LẬP TRÌNH HỆ THỐNG MẠNG

MẠNG MÁY TÍNH

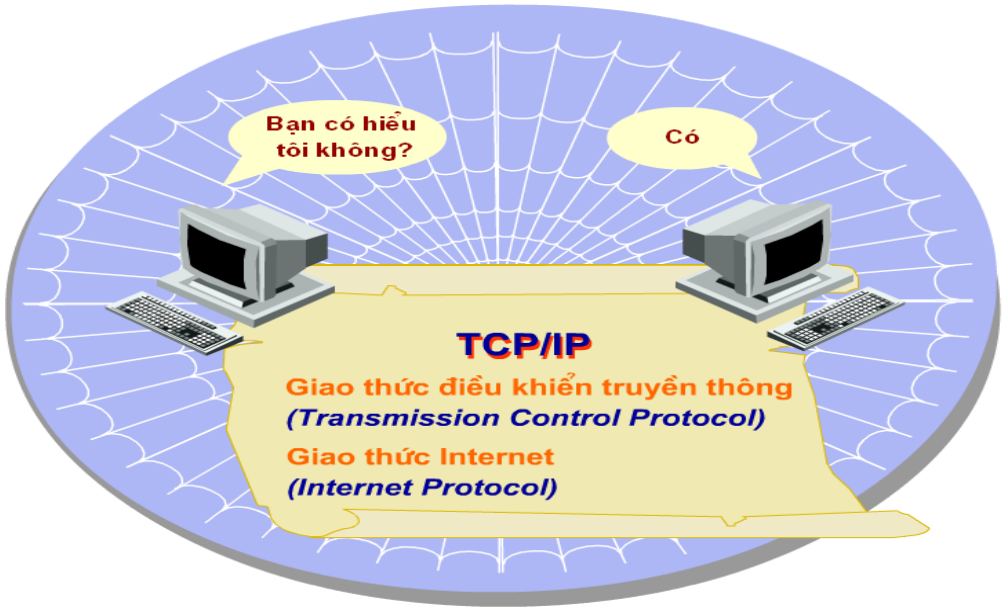
Nguyễn Hữu Thể

Nội dung

1. Giao thức TCP/IP
2. Cách chuyển dữ liệu trên mạng
3. Kết nối Internet
4. Lập trình mạng dùng Socket

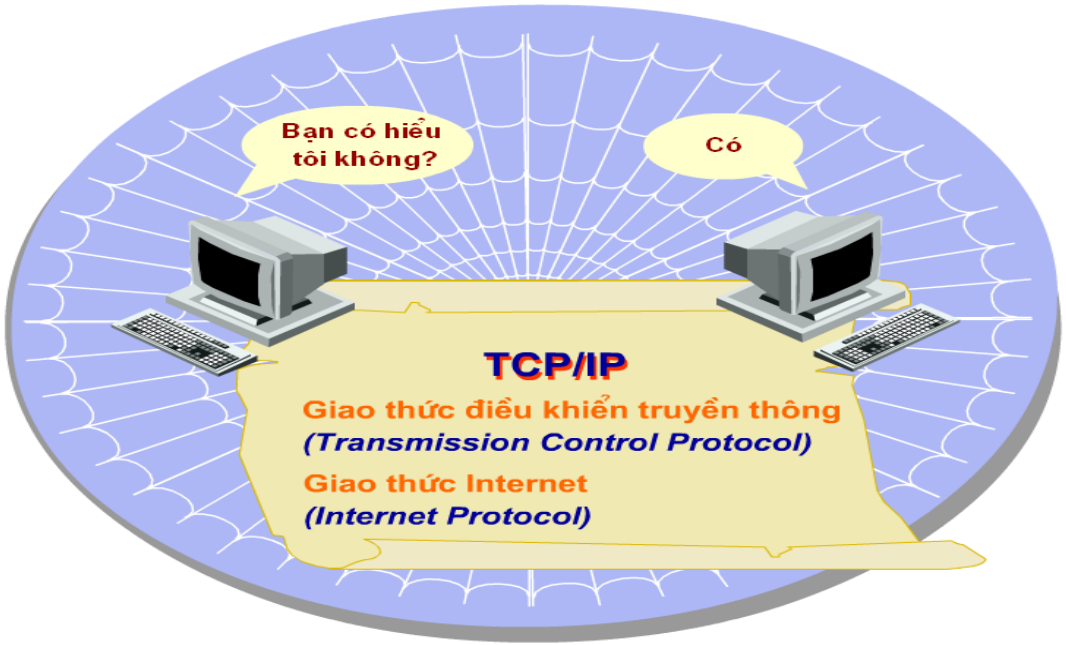
1. Giao thức TCP/ IP

- Mô hình TCP/IP ra đời trong những năm 1970, được phát triển dựa trên mô hình OSI.
- Giao thức TCP/IP cung cấp các phương tiện truyền thông liên mạng giúp cho các hệ thống mạng khác nhau có thể làm việc với nhau.
- TCP/IP được sử dụng trên mạng Internet.



Tổng quan về TCP/ IP

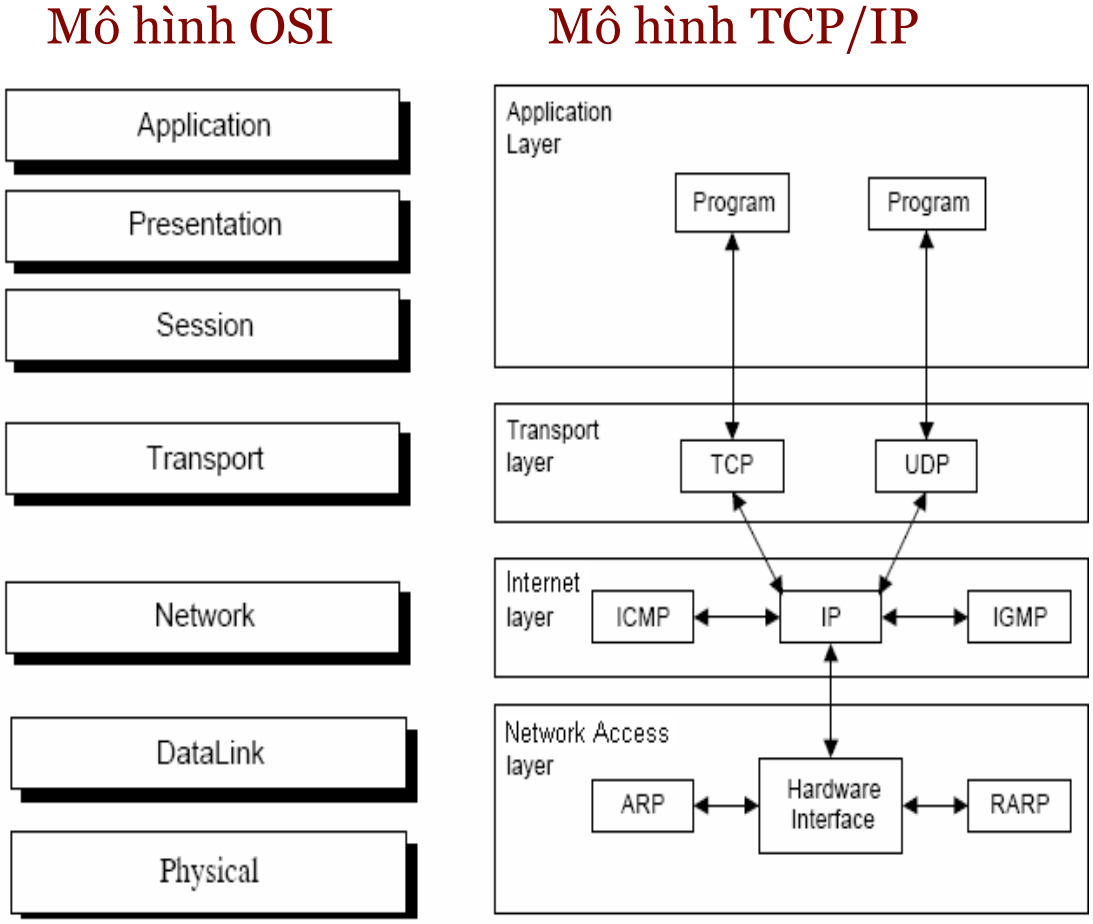
- TCP (Transmission Control Protocol) là giao thức thuộc tầng vận chuyển
- IP (Internet Protocol) là giao thức thuộc tầng mạng của mô hình OSI.



Các tầng của mô hình TCP/IP

Mô hình TCP/IP gồm có 4 tầng:

- ứng dụng (Application),
- vận chuyển (Transport) ,
- tầng Internet (Internet),
- truy xuất mạng(NetworkAccess)



các tầng của TCP/IP so với 7 tầng tương ứng của OSI.

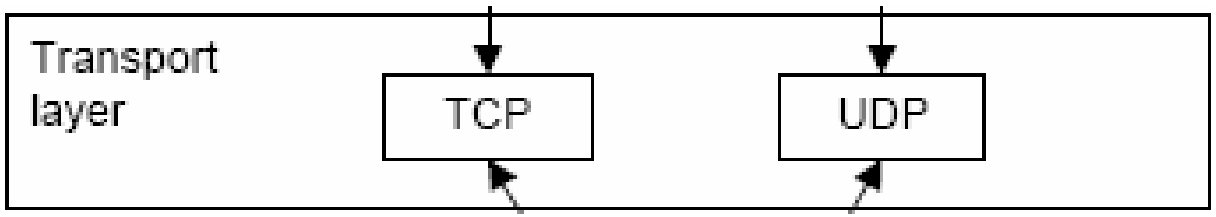
Tầng ứng dụng (Application Layer)

- Trao đổi thông tin người dùng.
- Một số giao thức thông dụng của lớp này là:

Telnet	Tele Comunication	Dịch vụ truy cập từ xa.
FTP	File Transfer Protocol	Dịch vụ truyền File.
SMTP	Simple Mail Transfer Protocol	Dịch vụ truyền thư đơn giản.
DNS	Domain Name System	Hệ thống tên miền
SNMP	Simple Network Management Protocol	Giao thức quản lý mạng đơn giản
RPC	Remote Procedure Call	Thủ tục gọi từ xa
RIP	Routing Information Protocol	Giao thức định tuyến thông tin
TCP	Transmission Control Protocol	Giao thức TCP
UDP	User Datagram Protocol	Giao thức dữ liệu của người dùng.
IP	Internet Protocol	Giao thức IP
ICMP	Internet Control Message Protocol	G.thức kiểm soát message giữa các mạng.
FDDI	Fiber Distributed Data Multiplexing	

Tầng vận chuyển (Transport Layer)

- Quản lý và kiểm tra truyền thông giữa các máy tính.
- Chuyển tiếp dữ liệu nhận được lên lớp trên hoặc xuống lớp dưới.
- Sử dụng cổng (port) để xác định ứng dụng nào của tầng trên sẽ nhận dữ liệu.
- Hai giao thức chính là :
  - TCP (Transmission Control Protocol)
  - UDP (User Datagram Protocol)



Tầng vận chuyển (Transport Layer)

TCP (Transmission Control Protocol) :

- Cung cấp dịch vụ truyền thông tin cậy và hướng kết nối
- Các ứng dụng cần truyền **dữ liệu lớn** tại một thời điểm
- Ứng dụng cần cơ chế **xác nhận** dữ liệu.
- **Hướng kết nối** gồm ba giai đoạn :
  - thiết lập kết nối,
  - truyền dữ liệu,
  - giải phóng kết nối.

Tầng vận chuyển (Transport Layer)

UDP (User Datagram Protocol) :

- Cung cấp dịch vụ truyền thông không kết nối
- **Không** đảm bảo truyền dữ liệu thành công.
- Thường chỉ truyền một lượng **dữ liệu nhỏ** tại một thời điểm.

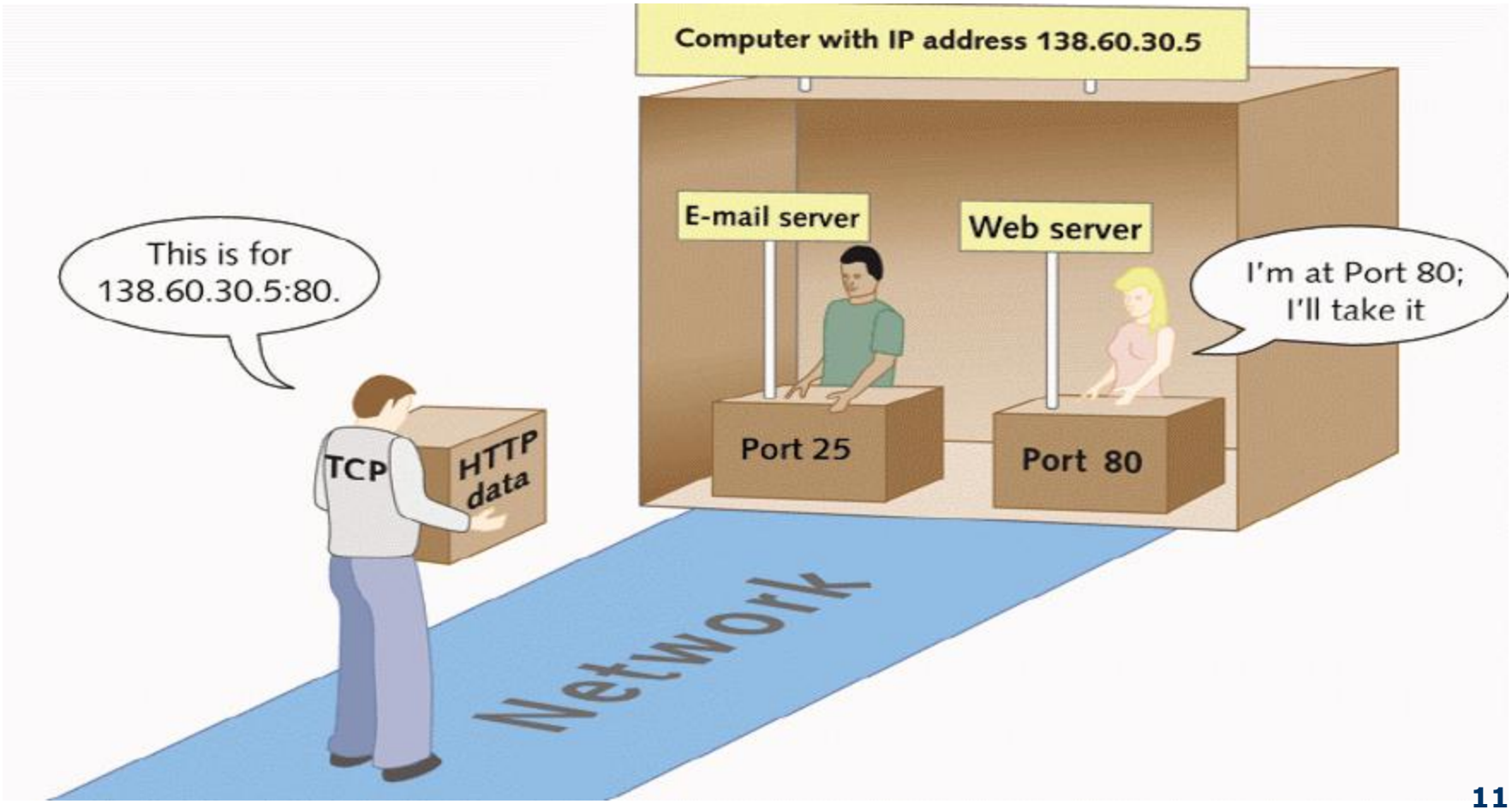
Các giao thức trong TCP/IP

❖ TCP/IP Port

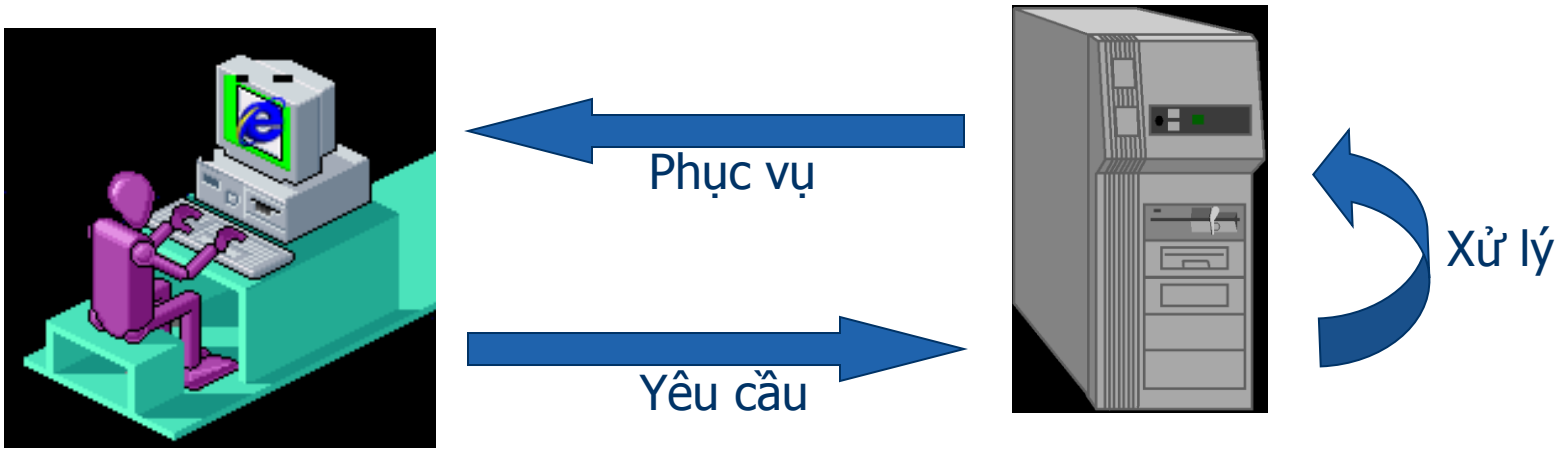
Port	Protocol	Service	Description
20	FTP	FTP	File transfer data
21	FTP	FTP	File transfer control information
23	Telnet	Telnet	Telnet, an application used by Unix computers to control a computer remotely
25	SMTP	Email	Simple Mail Transfer Protocol; used by client to send email
80	HTTP or HTTPS	Web browser	World Wide Web protocol
109	POP2	Email	Post Office Protocol, version 2; used by client to receive email
110	POP3	Email	Post Office Protocol, version 3; used by client to receive Email
119	NNTP	News server	News servers
143	IMAP	Email	Internet Message Access Protocol, a newer protocol used by clients to receive email

2. Cách chuyển dữ liệu trên mạng

- **Dùng địa chỉ IP và số Port để nhận dạng dịch vụ**



Mô hình Client / Server



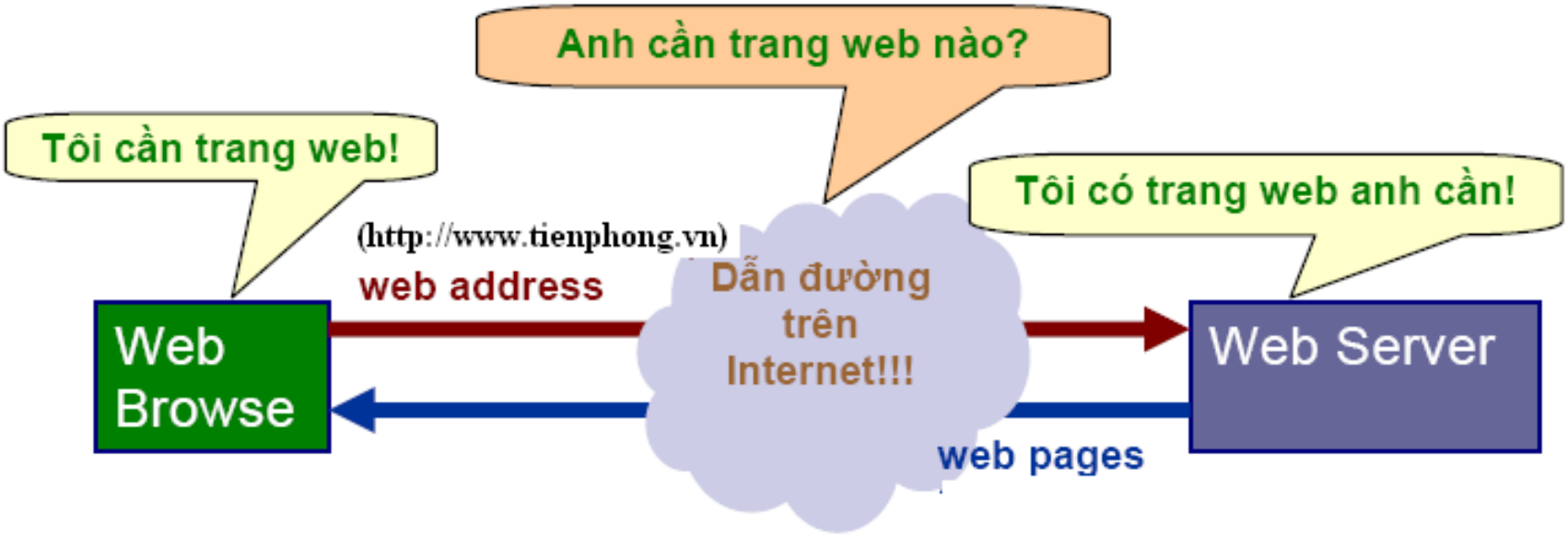
- Tạo ra 1 yêu cầu
- Gửi yêu cầu qua Server
- Chờ Server xử lý
- Nhận kết quả trả về và xử lý theo mục đích riêng

- Lắng nghe yêu cầu
- Nhận yêu cầu
- Xử lý yêu cầu
- Gửi kết quả trả về cho Client

Dạng thức của thông điệp phải tuân theo protocol của dịch vụ.

3. Kết nối Internet

Mô hình hoạt động của Web



4. Lập trình mạng dùng Socket

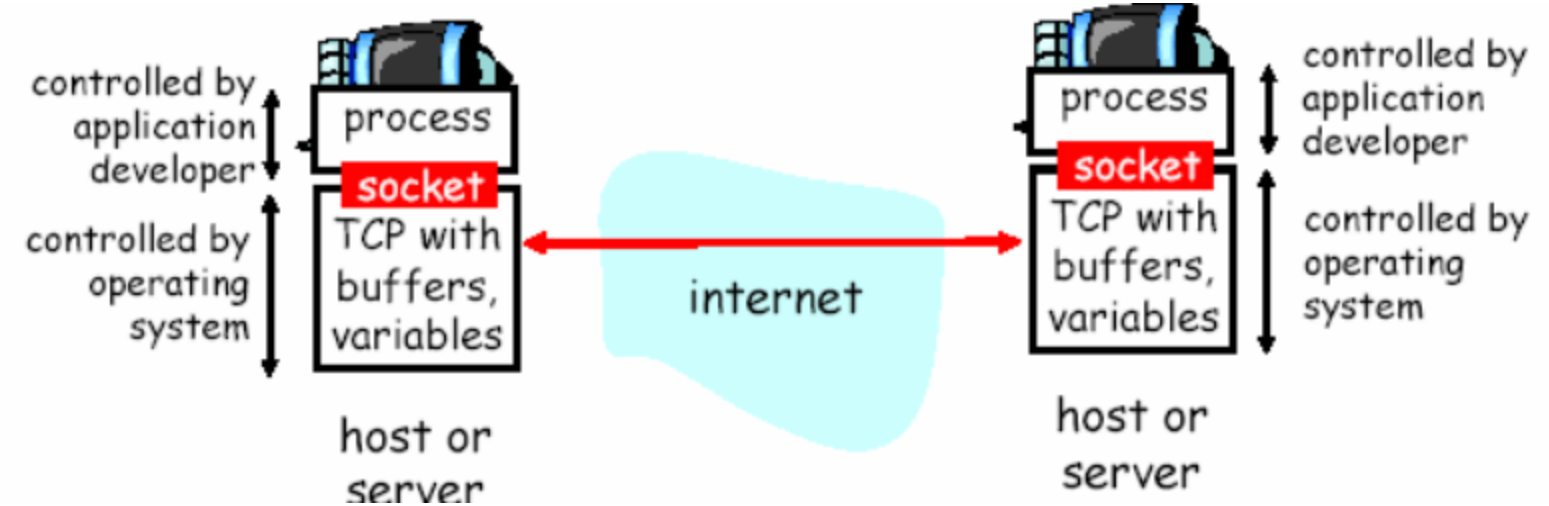
KHÁI NIỆM VỀ SOCKET

- **Socket API**
  - Được giới thiệu ở UNIX, 1981
  - Được ứng dụng khởi tạo, sử dụng và hủy bỏ.
  - Dùng cơ chế client/server
  - Cung cấp hai dịch vụ chuyển dữ liệu thông qua socket API:
    - unreliable datagram
    - reliable, byte stream-oriented

4. Lập trình mạng dùng Socket

Socket (cơ chế ổ cắm): kết nối máy khách và máy chủ.

- Máy khách: xem như phích cắm
- Máy chủ: xem như ổ cắm
- Nhiều máy khách kết nối vào 1 máy chủ.



4. Lập trình mạng dùng Socket

Socket kết nối máy khách và máy chủ cần 2 yêu cầu:

- Địa chỉ máy cần kết nối
- Số hiệu cổng của chương trình dịch vụ (vùng cho phép từ 1 đến ~16000)

Java cung cấp gói thư viện **Java.net** gồm các lớp:

- Socket: dùng cho máy khách
- ServerSocket: dùng cho máy chủ

Stop!

LẬP TRÌNH HỆ THỐNG MẠNG

Web service

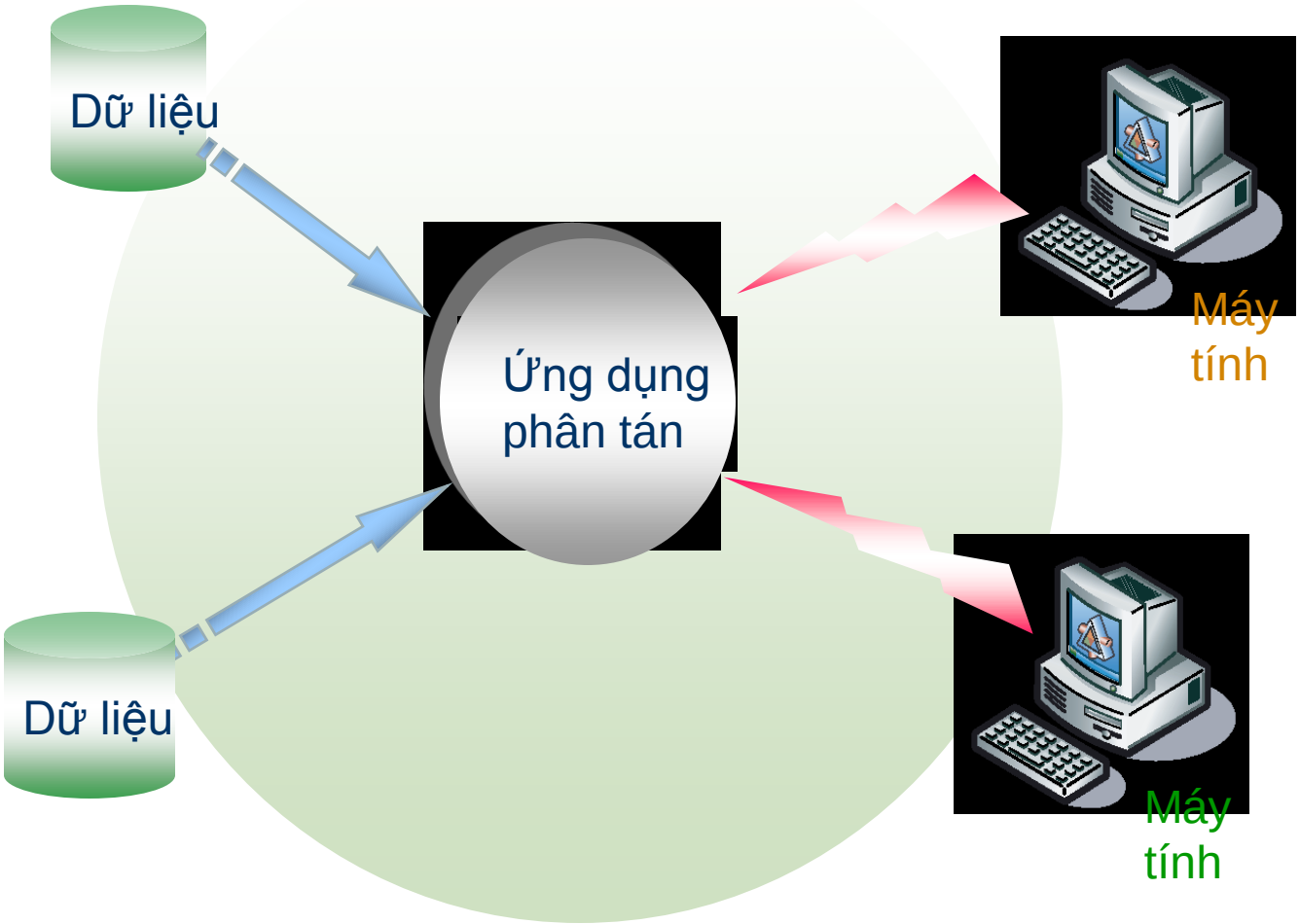
Nguyễn Hữu Thể

Nội dung

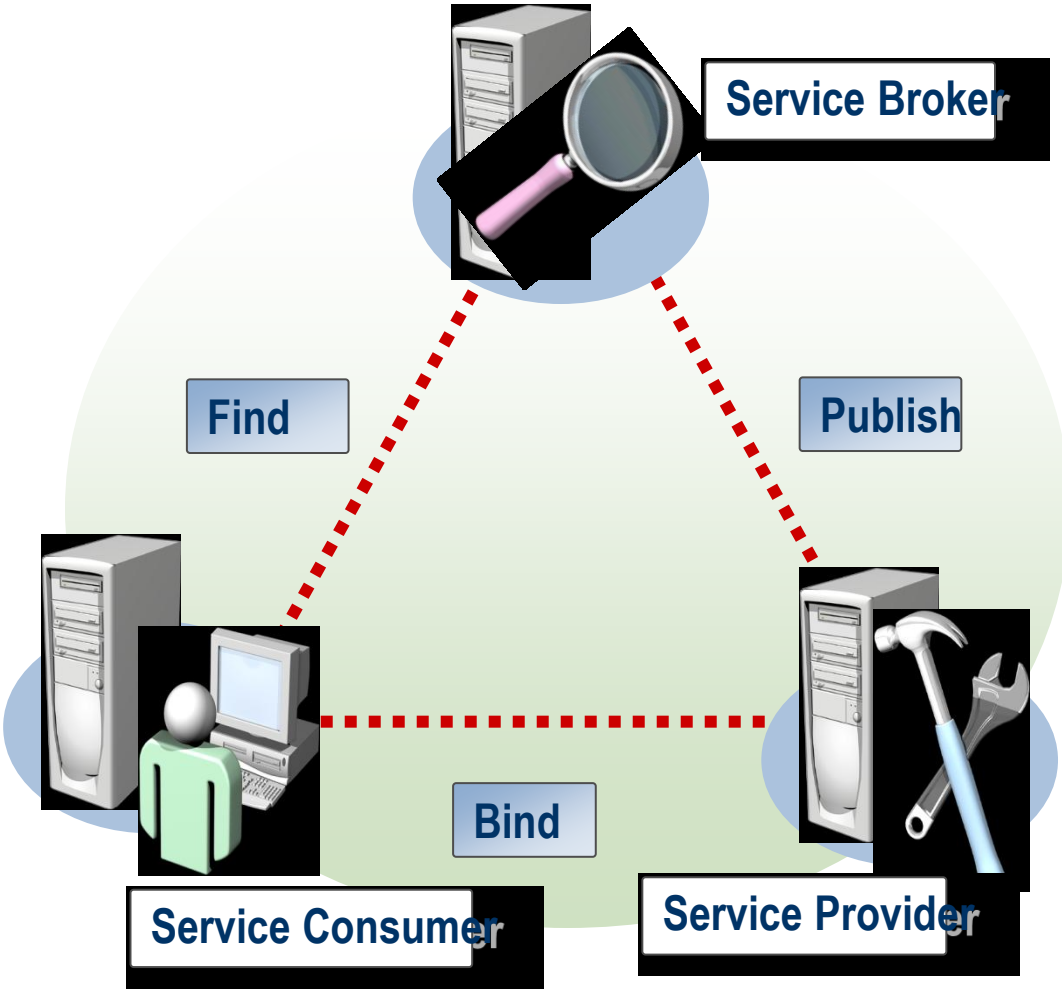
1. Các ứng dụng phân tán
2. Kiến trúc hướng dịch vụ
3. Web service
4. Java Web service

Tài liệu tham khảo: bài giảng trên tailieu.vn, Internet

Các ứng dụng phân tán (Distributed)



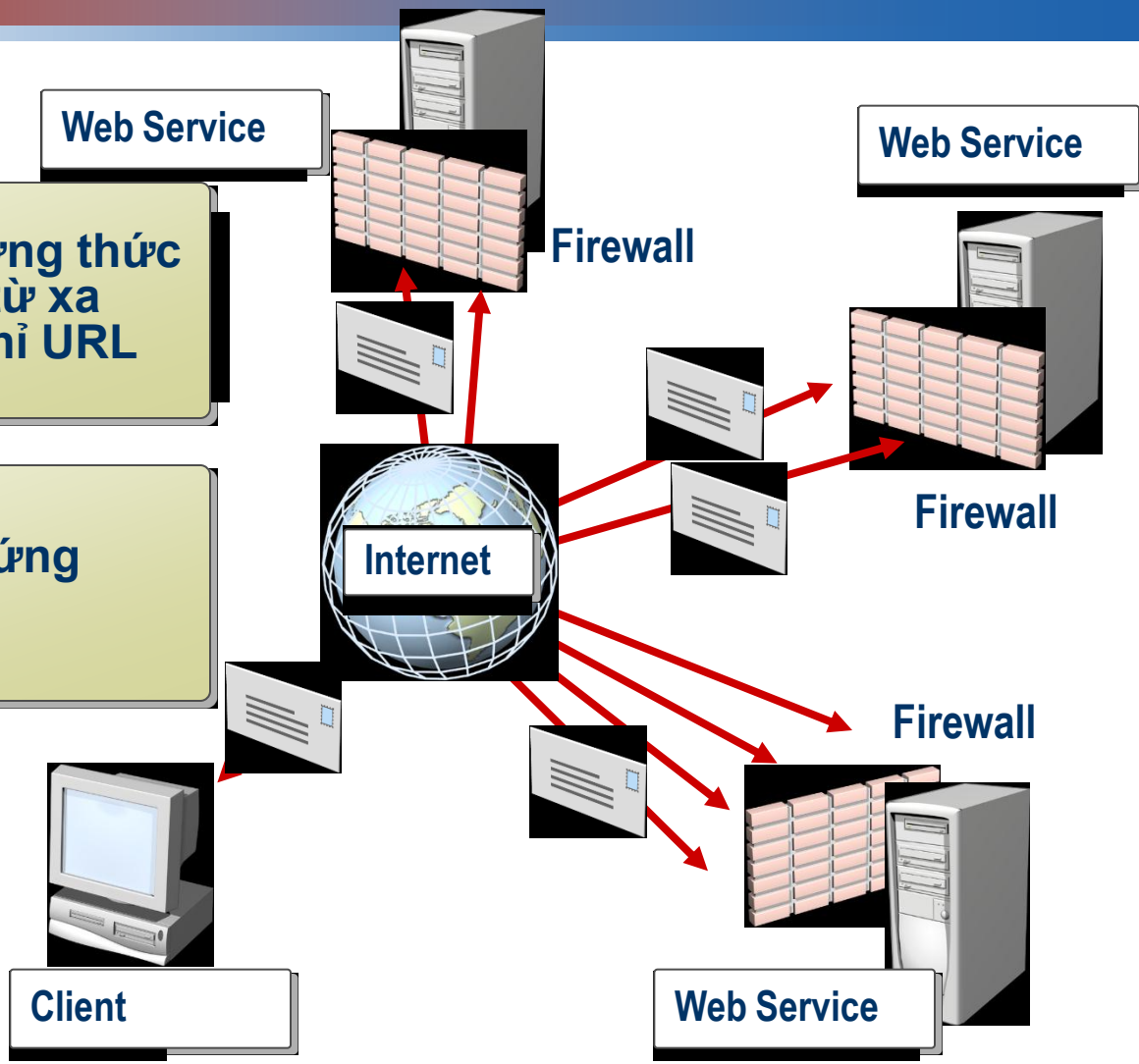
Kiến trúc hướng dịch vụ (SOA)



Web service

1. một tập các phương thức
ợc gọi thực hiện từ xa
ông qua một địa chỉ URL

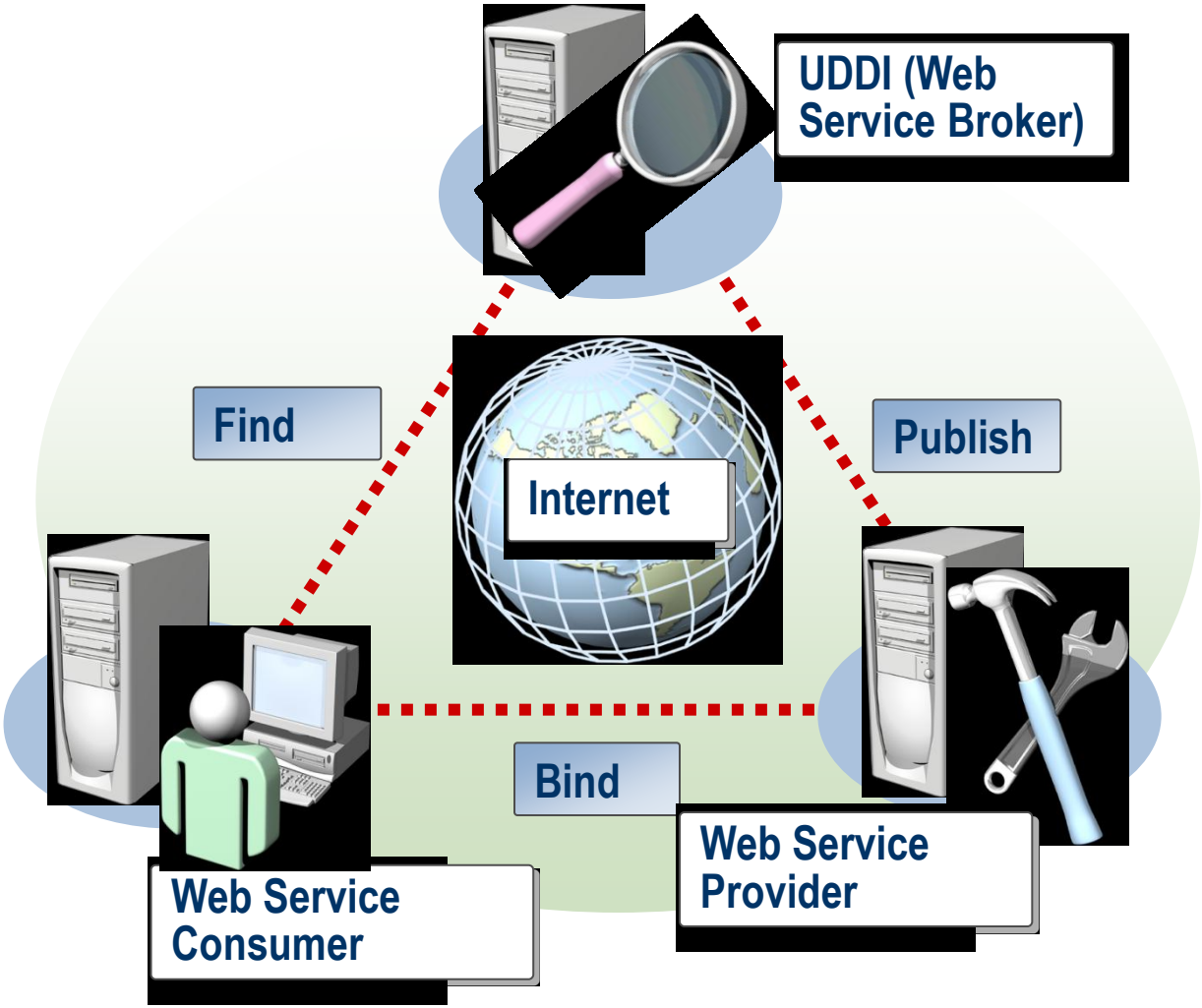
2. ụng để tạo các ứng
ng phân tán



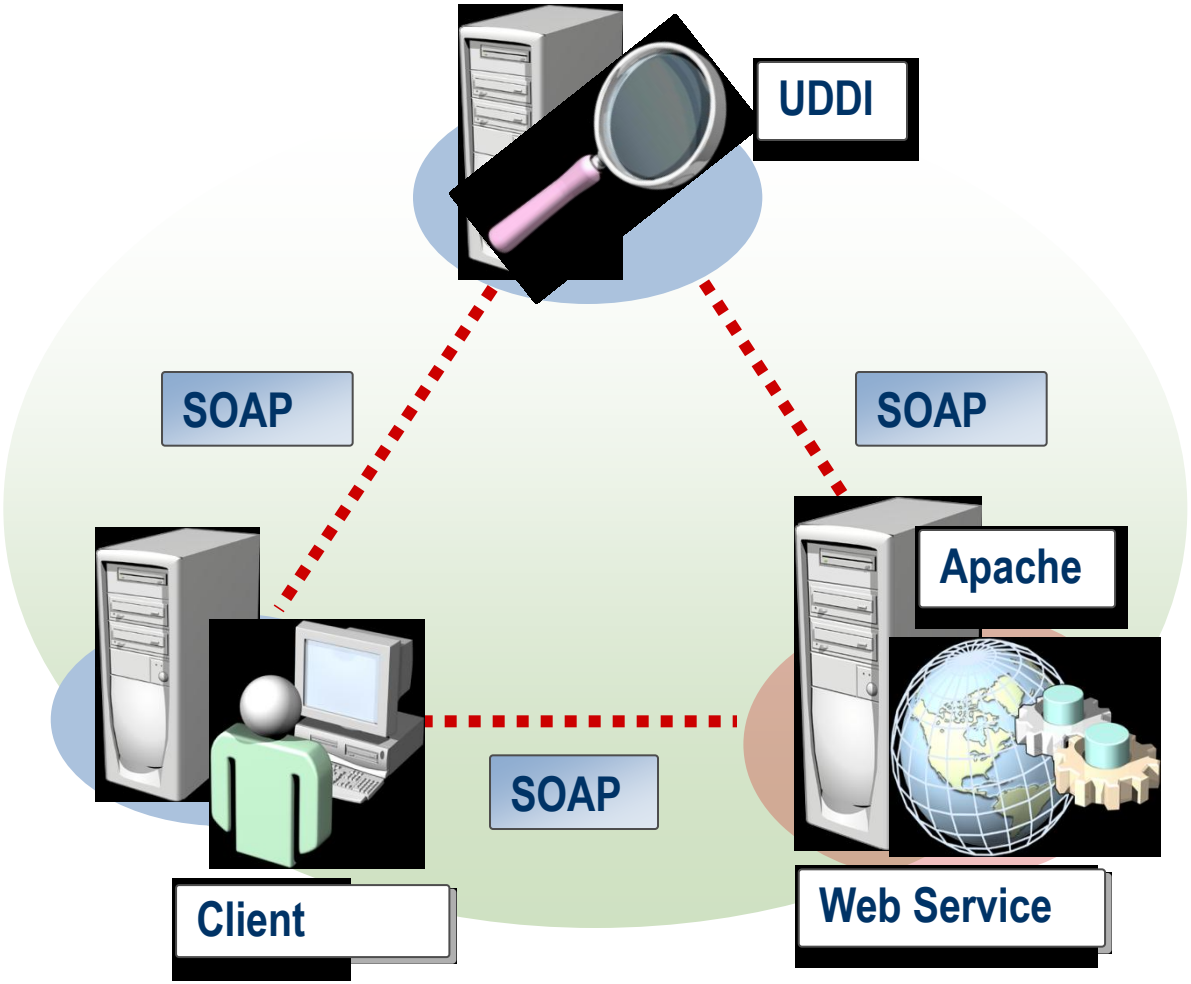
Đặc điểm

1. hông phụ thuộc vào ngôn ngữ lập trình
2. ruy cập bất cứ ứng dụng nào
3. ỗ trợ thao tác giữa các thành phần không đồng nhất
4. hi phí phát triển thấp
5. ễ bảo trì

