A
*J :    CALL    SRCH
        POP     PSW
        JMP     *A
*K :    CALL    STEP
        POP     PSW
        JMP     *A
*L :    CALL    INS
        POP     PSW
        JMP     *A
*M :    CALL    PC
        POP     PSW
        JMP     *A
```

KYBRD

- KYBRD : Là chương trình con có nhiệm vụ lấy mã của phím ấn.
- Input: Chờ phím ấn
- Output: Chương trình sẽ dùng lại cho đến khi nào có một phím được gõ, khi đó (A) = Mã phím

  Ngoài A không thanh ghi nào bị điều chỉnh

Có gọi KYCODE



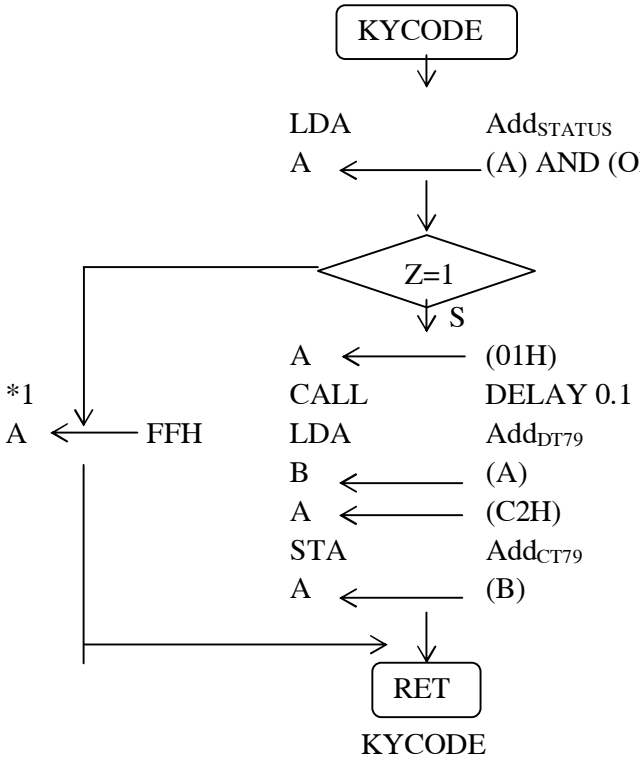

```
        PUSH    B
        PUSH    D
        PUSH    H
*1:     CALL    KYCODE
        (A) = FFH
        Đ
        S
        POP     H
        POP     D
        POP     B
        RET
```

KYRBD

```
        PUSH    B
        PUSH    D
        PUSH    H
*1:CALL         KYCODE
        CPI     FFH
        JZ      *1
        POP     H
        POP     D
        POP     B
        RET
```

KYCODE

- KYCODE : Đây là chương trình con .Nó có tác vụ lấy mã của phím ấn từ FIFO của 8279 đặt vào thanh ghi A
- Input: Gõ phím
- Output: Nếu không gõ phím : (A) = FFH
  Nếu có phím được gõ : (A) = Mã phím
  Các thanh ghi bị điều chỉnh : A,B
  Gọi DELAY 0.1 (Xem phụ lục)

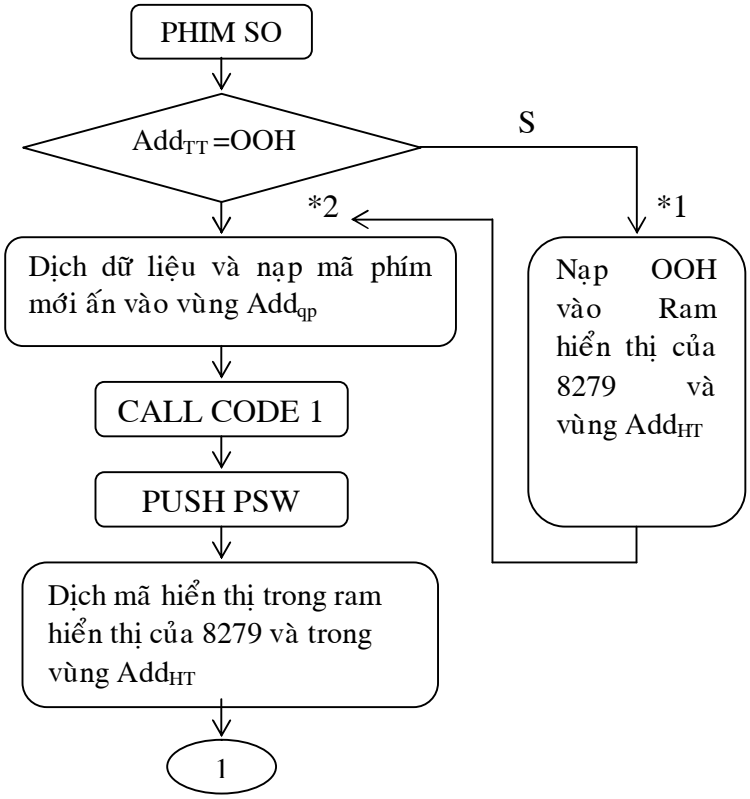


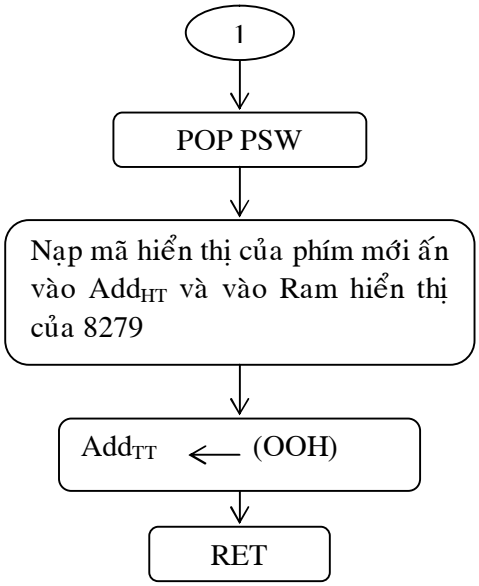
KYCODE

```
LDA     Add_STATUS
ANI     OFH
JZ      *1
MVI     A,01H
CALL    DELAY 0.1
LDA     Add_DT79
MOV     B,A
        MVI     A,C2H
        STA     Add_CT79
        MOV     A,B
        RET
*1 :    MVI     A,FFH
        RET
```

PHIM SO

- PHIM SO: Đây là một chương trình con có tác vụ hiển thị phím ấn và lưu trữ mã phím ấn.
- Input : Mã phím ấn
- Output : Hiển thị phím ấn và lưu trữ phím ấn
  Không thanh ghi nào bị điều chỉnh.
  Add_{TT} = OOH





Hình 4.3 là lưu đồ chương trình phím số

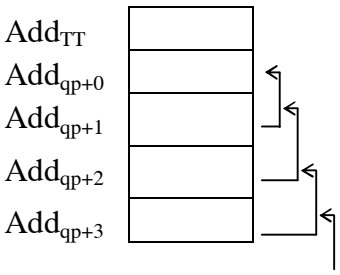
Add_{qp} : là 4 ô nhớ lưu giữ mã phím của 4 lần ấn phím số sau cùng.

Add_{HT}: 4 ô nhớ lưu giữ mã 7 đoạn tương ứng với 4 mã phím sau cùng

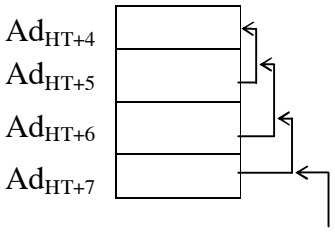
Add_{TT} : cho biết phím sau cùng là phím chức năng hay phím số

CODE : là chương trình con đổi các mã Hexa từ OOH OFH sang một kí tự mã

7 đoạn (xem phụ lục)



Mã phím mới ấn

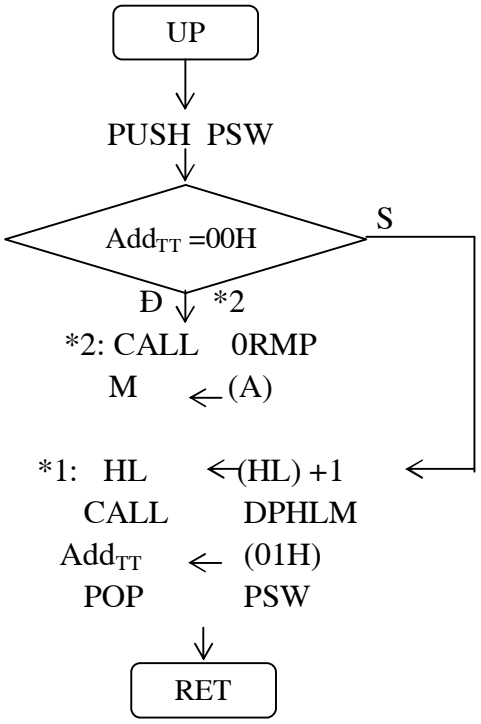


Mã 7 đoạn của phím mới ấn

UP

- UP: Chương trình con có tác vụ tăng địa chỉ hiện hành và nạp dữ liệu hiển thị vào địa chỉ hiện hành đó.
- Input: Thanh ghi HL
- Output: Hiển thị dữ liệu và địa chỉ của ô dữ liệu đó chứa trong HL

Các thanh ghi còn lại không bị điều chỉnh.
Add_{TT} =01H
Add_{TT} : địa chỉ ô nhớ chứa cơ trạng thái
Add_{TT} =00H: phím ấn sau cùng là phím số
Add_{TT} ≠00H: phím ấn sau cùng là phím chức năng



Hình 4.4 Lưu đồ chương trình phím UP

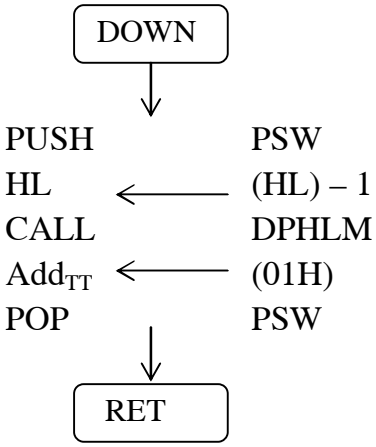
ORMP: là chương trình con hợp hai ô quét phím (xem phụ lục)
DPHLM: là chương trình con có tác vụ hiển thị địa chỉ và dữ liệu trong ô nhớ có địa chỉ trong HL (xem phụ lục).

UP

```
      PUSH    PSW          |     *1:  INX     H
      CPI     OOH          |          CALL    DPHLM
      JZ      *2           |          MVI     A, 01H
      JMP     *1           |          STA     Add_TT
*2:   CALL    ORMP         |          POP     PSW
      MOV     M , A        |          RET
```

- DOWN : Là chương trình con có tác vụ giảm địa chỉ hiện hành xuống 1 đơn vị
- Input : Thanh ghi HL
- Output : Hiển thị địa chỉ và dữ liệu

Add_{TT} = 01H



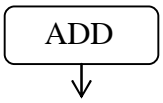

```
DOWN
PUSH    PSW
DCX     H
CALL    DPHLM
MVI     A,01H
STA     Add_TT
POP     PSW
RET
```

- DPHLM (Xem phụ lục)

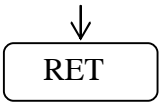
- ADD : Là chương trình con có tác vụ nhận địa chỉ mới , hiển thị địa chỉ và dữ liệu mới đó.
- Input : Lấy dữ liệu trong 4 ô quét phím
- Output : Hiện thị địa chỉ và dữ liệu

  Add_{TT} = 01H

  Không điều chỉnh các thanh ghi



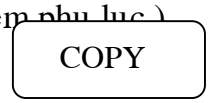

```
PUSH        PSW
PUSH        D
CALL        DEQP
HL    <---  DE
CALL        DPHLM
Add_TT <--- (01H)
POP         D
POP         PSW
```



ADD

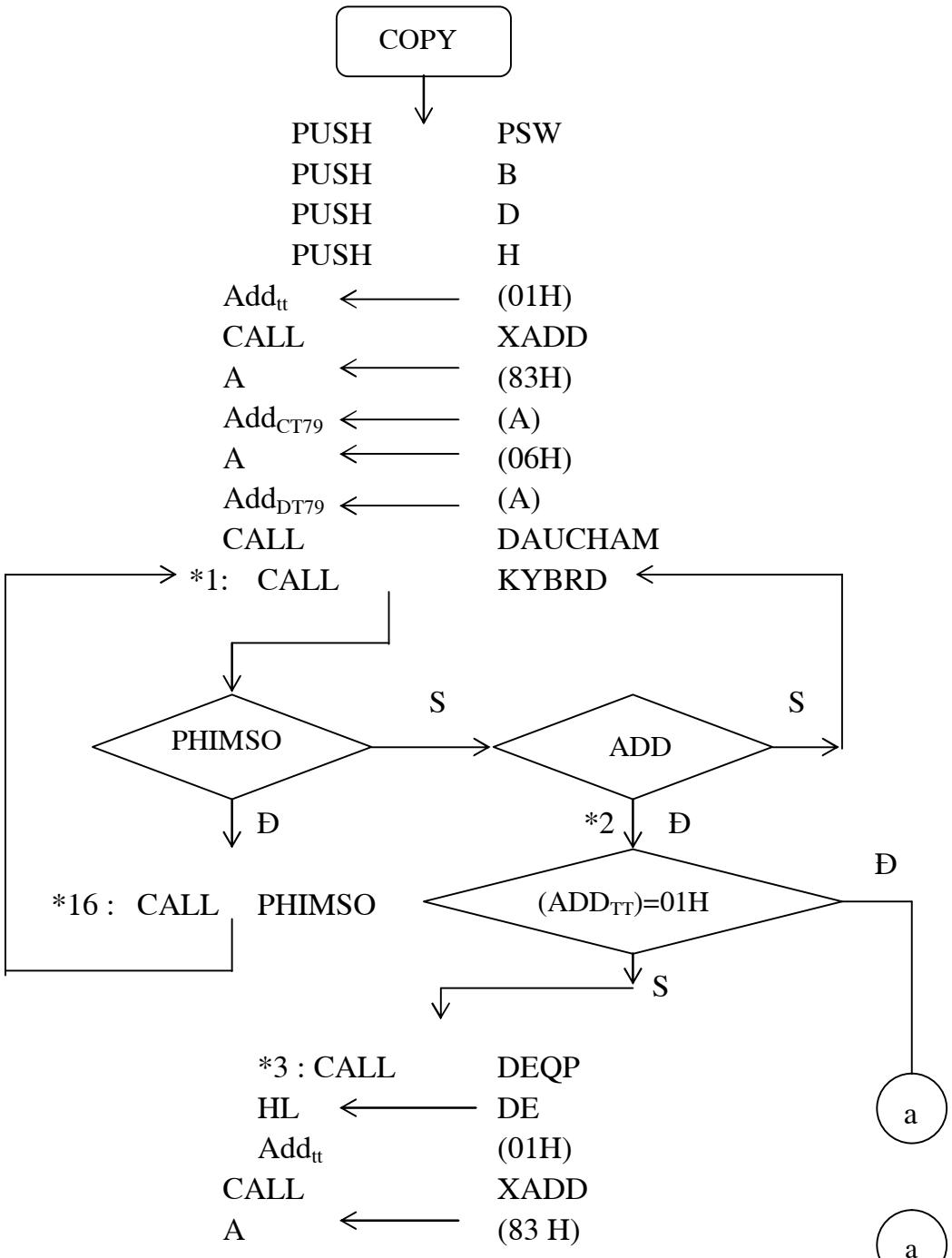
```
PUSH        PSW
PUSH        D
CALL        DEQP
XCHG
CALL        DPHLM
MVI         A,01H
STA         Add_TT
POP         D
POP         PSW
RET
```

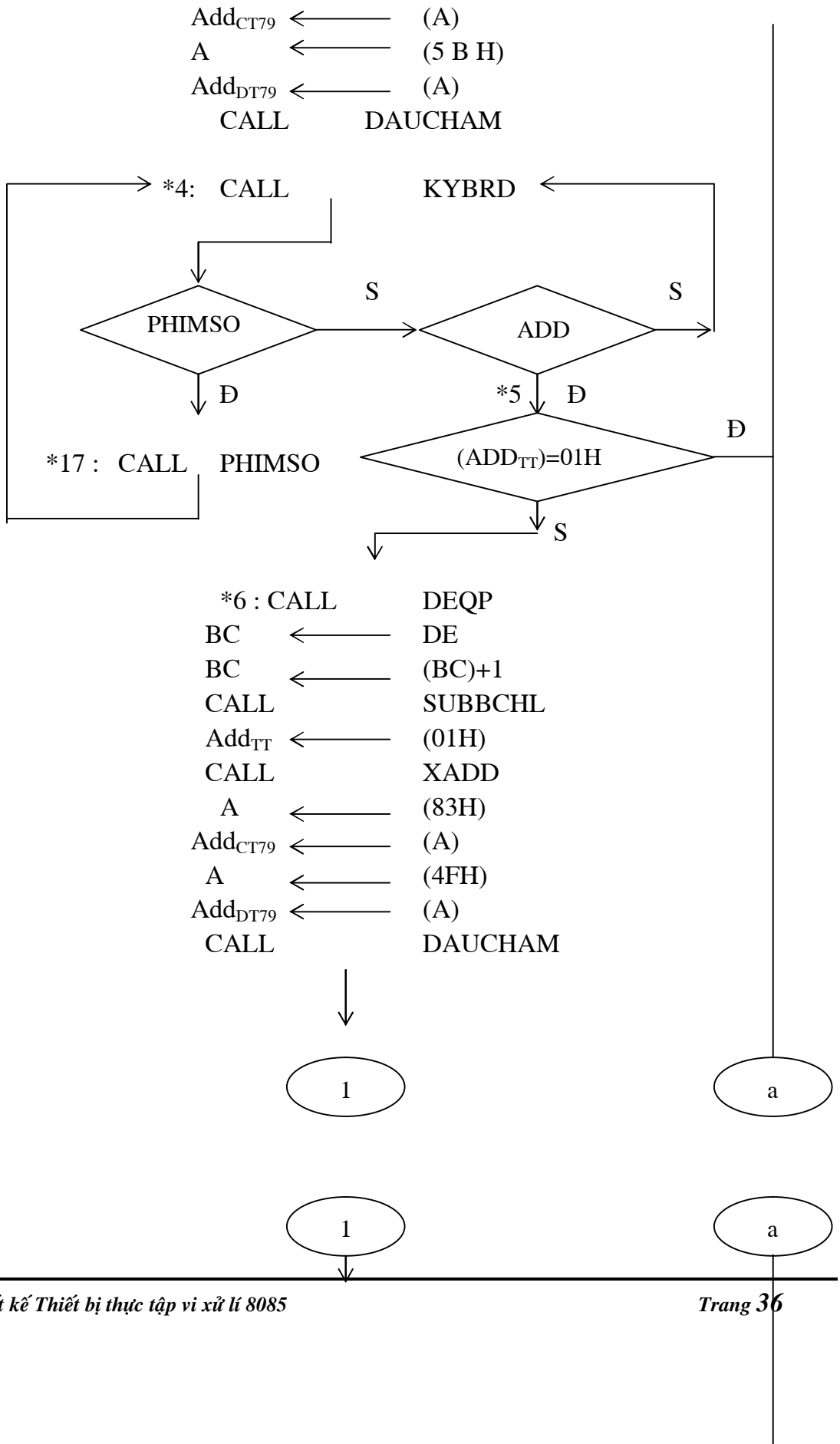
- DPHLM (xem phụ lục)

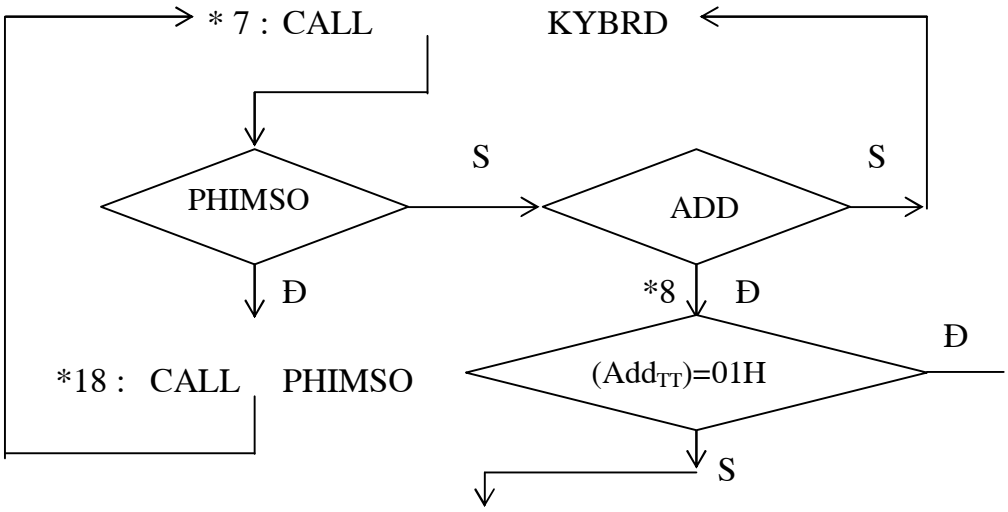


- COPY : Là chương trình con có tác vụ copy dữ liệu từ vùng này sang vùng khác
- Input : Nạp địa chỉ đầu, địa chỉ cuối của vùng gốc và địa chỉ đầu của vùng đích
- Output : Copy , hiển thị địa chỉ và dữ liệu

Chương trình không làm thay đổi nội dung các thanh ghi




Add$_{CT79}$ ⟵ (A)
A ⟵ (5 B H)
Add$_{DT79}$ ⟵ (A)
CALL DAUCHAM
*4: CALL KYBRD
PHIMSO
S
ADD
S
Đ
*5
Đ
Đ
*17 : CALL PHIMSO
(ADD$_{TT}$)=01H
S
*6 : CALL DEQP
BC ⟵ DE
BC ⟵ (BC)+1
CALL SUBBCHL
Add$_{TT}$ ⟵ (01H)
CALL XADD
A ⟵ (83H)
Add$_{CT79}$ ⟵ (A)
A ⟵ (4FH)
Add$_{DT79}$ ⟵ (A)
CALL DAUCHAM
1
a
1
a



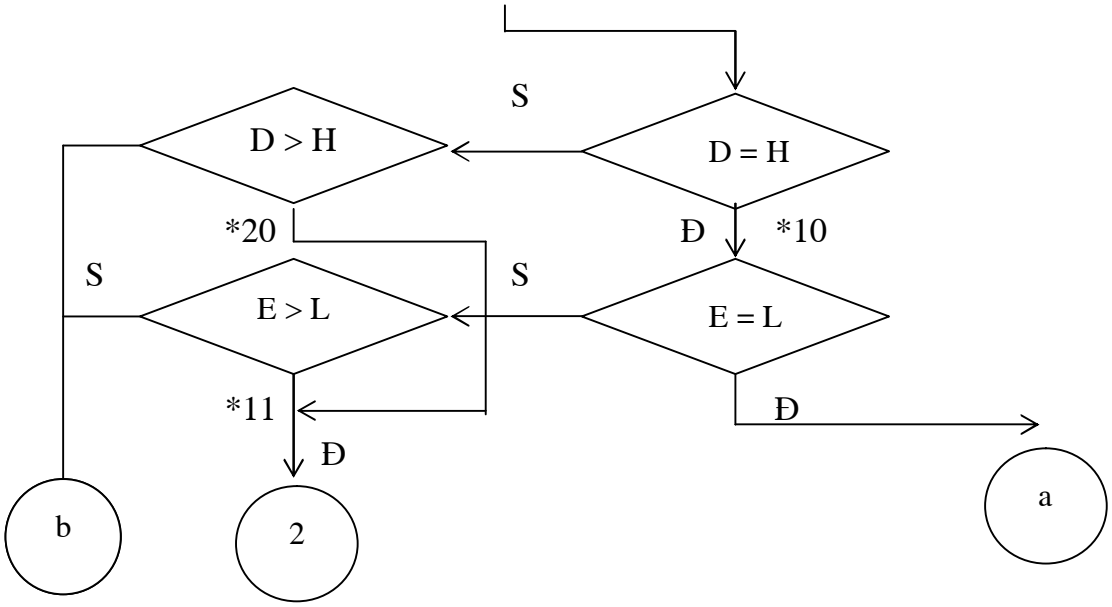
*9 : CALL ORQP

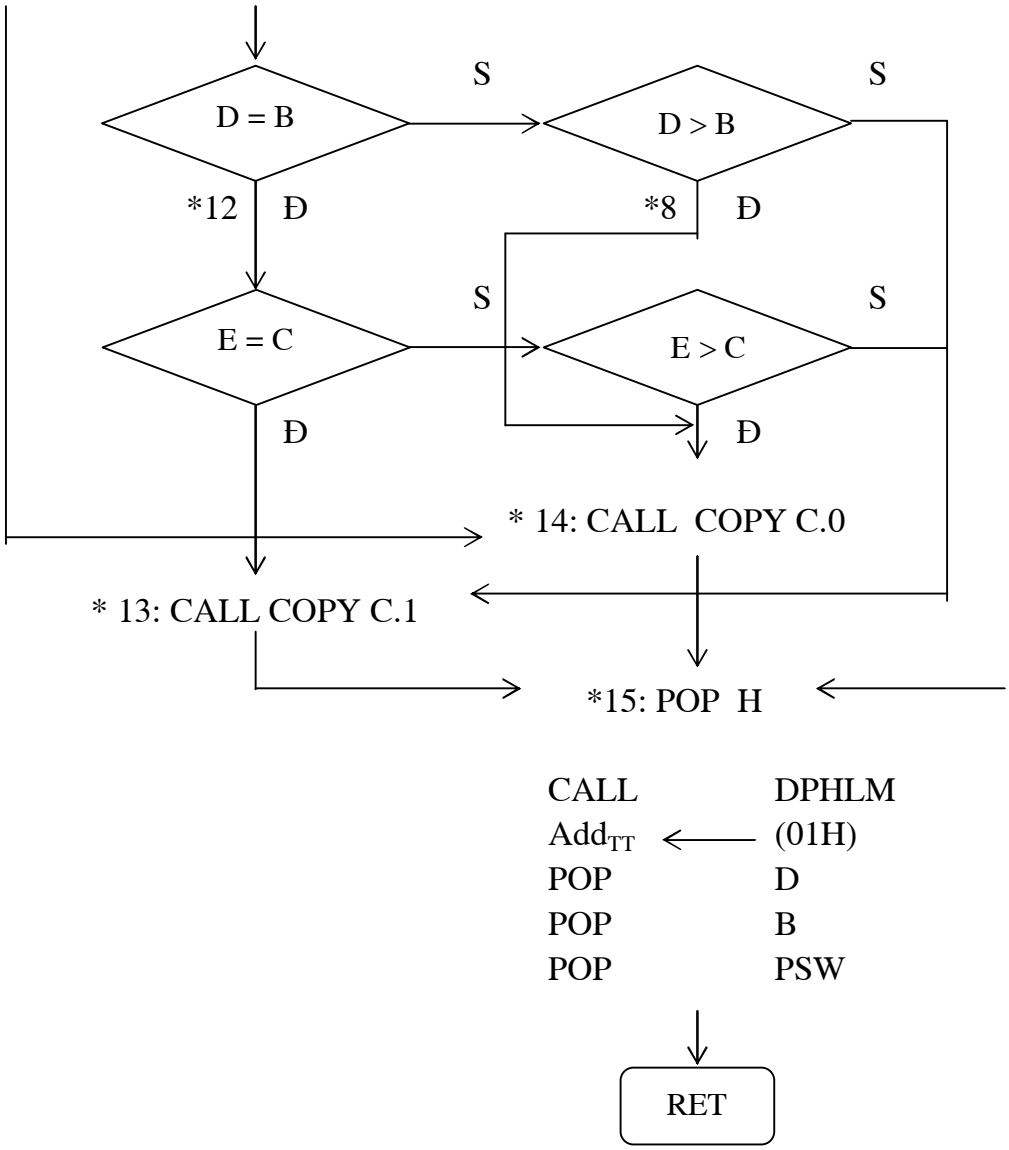
A ← (Add$_{copy+1}$)

Add$_{copy+3}$ ← (A) + (E)

A ← (Add$_{copy+0}$)

Add$_{copy+2}$ ← (A) + (D) + Cy





COPY

PUSH	PSW			JZ	*5
PUSH	B			JMP	*4
PUSH	D		*5 :	LDA	Add_{TT}
PUSH	H			CPI	01H
MVI	A, 01H			JZ	*15
STA	Add_{TT}		*6 :	CALL	DEMP

