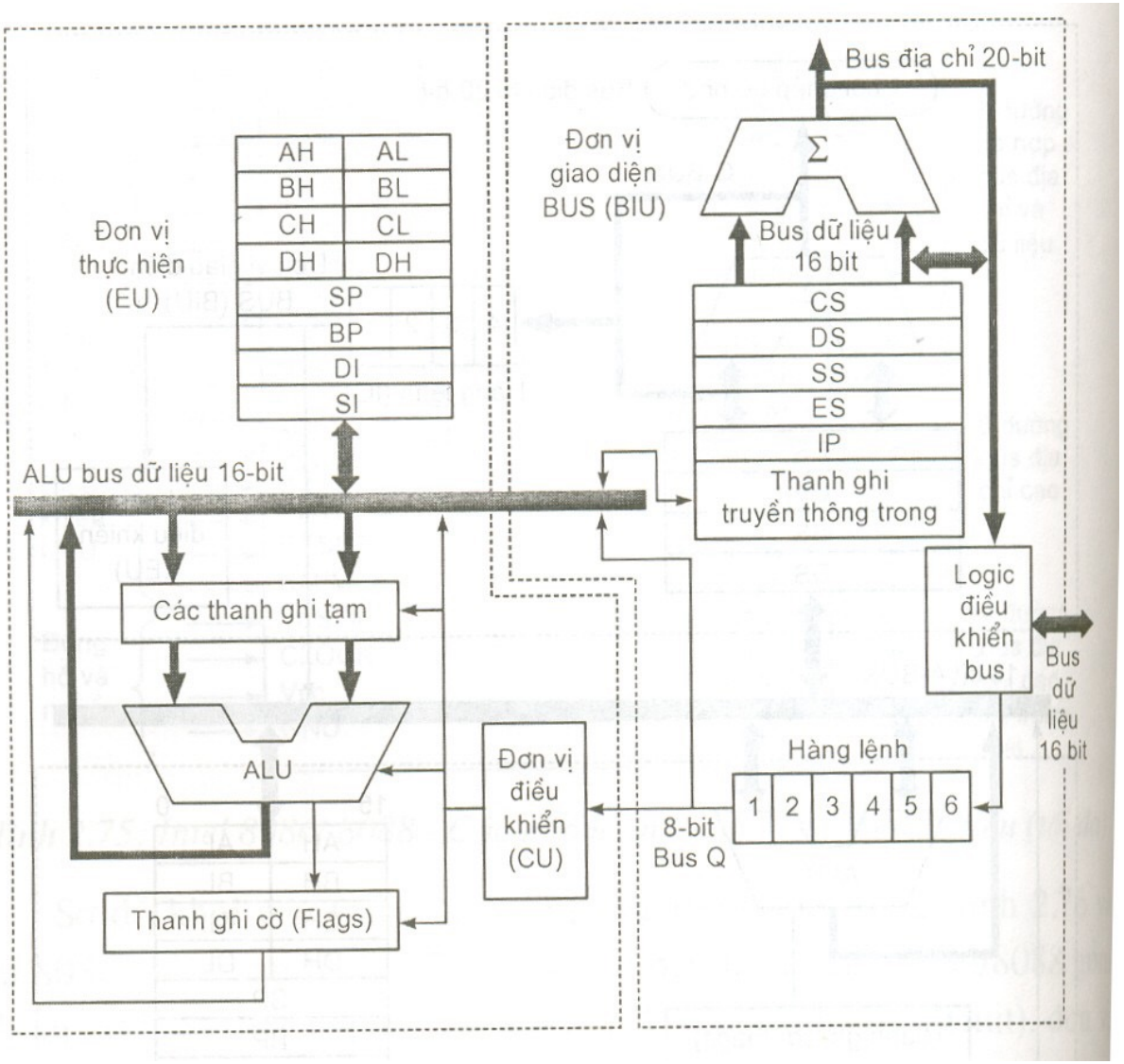
LẬP TRÌNH HỢP NGỮ VỚI 8086/8088

NỘI DUNG CHÍNH

1. Giới thiệu về hợp ngữ
2. Cú pháp của chương trình hợp ngữ
3. Dữ liệu cho chương trình hợp ngữ
4. Biến và hằng
5. Khung chương trình hợp ngữ
6. Các cấu trúc điều khiển
7. Giới thiệu phần mềm mô phỏng emu8086
8. Một số ví dụ
9. Chương trình con
10. Marco
11. Giới thiệu thiết bị ảo – Đèn giao thông

Sơ đồ khối vi xử lý 8086/8088



3.1. Giới thiệu về hợp ngữ

- Hợp ngữ (Assembler) là ngôn ngữ lập trình bậc thấp, chỉ cao hơn ngôn ngữ máy;
- Hợp ngữ là ngôn ngữ gắn liền với các dòng vi xử lý (processor specific).
  - Các lệnh dùng trong hợp ngữ là lệnh của VXL
  - Chương trình hợp ngữ viết cho một VXL có thể không hoạt động trên VXL khác.
- Chương trình hợp ngữ khi dịch ra mã máy có kích thước nhỏ gọn, chiếm ít không gian nhớ.
- Hợp ngữ thường được sử dụng để viết:
  - Các trình điều khiển thiết bị
  - Các môđun chương trình cho vi điều khiển
  - Một số môđun trong nhân HĐH (đòi hỏi kích thước nhỏ gọn và tốc độ cao)

3.2. Cú pháp của chương trình hợp ngữ

- Trong chương trình hợp ngữ, mỗi lệnh được đặt trên một dòng – dòng lệnh;
- Lệnh có 2 dạng:
  - Lệnh thật: là các lệnh gợi nhớ của VXL
    - VD: MOV, SUB, ADD,...
    - Khi dịch, lệnh gợi nhớ được dịch ra mã máy
  - Lệnh giả: là các hướng dẫn chương trình dịch
    - VD: MAIN PROC, .DATA, END MAIN,...
    - Khi dịch, lệnh giả không được dịch ra mã máy mã chỉ có tác dụng định hướng cho chương trình dịch.
- Không phân biệt chữ hoa hay chữ thường trong các dòng lệnh hợp ngữ khi được dịch.

