

TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ
BỘ MÔN ĐIỆN TỬ - VIỄN THÔNG



Giáo trình thực hành

LẬP TRÌNH HỆ THỐNG

Biên soạn:

ThS. Nguyễn Hứa Duy Khang
Ks. Trần Hữu Danh

**-ĐHCT-
5-2008**

NỘI DUNG

Giới thiệu

BÀI 1: NHẬP MÔN HỢP NGỮ	1
1. Mục tiêu	1
2. Kiến thức cần chuẩn bị	1
3. Nội dung thực hành	1
3.1. Khảo sát lệnh Intel-8086	1
3.2. Cấu trúc chương trình dạng EXE	1
3.3. Viết chương trình đơn giản	3
4. Bài tập đề nghị	3
BÀI 2: XUẤT NHẬP KÝ TỰ	5
1. Mục tiêu	5
2. Kiến thức cần chuẩn bị	5
3. Nội dung thực hành	5
3.1. In một ký tự ra màn hình	5
3.2. In chuỗi ký tự ra màn hình	6
3.3. Nhận một ký tự từ bàn phím	6
3.4. Nhận chuỗi ký tự từ bàn phím	7
4. Bài tập đề nghị	8
BÀI 3: CẤU TRÚC RỄ NHÁNH – VÒNG LẶP	9
1. Mục tiêu	9
2. Tóm tắt lý thuyết	9
3. Nội dung thực hành	12
3.1. Cấu trúc rẽ nhánh	12
3.2. Cấu trúc vòng lặp	13
4. Bài tập đề nghị	13
BÀI 4: NHẬP XUẤT SỐ DẠNG BIN-HEX-DEC	15
1. Mục tiêu	15
2. Kiến thức cần chuẩn bị	15
3. Nội dung thực hành	15
3.1. Nhập xuất số nhị phân	15
3.2. Nhập xuất số thập lục phân	16
3.3. Xuất số thập phân nguyên dương	18
4. Bài tập đề nghị	19
BÀI 5: XỬ LÝ TẬP TIN	21
1. Mục tiêu	21
2. Kiến thức cần chuẩn bị	21
3. Nội dung thực hành	21
3.1. Tạo tập tin mới	21

3.2. Ghi nội dung tập tin	22
3.3. Đọc nội dung tập tin	23
3.4. Xóa tập tin	25
3.5. Đổi tên tập tin	26
4. Bài tập đề nghị	27
BÀI 6: XỬ LÝ CHUỖI KÝ TỰ	28
1. Mục tiêu	28
2. Kiến thức cần chuẩn bị	28
3. Nội dung thực hành	28
3.1. So sánh chuỗi	28
3.2. Di chuyển chuỗi	30
3.3. Dò tìm trong chuỗi	31

Giới Thiệu

Thực hành Lập trình Hệ Thống giúp cho sinh viên viết được chương trình bằng ngôn ngữ Assembly trên máy tính PC. Giáo trình này chỉ hướng dẫn sinh viên những kỹ năng rất cơ bản trong việc lập trình bằng Assembly như: Sử dụng trình biên dịch hợp ngữ trong môi trường Windows, biên dịch, sửa lỗi và liên kết, khảo sát tập lệnh, các ngắt đơn giản của Hệ điều hành DOS. Để vận dụng và nâng cao được kỹ năng lập trình hệ thống bằng Hợp ngữ, đòi hỏi sinh viên phải nỗ lực rất nhiều trong việc tự học, đọc thêm tài liệu để bổ sung những kiến thức nhất định về phần cứng máy tính cũng như nguyên lý vận hành của các thiết bị ngoại vi có liên quan, như: Máy in, hệ vi điều khiển, cổng vào ra nối tiếp/ song song

Thời lượng của môn học có giới hạn, nên các bài thực hành được tổ chức thành các chủ đề sau đây, mỗi chủ đề thực hành trong 1 buổi (5 tiết):

- Bài 1: Nhập môn hợp ngữ
- Bài 2: Xuất nhập ký tự.
- Bài 3: Cấu trúc rẽ nhánh – Vòng lặp
- Bài 4: Xử lý ký tự - Ký số
- Bài 5: Xử lý tập tin
- Bài 6: Nhập xuất số BIN-HEX-DEC

Để thực hành đạt hiệu quả cao, sinh viên cần phải chuẩn bị cho mỗi bài trước khi vào bắt đầu thực hành. Muốn đạt được điều này, sinh viên phải tuân thủ phương pháp học như sau:

Trước khi vào thực hành:

- Nắm được mục tiêu của bài thực hành.
- Xem lại các kiến thức cần chuẩn bị được nêu ra cho mỗi bài thực hành.
- Nắm được các nội dung cần phải làm trong buổi thực hành.

Trong khi thực hành:

- Tuyệt đối tuân thủ thực hành theo thứ tự của nội dung thực hành. Hoàn thành các vấn đề và trả lời được các câu hỏi đặt ra trong phần trước mới chuyển sang thực hành phần sau.
- Quan sát hiện tượng, những thay đổi, xem xét đánh giá kết quả sau mỗi thao tác thực hành.
- Lập lại các thao tác thực hành nhiều lần, tìm cách giải quyết khác sau khi đã thực hành theo yêu cầu cho mỗi vấn đề. So sánh, nhận xét các cách giải quyết khác nhau.

Sau khi thực hành:

- Đối chiếu từng mục tiêu của bài thực hành với những gì đã thực hành được. Nếu mục tiêu nào chưa thành thạo thì phải tìm cách lập lại thực hành đó để nắm được mục tiêu vững chắc hơn.

Trong quá trình biên soạn, không thể tránh khỏi sơ xuất, hãy chân thành góp ý chỉnh sửa để giáo trình ngày càng hoàn chỉnh hơn.

ThS. Nguyễn Hứa Duy Khang

Email: nhdkhang@ctu.edu.vn

Bài 1: Nhập Môn Hợp Ngữ

1. MỤC TIÊU

- Sử dụng được công cụ Emu8086 để khảo sát các lệnh của Intel-8086.
- Sử dụng được các chức năng cơ bản của công cụ RadASM như: Soạn thảo, Hợp dịch (Assemble), Liên kết (Link) và Chạy chương trình (Run).
- Viết đúng cấu trúc của chương trình hợp ngữ dạng tái định (EXE).
- Đọc hiểu và sửa lỗi chương trình.

2. KIẾN THỨC CẦN CHUẨN BỊ

- Các thao tác cơ bản trên hệ điều hành Windows.
- Cấu trúc chương trình hợp ngữ dạng EXE.
- Quy trình Soạn thảo – Dịch chương trình.
- Các lệnh đơn giản của Intel-8086 thường dùng như: MOV, ADD, SUB, INC, DEC, AND, OR. (Xem cú pháp trong giáo trình)

3. NỘI DUNG THỰC HÀNH

3.1. Khảo sát lệnh Intel-8086:

3.1.1. Nhập vào Emu8086 đoạn lệnh sau đây và dự đoán trước kết quả:

```
MOV AH, 80      ; AH ← 80 (AX = ?)
MOV AL, 86      ; AL ← 86 (AX = ?)
MOV BX, AX      ; BX ← AX (BH = ?, BL = ?)
MOV DH, BL      ; DH ← BL (DH = ?, DX = ?)
MOV DL, BH      ; DL ← BH (DL = ?, DX = ?)
MOV SI, CS      ; SI ← CS (SI = ?)
```

Thực hiện từng lệnh, sau mỗi lệnh ghi lại kết quả các thanh ghi trong ngoặc để đối chiếu với kết quả dự đoán trên và giải thích.

3.1.2. Thực hành tương tự như câu 3.1.1 đối với đoạn lệnh sau:

```
MOV AX, 8086    ; AX ← 8086 (AH = ?, AL = ?)
ADD AL, 3        ; AL ← AL + 3 (AL = ?, AX = ?)
DEC AX          ; AX ← AX - 1 (AH = ?, AL = ?, AX = ?)
SUB AH, 10h      ; AH ← AH - 10h (AH = ?, AL = ?, AX = ?)
AND AX, 0FF0h    ; AX ← AX and 0FF0h (AX = ?)
```

3.1.3. Sinh viên chủ động lập lại ít nhất 1 lần câu 3.1.1 và 3.1.2 với các giá trị toán hạng khác trong mỗi dòng lệnh.

3.2. Cấu trúc chương trình dạng EXE:

3.1.1. [HELLO.ASM] Dùng RadASM để soạn thảo chương trình Hợp ngữ sau đây:

Lưu ý: - Chương trình hoàn toàn không có lỗi.

- Trong đó có những lệnh mà sinh viên chưa học đến, điều này không cần quan tâm, điều cần quan tâm trong bài thực hành này là Cấu trúc chương trình hợp ngữ.
- Đặt tên file chương trình nguồn là HELLO.ASM

```
DSEG SEGMENT ; Tạo đoạn DSEG
    chuoi DB "Hello World!$" ; Khai báo biến chuỗi
DSEG ENDS
CSEG SEGMENT ; Tạo đoạn CSEG
    ASSUME CS: CSEG, DS: DSEG ; CSEG là đoạn lệnh, DSEG là dữ liệu
begin: MOV AX, DSEG ; Khởi động địa chỉ đoạn dữ liệu
        MOV DS, AX
        MOV AH, 09h ; AH ← 09h
        LEA DX, chuoi ; DX ← địa chỉ offset biến chuoi
        INT 21h ; gọi ngắt 21h
        MOV AH, 01h ; AH ← 01h
        INT 21h ; gọi ngắt 21h
        MOV AH, 4Ch ; Thoát chương trình
        INT 21h
CSEG ENDS
END begin
```

- Hợp dịch chương trình HELLO.ASM và kiểm tra xem file HELLO.OBJ đã được tạo ra chưa.
- Liên kết chương trình HELLO, kiểm tra xem file HELLO.EXE đã được tạo ra chưa
- Chạy chương trình HELLO.EXE, quan sát trên màn hình, chương trình trên làm gì?
- Thay đổi "Hello World!\$" thành "Wellcome to Assembly\$". Làm lại các bước a, b và c. Chương trình trên làm gì?

