Chương 1
TỔ CHỨC BỘ XỬ LÝ INTEL 8086

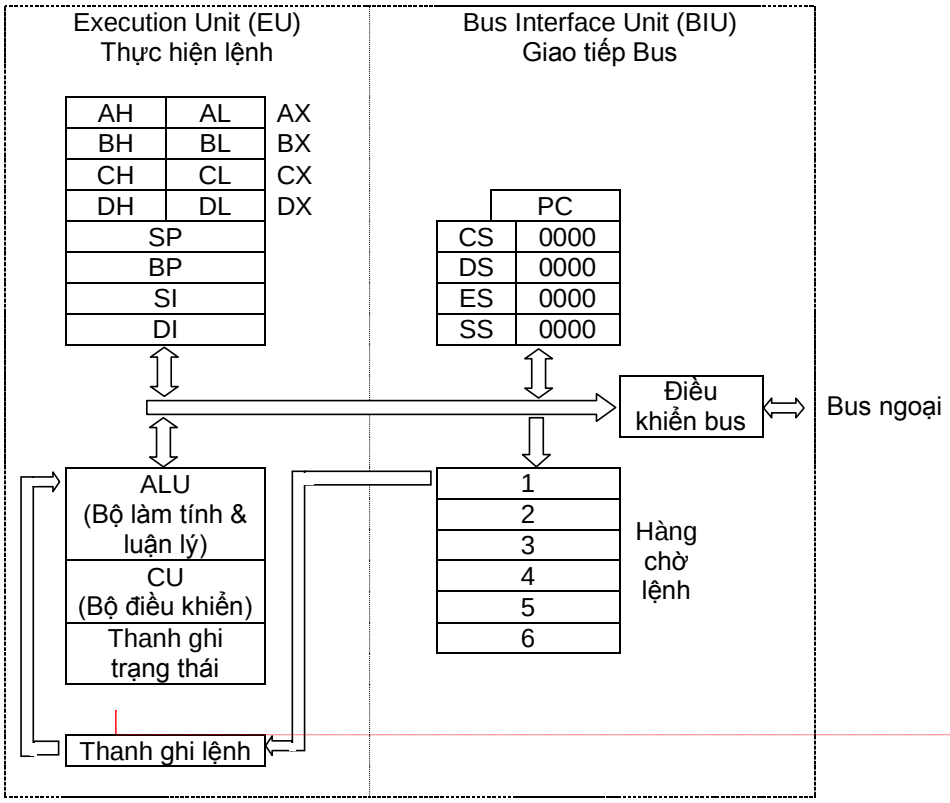
I. BỘ XỬ LÝ (CPU) INTEL 8086

1. Tổ chức tổng quát

CPU 8086 là CPU 16 bit (bus số liệu có 16 dây). Nó được dùng để chế tạo các máy vi tính đầu tiên của hãng IBM. Thật ra IBM dùng CPU 8088 với đường bus số liệu ra ngoài là 8 bit.

Cho đến nay, CPU đã không ngừng cải tiến và đã trải qua các dạng 80186, 80286, 80386, 80486, 80586 (Pentium), Pentium Pro, Pentium II, PIII, PIV.

Các CPU tương thích từ trên xuống (downward compatible) nghĩa là tập lệnh mới bao gồm tập lệnh của CPU cũ và thêm nhiều lệnh mới nữa.



Hình: *Sơ đồ khối của CPU 8086*

Ta thấy CPU 8086 chia thành 2 bộ phận chính: Bộ phận thực hiện lệnh (Execution Unit: EU) và bộ phận giao tiếp Bus (Bus Interface Unit: BIU).

Bộ phận thực hiện lệnh EU: Kiểm soát các thanh ghi, giải mã và thi hành lệnh tức là làm các tác vụ mà lệnh yêu cầu. Như vậy EU làm hầu hết công việc của CPU cổ

Comment [Mr1]: Phím Enter

điển. Các thanh ghi và đường bus trong EU điều là 16 bit. EU không nối với bus hệ thống bên ngoài, nó lấy lệnh từ hàng chờ lệnh mà BIU cung cấp. Khi có yêu cầu truy xuất bộ nhớ hay ngoại vi thì EU yêu cầu BIU làm việc. BIU có thể tái định địa chỉ cho phép EU truy xuất đầy đủ 1MB (8086 có 20 đường địa chỉ ngoại).

Bộ phận giao tiếp bus (BIU): BIU thực hiện tất cả các tác vụ về bus của EU. Trong khi EU đang thực hiện lệnh thì BIU lấy lệnh từ bộ nhớ trong và cất giữ vào trong ô nhớ (gọi là hàng chờ lệnh) bên trong CPU. Do đó, EU không phải đợi lấy lệnh từ bộ nhớ. Đây là một cách đơn giản của cache.

2. Các thanh ghi của 8086

a. Thanh ghi đa dạng: CPU 8086 có 4 thanh ghi đa dụng 16bit, có thể chia đôi thành 8 thanh, mỗi thanh 8 bit.

- AX (accumulator): là thanh ghi tích luỹ cơ bản, mọi tác vụ vào/ra đều dùng thanh ghi này, tác vụ dùng số liệu tức thời, một số tác vụ chuỗi ký tự và các lệnh tính toán đều dùng thanh ghi AX.
- BX (base register): là thanh ghi nền thường dùng để tính toán địa chỉ ô nhớ.
- CX (count register): là thanh ghi đếm thường dùng để đếm số lần trong một lệnh vòng lặp hoặc xử lý chuỗi ký tự.
- DX (data register): thường chứa địa chỉ của một số lệnh vào ra, lệnh tính toán số học (kể cả nhân và chia).

b. Thanh ghi con trỏ: Dùng để thâm nhập số liệu trên ngăn xếp.

- SP (stack pointer): Thanh ghi con trỏ ngăn xếp.
- BP (base pointer): Thanh ghi con trỏ nền dùng để lấy số liệu từ ngăn xếp.

c. Thanh ghi chỉ số

- SI (source index): Thanh ghi chỉ số nguồn.
- DI (destination index): Thanh ghi chỉ số đích.

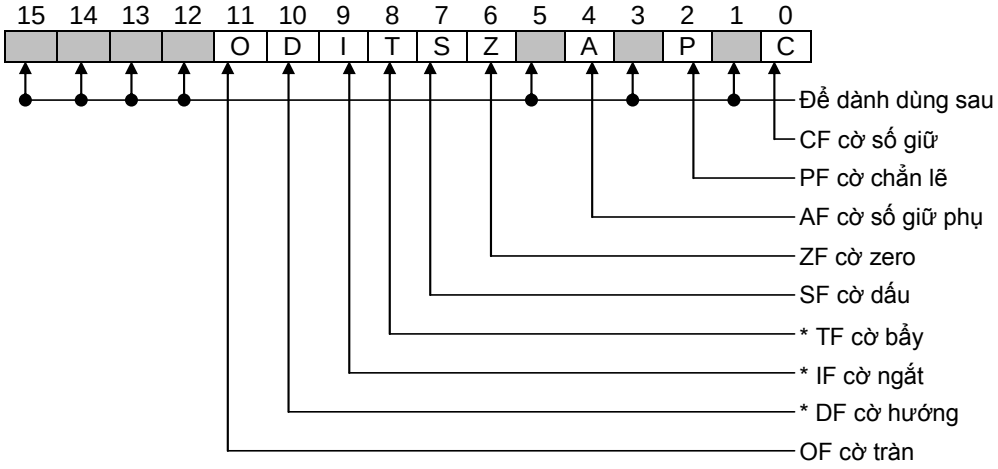
d. Thanh ghi đoạn: Được dùng trong mọi tính toán địa chỉ ô nhớ. Mỗi thanh ghi đoạn xác định 64 KB ô nhớ trong bộ nhớ trong.

- CS (code segment): Thanh ghi đoạn mã lệnh.
- DS (data segment): Thanh ghi đoạn dữ liệu.
- ES (extra segment): Thanh ghi đoạn thêm. Các pháp tính chuỗi dùng DI đều liên quan đến ES.
- SS (stack segment): Thanh ghi đoạn ngăn xếp. Con trỏ SP luôn trỏ tới đỉnh của ngăn xếp.

e. Thanh ghi cờ: Phản ảnh kết quả của phép tính toán số học và luận lý, xác định trạng thái hoạt động của CPU. Các bit trên thanh ghi cờ có ý nghĩa được trình bày dưới đây.

- CF: thể hiện số giữ thoát ra từ bit cao nhất của thanh ghi kết quả sau một phép tính toán.
- OF: thể hiện việc tính toán vượt quá khả năng của CPU.
- AF: thể hiện số giữ thoát ra từ bít thứ 4 (bit 3) của thanh ghi kết quả.
- PF: bằng 1 nếu 8 bít thấp của thanh ghi kết quả một phép tính toán có số con số 1 chẵn (và ngược lại).
- ZF: bằng 1 khi kết quả phép tính bằng 0 (và ngược lại).

- DF: có thể lập trình được, bằng 1 thì SI và DI giảm 1 cho mỗi vòng lặp.
- IF: có thể lập trình được, bằng 1 cho phép ngắt.
- TF: có thể lập trình được, bằng 1 khi cho phép chương trình chạy từng bước để phục vụ sửa sai một chương trình.



3. Tổ chức bộ nhớ trong

Bộ nhớ trong được tổ chức thành từng mảng gồm các ô nhớ 8 bit. Các dữ liệu có thể cất giữ hoặc lấy ra từ bất kỳ ô nhớ nào. Mỗi ô nhớ có một địa chỉ.

Theo qui ước của Intel các dữ kiện 16 bit cất giữ vào ô nhớ với byte cao ở địa chỉ cao và byte thấp ở ô nhớ có địa chỉ thấp.

4. Sự phân đoạn trong bộ nhớ trong

CPU 8086 có không gian địa chỉ là 1 MB (ứng với 20 đường dây địa chỉ). Vậy CPU 8086 có thể quản lý bộ nhớ trong là 2^{20} = 1 MB.

Bộ nhớ 1MB này có thể chia thành nhiều đoạn 64 KB. Các đoạn có thể chồng lên nhau.

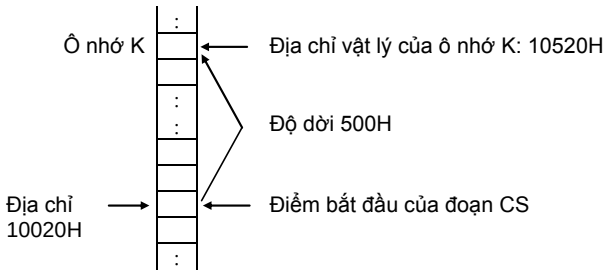
Mỗi địa chỉ ô nhớ xác định bởi 2 số:

- Đoạn 16 bit.
- Độ dời (offset).
- Địa chỉ cụ thể còn gọi là địa chỉ vật lý được tính bằng cách dịch trái thanh ghi đoạn 4 bit (nhân cho 16) rồi cộng vào độ dời.