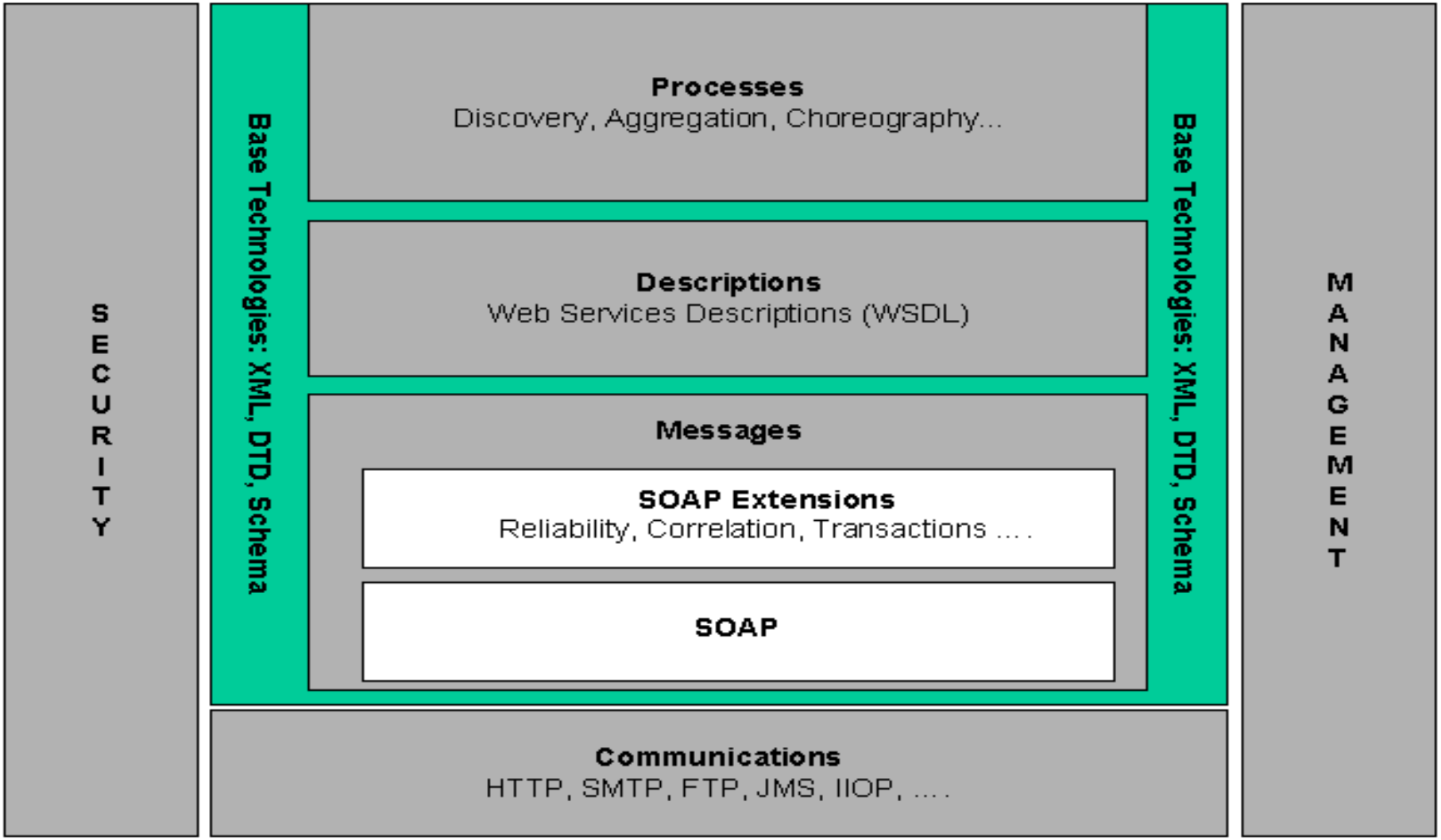
Kiến trúc của Web Service



Kiến trúc của Web Service



Kiến trúc phân tầng của Web Service



Web service provider

1. eb Server
2. ung cấp Web service

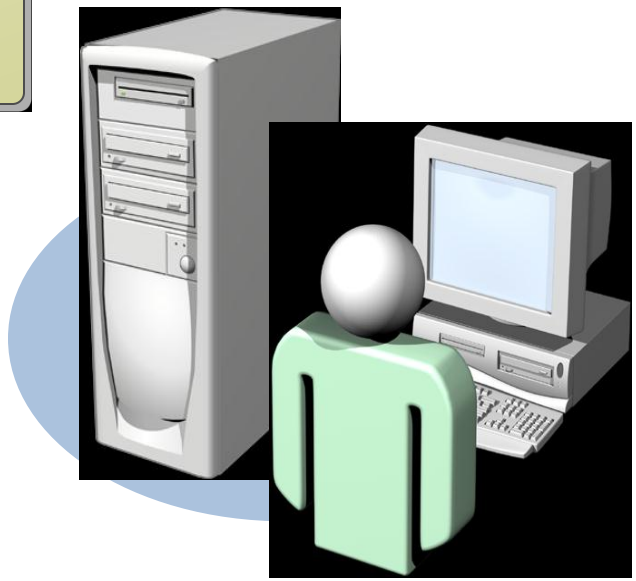
Apache, Glassfish

Yêu cầu

1. ỗ trợ 1 protocol listenner
2. ỗ các cơ chế bảo mật
3. ung cấp đúng service mà nsumer yêu cầu

Web service consumer

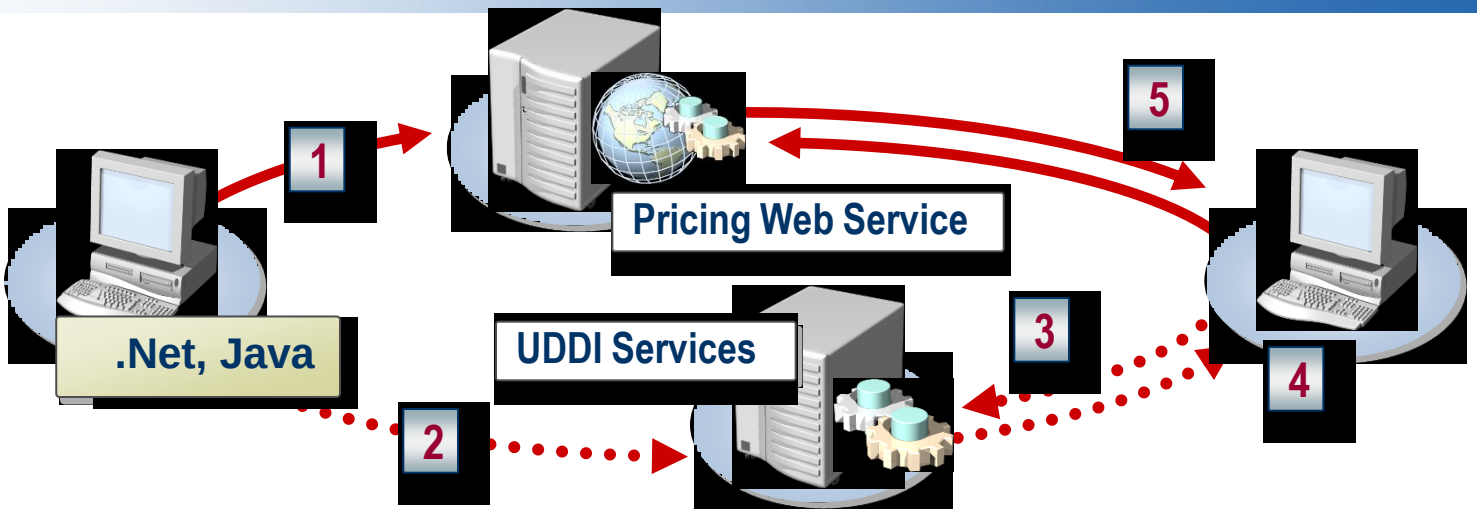
Sử dụng Web service do WS Provider cung cấp để xây dựng ứng dụng



Web service broker

1. Là môi giới giữa consumer và provider
2. Tương tác với provider : để lấy các thông tin về web service
3. Tương tác với consumer :cung cấp cho consumer địa chỉ các Web service
4. Sử dụng UDDI registries (Universal Description, Discovery and Integration)

Mô hình tương tác



1. **Web service developer** builds and deploys a pricing Web service
2. **Web service developer** registers and categorizes the Web service
3. **Web service consumer** queries UDDI for "pricing" services
4. **Web service consumer** determines the most appropriate "pricing" service
5. **Web service developer** builds a solution that directly consumes the Web service data

Nền tảng của Web Service

HTTP

HTTP Request

```
POST /TheStockExchange/Trading/GetStockPrice.asp HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

Symbol=MSFT
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 75

<?xml version="1.0" encoding="utf-8"?>
<stock symbol="MSFT" Price="71.50" />
```

XML – Extend Markup Language

1. ử dụng để mô tả Web service interface
2. OAP Message là một tài liệu XML
3. ử dụng để serialize các đối tượng thành một tài liệu XML System.Xml.Serialization)

SOAP(1)

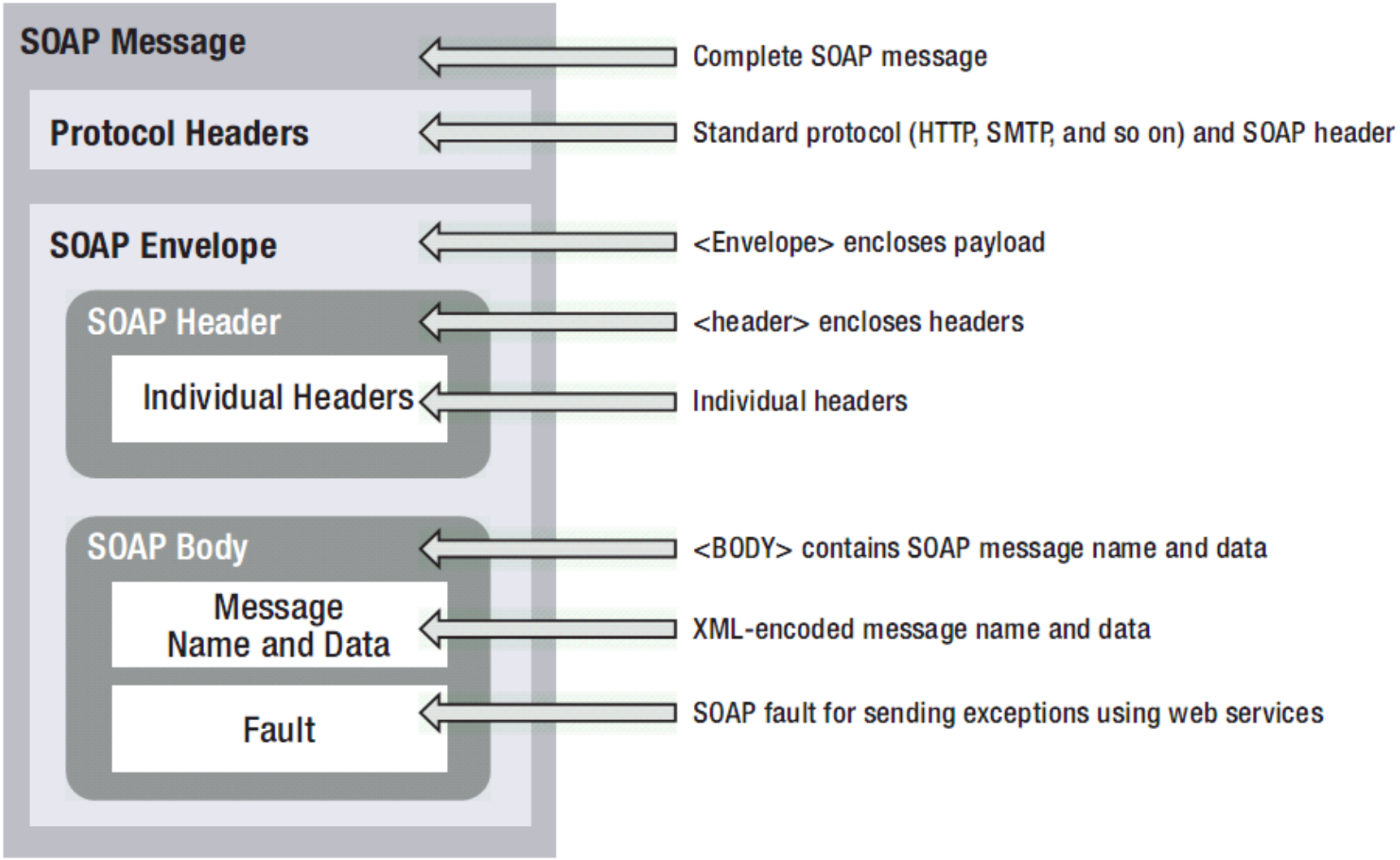
1. OAP – Simple Object Access Protocol
2. rotocol for message-based communication
3. ML-Based protocol
   - An envelope for handling extensibility and modularity
   - An encoding mechanism for representing types within an envelope
4. OAP ≈ HTTP + XML
5. ubmitted to W3C

SOAP (2)

6 OAP Request : HTTP POST Request

```
POST /WebCalculator/Calculator.asmx HTTP/1.1
Content-Type: text/xml
SOAPAction: “http://tempuri.org/Add”
Content-Length: 386

<?xml version=“1.0”?>
<soap:Envelope ...>
  ...
</soap:Envelope>
```

SOAP(3) - Message Structure



SOAP(4) - Example of a SOAP Request

Request

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
        SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
                <ns1:doubleAnInteger xmlns:ns1="urn:MySoapServices">
                        <param1 xsi:type="xsd:int">12</param1>
                </ns1:doubleAnInteger>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP(5) - Example of a SOAP Response

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
<ns1:doubleAnIntegerResponse
        xmlns:ns1="urn:MySoapServices"
                        SOAP-ENV: encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">
                        <return xsi:type="xsd:int">24</return>
                </ns1:doubleAnIntegerResponse>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

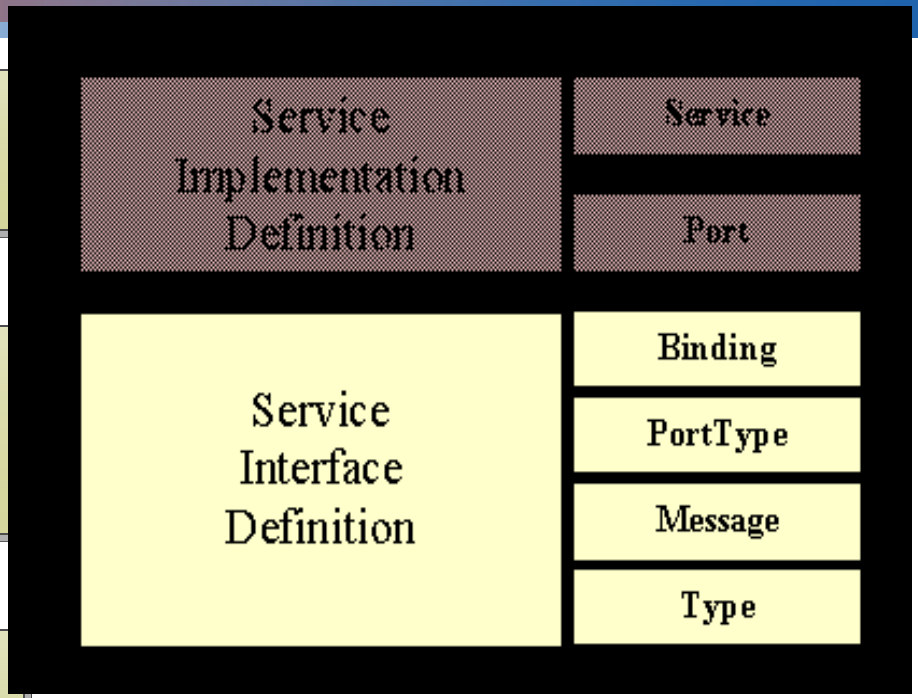
Message response mang thông tin kết quả trả về trong phần body.
Kết quả của phép nhân đôi số nguyên 12 là 24.

Web Service Consumer

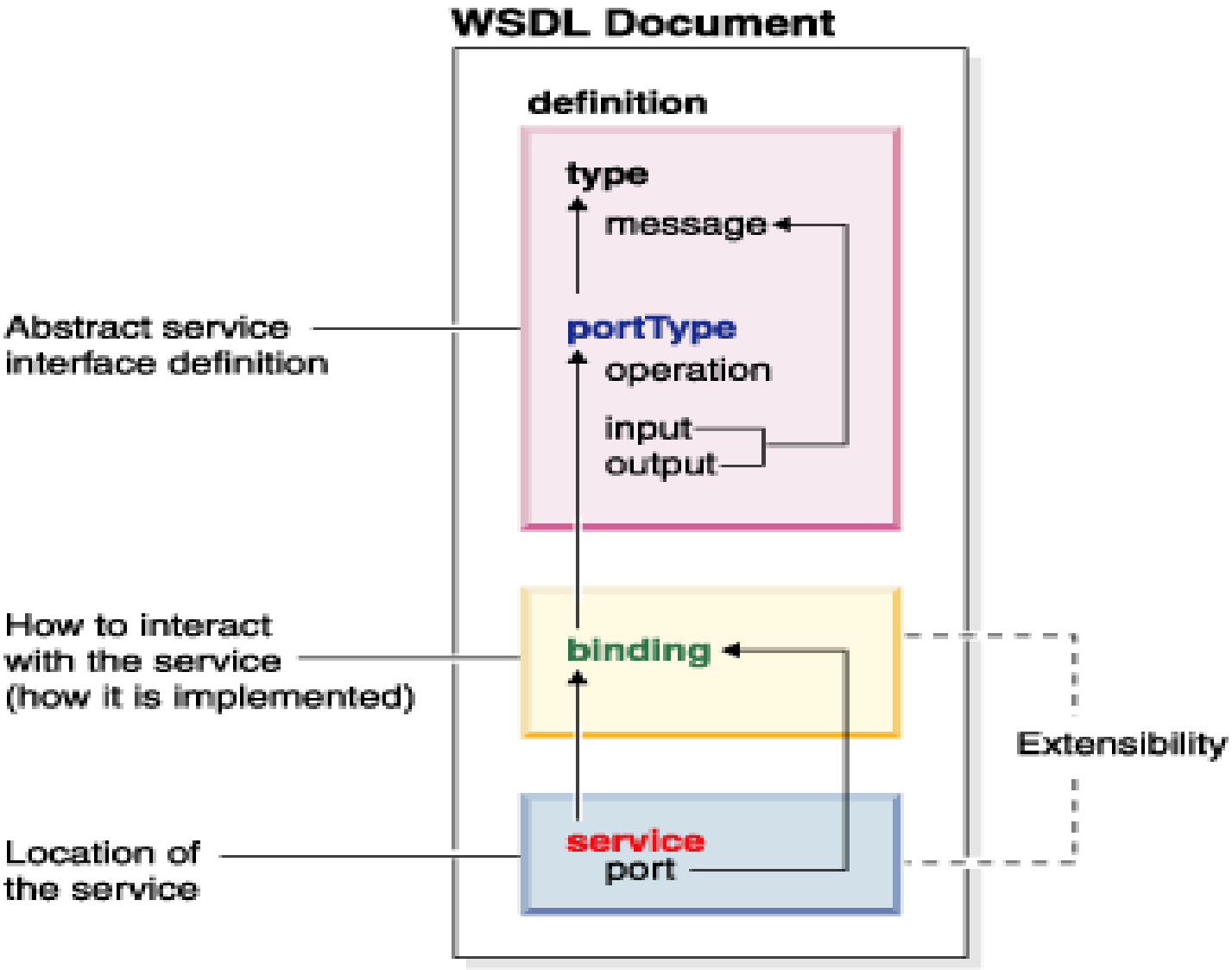
1. WSDL Documents
2. Proxy class
3. Consuming Web service in Java

WSDL Documents

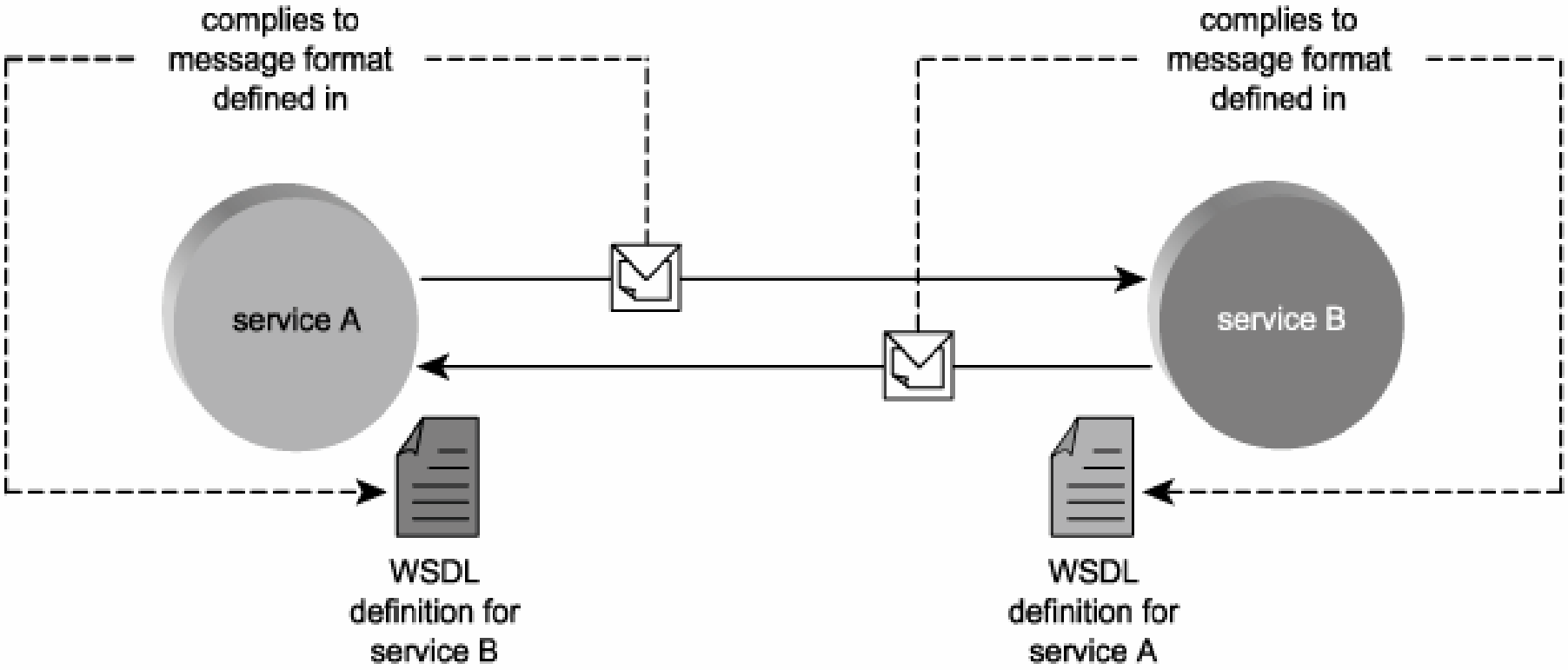
1. Web Service Description Language
2. XML schema : Mô tả giao diện của web service
3. Sử dụng đề phát sinh proxy class
4. Xem WSDL Document của 1 web service http://localhost:8080/TestService/service/Hello?wsdl



WSDL Documents



WSDL Documents



Hình: WSDL cho phép kết nối lỏng lẻo giữa các dịch vụ

Proxy class

1. Proxy : Ủy nhiệm
2. Được phát sinh từ WSDL Document
3. Được sử đụng để giao tiếp với Web service

Ứng dụng của Web Services

1. WS là một tập hợp các chức năng được thực thi qua môi trường mạng, được các ứng dụng khác sử dụng
2. Một WS có thể kết hợp với các WS khác để đưa ra tính năng cao hơn
3. Mô hình ứng dụng trong tương lai sẽ là sự kết hợp giữa các service
4. …

Stop!

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

Giáo trình

LẬP TRÌNH HỆ THỐNG

Biên soạn:

ThS. Nguyễn Hứa Duy Khang
Ks. Trần Hữu Danh

-ĐHCT-
Tháng 5-2008

NỘI DUNG

Chương 1 - TỔ CHỨC BỘ XỬ LÝ INTEL-8086 1
1.1. Bộ xử lý Intel-8086 (CPU-8086) 1
1.1.1. Cấu trúc tổng quát 1
1.1.2. Các thanh ghi của 8086 2
1.1.3. Trạng thái tràn: 5
1.2. Bộ nhớ trong của Intel-80x86 5
1.2.1. Tổ chức dữ liệu 5
1.2.2. Sự phân đoạn bộ nhớ trong 6
1.3. Địa chỉ các ngoại vi 7
1.4. Các bộ xử lý Intel khác 8
1.4.1. Bộ xử lý Intel-80386 8
1.4.2. Tập thanh ghi của bộ xử lý Intel-80386: 8
1.4.3. Các chế độ vận hành của bộ xử lý Intel-80386 9
1.4.4. Bộ xử lý Intel-80486: 10
1.4.5. Bộ xử lý Intel PENTIUM: 11
BÀI TẬP CHƯƠNG 1 13
Chương 2 - HỢP NGỮ 15
2.1. Ngôn ngữ máy và hợp ngữ 15
2.2. Đặc tính tổng quát của hợp ngữ 16
2.2.1. Cấu trúc của một dòng lệnh hợp ngữ. 16
2.2.2. Macro 17
2.2.3. Chương trình con 17
2.2.4. Biến toàn cục (global), biến địa phương (local) 18
2.2.5. Các bảng, thông báo: 18
2.2.6. Hợp ngữ chéo (cross assembler) 19
2.3. Hợp ngữ MASM (hợp ngữ của CPU-8086) 19
2.3.1. Cấu trúc của một hàng lệnh 19
2.3.2. Tên 19
2.3.3. Từ gợi nhớ mã lệnh, lệnh giả 20
2.3.4. Toán hạng và toán tử 27
2.4. Cấu trúc của chương trình hợp ngữ MASM 30
2.4.3. Tập tin thi hành dạng COM và dạng EXE 31
2.4.4. Ví dụ 32
2.5. Cách tạo chương trình hợp ngữ 33

Chương 3 - TẬP LỆNH CPU-8086 ĐƠN GIẢN và KIỂU ĐỊNH VỊ 36
3.1. Tập lệnh của CPU-8086 36
3.1.1. Lệnh sao chép dữ liệu, địa chỉ: 36
3.1.2. Lệnh tính toán số học. 38
3.1.3. Nhóm lệnh logic và ghi dịch 39
3.1.4. Nhóm lệnh vào ra ngoại vi. 42
1.3.5. Nhóm lệnh hệ thống 43
3.2. Kiểu định vị 43
3.2.1. Định vị tức thì: 44
3.2.2. Định vị thanh ghi 44
3.1.3. Định vị trực tiếp (bộ nhớ): 44
3.1.4. Định vị gián tiếp thanh ghi 45
3.1.5. Định vị nền 45
3.1.6. Định vị chỉ số 46
3.1.7. Định vị chỉ số nền 46
3.1.8. Định vị chuỗi 46
3.1.9. Định vị cổng vào/ra 47
BÀI TẬP CHƯƠNG 3 48
Chương 4 - HỆ THỐNG NGẮT MỀM 50
4.1. Những cơ sở của ngắt mềm 50
4.2. Sử dụng ngắt trong hợp ngữ 50
4.3. Ngắt MS-DOS 51
4.4 Các ví dụ 56
Chương 5 - LỆNH NHẢY VÀ VÒNG LẶP 60
5.1. Lệnh nhảy (chuyển điều khiển) 60
5.1.1. Lệnh nhảy không điều kiện 60
5.1.2. Lệnh nhảy có điều kiện: 61
5.2. Vòng lặp 64
BÀI TẬP CHƯƠNG 5 66
Chương 6 - NGĂN XẾP VÀ CHƯƠNG TRÌNH CON 68
6.1. Ngăn xếp 68
6.1.1. Tổ chức và vận hành 68
6.1.2. Truy xuất ngăn xếp 69
6.2. Chương trình con 70
6.2.1. Khai báo chương trình con (Thủ tục) 70
6.2.2. Gọi thủ tục 71
6.3. Các ví dụ 71
BÀI TẬP CHƯƠNG 6 75

Chương 7 - XỬ LÝ KÝ SỐ VÀ XỬ LÝ CHUỖI 76
7.1. Xử lý ký tự 76
7.1.1. Nhập xuất số nhị phân (Binary) 76
7.1.2. Nhập xuất số thập lục phân (Hexa) 77
7.2. Lệnh xử lý chuỗi 78
7.2.1. Hướng xử lý chuỗi 79
7.2.2. Các tiền tố lập REP (Repeat) 79
7.2.3. Lệnh Ghi vào chuỗi 80
7.2.4. Lệnh Nạp từ chuỗi 81
7.2.5. Lệnh di chuyển chuỗi 81
7.2.6. Lệnh So sánh hai chuỗi 83
7.2.7. Lệnh dò tìm trong chuỗi 85
BÀI TẬP CHƯƠNG 7 87
Phụ lục 1 - Hướng Dẫn Sử Dụng Emu8086 88
Phụ lục 2 – Tập Lệnh Intel-8086 93
Phụ lục 3 – Bảng mã ASCII 117

GIỚI THIỆU MÔN HỌC

I. MỤC ĐÍCH YÊU CẦU

Môn Lập Trình Hệ Thống (CT143) cung cấp cho sinh viên những kiến thức cơ bản về lập trình hệ thống trên máy tính IBM/PC thông qua Hợp Ngữ (Assembly). Môn học này là nền tảng để tiếp thu hầu hết các môn học khác trong chương trình đào tạo. Mặt khác, nắm vững Hợp ngữ là cơ sở để phát triển các ứng dụng điều khiển thiết bị.

Học xong môn này, sinh viên phải nắm được các vấn đề sau:

- Tổ chức bộ xử lý Intel-8086
- Cấu trúc chương trình Hợp ngữ
- Tập lệnh của Intel-8086
- Hệ thống ngắt mềm trên máy tính IBM/PC
- Lệnh nhảy và vòng lập trong Assembly
- Ngăn xếp và Thủ tục
- Xử lý số và Chuỗi

II. ĐỐI TƯỢNG MÔN HỌC

Môn học được dùng để giảng dạy cho các sinh viên sau:

- Sinh viên năm thứ 3 của các chuyên ngành Điện tử

III. NỘI DUNG CỐT LÕI

Giáo trình được cấu trúc thành 7 chương:

Chương 1: Tổ chức bộ xử lý Intel-8086
Chương 2: Hợp ngữ
Chương 3: Tập lệnh và Kiểu định vị
Chương 4: Hệ thống ngắt mềm
Chương 5: Lệnh nhảy và Vòng lập
Chương 6: Ngăn xếp và Chương trình con
Chương 7: Xử lý số và Chuỗi

IV. KIẾN THỨC LIÊN QUAN

Để học tốt môn Lập trình Hệ thống, sinh viên cần phải có các kiến thức nền tảng sau:

- Kiến thức Kỹ thuật số.
- Kiến thức Kiến trúc máy tính
- Kiến thức Ngôn ngữ lập trình cấp cao: C, Pascal, Delphi ...
- Kỹ năng thao tác sử dụng máy tính.

V. DANH MỤC TÀI LIỆU THAM KHẢO

[1] Nguyễn Văn Linh, Lâm Hoài Bảo, Dương Văn Hiếu, *Giáo trình Lập trình căn bản A*, Khoa Công Nghệ Thông Tin, Đại học Cần Thơ, 2005.

[2] Nguyễn Đình Tê, Hoàng Đức Hải , *Giáo trình lý thuyết và bài tập ngôn ngữ C*; Nhà xuất bản Giáo dục, 1999.

[3] Nguyễn Cẩn, *C – Tham khảo toàn diện*, Nhà xuất bản Đồng Nai, 1996.

[4] Brain W. Kernighan & Dennis Ritchie, *The C Programming Language*, Prentice Hall Publisher, 1988.

[5] Võ Văn Chín, *Bài giảng Ngôn ngữ hệ thống*, Khoa Công Nghệ Thông Tin, Đại học Cần Thơ, 1994.

Chương 1

TỔ CHỨC BỘ XỬ LÝ INTEL-8086

Mục đích:

- Cấu trúc bên trong CPU Intel-8086
- Tập thanh ghi
- Tổ chức bộ nhớ và dữ liệu
- Khái quát các bộ xử lý Intel khác như: 80386, 80486, Pentium

1.1. BỘ XỬ LÝ INTEL-8086 (CPU-8086)

1.1.1. Cấu trúc tổng quát

Intel-8086 là một CPU 16 bit (bus dữ liệu ngoại có 16 dây). Nó được dùng để chế tạo các máy vi tính PC-AT đầu tiên của hãng IBM vào năm 1981. Trong thực tế, hãng IBM đã dùng CPU 8088 (là một dạng của CPU 8086 với bus số liệu giao tiếp với ngoại vi là 8 bit) để chế tạo máy vi tính cá nhân đầu tiên gọi là PC-XT.

Cho đến nay CPU 8086 đã không ngừng cải tiến và đã trải qua các phiên bản 80186, 80286, 80386, 80486, Pentium (80586), Pentium Pro, Pentium MMX, Pentium II, III, 4. Các CPU trên tương thích từ trên xuống (downward compatible) nghĩa là tập lệnh của các CPU mới chế tạo gồm các tập lệnh của CPU chế tạo trước đó được bổ sung thêm nhiều lệnh mạnh khác.

Cấu trúc tổng quát của CPU-8086 có dạng như hình 1.1, gồm 2 bộ phận chính là: Bộ thực hiện lệnh và bộ phận giao tiếp bus.

1. Bộ phận thực hiện lệnh (EU):

Thi hành các tác vụ mà lệnh yêu cầu như: Kiểm soát các thanh ghi (đọc/ghi), giải mã và thi hành lệnh. Trong EU có bộ tính toán và luận lý (ALU) thực hiện được các phép toán số học và luận lý. Các thanh ghi đa dụng là các ô nhớ bên trong CPU chứa dữ liệu tương tự như ô nhớ trong bộ nhớ. Cờ cũng là một thanh ghi dùng để ghi lại trạng thái hoạt động của ALU. Thanh ghi lệnh chứa nội dung lệnh hiện tại mà CPU đang thực hiện.

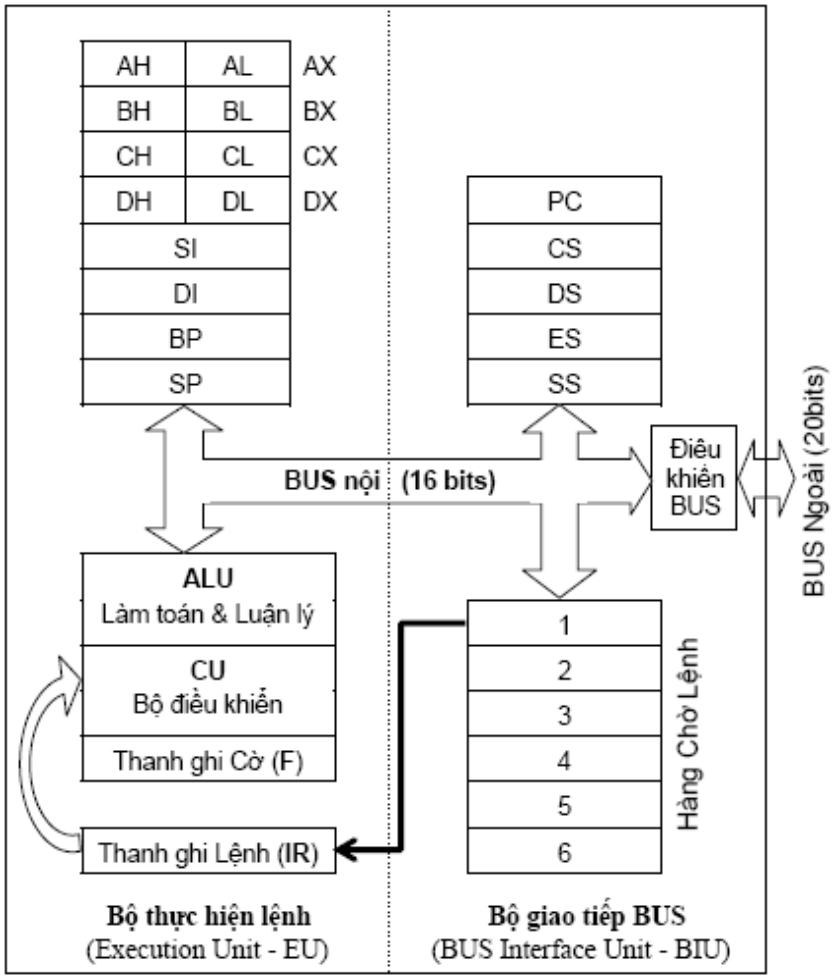
Các thanh ghi và bus trong EU đều là 16 bit. EU không kết nối trực tiếp với bus hệ thống bên ngoài. Nó lấy lệnh từ hàng chờ lệnh mà BIU cung cấp. Khi có yêu cầu truy xuất bộ nhớ hay ngoại vi thì EU yêu cầu BIU làm việc. BIU có thể tái định địa chỉ để cho phép EU truy xuất đầy đủ 1 MB (8086 có 20 đường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

2. Bộ giao tiếp bus (BIU):

BIU thực hiện chức năng giao tiếp giữa EU với bên ngoài (Bộ nhớ, thiết bị ngoại vi …) thông qua hệ thống BUS ngoại (bus dữ liệu và bus địa chỉ). BIU thực hiện tất cả các tác vụ về bus mỗi khi EU có yêu cầu. Khi EU cần trao đổi dữ liệu với bên ngoài, BIU sẽ tính toán địa chỉ và truy xuất dữ liệu để phục vụ theo yêu cầu EU.