3.2.2. [HELLO2.ASM] Sửa file HELLO.ASM ở trên sao cho giống hệt như chương trình sau và đặt tên lại là HELLO2.ASM (chỉ khác ở những chỗ *in nghiêng*) - Lưu ý: Chương trình sẽ có vài lỗi

- Dịch chương trình HELLO2.ASM, ghi lại các thông tin về lỗi: Số lỗi, những lỗi gì, trên dòng nào?
- Kiểm tra xem file HELLO2.OBJ được tạo ra không? Tại sao?
- Sửa từng lỗi một từ dòng trên xuống, rồi lập lại câu a cho đến khi nào hết lỗi.
- Liên kết chương trình HELLO2. Kiểm tra xem có file HELLO2.EXE không?
- Chạy chương trình HELLO2.EXE, so sánh kết quả với **3.1.1.d**

```
DSEG SEGMENT ; Tạo đoạn DSEG
        chuoi DW "Wellcome to Assembly$" ; Khai báo biến chuỗi
DSEG ENDS
CSEG SEGMENT ; Tạo đoạn CSEG
        ASSUME CS: CSEG, DS: DSEG ; CSEG là đoạn lệnh, DSEG là dữ liệu
begin: MOV BX, SSEG ; Khởi động địa chỉ đoạn dữ liệu
        MOV DS, BX
        MOV AH, 09h ; AH ← 09h
        LEA DH, chuoi ; DX ← địa chỉ offset biến chuỗi
        INT 21h ; gọi ngắt 21h
        MOV AH, 01h ; AH ← 01h
        INT 21h ; gọi ngắt 21h
        MOV AH, 4Ch ; Thoát chương trình
        INT 21h
CSEG ENDS

        END Begin
```

3.3. Viết các chương trình đơn giản:

3.3.1. [SUM1.ASM] Viết chương trình dạng EXE để tính kết quả biểu thức sau, lưu trữ kết quả trong AX:

$$10 + 8086 - 100h + 350 + 0FAh$$

Lưu ý: - Chỉ khai báo 1 đoạn lệnh để viết chương trình.

- Dịch sửa lỗi (nếu có lỗi) và chạy chương trình.
- Dùng Emu8086 để chạy chương trình trên và kiểm tra kết quả lưu trong AX.

3.3.2. [SUM2.ASM] Viết chương trình dạng EXE để tính kết quả biểu thức có dạng tổng quát như sau:

$$KQUA = A + B - C + D + E$$

Trong đó: KQUA, A, B, C, D, E là các biến 2 byte khai báo trong đoạn dữ liệu.

Lưu ý: - Chương trình gồm 2 đoạn: Đoạn lệnh và Đoạn dữ liệu dùng để chứa các Biến.

- Gán giá trị các biến A = 1000, B = 10, C = 1Fh, D = 30h, E = 300Ah. Dịch và chạy chương trình.
- Dùng Emu8086 để kiểm tra kết quả của câu a.
- Áp dụng SUM2.ASM để tính biểu thức đã cho ở câu 3.3.1. Dùng Emu8086 để kiểm tra kết quả.

4. BÀI TẬP ĐỀ NGHỊ:

- Dùng Emu8086 để khảo sát các lệnh khác trong tập lệnh của Intel-8086.
- Tự tìm hiểu thêm những chức năng khác của Emu8086

4.3. Viết từng chương trình tính các biểu thức sau: (Phải viết theo kiểu sử dụng biến để chứa toán hạng và kết quả, SV tự đặt tên biến theo ý của mình)

a. $15h * 250$

d. $1000 \div 100$

b. $16 * 0AF1h$

e. $1000 \div 100h$

c. $300 * 400$

f. $3AB45Eh \div 0A1h$

4.4. Sử dụng Emu8086 để kiểm chứng kết quả của các chương trình đã viết cho 4.3.

Bài 2: Xuất Nhập Ký Tự

1. MỤC TIÊU

- Sử dụng được các ngắt mềm để viết được chương trình: in ký tự - chuỗi ký tự lên màn hình và nhập ký tự - chuỗi ký tự từ bàn phím.
- Hiểu được cách quản lý ký tự và ký số trong Hợp ngữ.

2. KIẾN THỨC CẦN CHUẨN BỊ

- Kết quả bài thực hành 1.
- Các hàm 01h, 02h, 06h, 07h, 08h, 09h, 0Ah của ngắt 21h.
- Bảng mã ASCII.

3. NỘI DUNG THỰC HÀNH

3.1. In 1 ký tự ra màn hình

- Chương trình sử dụng hàm 2, ngắt 21h để in ký tự **B** ra màn hình được viết như sau. Hãy soạn thảo lưu lại thành tập tin nguồn có tên là **BAI_2A.ASM**.

```
CSEG      SEGMENT
          ASSUME CS: CSEG
start:    mov ah, 02h   ; Hàm 2, in 1 ký tự ra màn hình
          mov dl, 'B'   ; DL chứa ký tự cần in
          int 21h       ; gọi ngắt để thực hiện hàm
          mov ah, 08h   ; Hàm 08h, ngắt 21h
          int 21h
          mov ah, 4Ch   ; Thoát khỏi chương trình
          int 21h
CSEG      ENDS
          END start
```

- Dịch sửa lỗi (nếu có) và chạy chương trình để xem kết quả in ra màn hình.
- Các dòng lệnh nào thực hiện chức năng in ký tự **'B'** ra màn hình? Các dòng lệnh khác dùng làm gì?
- Sửa lại chương trình trên để in ra màn hình ký tự **'D'**. Chạy chương trình kiểm chứng kết quả.
- Viết chương trình để in ra màn hình số **9**
- Viết chương trình để in ra màn hình số **89**
- Hai dòng lệnh 6 và 7 có chức năng gì trong chương trình? Nếu không có 2 dòng lệnh ấy thì chương trình thực hiện như thế nào? (*Thử xóa bỏ 2 dòng lệnh ấy rồi chạy chương trình, quan sát kết quả để phát hiện chức năng*)

3.2. In chuỗi ký tự ra màn hình

- Muốn in 1 chuỗi ký tự ra màn hình thì sử dụng hàm 9, ngắt 21h như chương trình sau đây, hãy soạn thảo và đặt tên tập tin nguồn là **BAI_2B.ASM**.
- Dịch, sửa lỗi (nếu có) và chạy chương trình để xem kết quả trên màn hình.
- Viết lại chương trình trên để in ra màn hình chuỗi **“Truong Dai Hoc Can Tho”**.
- Sửa khai báo biến **chuoi** có dạng như sau:
chuoi DB ‘Truong Dai Hoc’, 10, 13, ‘Can Tho\$’
- Dịch và chạy chương trình để xem kết quả. Trong khai báo biến chuoi, 2 giá trị **10, 13** có ý nghĩa gì trong việc in chuỗi ra màn hình.
- Sửa lại chương trình để in ra màn hình số **2006**.

```
DSEG SEGMENT
    chuoi DB 'Chao sinh vien nganh Cong Nghe Thong Tin.$'
DSEG ENDS
CSEG SEGMENT
    ASSUME CS: CSEG, DS: DSEG
start:    mov ax, DSEG
          mov ds, ax
          mov ah, 09h      ; Hàm 9, in chuỗi ký tự ra màn hình
          lea dx, chuoi    ; dl chứa ký tự cần in
          int 21h          ; gọi ngắt thực hiện
          mov ah, 08h
          int 21h
          mov ah, 4Ch      ; thoát khỏi chương trình
          int 21h
CSEG ENDS
          END start
```

3.3. Nhận 1 ký tự từ bàn phím

- Chương trình sau đây (trang 3) sẽ sử dụng hàm 01, ngắt 21h để nhận 1 ký tự từ bàn phím. Soạn thảo và đặt tên tập tin nguồn là **BAI_2C.ASM**.
- Dịch, sửa lỗi (nếu có) và chạy chương trình, gõ phím cần nhập. Quan sát kết quả trên màn hình.
- Ký tự đã nhập được lưu trữ ở đâu và được CPU quản lý ở dạng thức gì? (*Dùng Emu8086 để khảo sát*)
- Sửa chương trình để đọc ký tự bằng hàm 7, ngắt 21h.
- Chạy chương trình và so sánh hoạt động giữa hàm 1 và hàm 7.

```
DSEG SEGMENT
    tbao DB 'Hay go vao 1 phim: $'
DSEG ENDS
CSEG SEGMENT
    ASSUME CS: CSEG, DS: DSEG
start: mov ax, DSEG
        mov ds, ax
        mov ah, 09h    ; In câu thông báo ra màn hình
        lea dx, tbao
        int 21h
        mov ah, 01h    ; Ham 1, nhan ky tu tu ban phim
        int 21h        ; goi ngat thuc hien ham
        mov ah, 4Ch    ; tro ve he dieu hanh
        int 21h
CSEG ENDS
    END start
```

3.4. Nhận 1 chuỗi ký tự từ bàn phím

- Chương trình sử dụng hàm 0Ah, ngắt 21h để nhập 1 chuỗi ký tự từ bàn phím viết như sau. Sinh viên soạn thảo thành tập tin chương trình có tên là **BAI_2D.ASM**.

```
DSEG SEGMENT
    max      DB 30
    len      DB 0
    chuoi    DB 30 dup(?)
    tbao     DB 'Hay go vao 1 chuoi: $'
DSEG ENDS
CSEG SEGMENT
    ASSUME CS: CSEG, DS: DSEG
start: mov ax, DSEG
        mov ds, ax
        mov ah, 09h    ; In câu thông báo ra màn hình
        lea dx, tbao
        int 21h
        mov ah, 0Ah    ; Ham 0Ah, nhap chuoi ky tu tu ban phim
        lea dx, MAX     ; dx chua dia chi vung dem ban phim
        int 21h        ; goi ngat thuc hien ham
        mov ah, 4Ch    ; tro ve he dieu hanh
        int 21h
CSEG ENDS
    END start
```

- Dịch, sửa lỗi và thi hành chương trình trong từng trường hợp sau đây:
 1. Nhập từ bàn phím chuỗi ít hơn 30 ký tự.
 2. Nhập từ bàn phím chuỗi nhiều hơn 30 ký tự.
- Giá trị biến **len** trong mỗi trường hợp là bao nhiêu?
- Tại sao không thể nhập nhiều hơn 30 ký tự? Chuỗi ký tự nhập vào được lưu trữ ở biến nào?
- Sửa chương trình để có thể nhập nhiều hơn 30 ký tự (60 ký tự chẳng hạn).
- Tổng quát, khả năng tối đa của hàm 0Ah, ngắt 21h là nhận chuỗi bao nhiêu ký tự?

4. BÀI TẬP ĐỀ NGHỊ

- 4.1. Viết chương trình sử dụng hàm 7, ngắt 21h để nhận 1 ký tự từ bàn phím, dùng 1 biến để lưu trữ ký tự nhận được (do sinh viên tự đặt tên biến), sau đó sử dụng hàm 2, ngắt 21h để in ra màn hình ký tự nhận được đang lưu trong biến ấy. Chương trình phải có đủ các câu thông báo nhập và xuất.