Ví dụ: Đoạn CS có giá trị là 1002H, thì địa chỉ vật lý của ô nhớ K trong đoạn CS có độ dời 500H (thường viết CS:500H) là:

```
    10020H   Vì 1002H dịch trái 4bit = 10020H
+     500H
-------------
    10520H   ← Đây là địa chỉ vật lý của ô nhớ K
```

Trong ví dụ ta thấy đoạn CS có điểm bắt đầu ở địa chỉ vật lý 10020H. Độ dời 500H là khoảng cách từ địa chỉ của điểm bắt đầu của đoạn CS đến ô nhớ K.



Chính BIU quyết định sẽ dùng đoạn nào theo tính chất của số liệu.

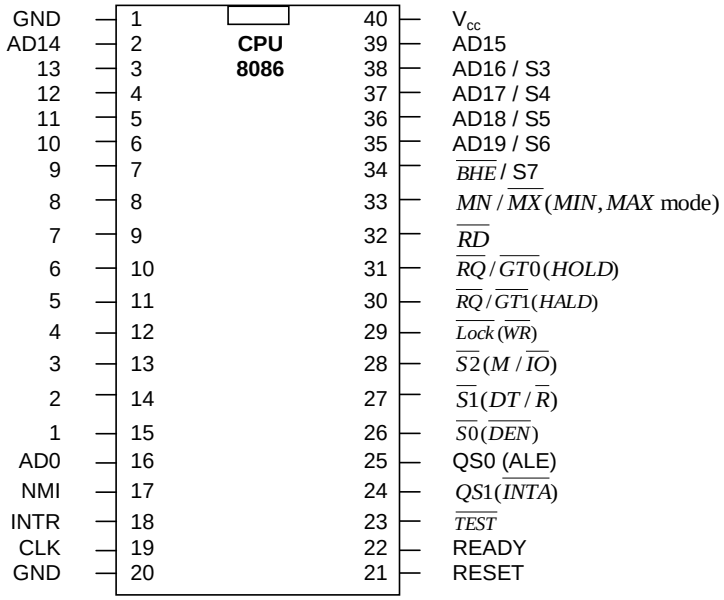
- Nếu số liệu là mã lệnh thì BIU sẽ dùng đoạn CS.
- Nếu số liệu là dữ liệu của chương trình thì BIU sẽ dùng đoạn DS.
- Nếu số liệu là dữ liệu nằm trên ngăn xếp thì BIU sẽ dùng đoạn SS.
- Nếu dùng các phép tính chuỗi thì thanh ghi DI luôn chứa độ dời của ô nhớ trong đoạn ES.
- Lúc khởi động CPU 8086 đến địa chỉ vật lý cao của bộ nhớ trong (đoạn CS=0FFFFH và độ dời 0) để lấy lệnh. Địa chỉ này ứng với địa chỉ ROM của bộ nhớ trong.

5. Địa chỉ các ngoại vi

Các ngoại vi đều có địa chỉ riêng từ 0 đến 64 KB. CPU 8086 dùng các lệnh riêng biệt để truy xuất ngoại vi và bộ nhớ trong. Muốn truy xuất ngoại vi, BIU chỉ cần đưa địa chỉ của ngoại vi lên 16 bit thấp của bus địa chỉ (không có đoạn).

6. Các chân của vi mạch 8086

AD0 … AD15 + AD16 … AD19 là 20 chân của bus địa chỉ, các chân từ AD0 đến AD15 được đa hợp (multiplex) với bus số liệu, các chân từ AD16 đến AD19 được đa hợp với các nhiệm vụ về trạng thái thể hiện ở các chân S3, S4, S5, S6.



Hình: *Vi mạch CPU 8086.*

Sau chu kỳ máy thứ nhất S3, S4 cho ta biết đoạn nào được dùng để tạo địa chỉ.

S3	S4	Ý nghĩa
0	0	Đoạn ES
0	1	Đoạn SS
1	0	Đoạn CS hoặc không đoạn nào
1	1	Đoạn DS

S5 thể hiện trạng thái cờ ngắt (interrupt flag).
S6 được giữ ở trạng thái thấp nếu CPU đang sử dụng hệ thống bus ngoài.

S7 lưu giữ trạng thái của $\overline{BHE}$ ở chu kỳ máy thứ nhất.

$\overline{RD}$: CPU dùng tín hiệu này để đọc số liệu từ ô nhớ hay từ các thiết bị ngoại vi.
Ready: Ô nhớ hoặc ngoại vi có thể dùng tín hiệu này để báo cho CPU biết nó đang sẵn sàng chuyển dữ liệu.

$\overline{TEST}$: Khi ta dùng lệnh WAIT thì CPU ở trạng thái nghỉ cho đến khi tín hiệu ở chân này xuống thấp thì CPU mới thi hành lệnh kế sau lệnh WAIT.
INTR: Các ngoại vi tác động vào chân này khi cần ngắt CPU.
NMI (non maskable interrupt): Đây là ngã vào của ngắt không che, ngắt không che có ưu tiên tuyệt đối.
Reset: Khởi động lại hệ thống.

CPU 8086 có hai chế độ vận hành MAX ($\overline{MX}$) và MIN (MN). Nhiệm vụ của các chân tương ứng với 2 chế độ vận hành như sau:

MIN (MN)	MAX ($\overline{MX}$)
$HOLD$	$\overline{RQ}/\overline{GT0}$
$HALD$	$\overline{RQ}/\overline{GT1}$
$\overline{WR}$	$\overline{Lock}$
$M/\overline{IO}$	$\overline{S2}$
$DT/\overline{R}$	$\overline{S1}$
$\overline{DEN}$	$\overline{S0}$
ALE	QS0
$\overline{INTA}$	QS1

◆ **Chế độ MN (hiệu điện thế ở chân $MN/\overline{MX}$ cao)**

$\overline{DEN}$ (data enable): Cho phép số liệu được nhận vào CPU hoặc đưa ra bus số liệu tuỳ theo tín hiệu ở chân $DT/\overline{R}$. Nếu chân $DT/\overline{R}$ có hiệu thế cao, CPU đưa số liệu ra bus hệ thống. Nếu $DT/\overline{R}$ có hiệu thế thấp, CPU nhận số liệu từ bus hệ thống.

$M/\overline{IO}$ (memory / input output): Chân này ở trạng thái cao nếu CPU liên hệ với bộ nhớ. Nó ở trạng thái thấp nế CPU làm việc với ngoại vi.
HOLD: Các ngoại vi tác động vào chân này nếu muốn sử dụng bus hệ thống.
HLDA (hold acknowledge): CPU dùng tín hiệu này để báo cho ngoại vi biết nó đang thả nổi bus hệ thống.
ALE (address latch enable): Tín hiệu ở chân này cho biết địa chỉ của ô nhớ đã được đưa ra bus hệ thống.
$\overline{INTA}$ (interrupt latch enable): Đây là tín hiệu cho biết CPU đã công nhận ngắt mà ngoại vi yêu cầu.
$\overline{WR}$: Tín hiệu dùng để viết số liệu vào bộ nhớ.

◆ **Chế độ MX (hiệu điện thế ở chân $MN/\overline{MX}$ thấp)**
S0, S1, S2 kết hợp, ý nghĩa như sau:

S0	S1	S2	Ý nghĩa
0	0	0	Công nhận ngắt
0	0	1	Đọc từ ngoại vi
0	1	0	Viết ra ngoại vi
0	1	1	Trạng thái dừng (HALT)
1	0	0	Tìm lệnh
1	0	1	Đọc bộ nhớ
1	1	0	Viết vào bộ nhớ
1	1	1	Không có hoạt động

$\overline{RQ}/\overline{GT0}$ (request / grant): Ngoại vi tạo một xung thấp ở chân này để báo cho CPU biết nó cần sử dụng bus hệ thống. CPU báo lại bằng một xung âm cho biết nó đã thả nổi bus hệ thống.

$\overline{RQ}/\overline{GT1}$ giống như $\overline{RQ}/\overline{GT0}$ nhưng ưu tiên thấp hơn.

QS0 và QS1 cho biết trạng thái của hàng chờ lệnh như sau:

QS0	QS1	Ý nghĩa
0	0	Chưa có tác vụ
0	1	Byte thứ nhất của lệnh được thực hiện
1	0	Hàng chờ lệnh đã đầy
1	1	Byte kế tiếp của lệnh đang được lấy đi từ hàng chờ lệnh

LOCK: Đây là tín hiệu báo CPU đang sử dụng bus hệ thống.

GND (ground): là chân mass (0 volt). V_{cc} là hiệu điện thế nguồn 5 volt.

CPU 8086 phải dùng chung với một số vi mạch khác như: vi mạch điều khiển bus, vi mạch tạo xung nhịp (clock),… mới tạo thành một máy vi tính.

II. CÁC LỆNH THƯỜNG DÙNG CỦA CPU 8086

1. Giới thiệu

Bộ xử lý 8086 có tập lệnh gồm 111 lệnh, chiều dài của lệnh từ 1 byte đến vài byte.

Tập lệnh của bộ xử lý Intel ngày càng có nhiều lệnh mạnh và phức tạp. Bộ xử lý Intel 80386 có 206 lệnh, các lệnh có chiều dài từ 1 đến 15 byte, một số lệnh cần 1000 chu kỳ xung nhịp để thực hiện.

Sau đây chúng ta chỉ đề cập một số lệnh thường dùng của CPU 8086. Tập lệnh đầy đủ của bộ xử lý 8086 được nêu ở phụ lục. Tập lệnh của bộ xử lý 80386 và Pentium không đề cập trong giáo trình này vì quá phức tạp.

Chúng ta dùng các qui ước sau:

- Reg (register): Thanh ghi.
- Reg8, Reg16: Thanh ghi 8 bit, 16 bit.
- Mem (memory): Ô nhớ.
- Mem8, Mem16: Ô nhớ 8 bit, 16 bit.
- Immed (immediate): Tức thì.
- Immed8, Immed16: Toán hạng tức thì 8 bit, 16 bit.
- Segreg (segment register): Thanh ghi đoạn.

2. Nhóm di chuyển số liệu

- **MOV (move)**: Di chuyển.
  MOV đích, nguồn

Lệnh này di chuyển số liệu từ nguồn sang đích. Nguồn có thể là Reg, Mem, Immed; đích có thể là Reg, Mem.

Ví dụ: MOV CX, BX
Lệnh trên thực hiện chuyển nội dung của thanh ghi BX vào thanh ghi CX. Nội dung của thanh ghi BX giữ nguyên. Sau lệnh này thì BX và CX có cùng nội dung.

- **PUSH (push)**: đẩy vào.
  PUSH nguồn
  Nguồn có thể là Reg16, Mem16.
  Lệnh PUSH là giảm con trỏ ngăn xếp SP xuống 2 đơn vị.

  Ví dụ: PUSH AX
  Lệnh trên thực hiện chuyển nội dung của thanh ghi AX vào ngăn xếp, đồng thời con trỏ SP giảm 2 đơn vị.
- **POP (pop: lấy, di chuyển)**: lấy dữ liệu ra từ ngăn xếp.
  POP đích
  Đích có thể là Reg16, Mem16.
  Lệnh POP là tăng con trỏ ngăn xếp SP xuống 2 đơn vị.

  Ví dụ: POP BX
  Lệnh trên thực hiện chuyển nội dung 2 byte ô nhớ mà SP trỏ tới để đưa vào BX, byte có địa chỉ thấp đưa vào BL, byte có địa chỉ cao đưa vào BH, đồng thời con trỏ SP tăng 2 đơn vị.