Trong BIU có 5 thanh ghi CS, DS, ES, SS và IP chứa địa chỉ. Thanh ghi IP chứa địa chỉ của lệnh sẽ được thi hành kế tiếp nên gọi là con trỏ lệnh.

EU và BIU liên lạc với nhau thông qua hệ thống bus nội. Trong khi EU đang thực hiện lệnh thì BIU lấy lệnh từ bộ nhớ trong nạp đầy vào hàng chờ lệnh (6 bytes). Do đó EU không phải đợi lấy lệnh từ bộ nhớ. Đây là một dạng đơn giản của cache để tăng tốc độ đọc lệnh.



Hình 1.1: Sơ đồ khối của CPU 8086

1.1.2. Các thanh ghi của 8086

Thanh ghi (register) là thành phần lưu trữ dữ liệu bên trong CPU, mỗi thanh ghi có độ dài nhất định (16 bit hoặc 8 bit) và được nhận biết bằng một tên riêng. Tùy vào độ dài và chức năng mà thanh ghi có công dụng chứa dữ liệu hoặc kết quả của phép toán, hoặc là các địa chỉ dùng để định vị bộ nhớ khi cần thiết.

Nội dung của thanh ghi được truy xuất thông qua tên riêng của nó, do đó tên thanh ghi là từ khóa quan trọng cần phải lưu ý trong lập trình.

CPU-8086 có 16 thanh ghi, mỗi thanh ghi là 16 bit, có thể chia 4 nhóm sau:

1. **Thanh ghi đoạn:** Gồm 4 thanh ghi 16 bit: CS, DS, ES, SS. Đây là những thanh ghi dùng để chứa địa chỉ đoạn của các ô nhớ khi cần truy xuất. Mỗi thanh ghi đoạn quản lý 1 đoạn tối đa 64K ô nhớ trong bộ nhớ trong. Người sử dụng chỉ được phép truy xuất ô nhớ dựa vào địa chỉ tương đối. CPU (cụ thể là BIU) có nhiệm vụ chuyển đổi địa chỉ tương đối thành địa chỉ tuyệt đối để truy xuất vào ô nhớ tuyệt đối tương ứng trong bộ nhớ. *(Xem phần tổ chức bộ nhớ)*

   CS: Thanh ghi đoạn mã lệnh, lưu địa chỉ đoạn chứa mã lệnh chương trình của người sử dụng

DS: Thanh ghi đoạn dữ liệu, lưu địa chỉ đoạn chứa dữ liệu (các biến) trong chương trình.

ES: Thanh ghi đoạn dữ liệu thêm, lưu địa chỉ đoạn chứa dữ liệu thêm trong chương trình.

SS: Thanh ghi đoạn ngăn xếp, lưu địa chỉ đoạn của vùng ngăn xếp.

15 … 0	
CS	Code Segment
DS	Data Segment
ES	Extra data Segment
SS	Stack Segment

Thông thường bốn thanh ghi này có thể chứa những giá trị khác nhau, do đó chương trình có thể được truy cập trên bốn đoạn khác nhau và chương trình chỉ có thể truy cập cùng 1 lúc tối đa bốn đoạn. Mặc khác, đối với những chương trình nhỏ, chỉ sử dụng 1 đoạn duy nhất, khi đó cả bốn thanh ghi đều chứa cùng giá trị địa chỉ đoạn, gọi là đoạn chung.

2. **Thanh ghi đa dụng (General Register):** Bao gồm bốn thanh ghi đa dụng 16 bit (AX, BX, CX, DX). Mỗi thanh ghi đa dụng có thể được sử dụng với nhiều mục đích khác nhau, tuy nhiên từng thanh ghi có công dụng riêng của nó.

15 … 8	7 … 0	
AH	**AL**	**AX** (Accumulator)
BH	**BL**	**BX** (Base register)
CH	**CL**	**CX** (Count register)
DH	**DL**	**DX** (Data register)

AX : Là thanh ghi tích lũy 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 AX.

BX: Thanh ghi nền dùng để tính toán địa chỉ ô nhớ.

CX: Là thanh ghi đếm, thường dùng để đếm số lần trong một lệnh vòng lặp hoặc lệnh xử lý chuỗi ký tự.

DX: Thanh ghi dữ liệu, thường chứa địa chỉ của một số lệnh vào/ra, lệnh tính toán số học (kể cả lệnh nhân và chia).

Mỗi thanh ghi 16 bit có thể chia đôi thành 2 thanh ghi 8 bit. Do đó, CPU-8086 có 8 thanh ghi 8 bit là: AH, AL; BH, BL; CH, CL; DH, DL (thanh ghi AH và AL tương ứng với byte cao và byte thấp của thanh ghi AX, tương tự cho các thanh ghi 8 bit còn lại).

Ví dụ: AX = 1234h => AH = 12h, AL = 34h

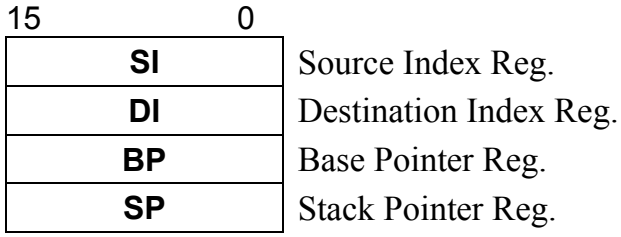
3. **Thanh ghi con trỏ và chỉ số (Pointer & Index register)**: Chức năng chung của nhóm thanh ghi này là chứa địa chỉ độ dời của ô nhớ trong vùng dữ liệu hay ngăn xếp.

SI : Thanh ghi chỉ số nguồn

DI : Thanh ghi chỉ số đích

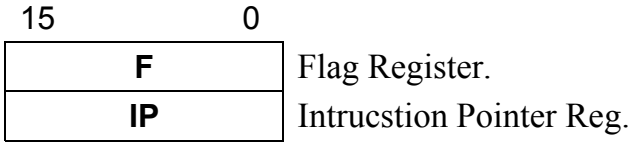
BP: Thanh ghi con trỏ nền dùng để lấy số liệu từ ngăn xếp.

SP : Thanh ghi con trỏ ngăn xếp luôn chỉ vào đỉnh ngăn xếp.



SI và DI chứa địa chỉ độ dời của ô nhớ tương ứng trong đoạn có địa chỉ chứa trong DS hoặc ES *(dữ liệu, còn gọi là Biến)*. Còn BP và SP chứa địa chỉ độ dời của ô nhớ tương ứng trong đoạn có địa chỉ chứa trong SS, dùng để thâm nhập số liệu trong ngăn xếp.

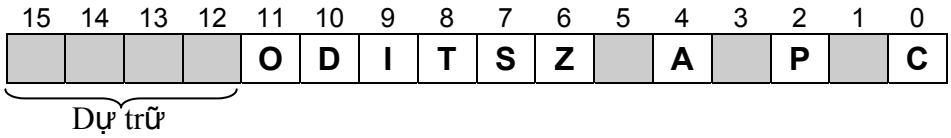
4. Thanh ghi Đếm chương trình và thanh ghi trạng thái (Cờ):



- Thanh ghi con trỏ lệnh **IP** *(còn gọi là PC – đếm chương trình)* là thanh ghi 16 bit chứa địa chỉ của lệnh kế tiếp mà CPU sẽ thực hiện trong. Các lệnh của chương trình có địa chỉ đoạn trong CS.
- Thanh ghi Cờ (**F**) dài 16 bit, mỗi bit là một cờ. Mỗi cờ có giá trị 1 *(gọi là SET –Đặt)* hoặc 0 *(gọi là CLEAR – Xóa)*. Hình 1.2 mô tả 9 bit trong số 16 bit tương ứng với 9 cờ trạng thái (các bit còn lại dùng cho dự trữ mở rộng khi thiết kế các CPU khác)

  Thanh ghi cờ được chia thành hai nhóm:

  - Nhóm cờ điều khiển (bảng 1.1) bao gồm các cờ dùng để điều khiển sự hoạt động của CPU và giá trị của cờ được thiết lập bằng các lệnh phần mềm.
  - Nhóm cờ trạng thái (bảng 1.2) bao gồm các cờ phản ánh kết quả thực hiện lệnh cũng như trạng thái của CPU



Hình 1.2: Cấu trúc thanh ghi Cờ

KÝ HIỆU	TÊN	Ý NGHĨA
Nhóm cờ điều khiển		
TF	Bẫy (Trap)	TF = 1: cho phép chương trình chạy từng bước
IF	Ngắt (Interrupt)	IF = 1: cho phép ngắt phần cứng
DF	Hướng (Direction)	DF = 1: thì SI và DI giảm 1 cho mỗi vòng lặp

Nhóm cờ trạng thái		
CF	Số giữ (Carry)	CF = 1: khi có số nhớ hoặc mượn từ MSB trong phép cộng hoặc trừ. *(Có tthể bị thay đổi theo lệnh ghi dịch và quay)*
PF	Chẳn lẻ (Parity)	PF = 1: khi byte thấp của thanh ghi kết quả một phép tính có số lượng bit 1 chẳn
AF	Số giữ phụ (Hafl)	AF = 1: khi có nhớ hoặc mượn từ bit 3 trong phép cộng hoặc trừ. Dùng trong các lệnh với số BCD
ZF	Zero	ZF = 1: khi kết quả của một phép tính bằng 0
SF	Dấu (Sign)	SF = 1 khi kết quả phép tính là âm (MSB=1). SF = 0 khi kết quả dương (MSB=0)
OF	Tràn (Overflow)	OF = 1: nếu kết quả vượt quá khả năng tính toán của CPU

Bảng 1.1: Ý nghĩa cờ

1.1.3. Trạng thái tràn:

Trạng thái tràn có thể không xảy ra (nếu không tràn) hoặc xảy ra (nếu tràn có dấu, tràn không dấu, đồng thời tràn có dấu và không dấu). Nói chung là có 2 trạng thái tràn: Tràn không dấu và Tràn có dấu.

Lưu ý: Nếu một giá trị có MSB=1 (bit dấu) thì CPU luôn luôn cho đó là số có dấu.

a. Tràn không dấu: CF=1

Ví dụ: ADD AX, BX ; với AX = 0FFFFh, BX = 1

- Nếu xem đây là các số không dấu thì AX không đủ chứa kết quả nên TRÀN không dấu, vậy CF = 1
- Nếu xem đây là các số có dấu thì kết quả sẽ là 0 (vì AX = -1) nên không tràn, do đó OF = 0

b. Tràn có dấu: OF = 1

Ví dụ: ADD AX, BX ; với AX = BX = 7FFFh = 32767

- Nếu xem đây là các số không dấu thì AX = 7FFFh + 7FFFh = 0FFFEh = 65534 nên không tràn.
- Nếu xem đây là các số có dấu thì tràn vì kết quả vượt quá phạm vi cho phép đối với số có dấu (cộng 2 số dương, kết quả lại là số âm). Thật sự trong trường hợp này, CPU sẽ làm cho OF = 1 theo qui tắc *"Nhớ ra và vào MSB xảy ra không đồng thời"* nghĩa là có nhớ vào MSB nhưng không có nhớ ra hoặc ngược lại thì tràn và không có hoặc có nhớ ra và vào MSB thì không tràn.

1.2. BỘ NHỚ TRONG CỦA INTEL-80x86

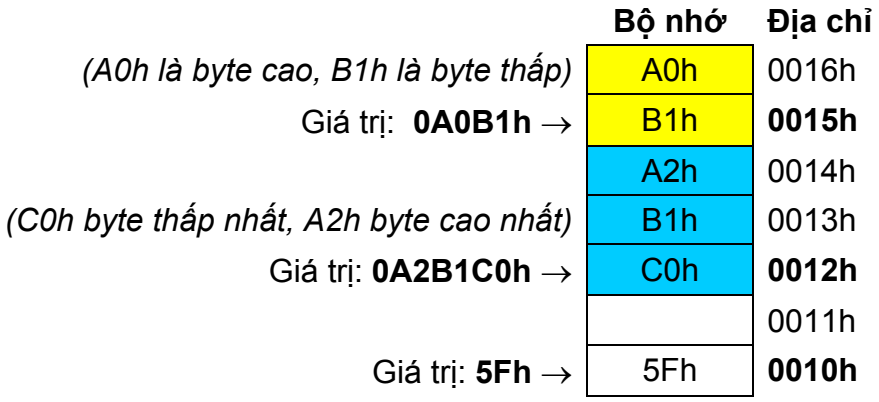
1.2.1. Tổ chức dữ liệu

Bộ nhớ trong được tổ chức thành mảng gồm các ô nhớ 8 bit liên tực nhau. Các dữ liệu có thể được ghi vào hoặc đọc ra (gọi là truy xuất) từ bất cứ vị trí ô nhớ nào. Mỗi ô nhớ 8 bit được phần cứng quản lý bằng một địa chỉ vật lý duy nhất. Việc truy xuất nội dung ô nhớ phải bằng địa chỉ vật lý này.

Dữ liệu 8 bit được lưu trữ bằng một ô nhớ và địa chỉ của ô nhớ chính là địa chỉ dùng để truy xuất dữ liệu. Dữ liệu nhiều hơn 8 bit được lưu trữ bởi nhiều ô

nhớ liên tục nhau. Theo quy ước Intel, byte dữ liệu cao được lưu ở ô nhớ có địa chỉ cao và byte dữ liệu thấp hơn lưu ở ô nhớ có địa chỉ thấp hơn. Khi đó, địa chỉ dùng để truy xuất dữ liệu là địa chỉ của ô nhớ thấp *(ô nhớ chứa byte thấp nhất của dữ liệu)*

Hình 1.3 mô tả việc tổ chức các dữ liệu có độ dài khác nhau trong bộ nhớ. Giá trị 5Fh (1 byte) được lưu trữ ở địa chỉ 0010h. Giá trị 0A0B1h (2 byte) được lưu trữ bởi 2 ô nhớ có địa chỉ 0015h và 0016h, địa chỉ để truy xuất giá trị này là 0015h. Còn giá trị 0A2B1C0h (3 byte) được lưu trữ bởi 3 ô nhớ 0012h, 0013h và 0014h, do đó địa chỉ truy xuất giá trị ấy là 0012h.



Hình 1.3: Tổ chức dữ liệu trong bộ nhớ

1.2.2. Sự phân đoạn bộ nhớ trong

CPU 8086 có không gian địa chỉ là 1MB (ứng với 20 bit địa chỉ) Vậy CPU 8086 có thể quản lý bộ nhớ trong là 2^{20} = 1MB. Bộ nhớ 1 MB này được CPU-8086 quản lý bằng nhiều đoạn 64 KB. Các đoạn có thể tách rời hoặc có thể chồng lên nhau.

Mỗi đoạn có một địa chỉ đoạn 16 bit duy nhất, tùy vào mục đích sử dụng đoạn mà địa chỉ đoạn được lưu trữ trong thanh ghi đoạn tương ứng.

Đối với người lập trình, Địa chỉ của ô nhớ trong bộ nhớ được xác định bởi hai thông số 16 bit (gọi là địa chỉ logic): Địa chỉ Đoạn (segment) và địa chỉ độ dời (offset).

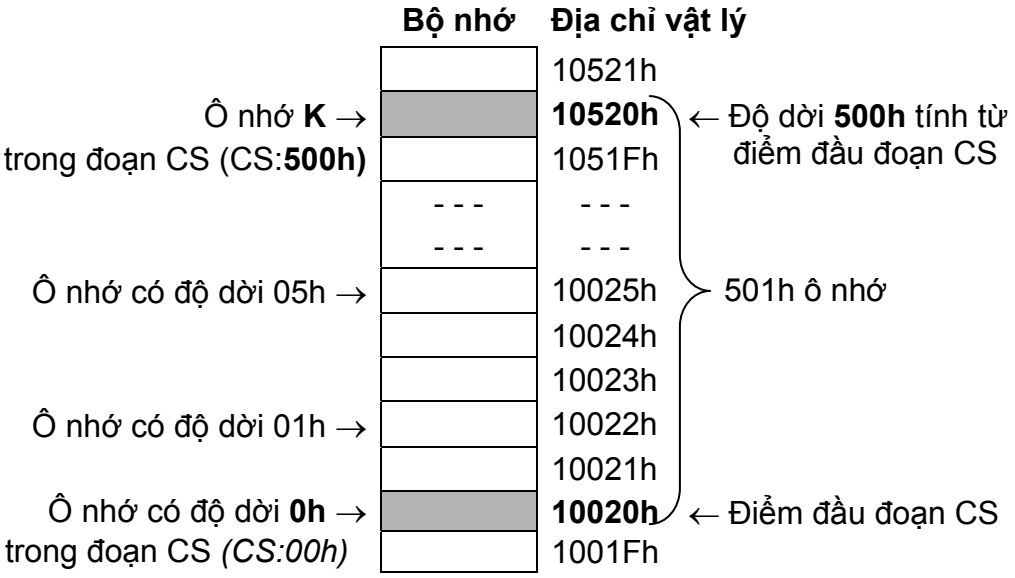
Cách viết: **Segment : Offset**

Địa chỉ vật lý của ô nhớ khi truy xuất sẽ được BIU tự động chuyển đổi từ địa chỉ logic bằng cách dịch trái thanh ghi đoạn bốn bit *(tức nhân nội dung của thanh ghi đoạn cho 16)* rồi cộng với địa chỉ độ dời. Vì vậy, người lập trình không cần địa chỉ vật lý của ô nhớ mà chỉ cần biết địa chỉ logic của ô nhớ.

Ví dụ: đoạn CS có giá trị là 1002h, địa chỉ độ dời của ô nhớ K trong đoạn CS là 500h (CS:0500h hoặc 1002h:0500h). Khi đó, địa chỉ vật lý của ô nhớ K được tính như sau:

```
  10020h  (dịch trái địa chỉ đoạn 4 bit)
+ 0500h   (độ dời)
-------
  10520h  (địa chỉ vật lý)
```

Trong ví dụ trên, 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 *(xem hình 1.4)*



Hình 1.4: Địa chỉ vật lý của ô nhớ K trong đoạn CS

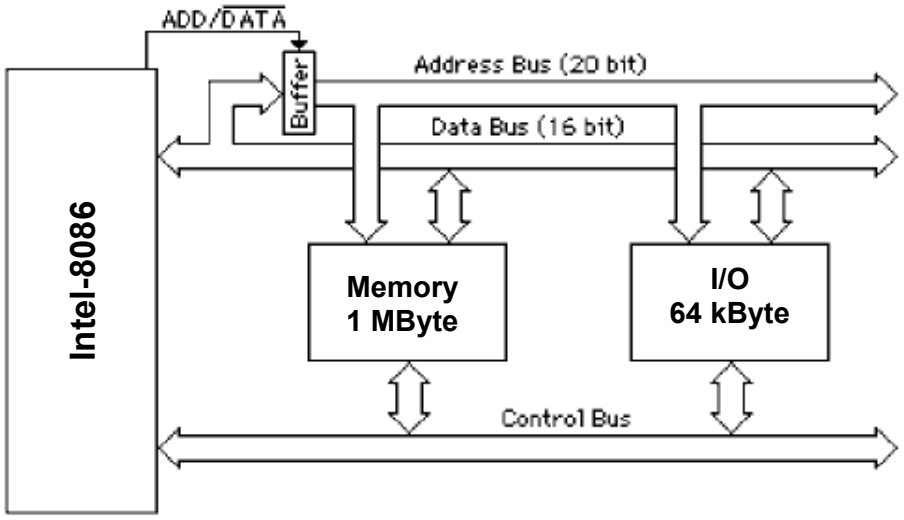
Khi truy xuất ô nhớ, BIU lấy sẽ sử dụng địa chỉ đoạn trong thanh ghi đoạn tương ứng với tính chất của ô nhớ cần truy xuất:

- Ô nhớ là Code (Mã lệnh) thì đoạn tương ứng là CS.
- Ô nhớ là Data (dữ liệu) thì đoạn tương ứng là DS.
- Ô nhớ nằm trên ngăn xếp thì dùng đoạn SS.
- Khi truy xuất chuỗi, DI và SI luôn chứa độ dời của ô nhớ trong đoạn DS hay ES

Khi khởi động, CPU 8086 nhảy đến địa chỉ vật lý cao nhất của bộ nhớ trong *(đoạn CS = 0FFFFh và độ dời 0)* để lấy lệnh. Địa chỉ này ứng với địa chỉ của ROM-BIOS của bộ nhớ trong nơi chứa chương trình khởi động máy tính.

1.3. Địa chỉ các ngoại vi

Các ngoại vi đều có địa chỉ riêng từ 0 đến 64K. 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).



Hình 1.5: Cấu trúc đơn giản của máy tính

1.4. CÁC BỘ XỬ LÝ INTEL KHÁC

Ngoài CPU-8086, Intel đã cho ra đời thế hệ CPU mới hơn, nhiều tính năng hơn và mạnh hơn như: 80186, 80286, 80386, 80486, Celeron và Pentium (80586). Ngày nay, sức mạnh và tính năng của CPU Intel tăng vượt bậc nhờ công nghệ mới như: Centrino, Hyper Threading, Core Duo.

Bắt đầu từ CPU 80286, hãng Intel đã đưa vào một số cải tiến có ý nghĩa như tăng bus địa chỉ lên 24 bit và có thể vận hành với chế độ bảo vệ (protected mode). Chế độ này cho phép CPU 80286 vận hành trong một hệ điều hành đa nhiệm (multitasking).

1.4.1. Bộ xử lý 80386

Hãng Intel đã thành công lớn khi chế tạo CPU 80386. Đây vẫn còn là một CPU CISC thuần túy có bus địa chỉ 32 bit và bus số liệu 32 bit (ta gọi CPU 80386 là một CPU 32 bit).

Các thanh ghi của CPU 80386 đều là thanh ghi 32 bit (Hình 1.5). Một số thanh ghi đa dụng có thể chia thành thanh ghi 16 bit hoặc chia thành thanh ghi 8 bit để đảm bảo tính tương thích với các CPU chế tạo trước đó.

Với bus địa chỉ 32 bit, không gian địa chỉ của CPU 80386 là 4 GB tức 4096 MB. CPU 80386 có 64K cửa vào/ra 8 bit, 16 bit và 32 bit.

CPU 80386 cũng có thể hoạt động với bộ đồng xử lý toán học 80387 (math coprocessor). Bộ đồng xử lý toán học dùng xử lý các phép tính trên các số có dấu chấm động (số lẻ).

I.4.2. Tập thanh ghi của bộ xử lý 80386:

Hình 1.6 mô tả đầy đủ tập thanh ghi của CPU-80386.

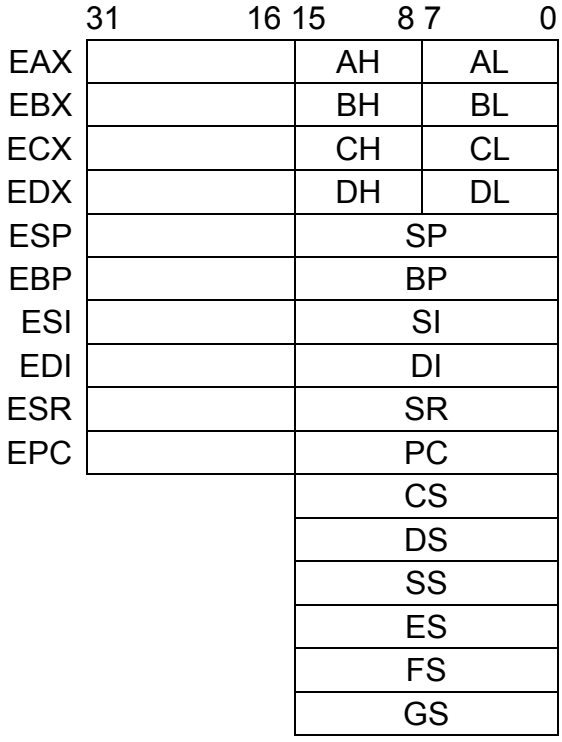
Các thanh ghi đa dụng và thanh ghi con trỏ được mở rộng thành thanh ghi 32 bit được gọi tên là: EAX, EBX, ESP, EDI... . Tuy nhiên ta vẫn có thể sử dụng các thanh ghi 16 bit (AX, BX, ...) hoặc 8 bit (AH, AL, BH, BL ...) giống như các thanh ghi 16 bit hoặc 8 bit của bộ xử lý 8086.

Chiều dài các thanh ghi đoạn vẫn giữ nguyên 16 bit nhưng có thêm hai thanh ghi đoạn thêm là FS và GS. Các thanh ghi FS và GS được dùng giống như thanh ghi ES. Nghĩa là CPU-80386 quản lý được bốn đoạn dữ liệu.

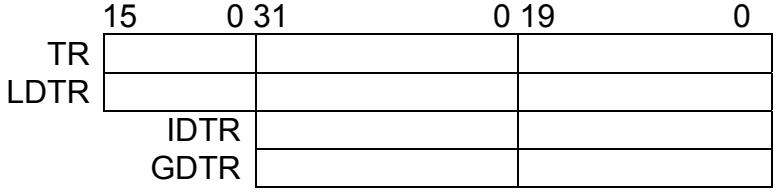
Thanh ghi trạng thái SR (Status register) và thanh ghi đếm chương trình PC (program counter) cũng được nâng lên 32 bit. Ngoài các bit trạng thái đã thấy trong thanh ghi trạng thái của CPU-8086 (C, Z, S, ...) thanh ghi trạng thái của CPU-80386 còn có thêm các bit trạng thái như sau:

- IOP (Input/Output protection: bảo vệ vào/ra): Đây là hai bit trạng thái dùng trong chế độ bảo vệ để xác định mức ưu tiên mà một tiến trình phải có để có thể thâm nhập một vùng vào ra. Chỉ hệ điều hành mới có quyền dùng các bit này.
- N (Nested task: tiến trình lồng vào nhau): Trong chế độ bảo vệ, các hệ điều hành dùng bit này để biết có nhiều tiến trình đang vận hành và ít nhất có một tiến trình đang bị gián đoạn.
- R (Resume: tải trục): Bit này cho phép một tiến trình được tiếp tục vận hành lại sau khi bị gián đoạn.

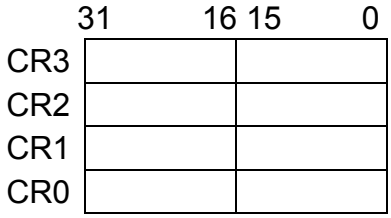
- V (Virtual 8086 mode: chế độ 8086 ảo): Bit này cho phép 80386 đang vận hành ở chế độ bảo vệ, chuyển sang chế độ 8086 ảo.



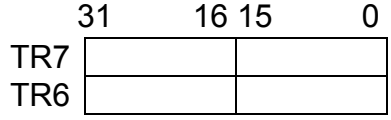
Hình 1.6a: Thanh ghi đa dụng và thanh ghi con trỏ



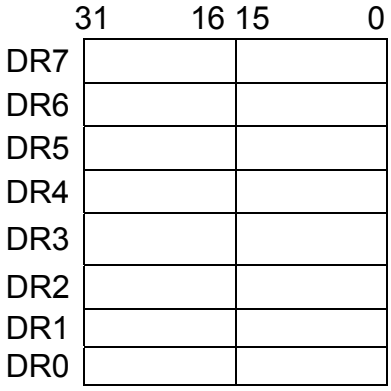
Hình 1.6b: Thanh ghi quản lý bộ nhớ



Hình 1.6c: Thanh ghi điều khiển



Hình 1.6d: Thanh ghi kiểm tra



Hình 1.6e: Thanh ghi gỡ rối

Hình 1.6: Các thanh ghi của CPU 80386

1.4.3. Các chế độ vận hành của bộ xử lý 80386

CPU-80386 có thể vận hành theo một trong ba chế độ khác nhau: chế độ thực (real mode), chế độ bảo vệ (protected mode) và chế độ 8086 ảo (virtual 8086 mode). Chế độ vận hành của CPU phải được thiết lập trước bằng phần cứng.

- **Chế độ thực:** chế độ thực của bộ xử lý 80386 hoàn toàn tương thích với chế độ vận hành của bộ xử lý 8086. Trong chế độ này, không gian địa chỉ của 80386 bị giới hạn ở mức 2^{20} = 1MB giống như không gian địa chỉ của 8086 mặc dù bus địa chỉ của 80386 có 32 đường dây.
- **Chế độ bảo vệ:** (Còn gọi là chế độ đa nhiệm) chế độ bảo vệ đã được đầu tiên đưa vào bộ xử lý 80286. Chế độ này cho phép bộ xử lý 80386 dùng hết không gian địa chỉ của nó là 2^{32} = 4096 MB và cho phép nó vận hành dưới một hệ điều hành đa nhiệm. Trong hệ điều hành đa nhiệm, nhiều tiến trình có thể chạy đồng thời và được bảo vệ chống lại các thâm nhập trái phép vào vùng ô nhớ bị cấm.

  Trong chế độ bảo vệ, các thanh ghi đoạn không được xem như địa chỉ bắt đầu của đoạn mà là thanh ghi chọn (selector) gán các ưu tiên khác nhau cho các tiến trình. Phần ưu tiên khác nhau cho các tiến trình. Phần cốt lõi của hệ điều hành có ưu tiên cao nhất và người sử dụng có ưu tiên thấp nhất.
- **Chế độ 8086 ảo:** Chế độ này cho phép thiết lập một kiểu vận hành đa nhiệm trong đó các chương trình dùng trong chế độ thực, có thể chạy song song với các tiến trình khác.

1.4.4. Bộ xử lý Intel 80486:

CPU-80486DX được phát hành năm 1989. Đó là bộ xử lý 32bit chứa 1.2 triệu transistor. Khả năng quản lý bộ nhớ tối đa giống như 80386 nhưng tốc độ thi hành lệnh đạt được 26.9 MIPS *(Mega Instructions Per Second - triệu lệnh mỗi giây)* tại xung nhịp 33 MHz

Nếu bộ xử lý 80386 là bộ xử lý CISC thuần túy với bộ đồng xử lý toán học 80387 nằm bên ngoài bộ xử lý 80386, thì bộ xử lý 80486 là một bộ xử lý hỗn tạp CISC và RISC với bộ đồng xử lý toán học và với 8K cache nằm bên trong bộ xử lý 80486.

Trong bộ xử lý 80486, một số lệnh thường dùng, ví dụ như lệnh MOV, dùng mạch điện (kỹ thuật RISC) để thực hiện lệnh thay vì dùng vi chương trình như trong các CPU CISC thuần túy. Như thế thì các lệnh thường dùng này được thi hành với tốc độ nhanh hơn. Kỹ thuật ống dẫn cũng được đưa vào trong bộ xử lý 80486.

Với các kỹ thuật RISC được đưa vào, bộ xử lý 80486 nhanh hơn bộ xử lý 80386 đến 3 lần (nếu tốc độ xung nhịp là như nhau).

Bộ xử lý 80486 hoàn toàn tương thích với 2 bộ xử lý 80386 và 80387 cộng lại và như thế nó có các chế độ vận hành giống như 80386.

Bộ xử lý 80486 tỏ ra rất mạnh đối với các chương trình cần tính toán nhiều và các chương trình đồ họa, vì bộ đồng xử lý toán học nằm ngay trong bộ xử lý 80486. Hàng chờ lệnh của bộ xử lý 80486 là 32 byte.

Hình 1.7: CPU-80386

Hình 1.8: CPU-80486

1.4.5. Bộ xử lý Intel PENTIUM:

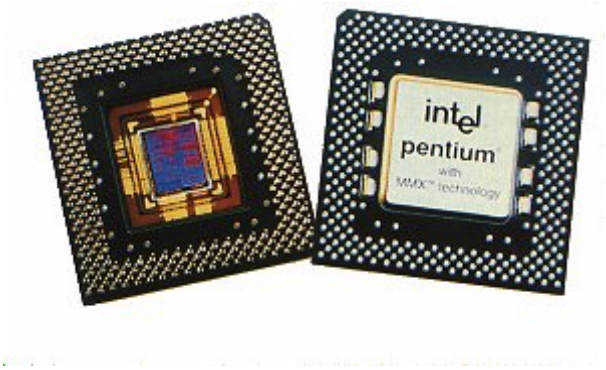
Được đưa ra thị trường vào giữa năm 1993, các bộ xử lý Pentium được chế tạo với hơn 3 triệu transistor, thụ hưởng các tính năng kỹ thuật của các bộ xử lý 80486 DX/2 và được có thêm nhiều tính năng mới.

Theo hãng sản xuất Intel, nếu hoạt động ở với xung nhịp 66 MHz thì tốc độ thi hành lệnh của Pentium là 112 MIPS thay vì 54 MIPS cho bộ xử lý 80486 DX/2 cùng xung nhịp.

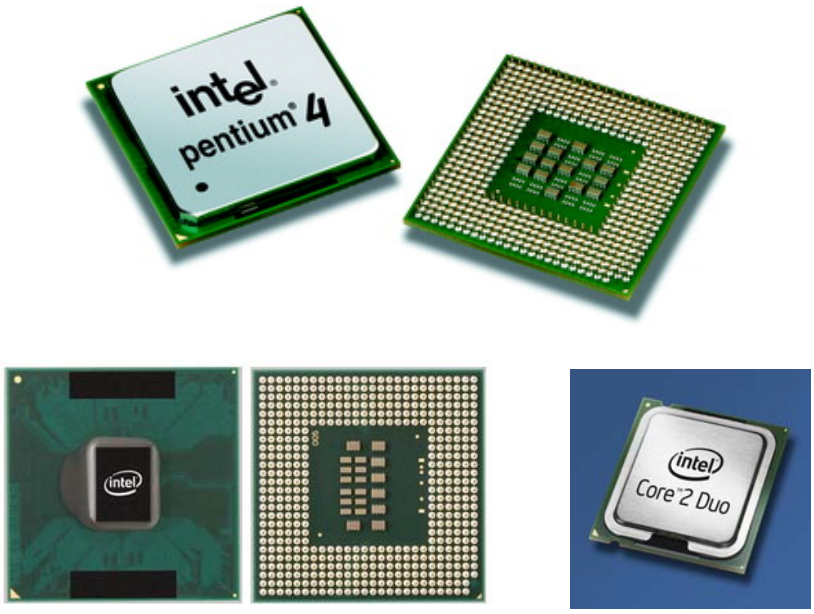
Các tính năng nổi bật của bộ xử lý Pentium là :

- Hoàn toàn tương thích với các bộ xử lý được chế tạo trước đó (8086, 80286, 80386, 80486).
- Dùng kỹ thuật ống dẫn tốt hơn với 2 ống dẫn số nguyên độc lập và một ống dẫn số lẻ. Kỹ thuật siêu vô hướng và tiên đoán lệnh nhảy cũng được đưa vào.
- Dùng cache lệnh và cache dữ liệu riêng biệt.
- Bus số liệu là 64 bit với cách vận chuyển theo từng gói.
- Dùng cách quản lý hệ thống cho phép tiết kiệm năng lượng tiêu hao trong bộ xử lý Pentium.
- Có tối ưu hóa các chuỗi mã lệnh.

Petium MMX: Các lệnh xử lý multimedia được đưa vào tập lệnh của CPU nên việc thi hành chương trình multimedia được cải thiện rất nhiều.



Pentium Celeron



Hình 1.9: Hình ảnh các loại bộ xử lý Intel

BÀI TẬP CHƯƠNG 1

1. Chọn các thanh ghi đa dụng để lưu trữ các dữ liệu sau đây, sao cho mỗi thanh ghi lưu trữ 1 giá trị và không trùng nhau (Giải thích việc chọn thanh ghi): 15h, 0AFh, 01234h, 230, 257, 'H', 8086.

   Ghi chú: Số có tận cùng bằng h (hay H) là số thập lục phân (Hexa);
   'H' : Ký tự H

2. Các thanh ghi đang lưu trữ giá trị như sau:

   AH = 11h AL = 22h CL = 15
   CH = 10 BX = 0A1D4h DX = 8086

   Hãy cho biết giá trị thập lục phân của những thanh ghi sau và giải thích:

   AX, CX, BH, BL, DH, DL

3. Mô tả các cách có thể sử dụng được để lưu trữ giá trị vào thanh ghi sau:

   a. 1234h vào thanh ghi SI b. 5678h vào thanh ghi AX
   c. 100 vào thanh ghi DI d. 100 vào thanh ghi DX

4. Sử dụng mô hình bộ nhớ gồm 17 ô nhớ như hình A1 để ghi các dữ liệu sau đây vào bộ nhớ sao cho các giá trị không chồng lên nhau (sinh viên tự chọn địa chỉ ô nhớ để lưu trữ): 15h, 0AFh, 01234h, 230, 257, 'H', 8086, 3A4B5h, 0F1D2E3h

Ô nhớ	**Địa chỉ**
	00010h
…..	…..
	00002h
	00001h
	00000h

Hình A1: Mô hình bộ nhớ

5. Với mô hình bộ nhớ kết quả của câu 4, hãy cho biết giá trị dữ liệu 8/16/24/32 bit tại mỗi địa chỉ sau đây ở dạng thập lục phân:

   a. 00004h b. 00008h c. 0000Bh d. 0000Dh

6. Đổi sang địa chỉ vật lý tương ứng với mỗi địa chỉ logic sau:

   a. 0000:0001h b. 0100:1234h
   c. ABCD:3AFFh d. AF70:00CFh

7. Viết ra 4 địa chỉ logic khác nhau đối với mỗi địa chỉ vật lý sau:

   a. 40000h b. 0D32FCh

8. Tìm tất cả các địa chỉ logic khác nhau có thể có của mỗi ô nhớ có địa chỉ vật lý sau đây:

   a. 00000h b. 0000Fh c. 00010h d. 0001Fh

   Có nhận xét gì về các kết quả trên?

9. Cho một chương trình bao gồm 100 byte lệnh (Code), 200 byte dữ liệu (Data) và 16 KB ngăn xếp (Stack). Vẽ hình mô tả tổ chức vùng nhớ của chương trình trên

trong bộ nhớ có mô hình như hình A2 theo hai cách sau (sinh viên tự gán địa chỉ vật lý và logic thích hợp với đầu và cuối cho từng vùng):

a. Dùng chung 1 đoạn duy nhất cho cả 3 vùng Code, Data và Stack

b. Dùng 3 đoạn riêng biệt không chồng nhau cho Code, Data, Stack

Địa chỉ logic	**Ô nhớ**	**Địa chỉ vật lý**
….:….		….
…..:…	…..	…..
….:….		….
….:….		….
….:….		….

Hình A2: Mô hình tổ chức bộ nhớ

10. Bằng mô hình bộ nhớ kết quả của câu 9a, hãy thiết lập giá trị các thanh ghi sao cho CPU-8086 truy xuất được những ô nhớ trong mỗi trường hợp sau:

a. Byte lệnh đầu tiên

b. Byte lệnh thứ 20

c. Byte dữ liệu đầu tiên

d. Byte dữ liệu thứ 10

Chương 2
HỢP NGỮ

Mục Đích

- So sánh ngôn ngữ máy và hợp ngữ
- Lệnh giả của MASM
- Các cấu trúc chương trình Hợp ngữ
- Cách tạo chương trình hợp ngữ

2.1. NGÔN NGỮ MÁY VÀ HỢP NGỮ

Chương trình là một tập hợp các lệnh được đưa vào bộ nhớ trong của máy tính để bộ xử lý thực hiện. Các lệnh có thể được thể hiện ở những dạng (ngôn ngữ) khác nhau. Bộ xử lý hoạt động dựa trên kỹ thuật số nên chỉ hiểu được những giá trị nhị phân *(để đơn giản, giá trị nhị phân còn được viết thành giá trị thập lục phân)*. Do đó, ngôn ngữ mà CPU hiểu được ấy gọi là **ngôn ngữ máy** (Machine language). Mỗi bộ xử lý có tập giá trị phân trong đó mỗi giá trị nhị phân sẽ điều khiển CPU thực hiện một tác vụ (gọi là mã lệnh), tập hợp các mã lệnh nhị phân ấy gọi là tập lệnh. Những bộ xử lý khác nhau sẽ có tập lệnh khác nhau, do đó ngôn ngữ máy của CPU nào thì chỉ thực hiện được trên CPU đó. Bảng 2.1 trình bày đoạn chương trình ngôn ngữ máy viết cho họ Intel-8086.