```
        CALL    XADD
        MVI     A, 83H
        STA     Add_CT79
        MVI     A,06H
        STA     Add_DT79
        CALL    DAUCHAM
*1 :    CALL    KYBRD
        CPI     10H
        JM      *16
        CPI     'ADD'
        JZ      *2
        JMP     *1
*2 :    LDA     Add_TT
        CPI     01H
        JZ      *15
*3 :    CALL    DEMP
        MOV     H, D
        MOV     L, E
        MVI     A,01H
        STA     Add_TT
        CALL    XADD
        MVI     A, 83H
        STA     Add_CT79
        MVI     A, 5BH
        STA     Add_DT79
        CALL    DAUCHAM
*4 :    CALL    KYBRD
        CPI     10H
        JM      *17
        CPI     'ADD'

        JP      *11
        JMP     *14
*10 :   MOV     A,E
        CMP     L
        JZ      *15
        JP      *11
        MOV     B, D
        MOV     C,E
        CALL    SUBBCHL
        MVI     01H
        STA     Add_TT
        CALL    XADD
        MVI     A, 83H
        STA     Add_CT79
        MVI     A, 4 F  H
        STA     Add_DT79
        CALL    DAUCHAM
*7 :    CALL    KYBRD
        CPI     10H
        JM      *18
        CPI     'ADD'
        JZ      *8
        JMP     *7
*8 :    LDA     Add_TT
        CPI     01H
        JZ      *15
*9 :    CALL    DEMP
        LDA     Add_copy+1
        ADD     E
        STA     Add_copy+3
        LDA     Add_copy+0
        ADC     D
        STA     Add_copy+2
        MOV     A,D
        CMP     H
        JZ      *10

*13 :   CALL    COPY C.1
        JMP     *15
*14 :   CALL    COPY C.0
*15 :   POP     H
        CALL    DPHLM
        MVI     A, 01H
```

```
        JMP     *14                     STA     Add_TT
*11     MOV     A,D                     POP     D
        CMP     B                       POP     B
        JZ      *12                     POP     PSW
        JP      *14                     RET
        JMP     *13             *16 :   CALL    PHIMSO
*12     MOV     A,E                     JMP     *1
        CMP     C               *17 :   CALL    PHIMSO
        JZ      *13                     JMP     *4
        JP      *14             *18 :   CALL    PHIMSO
        JMP     *13                     JMP     *7
```

Vùng nhớ Add_{copy} dùng để lưu trữ kết quả của các phép cộng trừ cặp thanh ghi phục vụ cho các chương trình con SUBBCHL và COPY C.1 vùng nhớ Add_{copy} nằm trong vùng nhớ Add_{REC}.

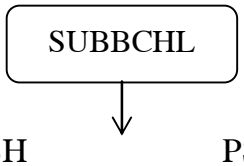
Add_{copy+0} : 9FD2H	(C) – (L)	
Add_{copy+1} : 9FD3H	(B) – (H) – Cy	
Add_{copy+2} : 9FD4H	(A) + (E)	A ⟵ (B) – (H) - Cy
Add_{copy+3} : 9FD5	(A) + (D) + Cy	A ⟵ (C) – (L)

Các chương trình con được gọi :

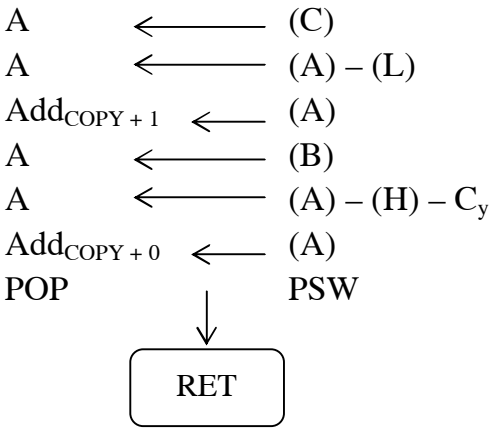
SUBBCHL

SUBBCHL: Là chương trình con có tác vụ trừ nội dung thanh ghi BC cho nội dung trong HL.

- Input : Nạp số bị trừ vào BC
  Nạp số trừ vào HL
- Output : Hiệu số được nạp vào 2 ô nhớ Add_{copy+0} và Add_{copy+1} không làm thay đổi nội dung các thanh ghi.



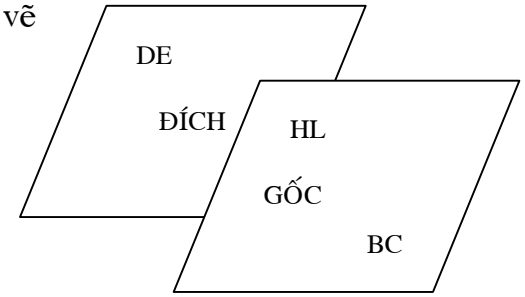
PUSH PSW



SUBBCHL

PUSH	PSW	SBB	H
MOV	A,C	STA	$Add_{COPY + 0}$
SUB	L	POP	PSW
STA	$Add_{COPY + 1}$	RET	
MOV	A,B		

• COPY C.O : Đây là chương trình con sao chép dữ liệu từ vùng này qua vùng khác. Hai vùng gốc và đích không giao nhau hoặc giao nhau như hình vẽ

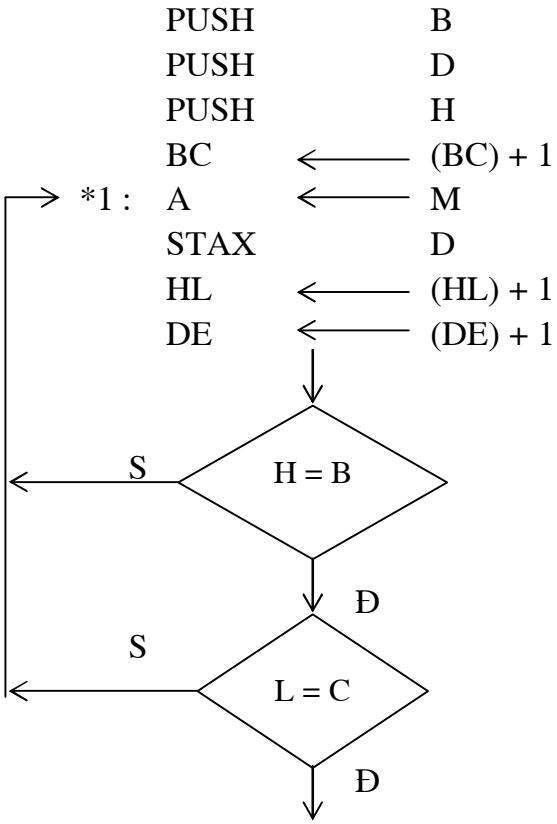


• Input : Nạp địa chỉ đầu của vùng gốc vào HL
Nạp địa chỉ cuối của vùng gốc vào BC
Nạp địa chỉ đầu của vùng đích vào DE

• Output : Không làm thay đổi nội dung các thanh ghi.

COPY C.0

PUSH PSW

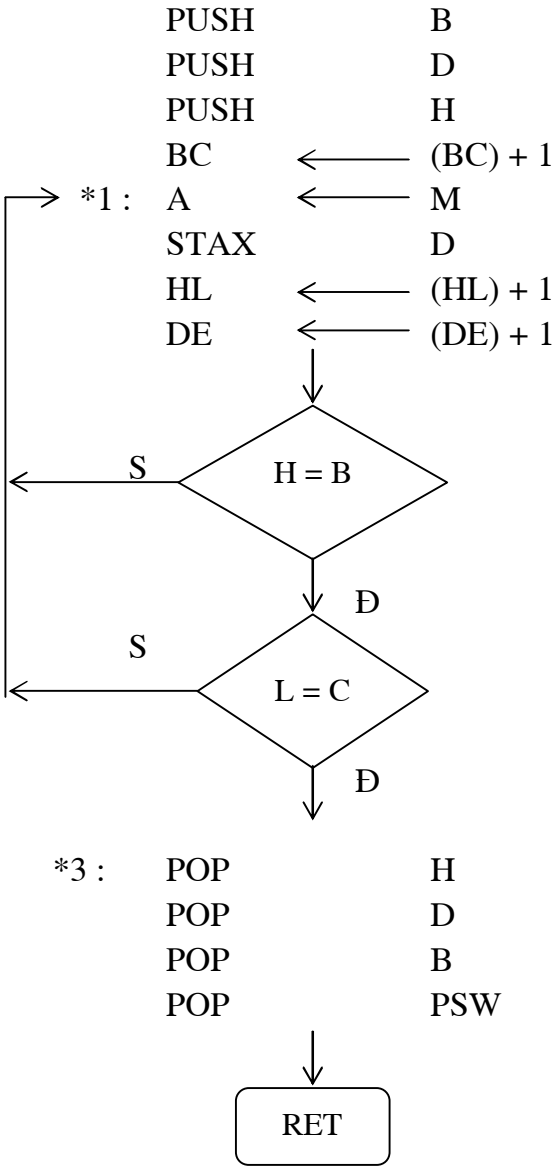


COPY C.0

- Input : Nạp số bị trừ vào BC
  Nạp số trừ vào HL
- Output : Hiệu số được nạp vào 2 ô nhớ Add_{copy+0} và Add_{copy+1} không làm thay đổi nội dung các thanh ghi.

COPY C.0

PUSH PSW



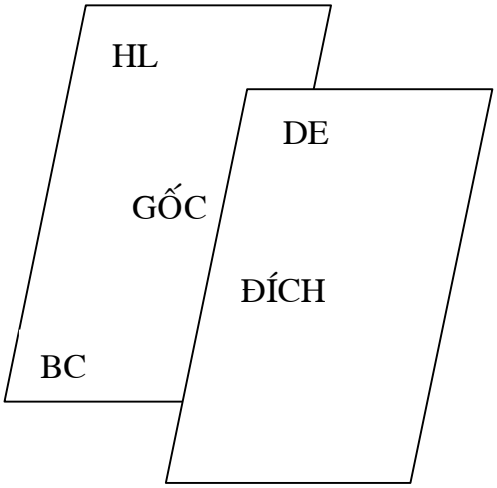
COPY C.0

```
        PUSH    PSW
        PUSH    B
        PUSH    D
        PUSH    H
        INX     B
*1 :    MOV     A, M
        STAX    D
        INX     H
        JZ      *2
        JMP     *1
*2:     MOV     A, C
        CMP     L
        JZ      *3
        JMP     *1
*3 :    POP     H
        POP     D
```

INX	D	POP	B
MOV	A, B	POP	PSW
CMP	H	RET	

COPY C.1

- COPY C.1 Đây là chương trình con chép đoạn dữ liệu trùng nhau như hình vẽ.

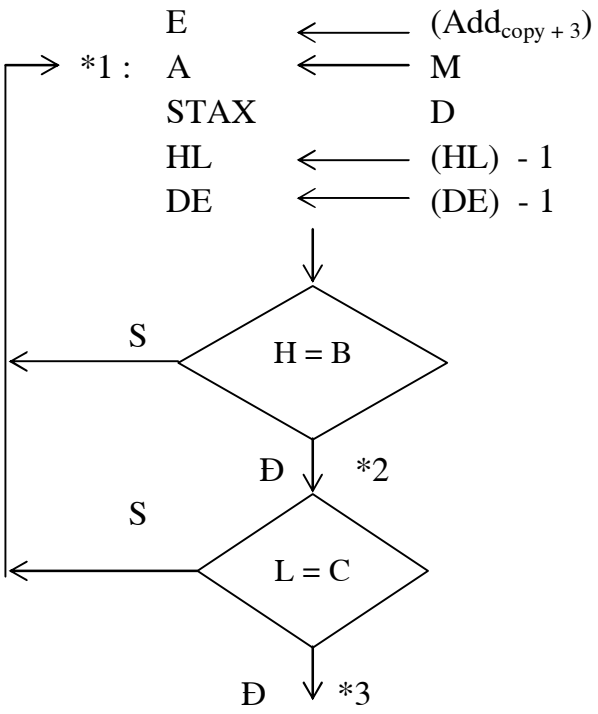


Input : Nạp địa chỉ đầu của vùng gốc vào HL
Nạp địa chỉ cuối của vùng vào BC
Thanh ghi D lấy dữ liệu từ Add_{copy+2}
Thanh ghi E lấy dữ liệu từ Add_{copy+3}

Output : Không thay đổi nội dung các thanh ghi.

COPY C.1

PUSH		PSW
PUSH		B
PUSH		D
PUSH		H
DE	←	HL
HL	←	BC
BC	←	DE
BC	←	(BC) – 1
D	←	$(Add_{copy + 2})$



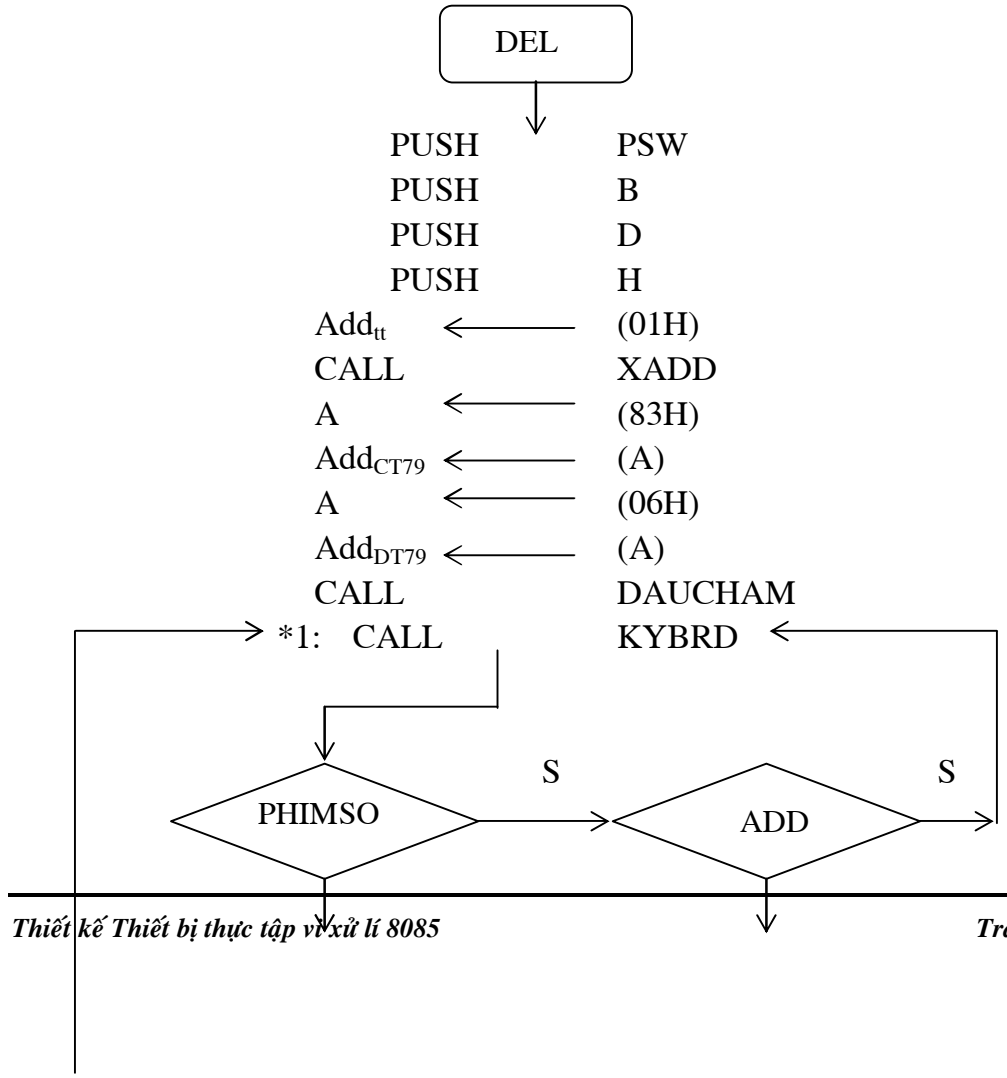