***Ghi chú*:** Qua 2 lệnh PUSH và POP, ta thấy ngăn xếp ô nhớ đi từ ô nhớ có địa chỉ cao, đến ô nhớ có địa chỉ thấp, nghĩa là số liệu đưa vào ngăn xếp trước thì địa chỉ cao, số liệu đưa và ngăn xếp sau thì ở địa chỉ thấp hơn.

3. Nhóm lệnh chuyển địa chỉ

- **LEA (load effective address)**: nạp địa chỉ hiệu dụng.
  LEA Reg16, Mem16
  Chuyển độ dời của ô nhớ Mem16 vào thang ghi Reg16.

  Ví dụ: LEA DX, StringVar
  Lệnh trên thực hiện chuyển độ dời của biến StringVar vào thanh ghi DX. Ta cũng có thể viết (MASM): **MOV DX, offset StringVar** tương đương lệnh trên.

4. Nhóm lệnh chuyển cờ hiệu (thanh ghi trạng thái)

- **PUSHF (push flag)**: lưu giữ cờ.
  PUSHF (không có đối)
  Đây là lệnh lưu giữ thanh ghi cờ vào ngăn xếp.
- **POPF (pop flag)**: lấy cờ ra.
  POPF (không có đối)
  Đây là lệnh lấy 2 byte từ ngăn xếp đưa vào thanh ghi cờ.

  Ví dụ: Để đưa nội dung của thanh ghi cờ vào thanh ghi AX ta làm như sau:
  PUSHF
  POP AX

5. Nhóm lệnh vào ra ngoại vi

- **IN (in: vào)**: lấy số liệu từ ngoại vi.
  IN AL, địa chỉ cổng 8bit
  Đây là lệnh đưa số liệu lưu giữ ở ô nhớ đệm ngã ra của cổng vào thanh ghi AL.
  Nếu địa chỉ của cổng là 16bit thì phải đưa địa chỉ của cổng này vào thanh ghi DX trước khi sử dụng lệnh IN.

  Ví dụ:
  IN AL, 3FH ;3FH là địa chỉ của cổng 8bit.
  MOV DX, 3F8H ;3F8H là địa chỉ cổng 16 bit.
  IN AL, DX
- **OUT (out: ra)**: đưa số liệu ra ngoại vi.
  OUT địa chỉ cổng 8bit, AL
  MOV DX, địa chỉ cổng 16 bit.
  OUT DX, AL

6. Nhóm lệnh điều khiển

- **JMP (jump: nhảy)**: đây là lệnh nhảy vô điều kiện đến một địa chỉ khác.
- JMP đích
  Nhảy đến địa chỉ được đánh dấu bằng một nhãn hay một con số ám chỉ độ dời. Nhãn nằm trong đoạn CS hiện tại.
  Tuỳ theo khoảng cách của đích đến lệnh JMP mà ta có 3 kiểu lệnh này.
  JMP near đích
  Lệnh này (còn được viết: JMP đích) nhảy đến đoạn nằm trong CS hiện tại.
  JMP short đích
  Dùng để nhảy đến địa chỉ trong khoảng –128 đến +127 tính từ lệnh JMP.
  JMP far đích
  Dùng để nhảy ra khỏi đoạn CS hiện tại.
- Lệnh nhảy có điều kiện: Lệnh này kiểm tra điều kiện trước khi nhảy. Nếu điều kiện đúng thì nhảy tới đích, ngược lại thi hành lệnh kế đó một cách bình thường.
- **JA** (jump if above: nhảy nếu lớn hơn). Nếu 2 cờ CF=ZF=0 thì nhảy đến đích.
- **JB** (jump if below: nhảy nếu nhỏ hơn). Nếu cờ CF=1 thì nhảy đến đích.
- **JZ** (jump if zero: nhảy nếu bằng 0). Nếu cờ ZF=1 (phép toán trước đó bằng 0 hoặc so sánh bằng nhau) thì nhảy.
- **JNZ** (jump if not zero: nhảy nếu khác 0). Nếu cờ ZF=0 (phép toán trước đó khác 0 hoặc so sánh khác nhau) thì nhảy.

7. Nhóm lệnh so sánh

- **Lệnh CMP**
- CMP trái, phải

Nếu trái > phải thì ZF = 0 và CF =0
Nếu trái = phải thì ZF = 1 và CF =0
Nếu trái < phải thì ZF = 0 và CF =1

Ví dụ:

```
                MOV AX, 1000H
                CMP AX, 200H
                JZ NHANDEN      ; nhãn đến
                …
                …
NHANDEN:
                ADD AX, BX
```

8. Nhóm lệnh vòng lặp

- **LOOP (loop: nhảy vòng)**: Lệnh này làm giảm thanh ghi CX xuống 1 và nhảy tới một nhãn (trong vòng –128 đến 127) nếu CX khác 0.

  Ví dụ:

```
                MOV AH, 02H
                MOV DL, 48
                MOV CX, 10      ; lặp 10 lần
BATDAU:
                INT 21H
                INC DL
                LOOP BATDAU
                …
                …
```

- LOOPZ (loop if zero: nhảy nếu bằng 0)
  Nhảy vòng nếu cờ ZF =1.
- LOOPNZ (loop if not zero: nhảy nếu khác 0)
  Nhảy vòng nếu cờ ZF =0.

9. Nhóm lệnh gọi chương trình con

- **CALL (call: gọi)**: lệnh gọi chương trình con.
  CALL Nhản (hoặc tên chương trình con)
  Lệnh gọi chương trình con là một lệnh đặc biệt vì trước khi nhảy tới nhãn thì CPU tự động lưu địa chỉ trở về (là địa chỉ sau lệnh CALL) vào ngăn xếp.
- **RET (return: trở về)**. Lệnh kết thúc chương trình con.
  Khi gặp lệnh này thì CPU 8086 lấy địa chỉ trở về ở ngăn xếp để tiếp tục thi hành lệnh ở chương trình chính.

Ví dụ: Chương trình sau sử dụng Macro, Procedure.

```
INCHU Macro
        mov ah,02
        mov dl, 'A'
        int 21h
ENDM

dulieu segment
        thongbao db 'hello!$'
```

```
dulieu ends

malenh1 segment
        P2 Proc far
                mov ah,02
                mov dl, 'C'
                int 21h
                ret
        P2 Endp
malenh1 ends

malenh2 segment
        malenh group malenh1, malenh2
        assume cs: malenh, ds: dulieu
batdau: mov ax, dulieu
        mov ds, ax          ; khoi dong

        lea dx, thongbao  ; in thong bao nhap chuoi
        mov ah, 09h
        int 21h

        inchu     ;goi macro
        call pinchu          ;goi ctrinh con
        call p2    ;goi ctrinh con

        mov ah, 4ch
        int 21h

        PINCHU Proc near
                mov ah,02
                mov dl, 'B'
                int 21h
                ret
        PINCHU Endp
malenh2         ends
end     batdau
```

10. Nhóm lệnh tính toán số học

- **ADD (add: cộng)**: cộng 2 số nguyên, lấy nguồn cộng vào đích và kết quả lưu ở đích.
  ADD đích, nguồn
  Đích là Reg, Mem; nguồn là Reg, Mem hoặc Immed.

  Ví dụ:

  ```
  MOV AL, 02
  ADD AL, 06
  (Sau 2 lệnh này AL = 08 tức là AL = 02+06)
  ```

- **INC (increment: tăng)**
  INC đích
  Đích là Reg, Mem. Lệnh này tăng nội dung của đích lên 1 đơn vị.

  Ví dụ:

  ```
  MOV AL, 02
  INC AL      ;Sau 2 lệnh này AL = 3.
  ```

- **SUB (subtract: trừ ra)**: lấy đích trừ nguồn và kết quả lưu ở đích.
  SUB đích, nguồn
  Đích là Reg, Mem; nguồn là Reg, Mem hoặc Immed.

  Ví dụ:

MOV AL, 09
SUB AL, 06
(Sau 2 lệnh này AL = 03 tức là AL = 09-06)

- **DEC (decrement: giảm)**
  DEC đích
  Đích là Reg, Mem. Lệnh này giảm nội dung của đích xuống 1 đơn vị.

  Ví dụ:

  MOV AL, 02
  DEC AL ;Sau 2 lệnh này AL = 1.
- **MUL (multiplication: phép nhân)**: nhân số không dấu.
  MUL nguồn
  Nguồn là Reg, Mem. Lệnh này lấy thanh ghi tích luỹ (AX) nhân cho nguồn.
  Nếu nguồn 8bit thì số này được nhân AL và kết quả đặt trong AX.
  Nếu nguồn là 16 bit, số này được nhân với AX và kết quả để trong DX:AX (DX:AX là một số 32 bit, 16 bit cao trong DX và 16 bit thấp trong AX).

  Ví dụ:

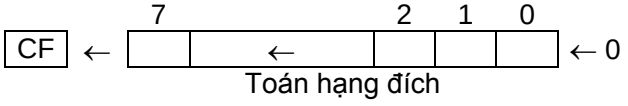
  MOV AL, 9
  MOV BL, 2
  MUL BL
  (Sau 3 lệnh này AL có giá trị bằng 18).
- **DIV (division: phép chia)**: chia số không dấu.
  DIV nguồn
  Nguồn là Reg, Mem. Lệnh này lấy thanh ghi tích lũy (AX) chia cho nguồn.
  Nếu nguồn 8bit thì lấy AX chia cho nguồn, kết quả đặt trong AL và số dư đặt trong AH.
  Nếu nguồn là 16 bit thì lấy DX:AX chia cho nguồn. Thương số đặt trong AX, số dư đặt trong DX.

  Ví dụ:

  MOV AX, 9520
  MOV BL, 100
  DIV BL
  (Sau 3 lệnh này AL = 95 và AH = 20 vì 9520 : 100 = 95 và dư 20).

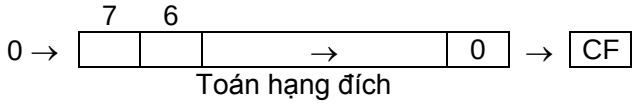
11. Nhóm lệnh dịch chuyển và quay

- **SHL (logical shift left)**: dịch trái logic.
  SHL đích, 1 ; dịch trái toán hạng đích 1 bit (dịch từ 2 bit trở lên phải đặt số lần dịch trong CL).
  SHL đích, CL ; dịch trái toán hạng số bit bằng nội dung của CL.



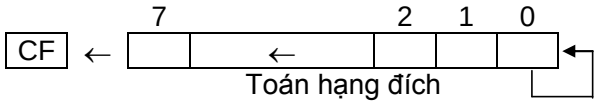
Trong hình trên, giả sử toán hạng đích là 8bit thì lệnh **SHL đích, 1** làm bít thứ 7 dịch sang bit CF (cờ), bít 6 chuyển sang bit7, bít5 sang bit6,… bit0 qua bit1 và số 0 vào bit0.

- **SHR (logical shift right)**: dịch phải logic.
  Lệnh này giống như SHL nhưng bây giờ dịch phải.