Ví dụ: Hay go 1 phim: **B**

Ky tu nhan duoc la: **B**

- 4.2. Sửa lại chương trình 4.1 sao cho không cần sử dụng biến để lưu trữ ký tự mà kết quả chạy chương trình vẫn không thay đổi.
- 4.3. Viết chương trình nhận 1 ký tự từ bàn phím, sau đó in ra màn hình ký tự kế trước và kế sau của ký tự vừa nhập

Ví dụ: Hay go 1 phim: **B**

Ky tu ke truoc : **A**

Ky tu ke sau : **C**

- 4.4. Viết chương trình cho phép nhập từ bàn phím tên của 1 người, sau đó in ra màn hình chuỗi có dạng như sau:

Xin chao <tên_đã_nhập>

Ví dụ: Khi chạy chương trình, nhập vào là: **Nguyen Hua Duy Khang**

Chuỗi in ra màn hình sẽ là: **Xin chao Nguyen Hua Duy Khang**

Bài 3: Cấu trúc rẽ nhánh – Vòng lặp

1. MỤC TIÊU

- Hiểu cách so sánh hai số trong hợp ngữ
- Hiểu cách thay đổi thứ tự thực hiện các lệnh
- Biết cách sử dụng các lệnh so sánh, nhảy và lặp

2. TÓM TẮT LÝ THUYẾT

2.1. Lệnh so sánh

Cú pháp: CMP Trái, Phải ; Cờ \leftarrow Trái – Phải

Nếu Trái > Phải \Rightarrow Trái - Phải > 0 : CF = 0 và ZF = 0

Nếu Trái < Phải \Rightarrow Trái - Phải < 0 : CF = 1 và ZF = 0

Nếu Trái = Phải \Rightarrow Trái - Phải = 0 : CF = 0 và ZF = 1

Trái, Phải: Immed, Reg, Mem

Bản chất của lệnh CMP là lệnh SUB Đích, Nguồn (thực hiện phép tính Đích – Nguồn) nhưng kết quả của phép tính không được lưu vào Đích như trong lệnh SUB mà tính chất của kết quả được thể hiện thông qua cờ

Ví dụ: so sánh hai số nguyên dương

MOV AH, 1 ; AH \leftarrow 1

MOV AL, 2 ; AL \leftarrow 2

CMP AH, AL ; CF \leftarrow 1, ZF \leftarrow 0 vì AH < AL

Sau khi thực hiện các lệnh trên, cờ Carry bật (CF=1), báo hiệu rằng AH < AL

2.2. So sánh nhị phân

Cú pháp: TEST Trái, Phải ; Cờ \leftarrow Trái and Phải

Nếu Trái and Phải = 0 thì ZF = 1, ngược lại thì ZF = 0

Bản chất của lệnh TEST là lệnh AND Đích, Nguồn nhưng kết quả của phép tính không được lưu vào Đích như trong lệnh AND mà ảnh hưởng lên cờ.

Ví dụ: kiểm tra hai bit cuối cùng của AL

TEST AL, 3 ; 3h = 11b

Nếu cờ Zero bật (ZF=1), có nghĩa là cả hai bit 0 và 1 của AL đều bằng 0.

2.3. Lệnh nhảy không điều kiện

Cú pháp: JMP <target>; Nhảy đến địa chỉ <Target>

Có các trường hợp sau:

- JMP SHORT <nhân> ; (short jump). Kiểu này chỉ nhảy trong phạm vi từ -128 đến +127 byte so với vị trí hiện tại.

Ví dụ: JMP SHORT Calculate

- **JMP <nhãn>** ; (near jump). Kiểu này nhảy tùy ý trong phạm vi segment.

Ví dụ: JMP Calculate

- **JMP FAR PTR <nhãn>** ; (far jump). Kiểu này nhảy đến bất kì chỗ nào.

Ví dụ: JMP FAR PTR Calculate

- **JMP <con trỏ 2 byte>** ; (near indirect jump). Khi thực hiện, thanh ghi PC sẽ được gán bằng giá trị lưu tại địa chỉ này. Có thể kết hợp dùng với định vị chỉ số.

Ví dụ:

myPointer DW Prepare, Calculate, Check, Output

...

MOV BX, 2 ; chỉ số trong mảng con trỏ

SHL BX, 1 ; nhân đôi

JMP myPointer[BX]

...

Prepare: ; công việc 0

...

Calculate: ; công việc 1

...

Check: ; công việc 2 – nơi cần nhảy đến

...

Output: ; công việc 3

...

- **JMP <con trỏ 4 byte>** ; (far indirect jump). Tương tự trường hợp trên, nhưng con trỏ gồm cả segment và offset. Chỉ khác ở khai báo con trỏ
- **JMP <thanh ghi 2 byte>** ; (indirect jump via regs). Nhảy đến địa chỉ lưu trong thanh ghi AX.

Ví dụ: MOV AX, offset Calculate

...

JMP AX ; PC ← AX

2.4. Lệnh nhảy có điều kiện

Cú pháp: J<điềukiện> <Label>

Các lệnh nhảy có điều kiện bắt đầu bằng chữ J sau đó là các chữ cái biểu thị điều kiện (ví dụ JGE: **J**ump if **G**reater than or **E**qual, nhảy nếu lớn hơn hay bằng), tiếp sau là một tên nhãn.

Điều kiện để lệnh nhảy xem xét khi thi hành là giá trị các cờ được tạo ra từ lệnh CMP hay TEST. Khi sử dụng lệnh nhảy có điều kiện sau khi thực hiện phép so sánh,

phải đặc biệt lưu ý toán hạng trong phép so sánh là số có dấu (signed) hay không có dấu (unsigned) để lựa chọn lệnh cho phù hợp.

Ví dụ: MOV AH,AL ; AL hiện bằng 128
CMP AH,1
JGE Greater ; AH > 1 nhưng không nhảy ????

...

Greater:

Ví dụ: nếu AL là số nguyên không dấu thì đoạn chương trình ở trên phải sửa lại như sau:

MOV AH,AL
CMP AH,1
JAE Greater

...

Greater:

Một số lệnh nhảy có điều kiện thường dùng:

Lệnh	Ý Nghĩa	Điều Kiện
JB	Nhảy nếu nhỏ hơn (Jump if Below)	CF = 1
JNAE	Nhảy nếu không lớn hơn hoặc bằng	
JAЕ	Nhảy nếu lớn hơn hoặc bằng (Jump if Above or Equal)	CF = 0
JNB	Nhảy nếu không nhỏ hơn	
JBE	Nhảy nếu nhỏ hơn hoặc bằng (Jump if Below or Equal)	CF = 1 và ZF = 1
JNA	Nhảy nếu không lớn hơn	
JA	Nhảy nếu lớn hơn (Jump if Above)	CF = 0 và ZF = 0
JNBE	Nhảy nếu không nhỏ hơn hoặc bằng	
JE	Nhảy nếu bằng (Jump if Equal)	ZF = 1
JZ	Nhảy nếu bằng (Jump if Zero)	
JNE	Nhảy nếu không bằng (Jump if Not Equal)	ZF = 0
JNZ	Nhảy nếu không bằng (Jump if Not Zero)	

2.5. Lệnh lặp

Bằng cách dùng các lệnh nhảy có thể tạo ra vòng lặp. Tuy nhiên, để viết chương trình tiện lợi và ngắn gọn, có thể dùng thêm các lệnh lặp như LOOP, LOOPZ,...

Cú pháp: LOOP <Label>

tự động giảm CX một đơn vị, sau đó kiểm tra xem CX có bằng 0, nếu không bằng thì nhảy đến nhãn <Label>

Cú pháp: LOOPZ <Label>

tự động giảm CX một đơn vị, sau đó kiểm tra xem CX có bằng 0 hoặc cờ ZF có bật không (ZF=1), nếu cả hai điều này không xảy ra thì nhảy đến nhãn <Label>

Ví dụ: Nhập mảng A gồm 10 ký tự

```
MOV SI, 0 ; chỉ số mảng
MOV CX, 10 ; số lần lặp
LAP: MOV AH, 1 ; nhập ký tự
      INT 21H
      MOV A[SI], AL
      INC SI
```

3. NỘI DUNG THỰC HÀNH

3.1. Cấu trúc Rẽ nhánh

- Chương trình sau đây nhận 1 ký tự. Nếu là ký tự HOA thì in ra màn hình "*Ky tu HOA*". Ngược lại in ra câu "*Ky tu thường*". (Mã ASCII của ký tự HOA \leq 'Z'). Soạn thảo và lưu với tên **BAI_3A.ASM**

```
DSEG SEGMENT
    tbao1 DB "Ky tu HOA.$"
    tbao2 DB "Ky tu thường.$"
DSEG ENDS
CSEG SEGMENT
    ASSUME CS: CSEG, DS: DSEG
start: mov ax, DSEG
       mov ds, ax
       mov ah, 01h
       int 21h
       cmp al, 'Z' ; so sánh với 'Z'
       ja nhan ; Nếu lớn hơn => ký tự thường
       mov ah, 09 ; Nếu không lớn hơn => ký tự HOA
       lea dx, tbao1 ; in "Ky tu HOA"
       int 21h
       jmp exit
nhan:  mov ah, 09 ; in "Ky tu thường"
       lea dx, tbao2
       int 21h
exit:  mov ah, 7
       int 21h
       mov ah, 4Ch ; trở về hệ điều hành
       int 21h
CSEG ENDS
END start
```

- Dịch và chạy CT ở những trường hợp khác nhau để xem kết quả trên màn hình.
- Vẽ lưu đồ điều khiển của chương trình.

- Tại sao cần phải có lệnh **JMP EXIT**? Nếu không có lệnh ấy thì chương trình thực hiện như thế nào? Chạy chương trình để kiểm chứng.
- Thay lệnh **JA NHAN** bằng lệnh **JNA NHAN**. Sửa chương trình sao cho kết quả không thay đổi.
- Khi ký tự nhập vào không phải là chữ cái thì kết quả in ra màn hình là gì? Tại sao?

3.2 Cấu trúc vòng lặp

- Xem chương trình in ra màn hình lần lượt các ký tự từ A đến Z được viết như sau. Hãy soạn thảo và đặt tên tập tin là **BAI_3B.ASM**.
- Dịch và chạy chương trình để xem kết quả trên màn hình.
- Vòng lặp trong chương trình bao gồm đoạn lệnh nào? Viết theo kiểu while do hay repeat ... until hay for? Vẽ lưu đồ chương trình.
- Sửa chương trình để in ra màn hình lần lượt các ký tự từ 'Z' đến 'A'.
- Tiếp tục sửa chương trình sao cho giữa các ký tự có 1 khoảng trống (Z YB A)
- Dùng lệnh LOOP để viết lại chương trình BAI_3B.ASM theo cấu trúc vòng lặp **for**.

```
CSEG SEGMENT
    ASSUME CS: CSEG
start:mov dl, 'A'      ; DL chứa ký tự đầu tiên 'A'
nhan:mov ah, 02h       ; in ký tự trong DL ra màn hình
      int 21h
      inc dl           ; DL chứa ký tự kế cần in
      cmp dl, 'Z'      ; So sánh DL với 'Z'
      jna nhan         ; Nếu <= 'Z' thì tiếp tục in
      mov ah, 08h      ; Nếu > 'Z' thì thoát (không in tiếp)
      int 21h
      mov ah, 4Ch
      int 21h
CSEG ENDS
      END start
```

4. BÀI TẬP KIỂM TRA:

4.1 Viết chương trình cho nhập 1 ký tự từ màn hình và xuất câu thông báo tương ứng sau:

- Nếu ký tự nhập là 'S' hay 's' thì in ra "Good morning!"
- Nếu ký tự nhập là 'T' hay 't' thì in ra "Good Afternoon!"
- Nếu ký tự nhập là 'C' hay 'c' thì in ra "Good everning!"