```
*3 :  POP   H
      POP   D
      POP   B
      POP   PSW
```

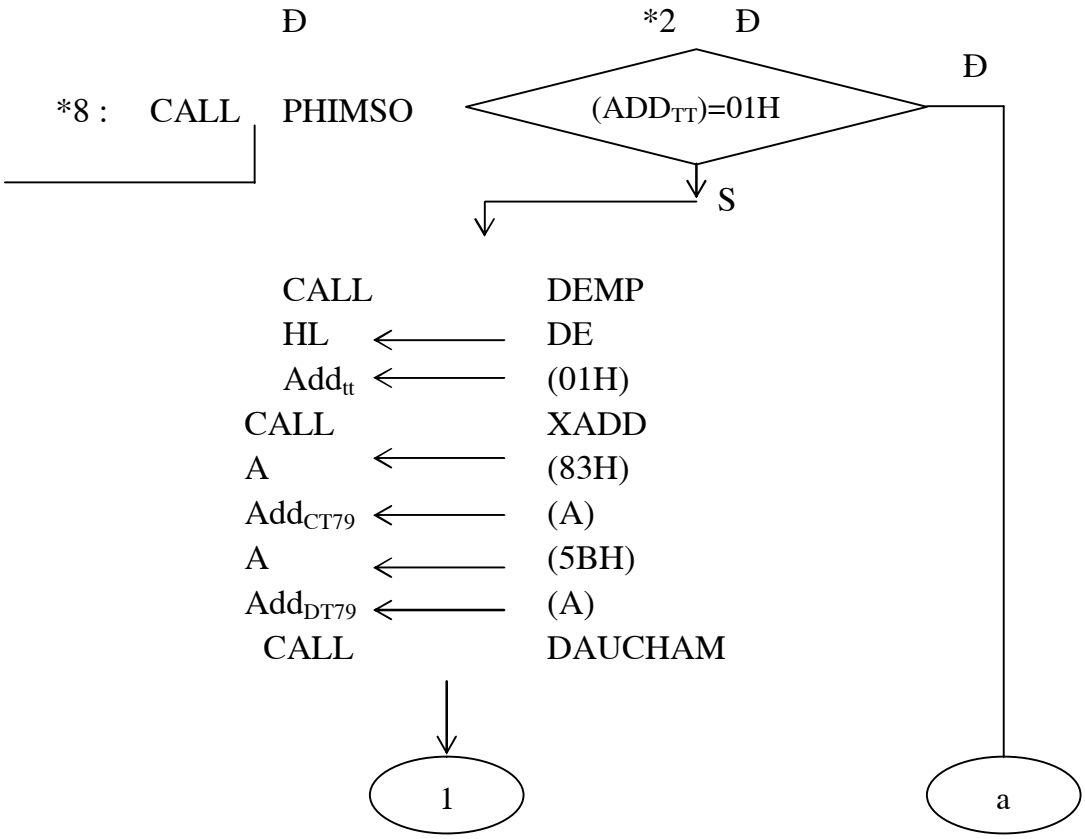

```
PUSH   PSW
PUSH   B
PUSH   D
PUSH   H
XCHG
MOV    H,B
MOV    L,C
MOV    B,D
MOV    C,E
LDA    Add copy + 2
MOV    D,A
LDA    Add copy + 3
MOV    E,A
       DCX   D
       MOV   A, B
       CMP   H
       JZ    *2
       JMP   *1
*2:    MOV   A, C
       CMP   L
       JZ    *3
       JMP   *1
*3 :   POP   H
       POP   D
       POP   B
       POP   PSW
```

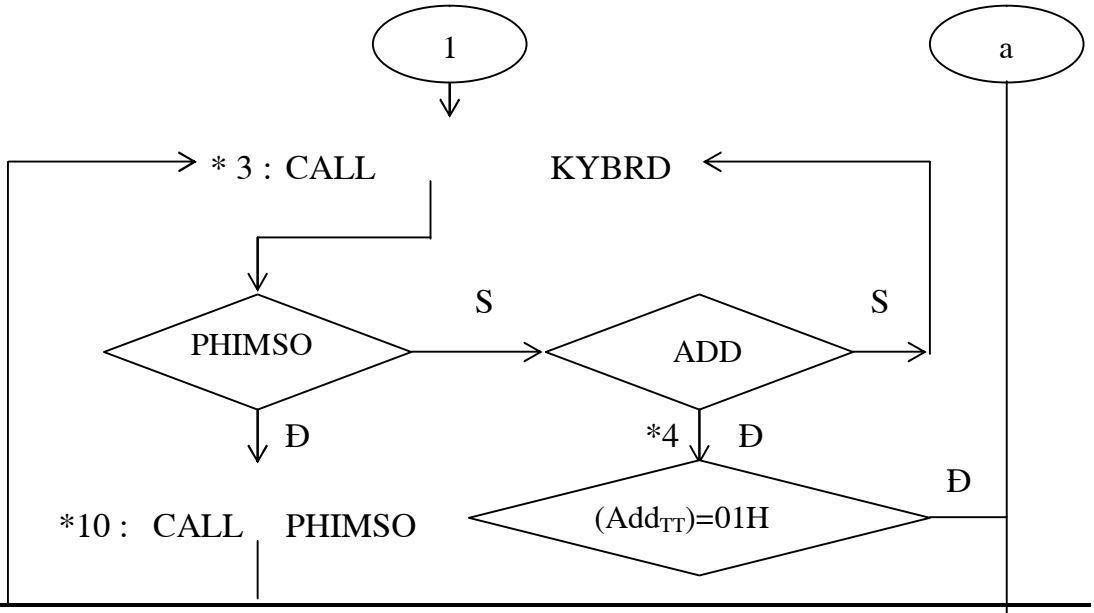
```
        DCX     B                       RET
*1 :    MOV     A, M
        STAX    D
        DCX     H
```

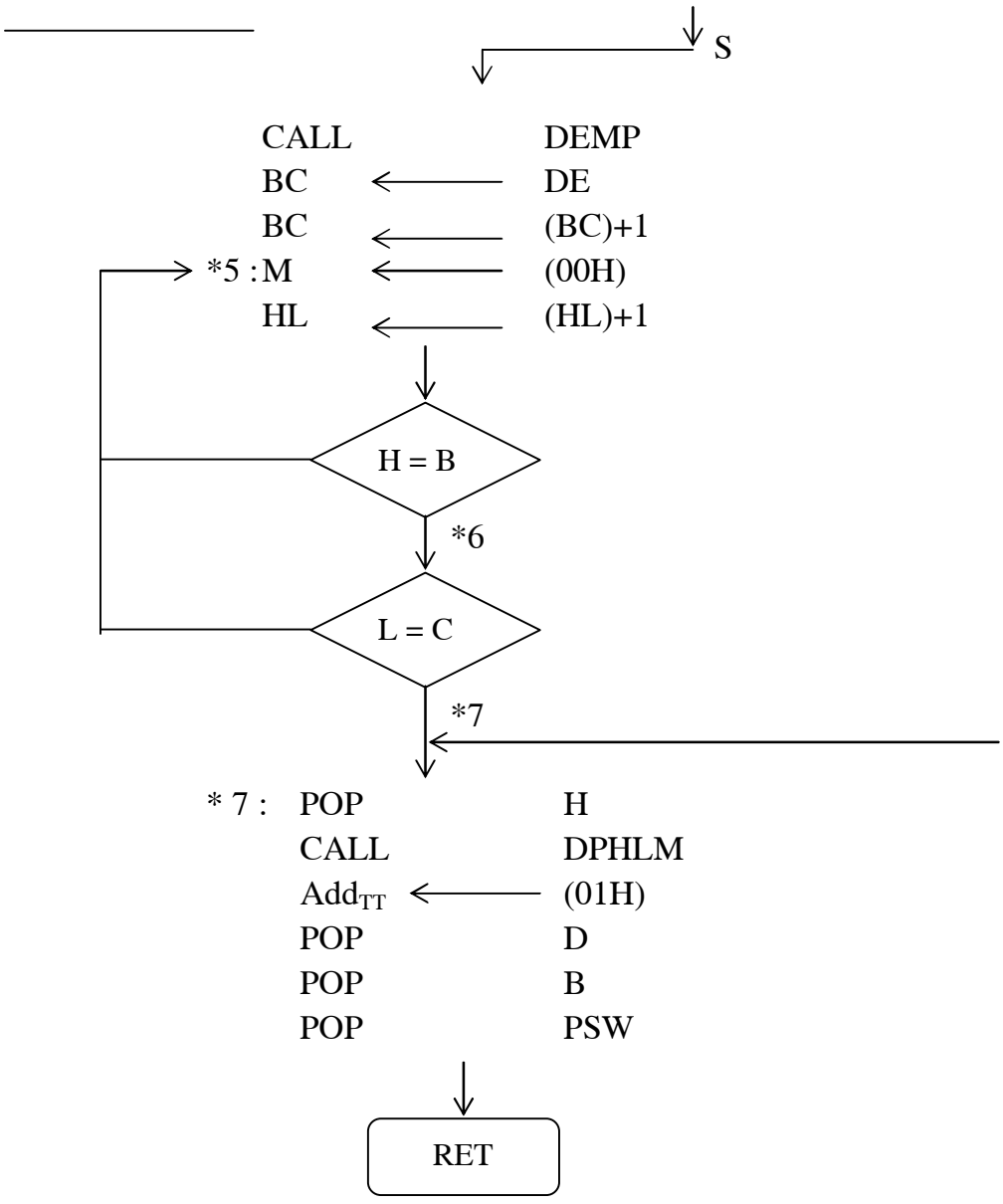
- Các chương trình con sau đây xem phụ lục
  + DAUCHAM
  + DEMP
  + XADD

- DEL : Là chương trình con dễ xoá (nạp 00H) một vùng dữ liệu
- Input : Nạp địa chỉ đầu của vùng cần xoá
  Nạp địa chỉ cuối của vùng cần xoá
- Output : Chương trình không làm thay đổi nội dung các thanh ghi




Đ
*2
Đ
Đ
*8 : CALL PHIMSO
(ADDTT)=01H
S
CALL DEMP
HL ← DE
Addtt ← (01H)
CALL XADD
A ← (83H)
AddCT79 ← (A)
A ← (5BH)
AddDT79 ← (A)
CALL DAUCHAM
1
a


1
a
* 3 : CALL KYBRD
PHIMSO
S
ADD
S
Đ
*4
Đ
Đ
*10 : CALL PHIMSO
(AddTT)=01H



DEL

```
PUSH   PSW          |        STA    Add_DT79
PUSH   B            |        CALL   DAUCHAM
PUSH   D            |  *3 :  CALL   KYBRD
PUSH   H            |        CPI    10H
MVI    A, 01H       |        JM     *10
STA    Add_TT       |        CPI    'ADD'
CALL   XADD         |        JZ     *4
MVI    A, 83H       |        JMP    *3
```

```
        STA     Add_CT79        *4 :    LDA     Add_TT
        MVI     A, 06H                  CPI     01H
        STA     Add_DT79                JZ      *7
        CALL    DAUCHAM                 CALL    DEMP
*1 :    CALL    KYBRD                   MOV     B, D
        CPI     10H                     MOV     C, E
        JM      *8                      INX     B
        CPI     'ADD'           *5 :    MOV     M, 00H
        JZ      *2                      INX     H
        JMP     *1                      MOV     A, H
*2 :    LDA     Add_TT                  CMP     B
        CPI     01H                     JZ      *6
        IZ      *7                      JMP     *5
        CALL    DEMP            *6 :    MOV     A, L
        MOV     H, D                    CMP     C
        MOV     L, E                    JZ      *7
        MVI     A, 01H                  JMP     *5
        STA     Add_TT          *7 :    POP     H
        CALL    XADD                    CALL    DPHLM
        MVI     A, 83H                  MVI     A, 01H
        STA     Add_CT79                STA     Add_TT
        MVI     A, 5B                   POP     D

        POP     B                       JMP     *1
        POP     PSW             *9 :    CALL    PHIMSO
        RET                             JMP     *3
*8 :    CALL    PHIMSO
```

Các chương trình con được gọi :

- DAUCHAM (xem phụ lục)
- XADD (xem phụ lục)
- DEMP (xem phụ lục)
- KYBRD (xem chương 4)

- REG : Đây là chương trình con có chức năng hiển thị nội dung và nạp nội dung các thanh ghi.
- Input : Nội dung các thanh ghi.
- Output : Hiển thị nội dung các thanh ghi.
  Add_{TT} = 01H

XEM GIẢI THUẬT **REG** CUỐI CHƯƠNG 4

REG

```
PUSH    F
STA     Add_REG+0
MOV     A, H
STA     Add_REG+6
MOV     A, L
STA     Add_REG+7
POP     H
MOV     A, L
STA     Add_REG+1
MOV     A, B
STA     AdD_REG+2
MOV     A, C
STA     Add_REG+3
MOV     A, D
STA     Add_REG+4
MOV     A, E
STA     Add_REG+5
*1 :    MVI     A, 01H
        STA     Add_TT
        LXI     H, Add_REG+0
        MVI     B, 77
        CALL    KYTUB
        CALL    DPM
*2 :    CALL    KYBRD
        CPI     10H
        JM      *18
        CPI     'UP'
        JZ      *3
```

```
      CPI     "DOWN"
      JZ      *17
      CPI     "REG"
      JZ      *19
*3 :  LDA     Add_TT
      CPI     00H
      JZ      *4
      JMP     *5
*4 :  CALL    ORQP
      MOV     M, A
*5 :  INX     H
*6 :  MOV     A, H
      CPI     9 F H
      JZ      *7
      JMP     *1
*7 :  MOV     A, L
      CPI     D 2 H
      JZ      *8
      CPI     D3
      JZ      *9
      CPI     D 4
      JZ      *10
      CPI     D 5
      JZ      *11
      CPI     D 6
      JZ      *12
      CPI     D 7
      JZ      *13
      CPI     D 8
      JZ      *14
      CPI     D 9
      JZ      *15
      JMP     *1
*8 :  MVI     B, 77 H
      CALL    KYTUB
      JMP     *16
*9 :  MVI     B, 71 H
      CALL    KYTUB
      JMP     *16
*10 : MVI     B, 7 C H
      CALL    KYTUB
      JMP     *16
*11 : MVI     B, 39 H
      CALL    KYTUB
      JMP     *16
*12 : MVI     B, 5 E H
      CALL    KYTUB
      JMP     *16
*13 : MVI     B, 79 H
      CALL    KYTUB
      JMP     *16
*14 : MVI     B, 76 H
      CALL    KYTUB
      JMP     *16
*15 : MVI     B, 38 H
      CALL    KYTUB
      JMP     *16
*16 : CALL    DPM
      MVI     A, 01 H
      STA     Add_TT
      JMP     *2
*17 : DCX     H
      JMP     *6
*18 : CALL    PHIMSO
      JMP     *2
* 19 : LXI    H, Add_REG+0
      MOV     B, M; lấy AF
      INX     H; thông qua
      MOV     C, M; ngăn xếp
      PUSH    B; và BC
```

INX	H	MOV	E, M
MOV	B, M	XCHG	
INX	H	POP	D
MOV	D, M	CALL	DPHLM
INX	H	MVI	A, 01 H
MOV	E, M	STA	Add_{TT}
PUSH	D	POP	PSW; lấy AF
INX	H ; lấy HL		thông qua BC
MOV	D, M;thông		và ngăn xếp
INX	H; qua DE	RET	

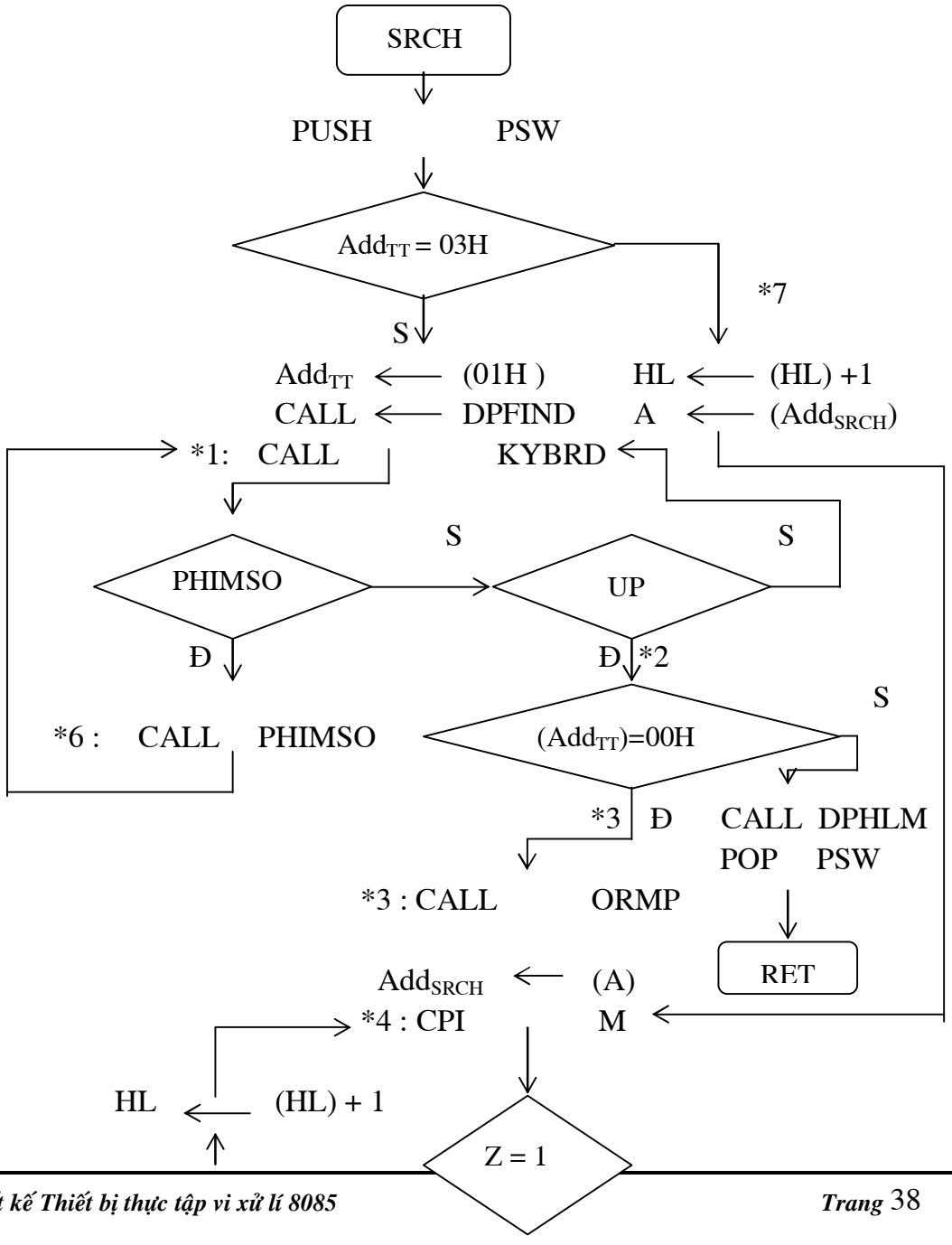
- DPM : Chương trình con hiển thị nội dung ô nhớ có địa chỉ trong HL (xem phụ lục).
- PHIMSO : Hiển thị địa chỉ và dữ liệu (xem phụ lục).
- KYBRD : Chương trình con quét phím.
- KYTUB : Hiển thị kí tự và dấu bằng (xem phụ lục)
- ORMP : Hợp hai ô quét phím (xem phụ lục)

Vùng nhớ Add_{REG} dùng để lưu nội dung của các thanh ghi.

Add_{REG+0} : 9 F D 2H	A
Add_{REG+1} : 9 F D 3H	F
Add_{REG+2} : 9 F D 4H	B
Add_{REG+3} : 9 F D 5H	C
Add_{REG+4} : 9 F D 6H	D
Add_{REG+5} : 9 F D 7H	E
Add_{REG+6} : 9 F D 8H	H
Add_{REG+7} : 9 F D 9H	L

SRCH

- SRCH : Là chương trình con có tác vụ tìm kiếm dữ liệu trong bộ nhớ.
- Input : Nạp dữ liệu cần tìm.
  Nạp địa chỉ cuối của vùng cần xoá
- Output : Hiển thị địa chỉ và dữ liệu vừa tìm được. Thay đổi nội dung lặp thanh ghi HL.
  Add_{TT} = 03H



S

Đ

*5 :	Add_{TT}	03H
	CALL	DPHLM
	POP	PSW

RET

SRCH

	PUSH	PSW
	LDA	Add_{TT}
	CPI	03H
	JZ	*7
	MVI	A, 01H
	STA	Add_{TT}
	CALL	DPFIND
*1 :	CALL	KYBRD
	CPI	10H
	JM	*6
	CPI	‘UP’
	JZ	*2
	JMP	*1
*2 :	LDA	Add_{TT}
	CPI	00H
	JZ	*3
	CALL	DPHLM
	POP	PSW
	RET	

	CALL	ORMP
	STA	Add_{SRCH}
*4 :	CMP	M
	JZ	*5
	INX	H
	JMP	*4
*5 :	MVI	A, 03H
	STA	Add_{TT}
	CALL	DPHLM
	POP	PSW
	RET	
*6 :	CALL	PHIMSO
	JMP	*1
*7 :	INX	H
	LDA	Add_{SRCH}
	JMP	*4

* Các chương trình được gọi :

- DPFIND (xem phụ lục)

- DPHLM (xem phụ lục)
- KYBRD (xem chương 4)
- ORMP (xem phụ lục)
- PHIMSO (xem chương 4)

STEP

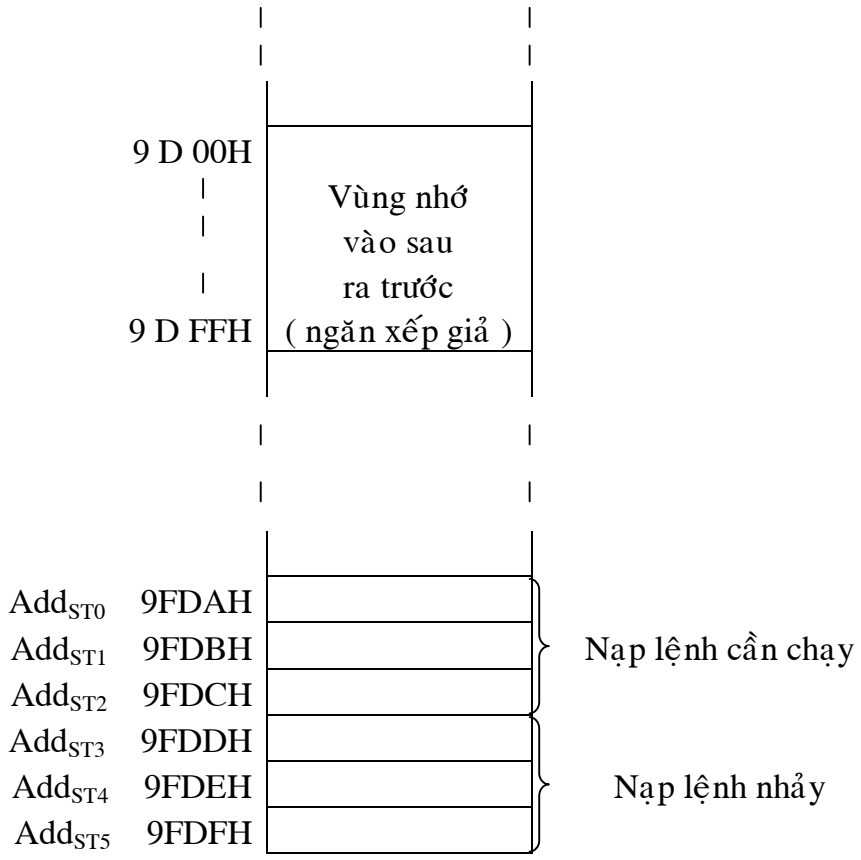
- STEP : Đây là một chương trình cho phép chạy từng lệnh của chương trình người sử dụng.
- Input : Nhận địa chỉ hiện hành vào HL.
- Output : Chỉ thay đổi thanh ghi HL.

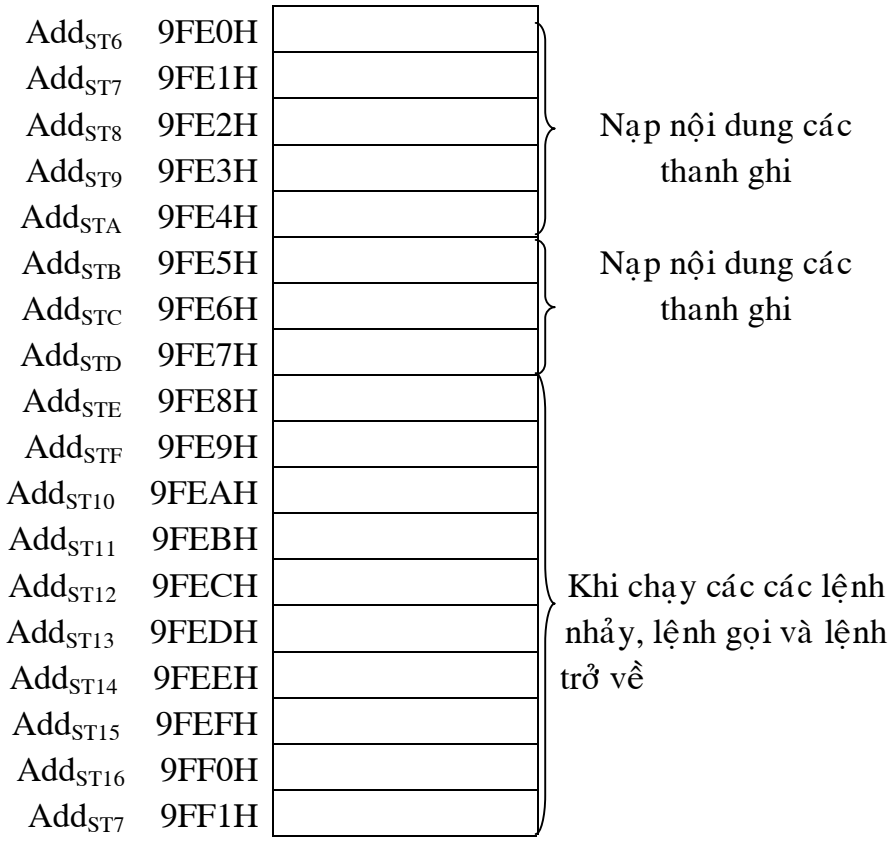
Các chương trình con được gọi :

LMB : Chương trình nhận biết được lệnh sắp thực hiện mấy byte.

DPHLM : Hiển thị địa chỉ vào dữ liệu tại địa chỉ đó.

Chương trình sử dụng các vùng nhớ sau :





XEM GIẢI THUẬT **STEP** CUỐI CHƯƠNG 4

STEP

	PUSH	B	JZ	*ST2
	MOV	A,M	CPI	31H
	STA	Add$_{STO}$	JZ	*ST2
	MVI	A,OOH	CPI	33H
	STA	Add$_{ST17}$	JZ	*ST13
	CALL	LMB	CPI	20H
	MOV	A,C	JZ	*ST3
	CPI	3OH	CPI	10H
	JZ	*ST1	JZ	*ST5
	CPI	32H	CPI	11H
			JZ	*ST6
*ST1:	INX	H	CPI	A0H
	MOV	A,M	JZ	*ST22
	STA	Add$_{ST10}$	JMP	*ST20