3.2. Cú pháp của chương trình hợp ngữ

❖ Cấu trúc dòng lệnh hợp ngữ:

[Tên] [Mã lệnh] [Các toán hạng] [Chú giải]

START: MOV AH, 100 ; Chuyển 100 vào thanh ghi AH

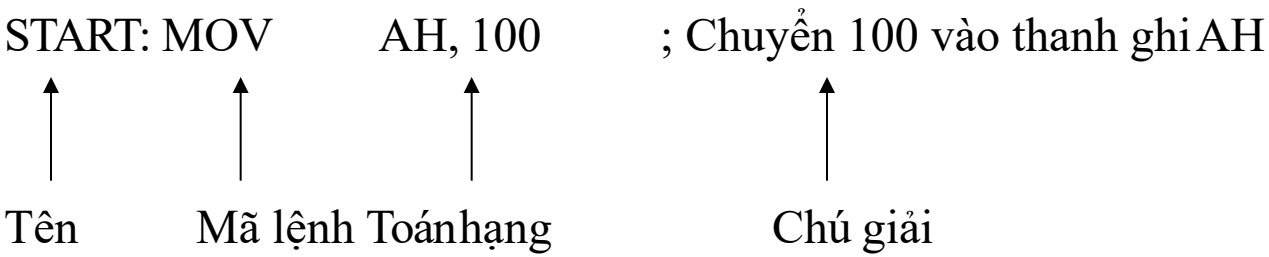
❖ Các trường của dòng lệnh:

- Tên:
  - Là nhãn, tên biến, hằng hoặc thủ tục. Sau nhãn là dấu hai chấm (:)
  - Các tên sẽ được chương trình dịch gán địa chỉ ô nhớ.
  - Tên chỉ có thể gồm các chữ cái, chữ số, dấu gạch dưới và phải bắt đầu bằng 1 chữ cái
- Mã lệnh: có thể gồm lệnh thật và giả

3.2. Cú pháp của chương trình hợp ngữ

❖ Các trường của dòng lệnh:

- Toán hạng:
  - Số lượng toán hạng phụ thuộc vào lệnh cụ thể
  - Có thể có 0, 1 và 2 toán hạng.
- Chú giải:
  - Là chú thích cho dòng lệnh
  - Bắt đầu bằng dấu chấm phảy (;)



3.3. Dữ liệu cho chương trình hợp ngữ

- Dữ liệu số:
  - Thập phân: 0-9
  - Thập lục phân: 0-9, A-F
    - Bắt đầu bằng 1 chữ (A-F) thì thêm 0 vào đầu
    - Thêm ký hiệu H (Hexa) ở cuối
    - VD: 80H, 0F9H
  - Nhị phân: 0-1
    - Thêm ký hiệu B (Binary) ở cuối
    - VD: 0111B, 1000B
- Dữ liệu ký tự:
  - Bao trong cặp nháy đơn hoặc kép
  - Có thể dùng ở dạng ký tự hoặc mã ASCII
    - ‘A’ = 65, ‘a’ = 97

3.4. Hằng và biến

❖ Hằng (constant):

- Là các đại lượng không thay đổi giá trị
- Hai loại hằng:
  - Hằng giá trị: ví dụ 100, ‘A’
  - Hằng có tên: ví dụ MAX_VALUE
- Định nghĩa hằng có tên:
  <Tên hằng> EQU <Giá trị>
  VD:

```
MAX       EQU   100
ENTER     EQU   13
ESC       EQU   27
```

3.4. Hằng và biến

❖ Biến (variable):

- Là các đại lượng có thể thay đổi giá trị
- Các loại biến:
  - Biến đơn
  - Biến mảng
  - Biến xâu ký tự
- Khi dịch biến được chuyển thành địa chỉ ô nhớ

3.4. Hằng và biến

❖ Định nghĩa biến đơn:

Tên biến DB Giá trị khởi đầu: Định nghĩa biến byte

Tên biến DW Giá trị khởi đầu: Định nghĩa biến word

Tên biến DD Giá trị khởi đầu: Định nghĩa biến double word

Ví dụ:

```
X    DB    10      ; Khai báo biến X và khởi trị 10
Y    DW    ?       ; Khai báo biến Y và không khởi trị
Z    DD    1000    ; Khai báo biến X và khởi trị 1000
```

3.4. Hằng và biến

❖ Định nghĩa biến mảng:

Tên mảng DB D/s giá trị khởi đầu

Tên mảng DB Số phần tử Dup(Giá trị khởi đầu)

Tên mảng DB Số phần tử Dup(?)

Định nghĩa tương tự cho các kiểu DW và DD

Ví dụ:

```
X   DB    10, 2, 5,  6, 1     ; Khai báo mảng X gồm 5 phần tử có khởi trị
Y   DB    5          DUP(0)   ; Khai báo mảng Y gồm 5 phần tử khởi trị 0
Z   DB    5          DUP(?)   ; Khai báo mảng Z gồm 5 phần tử không khởi trị
```

3.4. Hằng và biến

- Định nghĩa biến xâu ký tự: có thể được định nghĩa như một xâu ký tự hoặc một mảng các ký tự

  Ví dụ:

  str1 DB ‘string’

  str2 DB 73H, 74H, 72H, 69H, 6EH, 67H

  str3 DB 73H, 74H, ‘r’, ‘i’, 69H, 6EH, 67H

3.5. Khung chương trình hợp ngữ

❖ Khai báo qui mô sử dụng bộ nhớ:

.Model <Kiểu kích thước bộ nhớ>

❖ Các kiểu kích thước bộ nhớ:

- Tiny (hẹp): mã lệnh và dữ liệu gói gọn trong một đoạn
- Small (nhỏ): mã lệnh gói gọn trong một đoạn, dữ liệu gói gọn trong một đoạn
- Medium (vừa): mã lệnh không gói gọn trong một đoạn, dữ liệu gói gọn trong một đoạn
- Compact (gọn): mã lệnh gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn
- Large (lớn): mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, không có mảng lớn hơn 64K
- Huge (rất lớn): mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, có mảng lớn hơn 64K.