Thứ tự lệnh	Giá trị thập lục phân	Mã lệnh nhị phân
1	B4 09	10110100 00001001
2	BA 03 01	10111010 00000011 00000001
3	40	01000000

Bảng 2.1: Đoạn chương trình ngôn ngữ máy họ Intel-80x86

Đối với CPU có kiến trúc CISC, chiều dài các lệnh có thể khác nhau do đó trong đoạn chương trình trên gồm 3 lệnh, có chiều dài lần lượt là 2 byte, 3 byte và 1 byte. Byte đầu tiên của mỗi lệnh là Tác vụ (Op-Code) mà CPU phải thực hiện còn các byte còn lại là Tác tử (Operand) xác định dữ liệu hoặc nơi chứa dữ liệu mà lệnh tác động vào. Để có thể lập trình với ngôn ngữ máy này, người lập trình phải hiểu rõ tổ chức phần cứng của máy đang sử dụng.

Vì là ngôn ngữ riêng của CPU và được CPU thực hiện ngay khi đọc được lệnh nên chương trình viết bằng ngôn ngữ máy thực hiện rất nhanh và chiếm ít bộ nhớ trong. Tuy nhiên, dạng nhị phân của ngôn ngữ máy rất khó nhớ dễ nhầm lẫn nên khó viết.

Để khắc phục nhược điểm khó nhớ của dạng nhị phân, người ta dùng những từ ngữ dễ nhớ để thay thế cho những mã lệnh nhị phân (gọi là từ gợi nhớ mã lệnh – mnemonic). Chương trình viết bằng Từ gợi nhớ mã lệnh gọi là **Hợp Ngữ** (Assembly).

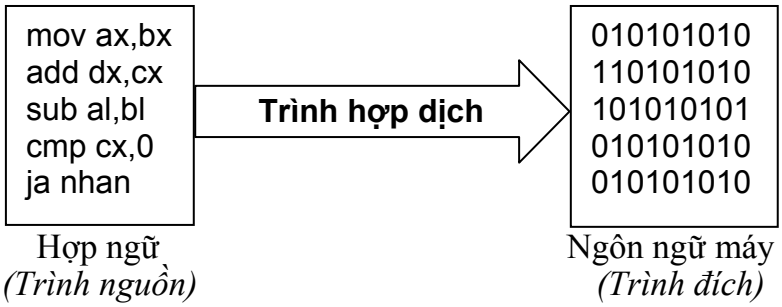
Mỗi mã lệnh nhị phân sẽ có một lệnh tương ứng trong Hợp ngữ và mỗi CPU khác nhau sẽ có Hợp ngữ khác nhau. Các tài liệu tham khảo về tập lệnh của CPU thường cung cấp cho lập trình viên tập lệnh của CPU ở dạng Hợp Ngữ.

Bảng 2.2 viết lại đoạn chương trình trong bảng 2.1 ở dạng Hợp ngữ. Qua đó người đọc có thể hiểu được phần nào chức năng của mỗi dòng lệnh trong đoạn chương trình đó thông qua lệnh Hợp ngữ.

Thứ tự lệnh	Giá trị thập lục phân	Hợp Ngữ	Ý nghĩa
1	B4 09	MOV AH, 09h	AH ← 09h
2	BA 03 01	MOV DX, 0103h	DX ← 0103h
3	40	INC AX	AX ← AX + 1

Bảng 2.1: Đoạn chương trình Hợp ngữ cho họ Intel-80x86

Như vậy, Hợp ngữ sẽ làm cho lập trình viên dễ dàng hơn trong việc viết chương trình nhưng CPU thì không hiểu được những từ gợi nhớ mã lệnh đó. Do đó, từng lệnh trong chương trình viết bằng hợp ngữ phải được dịch sang ngôn ngữ máy tương ứng (hình 2.1). Chương trình làm nhiệm vụ này gọi là Trình hợp dịch (Assembler)



Hình 2.1: Hợp ngữ và Ngôn ngữ máy

Khi mới ra đời, các chương trình hợp ngữ chỉ làm việc đơn thuần là dịch những từ gợi nhớ mã lệnh sang mã máy nhưng dần dần các trình hợp dịch bổ sung thêm các lệnh riêng của trình hợp dịch cho phép dùng các nhãn, các ký hiệu, biến đổi dạng lệnh, khai báo biến, phân phối bộ nhớ, macro… nhằm giúp cho người lập trình viết chương trình hợp ngữ dễ dàng hơn, các lệnh riêng đó gọi là lệnh giả (pseudo code).

Các lệnh giả không có mã máy nhị phân tương ứng nên bản thân các lệnh giả không điều khiển CPU thực hiện tác vụ mà chỉ dùng để hướng dẫn trình hợp dịch trong quá trình dịch chương trình. Do đó, khi gặp từ gợi nhớ mã lệnh, trình hợp dịch sẽ dịch sang mã máy nhị phân; còn khi gặp lệnh giả, trình hợp dịch không dịch thành mã nhị phân.

2.2. ĐẶC TÍNH TỔNG QUÁT CỦA HỢP NGỮ

2.2.1. Cấu trúc của một dòng lệnh hợp ngữ.

Đối với hợp ngữ, trên một dòng văn bản chỉ cho phép viết một dòng lệnh duy nhất. Hình 2.2 trình bày bốn thành phần có thể có trong cấu trúc của dòng lệnh hợp ngữ.

Tên	Mã lệnh	Toán hạng	Chú thích

Hình 2.2: Cấu trúc dòng lệnh hợp ngữ

Bốn thành phần của dòng lệnh hợp ngữ được sắp xếp theo thứ tự trên một hàng lệnh và được phân cách nhau bằng một ký tự đặc biệt gọi là ký tự ngăn cách (delimiter). Các ký tự ngăn cách có thể gồm: Khoảng trống, hai chấm, chấm phẩy, dấu

phẩy, dấu hỏi... tùy theo hợp ngữ. Chính các ký tự ngăn cách này giúp cho trình hợp dịch phân biệt được các vùng khác nhau trong dòng lệnh. Vì vậy, khi khảo sát một hợp ngữ nào đó, chúng ta phải chú ý và sử dụng đúng các ký tự ngăn cách.

1. Tên (name)

Vùng tên cho phép gán tên cho một địa chỉ (nhãn) hay một dữ liệu (Biến, hằng). Khi đó ta có thể sử dụng tên này để thay thế địa chỉ hay dữ liệu khi tham khảo đến địa chỉ hay dữ liệu ấy. Mỗi tên chỉ được thay thế cho duy nhất một địa chỉ hoặc một trong chương trình và được đặt theo quy cách của trình hợp dịch.

2. Mã lệnh

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

- Từ gợi nhớ mã lệnh được trình bày ở chương 3 (Tập lệnh CPU-8086).
- Mỗi trình hợp dịch có thể xây dựng tập lệnh giả riêng, nên trong phần 2.3 sẽ trình bày các lệnh giả thường dùng của trình hợp dịch MASM.

3. Toán hạng: Chứa các toán hạng mà lệnh cần. Số toán hạng tùy vào mã lệnh

4. Chú thích: Dùng để ghi chú, giải thích cho dòng lệnh.

2.2.2. Macro

Macro là một nhóm lệnh nào đó được dùng nhiều lần trong cùng một chương trình nên ta gán cho nó một tên. Mỗi khi sử dụng nhóm lệnh ấy chỉ cần gọi tên đã gán cho nhóm lệnh đó.

2.2.3. Chương trình con

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

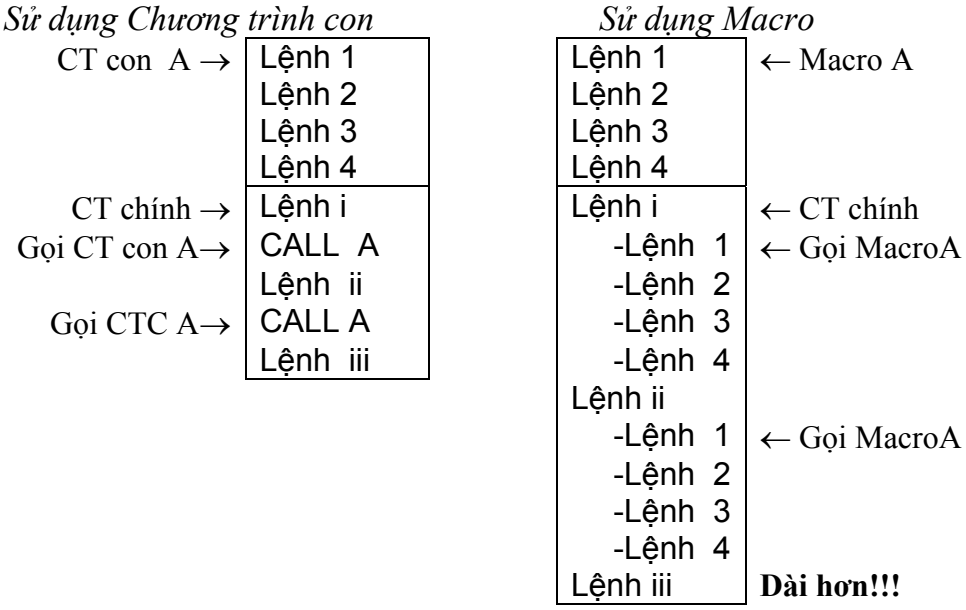
Một số hợp ngữ còn cho phép làm một thư viện chương trình con để sử dụng chung cho nhiều chương trình khác nhau.

Muốn sử dụng chương trình con thì phải dùng lệnh CALL hay lệnh JUMP để chuyển điều khiển đến chương trình con đó. Do đó phải lưu địa chỉ trở về chương trình chính ở ngăn xếp và làm chậm đi việc thực hiện chương trình chính.

Đối với macro, thì mỗi lần chương trình chính gọi macro để thực hiện thì toàn bộ nhóm lệnh trong macro được xen vào ngay điểm gọi, không cần dùng lệnh CALL hay JUMP, nên chương trình chính được thực hiện nhanh hơn .

Tuy nhiên mỗi lần gọi macro thì đoạn mã lệnh trong macro được xen vào chương trình chính làm cho chương trình chính ngày càng dài ra và chiếm nhiều ô nhớ trong hơn.

Chương trình con chỉ chiếm một vùng ô nhớ nhất định. Khi chương trình chính gọi nó thì chương trình chính lưu trữ địa chỉ trở về, nhảy đến địa chỉ bắt đầu của chương trình con để thực hiện chương trình này và khi thực hiện xong thì lấy lại địa chỉ trở về để tiếp tục thực hiện chương trình chánh. Dùng chương trình con ít chiếm bộ nhớ trong.



Hình 2.3: So sánh chương trình dùng Chương trình con và Macro

Trước đây, do bộ nhớ trong của máy tính còn hạn hẹp và tốc độ chưa cao nên việc dùng macro hay chương trình con được xem xét rất cẩn thận đảm bảo được chương trình ngắn và hoạt động không chậm. Ngày nay, dung lượng bộ nhớ trong khá lớn và với tốc độ nhanh của bộ xử lý đã làm cho người lập trình thoải mái hơn trong việc sử dụng macro hay chương trình con.

2.2.4. Biến toàn cục (global), biến địa phương (local)

Các biến được khai báo trong chương trình chính được gọi là biến toàn cục. Các biến này có thể dùng cho chương trình chính, trong macro và các chương trình con.

Các biến được khai báo bên trong macro hay chương trình con được gọi là biến địa phương và chỉ được dùng cho nội bộ một macro hoặc chương trình con. Nói cách khác, nếu có 2 biến cùng tên được khai báo ở 2 chương trình con khác nhau là 2 biến hoàn toàn khác nhau.

Việc hiểu biết cặn kẽ về biến toàn cục và địa phương giúp cho người lập trình quản lý biến và sử dụng biến hiệu quả hơn, tránh trường hợp không bị trùng lấp khi sử dụng tên biến trong chương trình lớn.

2.2.5. Các bảng, thông báo:

Đa số các trình hợp ngữ khi tiến hành hợp dịch có thể cung cấp các bảng và thông báo cho người lập trình. Các bảng thông báo được cung cấp dưới dạng tập tin văn bản, bao gồm các bảng như sau:

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

2.2.6. Hợp ngữ chéo (cross assembler)

Một hợp ngữ chạy trên một máy tính nào đó để dịch ra mã máy cho chương trình viết phục vụ một CPU khác chủng loại với CPU của máy tính mà hợp ngữ đó đang dùng thì được gọi là hợp ngữ chéo.

Ví dụ, hiện nay rất khó tìm được một máy tính dùng CPU-Z80. Muốn dịch một chương trình hợp ngữ P1 thành chương trình mã máy P2 dùng cho CPU-Z80, người ta phải dùng hợp ngữ chéo chạy trên máy IBM-PC (có CPU-8086) thông dụng. Chương trình P2 không chạy được trên máy IBM-PC mà chỉ chạy được trên máy tính hoặc hệ vi xử lý do CPU-Z80 điều khiển.

2.3. HỢP NGỮ MASM (HỢP NGỮ CỦA CPU-8086)

MASM (Microsoft Macro Assembler) là trình hợp ngữ do hãng phần mềm Microsoft phát triển cùng với phiên bản hệ điều hành DOS. Ngoài ra, hãng Borland – chuyên xây dựng các chương trình dịch – cũng phát hành chương trình dịch hợp ngữ gọi là TASM (Turbo Assembler). Các phiên bản này chỉ cho phép viết chương trình trên hệ điều hành DOS, cho nên không viết được chương trình cho Windows bằng các phiên bản này. Hiện nay, đã có các phiên bản cho phép viết chương trình trên Windows, như MASM32 (MASM 32 bits).

Ta dùng các từ gợi nhớ mã lệnh, các lệnh giả, các ký hiệu ... do trình hợp ngữ MASM qui định để viết ra một chương trình mà ta gọi là chương trình hợp ngữ nguồn (source file). Chương trình nguồn được lưu trữ trong đĩa từ dưới một tên tập tin có đuôi là ASM. MASM sẽ dịch chương trình nguồn thành chương trình đối tượng có đuôi OBJ (Object file). Chương trình đối tượng sẽ được liên kết (LINK) thành chương trình có thể chạy được có đuôi là EXE (Executive file).

Các loại hợp ngữ khác nhau đều có cú pháp giống nhau, nhưng cũng có những đặc điểm riêng thể hiện khả năng mạnh mẽ của nó. Sau đây chúng ta nghiên cứu các qui định, các cú pháp thường dùng cho việc viết một chương trình hợp ngữ.

2.3.1. Cấu trúc của một hàng lệnh :

Một hàng lệnh gồm có bốn thành phần:

Tên	Từ gợi nhớ mã lệnh hoặc lệnh giả	Toán hạng hoặc biểu thức	Ghi chú

Hình 2.4: Cấu trúc dòng lệnh hợp ngữ MASM

Ví dụ:

BDAU**:** MOV DX**,** 3F8h **;** 3F8H là địa chỉ cổng vào ra nối tiếp

Trong đó: *BDAU* – Tên với vai trò là nhãn

: – (Dấu 2 chấm) phân cách Tên và Từ gợi nhớ

MOV – Từ gợi nhớ mã lệnh

DX, 3F8h – Hai toán hạng, phân cách bằng dấu phẩy

3F8H là địa chỉ cổng vào ra nối tiếp – Câu chú thích

; – (Dấu ;) Phân cách Toán hạng và Chú thích

2.3.2. Tên:

Tên có thể là *nhãn*, *biến* hay *ký hiệu (hằng)*.

Tên có chiều dài tối đa là 31 ký tự và phải bắt đầu bằng một chữ cái. Các khoảng trống và ký hiệu toán học không được dùng để đặt tên.

1. NHÃN: Nhãn dùng để đánh dấu một địa chỉ mà các lệnh như: lệnh nhảy, gọi chương trình con, lệnh lập cần đến. Nó cũng được dùng cho các lệnh giả LABEL hoặc PROC hoặc EXTRN

Ví dụ:

NH: MOV AX, DS ; NH là nhãn đánh dấu một địa chỉ ô nhớ

FOO LABEL near ; đặt tên cho địa chỉ ô nhớ sau lệnh giả này

CTCON PROC FAR ; địa chỉ bắt đầu của chương trình con

EXTRN **NH** near ; NH là nhãn ngoài chương trình gốc nầy

2. BIẾN: Biến dùng làm toán hạng cho lệnh hoặc biểu thức. Biến tượng trưng cho một địa chỉ nơi đó có giá trị mà ta cần.

Ví dụ: **TWO** DB 2 ; khai báo biến TWO có giá trị là 2

3. KÝ HIỆU: Ký hiệu là một tên được định nghĩa để thay cho một 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 ; ký hiệu FOO thay thế giá trị 07h
TOTO = 0Fh ; ký hiệu TOTO thay thế giá trị 0Fh

2.3.3. Từ gợi nhớ mã lệnh, lệnh giả:

Từ gợi nhớ mã lệnh sẽ được trình bày trong các chương sau về tập lệnh của bộ xử lý 8086. Trong phần này sẽ trình bày các lệnh giả thường dùng của MASM.

Lệnh giả có thể được chia thành 5 nhóm: cấu trúc chương trình, khai báo dữ liệu, dịch có điều kiện, Macro và Liệt kê

A. Nhóm cấu trúc chương trình:

SEGMENT và **ENDS**: Khai báo đoạn.

Cú pháp: <tên đoạn> SEGMENT [align] [combine] [‘class‘]
….. ; nội dung của đoạn
<tên đoạn> ENDS

[align] xác định nơi bắt đầu của đoạn như sau, gồm các giá trị :

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 ở địa chỉ là bội số của 16.

Page : Đoạn phải bắt đầu ở địa chỉ là bội số của 256.

[combine] xác định cách kết hợp phân đoạn này với phân đoạn khác:

PUBLIC: các đoạn cùng tên và cùng class được ghép nối tiếp nhau khi liên kết.

COMMON: các đoạn cùng tên và cùng class được ghép phủ lấp lên nhau khi liên kết.

AT <biểu thức> : Đoạn được đặt tại một địa chỉ là bội số 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: các đoạn cùng tên và cùng class không được ghép vào nhau

Ví dụ: Khai báo hai đoạn có tên là DSEG và CSEG: Không phủ lấp lên nhau

```
DSEG    SEGMENT
        ……. ; Khai báo dữ liệu
DSEG    ENDS
CSEG    SEGMENT
        ……. ; Các lệnh trong đoạn
CSEG    ENDS
```

ASSUME: Chỉ định loại của một đoạn.

Cú pháp: ASSUME <SegReg>: <Tên1>, <SegReg>: <Tên2>

Ví dụ: Chỉ định đoạn có tên DATA là đoạn dữ liệu (DS) và đoạn có tên CODE là đoạn lệnh (CS):

```
ASSUME  DS: DATA, CS: CODE
```

Lưu ý: Lệnh ASSUME chỉ được viết trong đoạn lệnh.

Ví dụ: Báo cho hợp ngữ biết là không có đoạn nào được chỉ định loại. Khi đó, mỗi lần liên hệ đến một nhản (biến) phải dùng cả địa chỉ đoạn của chúng:

```
ASSUME  NOTHING
```

COMMENT: ghi chú chương trình. Có thể viết trên nhiều dòng

Cú pháp: COMMENT * <ghi chú> *

EVEN : Làm cho thanh ghi đếm chương trình PC là giá trị chẵn

EXTRN: Cho biết một tên hay một ký hiệu đã được định nghĩa bên ngoài ở một module khác được sử dụng ở module chương trình hiện tại.

END: Điểm cuối chương trình nguồn và xác định điểm bắt đầu

Cú pháp: END <Nhãn>

Ví dụ: END begin ; điểm bắt đầu chạy CT là nhãn begin

GROUP: Nhóm các đoạn khác nhau có một tên nhóm dùng chung

Ví dụ: Nhóm 3 đoạn có tên DATA1, DATA2 và DATA3 có tên nhóm dùng chung là CGROUP:

```
CGROUP  GROUP   DATA1,DATA2,DATA3
```

INCLUDE : xen một tập tin hợp ngữ khác vào tập tin hiện hành

Ví dụ: Xen tập tin THEM.ASM trên ổ đĩa C:\ASM vào tập tin hiện hành ngay tại vị trí của lệnh giả INCLUDE:

```
INCLUDE    C:\ASM\THEM.ASM
```

LABEL : đánh dấu một địa chỉ là địa chỉ của lệnh hay số liệu kế

Ví dụ:

```
NHF  LABEL  FAR   ; nhản xa, đánh dấu vị trí NH của lệnh kế
NH:  MOV          AX, DATA
CH   DW   100 DUP (0) ; chuỗi từng từ
```

NAME : đặt tên cho một module hợp ngữ

Ví dụ: NAME Cursor ; đặt tên cho modun là cursor

ORG: ấn định địa chỉ bắt đầu cho đoạn chương trình

Ví dụ:

```
ORG  100h
MOV  AX,code ; Lệnh này được đặt tại địa chỉ CS:100h
```

PROC và **ENDP**: Khai báo chương trình con

Cú pháp: <Tên CTC> **PROC** [Near/Far]

<Tên CTC> **ENDP**

Ví dụ:

```
CTCON  PROC        Near
       MOV   AX, 10       ; Bắt đầu chương trình con
       ADD   DX, AX
       RET                ; Trở về CT chính
CTCON  ENDP
```

PUBLIC : khai báo các tên trong module hiện hành mà các module khác có thể sử dụng.

Ví dụ : PUBLIC FOO, NH, TOTO

B. Nhóm khai báo dữ liệu:

Tất cả dữ liệu trong chương trình hợp ngữ được trình hợp dịch chuyển sang dạng nhị phân. Do đó giá trị của dữ liệu có thể được viết ở các dạng như: ký tự, thập phân, nhị phân, bát phân hay thập lục phân.

B.1. Cách viết số:

- Hệ thập phân: 10, 150, 1234
- Hệ bát phân: kết thúc bằng ký tự *O* (hay *o*), như: 10o, 35O, 1230O
- Hệ thập lục phân: phải kết thúc bằng ký tự *H* (hoặc *h*) và bắt đầu là số, như: 10h, 13H, 2Fh, 0D4E1h

- Hệ nhị phân: kết thúc bằng ký tự *B* (hoặc *b*), như: 10b, 01100100B

B.2. Chuỗi ký tự

Ký tự hay chuỗi ký tự phải được kẹp giữa cặp dấu nháy đơn (‘) hoặc cặp dấu nháy kép (“), như: ‘A’, “b”, ‘Hello Assembly’, “This is a string”

Trình hợp dịch chuyển ký tự sang dạng nhị phân tương ứng với mã ASCII, do đó ‘A’, “A”, 41h hay 65 đều có nghĩa như nhau khi lưu trữ.

B.3. Định nghĩa dữ liệu (Khai báo biến)

Cú pháp chung: [Tên Biến] <Loại Biến> <Giá trị>

[Tên biến] đặt theo quy cách của Tên.

Tên biến (tên vùng nhớ), thực chất là địa chỉ tượng trưng của vùng nhớ và được chuyển thành địa chỉ thật sau khi dịch chương trình.

<Loại Biến> xác định kích thước của biến theo byte, bao gồm:

DB (Define Byte): khai báo biến 1 byte (1 ô nhớ)

DW (Define Word): khai báo biến 2 byte (1 từ máy tính)

DD (Define Double word): khai báo biến 4 byte (từ đôi)

DQ (Define Quad word): khai báo biến 8 byte (bốn từ).

DT (Define Ten byte): khai báo biến 10 byte.

<Giá trị> bao gồm các dạng:

- Số hay biểu thức: *01100b, 0A1D3h, 15* hay *(50+10h)*9*
- Ký tự hay chuỗi: *'A', 'CHAO BAN'*
- Rỗng (không gán trước giá trị): ***?***
- Mãng có n giá trị khác nhau: *<trị 1>, <trị 2>, ..., <trị n>*
- Mãng có X giá trị giống nhau: *X DUP(<giá trị>)*

Lưu ý: - Độ lớn của Trị không vượt quá khả năng lưu trữ của vùng nhớ đã định nghĩa bằng <Loại biến>

- Biến chuỗi ký tự phải khai báo bằng DB

Các ví dụ:

- Khai báo biến 1 byte, tên là SO và gán trước trị là 14:

```
SO      DB      14
```

	SO	*(Tên biến/ địa chỉ ô nhớ)*
1 byte nhớ→	0Eh	*(Trị của Biến/ nội dung ô nhớ)*

- Khai báo mãng M gồm 5 phần tử có trị lần lượt 1, 3, 5, 7, 9:

```
M       DB      1, 3, 5, 7, 9
```

M	M+1	M+2	M+3	M+4
01h	03h	05h	07h	09h

- Khai báo biến 2 byte, lưu trữ giá trị 10, tên là WA:

```
WA      DW      10      ;  10 = 000Ah
```

WA	WA+1
0Ah	00h

WB	WB+1
34h	12h

- Khai báo biến lưu trữ giá trị 1234h, đặt tên là WB:

```
WB    DW    1234h
```

- Khai báo biến, lưu trữ chuỗi "Hello", tên là str:

```
str     DB    'Hello'          ;  Lưu trong bộ nhớ bằng các mã ASCII
```

str	str+1	str+2	str+3	str+4
48h	65h	6Ch	6Ch	6Fh

- Khai báo biến mãng Str2 lưu trữ lần lượt các trị 'ABC', 0Ah, 0Dh, '$':

```
Str2    DB    'ABC', 0Ah, 0Dh, '$'          ; Mãng 6 phần tử 1 byte
```

Str2	Str2 +1	Str2 +2	Str2 +3	Str2 +4	Str2 +5
41h	42h	43h	0Ah	0Dh	24h

- Khai báo biến mãng 7 phần tử có cùng giá trị là '$', tên là Str3:

```
Str3    DB    7 DUP('$')      ; Mãng 7 phần tử 1 byte
```

Str3	Str3+1	Str3+2	Str3+3	Str3+4	Str3+5	Str3+6
24h	24h	24h	24h	24h	24h	24h

Trong hợp ngữ, kiểu của biến được hiểu đơn giản hơn thông qua số lượng ô nhớ của biến và được thể hiện bằng *<Loại Biến>*.

Các biến phải được khai báo trong đoạn dữ liệu (DS), một vài trường hợp đặt biệt vẫn được khai báo trong đoạn lệnh, nhưng phải luu ý đến tổ chức chương trình sao cho CPU không xem các biến như là lệnh. Các biến sẽ được phân phối bộ nhớ theo thứ tự được khai báo lần lượt từ ô nhớ có địa thấp đến cao. Trong một đoạn có nhiều biến được khai báo, biến được khai báo đầu tiên sẽ có địa chỉ độ dời trong đoạn dữ liệu bắt đầu là 0h và tiếp theo cho các biến được khai báo tiếp theo sau.

B.4. Dữ liệu có cấu trúc:

RECORD và **STRUC**: Khai báo biến kiểu có cấu trúc mẫu tin.

Cú pháp: [Tên Biến] <Kiểu cấu trúc> [trường: d] [,…]

d: Số nguyên dương, xác định độ lớn của *trường*.

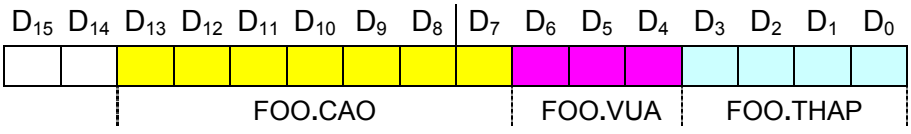
<Kiểu cấu trúc>:

RECORD: d tính bằng bít. Cấu trúc dài tối đa là 16 bít

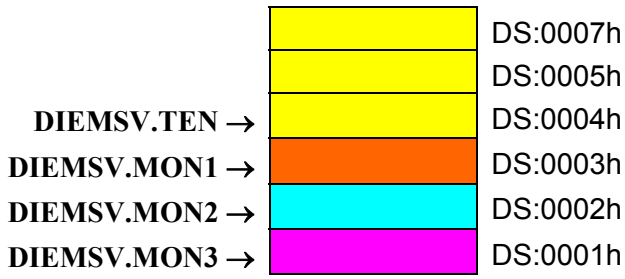
STRUC: d tính bằng byte.

Ví dụ: Hình 2.5 mô tả quản lý vùng nhớ cho 2 biến có cấu trúc sau:

```
FOO       RECORD   CAO : 7, VUA : 3, THAP : 4
DIEMSV  STRUC      TEN: 3, MON1: 1, MON2: 1, MON3:1
```



Hình 2.5a: Mô hình quản lý biến FOO



Hình 2.5b: Mô hình quản lý biến DIEMSV

B.5. Khai báo hằng

EQU và = : gán trị cho 1 ký hiệu.

EQU để gán trị cho ký hiệu (hằng số) chỉ một lần khi khai báo.
Muốn gán lại giá trị nhiều lần, ta dùng lệnh "="

Ví dụ: FOO EQU 2*10 ; gán trị 20 vào FOO
FOO = 2*10 ; gán trị 20 vào FOO

C. Nhóm lệnh giả về dịch (compile) có điều kiện:

Các lệnh giả về dịch có điều kiện nhằm báo cho hợp ngữ tiến hành dịch một nhóm lệnh nếu một điều kiện được thỏa mãn (đúng). Ngược lại sẽ không dịch khi điều kiện không thoả mãn (sai)

Cú pháp chung: <Điều kiện> ; Điều kiện dịch chương trình
Nhóm lệnh A
[ELSE]
Nhóm lệnh B
ENDIF ; Hết điều kiện dịch

<Điều kiện> có thể có những hình thức sau:

IFE <B-Thức> : Nếu <B-Thức> = 0 thì nhóm lệnh A được dịch.
Nếu <B-Thức> ≠ 0 thì nhóm lệnh B được dịch *(nếu có lệnh giả ELSE)*

IF1 : Nếu đang dịch lần 1 thì nhóm lệnh A được dịch

IF2 : Nếu đang dịch lần 2 thì nhóm lệnh A được dịch

IFDEF <ký hiệu>: Nếu ký hiệu đã định nghĩa thì dịch nhóm lệnh A.

IFNDEF <ký hiệu>: Nếu ký hiệu không được định nghĩa thì dịch nhóm lệnh A

IFB <ĐốiSố> : Nếu đối số là khoảng trống hoặc không có đối số thì dịch nhóm lệnh A.

IFNB <ĐốiSố> : Nếu có đối số thì dịch nhóm lệnh A

IFIDN <ĐốiSố1>, <ĐốiSố2>: Nếu ĐốiSố1=ĐốiSố2 thì dịch nhóm lệnh A

IFDIF <ĐốiSố1>,<ĐốiSố2>: Nếu ĐốiSố1 ≠ ĐốiSố2 thì dịch nhóm lệnh A

D. Nhóm lệnh giả về MACRO

Cú pháp: <Tên> MACRO [tham số]

```
….          ; đoạn chương trình
ENDM        ; chấm dứt Macro
```

Ví dụ : Viết MACRO tên **gen** với 3 tham số vào X, Y, Z như sau :

```
gen   MACRO   X, Y, Z
      MOV     AX, X     ; AX ← X
      ADD     AX, Y     ; AX ← AX + Y
      ADD     AX, Z     ; AX ← AX + Z
      ENDM
```

Trong chương trình, gọi macro **gen** bằng lệnh:

```
gen 10, 20, 30          ; X=10, Y=20, Z=30
```

Một đoạn chương trình sau đây được xen vào ngay lệnh gọi macro:

```
MOV     AX, 10          ; AX ← 10
ADD     AX, 20          ; AX ← AX + 20
ADD     AX, 30          ; AX ← AX + 30
```

Ngoài ra, trong Macro ta có thể dùng lệnh giả LOCAL, EXITM:

EXITM : Thoát ra khỏi MACRO trước khi gặp lệnh ENDM.

LOCAL: Định nghĩa các nhãn địa phương trong các MACRO khi chương trinh muốn gọi macro nhiều lần.

Ví dụ:

```
WAIT      MACRO     count
          MOV       CX, count
next:     ADD       AX, 1
          LOOP      next
          ENDM
```

MACRO nầy chỉ được phép gọi một lần vì nhãn next chỉ có thể xuất hiện trong chương trình một lần. Nếu muốn gọi macro WAIT nhiều lần thì ta phải thêm lệnh giả local như sau:

```
WAIT                MACRO     count
                    Local     next   ; nhãn địa phương
                    MOV       CX, count
next:               ADD       AX,1
                    LOOP      next
                    ENDM
```

E. Nhóm lệnh giả về liệt kê (listing)

Nhóm lệnh nầy dùng để điều khiển in ấn chương trình theo 1 định dạng văn bản ở đầu mỗi trang, như: Số trang, số cột, Tựa đề chương trình

PAGE số hạng, số cột

Ví dụ : PAGE 58, 60 ; mỗi trang liệt kê có 58 hàng, 60 cột.

TITLE (đề tựa) : cho phép đặt đề tựa chương trình

Ví dụ : TITLE *chương trình hợp ngữ thứ nhứt*.

SUBTTL (Subtitle: đề tựa con): liệt kê tựa đề con ở mỗi đầu trang.

% OUT <văn bản> : văn bản được liệt kê khi hợp ngữ dịch chương trình. Bao gồm các chỉ thị loại văn bản sau:

.LIST : Liệt kê tất cả dòng lệnh với mã của nó (mặc nhiên)
.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 : Liệt kê bảng đối chiếu chéo
.XCREF: Không liệt kê bảng đối chiếu chéo

2.3.4. Toán hạng và toán tử

A. Toán hạng:

Hợp ngữ phân biệt 3 loại toán hạng: tức thì, thanh ghi và ô nhớ. Để phân biệt, mỗi loại toán hạng có cách viết khác nhau hoàn toàn.

A.1. Toán hạng tức thì: có thể là một số, biểu thức hay một ký hiệu đã được định nghĩa (bằng lệnh EQU hay =).

Toán hạng tức thì có thể được sử dụng theo 4 dạng: thập phân, nhị phân, thập lục phân, bát phân và Ký tự

Cách viết: 10, 'B', 10b, 10h

Ký hiệu tổng quát đại diện cho toán hạng tức thì:

Immed : Toán hạng tức thì nói chung
Immed8 : Toán hạng tức thì 8 bit
Immed16 : Toán hạng tức thì 16 bit
Immed32 : Toán hạng tức thì 32 bit

A.2. Toán hạng thanh ghi: dùng để truy xuất đến nội dung thanh ghi.

Cách viết: AX, BX, CX, DX, AL, AH, SI, DS, ...

Ký hiệu tổng quát đại diện cho toán hạng thanh ghi:

Reg : Toán hạng thanh ghi đa dụng nói chung
Reg8 : Toán hạng thanh ghi đa dụng 8 bit
Reg16 : Toán hạng thanh ghi đa dụng 16 bit
SegReg : Toán hạng thanh ghi đoạn

A.3. Toán hạng ô nhớ: dùng để truy xuất đến nội dung ô nhớ.

Toán hạng ô nhớ thường được tượng trưng một địa chỉ của ô nhớ. Địa chỉ đó luôn luôn là địa chỉ độ dời trong một đoạn tương ứng.

Các cách viết cơ bản:

Tên_Biến : Truy xuất đến nội dung của Biến.

[Immed16] : Truy xuất nội dung của ô nhớ trong đoạn DS, ô nhớ đó có địa chỉ độ dời là *Immed16* . Còn gọi là trực tiếp ô nhớ.

[Reg16] : Truy xuất nội dung của ô nhớ trong đoạn DS, mà ô nhớ đó có địa chỉ độ dời lưu trong *Reg16*. Còn gọi là gián tiếp thanh ghi.

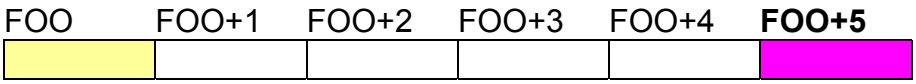
Ví dụ: MOV AH, FOO ; *AH ← nội dung biến FOO*
MOV DH, [1234h] ; *DH ← nội dung ô nhớ có địa chỉ là 1234h*
MOV AL, [BX] ; *AL ← nội dung ô nhớ có địa chỉ trong BX*

Ký hiệu tổng quát đại diện cho toán hạng ô nhớ:

Mem : Toán hạng ô nhớ nói chung
Mem8 : Toán hạng ô nhớ 8 bit
Mem16 : Toán hạng ô nhớ 16 bit
Mem32 : Toán hạng ô nhớ 32 bit

Ví dụ các cách viết khác của toán hạng ô nhớ:

FOO+5 , FOO[5] , 5[FOO] : chỉ đến địa chỉ "FOO cộng 5" *(FOO là tên của 1 biến)*



5[BX][SI] , [BX+5][SI] , [BX]5[SI] : tương đương với [BX+SI+5].

B. Toán tử:

Có 4 loại toán tử: Thuộc Tính, Số Học, Quan Hệ và Logic.

Cú pháp chung: <Toán tử> <Toán hạng>

Toán tử tác động lên toán hạng để cho ra kết quả, kết quả này dùng làm toán hạng trong lệnh.

B.1. Toán tử thuộc tính (attribute):

PTR (pointer): dùng thay đổi kiểu của các địa chỉ, số liệu

Ví dụ : CALL Word PTR [BX+SI] *; Gọi CT Con*

[BX+SI] mặc nhiên trỏ tới ô nhớ (dữ liệu) 1 byte, nhưng nếu muốn lấy dữ liệu 2 byte thì phải dùng *WORD PTR* để đổi.

: (Dấu hai chấm): dùng thay đổi đoạn mặc nhiên.

Ví dụ : MOV AH, ES **:** [BX+SI] ; *AH ← ES:[BX+SI]*

[BX+SI] mặc nhiên là địa chỉ độ dời của dữ liệu trong đoạn DS nhưng nếu dữ liệu trong đoạn ES thì phải viết *ES:*[BX+SI]

SHORT: dùng để thay đổi kiểu mặc nhiên là Near của lệnh JMP (lệnh nhảy) và báo cho hợp ngữ biết chỉ nhảy trong vòng từ -128 đến +127 so với vị trí lệnh JMP

THIS : tạo một toán hạng có giá trị tùy thuộc vào tham số của THIS

Ví dụ: NH EQU THIS BYTE ⇔ NH LABEL BYTE
SCH = THIS NEAR ⇔ SCH LABEL NEAR

SEG : trả về địa chỉ *đoạn* của một nhãn hay biến

Ví dụ: lấy địa chỉ đoạn của biến TENB vào AX

MOV AX, SEG TENB

OFFSET: trả về địa chỉ *độ dời* của một nhãn hay biến

Ví dụ: MOV BX, OFFSET FOO ; độ dời của biến FOO

TYPE: Xác định kiểu của biến hay nhãn

TYPE <*biến*>: cho số byte ô nhớ mà loại biến đó được khai báo (BYTE = 1, WORD= 2, DWORD = 4 ...)

TYPE <*nhãn*>: 0FFFFh nếu nhãn Near; 0FFFEh nếu nhãn Far

LENGTH : Trả về số phần tử của biến mãng khai báo bằng DUP()

SIZE : trả về tổng số byte mà một biến chiếm

Ví dụ: FOO DW 100 DUP (?)

```
MOV AX, type FOO                ; AX ← 2
MOV CX, length FOO              ; CX ← 100
MOV BX, size FOO                ; BX ← 200
MOV DX, type FOO * length FOO ; DX ← 200
```

B.2. Toán tử số học:

Ngoài các toán tử số học thông dụng (+, -, *, /) còn có các toán tử:

MOD : chia lấy số dư

SHR : dịch sang trái

SHL : dịch sang phải

– : Dấu trừ đứng trước một số để chỉ đó là số âm: -5, -300

Ví dụ: MOV AX, 100 **MOD** 17 ; *AX ← 15*

MOV AX, 1100000b **SHR** 5 ; *AX ← 11b*

B.3. Toán tử quan hệ:

Toán tử quan hệ dùng so sánh 2 toán hạng và thường được dùng trong việc dịch chương trình có điều kiện

Cú pháp: <toán hạng 1> <Toán Tử> <toán hạng 2>

<toán tử>:

EQ (Equal): trả về 1 (TRUE) nếu 2 toán hạng bằng nhau

NE (not equal): trả về TRUE nếu 2 toán hạng không bằng nhau

LT (less than): trả về TRUE nếu (toán hạng 1) < (toán hạng 2).

GT (greater than): trả về TRUE nếu (toán hạng 1)> (toán hạng 2)

LE (less than or equal): trả về TRUE nếu (t.hạng 1) ≤ (t.hạng 2)

GE (greater than or equal): trả về TRUE nếu (t.hạng 1) ≥ (t.hạng 2)

B.4. Toán tử logic: so sánh từng bit tương ứng giữa 2 toán hạng.