```
        INX     H
        MOV     A,M
        STA     Add_ST11
        MVI     A,_
        STA     Add_ST1
        MVI     A,_
        STA     Add_ST2
        SHLD    Add_ST12
        MVI     A,C3
        STA     Add_ST3
        MVI     A,_
        STA     Add_ST4
        MVI     A,_
        STA     Add_ST5
        JMP     *ST21
*ST2 :  MOV     A,M
        CPI     32H
        JZ      *ST10
        INX     H
*ST4 :  MOV     A,M
        STA     Add_ST1
        INX     H
*ST20 : MOV     A,M
        STA     Add_ST2
        JMP     *ST7
*ST10 : INX     H
        MOV     A,M
        CPI     00H
        JZ      *ST11
        JMP     *ST4
*ST11:  STA     Add_ST1
        INX     H
        MOV     A,M
        LHLD    Add_STE
        INX     H
        MVI     A,01H
        STA     Add_TT
```

```
*ST22 : STA     Add_ST2
        SHLD    Add_STE
        MVI     A,C3H
        STA     Add_CT79
        MVI     A,01H
        CALL    DELAY 0.1
        MVI     A,C3H
        STA     Add_ST3
        MVI     A,_
        STA     Add_ST4
        MVI     A,_
        STA     Add_ST5
        LDA     Add_ST8
        MOV     B,A
        LDA     Add_ST9
        MOV     C,A
        LHLD    Add_ST6
        PUSH    H
        POP     PSW
        LHLD    Add_STA
        XCHG
        LHLD    Add_STC
        JMP     Add_STO
*ST12 : SHLD    Add_STC
        PUSH    PSW
        POP     H
        SHLD    Add_ST6
        MOV     A,B
        STA     Add_ST8
        MOV     A,C
        STA     Add_ST9
        XCHG
        SHLD    Add_STA
        MOV     H,B
        MOV     L,C
        INX     H
        JMP     *ST9
```

```
        POP     B
        RET
*ST3 :  INX     H
        MOV     A,M
        STA     Add_ST1
        MVI     A,00H
        STA     Add_ST2
        JMP     *ST7
*ST5 :  MVI     A,00H
        STA     Add_ST1
        STA     Add_ST2
        JMP     *ST7
*ST6 :  SHLD    Add_ST15
        MVI     A,00H
        STA     Add_ST1
        STA     Add_ST2
        MVI     A,C3H
        STA     Add_ST+3
        MVI     A,_
        STA     Add_ST+4
        MVI     A,_
        STA     Add_ST+5
        CALL    *ST19
        LDA     Add_ST14
        MOV     L,A
        MVI     H,9DH
        MOV     B,M
        INX     H
        MOV     C,M
        DCRA
        DCRA
        STA     Add_ST14
        STA     Add_TT
        POP     B
        RET

*ST13 : INX     H
```

```
*ST7 :  SHLD    Add_STE
        MVI     A,C3H
        STA     Add_ST3
        MVI     A,_
        STA     Add_ST4
        MVI     A,_
        STA     Add_ST5
*ST21:  LDA     Add_ST8
        MOV     B,A
        LDA     Add_ST9
        MOV     C,A
        LHLD    Add_ST6
        PUSH    H
        POP     PSW
        LHLD    Add_STA
        XCHG
        LHLD    Add_STC
        JMP     Add_STO
*ST8 :  SHLD    Add_STC
        PUSH    PSW
        POP     H
        SHLD    Add_ST6
        MOV     A,B
        STA     Add_ST8
        MOV     A,C
        STA     Add_ST9
        XCHG
        SHLD    Add_STA
        LHLD    Add_STE
        INX     H
*ST9 :  CALL    DPHLM
        MVI     A,01H
        INR     A
        INR     A
        STA     Add_ST14
        LHLD    Add_ST12
        MOV     B,H
```

```
        MOV     A,M             MOV     C,L
        STA     Add_ST10        MOV     L,A
        INX     H               MVI     H,9DH
        MOV     A,M             MOV     M,B
        STA     Add_ST11        INX     H
        MVI     A,_             MOV     M,C
        STA     Add_ST1         MVI     A,01H
        MVI     A,_             STA     Add_ST17
        STA     Add_ST2         LHLD    Add_ST10
        SHLD    Add_ST12        RET
        MVI     A,C3
        STA     Add_ST3 *17 :   LDA     Add_ST17
        MVI     A,_             CMP     00H
        STA     Add_ST4         JZ      *ST15
        MVI     A,_             JMP     *ST9
        STA     Add_ST5 *18 :   LHLD    Add_ST15
        JMP     *ST21           INX     H
*ST14 : LHLD    Add_ST10        JMP     *ST9
        JMP     *ST9
*ST15 : LHLD    Add_ST12 *19 :  JMP     *ST21
        INX     H
        JMP     *ST9
*ST16 : LDA     Add_ST14
```

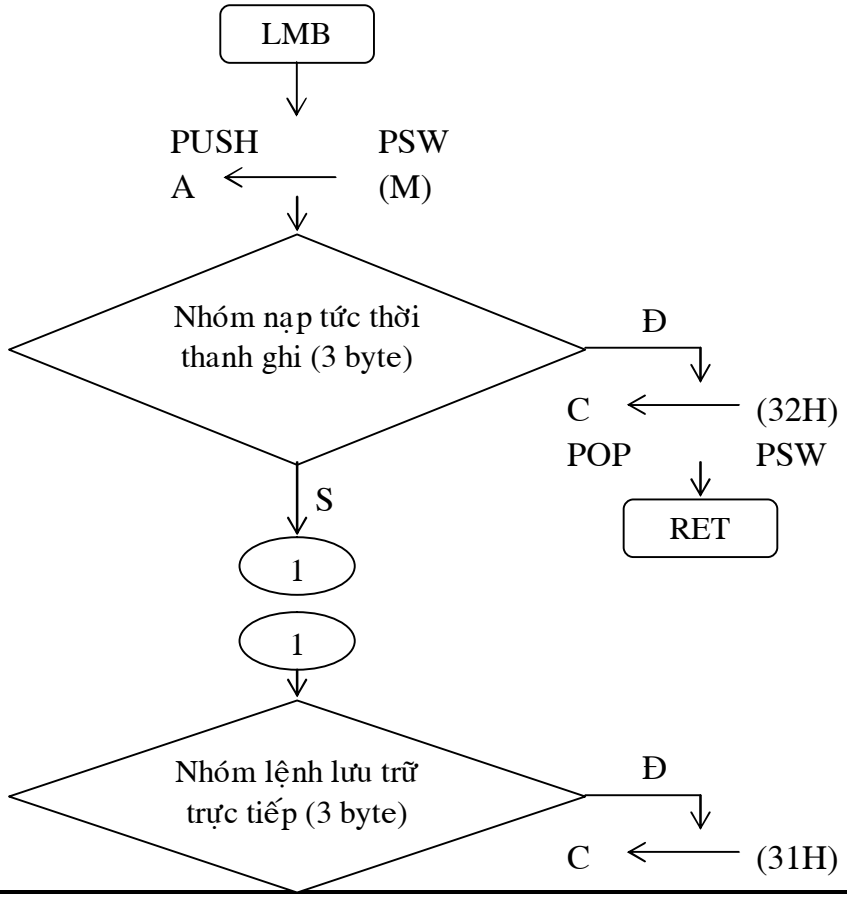
LMB

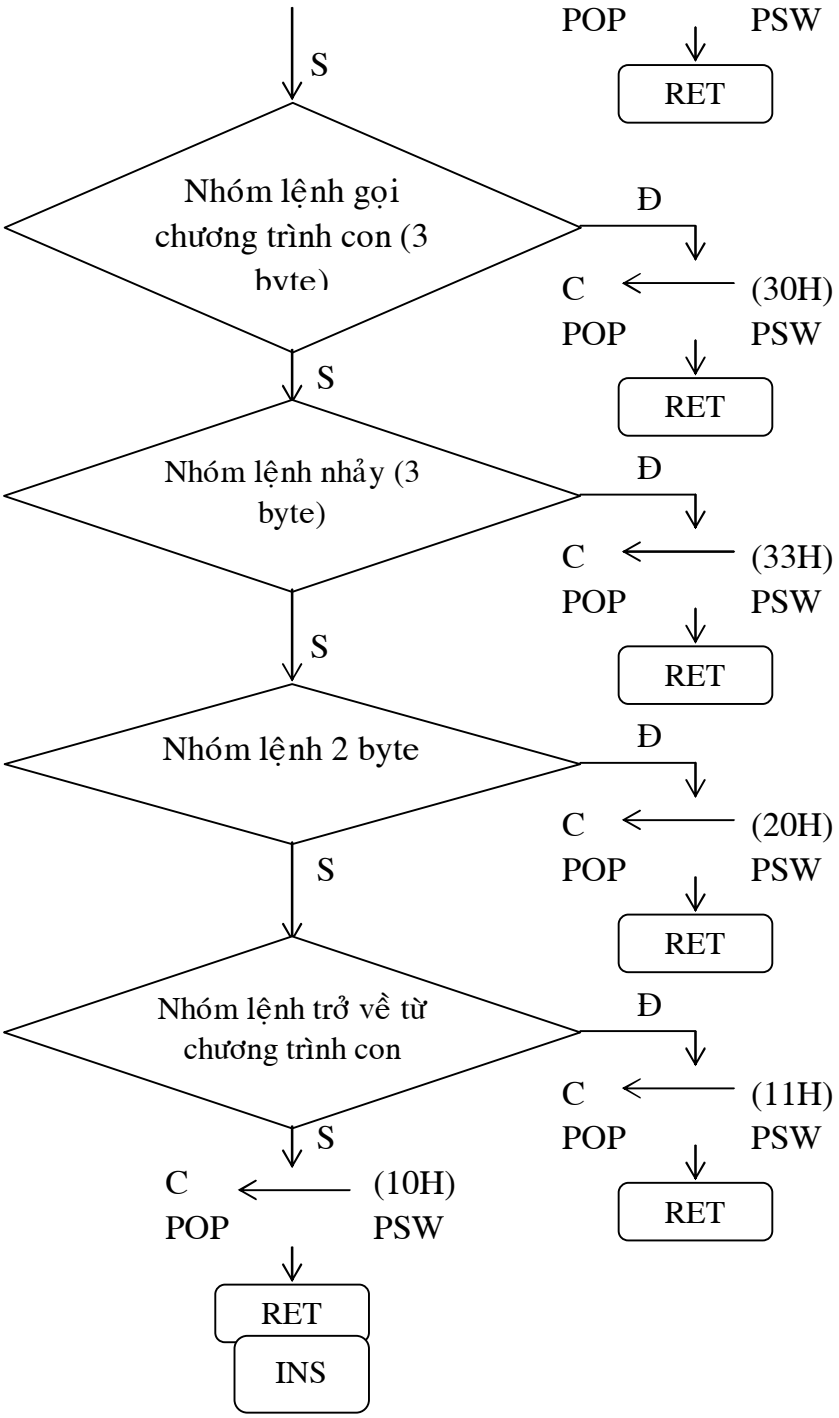
- LMB : Đây là một chương trình con có nhiệm vụ nhận biết lệnh mấy byte.
- Input : HL là địa chỉ chứa lệnh đó

- Output : C = 32H : Khi lệnh đó thuộc nhóm lệnh nạp tức thời cặp thanh ghi (3 byte)
  C = 31H : Khi lệnh đó thuộc nhóm lệnh lưu trữ trực tiếp (3 byte)
  C = 30H : Khi lệnh đó thuộc nhóm lệnh gọi chương trình con (3 byte)
  C = 33H : Khi lệnh đó thuộc nhóm lệnh nhảy (3 byte)
  C = 20H : Khi lệnh đó thuộc nhóm lệnh 2 byte
  C = 11H : Khi lệnh đó thuộc nhóm lệnh trở về từ chương trình con
  C = 10H : Khi lệnh đó thuộc nhóm lệnh 1 byte (kể cả những mã không thuộc bộ lệnh)

  Ngoài thanh ghi C chương trình không ảnh hưởng tới các thanh ghi khác.

  Do chương trình đơn giản nên chỉ trình bày thuật giải. Được gọi từ chương trình STEP và chương trình SDC.



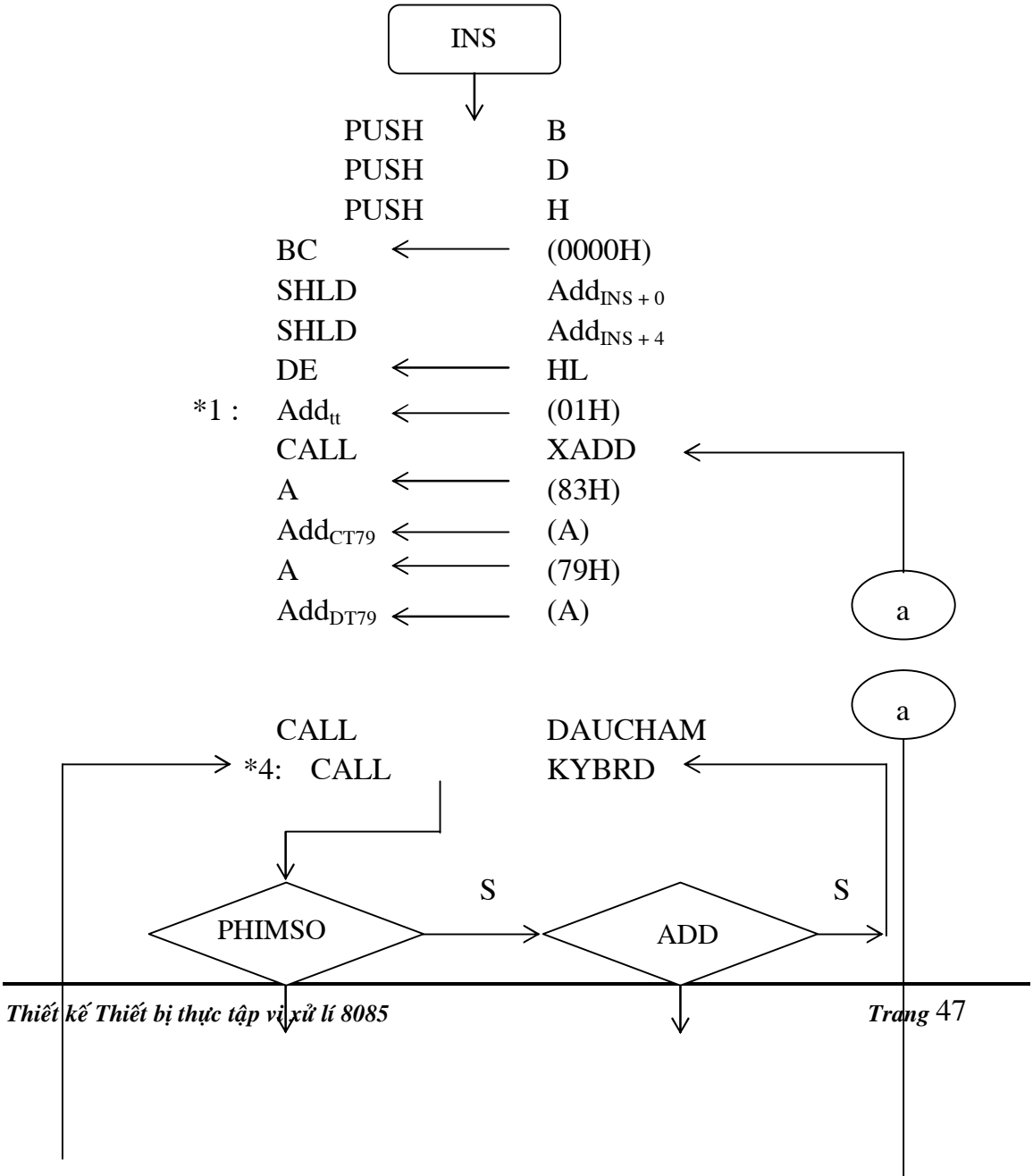


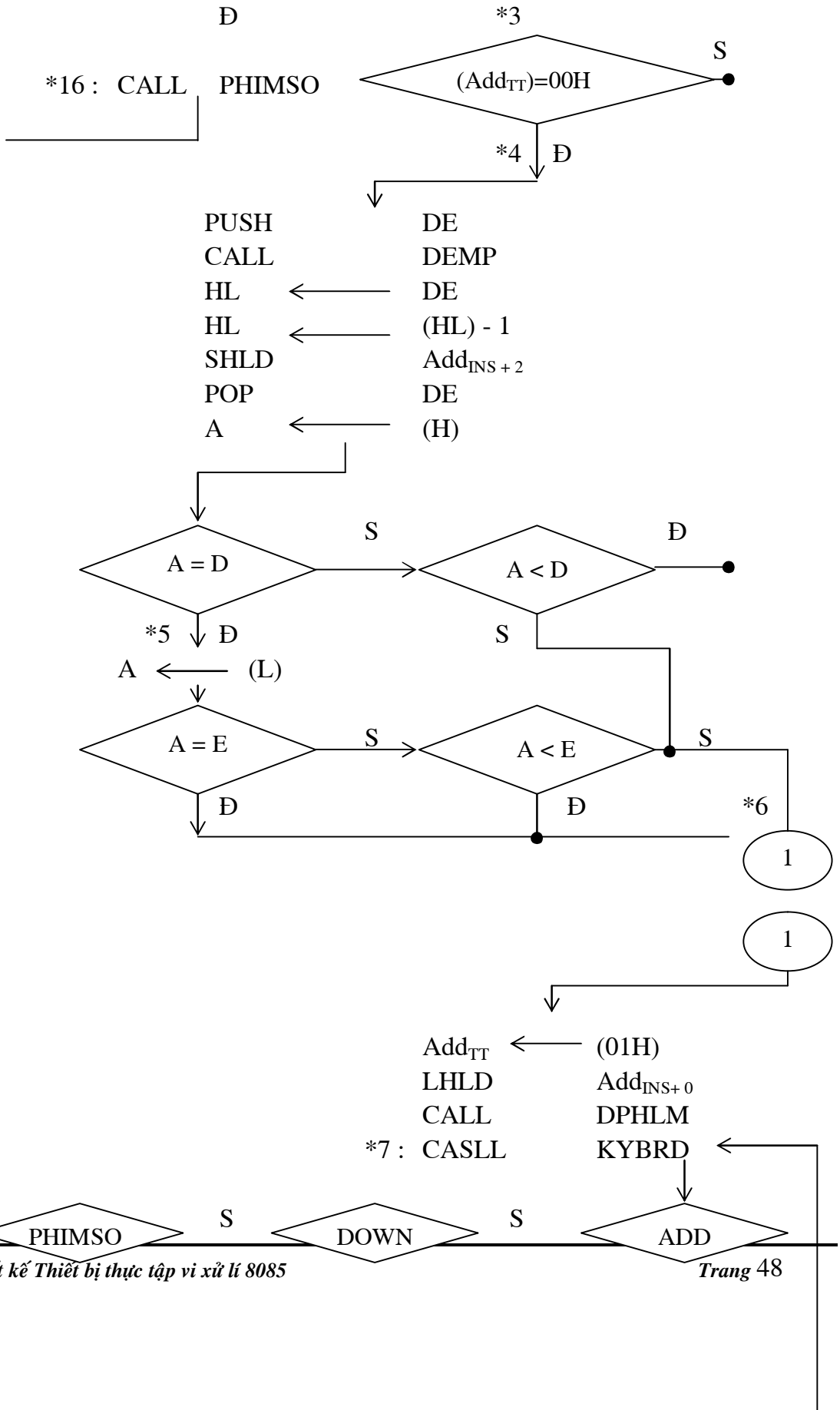
- INS : Đây là một chương trình con cho phép chèn dữ liệu tại địa chỉ hiện hành và tự sửa địa chỉ nếu muốn.
- Input : Nạp địa chỉ cuối vùng INS , nghĩa là chương trình chỉ có tác dụng trong khoảng từ địa chỉ hiện hành đến địa chỉ cuối mới nạp. Nếu sửa địa chỉ thì nạp địa chỉ lệnh đầu tiên vùng cần sửa,

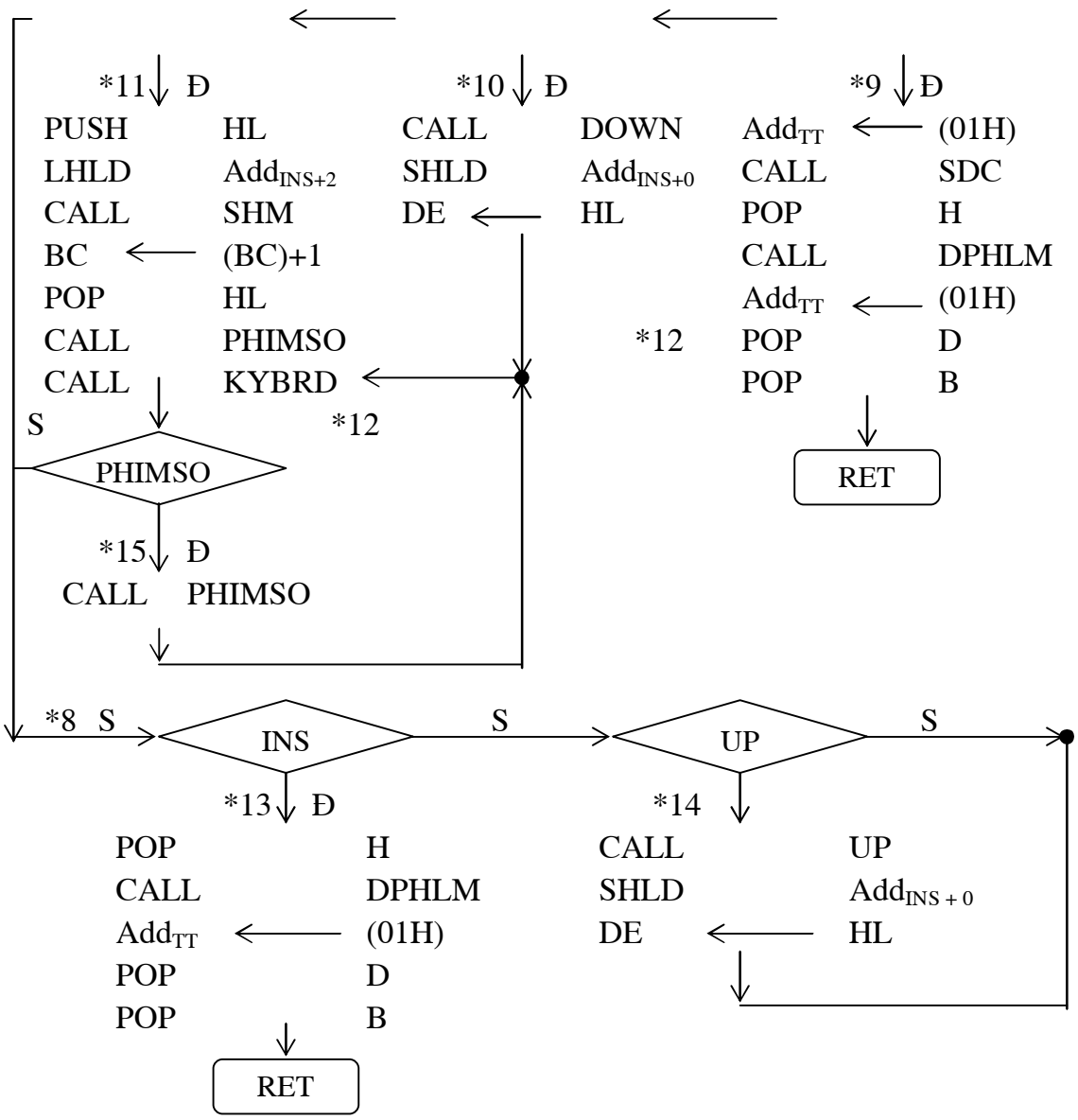
nghĩa là chương trình sửa địa chỉ chỉ có tác dụng từ địa chỉ mới nạp tới địa chỉ cuối vùng INS

- Output : Ngoài AF không thanh ghi nào bị điều chỉnh. Vùng địa chỉ sử dụng cho INS :

Add_{INS+0} : 9FDD		Địa chỉ trở về sau khi
Add_{INS+1} : 9FDE		INS
Add_{INS+2} : 9FDF		Địa chỉ kế cuối vùng
Add_{INS+3} : 9FE0		INS
Add_{INS+4} : 9FE1		Nạp địa chỉ ô nhớ
Add_{INS+5} : 9FE2		cần chèn




Đ
*3
S
*16 : CALL PHIMSO
(Add$_{TT}$)=00H
*4
Đ
PUSH DE
CALL DEMP
HL ← DE
HL ← (HL) - 1
SHLD Add$_{INS + 2}$
POP DE
A ← (H)
A = D
S
A < D
Đ
*5
Đ
S
A ← (L)
A = E
S
A < E
S
Đ
Đ
*6
1
1
Add$_{TT}$ ← (01H)
LHLD Add$_{INS+ 0}$
CALL DPHLM
*7 : CASLL KYBRD
S
PHIMSO
S
DOWN
S
ADD