3.5. Khung chương trình hợp ngữ

❖ Khai báo đoạn ngăn xếp:

.Stack <Kích thước ngăn xếp>

VD:

.Stack 100H; khai báo kích thước ngăn xếp 100H=256 byte

❖ Khai báo đoạn dữ liệu:

.Data

;Định nghĩa các biến và hằng
;Tất cả các biến và hằng phải được khai báo ở đoạn dữ liệu VD:

.Data

```
MSG     DB      'Hello!$'
ENTER   DB      13
MAX     DW      1000
```

3.5. Khung chương trình hợp ngữ

❖ Khai báo đoạn mã:

```
.Code
; Các lệnh của chương trình
        .Code
        Jmp Start
        ; khai bao du lieu
        Start:
         mov AX,@Data
         mov DS, AX
            ; các lệnh của chương trình chính
        Mov AH, 4CH
        Int 21h
        End Start              ; kết thúc chương trình chính
            ; các chương trình con – nếu có
```

3.5. Khung chương trình hợp ngữ - tổng hợp

```
.Model Small
.Stack 100H
.Data
    ; khai báo các biến và hằng
.Code
    jmp Start
Start:
    ; khởi đầu cho thanh ghi DS
    MOV AX,          ; nạp địa chỉ đoạn dữ liệu vào AX
    @Data  MOV       ; nạp địa chỉ đoạn dữ liệu vào DS
    DS, AX
    ; các lệnh của chương trình chính

    ; kết thúc, trở về chương trình gọi dùng hàm 4CH của ngắt 21H
    MOV AH, 4CH
    INT 21H
End Start
    ; các chương trình con (nếu có)
```

3.5. Khung chương trình hợp ngữ - ví dụ

```
; Chương trình in ra thông điệp: Hello World!
.Model Small
.Stack 100H
.Data
    ; khai báo các biến và hằng
    CRLF  DB 13,10, *       ; xuống dòng
    MSG   DB 'Hello World!$'
.Code
MAIN
Proc
    ; khởi đầu cho thanh ghi DS
    MOV AX, @Data ; nạp địa chỉ đoạn dữ liệu vào
    AX  MOV DS, AX        ; nạp địa chỉ đoạn dữ liệu
    vào DS
```

3.5. Khung chương trình hợp ngữ - ví dụ

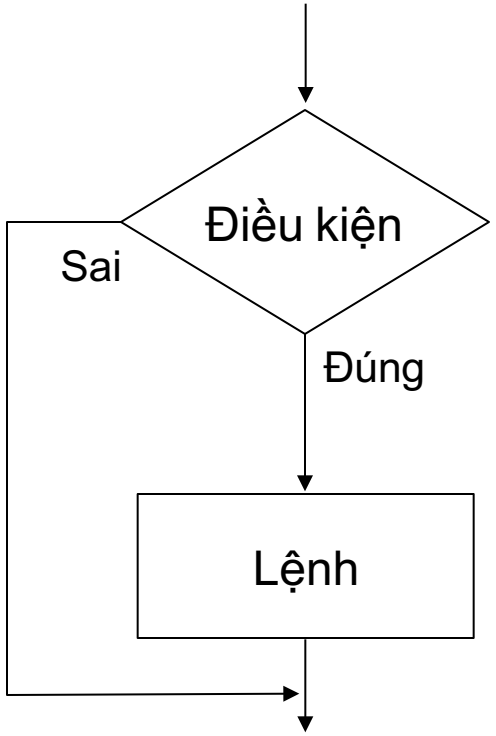
```
    ; xuống dòng
    MOV AH, 9
    LEA DX, CRLF     ; nạp địa chỉ CRLF vào DX
    INT 21H
    ; hiện lời chào dùng hàm 9 của ngắt 21H
    MOV AH, 9
    LEA DX, MSG      ; nạp địa chỉ thông điệp vào DX
    INT 21H          ; hiện thông điệp
    ; kết thúc, trở về chương trình gọi dùng hàm 4CH của ngắt 21H
    MOV AH, 4CH
    INT 21H
MAIN Endp
END MAIN
```

3.6. Các cấu trúc điều khiển

- Cấu trúc lựa chọn
  - Rẽ nhánh kiểu IF ... THEN
  - Rẽ nhánh kiểu IF ... THEN ... ELSE
  - Rẽ nhiều nhánh
- Cấu trúc lặp
  - Lặp kiểu for
  - Lặp kiểu repeat ... until

3.6. Các cấu trúc điều khiển - IF ... THEN

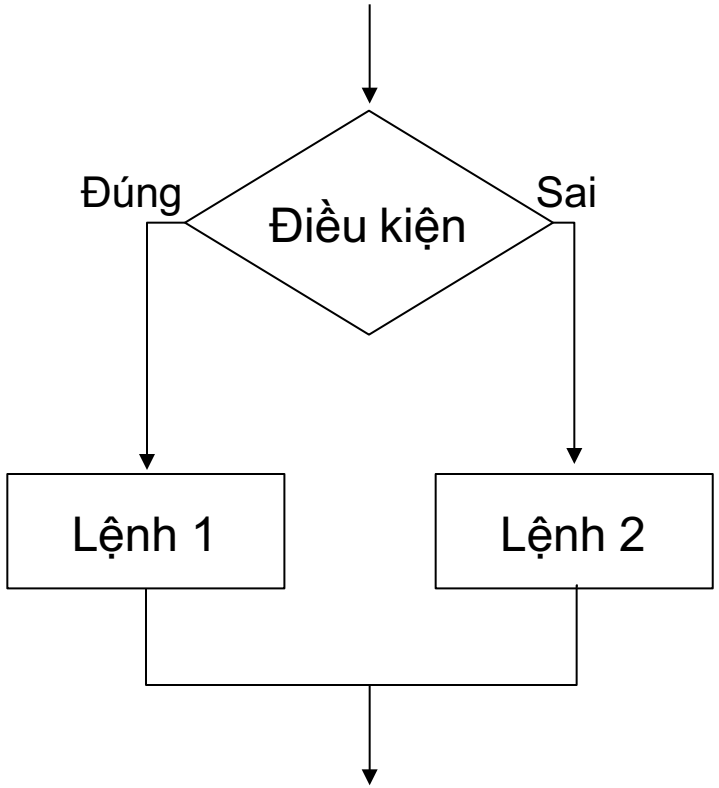
- IF điều kiện THEN thao tác
- Gán BX giá trị tuyệt đối AX
  1. CMP AX,0
  2. JNL GAN
  3. NEG AX
  4. GAN: MOV BX, AX



3.6. Các cấu trúc điều khiển
- IF ... THEN ... ELSE

Gán bít dấu của AX cho CL:

```
        CMP AX, 0   ; AX >0 ?
        JNS DG      ; đúng
        MOV CL, 1 ; không, CL←1
        JMP RA      ; nhảy qua nhánh kia
DG: MOV CL, 0 ; CL←0
RA:
```