4.2 Viết lại chương trình BAI_3A.ASM sao cho chương trình có thể phân biệt được 3 loại ký tự nhập từ bàn phím: "Ký tự HOA", "ký tự thường" và "ký tự khác".

4.3 Viết chương trình nhập từ bàn phím 1 ký tự thường. Sau đó in ra màn hình lần lượt các ký tự từ ký tự nhận được đến 'z' sao cho giữa các ký tự có 1 khoảng trống.

4.4 Không dùng hàm 0Ah/21h, hãy dùng lệnh lặp để viết chương trình nhập vào 1 chuỗi ký tự. Sau khi nhập xong đếm xem chuỗi có bao nhiêu ký tự. In ra màn hình chuỗi nhận được và số ký tự có trong chuỗi.

Ví dụ: S = "Hello world !" ==> Số ký tự trong chuỗi là 13.

4.5 Viết chương trình cho phép nhập vào một chuỗi bất kỳ. Sau đó:

- Đổi tất cả ký tự thường thành ký tự hoa và in ra màn hình.
- Đổi tất cả ký tự hoa thành ký tự thường và in ra màn hình.

Ví dụ: S = 'weLcOme To AssEmblY'

In ra: welcome to assembly - WELCOME TO ASSEMBLY

4.6 Nhập vào 2 chuỗi số, đổi 2 chuỗi thành số, sau đó cộng hai số, đổi ra chuỗi và xuất chuỗi tổng.

Ví dụ: S1 = "123" ==> N1 = 123

S2 = "456" ==> N2 = 456

N = N1 + N2 = 123 + 456 = 579 ==> S = "579" (xuất S ra màn hình)

4.7 Nhập 2 số nguyên dương A, B. Tính A/B, A*B (không dùng lệnh DIV, MUL) và in ra màn hình kết quả.

Ví dụ: A=18, B=3

Tính A/B: 18 - 3 - 3 - 3 - 3 - 3 - 3 = 0, vậy A/B = 6 (tổng trừ B cho đến khi A = 0).

Tính A*B = 18 + 18 + 18 = 54

Bài 4: Nhập xuất số dạng BIN – HEX - DEC

1. Mục Tiêu:

- Nhập từ bàn phím số ở dạng nhị phân, thập lục phân và thập phân
- In lên màn hình các số ở dạng nhị phân, thập lục phân và thập phân

2. Kiến thức cần chuẩn bị:

- Kết quả của các bài thực hành trước
- Các lệnh xử lý chuỗi.

3. Nội dung thực hành:

3.1. Nhập xuất nhị phân:

Chương trình mẫu sau đây cho phép nhập số nhị phân 8 bit, sau đó in ra màn hình số nhị phân nhận được (BAI_6A.ASM):

```
inhuoi    MACRO      huoi
            MOV  AH, 9h
            LEA  DX, huoi
            INT  21h
            ENDM

DSEG SEGMENT
    msg1 DB "Hay nhap so nhi phan 8 bit: $"
    msg2 DB "So nhi phan da nhap la: $"
    xdong DB 10, 13, '$'
    sobin DB ?      ; lưu trữ số nhị phân nhận được
DSEG ENDS

CSEG SEGMENT
    ASSUME CS:CSEG, DS:DSEG
begin: MOV AX, DSEG
        MOV DS, AX
        inhuoi msg1
        CALL bin_in
        MOV sobin, BL
        inhuoi xdong
        inhuoi msg2
        MOV BL, sobin
        CALL bin_out
        MOV AH, 01
        INT 21h
        MOV AH, 4Ch      ; thoát khỏi chương trình
        INT 21h

bin_in    PROC
            MOV BL, 0      ; Xóa BL
            MOV CX, 8      ; nhập đủ 8 bit thì dừng
```

```

        nhap:MOV  AH, 01h    ; Hàm nhập ký tự
            INT  21h
            CMP  AL, 0Dh    ; nếu là phím Enter thì thôi nhập
            JZ   exit       ; không phải Enter thì đổi sang bit
            SHL  BL, 1      ; Dịch trái BL 1 bit
            SUB  AL, 30h    ; Ký số - 30h = số
            ADD  BL, AL     ; Chuyển bit từ AL sang BL lưu trữ
            LOOP nhap
        exit:RET
bin_in   ENDP
bin_out  PROC
        MOV  CX, 8        ; Xuất 8 bit trong BL ra M.Hình
        xuat:MOV DL, 0
            SHL  BL, 1     ; CF chứa MSB, xuất ra màn hình
            RCL  DL, 1     ; đưa CF vào LSB của DL
            ADD  DL, 30h    ; Số + 30h = Ký số
            MOV  AH, 02h   ; In ra màn hình
            INT  21h
            LOOP xuat
            RET
bin_out  ENDP
CSEG    ENDS
        END begin
    
```

- Soạn thảo, Biên dịch và cho chạy file BAI_6A.ASM để kiểm tra kết quả.
- Sửa chương trình trên thành BAI_6A1.ASM sao cho có thể nhập và xuất số nhị phân 16 bit.
- Viết lại chương trình trên để nhập 1 ký tự từ bàn phím, sau đó in ra màn hình mã ASCII của ký tự nhận được ở dạng nhị phân.

3.2. Nhập xuất thập lục phân:

Chương trình sau đây cho phép nhập 1 ký tự từ bàn phím, sau đó in ra màn hình mã ASCII của ký tự nhận được ở dạng thập lục phân

- Soạn thảo, Biên dịch và cho chạy file BAI_6B.ASM để kiểm tra kết quả.
- Viết lại chương trình trên để nhập 2 số thập lục phân 8 bit A và B, sau đó in ra màn hình kết A + B ở dạng thập lục phân.

```

inchuoi  MACRO    chuoi
            MOV  AH, 9h
            LEA  DX, chuoi
            INT  21h
        ENDM
DSEG     SEGMENT
        msg1 DB "Hay nhap 1 ky tu: $"
        msg2 DB "Ma ASCII o dang Hex: $"
    
```

```
        xdong DB 10, 13, '$'
        kytu  DB ?
DSEG ENDS
CSEG SEGMENT
        ASSUME CS:CSEG, DS:DSEG
begin: MOV AX, DSEG
        MOV DS, AX
        inchuoi msg1
        MOV AH, 01h
        INT 21h
        MOV kytu, AL ; cất ký tự nhận được
        inchuoi xdong
        inchuoi msg2
        MOV BH, kytu ; Ký tự cần in
        CALL hex_out
        MOV AH, 02 ; in ra ký tự h sau số Hex
        MOV DL, 'h'
        INT 21h
        MOV AH, 01
        INT 21h
        MOV AH, 4Ch ; thoát khỏi chương trình
        INT 21h
hex_out PROC
        MOV CX, 4
        xuất: PUSH CX
        MOV CL, 4
        MOV DL, BH
        SHR DL, CL
        CMP DL, 09h
        JA kytu
        ADD DL, 30h ; Đổi thành ký số '0'-'9'
        JMP inra
        kytu: ADD DL, 37h ; Đổi thành ký tự 'A'-'F'
        inra: MOV AH, 02h ; In ra màn hình ký tự đã đổi
        INT 21h
        SHL BX, CL ; Quay trái BX 4 bit
        POP CX
        LOOP xuất
        RET
hex_out ENDP
CSEG ENDS
        END begin
```

3.3. Xuất số thập phân nguyên dương:

Chương trình sau đây cho phép nhập 1 ký tự từ bàn phím, sau đó in ra màn hình mã ASCII của ký tự nhận được ở dạng thập phân.

- Soạn thảo, Biên dịch và cho chạy file BAI_6C.ASM để kiểm tra kết quả.
- Đọc thủ tục DEC_OUT để tìm hiểu giải thuật xuất giá trị trong AX ra màn hình ở dạng thập phân. Từ đó đưa ra giải thuật nhập số thập phân từ bàn phím.
- Viết lại chương trình trên để nhập 2 số thập phân A và B có 2 chữ số, sau đó in ra màn hình kết quả A + B ở dạng thập phân.
- Nhập xuất số thập phân ÂM như thế nào?

```
inhuoi  MACRO      huoi
        MOV  AH, 9h
        LEA  DX, huoi
        INT  21h
        ENDM
DSEG SEGMENT
    msg1 DB "Hay nhap 1 ky tu: $"
    msg2 DB "Ma ASCII o dang Dec: $"
    xdong DB 10, 13, '$'
    kytu  DB ?
DSEG ENDS
CSEG SEGMENT
    ASSUME CS:CSEG, DS:DSEG
begin: MOV AX, DSEG
        MOV DS, AX
        inhuoi  msg1
        MOV AH, 01h
        INT 21h
        MOV kytu, AL ; cất ký tự nhận được
        inhuoi  xdong
        inhuoi  msg2
        XOR AX, AX
        MOV AL, kytu ; Ký tự cần in
        CALL dec_out
        MOV AH, 01
        INT 21h
        MOV AH, 4Ch ; thoát khỏi chương trình
        INT 21h
dec_out PROC
        XOR CX,CX ; CX đếm số chữ số thập phân
        MOV BX,10
    chia10: XOR DX,DX
            DIV BX ; DX:AX÷BX => AX: Thương, DX: số dư
            PUSH DX ; Cất số dư vào stack
            INC CX
            CMP AX, 0
            JNZ chia10 ; nếu AX>0 thì chia tiếp cho 10
    inra:  MOV AH,2 ; in ra màn hình
            POP DX ; lấy chữ số thập phân
```

```
ADD DL, 30h    ; đổi thành ký số
INT 21h
LOOP inra
RET
dec_out ENDP
CSEG ENDS
END begin
```

4. Bài tập kiểm tra:

4.1. Viết chương trình nhập 2 số nhị phân 16 bit A và B. Sau đó in ra màn hình các kết quả ở dạng nhị phân: A + B, A – B, A and B, A or B.