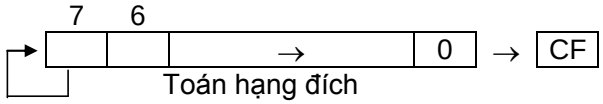


Trong hình trên, giả sử toán hạng đích là 8bit thì lệnh **SHR đích, 1** làm 0 vào bít7, bit7 sang bít 6,… bit0 dịch sang bit CF (cờ).

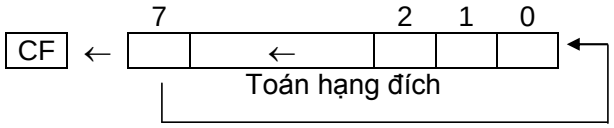
- SAL (shift arithmetic left): dịch trái số học.
  Giống như lệnh SHL nhưng bit0 được giữ nguyên



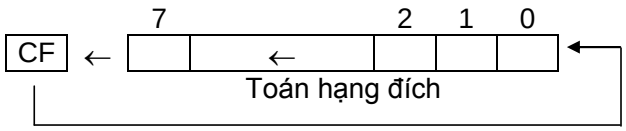
- **SAR (shift arithmetic right)**: dịch phải số học.
  Giống như lệnh SHL nhưng bit cao nhất được giữ nguyên.



- **ROL (rotate left)**: quay vòng sang trái.
  ROL đích, 1 ; quay vòng sang trái toán hạng đích 1 bit (quay vòng sang trái từ 2 bit trở lên phải đặt số lần dịch trong CL).
  ROL đích, CL ; dịch trái toán hạng số bit bằng nội dung của CL.



- **ROR (rotate right)**: quay vòng sang phải (giống như quay vòng sang trái nhưng bây giờ sang phải).
  ROR đích, 1
  ROL đích, CL
- **RCL (rotate through carry left)**: quay vòng qua bit số giữ sang trái, giống như lệnh ROL nhưng có sự tham gia của bit số giữ.



- **RCR (rotate through carry right)**: quay vòng qua bit số giữ sang phải, giống như lệnh RCL nhưng sang phải.

12. Nhóm lệnh logic

- **AND (and: và)**: lệnh này lấy từng bit của toán hạng đích and với từng bit của toán hạng nguồn, kết quả lưu ở đích.
  AND đích, nguồn
  Đích là Reg, Mem; nguồn là Reg, Mem hoặc Immed.

  Ví dụ:

  ```
  MOV AL, 01010101B
  AND AL, 00001111B
  (Sau 2 lệnh này AL = 00000101B)
  ```
- **OR (or: hoặc)**: lệnh này lấy từng bit của toán hạng đích or với từng bit của toán hạng nguồn, kết quả lưu ở đích.
  OR đích, nguồn
  Đích là Reg, Mem; nguồn là Reg, Mem hoặc Immed.

  Ví dụ:

  ```
  MOV AL, 01010101B
  OR AL, 00001111B
  (Sau 2 lệnh này AL = 01011111B)
  ```
- **XOR (xor: hoặc loại)**: lệnh này lấy từng bit của toán hạng đích xor với từng bit của toán hạng nguồn, kết quả lưu ở đích.
  XOR đích, nguồn
  Đích là Reg, Mem; nguồn là Reg, Mem hoặc Immed.

  Ví dụ:

  ```
  MOV AL, 01010101B
  XOR AL,  00001111B
  (Sau 2 lệnh này AL = 01011010B)
  ```
- **NOT (not: đảo)**: lệnh này lấy đảo từng bit của toán hạng đích.
  NOT đích
  Đích là Reg, Mem.

  Ví dụ:

  ```
  MOV AL, 01010101B
  NOT AL
  (Sau 2 lệnh này AL = 10101010B)
  ```
- **TEST (test: trắc nghiệm)**: lệnh này giống như lệnh AND nhưng không lưu giữ kết quả mà chỉ ảnh hưởng đến các cờ.
  TEST đích, nguồn

13. Nhóm lệnh xử lý chuỗi

- MOVSB (move string byte): di chuyển chuỗi từng byte một.
- MOVSW (move string word): di chuyển chuỗi từng 16 bit.
- CMPSB (compare string byte): so sánh chuỗi từng byte một.
- CMPSW (compare string word): so sánh chuỗi từng 16 bit.
- SCASB (scan string byte): quét chuỗi từng byte một.

- SCASW (scan string word): quét chuỗi từng 16 bit.
- LODSB (load string byte): nạp chuỗi từng byte một.
- LODSW (load string word): nạp chuỗi từng 16 bit.
- STOSB (store string byte): lưu chuỗi từng byte một.
- STOSW (store string word): lưu chuỗi từng 16 bit.

Cách dùng các lệnh giống nhau và thường phải qua các bước sau:

Bước 1: Xác định chiều xử lý chuỗi.
DF = 0: chuỗi xử lý theo chiều địa chỉ tăng.
DF = 1: chuỗi xử lý theo chiều địa chỉ giảm.

Bước 2: Số lượng phần tử cần xử lý được nạp vào thanh ghi đếm CX.

Bước 3: Đưa địa chỉ của chuỗi vào đúng vị trí.
Địa chỉ chuỗi nguồn đưa vào DS:SI.
Địa chỉ chuỗi đích đưa vào ES:DI.

Bước 4: Chọn một vòng lặp thích hợp REP, REPE, REPNE.
Lệnh REP (repeat: lặp lại): lặp lại lệnh theo sau nó đến khi CX=0.
Lệnh REPE (repeat if equal: lặp lại nếu bằng nhau): lặp lại lệnh theo sau nó nếu ZF=1.
Lệnh REPNE (repeat if not equal: lặp lại nếu không bằng nhau): lặp lại lệnh theo sau nó nếu ZF=0.

Bước 5: Đặt lệnh xử lý chuỗi thích hợp.

Ví dụ 1: Chuyển 100 byte từ ô nhớ nguồn đến ô nhớ đích.

```
        CLD         ;DF=0, SI và DI tự động tăng 1 sau mỗi lần lặp,
                    ;lệnh này được giới thiệu phần dưới.
        LEA SI, nguồn
        LEA DI, đích
        MOV CX, 50
REP     MOVSW   ; Tự động chuyển 100 byte.
```

Ví dụ 2: So sánh 10 byte của 2 vùng ô nhớ trong DS.

```
        CLD         ;DF=0, muốn DF=1 ta dùng lệnh STD.
        PUSH DS
        PUSH ES
        MOV SI, 7000H
        MOV DI, F000H
        MOV CX, 10
REPE    CMPSB
        JNZ NOTEQ       ;NOTEQ là một nhãn chương trình sẽ nhảy
                        ;tới nếu xuất hiện 2 byte không giống nhau.
```

Ví dụ 3: Tìm chữ 'A' trong chuỗi 100 ký tự.

```
        CLD             ;DF=0, muốn DF=1 ta dùng lệnh STD.
        MOV CX, 100
```

```
        LEA DI, chuỗi
        MOV AL, 'A'
REPNE SCASB     ;so sánh mỗi byte của chuỗi với ký tự 'A'.
        JCXZ NOTE   ;NOTE là một nhãn chương trình sẽ nhảy
                    ;tới nếu không tìm thấy 'A'.
                    ;(jump if CX equal zero)
```

14. Các lệnh khác

- CLC (clear carry flag): xóa cờ zero - CF.
- CLD (clear direction flag): xóa cờ hướng - DF.
- CLD (clear interrupt flag): xóa cờ ngắt - IF.
- CMC (complement carry flag): đổi ngược cờ CF.
- HLT (halt): dừng, CPU ngưng hoạt động.
- INT (interrupt): gọi ngắt.
- IRET (return from interrupt): trở về chương trình chính từ chương trình phục vụ ngắt.
- LOCK: khoá bus hệ thống.
- NOP (no operation): không có tác vụ.
- WAIT: đợi cho đến khi có xung ở chân TEST của CPU 8086.

III. CÁC KIỂU ĐỊNH VỊ CỦA CPU 8086

1. Định vị tức thì

Toán hạng tức thì nằm ngay sau mã tác vụ của lệnh nên việc truy xuất toán hạng rất nhanh chóng.

```
MOV AL, 15H  ;15H là toán hạng tức thì.
```

2. Định vị trực tiếp

Địa chỉ của toán hạng (ô nhớ trong) nằm ngay sau mã tác vụ của lệnh.

```
VarI DB 25H
……
……
MOV AL, VarI  ;đưa số 25H ở địa chỉ của VarI vào AL.
```

3. Định vị thanh ghi

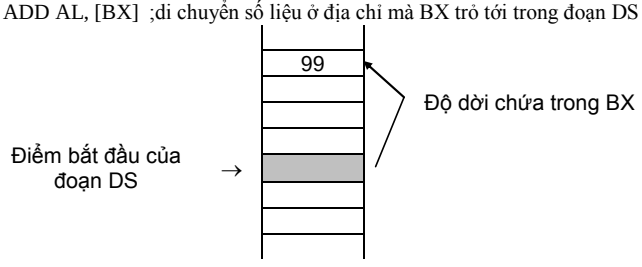
Định vị thanh ghi toán hạng nằm trên thanh ghi.

- Định vị trực tiếp thanh ghi.

```
ADD AL, BL    ;cộng 2 thanh ghi AL và BL đặt kết quả trong AL.
```

- Định vị gián tiếp thanh ghi.

```
ADD AL, [BX]  ;di chuyển số liệu ở địa chỉ mà BX trỏ tới trong đoạn DS vào thanh ghi AL.
```



4. Định vị nền

Địa chỉ toán hạng là tổng độ dời và thanh ghi nền BX hay BP.

MOV AL, [BX + 20H]

Di chuyển số liệu nằm ở ô nhớ BX+20H trong đoạn DS vào thanh ghi AL.

5. Định vị chỉ số

Địa chỉ của toán hạng là tổng độ dời và thanh ghi chỉ số SI hay DI.

MOV AL, [SI + 100H]

Di chuyển số liệu nằm ở ô nhớ SI+100H trong đoạn DS vào thanh ghi AL.

6. Định vị chỉ số và nền

Địa chỉ của toán hạng là tổng độ dời với thanh ghi chỉ số và thanh ghi nền.

MOV AL, [BX + SI + 50H]

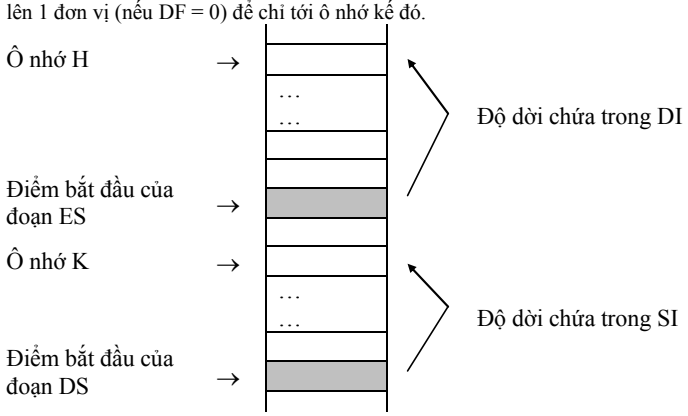
Di chuyển số liệu nằm ở ô nhớ BX + SI + 50H trong đoạn DS vào thanh ghi AL.