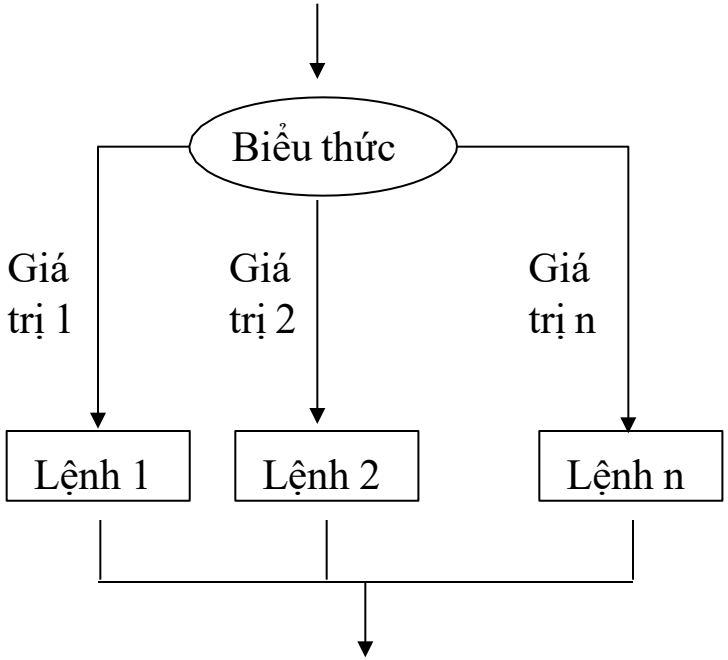


3.6. Các cấu trúc điều khiển - Rẽ nhiều nhánh

Gán giá trị cho CX theo qui tắc:

- Nếu AX<0 thì CX=-1
- Nếu AX=0 thì CX=0
- Nếu AX>0 thì CX=1

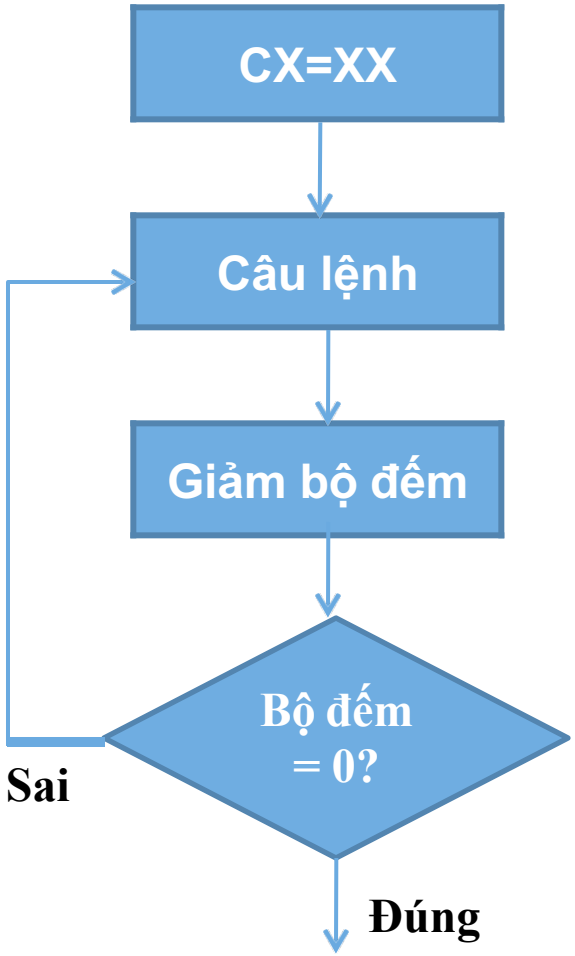
```
        CMP AX, 0
        JL AM
        JE KHONG
        JG DUONG
AM:     MOV CX, -1
        JMP RA
DUONG:  MOV CX, 1
        JMP RA
KHONG:  MOV CX, 0
RA:
```



3.6. Các cấu trúc điều khiển – Lặp kiểu for

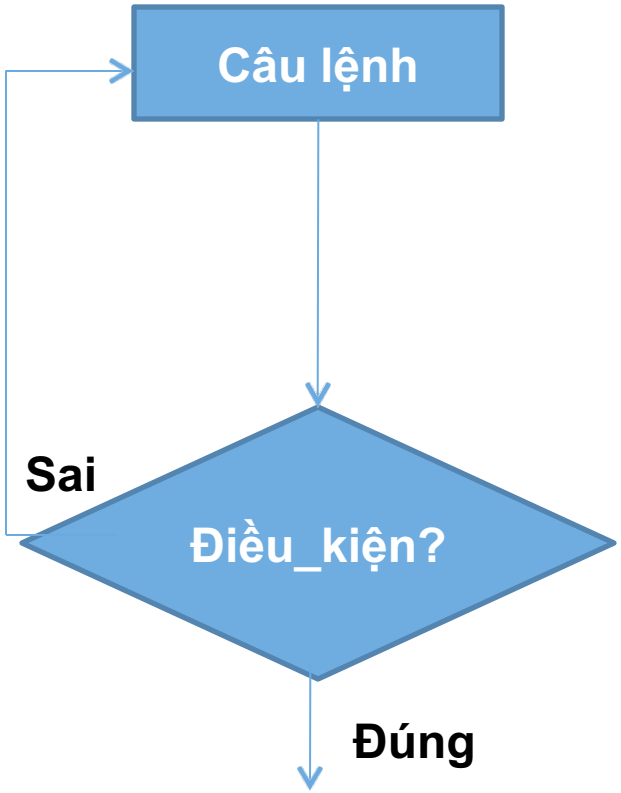
- Sử dụng lệnh LOOP
- Số lần lặp CX

```
1.          MOV CX,10
2.          MOV AH,2
3.          MOV DL,'9'
4.    Hien: INT 21H
5.          LOOP Hien
```