Ví dụ: Nhập số nhị phân A: 10101010

Nhập số nhị phân B: 01010101

A + B = 11111111

A – B = 01010101

A and B = 00000000

A or B = 11111111

4.2. Viết chương trình nhập 1 ký tự từ bàn phím, sau đó in ra màn hình mã ASCII của ký tự nhận được ở dạng thập lục phân, thập phân và nhị phân.

Ví dụ: Nhập 1 ký tự: A

Mã ASCII dạng Hex: 41h

Mã ASCII dạng Dec: 65

Mã ASCII dạng Bin: 01000001b

4.3. Viết lại chương trình bài 4.1 nhưng 2 số A và B được nhập theo dạng thập lục phân. Các kết quả được in ra màn hình ở dạng nhị phân.

4.4. Viết lại chương trình bài 4.1 nhưng 2 số A và B được nhập theo dạng thập phân. In các kết quả ở dạng thập phân: A + B, A – B.

4.5. Viết chương trình tính giai thừa n! Với n là số nguyên dương nhập từ bàn phím. In kết quả ra màn hình ở dạng thập phân. Cho biết, khả năng của 8086 tính được n lớn nhất là bao nhiêu?

Bài 5: Xử Lý Tập Tin

1. Mục Tiêu:

Viết được các chương trình xử lý tập tin như tạo tập tin, xóa tập tin, ghi tập tin, đọc nội dung của tập tin.

2. Kiến thức cần chuẩn bị:

- Kết quả của các bài thí nghiệm 1, 2, 3 và 4.
- Các hàm 3dh, 3ch, 3eh, 3fh, 40h, 41h, 42h và 56h của INT 21h để xử lý tập tin.
- Các hàm 01, 02h, 06h, 08h, 09h, 0Ah của INT 21h và các lệnh của CPU 8086

3. Nội dung thực hành:

3.1. Tạo tập tin mới:

Soạn thảo như đoạn chương trình mẫu phía dưới và lưu với tên là BAI_5A.ASM.

```
DSEG SEGMENT
    tenfile db "d:\tt_asm\data.txt",0
    thefile dw ?
DSEG ENDS
CSEG SEGMENT
    ASSUME cs:cseg, ds:dseg
begin: mov ax, dseg
        mov ds, ax
        mov ah, 3ch          ; tao tap tin moi
        lea dx, tenfile
        mov cx, 0            ; thuoc tinh tap tin
        int 21h
        mov thefile, ax      ; cat the file
        mov ah, 3eh          ; dong tap tin
        mov bx, thefile
        int 21h
        mov ah, 4ch          ; thoat ve Dos
        int 21h
CSEG ENDS
    END begin
```

- Biên dịch và cho chạy file BAI_5A.ASM để kiểm tra và xem kết quả. Gợi ý: Thư mục TT_ASM phải có sẵn trong ổ đĩa. Để biết chương trình chạy đúng hay sai, vào thư mục TT_ASM để xem có tập tin Data.txt hay không, nếu có thì OK.
- Tại sao thẻ file phải được khai báo như dạng *thefile DW ?*

- Tại sao trong trường hợp này, chúng ta không phải dùng hàm 08h của int 21h đứng trước hàm 4ch của int 21h ?
- Tại sao phải cất thẻ file. Nếu chúng ta không cần đóng file thì chúng ta có cần cất thẻ file hay không ?
- Trong đoạn chương trình mẫu trên có cần thiết phải đóng tập tin hay không ? Có thể bỏ biến thefile trong đoạn chương trình mẫu trên không ?. Khi đó chúng ta phải dùng các lệnh gì để thay thế điều đó. Nếu có thay đổi, hãy biên dịch và cho chạy chương trình để kiểm chứng lại kết quả.
- Hãy sửa đổi file BAI_5A.ASM và lưu với tên BAI_5A1.ASM để có thể thực hiện được yêu cầu sau: tạo một tập tin mới, tên tập tin được nhập từ bàn phím. Gợi ý: dùng hàm 0ah của int 21h để nhập vào tên file, chú ý cuối chuỗi chứa tên file phải có zero, nhưng khi dùng hàm 0ah thì chúng ta không thể nào nhập zero vào cuối chuỗi được, nếu ta nhập ký tự '0' vào thì đó là mã ascii của ký tự '0' chứ không phải là zero (con số 0). Do đó, để thực hiện được điều này chúng ta hãy dùng giải thuật đưa 0 về cuối chuỗi như sau:

```
xor cx, cx      ; dua zero ve cuoi chuoai
mov cl, len     ; khai bao bien de dung ham 0ah cua int 21h dang
lea bx, tenfile; max db 250      ; so ky tu toi da duoc nhap
mov dl, 0       ; len db ?      ; chieu dai chuoai da nhap
mov [bx], dl    ; tenfile db 250 dup(?); chua noi dung
duoc nhap
```

3.2. Ghi nội dung của biến string1 vào một tập tin mới có tên trong thư mục và ổ đĩa với đường dẫn như sau: “D:\TT_ASM\DATA.TXT”.

Soạn thảo như đoạn chương trình mẫu phía dưới và lưu với tên là BAI_5B.ASM.

- Biên dịch và cho chạy file BAI_5B.ASM để kiểm tra và xem kết quả. (vào D:\TT_ASM để xem tập tin DATA.TXT có trong đó hay chưa và có nội dung hay chưa ?, nếu có là OK.)
- Xem xét đoạn chương trình mẫu, hãy đưa ra giải thuật ghi nội dung của vùng dữ liệu vào một tập tin vừa tạo.
- Lệnh *len db \$ - string1* được dùng để làm gì ?.
- Lệnh *XOR CX, CX* có ý nghĩa gì? Sau khi thực hiện xong lệnh này, thanh ghi CX có giá trị bằng bao nhiêu? Có thể thay thế nó bằng lệnh nào khác được không ?
- Tại sao dùng lệnh *MOV CL, LEN* mà không dùng *MOV CX, LEN* hay *MOV CH, LEN* ? Khi dùng lệnh *MOV CX, LEN* thì cần phải thay đổi khai báo biến len như thế nào ? nếu không thay đổi thì sẽ có vấn đề gì xảy ra hay không ? Hãy thay đổi, biên dịch và chạy chương trình để kiểm chứng lại kết quả so với chương trình mẫu.
- Hãy sửa đổi file BAI_5B.ASM và lưu với tên BAI_5B1.ASM để có thể thực hiện được yêu cầu sau: nhập từ một chuỗi ký tự bất kỳ, sau đó lưu vào tập tin có tên là “d:\tt_asm\solieu.txt”. Gợi ý: dùng hàm 0Ah của int 21h để nhập vào một chuỗi ký tự, sau đó áp dụng toàn bộ giải thuật của BAI_5B.ASM.
- Hãy sửa đổi file BAI_5B1.ASM và lưu với tên BAI_5B2.ASM để có thể thực hiện được yêu cầu sau: tạo tên tập tin mới, tên tập tin được nhập từ bàn phím.

Sau đó nhập vào một chuỗi ký tự bất kỳ và lưu chuỗi ký tự đã nhập vào tập tin vừa tạo. Gợi ý: xem lại BAI_5A1.ASM để lấy lại giải thuật nhập vào tên file và đưa zero (con số 0) về cuối chuỗi đối với trường hợp tên file được nhập từ bàn phím và các vấn đề còn lại thì xem lại file BAI_5B1.ASM.

- Hãy sửa đổi file BAI_5B2.ASM và lưu với tên BAI_5B3.ASM để có thể thực hiện được yêu cầu sau: tạo tên tập tin mới, tên tập tin được nhập từ bàn phím. Sau đó nhập vào một chuỗi ký tự thường, sau đó đổi hoa ký tự đầu của mỗi từ và lưu chuỗi ký tự đã thay đổi vào tập tin vừa tạo. Gợi ý: xem lại các bài tập đã làm trong bài 5 xử lý ký tự và các file BAI_5B2.ASM.

```
dseg segment
    string1 db "Chao em co gai Lam Hong"
    len db $ - string1
    tenfile db "d:\tt_asm\data.txt",0
    thefile dw ?
dseg ends

cseg segment
    assume cs:cseg, ds:dseg
begin: mov ax, dseg
    mov ds, ax
    mov ah, 3ch          ; tao tap tin moi
    lea dx, tenfile
    mov cx, 0            ; tap tin co thuoc tinh binh thuong
    int 21h
    mov thefile, ax      ; cat the file
    mov ah, 40h          ; ghi file
    mov bx, thefile
    xor cx, cx
    mov cl, len
    lea dx, string1
    int 21h
    mov ah, 3eh          ; dong tap tin
    mov bx, thefile
    int 21h
    mov ah, 4ch          ; thoat ve Dos
    int 21h
cseg ends
    end begin
```

3.3. Đọc nội dung của tập tin (đã tồn tại trên đĩa và có nội dung). Hiển thị nội dung của tập tin lên màn hình.

Soạn thảo như đoạn chương trình mẫu phía dưới và lưu với tên là BAI_5C.ASM.

- Biên dịch và cho chạy file BAI_5C.ASM để kiểm tra và xem kết quả.