NOT : trả về TRUE nếu hai toán hạng logic bên trái và phải khác nhau

AND : trả về TRUE nếu cả hai toán hạng logic đều là TRUE.

OR : trả về TRUE nếu một trong hai toán hạng là TRUE

XOR : trả về TRUE nếu hai toán hạng khác nhau.

Ví dụ: MOV AH, 10 ; *AH ← 10*
MOV AL, 15 ; *AL ← 15*
MOV DH, AH *EQ* AL ; *DH ← 0 (vì AH ≠ AL)*
MOV DL, AH *EQ* 10 ; *DL ← 1 (vì AH = 10)*
MOV CH, AL *LT* AH ; *CH ← 0 (vì AL < AH)*
MOV BH, AH *AND* 0Ah ; *BH ← 1 (vì AL < AH)*

2.4. CẤU TRÚC CỦA CHƯƠNG TRÌNH HỢP NGỮ MASM

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

Mỗi module có thể viết theo một cấu trúc tổng quát như sau:

```
PAGE    60, 132
TITLE   ; chọn tiền đề cho module.
….      ; các phát biểu EXTRN hay PUBLIC (nếu có).

MCR     MACRO   ; Nếu có
        .....   ; viết macro
        ENDM

STACK   SEGMENT PARA STACK 'STACK'
        DB 64 DUP (?)
STACK   ENDS

DSEG    SEGMENT PARA PUBLIC 'DATA'
        ….      ; Khai báo Dữ liệu
DSEG    ENDS

CSEG    SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:CSEG, DS:DSEG, SS:STACK
BATDAU:MOV AX, DSEG
        MOV DS, AX          ; khởi động DS

        ....    ; (chương trình chính)

CTC     PROC NEAR  ; Nếu có
        ....       ; viết chương trình con
CTC     ENDP

CSEG    ENDS
        END BATDAU ; Chấm dứt CTrình và chọn địa chỉ bắt đầu
```

2.4.3. Tập tin thi hành dạng COM và dạng EXE

Hợp ngữ MASM cho phép tạo 2 loại tập tin thi hành. Tập tin dạng COM dùng cho các chương trình nhỏ và tập tin dạng EXE dùng để xây dựng các chương trình lớn. Mỗi loại có cấu trúc chương trình khác nhau.

1. Dạng COM:

- Chỉ dùng một đoạn duy nhất, các thanh ghi CS, DS, ES và SS đều có giá trị giống nhau.
- Kích thước chương trình tối đa là 64K.
- 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 một vùng ô nhớ từ độ dời 0 đến 0FFh để chứa thông tin cần thiết cho DOS. Vùng này gọi là vùng PSP (program segment prefix) và tất cả các thanh ghi đoạn đều phải chỉ tới vùng PSP này. Vì thế địa chỉ bắt đầu của tập tin COM phải có địa chỉ độ dời là 100h.

Cấu trúc chương trình điển hình của tập tin dạng COM như sau:

```
….. ; Khai báo hằng số  (Nếu có)
MNAME MACRO
        ….. ; Viết Macro (Nếu có)
         ENDM
CSEG   SEGMENT
        ASSUME    CS: CSEG, DS: CSEG
        ORG  100h
start:  MOV  AX, CSEG
        MOV  DS, AX       ; khởi động DS
        …   ; (Chương trình chính)
        INT   20h    ; thoát
… ; Khai báo dữ liệu (Nếu có)
CT_CON      PROC
            ….. ; Viết Thủ tục (Nếu có)
            RET
CT_CON      ENDP
CSEG   ENDS
        END  start
```

2.4.3.2. Dạng EXE:

- Chương trình lớn và nằm ở nhiều đoạn khác nhau.
- Có thể gọi các chương trình con dạng Far.
- Kích thước tập tin tùy ý.
- Có header ở đầu tập tin để chứa các thông tin điều khiển cần thiết.
- Thi hành chậm hơn tập tin dạng COM.

Tập tin dạng EXE, không dùng lệnh ORG 100h ở đầu chương trình.

Cấu trúc chương trình điển hình của tập tin dạng EXE như sau:

```
…. ; Khai báo hằng số (nếu có)
MNAME           MACRO  ; (Nếu có)
        …       ; Viết lệnh Macro
        ENDM
DSEG    SEGMENT
        …       ; Khai báo dữ liệu
DSEG    ENDS
CSEG    SEGMENT
        ASSUME  CS: CSEG, DS: DSEG
start:  MOV AX, DSEG
        MOV DS, AX      ; khởi động DS
        …….
        …….     ; (Chương trình chính)
        …….
        MOV  AH, 4Ch    ; hay MOV AX,4C00h
        INT    21h      ; thoát
CT_CON          PROC
                ……..    ; Viết các Thủ tục (Nếu có)
                RET
CT_CON          ENDP
CSEG    ENDS
        END   start
```

2.4.4. Ví dụ:

Đoạn chương trình hợp ngữ sau thực hiện việc in ký tự 'B' lên màn hình (sử dụng hàm 02h của ngắt 21h – *Ngắt này được trình bày cụ thể trong chương 4*)

```
MOV   AH, 02h     ; Sử dụng hàm in ký tự của ngắt 21h
MOV   DL, 'B'     ; DL chứa ký tự cần in
INT     21h       ; Gọi ngắt để thực công việc
```

1. Muốn đoạn chương trình này thực thi trên máy tính ở dạng COM thì phải dùng cấu trúc chương trình dạng COM để viết, như sau:

```
CSEG SEGMENT
        ASSUME     CS: CSEG
        ORG  100h
Begin:  MOV AH, 02h         ; Sử dụng hàm in ký tự của ngắt 21h
        MOV  DL, 'B'        ; DL chứa ký tự cần in
        INT    21h          ; Gọi ngắt để thực công việc
        INT 20h
CSEG ENDS
        END Begin
```

2. Muốn đoạn chương trình này thực thi trên máy tính ở dạng EXE thì phải dùng cấu trúc chương trình dạng EXE để viết, như sau:

```
DSEG SEGMENT
        ; Vì không có biến để khai báo nên đoạn DSEG có thể bỏ đi
DSEG ENDS
CSEG SEGMENT
        ASSUME     CS: CSEG, DS: DSEG
Begin:  MOV  AX, DSEG
        MOV  DS, AX
        MOV  AH, 02h        ; Sử dụng hàm in 1 ký tự của ngắt 21h
        MOV  DL, 'B'        ; DL chứa ký tự cần in
        INT    21h          ; Gọi ngắt để thực công việc
        MOV  AH, 4Ch
        INT    21h
CSEG ENDS
        END   Begin
```

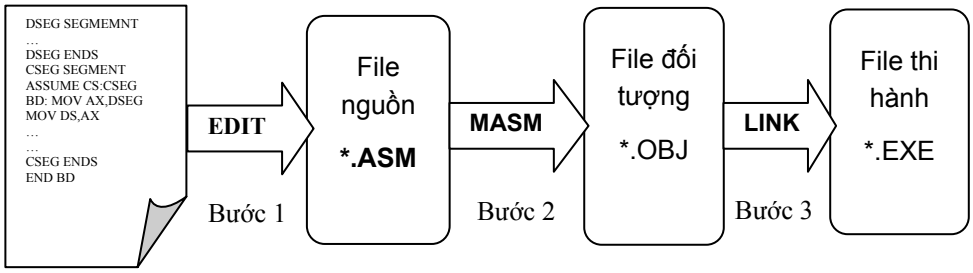
3. Viết lại chương trình trên nhưng không khai báo đoạn DSEG, khi đó chương trình chỉ có 1 đoạn duy nhất, như sau:

```
CSEG SEGMENT
        ASSUME     CS: CSEG, DS: CSEG
Begin:  MOV AX, CSEG
        MOV DS, AX
        MOV AH, 02h         ; Sử dụng hàm in ký tự của ngắt 21h
        MOV  DL, 'B'        ; DL chứa ký tự cần in
        INT    21h          ; Gọi ngắt để thực công việc
        MOV AH, 4Ch
        INT 21h
CSEG ENDS
        END Begin
```

II.5. CÁCH TẠO CHƯƠNG TRÌNH HỢP NGỮ:

Để tạo một chương trình viết bằng hợp ngữ chạy trên máy tính PC, bạn cần phải có bộ trình hợp dịch MASM (hay TASM). Trình hợp dịch MASM được cung cấp gồm các 2 tập tin cơ bản: MASM.EXE (hợp dịch) và LINK.EXE (liên kết). Ngoài ra còn một số tập tin khác như EXE2BIN.COM dùng để chuyển sang dạng COM và DEBUG.COM dùng để gở rối chương trình. Các tập tin này được thực thi dưới MS-DOS.

Cần ba hoặc bốn bước để tạo chương trình hợp ngữ chạy được trên máy tính PC. Hình 2.6 và 2.7 mô tả các bước và lưu đồ thực hiện qui trình này.



Hình 2.6: Các bước tạo chương trình hợp ngữ

BƯỚC 1: Thảo chương, tạo tập tin nguồn (*.ASM)

Dùng bất kỳ trình soạn thảo văn bản dạng TEXT quen thuộc để viết chương trình nguồn hợp ngữ, lưu thành file có phần mở rộng là ***.ASM,** *Ví dụ:* **HELLO.ASM**

BƯỚC 2: Hợp dịch file ASM thành file đối tượng (HELLO.OBJ)

```
C:\> MASM  <filename;>
```

Trình hợp dịch kiểm tra lỗi cú pháp trên từng dòng lệnh. Nếu có lỗi thì phải quay về bước 1 để sửa chữa và dịch lại cho đến hết lỗi.

BƯỚC 3: Liên kết một hay nhiều file OBJ thành file thi hành (*.EXE)

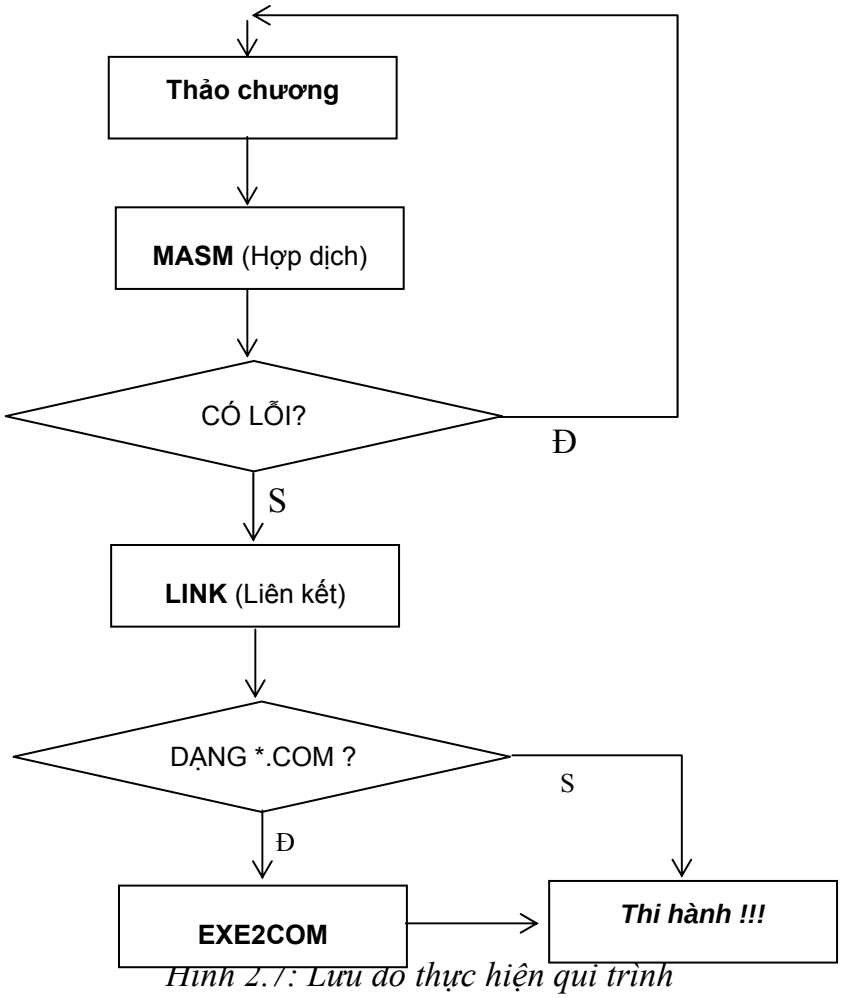
```
C:\> LINK <filename1;>  [<filename2;> …]  [<filename.exe>]
```

BƯỚC 4: Đổi từ EXE sang COM (Dành cho chương trình viết dạng COM)

```
C:\> EXE2BIN  <filename.EXE>  <filename.COM>
```

Ví dụ: Dịch file nguồn HELLO.ASM thành file thực thi HELLO.EXE, từng bước thực hiện các lệnh như sau

```
C:\>MASM HELLO;
C:\>LINK HELLO;
```



Hình 2.7: Lưu đồ thực hiện qui trình

Ngày nay, hầu hết các máy tính đều sử dụng hệ điều hành Windows, nên các lập trình viên đã phát triển các phần mềm chạy trên Windows 98/2000/XP để làm môi trường phát triển Hợp ngữ, như RadASM Assembly IDE (http://www.radasm.com). Khi đó, việc sử dụng các phần mềm này đơn giản hơn rất nhiều thông qua giao diện đồ họa, nó cung cấp sẵn giao diện soạn thảo chương trình nguồn mạnh và các bước hợp dịch, liên kết đều được thực hiện trên thanh công cụ hay bằng phím tắt.

Chương 3

TẬP LỆNH INTEL-8086 và KIỂU ĐỊNH VỊ

3.1. TẬP LỆNH CỦA CPU-8086

Bộ xử lý 8086 có tập lệnh gồm 111 lệnh với chiều dài của lệnh từ 1 byte đến vài byte. Tập lệnh của họ các bộ xử lý Intel 80x86 càng ngày càng có nhiều lệnh mạnh và phức tạp (CPU-80386 có 206 lệnh, Pentium có hơn 400 lệnh).

Lệnh của CPU-8086 có tối đa 2 toán hạng, mỗi lệnh có số toán hạng và loại toán hạng đã được xác định trước. Cú pháp tổng quát như sau:

<Tác Vụ Lệnh> TH1**,** TH2

Tác vụ lệnh cho biết công việc mà CPU sẽ thực hiện (là thành phần bắt buộc), các toán hạng (TH1 và TH2) phải cách nhau bởi dấu phẩy (**,**).

Đối với lệnh có hai toán hạng, thông thường TH2 sẽ là toán hạng nguồn còn TH1 là toán hạng đích và cũng là toán hạng nguồn còn lại. Kích thước của hai toán hạng phải bằng nhau:

- Nếu TH1 là 8 bit thì TH2 cũng phải là 8 bit và ngược lại.
- Nếu TH2 là 16 bit thì TH1 cũng phải là 16 bit và ngược lại.

Tập lệnh CPU-8086 được sắp xếp theo các nhóm như sau:

- Sao chép dữ liệu và địa chỉ
- Tính toán số học
- Tính toán logic, ghi dịch và quay
- Truy xuất ngoại vi
- Lệnh hệ thống
- Rẽ nhánh và vòng lập
- Chương trình con
- Xử lý chuỗi

Chương này chỉ trình bày các nhóm lệnh *đơn giản và thường dùng*. Các nhóm lệnh phức tạp như: rẽ nhánh, vòng lập, chương trình con và xử lý chuỗi) sẽ được trình bày trong những chương sau (chương 5, 6 và 7). Tập lệnh đầy đủ của bộ xử lý Intel-8086 sẽ được giới thiệu trong phụ lục 1.

3.1.1. Lệnh sao chép dữ liệu, địa chỉ:

1. MOV - Di chuyển dữ liệu

Cú pháp: MOV Đích, Nguồn
MOV SegReg, Reg16
MOV Reg16, SegReg

Nguồn (toán hạng nguồn) : Reg, Mem, Immed.
Đích (toán hạng đích) : Reg, Mem.

Ý nghĩa: Sao chép dữ liệu (xác định bởi toán hạng nguồn) vào vùng nhớ hoặc thanh ghi (xác định bởi toán hạng đích). Sau khi thực hiện xong thì giá trị toán hạng nguồn và toán hạng đích bằng nhau.

Lưu ý: - Lệnh MOV không ảnh hưởng đến thanh ghi trạng thái.

- Một trong hai toán hạng phải là thanh ghi.

Ví dụ: MOV CX, BX ; Sao chép nội dung thanh ghi BX vào thanh ghi CX. Sau khi thực hiện xong, BX=CX.

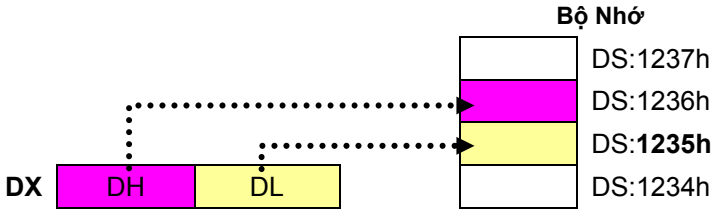
MOV DX, 1234h ; DL ← 34h, DH ← 12h

MOV DS, AX ; DS ← AX

MOV AH, [1234h] ; AH ← M[DS:1234h]

MOV [1235h], DX ; M[DS:1235h] ← DL, M[DS:1236h] ← DH

MOV AH, CX ; **SAI** vì AH có 8 bit trong khi CX là 16 bit.



*Hình 3.1: Mô tả lệnh MOV [**1235h**], DX*

Hình 3.1 mô tả cách sao chép dữ liệu 16 bit giữa thanh ghi và ô nhớ. Byte thấp của dữ liệu trong DX (là DL) đưa vào ô nhớ địa chỉ thấp (DS:**1235h**), còn byte cao trong DX (là DH) đưa vào ô nhớ có địa chỉ cao hơn (DS:1236h)

2. XCHG – Hoán chuyển dữ liệu

Cú pháp: XCHG Đích, Nguồn ; Đích ↔ Nguồn

Nguồn, đích: Reg, Mem.

Ý nghĩa: Lệnh XCHG hoán chuyển dữ liệu giữa nguồn và đích. Cả hai toán hạng không đồng thời là Mem.

Ví dụ: XCHG AL, BH ; AL ↔ BH

XCHG AX, [1235h] ; AL ↔ M[DS:1235h], AH ↔ M[DS:1236h]

XCHG bienA, bienB ; **SAI** vì bienA và bienB đều là vùng nhớ.

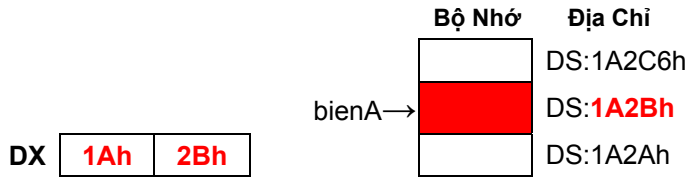
3. LEA - Lấy địa chỉ độ dời

Cú pháp: LEA Reg16, Mem16 ; Reg16 ← Địa chỉ độ dời

Ý nghĩa: Lấy địa chỉ độ dời của biến hay nhãn (mem16) đưa vào thanh ghi đích Reg16.

Ví dụ: LEA DX, bienA ; Lấy độ dời của biến bienA lưu vào DX

Theo hình 3.1, bienA được cấp phát tại vùng nhớ có địa chỉ độ dời là 1A2Bh, sau khi thực hiện lệnh thì thanh ghi DX = 1A2Bh.



Hình 3.2: Mô tả lệnh LEA DX, bienA

3.1.2. Lệnh tính toán số học.

1. ADD - Cộng hai số nguyên

Cú pháp: ADD đích, nguồn ; đích ← đích + nguồn

Nguồn : Reg, Mem, Immed.
Đích : Reg, Mem.

Ý nghĩa: Lấy toán hạng nguồn cộng toán hạng đích và lưu kết quả lưu toán hạng đích.

2. SUB - Trừ hai số nguyên

Cú pháp: SUB đích, nguồn ; đích ← đích – nguồn

Nguồn : Reg, Mem, Immed.
Đích : Reg, Mem.

Ý nghĩa: Lấy toán hạng đích (số bị trừ) trừ toán hạng nguồn (số trừ) và lưu kết quả ở toán hạng đích.

Ví dụ:
```
ADD  AL, 15          ; AL ← AL + 15
ADD  AX, DX          ; AX ← AX + DX
SUB  BX, 15h         ; BX ← BX – 15h
SUB  DX, CL          ; SAI vì DX là 16bit trong khi CL chỉ có 8bit
ADD  15h, AL         ; SAI vì đích là Immed
```

3. INC - Tăng 1 đơn vị (Increment)

Cú pháp: INC đích ; đích ← đích + 1

Đích : Reg, Mem.

Ý nghĩa: Tăng nội dung của toán hạng đích thêm 1 đơn vị.

4. DEC - Giảm 1 đơn vị (Decrmemt)

Cú pháp: DEC đích ; đích ← đích – 1

Đích : Reg, Mem.

Ý nghĩa: Giảm nội dung của đích đi 1 đơn vị.

Ví dụ:
```
INC  AL              ; AL ← AL + 1
DEC  AX              ; AX ← AX - 1
INC  bienC           ; bienC ← bienC + 1
```

5. MUL - Nhân hai số nguyên

Cú pháp: MUL nguồn

Nguồn: Reg, Mem

Ý nghĩa: Nhân thanh ghi tích lũy với toán hạng nguồn. Tùy vào kích thước của toán hạng nguồn mà CPU thực hiện phép nhân 8 hay 16 bit.

- ***Nhân 8 bit:*** dành cho toán hạng nguồn là 8 bit. Khi đó, CPU sẽ lấy thanh ghi AL nhân với toán hạng nguồn và lưu kết quả vào AX.

- ***Nhân 16 bit:*** dành cho toán hạng nguồn là 16 bit. Khi đó, CPU sẽ nhân thanh ghi AX với toán hạng nguồn và lưu kết quả vào cặp thanh ghi DX:AX *(nghĩa là kết quả có 32 bit, 16 bit cao lưu vào DX, còn 16 bit thấp lưu vào AX)*

Ví dụ: MUL BL ; AX ← AL * BL *(nhân 8 bit)*
MUL BX ; DX:AX ← AX * BX *(nhân 16 bit)*
MUL [1235h] ; AX ← AL * M[DS:1235h] *(nhân 8 bit)*

6. DIV - Chia hai số nguyên

Cú pháp: DIV nguồn

Nguồn: Reg, Mem

Ý nghĩa: lấy thanh ghi tích lũy (số bị chia) chia cho nguồn (số chia). Tùy vào kích thước của toán hạng nguồn mà CPU thực hiện phép chia 8 hay 16 bit.

- ***Chia 8 bit:*** dành cho toán hạng nguồn là 8 bit. Khi đó, CPU sẽ lấy thanh ghi AX chia cho toán hạng nguồn. Thương số của kết quả được lưu vào AL còn phần dư lưu vào AH.

- ***Chia 16 bit:*** dành cho toán hạng nguồn là 16 bit. Khi đó, CPU lấy DX:AX (số 32 bit) chia cho toán hạng nguồn. Thương số của kết quả được lưu vào AX còn phần dư thì lưu vào DX.

Ví dụ: DIV DL ; AL, AH ← AX ÷ DL
DIV BX ; AX, DX ← DX:AX ÷ BX

3.1.3. Nhóm lệnh logic và ghi dịch

1. AND, OR, XOR, TEST: Và, Hoặc, Cộng không nhớ, Kiểm tra bit

Cú pháp: AND đích, nguồn ; đích ← đích ∧ nguồn
OR đích, nguồn ; đích ← đích ∨ nguồn
XOR đích, nguồn ; đích ← đích ⊕ nguồn
TEST đích, nguồn ; Cờ ← đích ∧ nguồn
Nguồn: Reg, Mem hay Immed.
Đích: Reg, Mem.

Ý nghĩa: Thực hiện phép toán logic (and, or, xor) theo từng bit tương ứng giữa toán hạng đích và toán hạng nguồn, lưu giữ kết quả ở toán hạng đích.

- Lệnh TEST thực hiện kiểm tra bit, giống lệnh AND nhưng không lưu giữ lại kết quả mà kết quả của phép toán chỉ ảnh hưởng đến các cờ trạng thái.

Ví dụ: MOV AL, 01101110b ; AL ← 01101110b
AND AL, 00110110b ; AL ← AL ∧ 01101110b *(AL=00100110b)*
XOR AX, AX ; AX ← AX ⊕ AX *(Kết quả: AX = 0)*

AND DL, 11000011b ; Xóa 4 bit giữa trong thanh ghi DL

TEST DH, 00010000b ; Kiểm tra giá trị của bit 4 trong thanh ghi DH. Kết quả kiểm tra sẽ ảnh hưởng lên cờ Zero (ZF)

2. NOT – Đảo bit (Lấy Bù 1)

Cú pháp: NOT đích ; đích ← $\overline{\text{đích}}$

Đích: Reg, Mem

Ý nghĩa: Đảo ngược từng bit (hay lấy bù 1) toán hạng đích.

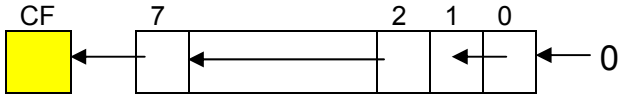
Ví dụ:
```
MOV  AH, 0Fh      ; AH ← 0Fh (00001111b)
NOT  AH           ; AH = 0F0h (11110000b)
```

3. SHL – Dịch trái logic (logical SHift Left)

Cú pháp:
```
SHL đích, 1     ;  dịch toán hạng đích sang trái 1 bit.
SHL đích, CL    ; dịch sang trái CL bít
```

Đích: Reg, Mem

Ý nghĩa: Dịch toán hạng đích sang trái, có thể dịch 1 bit hay nhiều bit hơn (CL chứa số bit dịch). LSB được nạp vào logic 0, còn MSB được dịch sang cờ Carry (CF).



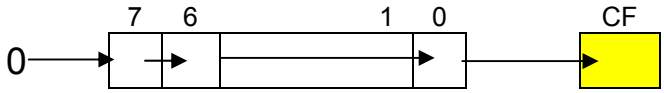
Hình 3.3: Dịch toán hạng đích 8 bit sang trái 1 bit

Ví dụ:
```
MOV  AL, 01101101b      ; AL = 01101101b
SHL  AL, 1              ; AL = 11011010b  và CF=0
```

4. SHR (logical shift right) dịch phải logic.

Cú pháp:
```
SHR đích, 1     ;  dịch trái toán hạng đích 1 bit.
SHR đích, CL    ; dịch trái đích với số bít trong CL
```

Ý nghĩa: Giống lệnh SHL nhưng bay giờ dịch toán hạng đích sang phải



Hình 3.4: Dịch toán hạng đích 8 bit sang phải 1 bit

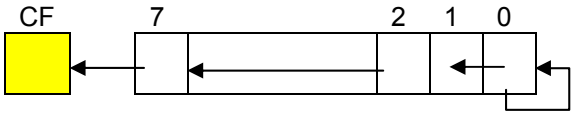
Ví dụ:
```
MOV  AL, 01101101b      ; AL = 01101101b
SHR  AL, 1              ; AL = 00110110b  và CF=1
```

5. SAL (Shift arithmetic left): dịch trái số học

Cú pháp:
```
SAL đích, 1     ;  dịch toán hạng đích sang trái 1 bit.
SAL đích, CL    ; dịch sang trái CL bít
```

Đích: Reg, Mem

Ý nghĩa: Lệnh nầy giống SHL nhưng LSB (bit thấp nhất) vừa được dịch lên bít 1 và giữ nguyên (bảo toàn bit 0).



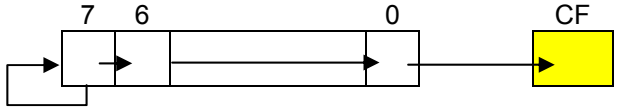
Hình 3.5: Dịch toán hạng đích 8 bit sang trái 1 bit

Ví dụ: MOV AL, 01101101b ; AL = 01101101b
SHL AL, 1 ; AL = 11011011b và CF=0

6. SAR (Shift arithmetic right) : dịch phải số học

Cú pháp: SAR đích, 1 ; dịch toán hạng đích sang trái 1 bit.
SAR đích, CL ; dịch sang trái CL bít
Đích: Reg, Mem

Ý nghĩa: Lệnh nầy giống SHR nhưng MSB (bit cao nhất) vừa được dịch xuống bit thấp hơn và giữ nguyên (bảo toàn bit MSB – *Bit dấu*).



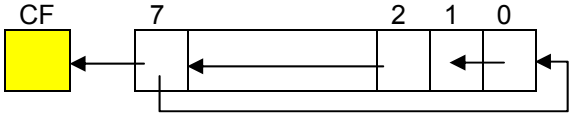
Hình 3.6: Dịch toán hạng đích 8 bit sang phải 1 bit

Ví dụ: MOV AL, 01101101b ; AL = 01101101b
SAR AL, 1 ; AL = 00110110b và CF=1

7. ROL (Rotate left) : quay vòng sang trái

Cú pháp: ROL đích, 1 ; quay vòng toán hạng đích sang trái 1 bit.
ROL đích, CL ; quay vòng sang trái số lần quay bằng CL.

Ý nghĩa: Giống SHL, nhưng thực hiện việc quay vòng, MSB dịch vào LSB



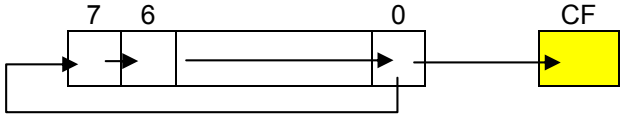
Hình 3.7: Quay toán hạng đích 8 bit sang trái 1 bit

Ví dụ: MOV AL, 01101101b ; AL = 01101101b
ROL AL, 1 ; AL = 11011010b và CF=0

8. ROR (Rotate right) : Quay vòng sang phải

Cú pháp: ROR đích, 1 ; Quay vòng toán hạng đích sang phải1 bit.
ROR đích, CL ; Quay vòng với số lần quay bằng CL
Đích: Reg, Mem

Ý nghĩa: Lệnh nầy giống ROL nhưng quay vòng sang phải, nghĩa là LSB (bit thấp nhất) vừa được dịch đến MSB.



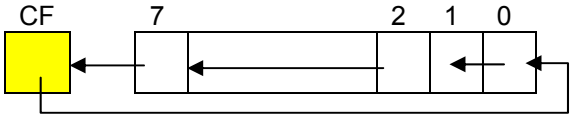
Hình 3.8: Quay toán hạng đích 8 bit sang phải 1 bit

Ví dụ: MOV AL, 01101101b ; AL = 01101101b
SAR AL, 1 ; AL = 00110110b và CF=1

9. RCL (Rotate through carry left): Quay trái qua cờ Carry

Cú pháp: RCL đích, 1 ; quay vòng TH. đích qua cờ sang trái 1 bit.
RCL đích, CL ; quay vòng qua cờ sang trái số lần bằng CL.

Ý nghĩa: Giống như lệnh ROL nhưng có sự tham gia của cờ carry (CF). Xem hình 3.9. Như vậy CF cũng là 1 bit tham gia vào vòng quay

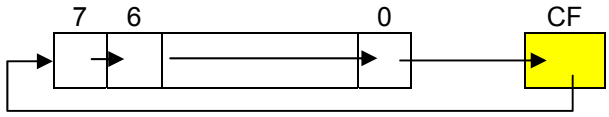


Hình 3.9: Quay toán hạng đích 8 bit qua cờ sang trái 1 bit

10. RCR (Rotate through carry right): Quay vòng qua cờ sang phải

Cú pháp: RCR đích, 1 ; Quay vòng TH. đích qua cờ sang phải 1 bit.
RCR đích, CL ; số lần quay bằng CL
Đích: Reg, Mem

Ý nghĩa: Lệnh nầy giống RCL nhưng quay vòng sang phải.



Hình 3.10: Quay toán hạng đích 8 bit qua cờ sang phải 1 bit

3.1.4. Nhóm lệnh vào ra ngoại vi.

1. IN - Lấy số liệu từ ngoại vi.

Cú pháp: IN AL, địa chỉ cổng 8 bít
IN AL, DX ; DX chứa địa chỉ cổng 16 bit

Ý nghĩa: Lấy dữ liệu ở ngoại vi có địa chỉ 8 bit đưa vào thanh ghi AL. Nếu địa chỉ của ngoại vi là 16 bít thì phải dùng DX chứa địa chỉ này.

Ví dụ : - Đọc dữ liệu ở ngoại vi 3Fh
IN AL, 3Fh ; AL ← port(3Fh) *(3Fh là địa chỉ cổng 8 bít)*

- Đọc dữ liệu ở ngoại vi 3F8h
MOV DX, 3F8h ; DX ← 3F8h *(3F8h là địa chỉ cổng 16 bit)*
IN AL, DX ; AL ← port(DX)

2. OUT - Đưa số liệu ra ngoại vi.

Cú pháp: OUT địa chỉ cổng 8 bít, AL ; port(8 bit) ← AL
OUT DX, AL ; port(DX) ← AL

Ý nghĩa: Giống lệnh IN nhưng là xuất dữ liệu trong AL ra ngoại vi.

Ví dụ: MOV DX, 04A5h ; DX ← 04A5h
OUT DX, AL ; port(DX) ← AL

1.3.5. Nhóm lệnh hệ thống

Nhóm lệnh hệ thống bao gồm các lệnh điều khiển hoạt động của CPU hay thiết lập giá trị các cờ. Các lệnh này không có toán hạng do đó cú pháp của lệnh rất đơn giản chỉ gồm các tác vụ lệnh sau:

CLC ; Xóa cờ CF (CF = 0) – *Clear Carry flag*

CMC ; Đảo ngược cờ CF – *Complement Carry flag*

CLD ; Xóa cờ hướng (DF=0) – *Clear Diretion flag*

STD ; Đặt cờ hướng (DF=1) – *Set Diretion flag*

CLI ; Xóa cờ ngắt (IF=0) – *Clear Interrupt flag*

STI ; Đặt cờ ngắt (IF=1) – *Set Interrupt flag*

HLT ; Dừng mọi hoạt động của CPU (treo máy) – *Halt*

INT ; Gọi ngắt phần mềm – *Interrupt*

IRET ; Trở về chương trình chính từ chương trình phục vụ ngắt – *Return from Interrupt*

LOCK ; Khóa Bus ngoài. CPU sẽ không giao tiếp với BUS

NOP ; Không có tác vụ – *No operation*

WAIT ; Tạm dừng CPU cho đến khi có tín hiệu điện (mức thấp) ở chân TEST của CPU-8086 thì hoạt động tiếp.

3.2. KIỂU ĐỊNH VỊ

Kiểu định vị (Addressing mode) là cách thức xác định vị trí của dữ liệu được mô tả trong câu lệnh. Thông qua kiểu định vị, CPU biết được vị trí lưu trữ toán hạng để truy xuất đến đúng giá trị mà lệnh cần xử lý.

Tùy vào loại toán hạng, CPU-8086 cung cấp các kiểu định vị sau:

- Định vị thanh ghi (register addressing)
- Định vị tức thì (immediate addressing)
- Định vị trực tiếp (direct addressing)
- Định vị gián tiếp thanh ghi (register indirect addressing)
- Định vị nền (based offset addressing)
- Định vị chỉ số (index offset addressing)
- Định vị chỉ số nền (based index offset addressing)
- Định vị chuỗi (string addressing)
- Định vị vào ra (port in/out addressing).

Trong các kiểu định vị trên thì kiểu định vị thanh ghi và tức thì có tốc độ truy xuất nhanh nhất. Đối với toán hạng tức thì, dữ liệu nằm ngay trong lệnh và giá trị của của nó được chèn trực tiếp vào trong mã máy; còn đối với kiểu định vị thanh ghi, do thanh ghi là một bộ phận bên trong của CPU, nên CPU không mất thời gian truy xuất bộ nhớ trong.

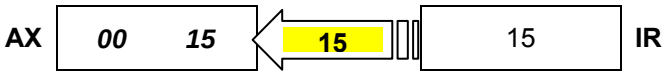
3.2.1. Định vị tức thì:

Là kiểu định vị trong đó phải có toán hạng nguồn là toán hạng tức thì và toán hạng đích phải là toán hạng thanh ghi.

Ví dụ: MOV AX, 15 ; *AX ← 15 (AH=0; AL=15)*
ADD AH, 'A' ; *AH ← AH + 41h ('A' = 41h)*
MOV DX, 3*1Ah ; *DX ← 4Eh (DH=0; DL=4Eh)*
IN AL, 0Fh ; Không phải kiểu định vị tức thì

Các giá trị tức thì là: 15, 'A' và 004Eh

Toán hạng tức thì là dữ liệu nằm ngay trong mã lệnh. Khi khi CPU thực hiện lệnh, mã lệnh được đọc vào thanh ghi lệnh (IR) do đó giá trị tức thì cũng tồn tại trong CPU khi lệnh được thi hành (Xem hình 3.11)



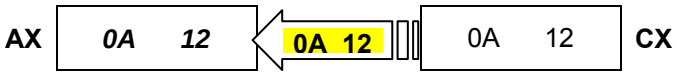
Hình 3.11: Vị trí toán hạng tức thì khi CPU thi hành lệnh

3.2.2. Định vị thanh ghi

Định vị thanh ghi là kiểu định vị mà tất cả các toán hạng đều phải là toán hạng thanh ghi. Kiểu định vị này được sử dụng phổ biến vì có tốc độ nhanh và ít chiếm bộ nhớ.

Ví dụ: MOV AX, CX ; *AX ← CX*
INC AX ; *AX ← AX +1*
OUT DX, AL ; Không phải kiểu định vị thanh ghi

Trong đó AX và CX đều là 2 toán hạng thanh ghi.



Hình 3.12: Hoạt động của lệnh MOV AX, CX

3.1.3. Định vị trực tiếp (bộ nhớ):

Địa chỉ hiệu dụng (EA - Effective Address): Là địa chỉ độ dời (Offset) mà EU tính toán cho một toán hạng bộ nhớ để có thể truy xuất nội dung toán hạng bộ nhớ đó. EU có thể tính địa chỉ hiệu dụng theo nhiều cách. Tùy theo cách tính mà ta có các kiểu định vị bộ nhớ khác nhau.

Định vị trực tiếp là kiểu định vị trong đó chỉ có một toán hạng là địa chỉ độ dời của ô nhớ chứa dữ liệu, toán hạng kia là toán hạng thanh ghi. Địa chỉ hiệu dụng nằm ngay trong lệnh. Địa chỉ hiệu dụng EA kết hợp với nội dung hiện tại của thanh ghi DS trong BIU để tính toán ra địa chỉ vật lý của ô nhớ theo công thức : PA = Segment * 10h + EA

Ví dụ: MOV AH, 12h
MOV FOO, AH ; FOO ← AH (=12h)
MOV BX, [**1234h**] ; BL ← M[DS:**1234h**], BH ← [DS:**1235h**]

Giả sử thanh ghi đoạn dữ liệu có địa chỉ segment là DS=0200h. Vậy địa chỉ vật lý (Physical Address) của toán hạng nguồn được tính như sau:

PA = DS*10h + EA = 200h * 10h + **1234h** = 03234h

Chú ý không có lệnh nào cho phép cả hai toán hạng đều tham chiếu đến vị trí bộ nhớ. Muốn làm được điều này chúng ta phải dùng kiểu định vị gián tiếp thanh ghi.

3.1.4. Định vị gián tiếp thanh ghi

Địa chỉ hiệu dụng là nội dung của một trong các thanh ghi BX, BP, SI, DI. Định vị gián tiếp thanh ghi cũng tương tự như định vị trực tiếp, địa chỉ hiệu dụng EA kết hợp với nội dung của thanh ghi phân đoạn DS để hình thành một địa chỉ vật lý của toán hạng. Tuy nhiên cũng có sự khác nhau về cách tính độ dời. Địa chỉ hiệu dụng EA là nội dung trong các thanh ghi BP, BX, SI và DI bên trong CPU 8086.

Ví dụ: MOV AL, [BX] ; AL ← M[DS:BX]
ADD CX, [BP] ; CX ← CX + M[DS:BP]
MOV [SI], DX ; M[DS:SI] ← DL, M[DS:SI+*1*] ← DH

Giả sử SI = 1234h và DS = 0300h, địa chỉ vật lý được tính là:

PA = DS * 10h + EA = 0300h*10h + 1234h = 04234h

(với EA = SI = 1234h)

3.1.5. Định vị nền

Giống như định vị gián tiếp thanh ghi, nhưng địa chỉ hiệu dụng EA trong trường hợp này là tổng của nội dung của thanh ghi BX hoặc BP với độ dời (giá trị 16 bit có dấu):

EA = BX + Độ Dời *hay* EA = BP + Độ Dời

Địa chỉ vật lý là tổng giá trị độ dời đô dịch chuyển trực tiếp hoặc gián tiếp nội dung của thanh ghi BX hoặc BP với giá trị chứa trong thanh ghi phân đoạn DS hoặc SS tương ứng.