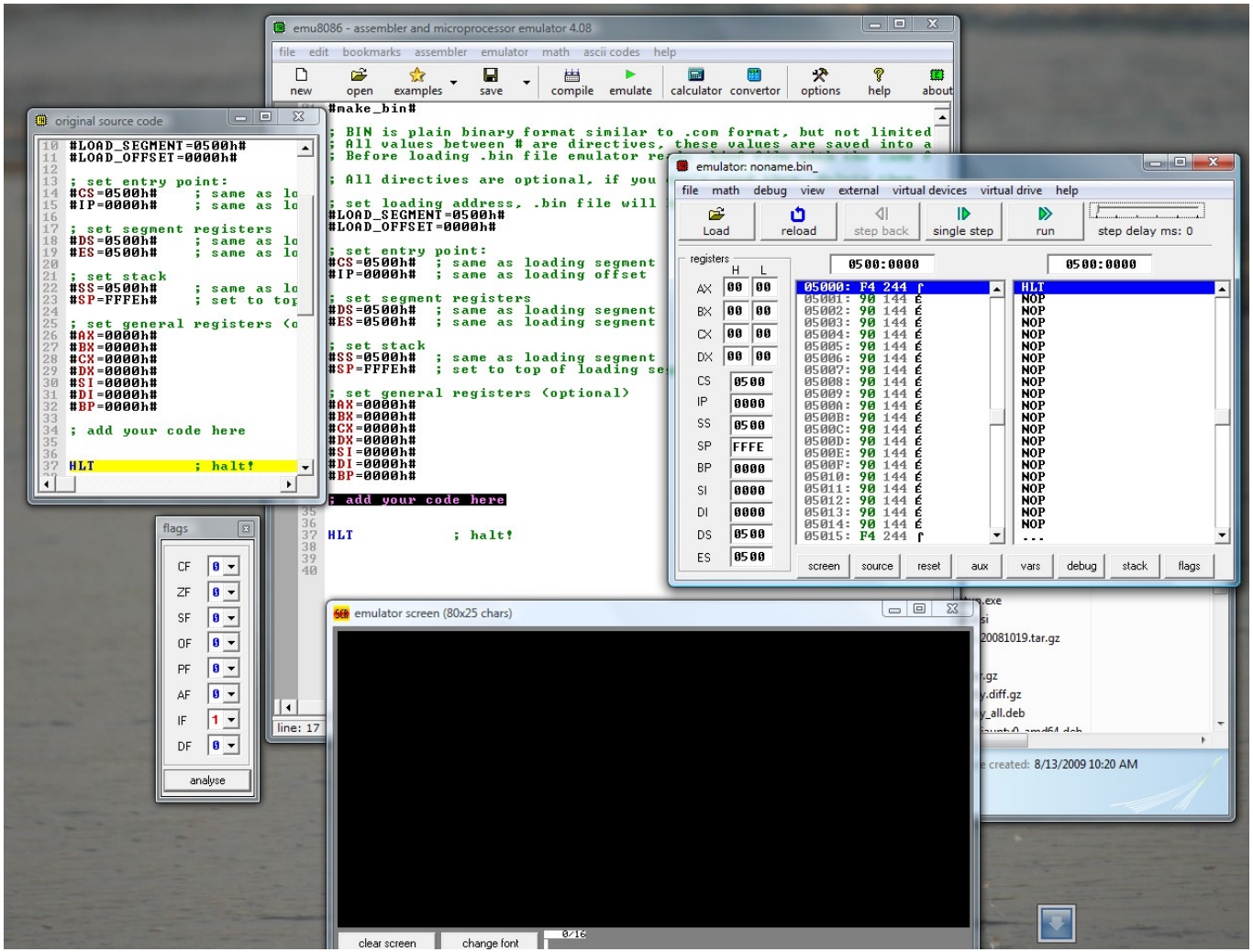


3.6. Các cấu trúc điều khiển – Lặp kiểu repeat ... until

1. …
2. Tiep:…
3. ….
4. CMP X,Y; điều kiện
5. Quay lại Tiếp nếu điều_kiện=sai;



3.7. Giới thiệu phần mềm mô phỏng emu8086



3.8. Ví dụ - Một số dịch vụ của ngắt 21H

- **Hàm 1 của ngắt INT 21H**: đọc 1 ký tự từ bàn phím

  Vào:AH = 1

  Ra:AL = mã ASCII của ký tự cần hiện thị

  AL = 0 khi ký tự gõ vào là phím chức năng

- **Hàm 2 của ngắt INT 21H**: hiện 1 ký tự lên màn hình

  Vào: AH = 2

  DL = mã ASCIIcủa ký tự cần hiện thị.

  Ra: Không

3.8. Ví dụ - Một số dịch vụ của ngắt 21H

- **Hàm 9 của ngắt INT 21H**: hiện chuỗi ký tự với $ ở cuối lên màn hình

  Vào: AH = 9

  DX = địa chỉ lệch của chuỗi ký tự cần hiện thị.

  Ra: Không

- **Hàm 4CH của ngắt INT 21H**: kết thúc chương trình kiểu EXE

  Vào: AH = 4CH

  Ra: Không

VD1- Hiện các lời chào ta và tây

```
. Model Small
. Stack 100
. Data
        CRLF        DB      13, 10, '$'
        Chao tay    DB      'hello!$'
        ChaoTa  DB          'Chao ban!$'
. Code
MAIN Proc
        MOV AX, @ Data ; khởi đầu thanh ghi DS
        MOV DS, AX
        ; hiện thị lời chào dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, ChaoTay
        INT 21H
```

VD1- Hiện các lời chào ta và tây

```
        ; cách 5 dòng dùng hàm 9 của INT 21H
        LEA DX, CRLF
        MOV CX, 6           ;CX chứa số dòng cách +1
LAP: INT 21H
        LOOP LAP
        ; hiện thị lời chào dùng hàm 9 của INT 21H
        LEA DX, ChaoTa
        INT 21H
        ; trở về DOS dùng hàm 4 CH của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN Endp
        END MAIN
```

VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

```
.Model small
.Stack 100H
.Data
; source string
str1 DB 'a','5', 'B', '?', 'd', 'g', 'P','N','k','*'
     DB 10,13,'$'
; destination string
str2 DB 10 DUP(' ')
       DB '$'
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    mov es,ax
```

VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

```
 ; make SI points to str1 and DI to
    str2  lea si, str1
    lea di,
    str2  cld
    mov cx, 10
Start:
    lodsb
    ; check if it is lower
    case  cmp al, 'a'
    jl
    NotLowerCase
    cmp al, 'z'
    jg NotLowerCase
    ; is lower case, convert to upper
    case  sub al, 20H
    ; store to new string
 NotLowerCase:
   stosb  loop Start
```

VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

```
; print the original
   string  lea dx, str1
   mov ah, 9
   int 21H
   ; print the
   output  lea dx,
   str2
   mov ah, 9
   int 21H

   ; end program
   mov ah, 4CH
   int 21H
main endp
end main
```

VD3- Tìm số lớn nhất trong 1 dãy

```
.Model small
.Stack 100H
.Data
; source string
list DB 1,4,0,9,7,2,4,6,2,5
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    cld
    mov cx, 9
    lea si, list        ; si points to list
    mov bl, [si]        ; max <-- 1st element
    inc si
```

VD3- Tìm số lớn nhất trong 1 dãy

```
Start:
    lodsb
    cmp al,
    bl
    jle BYPASS
    mov bl, al; al>bl --> bl to store new
    max  BYPASS:
    loop Start
    ; print the max
    add bl, '0' ; digit to
    char  mov dl,bl
    mov ah, 2
    int 21H
        ; end
    program  mov
    ah, 4CH
    int 21H
main
endp
End Main
```

3.9. Tạo và sử dụng chương trình con

- Chương trình con (còn gọi là thủ tục (procedure) hoặc hàm (function)):
  - Thường gồm một nhóm các lệnh gộp lại;
  - Được sử dụng thông qua tên và các tham số.
- Ý nghĩa của việc sử dụng chương trình con:
  - Chia chức năng giúp chương trình trong sáng, dễ hiểu, dễ bảo trì;
  - Chương trình con được viết một lần và có thể sử dụng nhiều lần.

3.9.1 Chương trình con – Khai báo và sử dụng

❖ Khai báo

```
<name> PROC

        ; here goes the code
        ; of the procedure ...

RET
<name> ENDP
```

❖ Sử dụng: gọi chương trình con

```
Call  <proc_name>
```

3.9.1 Chương trình con – Khai báo và sử dụng

```
MOV    AL, 1
MOV    BL, 2

CALL   m2
; other instructions
MOV CX, 30
;
; define a proc
; input: AL, BL
; Output: AX
m2      PROC
    MUL    BL          ; AX = AL * BL.
    RET                ; return to caller.
m2      ENDP
```

3.9.2 Chương trình con – Truyền tham số

- Phục vụ trao đổi dữ liệu giữa chương trình gọi và chương trình con;
- Các phương pháp truyền tham số:
  - Truyền tham số thông qua các thanh ghi
    - Đưa giá trị vào các thanh ghi lưu tham số cần truyền trước khi gọi hoặc trở về từ chương trình con
  - Truyền tham số thông qua các biến toàn cục
    - Biến toàn cục (định nghĩa trong đoạn dữ liệu ở chương trình chính) có thể được truy nhập ở cả chương trình chính và chương trình con.
  - Truyền tham số thông qua ngăn xếp
    - Sử dụng kết hợp các lệnh PUSH / POP để truyền tham số.

3.9.2 Chương trình con – Truyền tham số

- Bảo vệ các thanh ghi:
  - Cần thiết phải bảo vệ giá trị các thanh ghi sử dụng trong chương trình gọi khi chúng cũng được sử dụng trong chương trình con.
  - Giá trị của các thanh ghi có thể bị thay đổi trong chương trình con → sai kết quả ở chương trình gọi.
- Các phương pháp bảo vệ các thanh ghi:
  - Sử dụng PUSH và POP cho các thanh ghi tổng quát, chỉ số và con trỏ;
  - Sử dụng PUSHF và POPF cho thanh ghi cờ;
  - Sử dụng qui ước thống nhất về sử dụng các thanh ghi.

3.9.3 Chương trình con – Ví dụ 1

```
; Find max of a list and print out the max
.Model small
.Stack 100H
.Data
; source string
list DB 1,4,0,9,7,2,4,6,2,5
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    cld
    mov cx, 9
    lea si, list     ; si points to list
    mov bl, [si]    ; max <-- 1st element
    inc si
```

3.9.3 Chương trình con – Ví dụ 1

```
Start:
    lodsb
    cmp al,
    bl
    jle BYPASS
    mov bl, al; al>bl --> bl to store new
    max  BYPASS:
    loop Start

    ; print the max
    call printSingleDigit

    ; end program
    mov ah, 4CH
    int 21H
main endp
```

3.9.3 Chương trình con – Ví dụ 1

```
;_____________________
; proc to print out a single digit
number
; input: bl to contain the digit to
print  printSingleDigit proc
    push dx
    push ax
    add bl, '0' ; digit to
    char  mov dl,bl
    mov ah, 2
    int 21H
    pop ax
    pop dx
    ret
printSingleDigit
endp  end main
```

3.9.3 Chương trình con – Ví dụ 2

```
; convert lower case chars to upper cases
.Model small
.Stack 100H
.Data
; source string
str1 DB 'a','5', 'B', '?', 'd', 'g', 'P','N','k','*'
     DB 10,13,'$'
; destination string
str2 DB 10 DUP(' ')
     DB '$'
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    mov es,ax
    ; make SI points to str1 and DI to str2
    lea si, str1
    lea di, str2
    cld
    mov cx, 10
```

3.9.3 Chương trình con – Ví dụ 2

```
Start:
    lodsb
    ; check if it is lower
    case  cmp al, 'a'
    jl NotLowerCase
    cmp al, 'z'
    jg NotLowerCase
    ; is lower case, convert to upper
    case  sub al, 20H

    ; store to new
    string
    NotLowerCase:
    stosb
    loop Start

    ; print the original
     string  lea dx, str1
    call printString
```

3.9.3 Chương trình con – Ví dụ 2

```
    ; print the
    output  lea dx,
    str2  call
    printString
    ; end program
    mov ah, 4CH
    int 21H
main endp
; ____________________________
; proc to print a string
; input: DX to contain the relative address of the string
printString proc
    push ax     ; store AX into stack
    mov ah, 9
    int 21H
    pop ax      ; restore AX from stack
    ret
printString endp
end main
```

3.9.3 Chương trình con – Ví dụ 3

```
; Sort a list to accending order
; print out the original and sorted lists
.Model small
.Stack 100H
.Data
    LIST_COUNT EQU 10
    list DB 1,4,0,3,7,2,8,6,2,5
    CRLF DB 13,10,'$'
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax

    ; print the original list
    mov cx, LIST_COUNT
    lea si, list
    call printList
```