- Xem lại đoạn chương trình mẫu, hãy đưa ra giải thuật đọc nội dung của tập tin và hiển thị nội dung đó ra màn hình.
- Hãy thử thay đổi thuộc tính tập tin trong lệnh `mov al, 2` lần lượt thành các giá trị khác như 0, 1, 3 hoặc 4. Biện dịch và cho chạy chương trình để xem kết quả. Có nhận xét gì về các giá trị này ?.
- Thẻ file có vai trò như thế nào trong xử lý tập tin.
- Nếu số byte cần đọc (giá trị chứa trong thanh ghi `cx`) lớn hơn kích thước thật sự của tập tin thì có gây ra lỗi gì hay không ?. Sau khi đọc nội dung của tập tin vào vùng đệm bằng hàm `3fh` của `INT 21h`, thanh ghi `ax` sẽ có giá trị thay đổi hay không và nó chứa (giá trị) gì ?. Làm sao xác định được khi nào đọc xong nội dung thành tập tin ?. Hãy thử đưa ra hướng giải quyết.
- Hãy sửa đổi file `BAI_5C.ASM` và lưu với tên `BAI_5C1.ASM` để có thể thực hiện được yêu cầu sau: đọc nội dung của một tập tin và hiển thị nội dung đó lên màn hình. Tên tập tin được nhập từ bàn phím. Gợi ý: sử dụng lại giải thuật nhập tên file từ bàn phím và đưa zero về cuối chuỗi và các bài có liên quan trong bài 6.
- Hãy sửa đổi file `BAI_5C1.ASM` và lưu với tên `BAI_5C2.ASM` để có thể thực hiện được yêu cầu sau: copy nội dung của một tập tin bất kỳ sau đó paste sang một vị trí khác. Gợi ý: các giải thuật nhập tên file từ bàn phím thì có sẵn, giải thuật copy và paste như sau: trước hết phải mở tập tin đã có bằng hàm `3dh`, chúng ta định nghĩa sẵn đường dẫn chứa tên file cần mở, đọc nội dung của tập tin vào vùng đệm bằng hàm `3fh`, nhớ cất thẻ file; tạo tên tập tin mới bằng hàm `3ch`, chúng ta định nghĩa sẵn đường dẫn chứa tên file cần tạo, nhớ cất thẻ file; ghi nội dung của vùng đệm vào tập tin mới vừa tạo bằng hàm `40h`; đóng hai tập tin lại bằng hàm `3eh`, tất cả các hàm này đều của `int 21h`.
- Hãy sửa đổi file `BAI_5C2.ASM` và lưu với tên `BAI_5C3.ASM` để có thể thực hiện được yêu cầu sau: copy nội dung của một tập tin bất kỳ sau đó paste sang một vị trí khác. Tên tập tin được copy và tập tin sau khi paste đều nhập từ bàn phím. Gợi ý: hoàn toàn tương tự như `BAI_5C2.ASM`, nhưng chúng ta cần tận dụng lại giải thuật nhập tên file từ bàn phím từ `BAI_5B3.ASM`.
- Hãy sửa đổi file `BAI_5C3.ASM` và lưu với tên `BAI_5C4.ASM` để có thể thực hiện được yêu cầu sau: save as nội dung của một tập tin. Tên tập tin được copy và save as đều nhập từ bàn phím. Gợi ý: bài này giống tương tự như `BAI_5C3.ASM`.
- Hãy sửa đổi file `BAI_5C3.ASM` và lưu với tên `BAI_5C4.ASM` để có thể thực hiện được yêu cầu sau: đọc nội dung của tập tin và sau đó nhập một chuỗi ký tự bất kỳ và ghi tiếp theo sau nội dung của tập tin vừa mở. Tên tập tin nhập từ bàn phím. Gợi ý: xem lại `BAI_5C1.ASM`, `BAI_5B1.ASM`, `BAI_5B2.ASM` và hàm `42h` của `int 21h` (dời vị trí con trỏ tập tin).
- Hãy sửa đổi file `BAI_5C4.ASM` và lưu với tên `BAI_5C5.ASM` để có thể thực hiện được yêu cầu sau: mã hóa nội dung của tập tin. Tên tập tin cần mã hóa được nhập từ bàn phím. Gợi ý: nhập vào tên file cần mã hóa, mở một file đã có bằng hàm `3dh`; đọc nội dung của tập tin vào vùng đệm bằng hàm `3fh`; mã hóa vùng đệm đọc được bằng một trong các phép toán cộng, trừ, nhân, chia, and, or, not ..., vừa mã hóa vừa lưu nội dung trở lại vùng đệm; dời vị trí con trỏ tập tin

về đầu tập tin bằng hàm 42h; sau đó ghi nội dung của vùng đệm trở lại tập tin ban đầu thông qua thẻ file của nó. Tất cả các hàm sử dụng ở đây đều của int 21h.

- Hãy sửa đổi file BAI_5C5.ASM và lưu với tên BAI_5C6.ASM để có thể thực hiện được yêu cầu sau: giải mã nội dung của tập tin đã mã hóa. Tên tập tin cần giải mã được nhập từ bàn phím. Gợi ý: giải mã là trường hợp ngược lại của mã hóa, nếu mã hóa theo phương thức nào thì giải mã phải làm ngược lại phương thức mã hoá đó.

```
dseg segment
    tenfile db "d:\tt_asm\data.txt",0
    thefile dw ?
    buffer db 251 dup ('$')
dseg ends
cseg segment
    assume cs:cseg, ds:dseg
begin: mov ax, dseg
        mov ds, ax
        mov ah, 3dh          ; mở tập tin đã có
        lea dx, tenfile
        mov al, 2            ; thuộc tính tập tin
        int 21h
        mov thefile, ax      ; cất thẻ file
        mov ah, 3fh          ; đọc nội dung file vào vùng đệm
        mov bx, thefile
        lea dx, buffer
        mov cx, 250          ; số byte cần đọc từ file đã mở
        int 21h
        mov ah, 3eh          ; đóng tập tin
        mov bx, thefile
        int 21h
        mov ah, 09h          ; in nội dung của file ra màn hình
        lea dx, buffer
        int 21h
        mov ah, 08h          ; dùng màn hình để xem kết quả
        int 21h
        mov ah, 4ch          ; thoát về Dos
        int 21h
cseg ends
        end begin
```

3.4. Xóa tập tin.

Soạn thảo như đoạn chương trình mẫu phía dưới và lưu với tên là BAI_5D.ASM.

- Biên dịch và cho chạy file BAI_5D.ASM để kiểm tra và xem kết quả.
- Hãy sửa đổi file BAI_5D.ASM và lưu với tên BAI_5D1.ASM để có thể thực hiện được yêu cầu sau: xóa tên một tập tin. Tên tập tin cần được nhập từ bàn phím. Gợi ý: xem lại các bài trước để lấy giải thuật nhập tên file từ bàn phím.

```
dseg segment
    tenfile db "d:\tt_asm\data.txt",0
dseg ends
cseg segment
    assume cs:cseg, ds:dseg
begin: mov ax, dseg
        mov ds, ax
        mov ah,41h          ; xoa tap tin da co
        lea dx, tenfile
        int 21h
        mov ah, 4ch          ; thoat ve Dos
        int 21h
cseg ends
end begin
```

3.5. Đổi tên tập tin cũ thành một tập tin mới trong cùng thư mục

Soạn thảo như đoạn chương trình mẫu phía dưới và lưu với tên là BAI_5E.ASM.

- Biên dịch và cho chạy file BAI_5E.ASM để kiểm tra và xem kết quả.
- Hãy tạo một thư mục con có tên là baitap nằm trong thư mục tt_asm. Sửa đổi lệnh oldfile db "d:\tt_asm\data.txt",0 và newfile db "tt_asm\solieu.txt",0 lại thành oldfile db "d:\tt_asm\solieu.txt",0 và newfile db "tt_asm\baitap\data.txt",0. Biên dịch lại và cho chạy chương trình để xem xét kết quả. Có nhận xét gì về kết quả nhận được.
- Hãy sửa đổi file BAI_5E.ASM và lưu với tên BAI_5E1.ASM để có thể thực hiện được yêu cầu sau: đổi tên một tập tin. Tên tập tin cũ và mới được nhập từ bàn phím. Gợi ý: xem lại các bài tập trước để lấy giải thuật nhập tên file từ bàn phím

```
dseg segment
    oldfile db "d:\tt_asm\data.txt",0
    newfile db "d:\tt_asm\solieu.txt",0
dseg ends
cseg segment
    assume cs:cseg, ds:dseg, es: dseg
begin: mov ax, dseg
        mov ds, ax
        mov es, ax
        mov ah,56h          ; rename/remove tên file cu thanh moi
        lea dx, oldfile
        lea di, newfile
        int 21h
        mov ah, 4ch          ; thoat ve Dos
        int 21h
cseg ends
end begin
```

4. Bài tập kiểm tra:

- 4.1. Viết chương trình sử dụng hàm 41h/ INT 21h để xóa tập tin trên đĩa. Tên tập tin cần xóa được nhập từ bàn phím khi thực hiện chương trình.
- 4.2. Viết chương trình nhập 1 chuỗi từ bàn phím, sau đó ghép chuỗi nhận được vào cuối của nội dung tập tin có trên đĩa. Tên tập tin nhập từ bàn phím khi chạy chương trình.
- 4.3. Viết chương trình nhập 1 chuỗi từ bàn phím, sau đó chèn chuỗi nhận được vào đầu của nội dung tập tin có trên đĩa. Tên tập tin nhập từ bàn phím khi chạy chương trình.
- 4.4. Viết chương trình ghép nội dung 2 tập tin có sẵn trên đĩa thành 1 tập tin mới. Tên của các tập tin được nhập từ bàn phím khi chạy chương trình.
- 4.5. Viết chương trình đọc nội dung tập tin trên đĩa, sau đó đổi tất cả ký tự HOA thành ký tự thường và lưu lại vào tập tin đó. Tên tập tin phải được nhập từ bàn phím

Bài 6: Xử Lý Chuỗi Ký Tự

1. Mục Tiêu:

- Viết được các chương trình xử lý chuỗi ký tự bằng các lệnh xử lý chuỗi

2. Kiến thức cần chuẩn bị:

- Bảng mã ASCII.
- Kết quả của các bài thí nghiệm trước
- Các hàm 01h, 02h, 06h, 08h, 09h, 0Ah của INT 21h và các lệnh xử lý chuỗi như MOVSB/W, SCASB/W, STOSB/W, CMPSB/W....

3. Nội dung thực hành:

3.1. So sánh hai chuỗi oldpass và newpass. Nếu hai chuỗi này giống nhau thì kết luận giống nhau và ngược lại.

Soạn thảo như đoạn chương trình trên và lưu với tên là BAI_6A.ASM.

- Biên dịch và cho chạy file BAI_6A.ASM để kiểm tra và xem kết quả.
- Trong macro writeln, các lệnh nào có chức năng xuống dòng sau khi in xong chuỗi ký tự.
- Mục đích của việc khai báo LOCAL bien1 trong macro là gì?
- Hãy cho biết địa chỉ của DS và ES có giống nhau hay không? Điều này được thể hiện qua các câu lệnh nào trong đoạn chương trình mẫu? Tại sao người ta không khai báo DS và ES trên các phân đoạn khác nhau ?
- Tiền tố REPE trong đoạn chương trình mẫu trên có ý nghĩa như thế nào?
- Ta có thể thay đổi lệnh REPE CMPSB thành một nhóm lệnh khác được không? Nếu được hãy thay đổi nó, biên dịch và chạy chương trình để kiểm chứng.
- Thử thay đổi nội dung ở oldpass và newpass sao cho chúng giống nhau. Biên dịch và chạy chương trình xem kết quả, sau đó hãy giải thích cơ chế làm việc của đoạn lệnh từ lệnh CLD cho đến lệnh REPE CMPSB.
- Giả sử, người ta muốn thay thế lệnh cmpsb thành lệnh cmpsw, các bạn có cần sửa đổi các lệnh nào trong chương trình hay không ? Tại sao ?. Biên dịch và chạy chương trình để kiểm chứng.
- Lệnh jmp thoát trong đoạn chương trình trên có nhiệm vụ gì ?. Thử bỏ lệnh jmp thoát sau đó biên dịch và chạy chương trình xem kết quả.
- Hãy sửa đổi file BAI_6A.ASM và lưu với tên BAI_6A1.ASM để có thể thực hiện được nhiệm vụ sau: nhập vào một chuỗi ký tự có tối đa 10 ký tự, trong lúc nhập chỉ hiện thị ra ký tự “*”. Khi đã nhập đủ 10 ký tự hoặc khi gặp phím ESC thì sẽ in ra các ký tự đã nhập ra màn hình. Gợi ý: dùng hàm 08h, 02h hoặc 09h của int 21h, lệnh loop, cmp, ... Cần phải khai báo dùng đệm để lưu các ký tự đã nhập.

```
writeln macro bien1
LOCAL bien1
    mov ah,09
    lea dx, bien1
    int 21h
    mov ah,02h
    mov dl, 0ah
    int 21h
    mov dl, 0dh
    int 21h
endm
dseg segment
    tbao db "Chương trình so sánh oldpass và newpass$"
    oldpass db "0123456789"
    newpass db "1234567890"
    tbao1 db "Haichuoi giống nhau $"
    tbao2 db "Haichuoi không giống nhau $"
dseg ends
cseg segment
    assume cs:cseg, ds:dseg, es: dseg
begin: mov ax, dseg
    mov ds, ax
    mov es, ax
    writeln tbao
    cld                ; chonchieu xu ly chuoi
    mov cx, 10        ; so ky tu/so byte can so sanh
    lea si, oldpass; (DS:SI)--> dia chi cua chuoi nguon
    lea di, newpass; (ES:DI)--> dia chi cua chuoi dich
    repe cmpsb        ; so sanh tung ky tu/byte
    je intb1
    writeln tbao2
    jmp thoat
intb1:  writeln tbao2
thoat:
    mov ah,08h
    int 21h
    mov ah, 4ch
    int 21h
cseg ends
end begin
```