7. Định vị chuỗi

Trong định vị này ta dùng các lệnh mạnh xử lý chuỗi của 8086 (MOVSB, SCASB,…) dùng thanh ghi SI trỏ tới địa chỉ nguồn trong đoạn DS và thanh ghi DI trỏ tới địa chỉ đích trong đoạn ES. Sau mỗi lần thực hiện lệnh, thanh ghi SI và DI tự động tăng (hoặc giảm) 1 đơn vị - do ta qui định cờ hướng DF.

MOVSB

Lệnh này di chuyển 1 byte từ vị trí ô nhớ K đến vị trí ô nhớ H. Sau đó SI và DI tự động tăng lên 1 đơn vị (nếu DF = 0) để chỉ tới ô nhớ kế đó.



8. Định vị cửa vào ra

- Nếu địa chỉ cửa vào ra nằm trong khoảng từ 0 đến FFH, ta có thể dùng phép định vị trực tiếp.

  IN AL, 0F8H ;0F8H là địa chỉ của cổng vào ra.

- Nếu địa chỉ cửa vào ra lớn hơn FFH, ta có thể dùng phép định vị gián tiếp thanh ghi.

  MOV DX, 3F8H ;3F8H là địa chỉ của cổng vào ra.

  IN AL, DX

Chương 2
HỢP NGỮ

Hợp ngữ (Assembler) là chương trình dịch ra mã máy một chương trình gốc viết bằng từ gợi nhớ (mnemonic) của các lệnh mã máy. Lúc đầu, các chương trình hợp ngữ chỉ làm việc đơn thuần từ những từ gợi nhớ sang mã máy, nhưng dần dần các chương trình hợp ngữ cho phép dùng các nhãn, các ký hiệu biến đổi dạng lệnh, phân phối bộ nhớ, viết các macro nhằm giúp cho người lập trình viết các chương trình hợp ngữ dễ dàng hơn.

Sau đây, chúng ta sẽ đề cập đến các đặc tính tổng quát của hợp ngữ trước khi nghiên cứu hợp ngữ MASM dùng cho mã máy.

I. ĐẶC TÍNH TỔNG QUÁT CỦA HỢP NGỮ

1. Cấu trúc tổng quát của một lệnh hợp ngữ

Một lệnh của hợp ngữ thường có các vùng sau:

Tên	Từ gợi nhớ mã lệnh	Toán hạng	Chú thích

Ví dụ:

BATDAU: MOV DX, 3F8H ;3F8H là cổng vào ra nối tiếp.

Các vùng được sắp xếp trên một hàng lệnh và phân cách nhau bằng dấu chấm (:) hoặc dấu phẩy (,) hoặc chấm phẩy (;)… tuỳ theo lệnh của hợp ngữ.

a. Cấu trúc tổng quát vùng tên (name)

Vùng tên cho phép gán tên cho một địa chỉ hay một dữ liệu, lúc đó ta có thể dùng tên này để thay thế cho địa chỉ hay dữ liệu trên.

Mỗi tên chỉ xuất hiện một lần trong chương trình.

b. Cấu trúc tổng quát từ gợi nhớ mã lệnh

Đây là vùng duy nhất không thể thiếu của hàng lệnh. Vùng này có thể chứa một từ gợi nhớ mã lệnh hoặc một lệnh giả.

Khi gặp một từ gợi nhớ mã lệnh, hợp ngữ sẽ dịch nó sang mã máy.

Khi gặp một lệnh giả (còn gọi là hướng dẫn: directives) thì hợp ngữ không dịch thành mã nhị phân vì các lệnh giả chỉ giúp cho hợp ngữ định nghĩa ký hiệu, phân phối bộ nhớ, tạo bảng dữ liệu…

Thông thường, các chương trình hợp ngữ thường có các lệnh giả chủ yếu sau:

- ***ORG*** (origin: điểm gốc): Cho biết điểm bắt đầu của một đoạn chương trình hay một đoạn dữ liệu nào đó.
- ***EQU*** (equate: bằng nhau) hay DEFINE (define: định nghĩa): Cho phép đặt tên một dữ liệu nào đó.
- ***DS*** (define storage: định nghĩa vùng lưu trữ số liệu) hay RM (reserve memory: để dành vùng ô nhớ): Cho phép để dành vùng ô nhớ, để lưu trữ số liệu. Các lệnh giả DATA, DB (define byte), DW (define word) cũng cho phép để dành vùng ô nhớ.
- ***END***: Cho biết chấm dứt chương trình.

c. Cấu trúc tổng quát vùng toán hạng

Chứa các toán hạng mà lệnh mã máy cần.

d. Cấu trúc tổng quát vùng chú thích

Có thể có hoặc không, vùng này để dành cho người lập trình ghi các ghi chú, giải thích về câu lệnh.

2. Cấu trúc tổng quát của Macro

Macro là một nhóm lệnh nào đó được dùng nhiều lần nên ta gán cho nó một tên. Nhiều chương trình hợp ngữ cho phép dùng Macro.

Đối với Macro, mỗi lần chương trình chính gọi nó thì đoạn mã lệnh trong Macro được xen vào ngay mã lệnh gọi, không cần dùng lệnh CALL hay JUMP nên chương trình chính thực hiện nhanh hơn.

Tuy nhiên, mỗi lần gọi Macro thì mã lệnh trong Macro được xen vào trong chương trình chính làm cho chương trình ngày càng dài và chiếm nhiều ô nhớ trong bộ nhớ trong.

3. Cấu trúc tổng quát của chương trình con

Hợp ngữ thường cho phép dịch các chương trình con. Nó sẽ đánh dấu tham khảo chương trình con trong chương trình chính và chính chương trình liên kết (linker) sẽ gán địa chỉ của các chương trình con.

Một số loại hợp ngữ còn cho phép tạo một thư viện chương trình con.

Muốn gọi chương trình con thì chương trình chính phải dùng lênh CALL hay JUMP, do đó phải lưu địa chỉ trở về của chương trình chính ở ngăn xếp và làm chậm đi việc thực hiện chương trình chính.

Chương trình con chỉ chiếm một lượng ô nhớ nhất định. Khi chương trình chính chính gọi nó thì chương trình chính phải lưu địa chỉ trở về, sau đó nhảy đến địa chỉ của chương trình con và thực thi chương trình con, khi thực hiện xong thì lấy địa chỉ trở về để tiếp tục thực hiện chương trình chính.

4. Cấu trúc tổng quát của biến toàn cục (global), biến địa phương (local)

Các biến định nghĩa trong chương trình chính gọi là biến toàn cục. Các biến này có thể dùng trong chương trình chính, trong macro và trong các chương trình con.

Các biến được định nghĩa trong macro hay trong chương trình con gọi là các biến địa phương, chỉ được dùng trong nội bộ macro hay chương trình con khai báo nó.

5. Cấu trúc tổng quát của các bảng, thông báo mà hợp ngữ cung cấp cho người sử dụng

Đa số các hợp ngữ có thể cung cấp các bảng và thông báo cho người sử dụng các nội dung sau:

- Bảng liệt kê chương trình hợp ngữ và mã máy tương ứng.
- Bảng liệt kê các lỗi trong chương trình gốc.
- Bảng các tên được dùng trong chương trình gốc.
- Danh sách các tham khảo bên ngoài (các chương trình con, các biến dùng ở ngoài).
- Danh sách các macro, chương trình con và độ dài của chúng.

6. Cấu trúc tổng quát của hợp ngữ chéo (cross assembler)

Một hợp ngữ chạy trên một máy nào đó để dịch ra mã máy cho CPU khác với CPU trên máy mà hợp ngữ đó đang chạy, thì gọi là hợp ngữ chéo.

Ví dụ, hiện nay rất khó tìm 1 máy tính dùng CPU Z80. Muốn dịch một chương trình hợp ngữ thành mã máy của CPU Z80, người ta phải dùng chương trình hợp ngữ chéo chạy trên các máy IBM chẳn hạn để biên dịch ra mã máy cho CPU Z80 (dĩ nhiên chương trình mã máy sau khi dịch phải chạy trên máy tính hoặc vi mạch do CPU Z80 điều khiển).

II. HỢP NGỮ MASM - DÙNG CHO CPU 8086

1. Giới thiệu hợp ngữ MASM

Ta dùng các từ gợi nhớ mã lệnh, các lệnh giả, các ký hiệu… do hợp ngữ MASM qui định để viết ra một chương trình mà ta gọi là chương trình gốc (source file). Chương trình gốc được lưu trong đĩa từ với tập tin có đuôi là ASM. Hợp ngữ MASM sẽ dịch chương trình gốc thành chương trình đích có đuôi OBJ. Chương trình đích sẽ được nối kết LINK tạo ra chương trình chạy được có đuôi là EXE.

Chúng ta sẽ nghiên cứu các qui định, một số cú pháp thường dùng cho việc viết một chương trình hợp ngữ gốc.

2. Cấu trúc của một hàng lệnh hợp ngữ MASM

Một lệnh của hợp ngữ MASM thường có 4 vùng. Được tổ chức giống như tổ chức tổng quát của hợp ngữ.

Tên	Từ gợi nhớ mã lệnh	Toán hạng	Chú thích

Ví dụ:

```
BATDAU:    MOV DX, 3F8H     ;3F8H là cổng vào ra nối tiếp.
```

3. Tên trong hợp ngữ MASM

Trong hợp ngữ MASM, tên có thể là nhãn, biến, ký hiệu.
Tên có chiều dài là 31 ký tự và phải bắt đầu là chữ.

- ***Nhãn***: Dùng để đánh dấu một địa chỉ mà các lệnh như JUMP, CALL, LOOP cần đến. Nó cũng được dùng đến cho các lệnh giả LABEL hoặc PROC hoặc EXTRN.

  Ví dụ:

  ```
  NH:   MOV AX, DX      ;NH là nhãn đánh dấu địa chỉ ô nhớ.
  FOO LABEL NEAR        ;đặt tên cho địa chỉ ô nhớ theo sau lệnh giả.
  CTCON PROC FAR        ;địa chỉ bắt đầu của chương trình con.
  EXTRN NH FAR          ;cho biết NH nằm ngoài chương trình gốc.
  ```

- ***Biến***: Dùng làm toán hạng cho các lệnh hoặc các biểu thức. Biến tượng trưng cho địa chỉ nơi đó có giá trị mà ta cần.

  Ví dụ:

  ```
  TWO DB 2              ;biến TWO có giá trị là 2.
  ```

- ***Ký hiệu***: là một tên được định nghĩa để thay cho biểu thức, một từ gợi nhớ lệnh. Ký hiệu có thể dùng làm toán hạng trong biểu thức, trong lệnh hay trong lệnh giả.

  Ví dụ:

  FOO EQU 7H

  TOTO = 0FH

4. Từ gợi nhớ mã lệnh

a. Giới thiệu

Từ gợi nhớ mã lệnh dã được học ở chương I về tập lệnh bộ xử lý 8086 nên sau đây chúng ta chỉ nghiên cứu một số lệnh giả thường dùng của MASM.

Lệnh giả còn gọi là lệnh hướng dẫn (directive) dùng hướng dẫn chương trình dịch hợp ngữ về việc vào ra, về tổ chức bộ nhớ, về dịch chương trình với điều kiện, về điều khiển in danh sách và đối chiếu chéo, về các định nghĩa.