```
        PUSH    B
        PUSH    D
        PUSH    H
        LXI     B,0000H
        SHLD    Add_INS + 0
        SHLD    Add_INS + 4
        MOV     D,H
        MOV     E,L
*1 :    MVI     A,01
        STA     Add_TT
        JMP     *1
        JMP     *6
*5 :    MOV     A,L
        CMP     E
        JZ      *1
        JM      *1
*6 :    MVI     A,01H
        STA     Add_TT
        LHLD    Add_INS + 0
        CALL    DPHLM
*7 :    CALL    KYBRD
```

```
        CALL    XADD
        MVI     A,38H
        STA     Add_CT79
        MVI     A,79H
        STA     Add_DT79
        CALL    DAUCHAM
*2 :    CALL    KYBRD
        CPI     10H
        JM      *16
        CPI     'ADD'
        JZ      *3
        JMP     *2
*3 :    LDA     Add_TT
        CPI     00H
        JZ      *4
        JMP     *1
*4 :    PUSH    D
        CALL    DEMP
        MOV     H,D
        MOV     L,E
        DCX     H
        SHLD    Add_INS + 2
        POP     D
        MOV     A,H
        CMP     D
        JZ      *5
        CPI     'ADD'
        JZ      *9
        CPI     'DOWN'
        JZ      *10
        CPI     10H
        JM      *11
*8 :    CPI     'INS'
        JZ      *13
        CPI     'UP'
        JZ      *14
        JMP     *7
*9 :    MVI     A,01H
        STA     Add_TT
        CALL    SDC
        POP     H
        CALL    DPHLM
        MVI     A,01H
        STA     Add_TT
        POP     D
        POP     B
        RET
*10 :   CALL    DOWN
        SHLD    Add_INS +0
        MOV     D,H
        MOV     E,L
        JMP     *12

*11 :   PUSH    H
        LHLD    Add_INS + 2
        CALL    SHM
        INX     B
        POP     H
        CALL    PHIMSO
*12 :   CALL    KYBRD
        CPI     10H
        JZ      *15H
        STA     Add_TT
        POP     D
        POP     B
        RET
*14 :   CALL    UP
        SHLD    Add_INS + 0
        MOV     D,H
        MOV     E,L
        JMP     *7
```

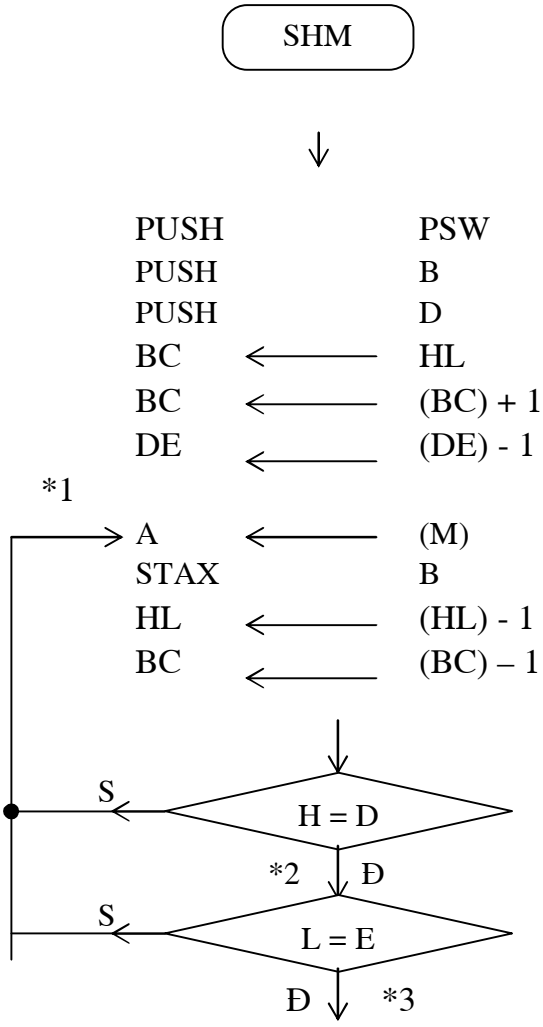
```
        JMP     *8              *15 :   CALL    PHIMSO
*13 :   POP     H                       JMP     *12
        CALL    DPHLM           *16 :   CALL    PHIMSO
        MVI     A,01H                   JMP     *2
```

* Các chương trình con được gọi :

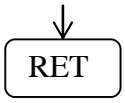
- DAUCHAM : xem phụ lục
- DEMP : xem phụ lục
- DPHLM : xem phụ lục
- DOWN : xem CHƯƠNG 4
- KYBRD : xem CHƯƠNG 4
- PHIMSO : xem CHƯƠNG 4
- SDC : xem CHƯƠNG 4
- SHM : xem CHƯƠNG 4
- XADD : xem phụ lục

SHM

- SHM : Là chương trình con có nhiệm vụ dịch dữ liệu xuống một ô nhớ.
- Input: Nạp HL là địa chỉ giáp chót của vùng INS. Nạp DE là địa chỉ ô nhớ được chèn vào.
- Output: Không điều chỉnh nội dung các thanh ghi



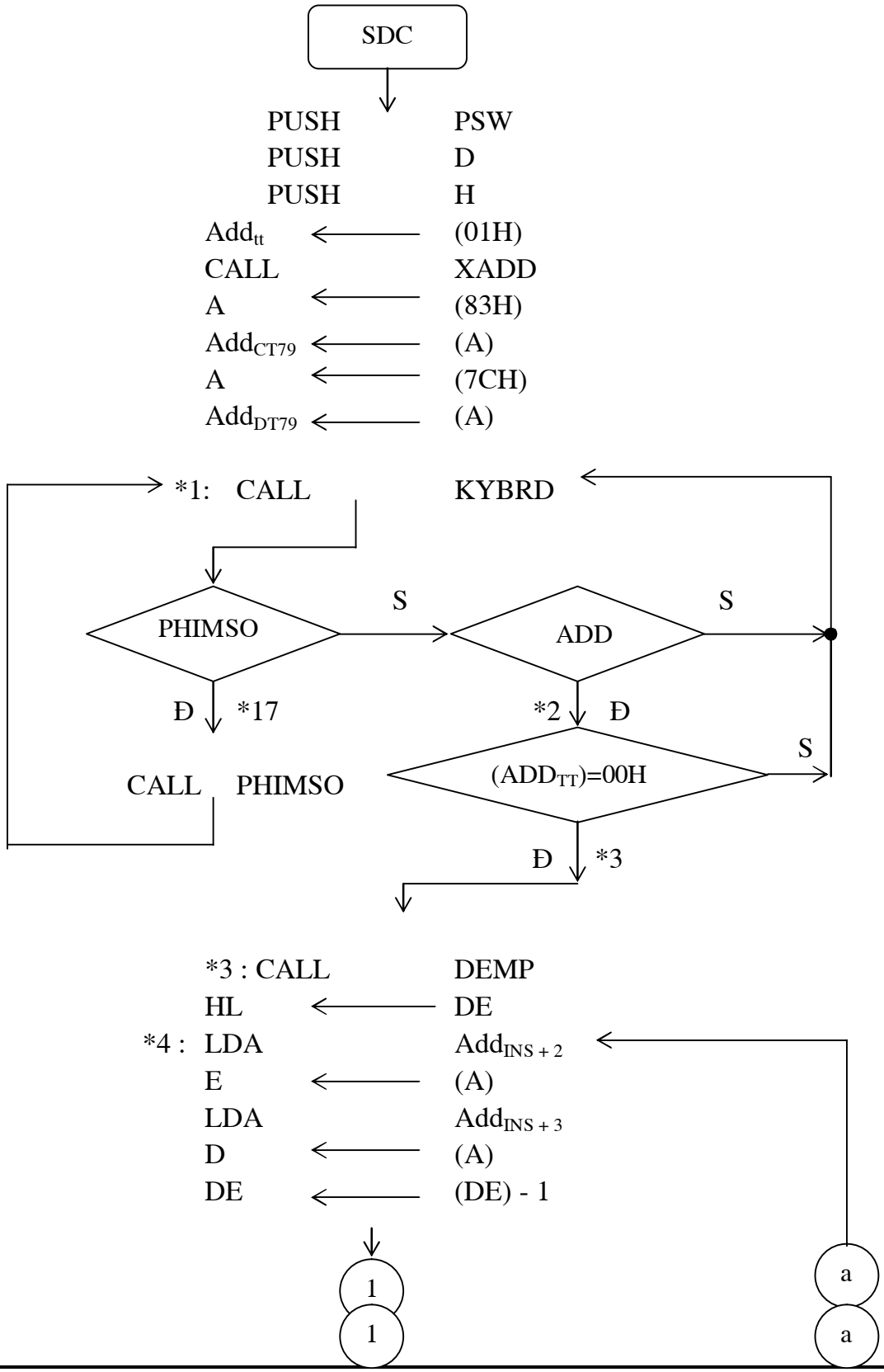
```
POP     D
POP     B
POP     PSW
```

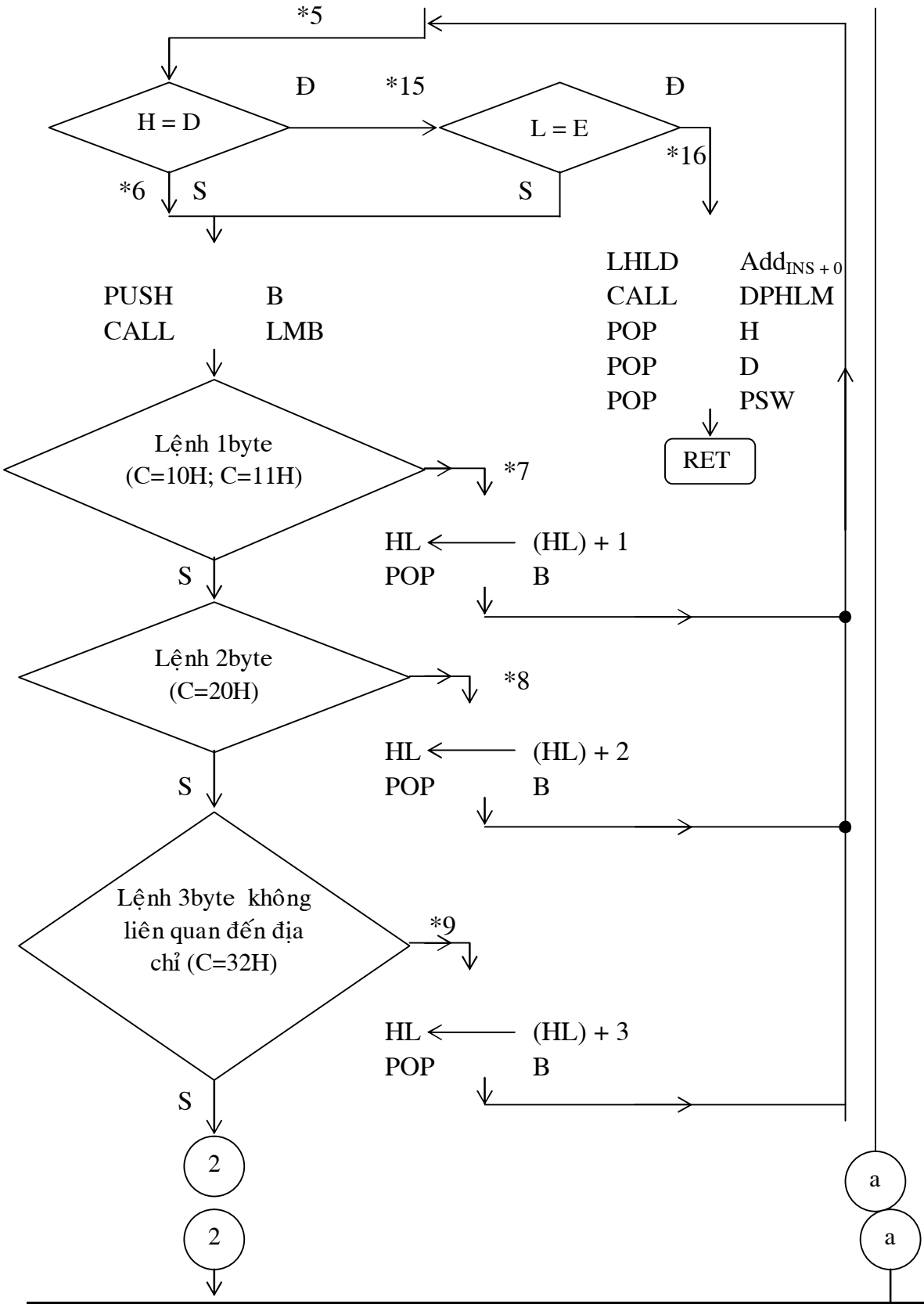


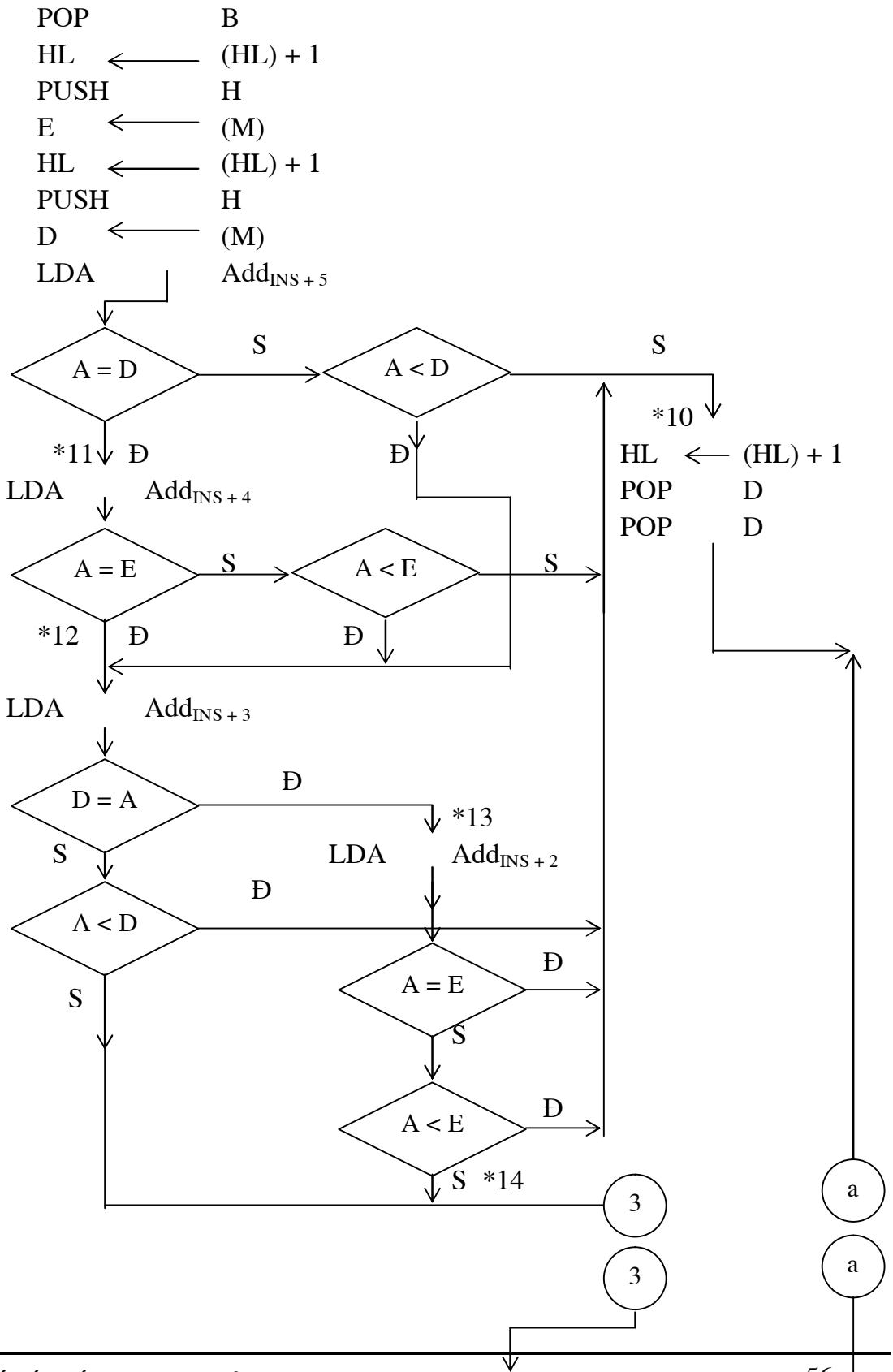
SHM

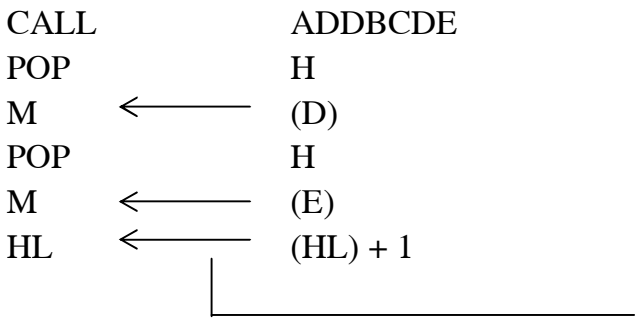
```
        PUSH    PSW
        PUSH    B
        PUSH    D
        MOV     B,H
        MOV     C,L
        INX     B
        DCX     D
*1 :    MOV     A,M
        STAX    B
        DCX     H
        DCX     B
        MOV     A,H
        CMP     D
        JZ      *2
        JMP     *1
*2 :    MOV     A,L
        CMP     E
        JZ      *3
        JMP     *1
*3 :    POP     D
        POP     B
        POP     PSW
        RET
```

- SDC : Là chương trình con có khả năng sửa địa chỉ những lệnh liên quan đến địa chỉ (lệnh nhảy, lệnh gọi, lệnh lưu trữ).
- Input : Nạp BC là khoảng thay đổi địa chỉ.
  Nạp địa chỉ ô nhớ cần chèn vào Add_{INS+4} và Add_{INS+5}
  Nạp địa chỉ giáp chót của vùng cần INS vào Add_{INS+2} và Add_{INS+3}
  Nạp địa chỉ trở về khi thực hiện xong chương trình vào Add_{INS+0} và Add_{INS+1}
  Nạp địa chỉ bắt đầu sửa địa chỉ
- Output : Hiển thị địa chỉ hiện hành trước lúc INS
  Chương trình không làm thay đổi nội dung các thanh ghi


SDC
PUSH PSW
PUSH D
PUSH H
Add_tt ← (01H)
CALL XADD
A ← (83H)
Add_CT79 ← (A)
A ← (7CH)
Add_DT79 ← (A)
*1: CALL KYBRD
PHIMSO
S
ADD
S
Đ
*17
*2
Đ
S
CALL PHIMSO
(ADD_TT)=00H
Đ
*3
*3 : CALL DEMP
HL ← DE
*4 : LDA Add_INS + 2
E ← (A)
LDA Add_INS + 3
D ← (A)
DE ← (DE) - 1
1
1
a
a


*5
Đ
*15
Đ
H = D
L = E
*16
*6
S
S
LHLD Add$_{INS + 0}$
PUSH B
CALL LMB
CALL DPHLM
POP H
POP D
POP PSW
RET
Lệnh 1byte
(C=10H; C=11H)
*7
HL ← (HL) + 1
POP B
S
Lệnh 2byte
(C=20H)
*8
HL ← (HL) + 2
POP B
S
Lệnh 3byte không
liên quan đến địa
chỉ (C=32H)
*9
HL ← (HL) + 3
POP B
S
2
2
a
a


POP B
HL ⟵ (HL) + 1
PUSH H
E ⟵ (M)
HL ⟵ (HL) + 1
PUSH H
D ⟵ (M)
LDA Add INS + 5
A = D
S
A < D
S
*10
HL ⟵ (HL) + 1
POP D
POP D
*11
Đ
Đ
LDA Add INS + 4
A = E
S
A < E
S
*12
Đ
Đ
LDA Add INS + 3
D = A
Đ
*13
S
LDA Add INS + 2
Đ
A < D
Đ
A = E
S
S
A < E
Đ
S *14
3
3
a
a



SDC