- Hãy sửa đổi file BAI_6A1.ASM, kết hợp với file BAI_6A.ASM (chương trình mẫu) và lưu với tên BAI_6A2.ASM để có thể thực hiện được nhiệm vụ sau: nhập vào một chuỗi ký tự có 10 ký tự, trong lúc nhập chỉ hiện thị ra ký tự “*”. Sau đó so sánh với một oldpass có nội dung tùy ý (nhưng chỉ có độ dài là 10 ký tự mà do

chúng ta gán trước, ví dụ như oldpass db “1234567890”). Nếu 10 ký tự vừa nhập có nội dung giống oldpass thì in ra câu thông báo “Ban đã nhập đúng rồi” và thoát, ngược lại thì in ra câu thông báo “Ban đã nhập sai rồi và vui lòng nhập lại” và quay trở lại nhập cho đến khi nào đúng mới thoát. Gợi ý: dùng hàm 08h, 02h, 09h của int 21h và các lệnh loop, cmpsb,....

3.2. Di chuyển 33 bytes từ nội dung của string1 sang string2, sau đó in nội dung của string2 ra màn hình.

Soạn thảo như đoạn chương trình mẫu và lưu với tên là BAI_6B.ASM.

- Biên dịch và cho chạy file BAI_6B.ASM để kiểm tra và xem kết quả.

```
dseg segment
    string1 db "Khong co gi quy hon doc lap tu do"
    string2 db 34 dup('$')
dseg ends
cseg segment
    assume cs:cseg, ds:dseg, es: dseg
begin: mov ax, dseg
        mov ds, ax
        mov es, ax
        cld
        ; chon chieu xu ly chuoi
        mov cx, 33
        ; so ky tu/so byte can di chuyen
        lea si, string1
        ; (DS:SI)--> dia chi cua chuoi nguon
        lea di, string2
        ; (ES:DI)--> dia chi cua chuoi dich
        rep movsb
        ; di chuyen tung byte
        mov ah, 09h
        lea dx, string2
        int 21h
        mov ah, 08h
        ; dung man hinh de xem ket qua
        int 21h
        mov ah, 4ch
        ; thoat ve Dos
        int 21h
cseg ends
end begin
```

- Tại sao chỉ di chuyển 33 ký tự/byte mà lại khai báo biến *string2 db 34 dup('\$')*. Thử thay 34 thành 33 và tiến hành biên dịch, chạy chương trình để xem kết quả. Có nhận xét gì về vấn đề này không ?. Giải thích ý nghĩa của việc khai báo này.
- Thay lệnh CLD trong đoạn chương trình mẫu thành STD. Biên dịch và cho chạy chương trình để kiểm chứng kết quả. Cho nhận xét về kết quả nhận được.
- Nếu thay lệnh movsb thành MOVSW thì chúng ta có phải thay đổi giá trị nào trong đoạn chương trình mẫu trên không ?. Nếu có thay đổi, hãy biên dịch và chạy chương trình để kiểm chứng lại kết quả.

- Có thể thay thế lệnh REP MOVSB bởi một số lệnh khác hay không? Nếu được thì hãy thay đổi và sau đó biên dịch, cho chạy chương trình để kiểm chứng lại kết quả.
- Giả sử ta có nội dung của một biến string1 có tổng số byte >256 byte thì lúc đó chúng ta phải khai báo lại các biến này như thế nào ?. Lúc này có khó khăn gì xảy ra không ?. Hãy thử sửa lại, sau đó biên dịch và chạy chương trình để xem kết quả.
- Có cách nào xác định chiều dài của một biến bất kỳ hay không ?. Hãy cho biết các cú pháp của lệnh có thể thực hiện được yêu cầu này. Sau đó thử áp dụng để xác định chiều dài của biến string1.

3.3. Tìm ký tự “A” có trong một chuỗi ký tự bất kỳ, nếu có thì in ra câu thông báo là có ký tự “A” trong chuỗi ký tự và ngược lại.

Soạn thảo như đoạn chương trình mẫu và lưu với tên là BAI_6C.ASM.

```
write macro bien1
    mov ah, 09h
    lea dx, bien1
    int 21h
endm
dseg segment
    string1 db "NGAC NHIEN CHUA ?"
    tb1 db "co ky tu A trong chuoi string1 $"
    tb2 db "khong ky tu A trong chuoi string1 $"
dseg ends
cseg segment
    assume cs:cseg, ds:dseg, es: dseg
begin: mov ax, dseg
    mov ds, ax
    mov es, ax
    cld                ; chonchieu xu ly chuoi
    mov cx, 17         ; so ky tu can tim
    mov al, 'A'        ; tim kien ky tu A trong string1
    lea di, string1    ; (ES:DI)--> dia chi cua chuoi dich
    repne scasb        ; lap lai viec tim kien ky tu cho den
    jne intb2          ; khi gap duoc hoac den het chuoi
    write tb1
    jmp thoat
intb2:  write tb2
thoat:  mov ah,08h      ; dung man hinh de xem ket qua
        int 21h
        mov ah, 4ch     ; thoat ve Dos
        int 21h
cseg ends
end begin
```

- Biên dịch và cho chạy file BAI_6C.ASM để kiểm tra và xem kết quả.

- Thay lệnh MOV AL, 'A' thành MOV AL, 'B', sau đó biên dịch và chạy chương trình để xem kết quả.
- Giải thích nhiệm vụ của các lệnh từ CLD cho đến JNE INTB2. Có thể thay thế lệnh REPNE SCASB thành các lệnh khác được không ?. Nếu được hãy thay thế chúng, biên dịch và chạy chương trình để kiểm chứng lại kết quả.




Phụ lục

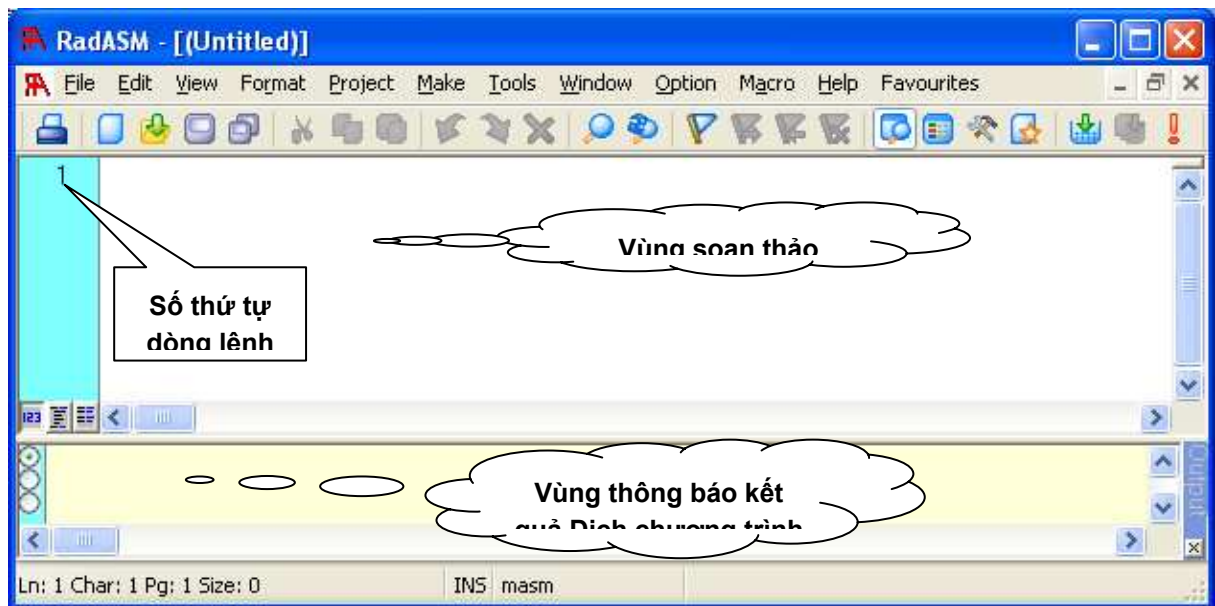
Môi trường phát triển hợp ngữ RadASM

RadASM là môi trường phát triển Hợp ngữ, được xây dựng để kết hợp những hợp ngữ khác nhau như MASM, TASM, HASM và chạy trên môi trường Windows. Mỗi hợp ngữ khác nhau khi kết hợp vào RadASM phải được cấu hình khác nhau. Vì mục đích cung cấp công cụ cho sinh viên sử dụng đơn giản, nên RadASM đã được cấu hình phù hợp với hợp ngữ MASM for DOS. Vì thế trong quá trình sử dụng, sinh viên không cần phải cấu hình gì thêm.