Lệnh giả được chia thành 4 nhóm và được trình bày dưới đây.

b. Nhóm lệnh giả liên quan đến bộ nhớ

- ***ASSUME***: Lệnh này cho biết một đoạn nào đó thuộc loại gì?

Ví dụ: ASSUME DS: DATA, CS: CODE dùng để báo cho hợp ngữ biết đoạn có tên DATA là đoạn DS và đoạn có tên CODE là đoạn CS.

ASSUME NOTHING báo cho hợp ngữ biết rằng không có tên đoạn nào được cho biết loại và như vậy mỗi lần liên hệ đến một nhãn, biến thì ta phải báo cho biết đoạn của chúng (ví dụ DS: NHAN cho biết NHAN thuộc đoạn DS).

- ***COMMENT***: Lệnh này cho ta ghi chú.

Ví dụ: COMMENT *Day la ghi chu* (dấu * là ký tự giới hạn).

Các lệnh dùng định nghĩa biến, khởi động vùng ô nhớ cho biến:

- ***DB*** (define byte: định nghĩa byte): dành ô nhớ trong để chứa từng byte.
- ***DW*** (define word: định nghĩa từ - 16bit): dành ô nhớ trong để chứa từng từ - mỗi từ tương đương 2 byte.
- ***DD*** (define double word: định nghĩa từng đôi từ): dành ô nhớ trong để chứa từng đôi từ - 4 byte.
- ***DQ*** (define qualword: định nghĩa từng 4 từ): dành ô nhớ trong để chứa từng 4 từ - 8 byte.
- ***DT*** (define ten byte: định nghĩa 10 byte): dành ô nhớ trong để chứa từng 10 byte.

Ta xem các ví dụvà phần giải thích sau:

```
MOTCHU DB ‘A’          ;dành một vị trí ô nhớ chứ số 65.
ARRAY DB 1, 2, 3, 4    ;định nghĩa 4 vị trí ô nhớ chứa 1, 2, 3, 4.
MES DB ‘CHAO BAN!’     ;dành 9 ô nhớ chứa chuỗi ‘CHAO BAN!’.
BUF DB 10 DUP(?)       ;dành 10 ô nhớ mà không khởi động các ô
                       ;nhớ đó nghĩa là các ô nhớ chứa bất kỳ số
                       ;nào.
BUF 100 DUP(‘A’)       ;dành 100 ô nhớ chứa toàn chữ ‘A’.
```

BSIZE DW 4*128 ;dành 2 ô nhớ để chứa số 512.

- ***END***: chấm dứt chương trình gốc.
- ***EQU*** (equate: bằng): gán trị cho tên.

  Ta dùng lệnh EQU để gán trị cho tên một lần, muốn gán đi lại nhiều lần ta dung lệnh = (bằng).

  Ví dụ: FOO EQU 2*10 ;gán trị 20 vào FOO.
- ***Dấu*** **=** (bằng): lệnh này giống như EQU nhưng cho phép ta gán lại nhiều lần.
- ***EVENT*** (chắn): làm cho thanh ghi đếm chương trình PC có nội dung là một số chắn.
- ***EXTRN*** (external: bên ngoài): cho biết một tên hay một ký hiệu đã được định nghĩa bên ngoài trong một modul khác với modul chương trình hiện tại.

  Ví dụ: EXTRN TAGN: NEAR, SO: WORD

  TAGN là một nhãn gần (2 byte), SO là một từ máy tính nằm ngoài modul hiện tại.
- ***GROUP*** (nhóm): gom các đoạn có tên khác nhau để gán chúng một tên mới.

  Ví dụ: NHOM123 GROUP NHOM1, NHOM2, NHOM3

  NHOM1, NHOM2, NHOM3 là tên 3 đoạn khác nhau, chúng được gom lại có tên là NHOM123.
- ***INCLUDE*** (bao gồm): Cho phép xen thêm một tập tin hợp ngữ vào tập tin hợp ngữ hiện hành.

  Ví dụ: INCLUDE C:\THEM.ASM

  Nội dung tập tin THEM.ASM trên đĩa C:\ được xen vào tậo tin hiện hành ngay lệnh giả INCLUDE.
- ***LABEL*** (nhãn): Cho phép đánh dấu một địa chỉ là địa chỉ của lệnh hay số liệu kê đó.

  Ví dụ:

  ```
  NHF  LABEL FAR          ;nhản xa, đánh dấu vị trí NH.
  NH:  MOV AX, DATA       ;nhản gần.
  MOV DS, AX
  …
  …
  CHB LABEL BYTE          ;đánh dấu vị trí CH có thể lấy từng byte.
  CH   DW 100 DUP(0)      ;chuỗi từng từ.
  …
  ```
- ***NAME*** (tên): Đặt tên cho một modul hợp ngữ.

  Ví dụ: NAME CURSR ;đặt tên cho modul là CURSR.

- ***ORG*** (origin: điểm gốc): Ấn định địa chỉ cho đoạn chương trình viết sau lệnh giả ORG.

  Ví dụ:

  ORG 100H

  MOV AX, Code
- ***PROC*** (procedure: thủ tục): Gán một tên cho chương trình con.

  Ví dụ:

  CTCON PROC NEAR

  MOV AX, CODE

  …

  …

  RET

  CTCON ENDP
- ***PUBLIC*** (public: công cộng): Dùng khai báo ký hiệu trong modul hiện hành mà các modul khác có thể sử dụng.

  Ví dụ:

  PUBLIC FOO, NH, TOTO
- ***RADIX***: Cho phép đổi cơ số, cơ số mặc nhiên là 10.

  Ví dụ:

  MOV BX, 0FFH ;tương đương với RADIX 16.
- ***RECORD*** (mẫu tin): Cho phép định nghĩa mẫu tin.

  Ví dụ:

  FOO RECORD CAO: 7, VUA: 3, THAP: 4

  Vậy FOO có chiều dài 16bit, 4bit thấp nhất có tên là THAP, 3bit kế có tên VUA, 7bit kế đó có tên là CAO. Còn 2 bít chưa được định nghĩa.
- ***STRUC*** (structure: cấu trúc): Giống như lệnh RECORD, chỉ khác là các vùng bên trong là byte.
- ***SEGMENT***: Cho phép định nghĩa đoạn.

  <***Tên***> **SEGMENT** [*align*] [*combine*] [‘*class*’]

  …

  …

  <***Tên***> **ENDS**

  Vùng [align] xác định đoạn bắt đầu như sau:

  Byte: Đoạn có thể bắt đầu ở địa chỉ bất kỳ.
  Word: Đoạn phải bắt đầu ở địa chỉ chẵn.
  Para: Đoạn phải bắt đầu với địa chỉ là bội của 16.
  Page: Đoạn phải bắt đầu với địa chỉ là bội của 256.

  Vùng [combine] xác định đoạn kết hợp với phân đoạn khác như sau:

  Public: Các đoạn cùng tên và cùng class được ghép nối với nhau khi liên kết.

Common: Các đoạn cùng tên và cùng class được phủ lắp lên nhau khi liên kết.
AT(biểu thức): Đoạn được đặt tại một địa chỉ là bội của 16 và được ghi trong biểu thức.
STACK: Giống như Public, tuy nhiên con trỏ ngăn xếp SP chỉ vào địa chỉ đầu tiên của ngăn xếp đầu tiên.
Private hoặc không để gì hết cho vùng combine thì các đoạn cùng tên và cùng class không được ghép vào nhau.

c. Nhóm lệnh giả về dịch (complie) có điều kiện

Các lệnh giả về dịch có điều kiện nhằm báo cho hợp ngữ dịch hay không dịch một nhóm lệnh nào đó nếu một điều kiện nào đó được thỏa.

IF ×××× [đối số]
…
…
[**ELSE**]
…
…
ENDIF

IF ×××× [đối số] có các hình thức sau:

- ***IFE*** <biểu thức>: nếu biểu thức tính ra bằng 0 thì đoạn chương trình IFE được dịch, nếu biểu thức tính ra khác 0 thì đoạn chương trình sau lệnh giả ELSE được dịch (nếu có lệnh giả ELSE).
- ***IF1***: Nếu hợp ngữ đang dịch lần 1 thì đoạn chương trình sau IF1 được dịch.
- ***IF2***: Nếu hợp ngữ đang dịch lần 2 thì đoạn chương trình sau IF2 được dịch.
- ***IFDEF*** <ký hiệu>: Nếu ký hiệu không được định nghĩa thì dịch chương trình sau IFDEF.
- ***IFB*** <đối số>: Nếu đối số là khoảng trắng hoặc không có đối số thì dịch đoạn chương trình sau IFB.
- ***IFNB*** <đối số>: Nếu có đối số thì dịch đoạn chương trình sau IFNB.
- ***IFIDN*** <đối số 1>, <đối số 2>: Nếu <đối số 1> bằng <đối số 2> thì đoạn chương trình sau IFIDN được dịch.
- ***IFDIF*** <đối số 1>, <đối số 2>: Nếu <đối số 1> khác <đối số 2> thì đoạn chương trình sau IFIDF được dịch.
- ***ENDIF***: Đánh dấu hết đoạn chương trình sau **IF ××××.**

d. Nhóm lệnh giả về MACRO

Lệnh giả MACRO giúp ta viết một đoạn chương trình mà ta có thể xen vào bất cứ nơi nào trong chương trình hợp ngữ bằng cách viết tên MACRO mà ta muốn gọi.

Ta định nghĩa một MACRO như sau:

```
<Tên> MACRO [tham số]
…
…
ENDM        ;chấm dứt MACRO
```

Ví dụ: Ta viết MACRO có tên là TEST như sau:

```
TEST MACRO X, Y, Z
        MOV AX, X
        MOV BX, Y
        MOV CX, Z
ENDM
```

Như vậy, trong một chương trình nào đó ta gọi TEST như sau:

```
TEST 10, 20, 30
```

Thì câu lệnh tương đương như sau:

```
MOV AX, 10
MOV BX, 20
MOV CX, 30
```

Bên trong MACRO ta có thể dùng các lệnh giả LOCAL, EXITM.

Lệnh EXITM dùng để thoát khỏi MACRO trước khi có lệnh chấm dứt MACRO.

Lệnh giả LOCAL cũng có thể dùng bên trong MACRO, lệnh này giúp ta định nghĩa các nhãn trong MACRO mà ta muốn gọi nhiều lần.

Ví dụ: Ta thiết kế hàm WAIT như sau:

```
WAIT MACRO count
        MOV CX, count
NEXT:  LOOP NEXT
ENDM
```

MACRO WAIT chỉ có thể gọi duy nhất một lần vì nhãn NEXT chỉ xuất hiện trong chương trình 1 lần.

Muốn có thể gọi lệnh WAIT nhiều lần, ta thiết kế lại lệnh WAIT như sau:

```
WAIT MACRO count
        LOCAL NEXT
        MOV CX, count
NEXT:  LOOP NEXT
ENDM
```

e. Nhóm lệnh giả về liệt kê (listing)

Nhóm này dùng để điều khiển liệt kê lên máy in, xem các đề tựa ở đầu trang.