```
      PUSH   PSW
      PUSH   D
      PUSH   H
      MVI    A,01H
      STA    Add_TT
      CALL   XADD
      MVI    A,83H
      STA    Add_CT79
      MVI    A,7CH
      STA    Add_DT79
*1 :  CALL   KYBRD
      CPI    10H
      JZ     *17
      CPI    'ADD'
      JZ     *2
      JMP    *1
*2 :  LDA    Add_TT
      CPI    00H
      JZ     *3
      JMP    *1
*3 :  CALL   DEMP
      MOV    H,D
      MOV    L,E
*4 :  LDA    Add_INS + 2
      MOV    E,A
      LDA    Add_INS + 3
      MOV    D,A
      DCX    D
*5 :  MOV    A,H
      CMP    D
      JZ     *15
*6 :  PUSH   B
      CALL   LMB
      CPI    10H
      JZ     *7
      CPI    11H
      JZ     *7
      CPI    20H
      JZ     *8
      CPI    32H
      JZ     *9
      POP    B
      INX    H
      PUSH   H
      MOV    E,M
      INX    H
      PUSH   H
      MOV    D,M
```

```
      LDA     Add_INS + 5
      CMP     D
      JZ      *11
      JM      *12
*10 : INX     H
      POP     D
      POP     D
      JMP     *4
*11 : LDA     Add_INS + 4
      CMP     E
      JZ      *12
      JM      *12
      JMP     *10
*12 : LDA     Add_INS + 3
      CMP     D
      JZ      *13
      JM      *10
      JMP     *14
*13 : LDA     Add_INS + 2
      CMP     E
      JZ      *10
      JM      *10
*14 : CALL    ADDBCDE
      POP     H
      MOV     M,D
      POP     H
      MOV     M,E
      INX     H
      JMP     *4
*15 : MOV     A,L
      CMP     E
      JZ      *16
      JMP     *6
*16 : LHLD    Add_INS +0
      CALL    DPHLM
      POP     H
      POP     D
      POP     PSW
      RET
*17 : CALL    PHIMSO
      JMP     *1
*7 :  INX     H
      POP     B
      JMP     *5
*8 :  INX     H
      INX     H
      POP     B
      JMP     *5
*9 :  INX     H
      INX     H
      INX     H
      POP     B
      JMP     *5
```

* Các chương trình con được gọi :

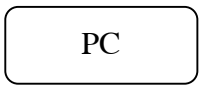
- DEMP : xem phụ lục
- DPHLM : xem phụ lục

- PHIMSO : xem CHƯƠNG 4
- LMP : xem CHƯƠNG 4
- ADDBCDE

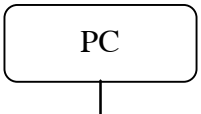
- ADDBCDE : Đây là chương trình con dùng để cộng hai cặp thanh ghi DE và BC
- Input : Nạp DE, BC
- Output : Kết quả trong DE
  Chỉ thanh ghi D , E bị thay đổi.

ADDBCDE

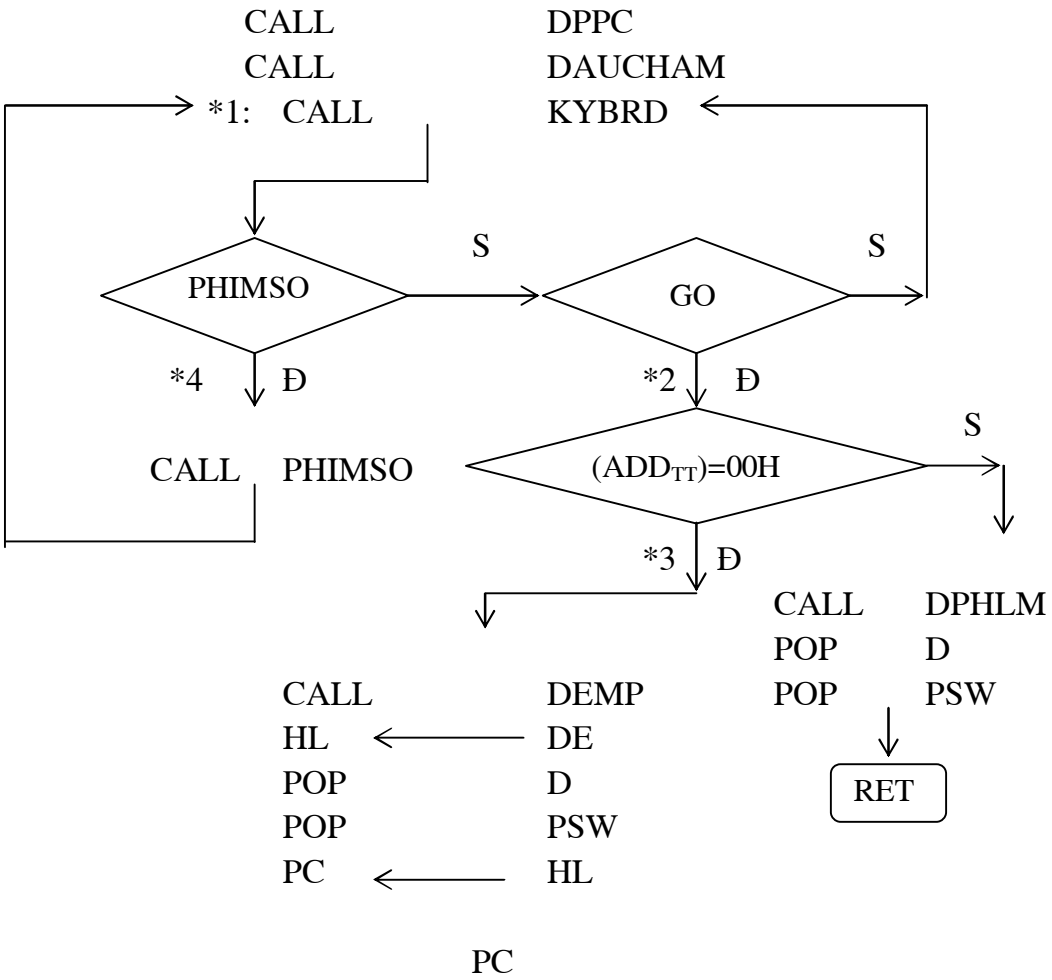
```
PUSH    PSW        |        ADC     B
PUSH    B          |        MOV     D,A
MOV     A,E        |        POP     B
ADD     C          |        POP     PSW
MOV     E,A        |        RET
MOV     A,D        |
```



- PC : Là chương trình con dùng để chạy chương trình của người sử dụng.
- Input : Nạp địa chỉ bắt đầu chạy
- Output : Thực hiện chương trình người sử dụng tới khi nào gặp lệnh dừng




```
PUSH          PSW
PUSH          D
Add_tt  <---  (01H)
```


CALL DPPC
CALL DAUCHAM
*1: CALL KYBRD
PHIMSO
S
GO
S
*4
Đ
*2
Đ
CALL PHIMSO
(ADDTT)=00H
S
*3
Đ
CALL DPHLM
POP D
POP PSW
RET
CALL DEMP
HL DE
POP D
POP PSW
PC HL
PC