3.9.3 Chương trình con – Ví dụ 3

```
lea si, list       ; si points to list
mov bl, 1          ; main counter
MainLoop:
   mov al, [si]    ; al <-- [si]
    mov di, si
    mov bh, bl     ; sub-counter
    mov dx, di     ; dx to store min position
    SubLoop:
      inc di
      inc bh
     cmp al,
    [di]  jle
    NotMin
    mov al, [di]
    mov dx, di
    NotMin:
     cmp bh, LIST_COUNT
     je ExitSub
     jmp SubLoop
  ExitSub:
```

3.9.3 Chương trình con – Ví dụ 3

```
; swap the position if min is different from
  first place  mov di, dx
  cmp si,
  di  je
  NoSwa
  p
  call swapMemLocation
NoSwap:
  inc bl
  cmp bl,
  LIST_COUNT
  je ExitMain
  inc si
  jmp
MainLoop
ExitMain:
```

3.9.3 Chương trình con – Ví dụ 3

```
; print the new line
   chars  lea dx, CRLF
   call printString

   ; print the sorted list
   mov cx,
   LIST_COUNT
   lea si, list   ; si points to
   list  call printList

   ; end
   program
   mov ah,
   4CH  int
   21H
main endp
```

3.9.3 Chương trình con – Ví dụ 3

```
;_____________________
; swap the value of 2 memory locations
; input: si points to the 1st memory
location
;       di points to the 2nd memory
location  swapMemLocation proc
    push ax
    mov al,
    [si]  mov
    ah, [di]
    mov [si],
    ah
    mov [di],
    al  pop ax
    ret
swapMemLocation
endp
```

3.9.3 Chương trình con – Ví dụ 3

```
;_____________________
; print the list
; input: SI to store the start address of the
list
;       CX to store the number of
elements  printList proc
    push dx
  StartPrint
  :
    mov dl, [si]
    call
    printSingleDigit
    inc si
    loop StartPrint
    pop
    dx  ret
printList endp
```

3.9.3 Chương trình con – Ví dụ 3

```
; print a string ending with $
;input: DX to point to
string  printString proc
    push ax
    mov ah, 9
    int 21H
    pop ax
    ret
printString endp
;_____________________________
; proc to print out a single digit number
; input: dl to contain the digit to
print  printSingleDigit proc
    push ax
    add dl, '0' ; digit to char
    mov ah, 2
    int 21H
    pop ax
    ret
printSingleDigit
endp  end main
```

3.10 Tạo và sử dụng macro

- Macro là một đoạn mã được đặt tên và có thể được chèn vào bất cứ vị trí nào trong đoạn mã của chương trình
- Đặc điểm của macro:
  - Macro hỗ trợ danh sách các tham số
  - Macro chỉ tồn tại khi soạn thảo mã. Khi dịch, các macro sẽ được thay thế bằng đoạn mã thực của macro.
  - Nếu một macro không được sử dụng, mã của nó sẽ bị loại khỏi chương trình sau khi dịch.
  - Macro nhanh hơn thủ tục/hàm do mã của macro được chèn trực tiếp vào chương trình và nó không đòi hỏi cơ chế gọi thực hiện (lưu địa chỉ) và trở về (khôi phục địa chỉ trở về) như chương trình con.

3.10 Tạo và sử dụng macro

❖ Định nghĩa macro:

name MACRO [parameters,...]

<instructions>

ENDM

❖ Sử dụng macro:

<macro_name> [real parameters]

3.10 Tạo và sử dụng macro

❖ Ví dụ

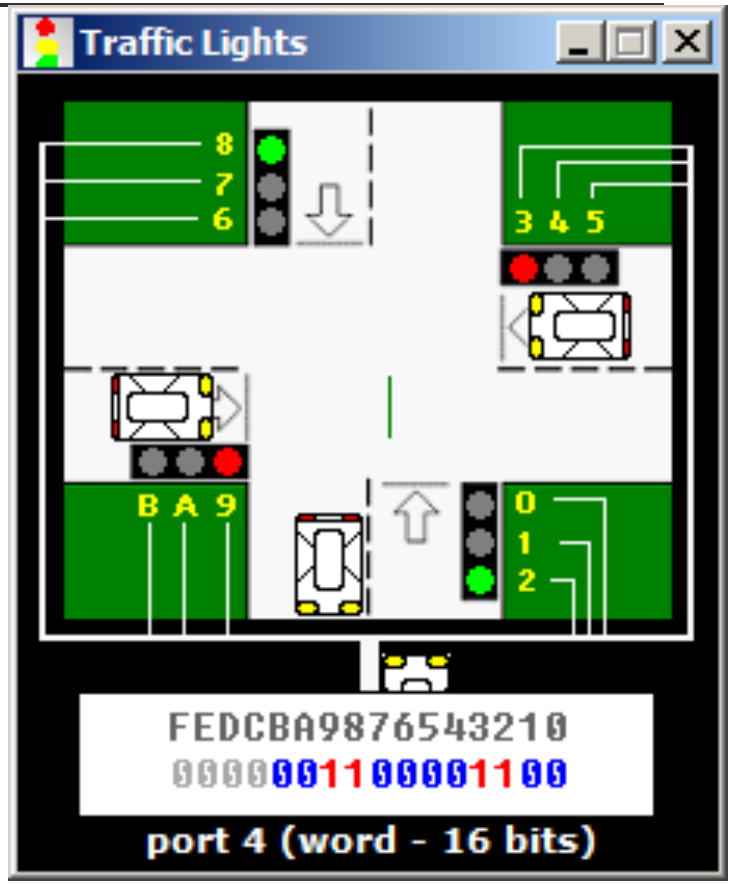
```
MyMacro     MACRO p1, p2, p3
    MOV AX, p1
    MOV BX, p2
    MOV CX, p3
ENDM
;...
MyMacro 1, 2, 3
MyMacro 4, 5, DX
```

Được chuyển thành sau dịch:

```
MOV   AX,   00001h
MOV   BX,   00002h
MOV   CX,   00003h
MOV   AX,   00004h
MOV   BX,   00005h
MOV CX, DX
```

3.11 Giới thiệu thiết bị ảo – Đèn giao thông

- Thiết bị ảo hệ thống đèn giao thông sử dụng cổng số 4 – cổng 16 bít để nhận thông tin điều khiển;
- Sử dụng 12 bít (0-11) cho 4 cụm đèn:
  - Mỗi cụm gồm 3 đèn Green, Yellow và Red;
  - Bít 0 – tắt đèn, bít 1 – bật đèn
- 4 bít (12-15) không sử dụng – nên đặt là 0.