- ***PAGE*** số hàng, số cột
  Ví dụ: PAGE 58, 60 ;mỗi trang liệt kê có 58 hàng và 60 cột.
- ***TITLE*** (đề tựa): Cho phép đặt đề tựa.
  Ví dụ: TITLE Chuong Trinh Hop Ngu

- ***SUBTTL*** (subtitle: đề tựa con): Cho phép liệt kê đề tựa con ở mỗi đầu trang.
- ***% OUT*** <văn bản>: Văn bản được liệt kê khi hợp ngữ dịch chương trình.
- ***XLIST***: Không cho liệt kê.
- ***XALL***: Liệt kê mã do MACRO tạo nên.
- ***LALL***: Liệt kê toàn bộ MACRO.
- ***SALL***: Không liệt kê MACRO.
- ***CREF***: Cho liệt kê bảng đối chiếu chéo.
- ***XCREF***: Không cho phép liệt kê bảng đối chiếu chéo.

f. Nhóm lệnh giả về toán hạng và toán tử

Có 3 loại toán hạng: tức thì, thanh ghi và ô nhớ.

- Toán hạng tức thì có thể là một số hoặc một ký hiệu đã được gán một số bằng các lệnh giả EQU và dấu = (bằng).
  Ví dụ:

```
SL EQU 15
        MOV CL, SL
        MOV AL, 20
```

- Toán hạng ô nhớ tượng trưng một địa chỉ ô nhớ. Nó luôn luôn là độ dời đối với một địa chỉ bắt đầu của đoạn tương ứng
  Ví dụ:

```
FOO DW 0FEFEH
MOV AX, FOO          ;FOO là địa chỉ của số liệu.
MOV FOO, AX
```

  FOO+5, FOO[5], 5[FOO] là tương đương nhau và chỉ đến địa FOO cộng 5.
  5 [BX] [SI], [BX+5][SI], [BX] 5 [SI] là tương đương với [BX+SI+5] trong định dạng nền + chỉ số.

Các toán tử có thể chia làm 4 loại: toán tử thuộc tính (attribute), toán tử số học, toán tử quan hệ (relational), toán tử logic.

i. Toán tử thuộc tính

- ***PTR*** (pointer: con trỏ): Dùng để thay đổi kiểu của các địa chỉ.
  Ví dụ: CALL Word PTR [BX+SI]
  [BX+SI] trỏ tới một byte trong ô nhớ nhưng ta muốn lấy 2 byte bắt đầu của chương trình con nên ta dùng WORD PTR.
- ***Dấu :*** (hai chấm): Dùng để thay đổi đoạn mặc nhiên.
  Ví dụ: MOV AX, ES:[BX+SI]
  [BX+SI] trỏ tới số liệu trong DS nhưng ta muốn lấy trong đoạn ES nên viết ES:[BX+SI].
- ***SHORT***: Dùng để thay đổi kiểu mặc nhiên là NEAR của lệnh JMP và báo cho hợp ngữ biết chỉ nhảy trong vòng –128 đến +127 so với vị trí lệnh nhảy JMP.

- ***THIS***: Tạo một toán hạng có giá trị tuỳ thuộc vào tham số của THIS.
  Ví dụ:
  NH EQU THIS BYTE tương đương với NH LABEL BYTE.
  SCH = THIS NEAR tương đương với SCH LABEL NEAR.
- ***SEG***: Cho ta giá trị của đoạn, của một nhãn hay biến.
  Ví dụ: MOV AX, SEG TENB
  Đưa vào AX giá trị của đoạn chứa TENB.
- ***OFFSET***: Cho độ dời của một nhãn hay biến.
  Ví dụ: MOV DX, OFFSET TENCHUOI
  Đưa vào DX độ dời của biến TENCHUOI, lệnh giả tương đương với lệnh giã này là LEA DX, TENCHUOI.
- ***TYPE*** <biến>: Cho biết số byte mà loại biến đó chiếm (BYTE=1, WORD=2, DWORD=4,…).
  Ví dụ: MOV AX, (TYPE NH) PTR [BX+SI]
  Với NH là một nhãn.
- ***LENGTH***: Cho biết số phần tử của một biến.
  Ví dụ:
  FOO DW 100 DUP(?)
  MOV CX, LENGTH FOO ;CX chứa 100.
- ***SIZE***: Cho biết số byte của một biến.
  Ví dụ:
  FOO DW 100 DUP(?)
  MOV CX, SIZE FOO ;CX chứa 200.

ii. Toán tử số học

- Các toán tử thông dụng như: + (cộng), – (trừ), * (nhân), / (chia).
- Dấu trừ đứng trước một số để chỉ đó là số âm.
- ***MOD***: Chia lấy số dư.
  Ví dụ: MOV AX, 13 MOD 4 ;AX chứa số 1.
- ***SHR*** (shift right): dịch phải.
  Ví dụ: MOV AX, 00011000B SHR 3 ;AX chứa số 00000011B.
- ***SHL*** (shift left): dịch trái.
  Ví dụ: MOV AX, 00011000 SHL 3 ;AX chứa số 11000000B.

iii. Toán tử quan hệ

Toán tử quan hệ dùng trong so sánh hai số hạng và thường được dùng trong việc dịch chương trình có điều kiện.

- ***EQ*** (<toán hạng 1> EQ <toán hạng 2>): Nếu toán hạng 1 bằng toán hạng 2 thì biểu thức cho giá trị đúng (true).
- ***NE*** (not equal: không bằng nhau): Trả về TRUE nếu 2 toán hạng không bằng nhau.

- ***LT*** (less than: nhỏ hơn): Trả về true nếu toán hạng bên trái LT nhỏ hơn toán hạng bên phải LT.
- ***LE*** (less than or equal: nhỏ hơn hoặc bằng nhau): Giống như LT nhưng thêm điều kiện bằng nhau.
- ***GT*** (greater than: lớn hơn): Trả về true nếu toán hạng bên trái LT lớn hơn toán hạng bên phải LT.
- ***GE*** (greater than or equal: lớn hơn hoặc bằng nhau): Giống như GT nhưng thêm điều kiện bằng nhau.

iv. Toán tử logic

Toán tử logic so sánh hai toán hạng từng bit một.

- ***NOT***: Trả về true nếu toán hạng bên trái và bên phải khác nhau.
- ***AND***: Trả về true nếu cả 2 toán hạng đều là true.
- ***OR***: Trả về true khi một trong hai (hoặc cả hai) toán hạng là true.
- ***XOR***: Trả về true khi hai toán hạng khác nhau (toán hạng true, toán hạng còn lại false).

III. LẬP TRÌNH DÙNG HỢP NGỮ MASM

1. Giới thiệu

Trong lập trình dùng hợp ngữ MASM, ta thường sử dụng các ngắt có sẵn trong ROM-BIOS (basic output input system: hệ thống vào ra cơ bản) hoặc trong DOS. Toàn bộ các ngắt này được nhiều sách lập trình hợp ngữ nêu lên đầy đủ. Trong giáo trình này ta chỉ nêu một vài ngắt thường dùng trong DOS.

2. Một số ngắt thường dùng trong HĐH MS-DOS

Các ngắt của DOS được đánh số từ 20H đến 27H. Ngắt đặt biệt quan trọng là ngắt 21H.

INT 20H: Chấm dứt chương trình, trở về DOS từ một chương trình có đuôi chấm COM.

INT 23H: Chận CONTROL – BREAK.

Thông thường, một chương trình chạy trong DOS, ta có thể dùng tổ hợp phím CONTROL – BREAK để kết thúc một cách đột nhiên. Tổ hợp phím này làm CPU nhảy đến chương trình phục vụ ngắt 23H, sau đó đến một thủ tục của DOS để kết thúc chương trình đang chạy và trở về DOS (dừng đột ngột).

Nếu không muốn cho phép chương trình kết thúc sớm bằng phím CONTROL – BREAK, ta có thể cho ngắt 23H trỏ tới một chương trình đặc biệt nào đó, để thông báo lên màn hình rằng không cho chương trình đang chạy kết thúc sớm chẳn hạn.

INT 27H: Kết thúc và đặt thường trú.

Ngắt này cho phép để trong bộ nhớ trong một chương trình nào đó cho đến khi tắt máy. Để sử dụng ngắt này, ta cho chạy chương trình gồm 2 phần. Một phần phụ trách việc nội trú cho phần kia.

INT 21H: Ngắt 21H có nhiều hàm, muốn gọi hàm nào đó của ngắt 21H, ta đặt số thứ tự của hàm đó vào thanh ghi AH và gọi INT 21H.

- ***Hàm 1***: Đợi đọc một ký tự từ bàn phím và in ra màn hình. Khi một phím được ấn thì ký tự tương ứng trong phím đó được lưu trong thanh ghi AL. Nếu phím được ấn là phím đặc biệt thì AL=0. Dùng tổ hợp phím CONTROL – BREAK để kết thúc công việc này.
- ***Hàm 2***: Đưa một ký tự ra màn hình. Ký tự cần đưa ra màn hình chứa trong DL.
- ***Hàm 3***: Đọc vào từ cổng nối tiếp. Ký tự được chứa trong AL.
- ***Hàm 4***: Đưa ký tự trong DL ra cổng nối tiếp.
- ***Hàm 5***: Đưa ký tự trong DL ra máy in.
- ***Hàm 6***: Hàm này có thể thực hiện cả vào lẫn ra.

  Muốn nhập, ta cho DL = FFH. Sau khi thực hiện hàm 6, nếu cờ ZF = 0 thì AL chứa ký tự mới nhập vào, nếu cờ ZF = 1 thì không có ký tự nào được nhập vào.

  Nếu DL chứa 1 số khác FFH thì ký tự chứa trong DL được đưa ra màn hình.
- ***Hàm 7***: Giống hàm 1 nhưng nó không in ra màn hình ký tự vừa nhập vào, không thể dùng CONTROL – BREAK để kết thúc.
- ***Hàm 8***: Giống hàm 1 nhưng nó không in ra màn hình ký tự vừa nhập ra màn hình.
- ***Hàm 9***: Đưa ra màn hình chuỗi ký tự đặt trong DS:DX (nghĩa là chuỗi ký tự đó nằm trong đoạn DS và độ dời của nó nằm trong DX). Lưu ý, chuỗi ký tự phải kết thúc bằng dấu '$'.
- ***Hàm 0AH***: Đọc vào vùng đệm bàn phím.

  Trước khi dùng hàm này ta phải tạo ra một vùng đệm trong bộ nhớ. Byte đầu tiên của xác định số ký tự tối đa có thể nhập vào, byte thứ 2 để cho DOS ghi số ký tự thực sự mà người dùng đã nhập vào. Các byte kế tiếp chứa các ký tự sẽ nhập. Địa chỉ vùng đện đặt trong DS:DX.
- ***Hàm 0BH***: Cho trạng thái bàn phím. Sau khi thực hiện hàm nếu AL=0 thì không có ký tự nào đang đợi, nếu AL khác 0 thì có một ký tự đang đợi CPU đọc vào.