Ví dụ: MOV AL, [BX + 20h] ; AL← M[DS:BX+20h]
MOV [BX].beta, CX ; M[DS:BX+beta] ← CL
; M[DS:BX+beta+*1*] ← CH

Giả sử DS = 0200h, BX = 1000h còn beta = 0FA34h. Vậy địa chỉ vật lý của toán hạng đích trong trường hợp này là:

PA = 0200h*10h + 1000h + 0FA34h = 10C34h

Nếu thanh ghi BP được sử dụng thay vì sử dụng thanh ghi BX thì việc tính toán địa chỉ vật lý của toán hạng đích khi đó sẽ phải sử dụng thanh ghi phân đoạn SS thay vì là DS như trên. Điều này cho phép truy xuất đến dữ liệu trong phân đoạn ngăn xếp của bộ nhớ.

3.1.6. Định vị chỉ số

Địa chỉ hiệu dụng là tổng của độ dời 16 bit và thanh ghi SI hoặc DI

EA = SI + Độ dời *hoặc* EA = DI + Độ dời

Cũng tương tự như kiểu định vị nền, địa chỉ vật lý là tổng độ dịch chuyển trực tiếp hoặc gián tiếp nội dung của thanh ghi SI hoặc DI với giá trị chứa trong thanh ghi phân đoạn DS hoặc ES tương ứng.

Ví dụ: MOV DL, [SI+ 25h] ; DL ← M[DS:SI+25h]

MOV AL , ARRAY[SI] ; AL ← M[DS:SI+ARRAY]

Giả sử DS = 0200h, SI = 2000h còn ARRAY là độ dời có giá trị là 1234h. Vậy địa chỉ vật lý của toán hạng đích trong trường hợp này là:

EA = SI + ARRAY = 2000h + 1234h = 3234h

PA = 0200h*10h + EA = 2000h + 3234h = 05234h

Nếu thanh ghi DI được sử dụng thay vì sử dụng thanh ghi SI thì việc tính toán địa chỉ vật lý của toán hạng đích khi đó sẽ phải sử dụng thanh ghi phân đoạn ES thay vì là DS như trên.

Ví dụ: MOV AL, [**DI** +1Fh] ; AL ← M[**ES**:DI+1Fh]

PA = **ES** * 10h + **DI** + 100h

3.1.7. Định vị chỉ số nền

Đây là kiểu định vị kết hợp từ kiểu định vị nền và định vị chỉ số. Địa chỉ hiệu dụng là tổng của thanh ghi nền (BX hay BP), thanh ghi chỉ số (SI hay DI) và độ dời 16 bit.

EA = BX + **SI** + Độ Dời ⇒ PA = **DS** * 10h + EA

EA = BX + **DI** + Độ Dời ⇒ PA = **ES** * 10h + EA

Ví dụ: MOV DL, [BX+SI+100] ; PA = **DS** * 10h + BX + **SI** + 100

MOV AL, [BX+DI+200] ; PA = **ES** * 10h + BX + **DI** + 200

3.1.8. Định vị chuỗi

Các lệnh xử lý chuỗi không dùng kiểu định vị bộ nhớ thông thường mà dùng 2 thanh ghi chỉ số SI và DI để chứa địa chỉ độ dời của chuỗi nguồn và chuỗi đích. Các lệnh xử lý chuỗi trong 8086 tự động dùng các thanh ghi chỉ mục nguồn và đích để tính toán địa chỉ của toán hạng nguồn và đích tương ứng. Kiểu định vị này gọi là định vị chuỗi và chỉ áp dụng cho các lệnh xử lý chuỗi.

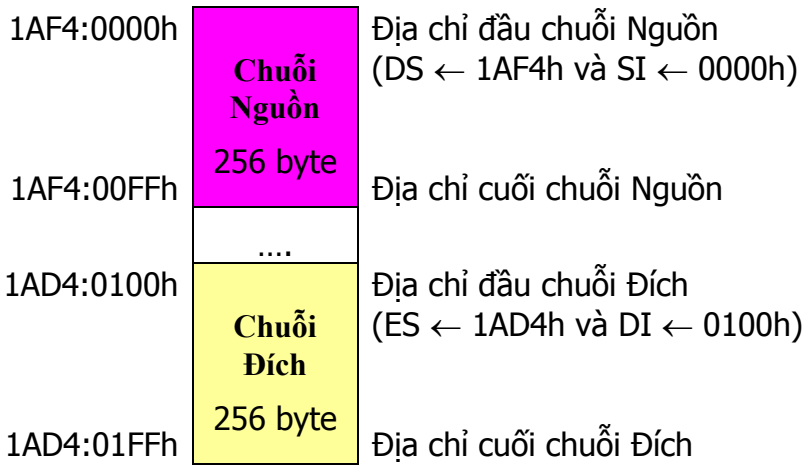
Các lệnh xử lý chuỗi của Intel-8086 được trình bày kỹ trong chương 6, bao gồm: MOVSB/W, SCASB/W, CMPSB/W, STOSB/W, LODSB/W.

Lệnh sẽ xử lý 1 lần 1 **B**yte hoặc 2 byte (**W**ord) - *còn gọi là phần tử*. Sau khi thực hiện xong lệnh, SI và DI tự động điều chỉnh tăng/giảm 1 đơn vị để chỉ đến phần tử kế tiếp cần xử lý.

SI và DI tự động cùng tăng hay cùng giảm được quyết định bởi cờ Hướng (DF). Nếu DF = 0 thì xử lý theo chiều tăng địa chỉ, còn DF = 1 thì xử lý theo chiều giảm địa chỉ. Tùy vào cách xử lý chuỗi mà người lập trình chọn hướng xử lý thích hợp. Khi đó:

- DS:SI phải chỉ đến byte đầu tiên cần xử lý của chuỗi nguồn.
- ES:DI phải chỉ đến byte đầu tiên cần xử lý của chuỗi đích.

Hình 3.13 mô tả vị trí 2 chuỗi Nguồn và Đích (mỗi chuỗi 256 byte) trong bộ nhớ với chiều xử lý tăng địa chỉ.



Hình 3.13: Mô hình bộ nhớ chứa chuỗi Nguồn và Đích

3.1.9. Định vị cổng vào/ra

Định vị cổng vào/ra chỉ áp dụng các lệnh vào ra ngoại vi trong tập lệnh của 8086 (IN và OUT) để truy xuất đến các port vào/ra của thiết bị ngoại vi. Có 2 cách truy xuất đến không gian địa chỉ của Port vào ra:

- **Trực tiếp:** Chỉ dùng cho cổng có địa chỉ 8 bit. Địa chỉ cổng được cung cấp trực tiếp trên dòng lệnh.

Ví dụ:
```
IN   AL, 0F8h ; Đọc dữ liệu tại cổng 0F8h đưa vào AL
OUT  01h, AL  ; Xuất giá trị AL ra cổng có địa chỉ 01h
```

- **Gián tiếp:** Đối với những cổng có địa chỉ 16 bit thì không sử dụng cách trực tiếp để truy xuất được, khi đó thanh ghi DX được dùng để chứa địa chỉ của thiết bị ngoại vi.

Ví dụ:
```
IN   AL, DX   ; AL ← port(DX)
OUT  DX, AL   ; port(DX) ← AL
```

BÀI TẬP CHƯƠNG 3

3.1. Cho biết nội dung của các toán hạng đích và giá trị các cờ CF, SF, ZF, PF, OF sau khi thực hiện mỗi lệnh sau:

a. ADD AX, BX ; Với AX = 7FFFh, BX = 1
b. SUB AL, BL ; Với AL = 1, BL = 0FFh
c. DEC AL ; Với AL = 0
d. NEG AL ; Với AL = 7Fh
e. XCHG AX, BX ; Với AX = 1ABCh, BX = 712Ah
f. ADD AL, BL ; Với AL = 80h, BL = 0FFh
g. SUB AX, BX ; Với AX = 0, BX = 8000h
h. NEG AX ; Với AX = 1

3.2. Cho biết kết quả sau khi thực hiện lệnh ADD AX, DX và trạng thái cờ tràn (OF) trong các trường hợp sau:

a. AX = 512Ch , DX = 4185h
b. AX = 0FE12h, DX = 1ACBh
c. AX = 0E1E4h, DX = 0DAB3h
d. AX = 7132h , DX = 7000h
e. AX = 6389h , DX = 1176h

3.3. Cho biết kết quả sau khi thực hiện lệnh SUB CX, BX và trạng thái cờ tràn (OF) trong các trường hợp sau:

a. CX = 2143h , BX = 1986h
b. CX = 81FEh , BX = 1986h
c. CX = 19BCh , BX = 81FEh
d. CX = 02h , BX = FE0Fh
e. CX = 8BCDh , BX = 71ABh

3.4. Cho AL = 0CBh và CF = 1. Cho biết giá trị AL sau khi thực hiện lệnh:

a. SHL AL, 1
b. SHR AL, 1
c. ROL AL, CL ; Với CL =2
d. ROR AL, CL ; Với CL = 3
e. SAR AL, CL ; Với CL = 2
f. RCL AL, 1
g. RCR AL, CL ; Với CL = 3

3.5. Dùng lệnh TEST để thiết lập giá trị các cờ sau:

a. ZF = 1 nếu AX = 0
b. ZF = 0 nếu AX lẻ
c. SF = 1 nếu DX < 0
d. ZF = 1 nếu DX >= 0
e. PF = 1 nếu BL chẵn.

3.6. Viết đoạn lệnh thực hiện từng yêu cầu sau:

a. Xóa các bit ở vị trí chẵn của AX, giữ nguyên các bit khác.
b. Đặt LSB và MSB của BL, giữ nguyên các bit khác.
c. Đảo MSB của AL, giữ nguyên các bit khác.
d. Xóa bốn bít cao của DL, giữ nguyên các bit khác.

3.7. Cho biết nội dung DX, AX và các cờ OF, CF sau khi thực hiện lệnh:

a. MUL BX ; với AX = 08h, BX = 03h
b. MUL BX ; với AX = 0FFFFh, BX = 1000h
c. DIV BX ; với AX = 07h, BX = 02h, DX = 0h
d. DIV CX ; với AX = 0FFFEh, CX = 10h, DX = 0h

3.8. Cho biết nội dung AX và các cờ OF, CF sau khi thực hiện lệnh:

a. MUL BL ; với AL = 0ABh, BL = 10h
b. MUL AH ; với AL = 0ABh, AH = 01h
c. DIV BL ; với AX = 0Dh, BL = 03h
d. DIV DL ; với AX = 0FEh, DL = 10h

3.9. Xem mô hình bộ nhớ trong hình C1. Cho AX = 500h, BX = 1000h, SI = 1500h, DI = 2000h, BETA là biến kiểu word nằm ở địa chỉ 1000h. Trong các lệnh sau đây, nếu hợp lệ thì hãy cho biết địa chỉ của các toán hạng nguồn hoặc thanh ghi và kết quả lưu trong toán hạng đích. Nếu không hợp lệ thì giải thích tại sao?

a. MOV DL, SI
b. MOV DX, [DI]
c. ADD AX, [SI]
d. SUB BX, [DI]
e. LEA BX, BETA[BX]
f. ADD [SI], [DI]
g. ADD BH, [BL]
h. ADD AH, [SI]
i. MOV AX, [BX+DI+BETA]

Ô nhớ	Địa chỉ
03h	DS:3001h
00h	DS:3000h
02h	DS:2501h
50h	DS:2500h
02h	DS:2001h
00h	DS:2000h
01h	DS:1501h
50h	DS:1500h
01h	DS:1001h
00h	DS:1000h

Hình C1: Mô hình bô nhớ

3.10. Với khai báo dữ liệu:

```
A       DB   1, 2, 3
B       DW   4, 5, 6
C       LABEL    WORD ; C chỉ là 1 nhãn kiểu word
                      ; (không chiếm vùng nhớ)
MSG     DB   'ABC'
```

và giả sử BX chứa địa chỉ của C. Hãy cho biết lệnh nào hợp lệ (nếu không thì giải thích) và trị của các toán hạng đích sau khi thi hành lệnh trong các trường hợp sau:

a. MOV AH, BYTE PTR A
b. MOV AX, WORD PTR B
c. MOV AX, C
d. MOV AX, MSG
e. MOV AH, BYTE PTR C

3.11. Dùng thanh ghi BP và kiểu định vị nền để viết các lệnh thực hiện yêu cầu sau (không được dùng các lệnh PUSH và POP)

a. Đưa 0 vào hai word ở đỉnh ngăn xếp.

b. Sao chép 5 word ở đỉnh ngăn xếp vào mãng ARR lần lượt theo thứ tự (nghĩa là phần tử ở đỉnh ngăn xếp vào phần tử 1 của ARR, phần tử kế trong ngăn xếp vào phần tử thứ 2 của ARR, ...)

Chương 4
HỆ THỐNG NGẮT MỀM

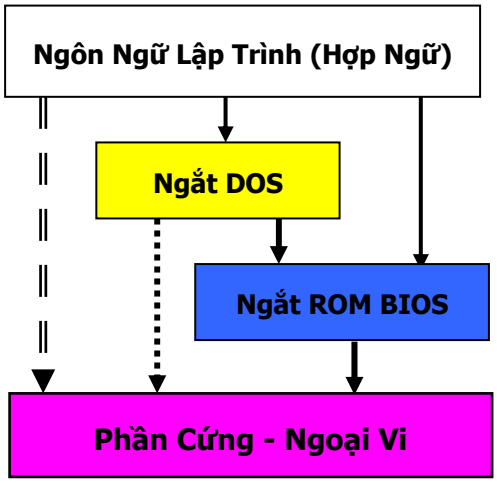
4.1. NHỮNG CƠ SỞ CỦA NGẮT MỀM

Một bí quyết để lập trình thành công và hiệu quả cao trên máy tính kiểu IBM-PC là việc sử dụng có hiệu quả các chương trình phục vụ của BIOS (Basic Input Output System) được cài đặt sẵn ngay trong vùng nhớ ROM-BIOS của máy tính hay các chương trình phục vụ của hệ điều hành DOS. Ngắt mềm (hay Ngắt) là cách gọi tên ngắn gọn của các chương trình phục vụ này.

Ngắt mềm là chương trình mã máy nhị phân (chính xác hơn là chương trình con) được thiết kế sẵn và tồn tại trong bộ nhớ của máy tính nhằm thực hiện các chức năng điều khiển phần cứng máy tính mỗi khi được gọi thực hiện. Khi đó, để điều khiển phần cứng nào thì người lập trình chỉ cần gọi các ngắt tương ứng để thực hiện công việc mà không cần phải biết đến cấu trúc của phần cứng là như thế nào, nguyên lý vận hành của phần cứng đó ra sao…. đây là công việc rất khó khăn đối với người làm phần mềm không quen thuộc với phần cứng là những mạch điện tử.

Các ngắt mềm xem như là cầu nối giữa phần cứng và các ngôn ngữ lập trình (kể cả hệ điều hành). Chúng làm việc trực tiếp với phần cứng và các thiết bị ngoại vi, thực hiện những chức năng cơ bản nhất của hệ thống như đọc/ghi từng byte dữ liệu ở bàn phím, ra màn hình hoặc ở đĩa từ…

Hình 4.1 trình bày các cách giao tiếp với phần cứng của ngôn ngữ lập trình thông qua các ngắt hay trực tiếp. Giao tiếp trực tiếp sẽ khó khăn hơn rất nhiều so với giao tiếp thông qua các Ngắt.



Hình 4.1: Giao tiếp phần cứng

Các ngắt của BIOS thì luôn sẵn có trong máy tính IBM-PC, còn các ngắt DOS chỉ tồn tại đối với hệ điều hành DOS. Nghĩ là, nếu máy tính IBM-PC được cài đặt hệ điều hành UNIX thì trên máy đó không tồn tại các ngắt DOS. Hệ điều hành Windows được phát triển trên cơ sở của hệ điều hành DOS nên các ngắt DOS vẫn sử dụng được trên hệ điều hành Windows. Tất nhiên có một vài ngắt DOS không được phép sử dụng vì đảm bảo độ an toàn cho vận hành của hệ điều hành Windows.

Toàn bộ tài liệu về các ngắt mềm được nhiều sách về lập trình hợp ngữ nêu lên đầy đủ. Giáo trình này chỉ giới thiệu những ngắt rất cơ bản và thường dùng.

4.2. SỬ DỤNG NGẮT TRONG HỢP NGỮ

Khi sử dụng ngắt cần lưu ý một số yếu tố:

- Chức năng của ngắt/hàm.
- Tên ngắt/hàm: Dùng số hex từ 0h đến 0FFh
- Tham số vào: Thanh ghi hay bộ nhớ do hàm qui định.
- Tham số ra: Thanh ghi hay bộ nhớ do hàm qui định.

Lệnh gọi 1 ngắt hay 1 hàm:

MOV AH, <tên hàm>

.... ; Khai báo tham số vào (nếu có)

INT <tên ngắt>

.... ; Lưu trữ tham số ra (nếu có)

Ví dụ: Hàm 02h của ngắt 21h thực hiện việc in ký tự trong thanh ghi DL ra màn hình. Như vậy, khi muốn hiển thị ký tự B trên màn hình, ta viết như sau đoạn lệnh như sau:

```
MOV  AH, 02h      ; Chọn hàm 02h
MOV  DL, 'B'      ; DL chứa ký tự cần in 'B'
INT  21h          ; Gọi ngắt thực hiện để in ký tự
```

Lưu ý: Việc sử dụng thanh ghi DL để chứa ký tự cần in là do hàm 02h qui định, không thể sử dụng thanh ghi khác để thay thế.

4.3. NGẮT MS-DOS

Các ngắt của DOS được đánh số từ 20h đến 0FFh. Ngắt quan trọng nhất là ngắt 21h – gồm nhiều chức năng (hàm) của DOS.

INT 20h: Chấm dứt chương trình và trở về DOS từ một chương trình có đuôi COM.

INT 23h: Chặn tổ hợp phím Ctrl- Break

Người sử dụng có thể chấm dứt sớm một chương trình bằng cách bấm tổ hợp phím Ctrl-Break.

Tổ hợp phím nầy làm CPU nhảy tới chương trình phục vụ ngắt 23h và sau đó đến một thủ tục của DOS để kết thúc chương trình đang chạy và trở về DOS.

Nếu muốn không cho phép chương trình kết thúc sớm bằng Ctrl-Break, ta có thể cho ngắt 23h trỏ đến một chương trình đặt biệt nào đó để thông báo lên màn hình rằng không cho chương trình đang chạy kết thúc sớm.

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 27h, ta cho chạy một chương trình gồm 2 phần. Một phần sẽ phụ trách việc thường trú cho phần kia.

INT 21h: Các chức năng của DOS

Hàm 01h	Đợi đọc 1 ký tự từ bàn phím *(có hiện trên màn hình)*
Vào	Không
Ra	AL : Mã ASCII của phím nhận được

Đợi đọc một ký tự từ bàn phím (phím nhận được sẽ hiển thị trên màn hình). Khi một phím được ấn thì ký tự tương ứng với phím đó được lưu trong thanh ghi AL. Nếu phím được ấn là một phím đặt biệt thì AL=0. Tổ hợp phím Ctrl-Break kết thúc công việc của hàm này.

Hàm 02h	Hiển thị 1 ký tự tại vị trí con trỏ trên màn hình
Vào	DL ← Mã ASCII của ký tự cần hiển thị
Ra	Không

Hàm 03h	Đọc dữ liệu vào từ cổng nối tiếp chuẩn (COM1)
Vào	Không
Ra	AL : byte dữ liệu đọc được

Hàm 04h	Xuất ký tự ra cổng nối tiếp chuẩn (COM1)
Vào	DL ← ký tự cần xuất
Ra	Không

Hàm 05h	Xuất byte dữ liệu ra cổng máy in chuẩn (LPT1)
Vào	DL : Byte dữ liệu cần xuất
Ra	Không

Hàm 06h	Nhập / Xuất từ thiết bị chuẩn (không đợi)
Vào	DL ← Mã ASCII ký tự cần xuất lên màn hình Nếu DL ← 0FFh : Thực hiện chức năng nhập ký tự
Ra	Nếu ZF = 0 thì AL chứa Mã ASCII ký tự nhận được Nếu ZF = 1 thì không nhận được ký tự

Hàm 07h	Giống hàm 1, nhưng không hiện ký tự ra màn hình
Vào	Không
Ra	AL : Mã ASCII ký tự nhận được

Hàm 08h	Giống hàm 1, nhưng không hiện ký tự ra màn hình
Vào	Không
Ra	AL : Mã ASCII ký tự nhận được

Lưu ý: Ctrl-Break không ảnh hưởng đến hoạt động của hàm 07h và 08h

Hàm 09h	Xuất 1 chuỗi ký tự lên màn hình
Vào	DS:DX ← địa chỉ logic của chuỗi ký tự cần xuất
Ra	Không

Lưu ý: DS ← địa chỉ segment của chuỗi trong bộ nhớ

DX ← địa chỉ offset của chuỗi trong bộ nhớ

Chuỗi ký tự phải chấm dứt bằng ký tự '**$**'

Ví dụ: In chuỗi "Hello Assembly" ra màn hình.

Khai báo biến chứa chuỗi trong đoạn dữ liệu DS:

```
chuoi   DB   'Hello Assembly$'
```

Viết chương trình trong đoạn lệnh:

```
MOV  AH, 09h       ; Gọi hàm 09h
LEA  DX, chuoi     ; DX ← Offset(chuoi)
INT  21h           ; Thực hiện hàm
```

Hàm 0Ah	Đợi đọc 1 chuỗi ký tự từ bàn phím, kết thúc bằng Enter
Vào	DS:DX ← địa chỉ của vùng đệm bàn phím trong bộ nhớ
Ra	LEN: Tổng số ký tự nhận được BUFF: Mã ASCII của các ký tự nhận được

Hàm 0Ah nhận 1 chuỗi ký tự từ bàn phím, kết thúc bằng phím Enter (Mã ASCII là 0Dh). Hàm trả về chuỗi nhận được và chiều dài chuỗi trong vùng đệm bàn phím.

Vùng đệm bàn phím là 1 biến mãng trong bộ nhớ gồm 3 phần MAX, LEN và BUFF có cấu trúc như sau:

MAX	LEN	BUFF	BUFF+1	BUFF+2	.....	....	BUFF+n
					…..	…..	

MAX: Phần tử chứa số ký tự tối đa có thể nhận được gán khi khai báo.

LEN: Phần tử chứa tổng số ký tự của chuỗi nhận được, do hàm trả về sau khi nhận được chuỗi.

BUFF: Phần tử chứa mã ASCII của từng ký tự trong chuỗi nhận được *(mỗi ký tự chiếm 1 byte nhớ)*. Như vậy, số byte của BUFF phải bằng hay lớn hơn giá trị của MAX.

Ví dụ: Nhận chuỗi tối đa 30 ký tự từ bàn phím

Khai báo vùng đệm trong đoạn dữ liệu:

```
MAX  DB   30          ; Tạo biến MAX
LEN  DB   ?           ; Tạo biến LEN
BUFF DB   31 DUP(?)   ; Vùng nhớ lưu trữ ký tự
```

Viết chương trình trong đoạn lệnh:

```
MOV  AH, 0Ah
LEA  DX, MAX     ; DX ← Đ/c vùng đệm bàn phím
INT  21h
```

Cách khai báo biến vùng đệm bàn phím khác:

```
BANPHIM   DB   30,  ?,  31 DUP(?)
```

Hàm 0Bh	Cho trạng thái bàn phím
Vào	DS:DX ← địa chỉ logic của vùng đệm bàn phím trong bộ nhớ
Ra	LEN: Tổng số ký tự nhận được BUFF: Mã ASCII của các ký tự nhận được

Hàm 3Ch	Tạo tập tin mới
Vào	DS:DX ← Địa chỉ chuỗi tên của tập tin mới. CX ← thuộc tính của tập tin (0: bình thường, 1: chỉ đọc, 2: ẩn, 4: tập tin hệ thống)
Ra	AX : Thẻ tập tin

Hàm 3Dh	Mở tập tin trên đĩa
Vào	DS:DX ← Địa chỉ chuỗi tên của tập tin mới. AL ← kiểu truy xuất tập tin (0: chỉ đọc, 1: chỉ ghi, 2: đọc/ghi)
Ra	AX : Thẻ tập tin

Khai báo tên tập tin và biến chứa thẻ file:

```
Tenf    DB    "C:\DULIEU\DATA.TXT",0
Thef    DW    ?
```

Chuỗi Tên tập tin có chứa cả tên đường dẫn của file. Nếu không có tên đường dẫn thì sẽ truy xuất trong thư mục hiện hành.

Thẻ tập tin là thành phần quan trọng trong việc truy xuất file nên sau khi mở hay tạo file phải lưu trữ lại để sử dụng trong việc đọc/ghi file. Mỗi file phải có một thẻ file, biến thẻ file phải là 2 bytes.

```
MOV Thef,AX      ;Lưu trữ thẻ file vào biến.
MOV BX,Thef      ;Đưa thẻ file vào BX để truy xuất file
```

Có thể mở hay tạo nhiều tập tin khác nhau trong cùng chương trình để xử lý. Khi đó, mỗi thẻ tập tin phải được lưu trữ trong mỗi biến khác nhau.

Hàm 3Eh	Đóng tập tin đang mở.
Vào	BX ← Thẻ tập tin
Ra	AX : Mã lỗi (nếu có)

Sau khi xử lý xong tập tin và trước khi thoát khỏi chương trình, tập tin phải được đóng lại để cập nhật thông tin hoặc nội dung mới của tập tin.

Hàm 3Fh	Đọc nội dung tập tin vào bộ nhớ
Vào	DS:DX ← Địa chỉ vùng nhớ chứa dữ liệu đọc được BX ← Thẻ tập tin cần đọc CX ← số byte cần đọc
Ra	AX : Số byte đọc được

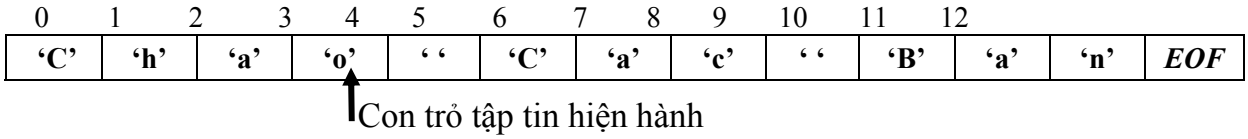
Hàm 40h	Ghi dữ liệu trong bộ nhớ vào tập tin
Vào	DS:DX ← Địa chỉ vùng nhớ chứa dữ liệu cần ghi BX ← Thẻ tập tin cần đọc CX ← số byte cần ghi
Ra	AX : Số byte ghi được

Các hàm 3Fh và 40h cũng có thể được sử dụng đọc hoặc ghi dữ liệu cho ngoại vi, vì DOS quản lý việc truy xuất ngoại vi bằng thẻ

Hàm 41h	Xóa tập tin trên đĩa
Vào	DS:DX ← Địa chỉ chuỗi tên tập tin
Ra	AX : Mã lỗi (nếu có)

Hàm 42h	Đời con trỏ tập tin hiện hành
Vào	AL ← Hướng dời CX:DX ← Cự ly dời = (CX*65536) + DX BX ← Thẻ tập tin
Ra	DX:AX : Vị trí con trỏ mới = (DX*65536)+AX

Cấu trúc dữ liệu trong tập tin:



Hàm 43h	Đọc hoặc thay đổi thuộc tính tập tin trên đĩa
Vào	DS:DX ← Địa chỉ chuỗi tên tập tin *Nếu Đọc thuộc tính:* AL ← 0 *Nếu Thay đổi thuộc tính:* AL ← 1 và CX ← thuộc tính mới
Ra	*Nếu Đọc thuộc tính:* CX : chứa thuộc tính tập tin

Hàm 4Ch	Kết thúc chương trình và trở về DOS
Vào	Không
Ra	Không

Hàm 56h	Đổi tên tập tin
Vào	DS:DX ← Địa chỉ chuỗi tên tập tin cũ ES:DI ← Địa chỉ chuỗi tên tập tin mới
Ra	Không

4.4 CÁC VÍ DỤ:

4.4.1. Chương trình nhận 1 ký tự từ bàn phím, sau đó cất ký tự nhận được vào 1 biến.

```
DSEG SEGMENT
       msg DB 'Hay nhap 1 ky tu tu ban phim: $'
       kytu DB ?  ; Biến chứa ký tự nhận được
DSEG ENDS
CSEG SEGMENT
       ASSUME CS: CSEG, DS: DSEG
start: MOV AX, DSEG
       MOV DS, AX
       MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
       LEA DX, msg     ; DX chứa địa chỉ chuỗicần in
       INT 21h         ; gọi ngắt thực hiện
       MOV AH, 01      ; Hàm nhập 1 ký tự, AL chứa Ký tự
       INT 21h
       MOV kytu, AL    ; Cất Ký tự vào biên
       MOV AH, 4Ch     ; thoát khỏi chương trình
       INT 21h
CSEG ENDS
     END start
```

4.4.2. Chương trình nhận 1 ký tự từ bàn phím, sau đó in ra màn hình ký tự kế sau của ký tự nhận được.

```
DSEG SEGMENT
       msg1 DB 'Hay nhap 1 ky tu tu ban phim: $'
       msg2 DB 10,13,'Ky tu ke sau ky tu nhan duoc la: $'
       kytu DB ?
DSEG ENDS
CSEG SEGMENT
       ASSUME CS: CSEG, DS: DSEG
start: MOV AX, DSEG
       MOV DS, AX
       MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
       LEA DX, msg1    ; DX chứa địa chỉ chuỗi msg1
       INT 21h         ; gọi ngắt thực hiện
       MOV AH, 01      ; Hàm nhập 1 ký tự, AL chứa Ký tự
       INT 21h
       MOV kytu, AL    ; cất ký tự vào biến
       MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
       LEA DX, msg2    ; DX chứa địa chỉ chuỗi msg2
       INT 21h         ; gọi ngắt thực hiện
```

```
        MOV DL, kytu
        ADD DL, 1       ; Ký tự kế sau
        MOV AH, 02      ; In ký tự
        INT 21h
        MOV AH, 4Ch     ; thoát khỏi chương trình
        INT 21h
CSEG ENDS
     END start
```

4.4.3. Chương trình nhận chuỗi A và B từ bàn phím, sau đó in ra màn hình B trước, chuỗi A sau.

```
DSEG SEGMENT
        msg1 DB 'Hay nhap vao chuoi A: $'
        msg2 DB 10,13, 'Hay nhap vao chuoi B: $'
        msg3 DB 10,13,'Cac chuoi nhan duoc la: $'
        max1 DB 30
        len1 DB ?
        strA DB 31 DUP('$')
        max2 DB 30
        len2 DB ?
        strB DB 31 DUP('$')
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV AX, DSEG
        MOV DS, AX
        MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
        LEA DX, msg1    ; DX chứa địa chỉ chuỗi msg1
        INT 21h         ; gọi ngắt thực hiện
        MOV AH, 0Ah     ; Nhập chuỗi A
        LEA DX, max1
        INT 21h
        MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
        LEA DX, msg2    ; DX chứa địa chỉ chuỗi msg2
        INT 21h         ; gọi ngắt thực hiện
        MOV AH, 0Ah     ; Nhập chuỗi B
        LEA DX, max2
        INT 21h
        MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
        LEA DX, msg3    ; DX chứa địa chỉ chuỗi msg3
        INT 21h         ; gọi ngắt thực hiện
        MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
```

```
        LEA DX, strB    ; DX chứa địa chỉ chuỗi B
        INT 21h         ; gọi ngắt thực hiện
        MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
        LEA DX, strA    ; DX chứa địa chỉ chuỗi A
        INT 21h         ; gọi ngắt thực hiện
        MOV AH, 4Ch     ; thoát khỏi chương trình
        INT 21h
CSEG ENDS
     END start
```

4.4.4. Chương trình ghi chuỗi "Welcome to Assembly!" vào tập tin mới có tên là W2A.TXT.

```
DSEG SEGMENT
        msg1 DB 'Da ghi xong file.$'
        msg2 DB 'Welcome to Assembly!'
        tenfile DB 'D:\W2A.TXT',0
        thefile DW ?
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV AX, DSEG
        MOV DS, AX
        MOV AH, 3Ch     ; Tạo file mới
        LEA DX, tenfile
        MOV CX, 0       ; thuộc tính bình thường
        INT 21h
        MOV thefile, BX ; cất thẻ file
        MOV AH, 40h     ; ghi file
        MOV BX, thefile
        MOV CX, 20      ; số byte cần ghi
        LEA DX, msg2    ; chuỗi cần ghi
        INT 21h
        MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
        LEA DX, msg1    ; DX chứa địa chỉ chuỗi msg1
        INT 21h         ; gọi ngắt thực hiện
        MOV AH, 3Eh     ; Đóng file
        MOV BX, thefile
        INT 21h
        MOV AH, 4Ch     ; thoát khỏi chương trình
        INT 21h
CSEG ENDS
     END start
```

4.4.5. Chương trình xóa tập tin trên đĩa. Tên file được nhập từ bàn phím khi chạy chương trình.

```
DSEG SEGMENT
       msg1 DB 'Ten file can xoa: $'
       msg2 DB 'Da xoa xong file.$'
       max  DB 30
       len  DB ?
       tenf DB 31 DUP(?)
DSEG ENDS
CSEG SEGMENT
       ASSUME CS: CSEG, DS: DSEG
start: MOV AX, DSEG
       MOV DS, AX
       MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
       LEA DX, msg1    ; DX chứa địa chỉ chuỗi msg1
       INT 21h
       MOV AH, 0Ah     ; Đọc tên file
       LEA DX, max
       INT 21h
       XOR CX, CX      ; đoạn chèn giá trị 0 vào cuối tên
       MOV CL, len     ;    tập tin
       LEA SI, tenf
       ADD SI, CX
       MOV [SI], CH    ; CH = 0
       MOV AH, 41h     ; Tạo file mới
       LEA DX, tenfile
       INT 21h
       MOV AH, 09      ; Hàm 9 in chuỗi ký tự ra màn hình
       LEA DX, msg1    ; DX chứa địa chỉ chuỗi msg1
       INT 21h         ; gọi ngắt thực hiện
       MOV AH, 4Ch     ; thoát khỏi chương trình
       INT 21h
CSEG ENDS
     END start
```

Chương 5
LỆNH NHẢY VÀ VÒNG LẶP

5.1. LỆNH NHẢY (CHUYỂN ĐIỀU KHIỂN)

Cách thực hiện các lệnh trong chương trình của bộ xử lý là tuần tự thực hiện các lệnh từ trên xuống, hết lệnh này đến lệnh khác. Lệnh nhảy là lệnh làm thay đổi thứ tự thực hiện lệnh của bộ xử lý. Đây cũng là cơ sở của việc xây dựng các cấu trúc rẽ nhánh trong hợp ngữ.

Có 2 loại lệnh nhảy: nhảy không có điều kiện và nhảy có điều kiện.

5.1.1. Lệnh nhảy không điều kiện

Cú pháp: JMP Đích : IP ← Đích

Đích: Immed16, Reg16, Mem16

Ý nghĩa: Luôn luôn nhảy đến thực hiện lệnh có địa chỉ là Đích. Phạm vi nhảy là trong đoạn (near jump).

Ví dụ: Xem đoạn chương trình sau

```
1.          MOV  AX, 1
2.          JMP  cong2 ; nhảy đến lệnh có nhãn cong2
3.    cong1:INC  AX
4.          JMP  setAX ; nhảy đến lệnh có nhãn setAX
5.    cong2:ADD  AX, 2
6.    setAX: ……
```

Trong đoạn chương trình trên, các lệnh luôn được thi hành là 1, 2, 5 và 6. Còn các lệnh 3 và 4 không bao giờ được thi hành.

Các dạng của lệnh JMP:

- JMP SHORT Đích ; (short jump). Nhảy trong phạm vi từ –128 đến +127 byte so với vị trí hiện tại.

  Ví dụ: JMP SHORT Calculate

- JMP FAR PTR Đích ; (far jump). Nhảy đến bất kì chỗ nào, có thể khác đoạn.

  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: có khai báo biến myPointer như sau:

```
myPointer   DW   Prepare, Calculate, Check, Output
```

  Xem đoạn lệnh:

```
MOV  BX, 2 ; chỉ số trong mảng con trỏ
SHL  BX, 1 ; nhân đôi
```

```
                JMP   myPointer[BX] ; nhảy đến Check
                ...
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 ; IP ← AX *(nhảy đến lệnh có địa chỉ trong AX)*

5.1.2. Lệnh nhảy có điều kiện:

Lệnh nhảy có điều kiện sẽ thực hiện việc nhảy đến nhãn hay không tùy thuộc vào điều kiện của lệnh nhảy. Lệnh nhảy có điều kiện chỉ nhảy trong phạm vi 128 byte. Muốn nhảy xa hơn, phải kết hợp với lệnh JMP.

Cú pháp tổng quát: J***xx*** Đích ; ***xx*** là các ký tự mô tả điều kiện

Đích: Immed, Reg, Mem

Ý nghĩa: Nếu điều kiện đúng thì nhảy tới thực hiện lệnh do toán hạng Đích, ngược lại thì tiếp tục thi hành lệnh ngay sau lệnh nhảy.

Các lệnh nhảy có điều kiện được tạo ra bằng cách ghép chữ J (JUMP) với các chữ cái mô tả điều kiện để nhảy. Các điều kiện này thường được thiết lập thông qua trạng thái thanh ghi cờ. Bảng 6.1 trình bày các lệnh nhảy thường dùng.

Ví dụ: Lệnh J**NE** **(J**ump if **N**ot **Z**ero), nhảy nếu không bằng, điều kiện này đúng khi ZF = 0 *(gọi là NZ)*.

Như vậy, giá trị các cờ trạng thái hiện tại là điều kiện để lệnh nhảy xem xét khi thi hành. Thông thường, lệnh so sánh (CMP) hay lệnh TEST được sử dụng để thiết lập điều kiện lệnh nhảy.

Lệnh so sánh:

Cú pháp: CMP Trái, Phải ; Cờ ← Trái – Phải

Nếu Trái > Phải ⇒ Trái - Phải > 0 : CF = 0 và ZF = 0
Nếu Trái < Phải ⇒ Trái - Phải < 0 : CF = 1 và ZF = 0
Nếu Trái = Phải ⇒ Trái - Phải = 0 : CF = 0 và ZF = 1

Trái, Phải: Immed, Reg, Mem

Bản chất của lệnh CMP là lệnh SUB (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 ← 1
MOV AL, 2  ; AL ← 2
CMP AH, AL       ; CF ← 1, ZF ← 0 vì AH < AL
```

Sau khi thực hiện các lệnh trên, cờ Carry bật (CF=1) nghĩa là AH < AL

Ví dụ: kiểm tra hai bit cuối cùng của AL

```
TEST AL, 3 ; 3h = 00000011b
```

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

Lệnh	**Ý Nghĩa**	**Điều Kiện**
JB JNAE JC	Nhảy nếu nhỏ hơn (Jump if Below) Nhảy nếu không lớn hơn hoặc bằng Nhảy nếu có nhớ	CF = 1
JAE JNB JNC	Nhảy nếu lớn hơn hoặc bằng (Jump if Above or Equal) Nhảy nếu không nhỏ hơn Nhảy nếu không có nhớ	CF = 0
JBE JNA	Nhảy nếu nhỏ hơn hoặc bằng (Jump if Below or Equal) Nhảy nếu không lớn hơn	CF = 1 hay ZF = 1
JA JNBE	Nhảy nếu lớn hơn (Jump if Above) Nhảy nếu không nhỏ hơn hoặc bằng	CF = 0 và ZF = 0
JE JZ	Nhảy nếu bằng (Jump if Equal) Nhảy nếu bằng (Jump if Zero)	ZF = 1
JNE JNZ	Nhảy nếu không bằng (Jump if Not Equal) Nhảy nếu không bằng (Jump if Not Zero)	ZF = 0
JL JNGE	Nhảy nếu nhỏ hơn Nhảy nếu không lớn hơn hoặc bằng	SF ≠ OF
JGE JNL	Nhảy nếu lớn hơn Nhảy nếu không nhỏ hơn	SF = OF
JLE JNG	Nhảy nếu nhỏ hơn hay bằng Nhảy nếu không lớn hơn	ZF=1 và SF ≠ OF
JG JNLE	Nhảy nếu lớn hơn Nhảy nếu không nhỏ hơn hay bằng	ZF=0 và SF = OF
JP JPE	Nhảy nếu có bít kiểm chẳn lẽ Nhảy nếu kiểm tra chẳn	PF = 1
JNP JPO	Nhảy nếu không có bít kiểm chẳn lẽ Nhảy nếu kiểm tra lẻ	PF = 0
JS JNS JO JNO	Nhảy nếu có dấu Nhảy nếu không có dấu Nhảy nếu có tràn Nhảy nếu không có tràn	SF = 1 SF = 0 OF = 1 OF = 0
JCXZ	Nhảy nếu CX = 0	CX = 0

Bảng 5.1: Các lệnh nhảy có điều kiện

Ví dụ :

```
MOV  AX, 1000h    ; AX=1000h
CMP   AX, 200h    ; so sánh AX với 200h
```

```
        JZ      NH      ; Nếu AX=200h thì nhảy đến nhãn NH
        MOV  CX, BX ; Nếu AX ≠ 200h thì thi hành các lệnh khác
NH:     ADD  AX, BX ; Vị trí nhảy đến được gắn nhãn NH
```

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 ; Giả sử AL = 128
        CMP AH, 1
        JGE Greater ; AH > 1 nhưng không nhảy ????
        . . .
Greater: …….
```

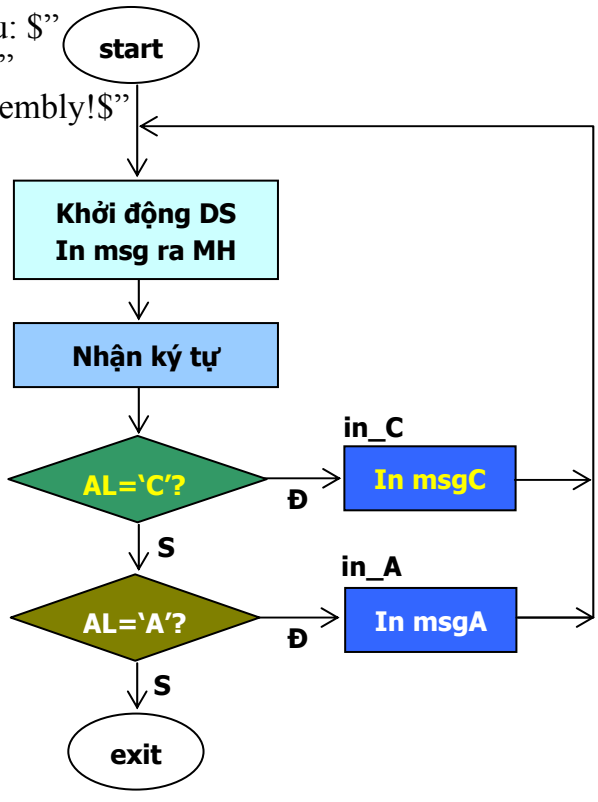
Ví dụ: nếu AL là số nguyên không dấu thì đoạn chương trình ở trên phải sửa lại như sau:

```
        MOV  AH,AL
        CMP  AH,1
        JAE  Greater
        . . . . . .
Greater: . . . . . .
```

Ví dụ 5-1: Viết chương trình nhận 1 ký tự từ bàn phím, sau đó:

- Nếu ký tự nhận được là 'C' thì in ra: "Welcome to C!"
- Nếu ký tự nhận được là 'A' thì in ra: "Welcome to Assembly!"
- Nếu là ký tự khác thì thoát khỏi chương trình.