```
        PUSH    PSW
        PUSH    D
        MVI     A,01H
        STA     Add_TT
        CALL    DPPC
        CALL    DAUCHAM
*1 :    CALL    KYBRD
        CPI     10H
        JM      *4
        CPI     'GO'
        JZ      *2
        JMP     *1
*2 :    LDA     Add_TT
        CPI     00H
        JZ      *3
        CALL    DPHLM
        POP     D
        POP     PSW
        RET
*3 :    CALL    DEMP
        MOV     H,D
        MOV     L,E
        POP     D
        POP     PSW
        PCHL
```

* Các chương trình con được gọi :

- DEMP : xem phụ lục
- DPHLM : xem phụ lục
- DAUCHAM : xem phụ lục
- PHIMSO : xem CHƯƠNG 4
- KYBRD : xem CHƯƠNG 4
- DPPC : Là chương trình con xuất hiện thông báo nạp PC (xem phụ lục)

Trên đây là một số chương trình con đã được thiết kế và chạy thử nghiệm trên SDK 85 (System Design Kit 8085) của trường ĐHSPKT, phần còn lại được trình bày trong phần phụ lục. Do thời gian có hạn nên không thể thực hiện như mong muốn, nhưng để thiết kế, chạy thử và thi công hơn

30 chương trình quả là một công việc không đơn giản. CHƯƠNG 5 sẽ cung cấp những thông tin hữu ích về vấn đề này.

Chương 5 :
THI CÔNG

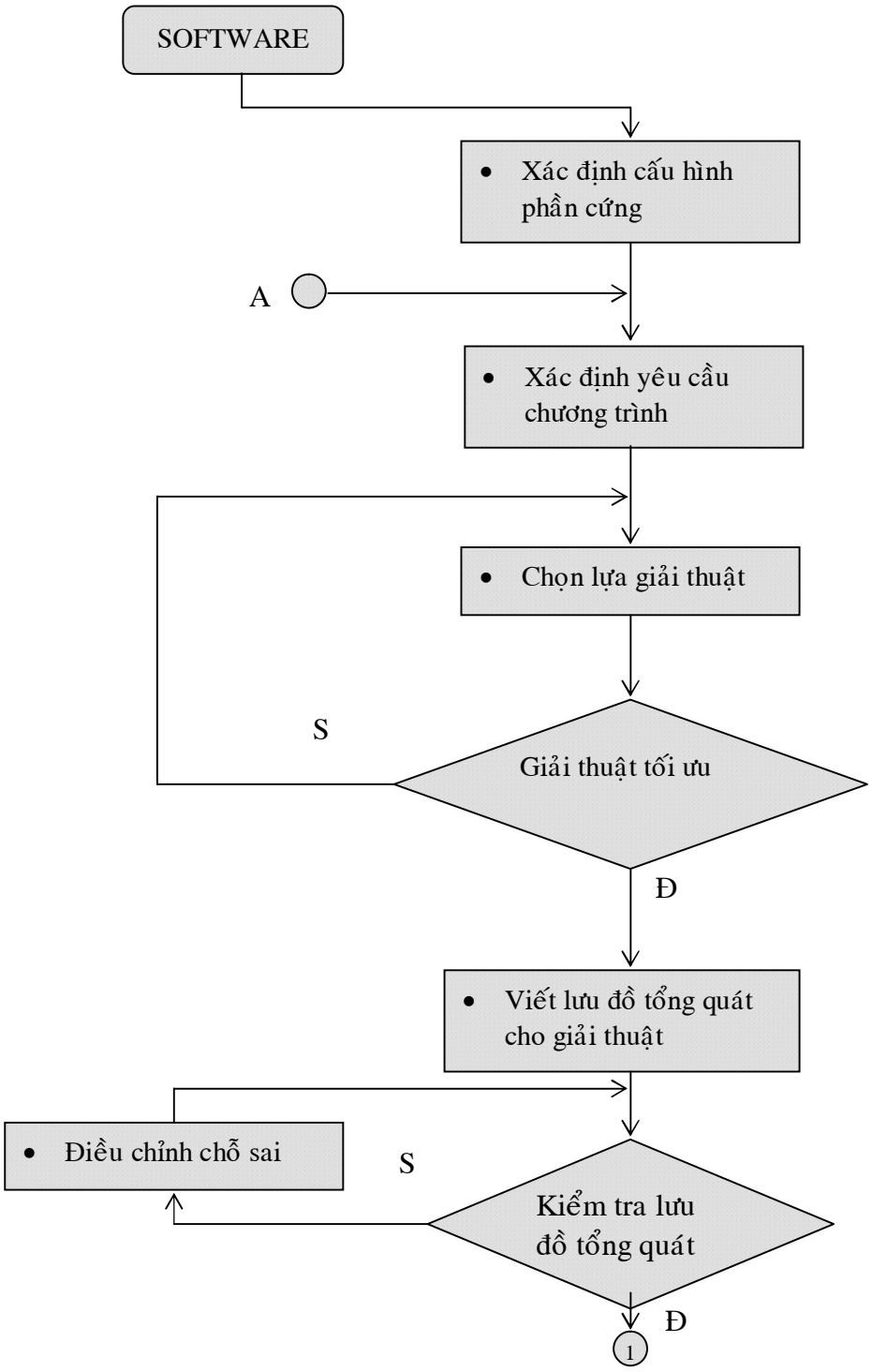
CHƯƠNG 5 : THI CÔNG

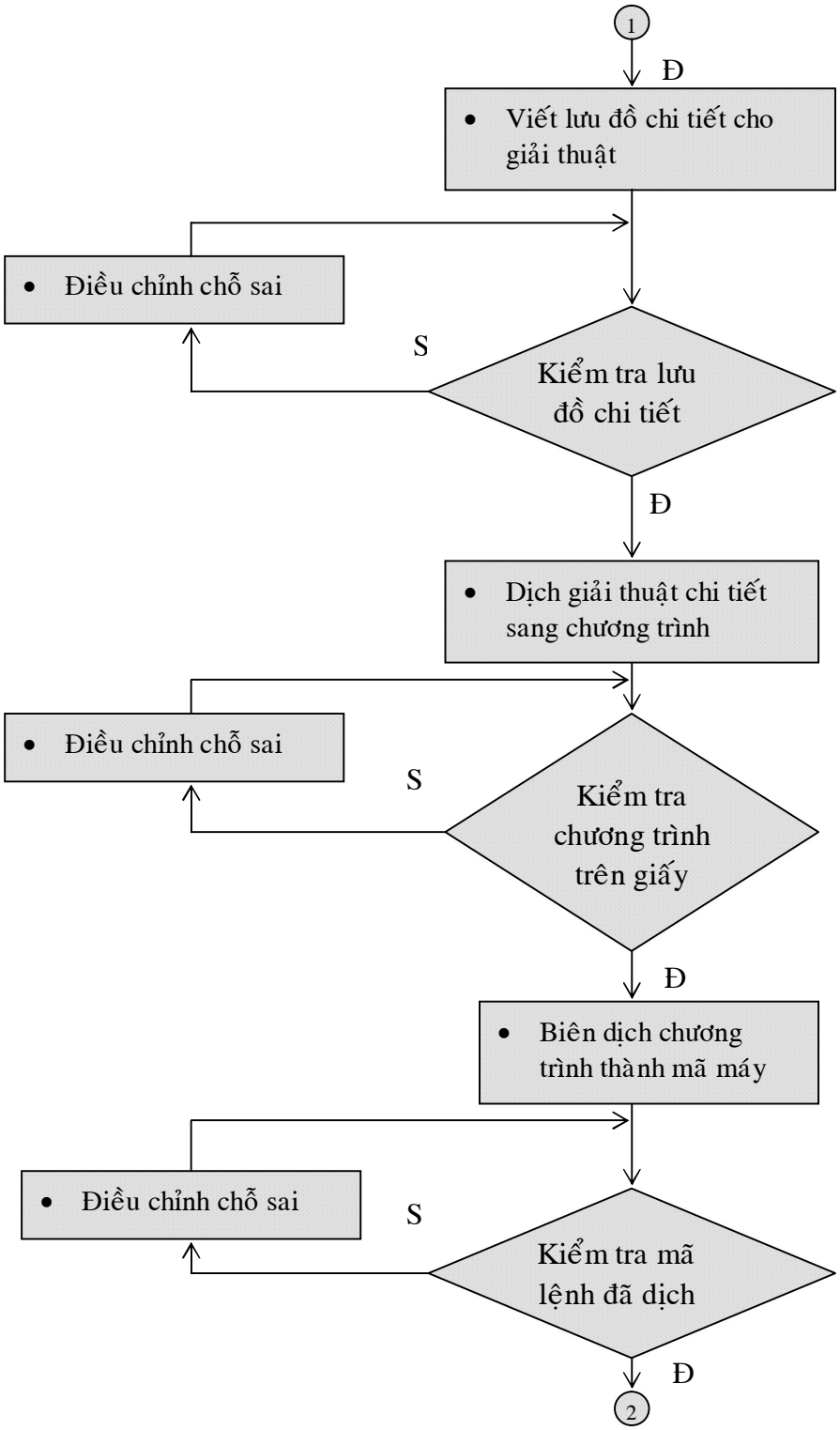
----------oOo----------

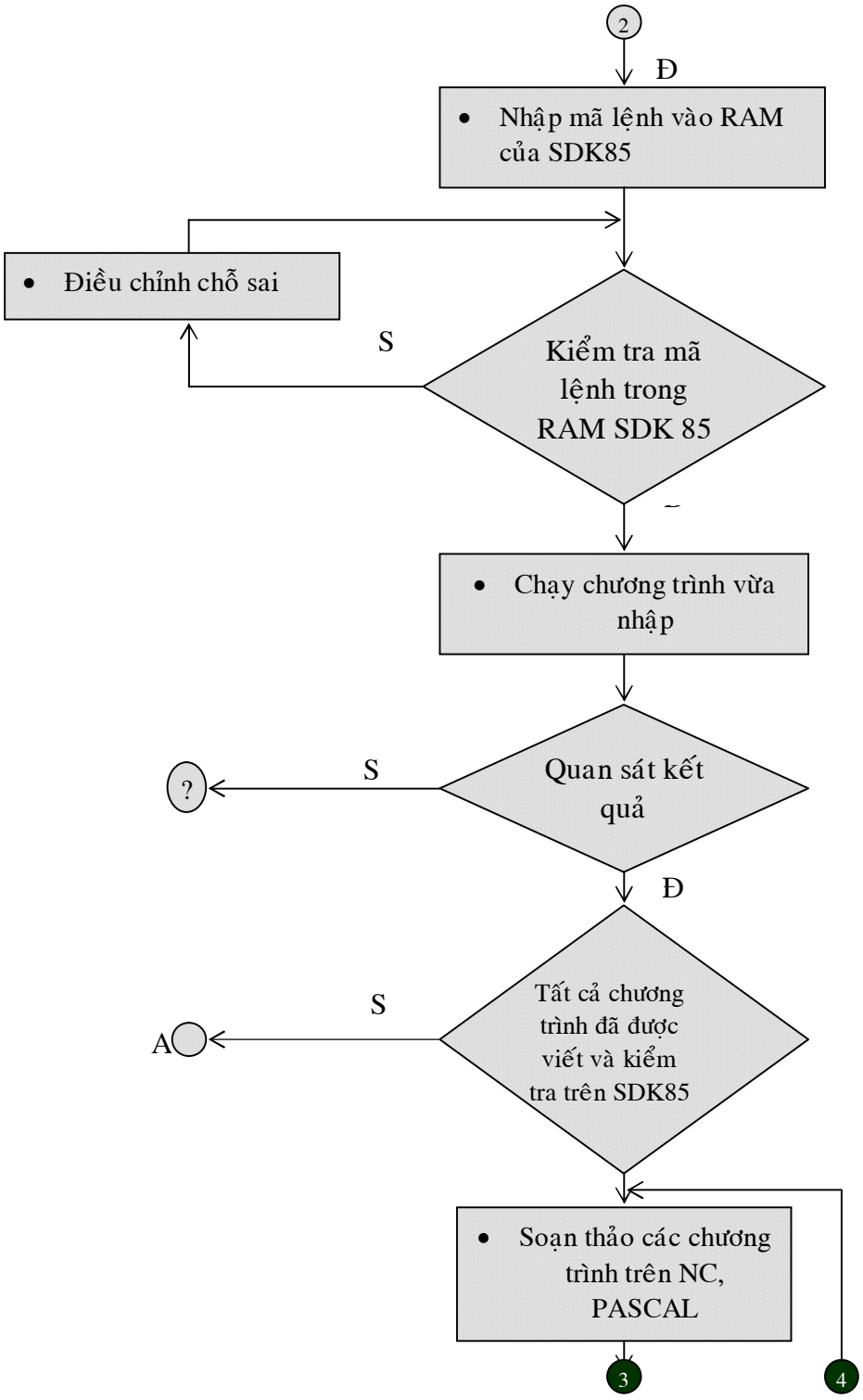
Thi công một hệ thống vi xử lí là một quá trình phức tạp. Bao gồm hai giai đoạn :

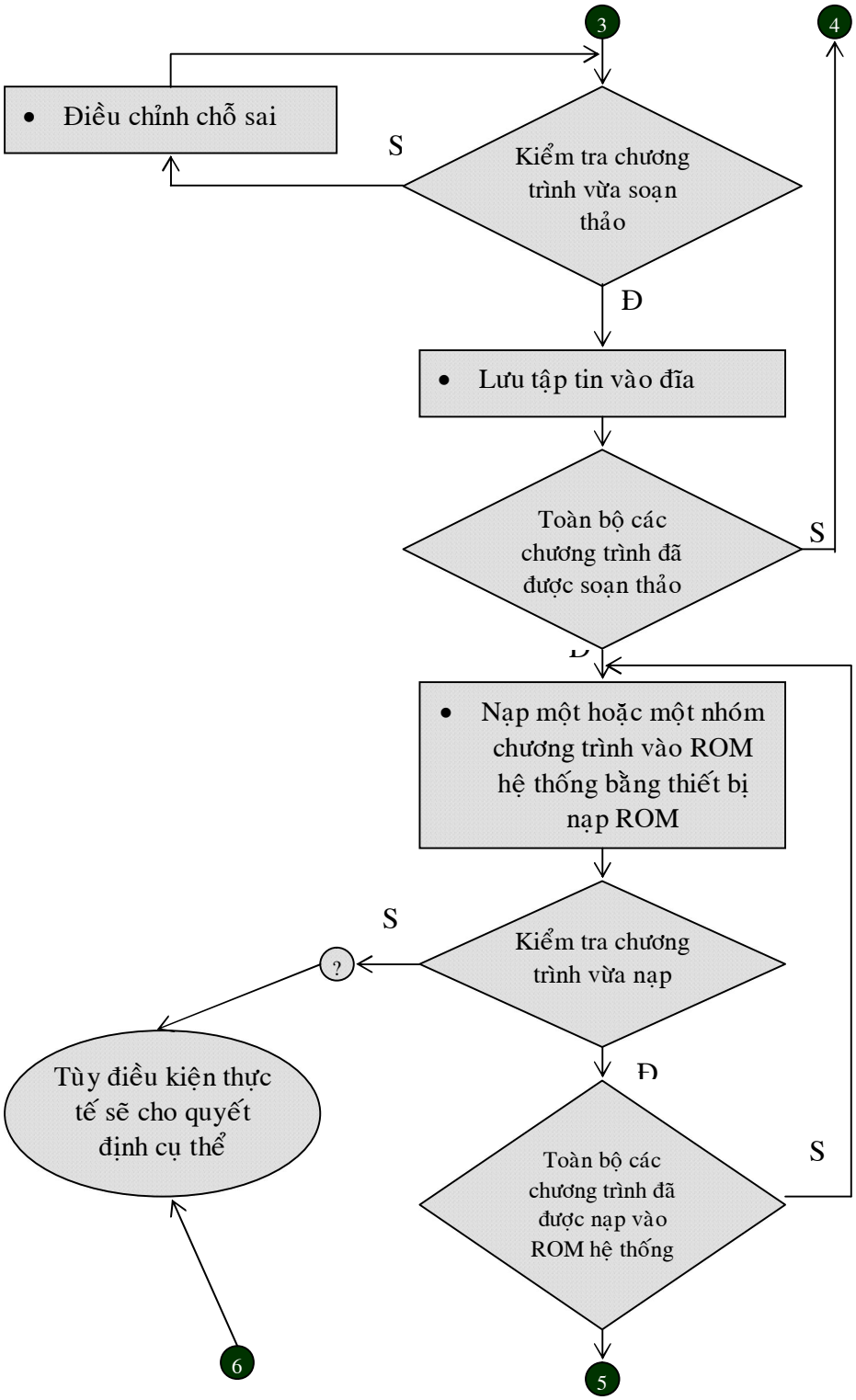
- Thi công phần cứng
- Thi công phần mềm

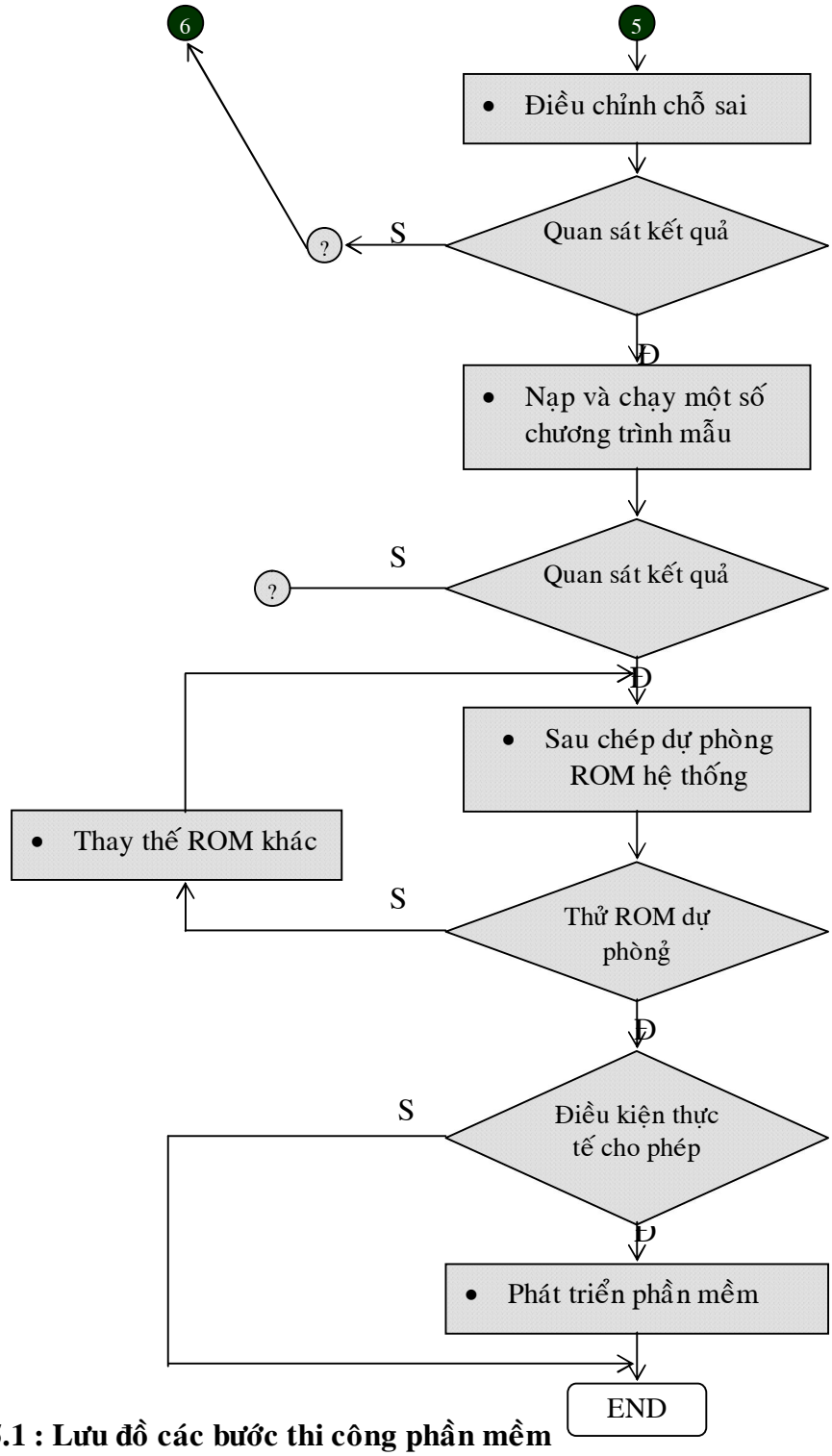
Hai giai đoạn này có thể tiến hành độc lập, không cần thiết phải theo trình tự, nhưng phải thống nhất. Sau đây là quá rình thi công phần mềm.


SOFTWARE
• Xác định cấu hình phần cứng
A
• Xác định yêu cầu chương trình
• Chọn lựa giải thuật
S
Giải thuật tối ưu
Đ
• Viết lưu đồ tổng quát cho giải thuật
• Điều chỉnh chỗ sai
S
Kiểm tra lưu đồ tổng quát
Đ
1


1
Đ
• Viết lưu đồ chi tiết cho giải thuật
• Điều chỉnh chỗ sai
S
Kiểm tra lưu đồ chi tiết
Đ
• Dịch giải thuật chi tiết sang chương trình
• Điều chỉnh chỗ sai
S
Kiểm tra chương trình trên giấy
Đ
• Biên dịch chương trình thành mã máy
• Điều chỉnh chỗ sai
S
Kiểm tra mã lệnh đã dịch
Đ
2


2
Đ
• Nhập mã lệnh vào RAM của SDK85
• Điều chỉnh chỗ sai
S
Kiểm tra mã lệnh trong RAM SDK 85
• Chạy chương trình vừa nhập
?
S
Quan sát kết quả
Đ
A
S
Tất cả chương trình đã được viết và kiểm tra trên SDK85
• Soạn thảo các chương trình trên NC, PASCAL
3
4


3
4
• Điều chỉnh chỗ sai
S
Kiểm tra chương trình vừa soạn thảo
Đ
• Lưu tập tin vào đĩa
Toàn bộ các chương trình đã được soạn thảo
S
Đ
• Nạp một hoặc một nhóm chương trình vào ROM hệ thống bằng thiết bị nạp ROM
S
Kiểm tra chương trình vừa nạp
?
Tùy điều kiện thực tế sẽ cho quyết định cụ thể
Đ
Toàn bộ các chương trình đã được nạp vào ROM hệ thống
S
6
5



Hình 5.1 : Lưu đồ các bước thi công phần mềm

Bảng 5.1 BẢNG TRA CỨU CÁC HƯƠNG TRÌNH PHỤC VỤ MONITOR

Tên chương trình (1)	Địa chỉ bắt đầu (2)	Địa chỉ cuối (3)	Dung lượng (4)
MNT	0000H	00FCH	252 byte
DAUCHAM	0200H	023AH	58 byte
DELAY 0.1	0250H	027AH	42 byte
DEMP	0290H	02A5H	21 byte
DPFIND	02C0H	030EH	78 byte
DP HL	0320H	0351H	49 byte
DP HLM	0360H	037DH	29 byte
DP M	0390H	03B3H	35 byte
DP PC	03C0H	03F5H	53 byte
DP REG	0400H	0422H	34 byte
CODE 1	0430H	0437H	7 byte
CODE 2	0440H	0467H	23 byte
HELLO	0480H	04EFH	111 byte
KYTUB	0500H	053EH	62 byte
ORMP	0550H	055FH	15 byte
XADD	0570H	059BH	43 byte
KYBRD	05A0H	05AFH	15 byte
KYCHECK	05C0H	05D9H	25 byte
KYCODE	05F0H	060BH	27 byte
PHIMSO	0620H	06C3H	163 byte
UP	06E0H	06F8H	24 byte
DOWN	0700H	070CH	12 byte
ADD	0720H	072EH	14 byte
SUBBCHL	0740H	074DH	13 byte
COPY C.0	0760H	077EH	30 byte
COPY C.1	0790H	07BBH	43 byte
COPY	07D0H	08CDH	253 byte
DEL	08E0H	0975H	149 byte
REG	09A0H	0AA0H	256 byte
SRCH	0AD0H	0B23H	83 byte
STEP	0B50H	0D23H	467 byte
LMB	0D50H	0E78H	296 byte
SHM	0E90H	0EAFH	31 byte
INS	0EC0H	0F9FH	223 byte
SDC	0FC0H	1088H	200 byte
ADDBCDE	10C0H	10CBH	11 byte
PC	10F0H	1123H	51 byte

Chương 6 : HƯỚNG DẪN SỬ DỤNG

CHƯƠNG 6 : HƯỚNG DẪN SỬ DỤNG

6.1 PHÍM ADD

Phím ADD dùng để thay đổi địa chỉ hiện hành. Muốn thực hiện ta thực hiện các bước sau :

* Ấn các phím số sao cho trên màn hình hiện thị các địa chỉ cần nhảy tới.

*Ấn phím ADD lập tức địa chỉ mới và dữ liệu tại địa chỉ đó được hiển thị.

6.2 PHÍM UP ; DOWN

Phím UP dùng để tăng địa chỉ hiện hành lên một và để xác nhận dữ liệu mới.

Ví dụ : Để nhập dữ liệu D3 vào ô nhớ có địa chỉ 7000 ta thực hiện như sau :

- Ấn cá phím số, trên màn hình xuất hiện 7000
- Ấn phím ADD, địa chỉ hiện hành được hiển thị
- Ấn các phím số, trên màn hình hiển thị D3
- Ấn phím UP : Dữ liệu D3 được nạp vào ô nhớ có địa chỉ 7000 và địa chỉ hiện hành được tăng lên 1 đơn vị.

Muốn giảm địa chỉ hiện hành thì nhấn phím DOWN. Phím DOWN không nhập dữ liệu mới.

6.3 PHÍM PC VÀ GO

Sau khi nạp chương trình vào RAM, để chạy thử ta nhấn PC. Trên màn hình xuất hiện thông báo "PC = • • • • " ;

- Ta nhấn phím số để nạp địa chỉ đầu của đoạn chương trình cần thử.
- Nhấn phím GO : Lập tức vi xử lí thực hiện chương trình đó. khi không nạp địa chỉ mới mà nhấn GO thì sẽ thoát chương trình PC.

6.4 PHÍM REG

Phím REG là phím chức năng thực hiện việc xem và nạp giá trị mới vào các thanh ghi.

Ứng dụng :

- Thiết lập giá trị cho các thanh ghi làm biến số trước khi chạy một chương trình nào đó. Chú ý : không nạp giá trị cho cặp thanh ghi HL trong trường hợp này (chương trình phím PC làm thay đổi HL).
- Thiết lập, xóa các cờ trước khi chạy chương trình
- Lưu trữ kết quả của chương trình vào các thanh ghi, để kiểm tra sau khi chạy xong chương trình
- Kiểm tra các cờ sau khi thực hiện các lệnh số học và logic
- Với chương trình Monitor sử dụng phím REG để kiểm tra kết quả của một lệnh trong thanh ghi, khi dùng phím STEP.

Cách sử dụng :

- Nhấn phím REG : Trên màn hình hiển thị thanh ghi A và nội dung trong thanh ghi A

VD :

- Nhấn phím UP, DOWN để xem nội dung của các thanh ghi khác..
- Khi muốn nạp nội dung mới vào thanh ghi.

VD : nạp vào C nội dung là A D

Ta thực hiện các bước sau :

+ Nhấn UP, DOWN để trên màn hình hiển thị thanh ghi và nội dung thanh ghi cần nạp.

+ Nhấn phím A, D (phím số), trên màn hình hiển thị A D

+ Nhấn phím UP : tức thì nội dung mới được nạp.

- Muốn thoát chương trình REG ta nhấn phím REG một lần nữa.

6.5 PHÍM COPY

Khi muốn chép một đoạn dữ liệu từ vùng này qua vùng khác ta nhấn phím COPY. Trên màn hình hiển thị thông báo nạp địa chỉ bắt đầu vùng gốc.

- Dùng phím số để nạp địa chỉ đó.
- Nhấn phím ADD xác định địa chỉ đó, trên màn hình xuất hiện thông báo nạp địa chỉ cuối vùng gốc.

- Dùng phím số để nạp địa chỉ đó.
- Nhấn phím ADD xác định địa chỉ đó, trên màn hình xuất hiện thông báo nạp địa chỉ đầu của vùng đích.

- Dùng phím số để nạp địa chỉ đó.
- Nhấn phím ADD xác định địa chỉ nới nạp, chương trình thực hiện việc COPY.

Nếu một trong các địa chỉ trên không được nạp mà ta nhấn phím ADD thì tự động thoát khỏi chương trình COPY.

6.6 PHÍM DEL

Khi muốn xóa (chép 00H) một đoạn dữ liệu ta nhấn phím DEL. Màn hình thông báo nạp địa chỉ đầu vùng cần xóa.

- Dùng phím số nạp địa chỉ đó.
- Nhấn phím ADD xác định. Màn hình xuất hiện thông báo nạp địa chỉ kết thúc.

- Dùng phím số nạp địa chỉ đó.
- Nhấn phím ADD xác định. Chương trình thực hiện chức năng DEL rồi thoát khỏi chương trình phím DEL.

Nếu một trong các địa chỉ trên không được nạp mà nhấn ADD thì sẽ thoát khỏi chức năng DEL.

6.7 PHÍM SRCH

Khi muốn tìm kiếm một byte dữ liệu nào đó ta nhấn phím SRCH. Màn hình hiển thị :

- Dùng phím số nạp giá trị byte cần tìm.
- Nhấn phím UP chương trình sẽ tìm kiếm từ địa chỉ hiện hành tới khi nào tìm thấy byte có giá trị đã nạp dừng và hiển thị. Nếu muốn tìm tiếp ta nhấn phím SRCH thì chương trình sẽ tìm tiếp (không cần nạp giá trị mới).

6.8 PHÍM STEP

- Phím STEP có chức năng gỡ rối chương trình khi thực hiện có kết quả sai mà chưa biết được lệnh sai.
- Sử dụng STEP có thể thực hiện và kiểm tra kết quả từng lệnh trong 1 chương trình.

Cách sử dụng :

Đầu tiên, ta phải dời con trỏ tới địa chỉ cần thực hiện. Sau đó ấn phím STEP chương trình tại đó sẽ thực hiện từng lệnh.

6.9 PHÍM INS

Phím INS sử dụng để chèn thêm một byte hoặc một số byte, có thể sửa địa chỉ.

Cách sử dụng :

- Di chuyển con trỏ tới địa chỉ cần chèn.
- Ấn phím Ins. Màn hình hiển thị thông báo nạp địa chỉ cuối vùng INS.

- Dùng phím số nạp địa chỉ cuối.
- Ấn phím ADD xác định địa chỉ đó
- Sử dụng phím số, phím UP để chèn dữ liệu mới.
- Phím DOWN chỉ cho phép giảm địa chỉ xuống 1 và không lùi liên tiếp. Nó có chức năng chỉnh lại dữ liệu vừa chèn vào.
- Nếu muốn sửa địa chỉ, ấn phím ADD. Màn hình thông báo nạp địa chỉ đầu.

- Ấn phím ADD một lần nữa chức năng sửa địa chỉ được thực hiện và thoát chương trình.
- Nếu muốn thoát chương trình không qua chức năng sửa địa chỉ, ấn phím INS một lần nữa.

Chương 7 :
TÓM TẮT
KẾT LUẬN
ĐỀ NGHỊ

CHƯƠNG 7 : TÓM TẮT – KẾT LUẬN – ĐỀ NGHỊ

7.1 TÓM TẮT ĐỀ TÀI

Đề tài được trình bày thành 7 chương nhưng quan trọng nhất là chương 4, nó trình bày từ tổng quát đến cụ thể chương trình Monitor và các chương trinh con phục vụ chương trình Monitor. Chương trình Monitor quản lí toàn bộ hệ thống và cho phép người sử dụng soạn thảo và thử nghiệm các chương trình một cách tiện nghi nhất.

Chương 5 trình bày những thông tin và phương pháp thi công hoàn chỉnh một phần mềm.

Chương 6 là phần hướng dẫn sử dụng thiết bị vừa thiết kế. Đây là, phần quan trọng nhằm giúp người sử dụng có thể khai thác hết những chức năng, ưu điểm của hệ thống.

7.2 KẾT LUẬN

Qua 7 tuần làm việc không mệt mỏi, không những biết tổng hợp và ứng dụng những kiến thức đã học vào thực tế mà còn rút ra dược những kết luận sau:

∴ Có thể dùng vi xử lí 8085 để thiết kế thiết bị thực tập vi xử lí, khả năng làm việc của thiết bị phụ thuộc chủ yếu vào phần mềm.

∴ Phần mềm và phần cứng có mối liên quan chặt chẽ, chúng có thể thay thế cho nhau ở một số chức năng nào đó .

∴ Đối với chức năng COPY chương trình phải nhận biết được hai vùng gốc và đích có trùng nhau không và trùng nhau như thế nào để khi thực hiện COPY không bị mất dữ liệu .

∴ Để tiện cho việc chạy thử chương trình và kiểm tra kết quả sau khi chạy thử chương trình thì chức năng REG phải nạp và xem được nội dung thanh ghi.

∴ Nếu chương trình phím PC sử dụng lệnh nạp địa chỉ trong cặp thanh ghi HL vào cặp thanh ghi PC thì thông số trong cặp thanh ghi HL đã nạp trước đó ở chức năng REG sẽ vô nghĩa.

∴ Chức năng STEP, chương trình người sử dụng và chương trình Monitor chạy chung với nhau, phải thiết kế sao cho chương trinh phím STEP không sử dụng stackpoiter, để ghành stackpoiter cho chương trình người sử dụng. Phải bảo toàn các thông số của chương trình người sau mỗi bước để thực hiện bước kế tiếp, nhất là đối với các lệnh nhảy, gọi, trở về có điều kiện. Đối với các lệnh gọi và trở về ta phải tạo ra một con trỏ MNT- STACKPOITER, là vùng nhớ theo kiểu LIFO, giống stackpoiter, để lưu giữ địa chỉ khi gặp lệnh nhảy và lấy lại địa chỉ đó khi gặp lệnh trở về, đặc biệt khi các chương trình con lồng vào nhau. Đối với các lệnh nhảy, lệnh gọi, lệnh trở về ta phải khống chế sao cho chương trình chay từng bước.

∴ Chức năng INS rất cần thiết khi chỉnh sửa chương trình như chèn thêm các lệnh thiếu, phải sửa được dịa chỉ của các lệnh nhảy, lệnh gọi, lệnh lưu trữ nếu địa chỉ của chúng chỉ đến vùng INS tác động.

7.3 ĐỀ NGHỊ

Mặc dù đã cố gắng hết sức mình, yếu tố thời gian đã bắt buộc người thực hiện phải dừng công việc nghiên cứu. Thực sự, người thực hiện chưa hài lòng lắm với những gì giải quyết được, hy vọng sau này có thời gian nghiên cứu thêm và những khoá sau sẽ cố gắng hoàn thiện đề tài, để nó trở thành một sản phẩm hoàn chỉnh.

Sau đây là một vài đề nghị tham khảo dành cho những ai có ý định phát triển đề tài:

∴ Xây dựng thêm chức năng chạy từng đoạn. Các đoạn đó được xác định bằng các điểm dừng được người sử dụng đặt.

∴ Xây dựng thêm chức năng sửa địa chỉ để sử dụng sau khi thực hiện chức năng COPY.

∴ Xây chương trình cho phép đổ dữ liệu từ máy tính vào thiết bị để chạy thử.

∴ Xây dựng các bài thực hành trên thiết bị vừa thiết kế.

∴ Xây dựng những chương trình tiện ích, thiết kế các modul phần cứng để ghép nối với thiết bị phục vụ cho các bài thực hành.

PHẦN C
TÀI LIỆU THAM KHẢO
PHỤ LỤC

TAØI LIEÄU THAM KHAÛO.

---Ω---

1. CHÂU KIM LANG
   "Phương pháp nghiên cứu khoa học"
   Ban Sư Phạm Kĩ Thuật trường Đại học Sư Phạm Kĩ Thuật Tp. HCM, 1989.

2. TRẦN VĂN TRỌNG
   "Kĩ thuật vi xử lí 8085A"
   Đại học Sư Phạm Kĩ Thuật Tp. HCM, 1995.

3. PAUL BATES, PENG
   "Truyền dữ liệu sử dụng vi mạch LSI" TỐNG VĂN ON (dịch).
   Đại học Bách Khoa Tp. HCM, 1994. Trang 42 – 76.

4. RONALD J. TOCCI
   "Digital Systems : Principles and Application"
   4/e Prentice Hall International Inc, 1998

PHỤ LỤC

Phần này trình bày những chương trình con đơn giản, không trình bày giải thuật. Các chương trình này được viết từ ban đầu khi còn chưa có kinh nghiệm, vì thế chúng còn mang tính "thủ công'. Các chương trình được sắp xếp theo mẫu tự A, B, C… để tiện việc tra cứu.

DAUCHAM

DAUCHAM : Đây là chương trình con có nhiệm vụ xuất các điểm sáng lan dần.

Input : Không

Output : Hiển thị

- Có gọi Delay 0.1
- Chương trình không thay đổi nội dung thanh ghi

DAUCHAM