3.11 Giới thiệu thiết bị ảo – Đèn giao thông

❖ Điều khiển đèn giao thông:

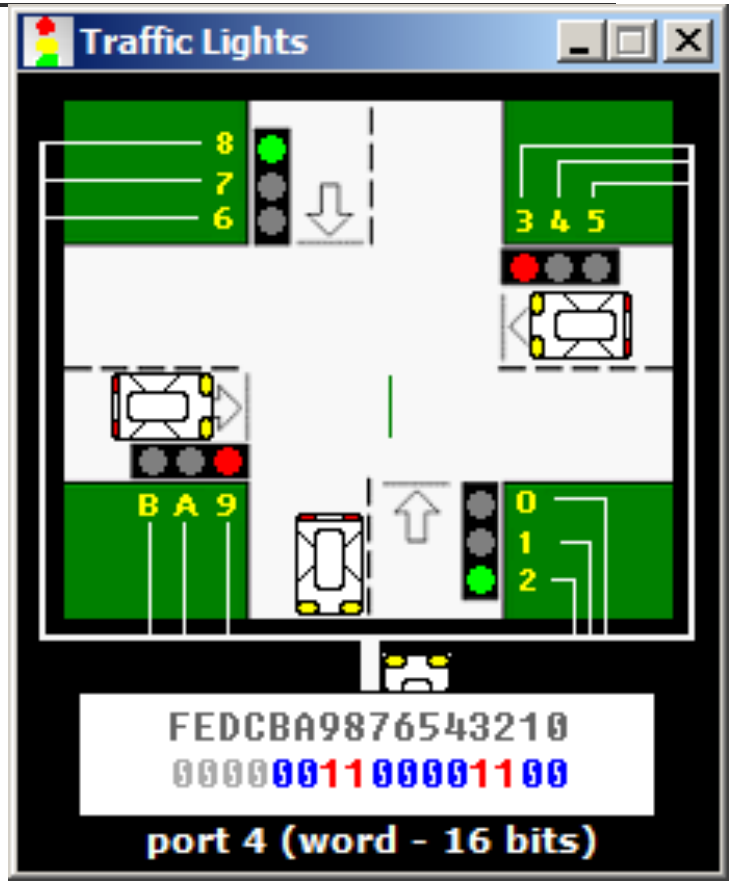
- Gửi từ điều khiển (2 bytes) ra cổng số 4;
- Các bít của từ điều khiển được đặt sao cho phù hợp với ý đồ điều khiển đèn (Bít 0 – tắt đèn, bít 1 – bật đèn)

VD: từ điều khiển:

0000 001 100 001 100

GYR GYR GYR GYR

- Dùng hàm 86h của ngắt BIOS 15h để tạo thời gian đợi – thời gian giữ trạng thái vừa thiết lập của cụm đèn. Số micro giây được đặt vào CX:DX trước khi gọi ngắt.



3.11 Giới thiệu thiết bị ảo – Đèn giao thông

- mov ax, all_red out 4, ax
- mov si, offset situation next:
- mov ax, [si] out 4, ax
- ; wait 5 seconds (5 million microseconds)

```
mov cx, 4Ch ; 004C4B40h = 5,000,000
mov     dx, 4B40h
mov     ah, 86h
int     15h
add si, 2 ; next situation
cmp si, sit_end
jb  next
mov si, offset situation
jmp next
```

3.11 Giới thiệu thiết bị ảo – Đèn giao thông

```
;                    FEDC_BA98_7654_3210
situation       dw  0000_0011_0000_1100b    dw
s1              0000_0110_1001_1010b        dw
s2              0000_1000_0110_0001b        dw
s3              0000_1000_0110_0001b        dw
s4              0000_0100_1101_0011b
sit_end = $


all_red         equ     0000_0010_0100_1001b
```

3.11 Giới thiệu thiết bị ảo – Đèn giao thông

```
.Model small
.Stack 100H
.Data
  ;                              GYR GYR GYR GYR
  R1  DW                    0000_0011_0000_1100b
  R2  DW                    0000_0010_1000_1010b
  R3  DW                    0000_1000_0110_0001b
  R4 DW                     0000_0100_0101_0001b
  ;                        FEDC_BA9 876 543 210
all_red          equ      0000_0010_0100_1001b
PORT EQU  4       ; output port

  ; time constants (in secs)
  WAIT_3_SEC_CX     EQU    2Dh
  WAIT_3_SEC_DX     EQU    0C6C0h

  WAIT_10_SEC_CX     EQU    98h
  WAIT_10_SEC_DX     EQU    9680h
```

3.11 Giới thiệu thiết bị ảo – Đèn giao thông

```
.code
; define a macro
waitMacro macro t1,
t2
  mov cx, t1
  mov dx, t2
  mov     ah, 86h
  int     15h
waitMacro endm
main proc
  ; initilize the ds and es
  registers  mov ax, @Data
  mov ds,ax

  ; set lights to Red for all
  direction  mov ax, all_red
  out PORT, ax
  waitMacro WAIT_3_SEC_CX,
  WAIT_3_SEC_DX
```

3.11 Giới thiệu thiết bị ảo – Đèn giao thông

```
Start:
  lea si, R1
  mov ax,
  [si]
  out PORT, ax

  waitMacro WAIT_10_SEC_CX,

  WAIT_10_SEC_DX  lea si, R2
  mov ax, [si]
  out PORT, ax

  waitMacro WAIT_3_SEC_CX,

  WAIT_3_SEC_DX  lea si, R3
  mov ax, [si]
  out PORT, ax
```

3.11 Giới thiệu thiết bị ảo – Đèn giao thông

```
waitMacro WAIT_10_SEC_CX,
WAIT_10_SEC_DX

 lea si, R4
 mov ax, [si]
 out PORT, ax

 waitMacro WAIT_3_SEC_CX,

 WAIT_3_SEC_DX  jmp Start


 ; end
 program
 mov ah, 4CH
 int 21H
main endp

end main
```