```
DSEG SEGMENT
        msg     DB      10, 13, "Hay nhap 1 ky tu: $"
        msgC    DB      10, 13, "Welcome to C!$"
        msgA    DB      10, 13, "Welcome to Assembly!$"
DSEG ENDS
CSEG SEGMENT
        ASSUME      CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        MOV  AH, 09h
        LEA  DX, msg
        INT  21h
        MOV  AH, 01h
        INT  21h
        CMP  AL, 'C'
        JE   in_C
        CMP  AL, 'A'
        JE   in_A
        JMP  exit
in_C:   MOV  AH, 09h
        LEA  DX, msgC
```



Hình 6.1: Lưu đồ ví dụ 5-1

```
        INT   21h
        JMP   start
in_A:   MOV   AH, 09h
        LEA   DX, msgA
        INT   21h
        JMP   start
exit:   MOV   AH, 4Ch
        INT   21h
CSEG    ENDS
        END   start
```

5.2. VÒNG LẶP

Vòng lặp là một đoạn chương trình được thực hiện nhiều lần cho đến khi thỏa một điều kiện nào đó thì dừng lại, do đó vòng lặp thường kết thúc bằng một lệnh nhảy có điều kiện. Tuy nhiên, ngoài các lệnh nhảy đã biết, Intel-8086 còn cung cấp thêm các lệnh vòng lặp như LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ… các lệnh này đều có cú pháp giống nhau.

Cú pháp: LOOP Đích ; *Đích: Immed, Reg, Mem*

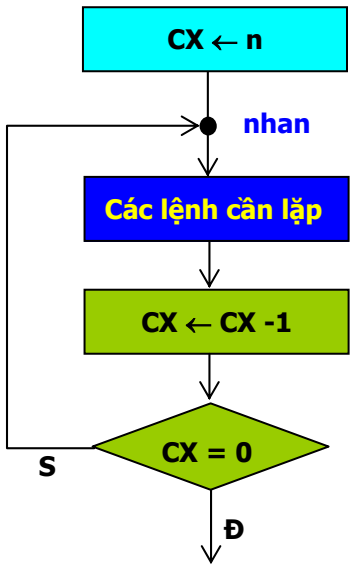
Ý nghĩa: tự động giảm CX một đơn vị, nếu CX ≠ 0 thì nhảy đến Đích, ngược lại nếu CX = 0 thì không nhảy đến Đích mà thực hiện lệnh ngay sau LOOP. Nói cách khác, vòng lặp LOOP dừng lại khi CX=0. Đây là vòng lặp *for* có số lần lập lưu trữ trong CX.

Cấu trúc vòng lặp for viết bằng LOOP:

```
        MOV CX, n  ; n là số lần lặp
nhan:   ……..      ; Các lệnh cần lặp lại
        ……...
        LOOP       nhan
```

Ví dụ: Đoạn lệnh in ra màn hình các ký tự từ A đến Z

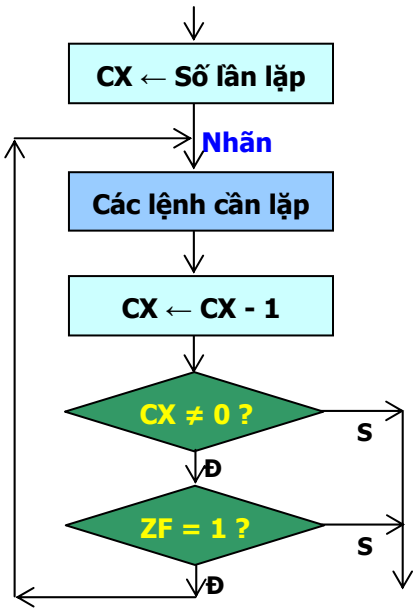
```
        MOV   DL, 'A' ; DL ← 'A'
        MOV   CX, 26  ; A→Z: 26 ký tự
inkytu: MOV   AH, 02h
        INT   21h
        INC   DL  ; DL tăng lên 1 để có ký tự kế
        LOOP  inkytu
```



Hình 6.2: Lưu đồ LOOP

LOOPZ/LOOPE:

Tự động giảm CX một đơn vị, nếu CX ≠ 0 và ZF = 1 thì nhảy đến Đích để thực hiện lệnh. Ngược lại, nếu CX = 0 hay ZF = 0 thì không nhảy, khi đó lệnh viết sau lệnh nhảy được thực hiện.



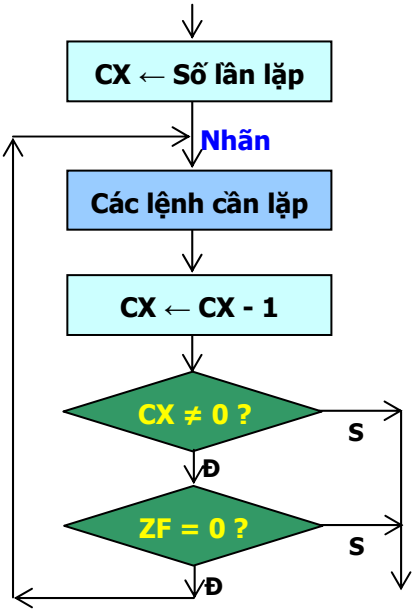
Hình 6.3: Lưu đồ LOOPE/LOOPZ

Cấu trúc vòng lặp do … while viết bằng LOOPZ:

```
        MOV CX, n
nhan:   <Các lệnh cần lập>
        LOOPZ nhan
```

LOOPNZ/LOOPNE:

Tự động giảm CX một đơn vị, nếu CX ≠ 0 và ZF = 0 thì nhảy đến Đích để thực hiện lệnh. Ngược lại, nếu CX = 0 hay ZF = 1 thì không nhảy, khi đó lệnh viết sau lệnh nhảy được thực hiện



Hình 6.4: Lưu đồ LOOPNE/LOOPNZ

Cấu trúc vòng lặp do … while viết bằng LOOPNZ:

```
        MOV CX, n
nhan:   <Các lệnh cần lập>
        LOOPNZ nhan
```

Lưu ý: Khi sử dụng các lệnh vòng lặp cần phải chú ý đến giá trị của CX.

- Nếu CX=0, vì LOOP giảm CX trước khi kiểm tra nên khi thực hiện lệnh LOOP thì CX = CX–1 = 0–1 = –1 = 0FFFFh. Như vậy LOOP sẽ thực hiện thêm 65535 lần nữa.
- Lệnh JCXZ (xem trong bảng 5.1) nhảy khi CX = 0 thường được dùng để kiểm tra giá trị CX trước khi thực hiện vòng lặp.

Ví dụ: Nhập mảng A gồm 10 ký tự, dừng lại nếu gặp phím Enter

```
        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
        CMP   AL, 0Dh
        LOOPNE  LAP
```

Ví dụ 5-2: Viết chương trình sử dụng hàm 01/21h để nhập chuỗi ký tự dài tối đa 128 ký tự hoặc kết thúc bằng phím Enter

```
DSEG SEGMENT
        chuoi  DB     128 DUP(?)
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        LEA   SI, chuoi       ; SI ← địa chỉ chuỗi
        MOV CX, 128           ; Chiều dài chuỗi tối đa (số vòng lặp)
key_in: MOV         AH, 01h       ; Hàm nhập 1 ký tự
        INT    21h
        MOV  [SI], AL         ; Cất ký tự vào biến chuoi
        INC    SI
        CMP   AL, 0Dh         ; Ký tự vừa nhập là Enter? nếu không phải Enter
        LOOPNE   key_in       ; hay chưa đủ 128 ký tự thì nhập tiếp
        MOV  AH, 4Ch
        INT    21h
CSEG ENDS
        END start
```

BÀI TẬP CHƯƠNG 5

5.1. Viết các lệnh để thực hiện các cấu trúc rẽ nhánh sau:

a.
```
IF (AX < 0) THEN BX = BX – 1
ENDIF
```

b.
```
IF (DL>=‘A’) and (DL<=‘Z’) THEN (In DL ra màn hình)
ENDIF
```

c.
```
IF (AX < BX) or (BX < CX) THEN DX = 0
ELSE     DX = 1
ENDIF
```

d.
```
IF (AX < BX)
If (BX < CX) THEN  AX = 0
Else       BX = 0
EndIf
ENDIF
```

5.2. Viết chương trình đọc 1 ký tự từ bàn phím.

- Nếu ký tự nhận được là 'A' thì chuyển con trỏ về đầu dòng.
- Nếu ký tự nhận được là 'B' thì chuyển con trỏ xuống dòng.
- Nếu nhận được ký tự khác thì thoát khỏi chương trình.

5.3. Viết đoạn lệnh để tính:

a. AX = 1 + 4 + 7 + …. + 148 + 151

b. AX = 100 +95 + 90 + …. + 10 + 5

5.4. Không sử dụng lệnh DIV, viết đoạn lệnh để thực hiện AX chia cho BX (*nếu BX ≠ 0*), phần thương chứa trong CX, số dư chứa trong AX. Giải thuật như sau:

```
CX = 0
WHILE (AX >= BX)
        CX = CX + 1
        AX = AX – BX
ENDWHILE
```

5.5. Không sử dụng lệnh MUL, viết đoạn lệnh để thực hiện AX nhân với BX (*nếu BX ≠ 0*), kết quả lưu trong CX. Giải thuật như sau:

```
CX = 0
DO
        CX = CX + AX
        BX = BX - 1
WHILE  BX > 0
```

5.6. Sử dụng hàm 08h, ngắt 21h để viết chương trình nhận 1 chuỗi đầy đủ 30 ký tự từ bàn phím.

- Nếu ký tự nhận được là HOA thì hiển thị ký tự đó lên màn hình.
- Nếu ký tự nhận được là thường thì hiển thị dấu '*' lên màn hình.

5.7. Viết chương trình nhập 1 chuỗi tối đa 256 ký tự từ bàn phím. Sau đó in đảo ngược chuỗi nhận được ra màn hình.

5.8. Viết chương trình nhận 1 chuỗi ký tự thường từ bàn phím. Sau đó đổi chuỗi nhận được thành chuỗi ký tự HOA và in ra màn hình.

5.9. Viết chương trình nhận 1 chuỗi ký tự từ bàn phím. Sau đó đếm số CHỮ có trong chuỗi nhận được và in ra màn hình số đếm được.

Chương 6
NGĂN XẾP VÀ CHƯƠNG TRÌNH CON

Ngăn xếp là vùng nhớ lưu trữ tạm thời dữ liệu cho chương trình hoặc lưu trữ địa chỉ trở về từ chương trình con (còn gọi là thủ tục).

6.1. NGĂN XẾP

6.1.1. Tổ chức và vận hành

Ngăn xếp là vùng nhớ đặc biệt trong bộ nhớ có cách truy xuất đặc biệt khác hẳn với việc truy xuất ngẫu nhiên các ô nhớ trong bộ nhớ. Hệ điều hành sẽ cấp phát vùng nhớ cho ngăn xếp là vùng nhớ có địa chỉ cao nhất trong bộ nhớ, mỗi chương trình khi thi hành trên máy tính sẽ có ngăn xếp riêng cho chương trình đó.

Việc truy xuất nội dung ngăn xếp theo cơ chế "Vào sau, ra trước" (Last In First Out – LIFO) nghĩa là dữ liệu nào được đưa vào sau cùng sẽ được lấy ra trước.

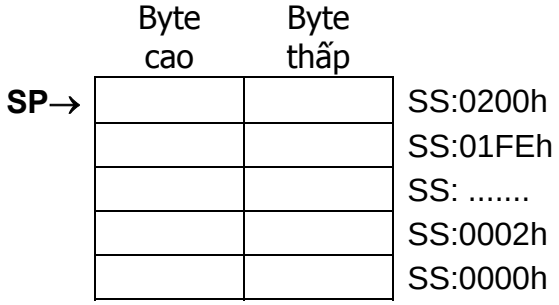
Ngăn xếp được tổ chức thành mãng nhiều phần tử, mỗi phần tử là 2 byte (word). Kích thước của ngăn xếp phụ thuộc vào chương trình và do người viết chương trình xác định bằng cách khai báo đoạn ngăn xếp.

Ví dụ: Khai bảo ngăn xếp 256 phần tử có tên là SSEG

```
SSEG SEGMENT   STACK   'STACK'
        DW     256 DUP(?)
SSEG SEGMENT
```

Địa chỉ logic của đỉnh ngăn xếp trong bộ nhớ được xác định bằng con trỏ ngăn xếp SS:SP (SS chứa địa chỉ đoạn ngăn xếp. SP chứa địa chỉ độ dời của đỉnh ngăn xếp). Đỉnh của ngăn xếp khi mới khởi tạo (gọi là đáy của ngăn xếp) luôn luôn là phần tử có địa chỉ cao nhất trong đoạn ngăn xếp.

Hình 6.1 mô tả vùng nhớ đoạn ngăn xếp trong ví dụ trên và giá trị của con trỏ ngăn xếp khi mới khởi tạo là phần tử cao nhất trong đoạn ngăn xếp (SP = 0200h).



Hình 6.1: Mô hình Đoạn Stack gồm 256 phần tử (256 word)

Khi cất dữ liệu vào ngăn xếp, SP giảm 2 trước khi lưu trữ dữ liệu vào. Khi lấy dữ liệu ra khỏi ngăn xếp, thì dữ liệu được đọc ra trước sau đó SP mới tăng lên 2 để chỉ phần tử kế tiếp trong ngăn xếp.

6.1.2. Truy xuất ngăn xếp

Cú pháp:

```
PUSH  Nguồn      ; SP ← SP – 2, Mem[SP] ← Nguồn
PUSHF            ; SP ← SP – 2, Mem[SP] ← Flag
POP   Đích       ; Đích ← Mem[SP], SP ← SP + 2
POPF             ; Flag ← Mem[SP], SP ← SP + 2
```

Nguồn, Đích: Reg16 hay Mem16.

Ý nghĩa: - Lệnh PUSH giảm con trỏ ngăn xếp (SP) xuống 2, sau đó lưu trữ toán hạng nguồn vào ngăn xếp.

- Lệnh PUSHF lưu giữ thanh ghi cờ vào ngăn xếp.

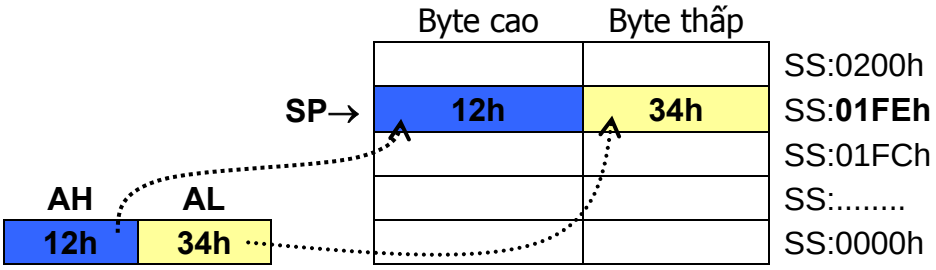
- Lệnh POP lấy 2 byte dữ liệu tại đỉnh ngăn đưa vào toán hạng đích, sau đó tăng SP lên 2.

- Lệnh POPF lấy 2 byte từ ngăn xếp đưa vào thanh ghi cờ.

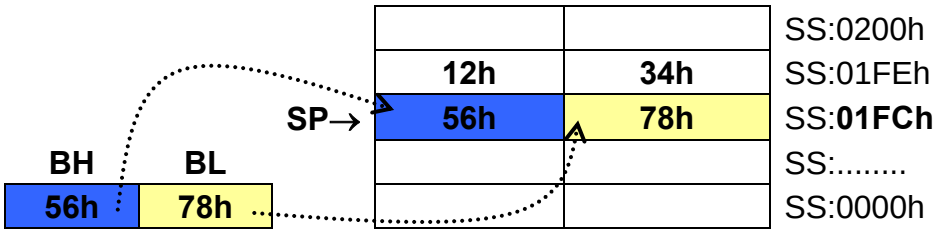
Ví dụ:

```
MOV  AX, 1234h
MOV  BX, 5678h
PUSH AX    ; SP ← SP – 2, Mem[SP] ← AX
PUSH BX    ; SP ← SP – 2, Mem[SP] ← BX
POP  DX    ; DX ← Mem[SP] ,  SP ← SP + 2
```

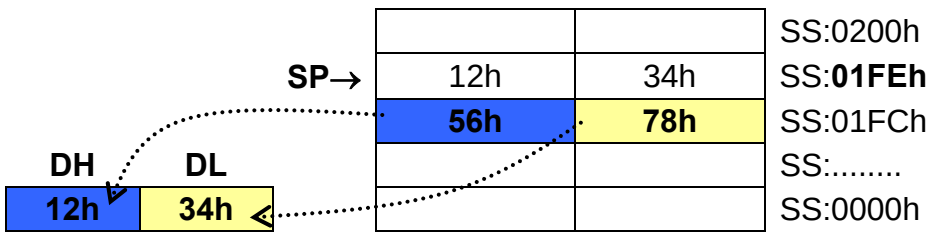
Hình 6.2 lần lượt mô tả hoạt động của ngăn xếp ứng với 3 lệnh truy xuất ngăn xếp trong ví dụ trên.



Hình 6.2a: PUSH AX (với AX=1234h)



Hình 6.2b: PUSH BX (với BX=5678h)



Hình 6.2c: POP DX (DX ← 5678h)

Ghi chú: Qua hai lệnh PUSH và POP, ta thấy ngăn xếp 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 ở địa chỉ cao và số liệu đưa vào ngăn xếp sau thì ở địa chỉ thấp hơn.

Ví dụ 6-1: Chương trình nhập 1 chuỗi từ bàn phím, sau đó in đảo ngược chuỗi nhận được ra màn hình sử dụng giải thuật của ngăn xếp *(Ký tự nhập vào sau cùng được in ra trước)*

```
DSEG SEGMENT
        msg1  DB    "Hay nhap chuoi ky tu, ket thuc bang Enter: $"
        msg2  DB    10, 13, "Chuoi dao nguoc la: $"
DSEG ENDS
SSEG SEGMENT  STACK  'STACK'
        DW    256    DUP(?)
SSEG ENDS
CSEG SEGMENT
        ASSUME    CS: CSEG, DS: DSEG, SS: SSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        MOV  AH, 09h
        LEA  DX, msg1
        INT  21h
        XOR  CX, CX
nhap:   MOV  AH, 01
        INT  21h
        CMP  AL, 0Dh        ; Có phải phím Enter không?
        JZ   inra           ; phải thì dừng, không phải thì nhập tiếp
        PUSH AX             ; Cất ký tự trong AL vào ngăn xếp
        INC  CX             ; đếm số ký tự nhập
        JMP  nhap           ; nhập tiếp ký tự
inra:   MOV  AH, 09h
        LEA  DX, msg2
        INT  21h
intiep: MOV  AH, 02
        POP  DX             ; Lấy ký tự trong ngăn xếp ra DL để in
        INT  21h
        LOOP intiep
        MOV  AH, 4Ch
        INT  21h
CSEG ENDS
        END start
```

6.2. CHƯƠNG TRÌNH CON

6.2.1. Khai báo chương trình con (Thủ tục)

Thủ tục là 1 đoạn chương trình có nhiệm vụ tương đối độc lập được sử dụng nhiều nơi trong chương trình chính. Thực chất, chương trình con hay thủ tục chỉ là 1 phần lệnh được viết riêng, giúp cho chương trình dễ đọc, linh hoạt và dễ bảo trì. Thủ tục thường được viết ở cuối chương trình, trong đoạn lệnh và có cấu trúc như sau:

TênThủTục **PROC** [kiểu]

….. ; Các lệnh trong thủ tục

RET ; chấm dứt thủ tục và trở về nơi gọi thủ tục.

TênThủTục **ENDP**

TênThủTục là một nhãn được người lập trình đặt theo qui cách đặt tên trong hợp ngữ. [kiểu] có thể là NEAR hay FAR dùng để xác định phạm vi của lệnh gọi thủ tục cùng hay khác đoạn với thủ tục. Nếu không khai báo rõ kiểu, thì mặc nhiên là NEAR.

RET (Return) là lệnh kết thúc thủ tục và trở về nơi gọi thủ tục để tiếp tục thi hành các lệnh sau lệnh gọi thủ tục trong chương trình. Lệnh RET sẽ lấy 2 byte địa chỉ trở về đang lưu trữ trong ngăn xếp để nạp vào thanh ghi IP (tương đương lệnh POP).

$$\text{RET} \Leftrightarrow \begin{cases} IP \leftarrow M[SS:SP] \\ SP \leftarrow SP + 2 \end{cases}$$

6.2.3. Gọi thủ tục

Cú pháp: CALL Đích;

$$\Leftrightarrow \begin{cases} SP \leftarrow SP - 2 \\ M[SS:SP] \leftarrow IP \\ IP \leftarrow \text{Đích} \end{cases}$$

Đích: tên thủ tục hay địa chỉ thủ tục

Khi thực hiện lệnh CALL, địa chỉ của lệnh ngay sau lệnh CALL (IP đang chứa địa chỉ này) được cất vào ngăn xếp, sau đó địa chỉ của thủ tục được nạp vào IP, do đó lệnh thi hành sau lệnh CALL sẽ là lệnh đầu tiên trong thủ tục được gọi. Như vậy, lệnh CALL cũng thực hiện thao tác nhảy gần giống như lệnh nhảy.

Thủ tục không có cơ chế truyền tham số vào trực tiếp trên dòng lệnh gọi thủ tục. Vì thể việc truyền tham số vào cho thủ tục phải thông qua các thanh ghi hay biến xác định trước khi gọi thủ tục. Dó đó, khi viết thủ tục, phải tự chọn thanh ghi hay biến làm tham số vào.

6.3. CÁC VÍ DỤ

6.3.1. Viết lại ví dụ 5-1, trong đó việc in chuỗi ra màn hình được thực hiện bằng thủ tục inchuoi.

```
DSEG SEGMENT
        msg    DB    "Hay nhap 1 ky tu: $"
        msgC   DB    "Welcome to C!$"
        msgA   DB    "Welcome to Assembly!$"
DSEG ENDS
CSEG SEGMENT
        ASSUME    CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        LEA  DX, msg
        CALL      inchuoi
        MOV  AH, 01h
```

```
        INT   21h
        CMP   AL, 'C'
        JE    in_C
        CMP   AL, 'A'
        JE    in_A
        JMP   exit
in_C:   LEA   DX, msgC
        CALL  inchuoi
        JMP   start
in_A:   LEA   DX, msgA
        CALL  inchuoi
        JMP   start
exit:   MOV   AH, 4Ch
        INT   21h
inchuoi         PROC
                PUSH DX
                MOV  AH, 02h      ; Xuống dòng trước khi in chuỗi
                MOV  DL, 13
                INT  21h
                MOV  DL, 10
                INT  21h
                POP  DX           ; DX chứa địa chỉ chuỗi cần in và
                MOV  AH, 09h      ;  chính là tham số vào của thủ tục.
                INT  21h
                RET
inchuoi         ENDP
CSEG ENDS
        END   start
```

6.3.2. Viết lại ví dụ 5-1, trong đó việc in chuỗi ra màn hình được thực hiện bằng macro inchuoi có tham số vào là biến chuỗi cần in. *(cú pháp MACRO được trình bày trong chương 2)*

```
inchuoi     MACRO    chuoi
            MOV  AH, 02h      ; Xuống dòng trước khi in chuỗi
            MOV  DL, 13
            INT  21h
            MOV  DL, 10
            INT  21h          ; DX chứa địa chỉ
            LEA  DX, chuoi    ; biến chuỗi cần in là tham số vào Macro
            MOV  AH, 09h
            INT  21h
            ENDM
```

```
DSEG SEGMENT
        msg    DB     "Hay nhap 1 ky tu: $"
        msgC   DB     "Welcome to C!$"
        msgA   DB     "Welcome to Assembly!$"
DSEG ENDS
CSEG SEGMENT
        ASSUME     CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        inchuoi    msg
        MOV  AH, 01h
        INT    21h
        CMP  AL, 'C'
        JE     in_C
        CMP  AL, 'A'
        JE     in_A
        JMP    exit
in_C:   inchuoi    msgC
        JMP    start
in_A:   inchuoi    msgA
        JMP    start
exit:   MOV  AH, 4Ch
        INT    21h
CSEG ENDS
        END    start
```

6.3.3. Viết lại ví dụ 6-1, trong đó việc nhập chuỗi ký tự và xuất đảo ngược chuỗi ký tự được viết bằng thủ tục. In chuỗi bằng Macro

```
inchuoi     MACRO      chuoi
            MOV  AH, 02h       ; Xuống dòng trước khi in chuỗi
            MOV  DL, 13
            INT    21h
            MOV  DL, 10
            INT    21h         ; DX chứa địa chỉ
            LEA    DX, chuoi   ; biến chuỗi cần in là tham số vào Macro
            MOV  AH, 09h
            INT    21h
            ENDM
DSEG SEGMENT
        msg1   DB     "Hay nhap chuoi ky tu, ket thuc bang Enter: $"
        msg2   DB     "Chuoi dao nguoc la: $"
        sokt   DW     ?
DSEG ENDS
SSEG SEGMENT   STACK   'STACK'
```

```
        DW    256    DUP(?)
SSEG ENDS
CSEG SEGMENT
        ASSUME     CS: CSEG, DS: DSEG, SS: SSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        inchuoi  msg1
        CALL nhapchuoi
        inchuoi          msg2
        CALL daochuoi
        MOV  AH, 4Ch
        INT    21h
nhapchuoi   PROC
                POP    BX
                XOR   CX, CX
        nhap:  MOV  AH, 01
                INT    21h
                CMP   AL, 0Dh ; Có phải phím Enter không?
                JZ      stop       ; phải thì dừng, không phải thì nhập tiếp
                PUSH AX       ; Cất ký tự trong AL vào ngăn xếp
                INC    CX       ; đếm số ký tự nhập
                JMP    nhap      ; nhập tiếp ký tự
        stop:  MOV  sokt, CX       ; cất số ký tự đã nhập
                PUSH BX
                RET
nhapchuoi   ENDP
daochuoi    PROC
                POP    BX
                MOV  CX, sokt
        intiep: MOV  AH, 02
                POP    DX     ; Lấy ký tự trong ngăn xếp ra DL để in
                INT    21h
                LOOP intiep
                PUSH BX
                RET
daochuoi    ENDP
CSEG ENDS
        END start
```

BÀI TẬP CHƯƠNG 6

6.1. Cho AX = 1234h, BX = 5678h, CX = 9ABCh và SP = 1000h. Hãy cho biết nội dung AX, BX, CX và SP sau khi thực hiện xong mỗi lệnh sau đây và vẽ mô hình ngăn xếp để minh họa quá trình thay đổi dữ liệu trong ngăn xếp.

```
PUSH   AX
PUSH   BX
XCHG  AX, CX
POP     CX
PUSH   AX
POP     BX
```

6.2. Vẽ mô hình ngăn xếp minh họa quá trình thay đổi dữ liệu trong ngăn xếp cho chương trình ví dụ 6-1.

6.3. Giả sử SP = 0200h và nội dung đỉnh ngăn xếp là 012Ah, Hãy cho biết trị của IP và SP sau khi thực hiện xong lệnh RET.

6.4. Vẽ mô hình ngăn xếp minh họa quá trình thay đổi dữ liệu trong ngăn xếp cho chương trình ví dụ 6.3.1

6.5. Với 2 lệnh sau đây, và giả định MOV nằm ở địa chỉ 08FD:0203h, PROC_1 là thủ tục NEAR tại địa chỉ 08FD:0300h, SP = 010AH. Hãy cho biết nội dung IP và SP sau mỗi lệnh.

```
CALL PROC_1
MOV  AX, BX
```

6.6. Viết chương trình nhập từ bàn phím một biểu thức đại số có chứa các dấu ngoặc tròn () hay []. Sau đó kiểm tra biểu thức nhận được là hợp lệ hay không hợp lệ và in kết quả ra màn hình.

Ví dụ: [a + (b – [c * (d – e)]) + f] là hợp lệ

[a + (b – [c * (d – e)) + f)] là không hợp lệ.

Hướng dẫn: dùng ngăn xếp để PUSH các dấu ngoặc trái '(', '[' vào ngăn xếp. Nếu gặp dấu ngoặc phải ')', ']' thì POP từ stack ra để so sánh. Nếu không POP được, hoặc POP ra không đúng loại với dấu ngoặc phải là không hợp lệ. Ngược lại là hợp lệ.

Chương 7
XỬ LÝ KÝ SỐ VÀ XỬ LÝ CHUỖI

7.1. XỬ LÝ KÝ TỰ

Như đã biết, việc xuất nhập trong hợp ngữ chỉ là xuất nhập 1 ký tự hay 1 chuỗi ký tự thông qua mã ASCII của nó và đó cũng chính là cơ chế hoạt động của bàn phím và màn hình. Do đó, khi cần nhập hay xuất các dạng số (nhị phân, thập phân, thập lục phân) thì phải xử lý các ký tự số (còn gọi là ký số) thành số sau khi nhập và xử lý số thành ký số để xuất ra màn hình.

7.1.1. Nhập xuất số nhị phân (Binary)

Số nhị phân nhập từ bàn phím là 1 chuỗi ký tự bao gồm các ký số '0' và '1'. Mỗi ký số đó được đưa vào máy tính ở dạng mã ASCII (8 bit), phải được xử lý lại thành dạng 1 bit. Như vậy:

- Ký số '0', mã ASCII là 30h, phải xử lý thành 1 bit 0.
- Ký số '1', mã ASCII là 31h, phải xử lý thành 1 bit 1.

Trước khi xuất số nhị phân ra màn hình thì phải xử lý ngược lại, nghĩa là lấy từng bit để đổi thành ký số tương ứng:

- Bit 0 phải xử lý thành mã ASCII là 30h (Ký số '0').
- Bit 1 phải xử lý thành mã ASCII là 30h (Ký số '1').

Ví dụ: Đoạn chương trình nhập 1 số nhị phân 8 bit từ bàn phím, lưu trữ trong thanh ghi BL. Sử dụng hàm 01/21h để nhập từng ký số.

```
        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:   ……
```

Ví dụ: Đoạn chương trình xuất số nhị phân 8 bit trong BL ra màn hình. Sử dụng hàm 02/21h để xuất từng ký số.

```
        MOV  CX, 8      ;  Xuất 8 bit
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
```

7.1.2. Nhập xuất số thập lục phân (Hexa)

Giải thuật nhập/xuất số thập lục phân cũng gần giống như số nhị phân. Cần lưu ý rằng:

- Các chữ số thập lục phân bao gồm: '0' … '9' và 'A' … 'F'. Khi đó 'A' chuyển thành 0Ah, …. 'F' chuyển thành 0Fh. Còn '0' đến '9' thì giống như trường hợp nhị phân.
- Nhưng mỗi số thập lục phân là 4 bit nhị phân.

Ví dụ: Đoạn chương trình nhập từ bàn phím số thập lục phân 16 bit (4 chữ số thập lục phân) vào thanh ghi BX. Sử dụng hàm 01/21h để nhập.

Giải thuật nhập:

```
        BX ← 0
lap:    Nhập ký tự
        Nếu ký tự là ký số thập lục phân:
                Đổi thành số tưng ứng
                Dịch trái BX 4 bit
                Đưa trị đã đổi vào 4 bit thấp của BX
        Nhảy về lap cho đến khi ký tự nhập là Enter
```

Đoạn chương trình thể hiện giải thuật trên:

```
        MOV  CL, 4
        XOR  BX, BX
nhap:   MOV  AH, 01
        INT  21h
        CMP  AL, 0Dh
        JZ   exit
        CMP  AL, 39h      ; Đổi ký số thành số tương ứng
        JA   kytu
        SUB  AL, 30h
        JMP  save
kytu:   SUB  AL, 37h
save:   SHL  BX, CL
        ADD  BL, AL
        JMP  nhap
exit:   …….
```

Ví dụ: Đoạn chương trình xuất giá trị BX ra màn hình ở dạng số thập lục phân (4 chữ số thập lục phân). Sử dụng hàm 02/21h để xuất.

Giải thuật xuất:

```
Lập 4 lần:  DL ← BH
            Dịch phải DL 4 bit
            Nếu DL < 10
                    Đổi thành ký số '0' … '9' tương ứng
            Nếu DL >= 10
```

Đổi thành ký tự ‘A’ …. ‘F’ tương ứng
In ra màn hình ký tự trong DL
Quay trái BX 4 bit

Đoạn chương trình thể hiện giải thuật trên:

```
        MOV  CX, 4
xuat:   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’ tương ứng
        JMP  inra
kytu:   ADD  DL, 37h     ; Đổi thành ký tự ‘A’ …. ‘F’ tương ứng
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 xuat
```

7.2. XỬ LÝ CHUỖI

7.2. LỆNH XỬ LÝ CHUỖI

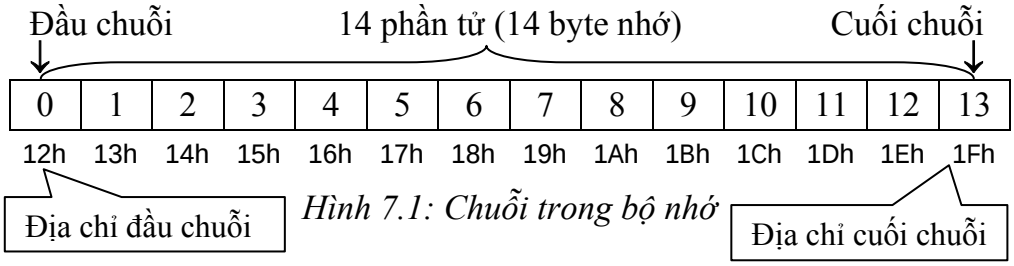
Khái niệm chuỗi trong máy tính không giới hạn ở chuỗi ký tự, mà là khái niệm mãng gồm nhiều phần tử, kiểu dữ liệu của phần tử là byte hay word. Các phần tử có thể chứa ký tự hay số liệu. Do đó, các lệnh thao tác trên chuỗi cho phép thao tác trên các mãng hay bất kỳ vùng đệm dữ liệu nào.

Chuỗi lưu trữ trong bộ nhớ có địa chỉ đầu và địa chỉ cuối chính là địa chỉ của phần tử đầu tiên và phần tử cuối trong chuỗi. Như vậy, thông số của 1 chuỗi trong bộ nhớ bao gồm: Địa chỉ đầu, địa chỉ cuối, số phần tử của chuỗi phải thỏa mãn công thức sau:

(Số byte của phần tử x số phần tử) = ĐC cuối - ĐC đầu + 1

Trong đó: (Số byte của phần tử x số phần tử) = Số byte của chuỗi

Hình 7.1 mô tả chuỗi gồm 14 phần tử, mỗi phần tử là 1 byte được lưu trữ trong bộ nhớ bắt đầu tại địa chỉ 12h.



Hình 7.1: Chuỗi trong bộ nhớ

Intel-8086 cung cấp nhiều lệnh xử lý chuỗi để thực hiện các thao tác như: chuyển chuỗi, so sánh chuỗi, dò tìm trong chuỗi …. Khi sử dụng những lệnh này thì việc viết chương trình sẽ ngắn hơn và thi hành nhanh hơn là sử dụng các lệnh MOV, CMP … trong các thao tác chuỗi. Các lệnh xử lý chuỗi gồm 3 nhóm trong bảng 7.1.

Các lệnh xử lý chuỗi không có toán hạng trên dòng lệnh, nên việc sử dụng các toán hạng mặc nhiên phải tuân thủ qui định của từng lệnh.

LỆNH	Ý NGHĨA
Nhóm di chuyển chuỗi	
MOVSB MOVSW	Di chuyển chuỗi từng byte (Move String Byte) Di chuyển chuỗi từng word (Move String Word)
LODSB LODSW	Nạp chuỗi từng byte (Load String Byte) Nạp chuỗi từng word (Load String Word)
STOSB STOSW	Ghi chuỗi từng byte (Store String Byte) Ghi chuỗi từng word (Store String Word)
Nhóm so sánh chuỗi	
CMPSB CMPSW	So sánh chuỗi từng byte (Compare String Byte) So sánh chuỗi từng word (Compare String Word)
Nhóm dò tìm giá trị trong chuỗi	
SCASB SCASW	Do tìm trong chuỗi từng byte (Scan String Byte) Do tìm trong chuỗi từng word (Scan String Word)

Bảng 7.1: Lệnh xử lý chuỗi

7.2.1. Hướng xử lý chuỗi

Khi xử lý 1 chuỗi có nghĩa là xử lý lần lượt các phần tử trong chuỗi, hết phần tử này đến phần tử khác cho đến khi hết chuỗi. Tùy vào đặc điểm của chuỗi, có hai hướng xử lý:

- Hướng tăng (từ trái qua phải) là xử lý phần tử đầu tiên trước và lần lượt đến phần tử cuối. Khi đó địa chỉ các phần tử sẽ *tăng* dần từ địa chỉ đầu cho đến địa chỉ cuối.
- Hướng giảm (từ phải qua trái) là xử lý phần tử cuối trước và lần lượt sau đó mới đến các phần tử đầu. Khi đó địa chỉ các phần tử sẽ *giảm* dần từ địa chỉ cuối cho đến địa chỉ đầu.

Thái thái cờ Hướng (DF) dùng để chọn hướng xử lý của chuỗi, do đó trước khi thực hiện lệnh xử lý chuỗi, phải chọn hướng xử lý chuỗi thích hợp bằng các lệnh thiết lập trạng thái DF như sau:

CLD ; DF = 0 : Hướng tăng

STD ; DF = 1 : Hướng giảm

7.2.2. Các tiền tố lập REP (Repeat)

Tiền tố REP có thể đặt trước các lệnh xử lý chuỗi như sau:

REP <Lệnh xử lý chuỗi>

Khi đó các lệnh xử lý chuỗi sẽ được lập lại với số lần lập xác định trong CX và sau mỗi lần lập, CX tự động giảm 1 cho đến khi CX = 0 thì kết thúc vòng lập *(Trong trường hợp này cũng có thể sử dụng lệnh vòng lập LOOP, nhưng lệnh sẽ dài dòng hơn)*.

Ví dụ: Đoạn lệnh thực hiện MOVSB lập lại 10 lần, dùng REP và LOOP như sau:

REP

```
MOV  CX, 10
REP  MOVSB
```

LOOP

```
     MOV  CX, 10
lap: MOVSB
     LOOP lap
```

Ngoài ra còn có các tiền tố lập tương tự như REP *(thường dùng cho các lệnh SCASB, SCASW, CMPSB, CMPSW)* như bảng 7.2 sau:

Lệnh	Ý Nghĩa
REPZ REPE	Lập lại lệnh theo sau nó nếu CX ≠ 0 và ZF = 1. Khi CX = 0 hay ZF = 0 sẽ không thực hiện vòng lập.
REPNZ REPNE	Lập lại lệnh theo sau nó nếu CX ≠ 0 và ZF = 0. Khi CX = 0 hay ZF = 1 sẽ không thực hiện vòng lập.

Bảng 7.2: Ý nghĩa các tiền tố lập

7.2.3. Lệnh Ghi vào chuỗi

STOSB sẽ ghi nội dung AL (1 byte) vào 1 phần tử trong chuỗi đích có địa chỉ xác định bởi ES:DI.

STOSW sẽ ghi nội dung AX (2 byte) vào 1 phần tử trong chuỗi đích có địa chỉ xác định bởi ES:DI.

STOSB	STOSW

Mem[ES:DI] ← AL
IF (DF = 0)
DI ← DI + 1
ELSE
DI ← DI – 1
	Mem[ES:DI] ← AX
	IF (DF = 0)
	DI ← DI + 2
	ELSE
	DI ← DI – 2

Ví dụ 7-1: Viết lại chương trình ở ví dụ 5-2, trong đó sử dụng lệnh STOSB để lưu các ký tự nhận được từ bàn phím vào biến chuỗi.