```
PUSH    PSW
MVI     A, 84H
STA     Add_CT79
MVI     A, 80H
STA     Add_DT79
MVI     A, 02H
CALL    DELAY 0.1
MVI     A, 85H
STA     Add_CT79
MVI     A, 80H
STA     Add_DT79
MVI     A, 02H
CALL    DELAY 0.1
MVI     A, 86H
STA     Add_CT79
MVI     A, 80H
STA     Add_DT79
MVI     A, 02H
CALL    DELAY 0.1
MVI     A, 87H
STA     Add_CT79
MVI     A, 80H
STA     Add_DT79
POP     PSW
RET
```

DELAY 0.1

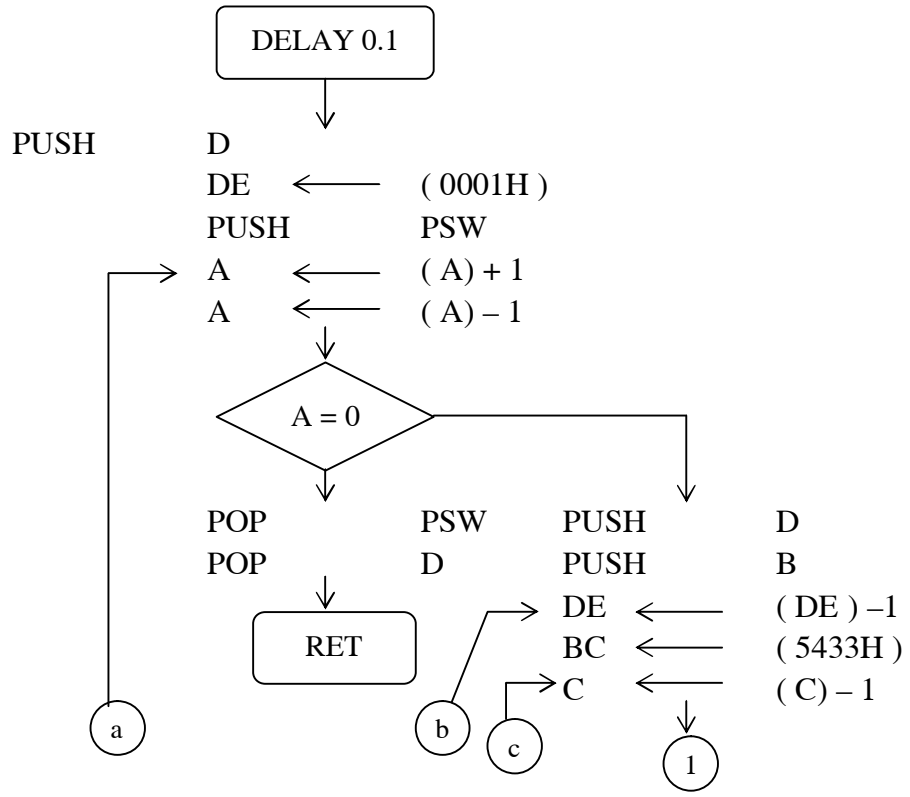
DELAY 0.1 : Đây là chương trình con có tác vụ trì hoãn; 0.1 giây (0….25,5 giây).

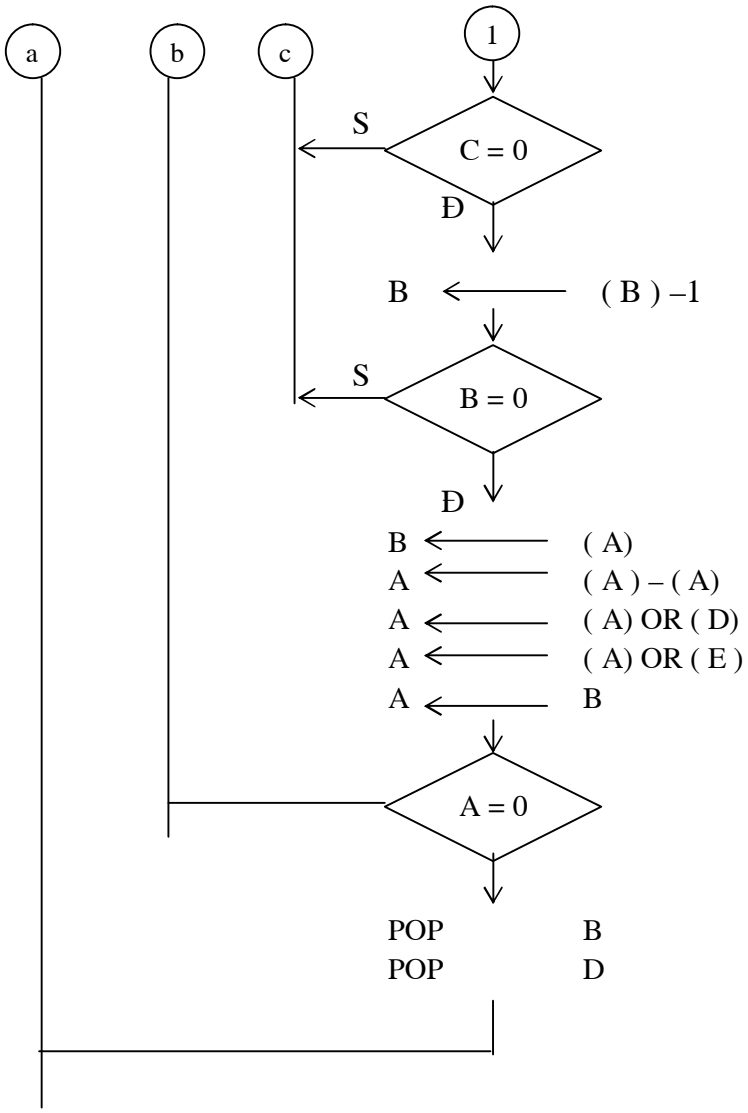
Input :Nạp thời gian cần trì hoãn vào thanh ghi A.

Output : Trì hoãn chương trình trong thời gian đã định.

DELAY 0.1

```
      PUSH    D
      LXI     D, 0001H
      PUSH    PSW
      INR     A
      JMP     *4
*1:   PUSH    D
      PUSH    B
*3 :  DCX     D
      LXI     B, 5433H
*2 :  DCR     C
      JNZ     *2
      DCR     B
      JNZ     *2
      MOV     B, A
      SUB     A
      ORA     D
      ORA     E
      MOV     A, B
      JNZ     3*
      POP     B
      POP     D
*4 :  DCR     A
      JNZ     *1
      POP     PSW
      POP     D
      RET
```




a
b
c
1
S
C = 0
Đ
B ← (B) –1
S
B = 0
Đ
B ← (A)
A ← (A) – (A)
A ← (A) OR (D)
A ← (A) OR (E)
A ← B
A = 0
POP B
POP D

DEMP

DEMP : Đây là chương trình con có nhiệm vụ lấy nội dung trong bốn ô mã phím.

Input : Lấy nội dung trong các ô mã phím.
(Add_{qp+0}, Add_{qp+1}, Add_{qp+2}, Add_{qp+3})

Output : Kết quả được lưu trữ vào cặp thanh ghi DE
- Có gọi ORMP
- Ngoài DE chương trình không thay đổi nội dung các thanh ghi.

DEMP

PUSH	PSW	LDA	Add_{QP+1}
PUSH	B	ORA	B
LDA	Add_{QP+0}	MOV	D, A
RLC		CALL	ORMP
RLC		MOV	E, A
RLC		POP	B
RLC		POP	PSW
MOV	B, A	RET	

DPFIND

DPFIND : Đây là chương trình con có nhiệm vụ hiển thị "FIND"

Input : Không

Output : Hiển thị
- Không thay đổi nội dung các thanh ghi
- Có gọi DELAY 0.1

DPFIND

PUSH	PSW
MVI	A, C3
STA	Add_{CT79}
MVI	A,01H
CALL	DELAY 0.1
MVI	A, 80H
STA	Add_{CT79}
MVI	A, 71
STA	Add_{DT79}
MVI	A, 81H
STA	Add_{CT79}
MVI	A, 10H
STA	Add_{DT79}
MVI	A, 82H
STA	Add_{CT79}
MVI	A, 54H
STA	ADD_{DT79}
MVI	A, 83H
STA	Add_{CT79}
MVI	A, 5EH
STA	Add_{DT79}
MVI	A, 86H
STA	Add_{CT79}
MVI	A, 80H
STA	Add_{DT79}
MVI	A, 02H
CALL	DELAY 0.1
MVI	A, 87H
STA	Add_{CT79}
MVI	A, 80H
STA	Add_{DT79}
POP	PSW
RET	

DPHL

DPHL :	Đây là chương trình con có nhiệm vụ hiển thị nội dung thanh ghi HL ra led 7 đoạn.
Input :	Nội dung thanh ghi HL.
Output :	Hiển thị • Không thay đổi nội dung các thanh ghi • Có gọi CODE2

DPHL

```
PUSH    PSW
PUSH    D
MOV     A, H
CALL    CODE2
MVI     A, 80H
STA     Add_CT79
MOV     A, D
STA     Add_DT79
MVI     A, 81H
STA     Add_CT79
MOV     A, E
STA     Add_DT79
MOV     A,L
CALL    CODE2
MVI     A, 82H
STA     Add_CT79
MOV     A, D
STA     Add_DT79
MVI     A, 83H
STA     Add_CT79
MOV     A,E
STA     Add_DT79
POP     D
POP     PSW
RET
```

DPHLM

DPHLM : Đây là chương trình con có nhiệm vụ hiển thị địa chỉ và dữ liệu hiện tại địa chỉ đó.

Input : HL chứa địa chỉ cần hiển thị.

Ouput : Hiển thị

- Không thay đổi nội dung các thanh ghi
- Có gọi DPHL và DPM.

DPHLM

```
PUSH    PSW
CALL    DPHL
MVI     A, 84H
STA     Add_CT79
MVI     A, 00H
STA     Add_DT79
MVI     A, 85H
STA     Add_CT79
MVI     A, 00H
STA     Add_DT79
CALL    DPM
POP     PSW
RET
```

DPM

DPM : Là chương trình con hiển thị thông báo nạp địa chỉ vào thanh ghi PC.
Input : Không
Output : Hiển thị
- Không ảnh hưởng các thanh ghi
- Có gọi DELAY 0.1

DPM

PUSH	PSW	MVI	A, 87H
PUSH	B	STA	Add_{CT79}
PUSH	D	MOV	A, E
MOV	A, M	STA	Add_{DT79}
CALL	CODE2	STA	Add_{HT+7}
MVI	A, 86H	POP	D
STA	Add_{CT79}	POP	B
MOV	A, D	POP	PSW
STA	Add_{DT79}	RET	
STA	Add_{HT+6}		

DP PC

DP PC : Là chương trình con hiển thị thông báo nạp địa chỉ vào thanh ghi PC.
Input : Không
Output : Hiển thị
- Không ảnh hưởng tới các thanh ghi
- Có gọi DELAY 0.1

DP PC

PUSH	PSW	MVI	A, 39H
MVI	A, C3H	STA	Add_{DT79}
STA	Add_{CT79}	MVI	A, 82H
MVI	A, 01H	STA	Add_{CT79}
CALL	DELAY 0.1	MVI	A, 48H
MVI	A, 80H	STA	Add_{DT79}
STA	Add_{CT79}	MVI	A, 83H
MVI	A,63H	STA	Add_{CT79}
STA	Add_{DT79}	MVI	A, 00H
MVI	A, 81H	STA	Add_{DT79}
STA	Add_{CT79}	POP	PSW
		RET	

DP REG

DP REG : Đây là chương trình con có nhiệm vụ hiển thị nội dung thanh ghi A.
Input : Nạp thanh ghi cần hiển thị vào A.
Output : Hiển thị
- Có gọi CODE2

DP REG

PUSH	PSW	MVI	A, 87H
PUSH	B	STA	Add_{CT79}
PUSH	D	MOV	A, E
CALL	CODE2	STA	Add_{DT79}
MVI	A, 86H	STA	Add_{HT+7}
STA	Add_{CT79}	POP	D
MOV	A, D	POP	B
STA	Add_{DT79}	POP	PSW
STA	Add_{HT+6}	RET	

CODE 1

CODE 1 : Là chương trình con đổi các số Hexa từ 00H → 0FH ra một kí tự 7 đoạn bằng cách tra bảng mã.

Input : Nạp mã cần đổi vào thanh ghi A

Output : Kết quả được ghi vào thanh ghi A.

- Không thay đổi nội dung các thanh ghi ngoài thanh ghi A

CODE 1

```
PUSH    H
MOV     L, A
MVI     H,      ; nạp địa chỉ cao bảng mã
MOV     A, M
POP     H
RET
```

CODE 2

CODE 2: Đây là chương trình con đổi mã hai kí tự Hexa ra hai kí tự led 7 đoạn.

Input : Nạp mã cần chuyển đổi vào A.

Output : Ký tự hiển thị cho 4 bit cao được lưu vào thanh ghi D
Ký tự hiển thị cho 4 bit thấp lưu vào thanh ghi E

- Có gọi CODE 1

CODE 2

```
PUSH    PSW          |  MOV    D, A
PUSH    B            |  MOV    A, B
MOV     B, A         |  ANI    0FH
RRC                  |  CALL   CODE 1
RRC                  |  MOV    E, A
RRC                  |  POP    B
RRC                  |  POP    PSW
ANI     0FH          |  RET
CALL    CODE 1       |
```

HELLO

HELLO :	Là chương trình con hiển thị chữ "HELLO", sau đó, hiển thị con trỏ.
Input :	Không
Output :	Hiển thị
	• Chương trình không thay đổi nội dung các thanh ghi

HELLO

PUSH	B
PUSH	PSW
MVI	B, 00H
MVI	A, C3H
STA	Add_{CT79}
MVI	A, 01H
CALL	DELAY 0.1
MVI	A,98H
STA	Add_{CT79}
MVI	A, 76H
STA	Add_{DT79}
MVI	A, 02H
CALL	DELAY 0.1
MVI	A, 79H
STA	Add_{DT79}
MVI	A. 02H
CALL	DELAY 0.1
MVI	A, 38H
STA	Add_{DT79}
MVI	A, 02H
CALL	DELAY 0.1
MVI	A, 38H
STA	Add_{DT79}
MVI	A, 02H
CALL	DELAY 0.1
MVI	A, 3F
STA	Add_{DT79}
MVI	A, 02H
CALL	DELAY 0.1
MVI	A, 00H
STA	Add_{DT79}
INR	B
MOV	A, B
CPI	08H
JZ	*2
JMP	*1
MVI	A, 91H
STA	Add_{CT79}
MVI	A, 3FH
STA	Add_{DT79}
STA	Add_{DT79}
STA	Add_{DT79}
STA	Add_{DT79}
MVI	A, 80H
STA	Add_{CT79}
POP	PSW
POP	B
RET	

KYTUB

KYTUB :	Là một chương trình con hiển thị kí tự và dấu bằng.
Input :	Nạp mã 7 đoạn của kí tự cần hiển thị vào thanh ghi B.
Output :	Hiển thị kí tự và dấu bằng.
	Không làm thay đổi nội dung các thanh ghi.

KYTUB

```
PUSH    PSW
MVI     A, 80H
STA     Add_CT79
MOV     A, B
STA     Add_DT79
MVI     A, 81H
STA     Add_CT79
MVI     A, 48H
STA     Add_DT79
MVI     A, 82H
STA     Add_CT79
MVI     A, 00H
STA     Add_DT79
MVI     A, 83H
STA     Add_CT79
MVI     A, 00H
STA     Add_DT79
MVI     A, 84H
STA     Add_CT79
MVI     A, 00H
STA     Add_DT79
MVI     A, 85H
STA     Add_CT79
MVI     A, 00H
STA     Add_DT79
POP     PSW
RET
```

ORMP

ORMP : Đây là chương trình con hợp hai byte mã phím thành một byte.
Input : Lấy dữ liệu trong Add_{qp+2} và Add_{qp+3}
Output : Kết quả được trữ trong thanh ghi A.
Chương trình không thay đổi các thanh ghi khác.

ORMP

PUSH	B	MOV	B, A
LDA	Add_{qp+2}	LDA	Add_{qp+3}
RLC		ORA	B
RLC		POP	B
RLC		RET	
RLC			

XADD

XADD : Đây là chương trình con hiển thị "ADD" ra màn hình.
Input : Không
Output : Hiển thị
- Chương trình không thay đổi nội dung các thanh ghi.
- Có gọi chương trình DELAY 0.1

XADD

PUSH	PSW	STA	Add_{CT79}
MVI	A, C3H	MVI	A, 5EH
STA	Add_{CT79}	STA	Add_{DT79}
MVI	A, 01H	MVI	A, 82H
CALL	DELAY 0.1	STA	Add_{CT79}
MVI	A, 80H	MVI	A, 5EH
STA	Add_{CT79}	STA	Add_{DT79}
MVI	A, 77H	POP	PSW
STA	Add_{DT79}	RET	
MVI	A, 81H		