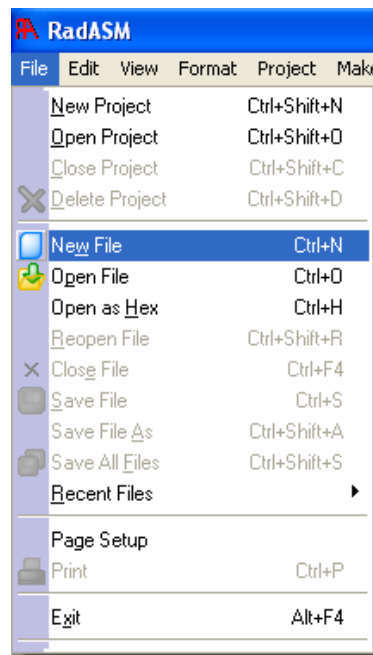
1. Khởi động RadASM

Nhấp đúp biểu tượng  trên desktop thì màn hình làm việc của RadASM xuất hiện như hình 1. Màn hình làm việc của RadASM chia thành 2 vùng:

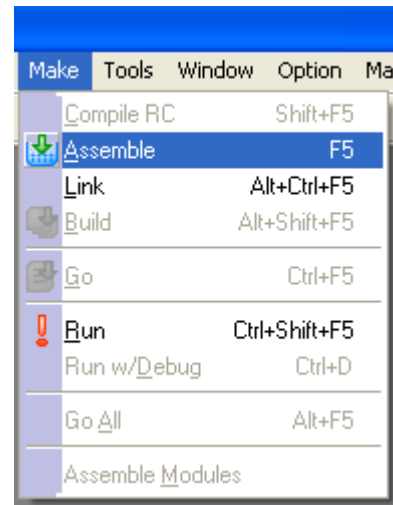
- Vùng soạn thảo dùng để soạn thảo chương trình nguồn ASM
- Vùng thông báo kết quả: Khi dịch chương trình, những kết quả hay lỗi sẽ xuất hiện tại vùng này



Hình 1: Màn hình làm việc của RadASM



Hình 2: Menu FILE



Hình 2: Menu MAKE

Các menu thường dùng là **FILE** (hình 2) và **MAKE** (hình 3), các lệnh thường dùng:

- **New File (Ctrl+N)**: Mở file mới để soạn thảo chương trình mới
- **Open File (Ctrl+O)**: Mở file đã lưu sẵn trên đĩa
- **Save File (Ctrl+S)**: Lưu file thành tên (*nhớ đặt tên file không có khoảng trắng, phần mở rộng ASM được thêm vào tự động*)
- **Assemble (F5)**: Hợp dịch file nguồn (ASM) thành file đối tượng (OBJ)
- **Link (Alt+Ctrl+F5)**: Liên kết file đối tượng (OBJ) thành file thực thi (EXE)
- **Run! (Ctrl+Shift+F5)**: Thực thi chương trình (EXE)

2. Soạn thảo file chương trình nguồn

Chọn lệnh **New File** trong menu **FILE** (*hay bấm phím nóng tương ứng*) để mở vùng soạn thảo mới, trong vùng soạn thảo, sử dụng các chức năng soạn thảo giống như các trình soạn thảo khác.

Sau khi soạn thảo file nguồn xong phải lưu lại thành tên file có phần mở rộng là ASM. Và trong tên file không có chứa khoảng trắng.

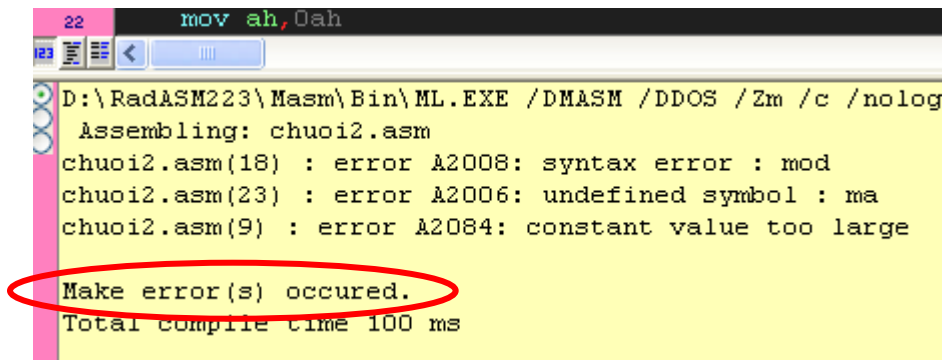
3. Hợp dịch (Assemble)

Sau khi lưu file chương trình nguồn xong, chọn **Assemble** trong menu **MAKE** (*Hay bấm phím nóng tương ứng*) để tiến hành hợp dịch chương trình nguồn.

- Nếu chương trình có lỗi cú pháp thì vùng thông báo có dạng hình 4. Trong đó từng lỗi được chỉ ra bởi số thứ tự dòng lệnh và mã lỗi

Ví dụ: **chuoi2.asm(18) : error A2008: syntax error : mod**

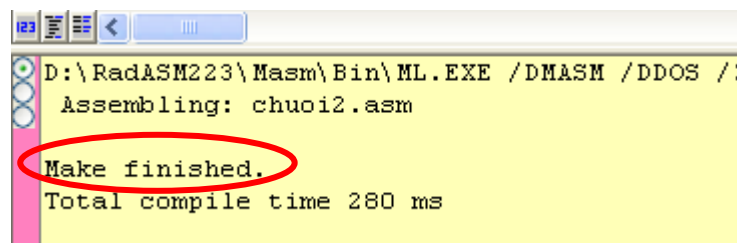
Lỗi ở dòng 18, mã lỗi A2008: sai cú pháp *mod*



```
22      mov ah, 0ah
D:\RadASM223\Masm\Bin\ML.EXE /DMASM /DDOS /Zm /c /nolog
Assembling: chuoi2.asm
chuoi2.asm(18) : error A2008: syntax error : mod
chuoi2.asm(23) : error A2006: undefined symbol : ma
chuoi2.asm(9) : error A2084: constant value too large
Make error(s) occurred.
Total compile time 100 ms
```

Hình 4: Thông báo kết quả Hợp dịch không thành công

- Khi chương trình xuất hiện lỗi thì file đối tượng (OBJ) không được tạo ra và người lập trình phải sửa lại cho đến khi không còn lỗi.
- Khi không có lỗi chương trình (Hợp dịch thành công) thì trong vùng thông báo xuất hiện như hình 5 và file đối tượng (OBJ) được tạo ra. Khi đó mới chuyển sang bước LIÊN KẾT



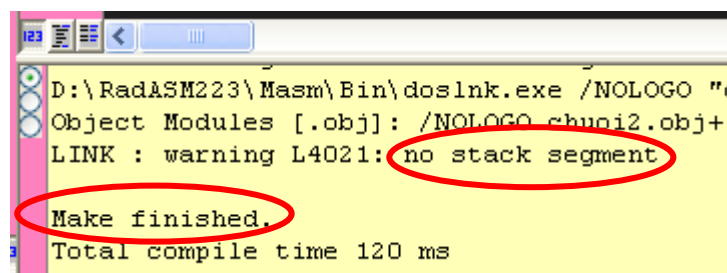
```
D:\RadASM223\Masm\Bin\ML.EXE /DMASM /DDOS /Z
Assembling: chuoi2.asm
Make finished.
Total compile time 280 ms
```

Hình 5: Thông báo kết quả Hợp dịch thành công

4. Liên kết (Link)

Chọn lệnh **Link** trong menu **MAKE**. (hay bấm phím nóng tương ứng)

Sau khi hoàn thành bước hợp dịch, thì bước liên kết là bước cuối cùng để tạo ra file thực thi (EXE). Trong bước này, không còn kiểm tra lỗi cú pháp nữa và thông thường sẽ liên kết thành công và file thực thi EXE sẽ được tạo ra. Nội dung thông báo khi liên kết thành công như hình 6.



```
D:\RadASM223\Masm\Bin\doslnk.exe /NOLOGO "c
Object Modules [.obj]: /NOLOGO chuoi2.obj+
LINK : warning L4021: no stack segment
Make finished.
Total compile time 120 ms
```

Hình 6: Liên kết thành công

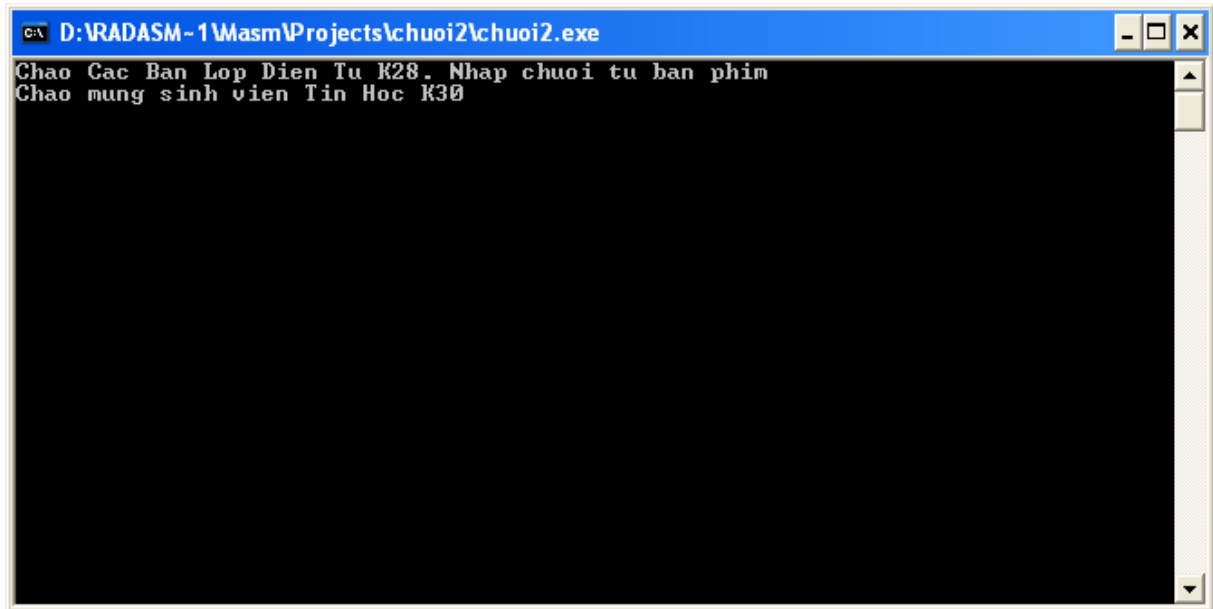
5. Thực thi chương trình EXE

Để thực thi chương trình vừa liên kết xong, chọn lệnh **RUN** trong menu **MAKE** (hay bấm phím nóng tương ứng).

Nếu chương trình có truy xuất đến bàn phím hay màn hình thì 1 cửa sổ (dạng màn hình của hệ điều hành DOS) xuất hiện như hình 7. Cửa sổ này được tạo ra để hiển thị kết quả hay để người dùng tương tác với chương trình đang chạy.

Nếu chương trình không có thao tác nào để truy xuất bàn phím hay màn hình thì chúng ta không thấy được cửa sổ này.

Như vậy, trong lập trình hợp ngữ, nếu muốn nhìn thấy kết quả gì đó thì chúng ta phải có những đoạn lệnh tương ứng để xuất giá trị ra màn hình.



Hình 7: Cửa sổ kết quả chương trình