  Tiếp theo hàm 0BH là một số hàm thực hiện công việc FCB (file control block) đã lỗi thời nên không còn nói đến. Thay vào đó ta dùng các hàm tương đương là thẻ tập tin (file handle).
- ***Hàm 25H***: Đổi vectơ ngắt, ta đặt trong DS:DX giá trị mới của địa chỉ vectơ ngắt, trong AL ta để số của ngắt.
- ***Hàm 30H***: Lấy ấn bản (version) của DOS. Sau khi thực hiện hàm này thì AL chứa phần chính, AH chứa phần phụ của ấn bản.
- ***Hàm 31H***: Kết thúc và ở lại thường trú. Hàm này giống ngắt INT 27H. Ta đặt trong thanh ghi DX số paraghaph (16 bytes) của phần tập tin muốn giữ thường trú.
- ***Hàm 35H***: Lấy giá trị của vectơ ngắt. Ta để trong AL số của ngắt. Sau khi thực hiện hàm này thì ES:BX chứa địa chỉ của vectơ ngắt.

- ***Hàm 3CH***: Tạo tập tin mới. Cặp thanh ghi DS:DX trỏ tới tên của tập tin mới. Cuối chuỗi ta phải có số 0. CX chứa thuộc tính của tập tin như sau: 0 - bình thường, 1 - chỉ đọc, 2 - ẩn, 4 tập tin hệ thống. Sau khi thực hiện thì thẻ tập tin được chứa trong AX. Ta phải giữ nó lại vì mỗi lần truy xuất tập tin ta phải cần thẻ tập tin.
- ***Hàm 3DH***: Mở 1 tập tin đã có trên đĩa. Cặp thanh ghi DS:DX trỏ tới tên tập tin (có 0 ở cuối chuỗi tên). Ta đặt trong AL kiểu truy cập của tập tin như sau: 0-chỉ đọc, 1-chỉ ghi, 2-đọc và ghi. Thẻ tập tin nằm trong AX.
- ***Hàm 3EH***: Đóng tập tin. Thẻ tập tin được đóng nằm trong BX.
- ***Hàm 3FH***: Đọc từ tập tin hoặc thiết bị, cặp thanh ghi DS:DX trỏ đến vùng bộ nhớ mà dữ liệu sẽ chuyển tới. BX chứa thẻ tập tin. Trước khi đọc phải mở tập tin và CX chứa số byte cần đọc. Sau khi thực hiện hàm thì AX chứa số byte đọc vào.

  Ta cũng có thể dùng hàm này để gán thẻ cho ngoại vi vì DOS cũng gán thẻ cho ngoại vi.
- ***Hàm 40H***: Ghi lên tập tin hoặc thiết bị, trước hết phải mở tập tin.

  DS:DX trỏ đến vùng chứa dữ liệu trong bộ nhớ.

  BX chứa thẻ tập tin.

  CX chứa số byte cần ghi lên đĩa.

  Có thể dùng hàm này để gởi dữ liệu ra ngoại vi.
- ***Hàm 41H***: Xoá tập tin trên đĩa. DS:DX chứa đến tên tập tin muốn xoá, ta không thể xoá tập tin đang mở hoặc tập tin có thuộc tính chỉ đọc.
- ***Hàm 43H***: Đọc hoặc thay đổi thuộc tính tập tin.

  Đọc thuộc tính:

  Đặt AL = 0
  DS:DX trỏ đến tên tập tin.
  Sau khi thực hiện hàm, thuộc tính nằm trong CX.

  Đổi thuộc tính:

  Đặt AL = 1
  CX chứa thuộc tính mới.
  DS:DX trỏ đến tên tập tin.
- ***Hàm 4CH***: Kết thúc chương trình và trở về DOS (cả 2 loại tập tin có đuôi .COM và .EXE).
- ***Hàm 56H***: Đổi tên tập tin.

  DS:DX trỏ tới tên cũ.

  ES:DI trỏ tới tên mới.

3. Cấu trúc một chương trình hợp ngữ MASM

Một chương trình hợp ngữ có thể gồm nhiều modul. Các modul có thể viết riêng lẽ bằng một chương trình xử lý văn bản và dịch riêng lẽ bằng MASM để cho ra các chương trình đích. Các chương trình đích sẽ được trình liên kết nối lại thành một chương trình chạy được có đuôi chấm EXE.

Mỗi modul có thể viết như sau:

```
PAGE 60, 132
TITLE       (Tiêu đề vị trí này)
            (Các phát biểu EXTRN hay PUBLIC nếu có)
MCR MACRO
            (viết macro)
ENDM
STACK SEGMENT PARA STACK 'STACK'
            DB 64 DUP(?)
STACK ENDS
DSEG SEGMENT PARA PUBLIC DATA
            (dành ô nhớ cho dữ liệu ở đây)
DSEG ENDS
CSEG SEMENT PARA PUBLIC 'CODE'
            ASSUME CS: CSEG, DS: DSEG
BATDAU:
      MOV AX, DSEG
      MOV DS, AX
      (chương trình chính)
      …
      …
CTC         PROC NEAR
      (viết chương trình con)
CTC ENDP
CSEG ENDS
END BATDAU    ;Chấm dứt chương trình và chọn địa chỉ bắt đầu của
              ;chương trình là BATDAU
```

4. Tập tin đuôi COM và tập tin đuôi EXE

a. Giới thiệu

Hợp ngữ MASM cho phép tạo tập tin đuôi COM dùng cho các chương trình nhỏ và tập tin đuôi EXE để xây dựng các chương trình lớn.

b. Đặc điểm của tập tin đuôi COM

- Chỉ có một đoạn duy nhất, các thanh ghi CS, DS, SS đều có chung một trị số.
- Kích thước tối đa là 64KB.
- Tập tin COM được nạp vào bộ nhớ trong và thực hiện nhanh hơn tập tin EXE.

  Khi DOS thực hiện tập tin COM, nó tạo vùng ô nhớ từ độ dời từ 0 đến 0FFH để chứa thông tin cần thiết cho DOS. Vùng này gọi là vùng PSP

(program segment prefix). Vì thế địa chỉ bắt đầu của tập tin COM là từ độ dời 100H.

Tất cả các thanh ghi đoạn đều phải chỉ đến PSP.

- Dạng điển hình của tập tin COM như sau:

Ví dụ chương trình gốc HELLO.ASM

```
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG, ES: CSEG, SS: CSEG
        ORG 100H
BATDAU:
        MOV AX, DSEG
        MOV DS, AX
        MOV AH, 09H
        MOV DS, offset Mess
        INT 21H
        INT 20H
Mess BD ‘Hello!$’
CSEG ENDS
END BATDAU
```

c. Đặc điểm của tập tin đuôi EXE

- Chương trình lớn và nằm ở nhiều đoạn khác nhau.
- Có thể gọi các chương trình con dạng Far.
- Kích thước tập tin tuỳ ý.
- Có Header ở đầu tập tin để chứa các thông tin cần thiết cho tập tin EXE. Tập tin EXE không dùng lệnh ORG 100H ở đầu chương trình.
- Dạng điển hình của tập tin EXE như sau:

Ví dụ chương trình gốc HELLO.ASM

```
DULIEU SEGMENT
        THONGBAO DB 'HELLO!$'
DULIEU ENDS
MALENH SEGMENT
        ASSUME CS: MALENH, DS: DULIEU
BATDAU:
        MOV AX, DULIEU
        MOV DS, AX

        MOV DX, Offset THONGBAO
        MOV AH, 09H
        INT 21H

        MOV AH, 4CH
        INT 21H
MALENH    ENDS
```

```
END  BATDAU
```

d. Các bước biên dịch chương trình trong hợp ngữ MASM

- Dùng một chương trình soạn thảo (Notepad chẳn hạn) để soạn thảo mã nguồn và lưu lại tên tập tin có phần mở rộng là ASM. Ví dụ **Hello.asm**.
- Dùng hợp ngữ MASM để dịch **Hello.asm** thành chương trình đích **Hello.obj**. Như trên, ta đánh lệnh: **Masm Hello;** ↵ (lưu ý có dấu ; sau lệnh biên dịch).
- Dùng chương trình LINK để biến **Hello.obj** thành chương trình thi hành được: **Hello.exe**. Như trên, ta đánh tiếp lệnh: **Link Hello;** ↵ (lưu ý có dấu ; sau lệnh biên dịch). Lúc này trên đĩa đã có chương trình Hello.exe ta có thể thực thi chương trình này từ DOS bằng cách đánh vào tên chương trìh vào ấn phím Enter. Như trên ta đánh: **Hello** ↵
- Cho tập tin có đuôi COM thì ngoài 2 lệnh trên ta phải dùng lệnh ngoại trú của DOS là EXE2BIN để biến đổi EXE thành COM. Như trên ta đánh: **Exe2Bin Hello.exe Hello.com** ↵

Mục lục

Chương 1 TỔ CHỨC BỘ XỬ LÝ INTEL 8086 1
I. BỘ XỬ LÝ (CPU) INTEL 8086 1
1. Tổ chức tổng quát 1
2. Các thanh ghi của 8086 2
3. Tổ chức bộ nhớ trong 3
4. Sự phân đoạn trong bộ nhớ trong 3
5. Địa chỉ các ngoại vi 4
6. Các chân của vi mạch 8086 4
II. CÁC LỆNH THƯỜNG DÙNG CỦA CPU 8086 6
1. Giới thiệu 6
2. Nhóm di chuyển số liệu 6
3. Nhóm lệnh chuyển địa chỉ 7
4. Nhóm lệnh chuyển cờ hiệu (thanh ghi trạng thái) 7
5. Nhóm lệnh vào ra ngoại vi 8
6. Nhóm lệnh điều khiển 8
7. Nhóm lệnh so sánh 8
8. Nhóm lệnh vòng lặp 9
9. Nhóm lệnh gọi chương trình con 9
10. Nhóm lệnh tính toán số học 10
11. Nhóm lệnh dịch chuyển và quay 11
12. Nhóm lệnh logic 13
13. Nhóm lệnh xử lý chuỗi 13
14. Các lệnh khác 15
III. CÁC KIỂU ĐỊNH VỊ CỦA CPU 8086 15
1. Định vị tức thì 15
2. Định vị trực tiếp 15
3. Định vị thanh ghi 15
4. Định vị nền 16
5. Định vị chỉ số 16
6. Định vị chỉ số và nền 16
7. Định vị chuỗi 16
8. Định vị cửa vào ra 16
Chương 2 HỢP NGỮ 17
I. ĐẶC TÍNH TỔNG QUÁT CỦA HỢP NGỮ 17
1. Cấu trúc tổng quát của một lệnh hợp ngữ 17
2. Cấu trúc tổng quát của Macro 18
3. Cấu trúc tổng quát của chương trình con 18
4. Cấu trúc tổng quát của biến toàn cục (global), biến địa phương (local) 18
5. Cấu trúc tổng quát của các bảng, thông báo mà hợp ngữ cung cấp cho người sử dụng 18
6. Cấu trúc tổng quát của hợp ngữ chéo (cross assembler) 19
II. HỢP NGỮ MASM - DÙNG CHO CPU 8086 19
1. Giới thiệu hợp ngữ MASM 19
2. Cấu trúc của một hàng lệnh hợp ngữ MASM 19
3. Tên trong hợp ngữ MASM 19

4. Từ gợi nhớ mã lệnh 20
III. LẬP TRÌNH DÙNG HỢP NGỮ MASM 27
1. Giới thiệu 27
2. Một số ngắt thường dùng trong HĐH MS-DOS 27
3. Cấu trúc một chương trình hợp ngữ MASM 29
4. Tập tin đuôi COM và tập tin đuôi EXE 30