```
DSEG    SEGMENT
            chuoi   DB      128 DUP(?)
DSEG    ENDS
CSEG    SEGMENT
            ASSUME CS: CSEG, DS: DSEG, ES: DSEG
start:      MOV     AX, DSEG
            MOV     DS, AX
            MOV     ES, AX
            LEA     DI, chuoi       ; SI ← địa chỉ chuỗi
            MOV     CX, 128         ; Chiều dài chuỗi tối đa (số vòng lập)
key_in:     MOV     AH, 01h         ; Hàm nhập 1 ký tự
            INT     21h
            STOSB                   ; Cất ký tự vào biến chuoi
            CMP     AL, 0Dh         ; Ký tự vừa nhập là Enter? nếu không phải Enter
            LOOPNE  key_in          ; hay chưa đủ 128 ký tự thì nhập tiếp
            MOV     AH, 4Ch
            INT     21h
```

```
CSEG    ENDS
        END start
```

7.2.4. Lệnh Nạp từ chuỗi

LODSB sẽ nạp 1 byte trong chuỗi nguồn có địa chỉ xác định bởi DS:SI vào thanh ghi AL.

LODSW sẽ nạp 2 byte trong chuỗi nguồn có địa chỉ xác định bởi DS:SI vào thanh ghi AX.

LODSB	LODSW

AL ← Mem[DS:SI]
IF (DF = 0)
SI ← SI + 1
ELSE
SI ← SI – 1
	AX ← Mem[DS:SI]
	IF (DF = 0)
	SI ← SI + 2
	ELSE
	SI ← SI – 2

Ví dụ 7-2: Viết chương trình dùng hàm 02/21h để in chuỗi "Welcome to Assembly" ra màn hình, trong đó sử dụng lệnh LODSB để nạp từng ký tự trong chuỗi vào AL

```
DSEG SEGMENT
        chuoi  DB     "Welcome to Assembly"     ; chuỗi gồm 19 ký tự
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        LEA   SI, chuoi       ; SI ← địa chỉ chuỗi
        MOV CX, 19            ; Chiều dài chuỗi
key_in: LODSB                 ; nạp từng byte trong chuoi vào AL
        MOV  DL, AL
        MOV  AH, 02h          ; Hàm nhập in 1 ký tự
        INT    21h
        LOOP   key_out ;
        MOV  AH, 4Ch
        INT    21h
CSEG ENDS
        END start
```

7.2.5. Lệnh di chuyển chuỗi

MOVSB sẽ chuyển 1 byte trong chuỗi nguồn có địa chỉ xác định bởi DS:SI đến 1 phần tử trong chuỗi đích có địa chỉ xác định bởi ES:DI.

MOVSW sẽ chuyển 2 byte trong chuỗi nguồn có địa chỉ xác định bởi DS:SI đến 2 byte nhớ trong chuỗi đích có địa chỉ xác định bởi ES:DI.

MOVSB	**MOVSW**

Mem[ES:DI] ← Mem[DS:SI]
IF (DF = 0)
SI ← SI + 1
DI ← DI + 1
ELSE
SI ← SI – 1
DI ← DI – 1
	Mem[ES:DI] ← Mem[DS:SI]
	IF (DF = 0)
	SI ← SI + 2
	DI ← DI + 2
	ELSE
	SI ← SI – 2
	DI ← DI – 2

Ví dụ 7-3: Viết chương trình nhập 2 chuỗi từ bàn phím bằng hàm 0Ah, trong đó chỉ được khai báo 1 vùng đệm bàn phím.

(Hướng dẫn: Sau khi nhận được 1 chuỗi thì chuyển chuỗi nhận được sang biến chuỗi khác)

```
inchuoi     MACRO    chuoi
            MOV  AH, 09h
            LEA  DX, chuoi
            INT  21h
            ENDM
nhapchuoi   MACRO    vungdem
            MOV  AH, 0Ah      ; Nhập chuỗi bằng hàm 0Ah/21h
            LEA  DX, vungdem
            INT  21h
            ENDM
chuyenchuoi MACRO    dich     ; Chuyển chuỗi từ vùng đệm buff đến chuỗi dich
            XOR  CX, CX       ;
            MOV  CL, len      ; Số ký tự của chuỗi nhận được
            CLD               ; Chiều tăng
            LEA  SI, buff     ; địa chỉ chuỗi nguồn
            LEA  DI, dich     ; địa chỉ chuỗi đích
            REP  MOVSB
            ENDM
DSEG SEGMENT
        msg1  DB    "Hay nhap chuoi thu nhat: $"
        msg2  DB    "Hay nhap chuoi thu hai: $"
        xdong DB    10, 13, '$'    ; Ký tự xuống dòng
        chuoi1      DB    30    DUP (?)    ; nơi chứa chuỗi thứ nhất
        chuoi2      DB    30    DUP (?)    ; nơi chứa chuỗi thứ hai
        max   DB    30
        len   DB    ?
        buff  DB    31 DUP(?)
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
```

```
        MOV DS, AX
        MOV ES, AX        ; Vì chuỗi nguồn và chuỗi đích cùng đoạn DS
        inchuoi     msg1
        nhapchuoi   max   ; Nhập chuỗi thứ nhất
        chuyenchuoi chuoi1; Chuyển vào chuoi1
        inchuoi     xdong
        inchuoi     msg2
        nhapchuoi   max   ; Nhập chuỗi thứ hai
        chuyenchuoi chuoi2; Chuyển vào chuoi2
        MOV AH, 4Ch
        INT 21h
CSEG ENDS
        END start
```

7.2.6. Lệnh So sánh hai chuỗi

CMPVSB sẽ so sánh 1 byte trong chuỗi nguồn có địa chỉ xác định bởi DS:SI với 1 byte trong chuỗi đích có địa chỉ xác định bởi ES:DI.

CMPSW sẽ so sánh 2 byte trong chuỗi nguồn có địa chỉ xác định bởi DS:SI với 2 byte trong chuỗi đích có địa chỉ xác định bởi ES:DI.

CMPSB	**CMPSW**

Cờ ← Mem[DS:SI] – Mem[ES:DI]
IF (DF = 0)
SI ← SI + 1
DI ← DI + 1
ELSE
SI ← SI – 1
DI ← DI – 1
	Cờ ← Mem[DS:SI] – Mem[ES:DI]
	IF (DF = 0)
	SI ← SI + 2
	DI ← DI + 2
	ELSE
	SI ← SI – 2
	DI ← DI – 2

Ví dụ 7-4: Viết tiếp ví dụ 7-3, sau khi đã nhập 2 chuỗi thì so sánh 2 chuỗi đã nhập được. Nếu giống nhau thì in ra màn hình "Hai chuoi giong nhau". Ngược lại thì in ra màn hình "Hai chuoi khac nhau"

```
inchuoi     MACRO     chuoi
            MOV  AH, 09h
            LEA  DX, chuoi
            INT  21h
            ENDM
nhapchuoi   MACRO     vungdem
            MOV  AH, 0Ah      ; Nhập chuỗi
            LEA  DX, vungdem
            INT  21h
            ENDM
chuyenchuoi MACRO     dich    ; Chuyển chuỗi từ vùng đệm bàn
            XOR  CX, CX       ;                phím đến chuỗi dich
            MOV  CL, len                ; Số ký tự của chuỗi nhận được
```

```
                CLD                     ; Chiều tăng
                LEA   SI, buff          ; địa chỉ chuỗi nguồn
                LEA   DI, dich          ; địa chỉ chuỗi đích
                REP   MOVSB
                ENDM
DSEG SEGMENT
        msg1  DB    “Hay nhap chuoi thu nhat: $”
        msg2  DB    “Hay nhap chuoi thu hai: $”
        msg3  DB    “Sau khi so sanh, ket qua: $”
        msg4  DB    “Hai chuoi giong nhau$”
        msg5  DB    “Hai chuoi khac nhau$”
        msg
        xdong DB    10, 13, ‘$’     ; Ký tự xuống dòng
        chuoi1        DB    30      DUP (‘$’)     ; nơi chứa chuỗi thứ nhất
        chuoi2        DB    30      DUP (‘$’)     ; nơi chứa chuỗi thứ hai
        max   DB    30
        len   DB    ?
        buff  DB    31      DUP(‘$’)
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        MOV  ES, AX         ; Vì chuỗi nguồn và chuỗi đích cùng đoạn DS
        inchuoi       msg1
        nhapchuoi     max   ; Nhập chuỗi thứ nhất
        chuyenchuoi chuoi1; Chuyển vào chuoi1
        inchuoi       xdong
        inchuoi       msg2
        nhapchuoi     max   ; Nhập chuỗi thứ hai
        chuyenchuoi chuoi2; Chuyển vào chuoi2
        ; ----- So sánh 2 chuỗi: chuoi1 và chuoi2 -----
        MOV  CX, 30                 ; 30 ký tự
        LEA  SI, chuoi1     ; chuoi1 là chuỗi nguồn
        LEA  DI, chuoi2     ; chuoi2 là chuỗi đích
        REPZ CMPSB          ; nếu so sánh bằng thì tiếp tục so sánh tiếp
        PUSHF
        inchuoi       xdong
        inchuoi       msg3
        POPF
        JE    giong
        inchuoi       msg5
        JMP   exit
giong: inchuoi        msg4
exit:  MOV  AH, 4Ch
```

```
        INT   21h
CSEG ENDS
        END start
```

7.2.7. Lệnh dò tìm trong chuỗi

SCASB sẽ so sánh nội dung AL (1 byte) với 1 byte trong chuỗi đích có địa chỉ xác định bởi ES:DI.

SCASW sẽ so sánh nội dung AX (2 byte) với 2 byte trong chuỗi đích có địa chỉ xác định bởi ES:DI.

SCASB	SCASW

Cờ ← AL – Mem[ES:DI]
IF (DF = 0)
DI ← DI + 1
ELSE
DI ← DI – 1
	Cờ ← AX – Mem[ES:DI]
	IF (DF = 0)
	DI ← DI + 2
	ELSE
	DI ← DI – 2

Ví dụ: Viết chương trình nhập từ bàn phím 1 chuỗi, sau đó kiểm tra trong chuỗi nhận được có ký tự ‘A” không. In kết quả ra màn hình.

```
inchuoi     MACRO     chuoi
            MOV  AH, 09h
            LEA   DX, chuoi
            INT   21h
            ENDM
nhapchuoi   MACRO     vungdem
            MOV  AH, 0Ah      ; Nhập chuỗi
            LEA   DX, vungdem
            INT   21h
            ENDM
DSEG SEGMENT
        msg1  DB    “Hay nhap chuoi (toi da 50 ky tu): $”
        msg2  DB    “Dang tim ky tu A trong chuoi…….$”
        msg3  DB    “Da tim gap ky tu A.$”
        msg4  DB    “Khong tim gap ky tu A.$”
        xdong DB    10, 13, ‘$’     ; Ký tự xuống dòng
        max   DB    30
        len   DB    ?
        buff  DB    31      DUP(‘$’)
DSEG ENDS
CSEG SEGMENT
        ASSUME CS: CSEG, DS: DSEG
start:  MOV  AX, DSEG
        MOV  DS, AX
        MOV  ES, AX       ; Vì chuỗi nguồn và chuỗi đích cùng đoạn DS
        inchuoi     msg1
        nhapchuoi   max   ; Nhập chuỗi thứ nhất
```

```
        inchuoi     xdong
        inchuoi     msg2
        XOR  CX, CX
        MOV  CL, len            ; số ký tự của chuỗi
        MOV  AL, 'A'        ; ký tự cần tìm
        LEA  DI, buff       ; địa chỉ chuỗi cần tìm
        REPNE  SCASB        ; khi gặp thì dừng
        PUSHF               ; Cất trạng thái cờ sau khi thực hiện SCASB
        inchuoi     xdong
        POPF                ; Lấy lại trạng thái cờ đã cất
        JE     gapA
        inchuoi     msg4
        JMP    exit
gapA:   inchuoi     msg3
exit:   MOV  AH, 4Ch
        INT    21h
CSEG ENDS
        END start
```

BÀI TẬP CHƯƠNG 7

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

Ví dụ: Hãy nhập 1 ký tự: **K**

Mã ASCII dạng nhị phân là: **01110101**

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

Ví dụ: Hãy nhập 1 ký tự: **K**

Mã ASCII dạng thập lục phân là: **75**

7.3. Viết chương trình nhập từ bàn phím 1 số nhị phân 8 bit, sau đó in ra màn hình giá trị nhận được ở dạng số thập lục phân.

Ví dụ: Hãy nhập số nhị phân 8 bit: **10110101**

Dạng thập lục phân là: **B5**

7.4. Viết chương trình nhập 2 chuỗi A và B từ bàn phím, sau đó ghép chuỗi A với chuỗi B để tạo thành chuỗi C. In ra màn hình chuỗi C.

Ví dụ: Hãy nhập chuỗi A: *Chao cac ban*

Hãy nhập chuỗi B: *Sinh vien CNTT*

Chuỗi ghép C là: *Chao cac ban Sinh vien CNTT*

7.5. Viết chương trình nhập 1 chuỗi tối đa 256 ký tự. Sau đó tìm xem có các ký tự ‘A’, ‘a’, ‘B’, ‘b’ có trong chuỗi hay không. In ra màn hình các kết quả tìm được.

Ví dụ: Hãy nhập chuỗi: *Chao cac ban Sinh vien CNTT*

Kết quả:

- Không có ‘A’.
- Có ‘a’.
- Không có ‘B’.
- Có ‘b’.

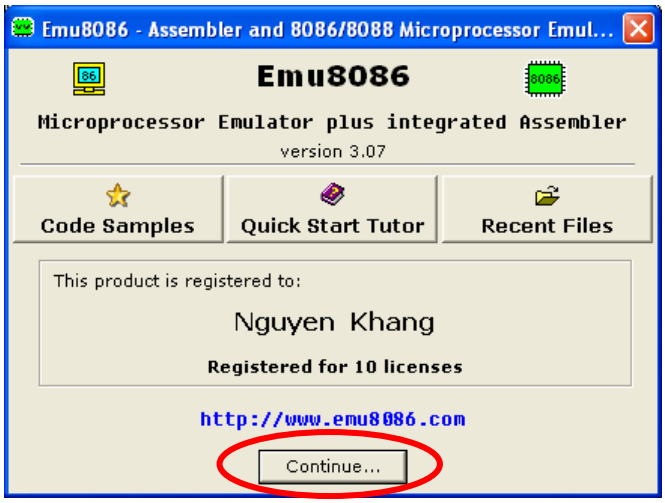
Phụ lục 1
Hướng Dẫn Sử Dụng Emu8086

Emu8086 là công cụ mạnh của người lập trình Hợp ngữ. Bao gồm nhiều chức năng như: thực thi chương trình dạng EXE (EXE Template), thực thi chương trình dạng COM (COM Template, Thực thi đoạn lệnh hợp ngữ (BIN Template), tạo đoạn Boot máy (BOOT Template). Trong phần hướng dẫn này chỉ hướng dẫn cho sinh viên cách sử dụng công cụ Emu8086 để khảo sát lệnh Intel-8086, BIN Template *(các chức năng khác, sinh viên tự tìm hiểu thêm thông qua phần Help của công cụ)*. Công cụ này giúp sinh viên thấy được sự thay đổi giá trị các thanh ghi, cờ, ngăn xếp, vùng nhớ … sau khi thực hiện lệnh. Qua đó sinh viên hiểu rõ hơn hoạt động của lệnh hay sự tác động lên giá trị thanh ghi/cờ/ngăn xếp/bộ nhớ khi thực hiện lệnh.

1. Khởi động Emu8086:

Nhấp đúp biểu tượng trên desktop, màn hình khởi động như hình 1 xuất hiện.

- **Code Samples** : Chọn file chương trình mẫu để thực hiện.
- **Quick Start Tutor**: Truy cập trang web hướng dẫn (phải có kết nối Internet).
- **Recent Files**: Chọn file trong danh sách file thường dùng.
- **Continue** …: Tiếp tục vào màn hình làm việc.



Hình 1: Màn hình khởi động Emu8086

Khi bấm **Continue**, màn hình làm việc xuất hiện như hình 2 với file chương trình mẫu "Hello World" mặc nhiên xuất hiện trong vùng soạn thảo.

Hình 3 là công cụ **Number Convertor** *(Bấm vào nút* **Convertor** *trên thanh công cụ)* rất hữu dụng khi muốn chuyển đổi giá trị giữa các hệ thống số với nhau.

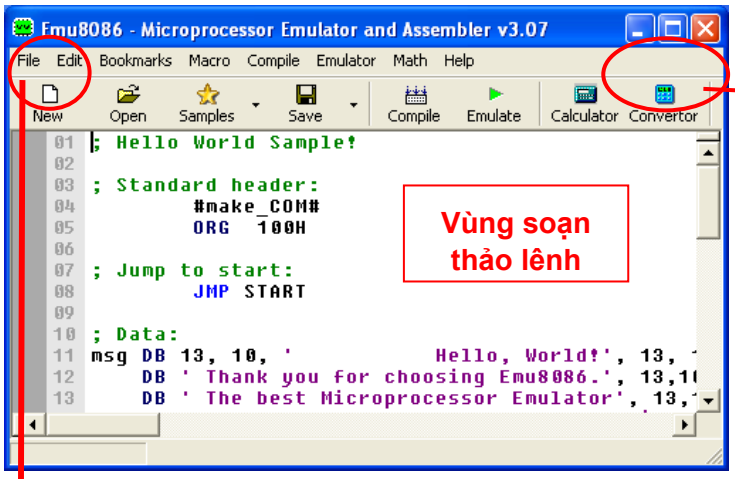
2. Soạn thảo lệnh hợp ngữ để khảo sát:

Để mở vùng làm việc mới chọn **NEW**, xuất hiện hình 4 để chọn Template

Để khảo sát lệnh Intel-8086 thì chọn chức năng thực thi lệnh (BIN Template). Vùng làm việc BIN Template xuất hiện như hình 5.

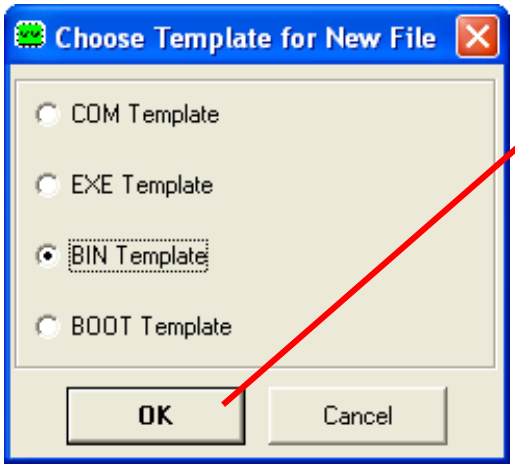
Trong BIN Template, quan trọng nhất là dòng đầu tiên **#make_bin#** dùng để xác định chế độ dịch lệnh của Emu8086. ***Tuyệt đối không được thay đổi dòng lệnh giả này.***

Các dòng còn lại dùng để khởi động các giá trị ban đầu cho các thanh ghi và thông số địa chỉ segment/offset cho chương trình. Các dòng này không quan trọng, có thể xóa bỏ được hoặc thay đổi giá trị khởi động khác. Khi các dòng này bị xóa bỏ thì các thông số và thanh ghi sẽ được khởi động theo giá trị mặc nhiên. Để đơn giản, nên xóa bỏ từ dòng 2 đến hết.

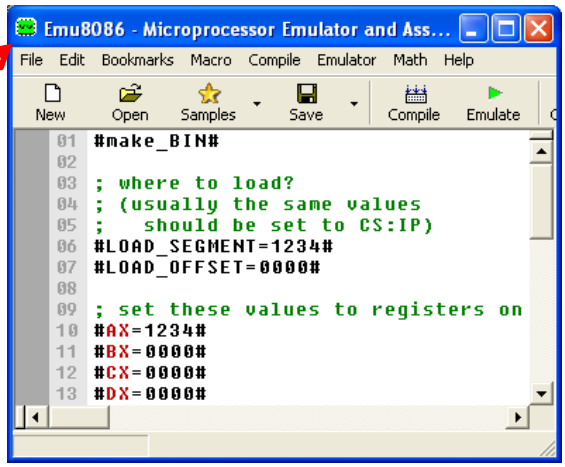


Hình 2: Màn hình Emu8086

Hình 3: Chức năng Convertor

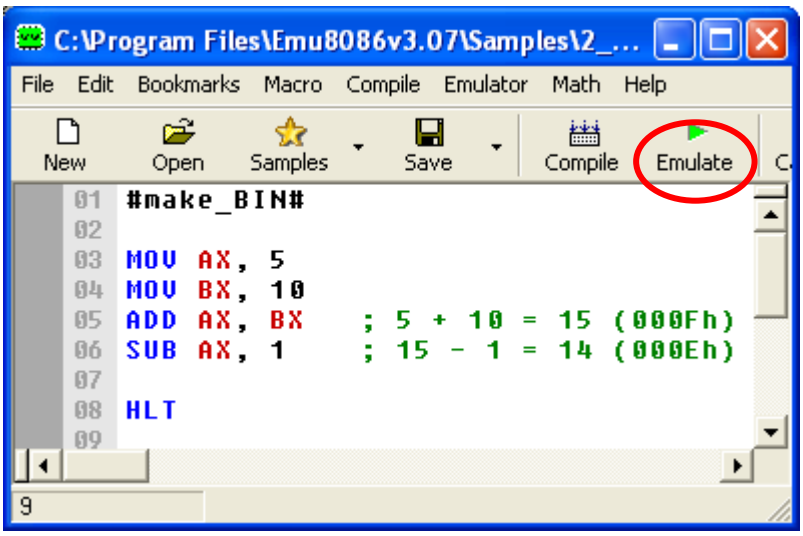


Hình 4: Chọn template



Hình 5: BIN Template

Ví dụ: Soạn đoạn lệnh như trong hình 6 sau để khảo sát:

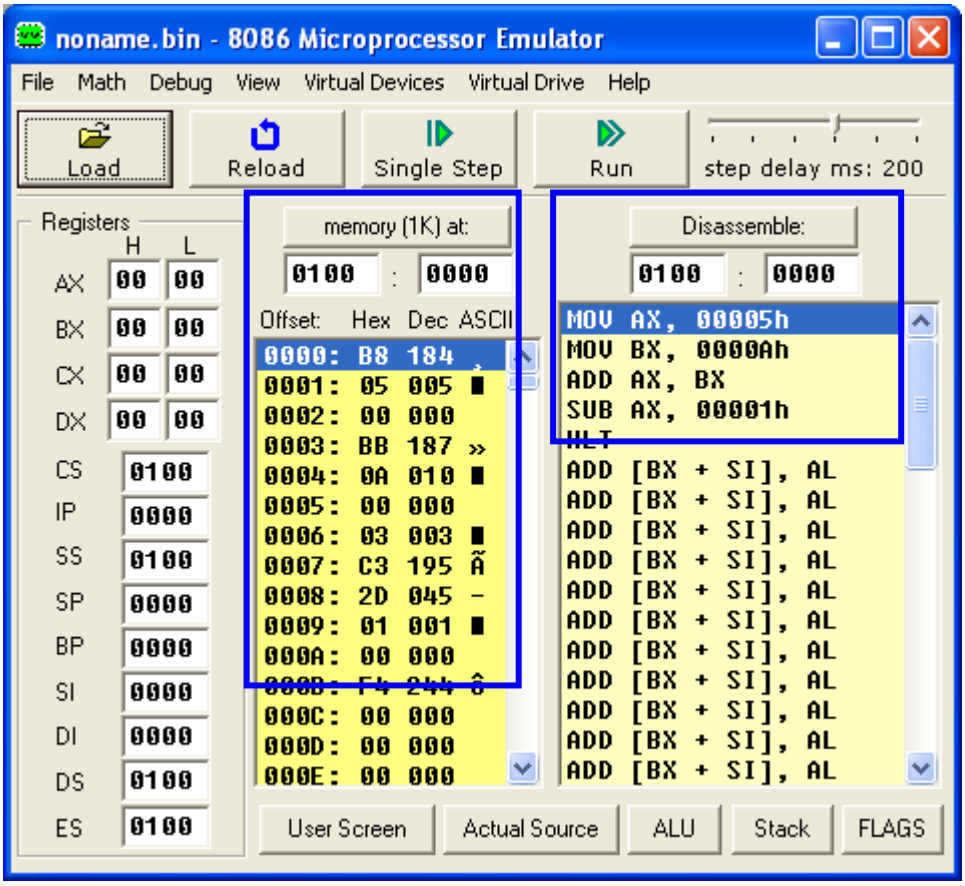


Hình 6: Ví dụ

3. Khảo sát lệnh (Giả lập - Emulate)

Để khảo sát lệnh bằng cách giả lập, chọn **Emulate**, khi đó màn hình giả lập xuất hiện như hình 7. Màn hình giả lập gồm 3 vùng: Thanh ghi (**Registers**), Bộ nhớ 1KB (**Memory**) chứa mã máy nhị phân và vùng hiển thị lệnh hợp ngữ tương ứng với mã máy nhị phân (**Disassemble**)

Giá trị các thanh ghi được trình bày ở dạng số Hex. Vùng bộ nhớ trình bày Hex – Dec – ASCII đối với từng ô nhớ (địa chỉ offset)

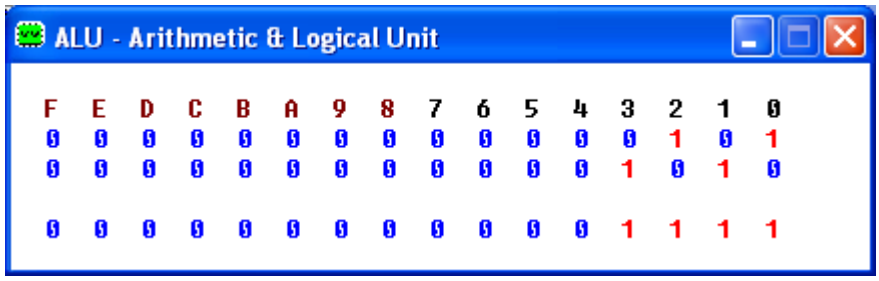


Hình 7: Màn hình giả lập

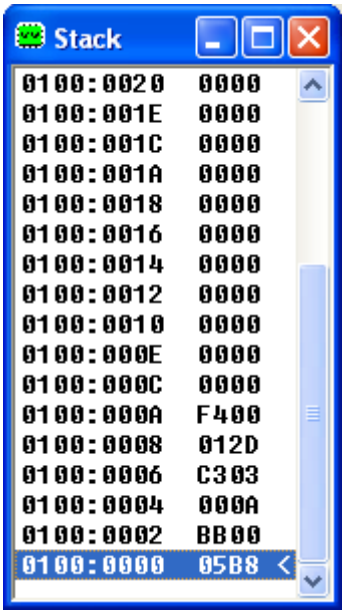
- **Reload**: Nạp lại đoạn lệnh
- **Run**: Chạy cả đoạn lệnh từ đầu cho đến khi gặp lệnh HLT (dừng)
- **Single Step:** Mỗi khi Single Step được bấm thì CPU chỉ chạy 1 lệnh hiện hành duy nhất (xác định bằng vệt sáng màu xanh) và dừng lại chờ cho đến khi Single Step được bấm tiếp. Như vậy, việc khảo sát lệnh có thể thực hiện thông qua Single Step.

Các thành phần khác còn có thể xem được trạng thái khi CPU thực hiện lệnh trong chế độ giả lập như ALU, Stack và FLAGS (thanh ghi Cờ) bằng cách bấm vào các nút tương ứng

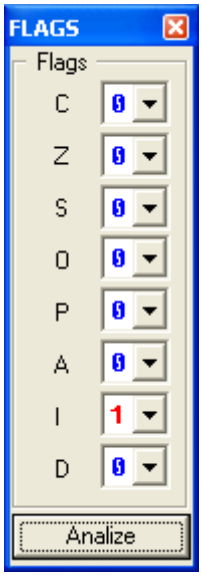
- Hình 8 cho biết trạng thái ALU khi thực hiện các phép toán (giá trị ở dạng nhị phân 16 bit). Dòng đầu tiên là thứ tự bit, dòng thứ 2 là giá trị toán hạng nguồn 1, dòng thứ 3 là giá trị toán hạng nguồn 2 và dòng cuối là giá trị kết quả sau khi thực hiện phép toán
- Hình 9 trình bày nội dung ngăn xếp ở dạng Hex 2 byte
- Hình 10 thể hiện trạng thái các cờ sau khi thực hiện phép toán



Hình 8: Trạng thái ALU



Hình 9: Stack



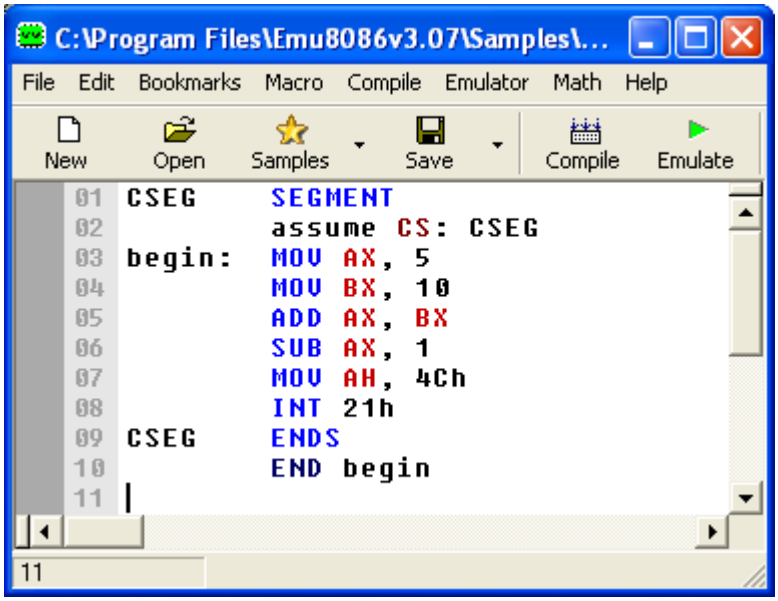
Hình 10: Thanh ghi Cờ

4. Thực thi chương trình dạng EXE hay COM

Emu8086 có thể thực thi chương trình Hợp ngữ viết theo cấu trúc dạng EXE hay COM.

Khi đó trong vùng soạn thảo, **không có dòng lệnh giả #make_bin#** và soạn thảo chương trình theo đúng cấu trúc dạng chương trình tương ứng.

Ví dụ: Chương trình dạng EXE như hình 11



Hình 11: Ví dụ chương trình dạng EXE

Phụ lục 2

Complete 8086 instruction set

http://www.emu8086.com/assembly_language_tutorial_assembler_reference/8086_instruction_set.html

Quick reference:

AAA	CMPSB	JAE	JNBE	JPO	MOV	RCR	SCASB
AAD	CMPSW	JB	JNC	JS	MOVSB	REP	SCASW
AAM	CWD	JBE	JNE	JZ	MOVSW	REPE	SHL
AAS	DAA	JC	JNG	LAHF	MUL	REPNE	SHR
ADC	DAS	JCXZ	JNGE	LDS	NEG	REPNZ	STC
ADD	DEC	JE	JNL	LEA	NOP	REPZ	STD
AND	DIV	JG	JNLE	LES	NOT	RET	STI
CALL	HLT	JGE	JNO	LODSB	OR	RETF	STOSB
CBW	IDIV	JL	JNP	LODSW	OUT	ROL	STOSW
CLC	IMUL	JLE	JNS	LOOP	POP	ROR	SUB
CLD	IN	JMP	JNZ	LOOPE	POPA	SAHF	TEST
CLI	INC	JNA	JO	LOOPNE	POPF	SAL	XCHG
CMC	INT	JNAE	JP	LOOPNZ	PUSH	SAR	XLATB
CMP	INTO	JNB	JPE	LOOPZ	PUSHA	SBB	XOR
	IRET				PUSHF		
	JA				RCL		

Operand types:

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
SREG: DS, ES, SS, and only as second operand: CS.
memory: [BX], [BX+SI+7], variable, etc....
immediate: 5, -24, 3Fh, 10001101b, etc...

Notes:

- When two operands are required for an instruction they are separated by comma. For example:
  REG, memory

- When there are two operands, both operands must have the same size (except shift and rotate instructions). For example:
  AL, DL
  DX, AX
  m1 DB ?
  AL, m1
  m2 DW ?
  AX, m2

- Some instructions allow several operand combinations. For example:
  memory, immediate
  REG, immediate

  memory, REG
  REG, SREG

These marks are used to show the state of the flags:

1 - instruction sets this flag to **1**.
0 - instruction sets this flag to **0**.
r - flag value depends on result of the instruction.
? - flag value is undefined (maybe **1** or **0**).

Instructions in alphabetical order:

Instruction	Operands	Description

AAA	No operands	ASCII Adjust after Addition.
		Corrects result in AH and AL after addition when working with BCD values.
		It works according to the following Algorithm:
		if low nibble of AL > 9 or AF = 1 then:
		AL = AL + 6
		AH = AH + 1
		AF = 1
		CF = 1
		else
		AF = 0
		CF = 0
		in both cases: clear the high nibble of AL.

AAD	Nooperands	ASCII Adjust before Division. Prepares two BCD values for division. Algorithm: AL = (AH * 10) + AL AH = 0 C Z S O P A ? r r ? r ?
AAM	No operands	ASCII Adjust after Multiplication. Corrects the result of multiplication of two BCD values. Algorithm: AH = AL / 10 AL = remainder C Z S O P A ? r r ? r ?

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>AAS</td><td>No operands</td><td>ASCII Adjust after Subtraction.
Corrects result in AH and AL after subtraction when working with BCD values.
Algorithm:
if low nibble of AL > 9 or AF = 1 then:
AL = AL - 6
AH = AH - 1
AF = 1
CF = 1
else
AF = 0
CF = 0
in both cases: clear the high nibble of AL.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>?</td><td>?</td><td>?</td><td>?</td><td>r</td></tr></table></td></tr>
<tr><td>ADC</td><td>REG,memory
memory, REG
REG, REG
memory,immediate
REG, immediate</td><td>Add with Carry.
Algorithm: operand1 = operand1 + operand2 + CF
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>ADD</td><td>REG, memory
memory, REG
REG, REG
memory,
immediate
REG, immediate</td><td>Add.
Algorithm: operand1 = operand1 + operand2
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>AND</td><td>REG, memory
memory, REG
REG, REG
memory,
immediate
REG, immediate</td><td>Logical AND between all bits of two operands. Result is stored in operand1.
These rules apply:
1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td></tr></table></td></tr>
<tr><td>CALL</td><td>procedure name
label
4-byte address</td><td>Transfers control to procedure, return address is (IP) is pushed to stack. 4-byte address may be entered in this form: 1234h:5678h, first value is a segment second value is an offset (this is a far call, so CS is also pushed to stack).</td></tr>
<tr><td>CBW</td><td>No operands</td><td>Convert byte into word.
Algorithm:
if high bit of AL = 1 then AH = 255 (0FFh)</td></tr>
</table>

<table>
<tr><td></td><td></td><td>else AH = 0</td></tr>
<tr><td>CLC</td><td>No operands</td><td>Clear Carry flag.
Algorithm: CF = 0</td></tr>
<tr><td>CLD</td><td>No operands</td><td>Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.
Algorithm: DF = 0</td></tr>
<tr><td>CLI</td><td>No operands</td><td>Clear Interrupt enable flag. This disables hardware interrupts
Algorithm: IF = 0</td></tr>
<tr><td>CMC</td><td>No operands</td><td>Complement Carry flag. Inverts value of CF.
Algorithm:
if CF = 1 then CF = 0
if CF = 0 then CF = 1</td></tr>
<tr><td>CMP</td><td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td><td>Compare.
Algorithm: operand1 - operand2
result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>CMPSB</td><td>No operands</td><td>Compare bytes: ES:[DI] from DS:[SI].
Algorithm: DS:[SI] - ES:[DI]
set flags according to result: OF, SF, ZF, AF, PF, CF
if DF = 0 then
SI = SI + 1
DI = DI + 1
else
SI = SI - 1
DI = DI - 1
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>R</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>CMPSW</td><td>No operands</td><td>Compare words: ES:[DI] from DS:[SI].
Algorithm: DS:[SI] - ES:[DI]
set flags according to result: OF, SF, ZF, AF, PF, CF
if DF = 0 then
SI = SI + 2
DI = DI + 2
else
SI = SI - 2
DI = DI - 2
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
</table>

<table>
<tr><td>CWD</td><td>No operands</td><td>Convert Word to Double word.
Algorithm:
if high bit of AX = 1 then: DX = 65535 (0FFFFh)
else DX = 0</td></tr>
<tr><td>DAA</td><td>No operands</td><td>Decimal adjust After Addition.
Corrects the result of addition of two packed BCD values.
Algorithm:
if low nibble of AL > 9 or AF = 1 then:
AL = AL + 6
AF = 1
if AL > 9Fh or CF = 1 then:
AL = AL + 60h
CF = 1
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>DAS</td><td>No operands</td><td>Decimal adjust After Subtraction.
Corrects the result of subtraction of two packed BCD values.
Algorithm:
if low nibble of AL > 9 or AF = 1 then:
AL = AL - 6
AF = 1
if AL > 9Fh or CF = 1 then:
AL = AL - 60h
CF = 1
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>DEC</td><td>REG
memory</td><td>Decrement.
Algorithm: operand = operand - 1
<table><tr><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table>CF - unchanged!</td></tr>
<tr><td>DIV</td><td>REG
memory</td><td>Unsigned divide.
Algorithm:
when operand is a byte:
AL = AX / operand
AH = remainder (modulus)
when operand is a word:
AX = (DX AX) / operand
DX = remainder (modulus)
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr></table></td></tr>
<tr><td>HLT</td><td>No operands</td><td>Halt the System.</td></tr>
</table>

<table>
<tr><td>IDIV</td><td>REG
memory</td><td>Signed divide.
Algorithm:

when operand is a byte:
AL = AX / operand
AH = remainder (modulus)

when operand is a word:
AX = (DX AX) / operand
DX = remainder (modulus)
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr></table></td></tr>
<tr><td>IMUL</td><td>REG
memory</td><td>Signed multiply.
Algorithm:
when operand is a byte: AX = AL * operand.
when operand is a word: (DX AX) = AX * operand.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>?</td><td>?</td><td>r</td><td>?</td><td>?</td></tr></table>CF=OF=0 when result fits into operand of IMUL.</td></tr>
<tr><td>IN</td><td>AL, im.byte
AL, DX
AX, im.byte
AX, DX</td><td>Input from port into AL or AX.
Second operand is a port number. If required to access port number over 255 - DX register should be used.</td></tr>
<tr><td>INC</td><td>REG
memory</td><td>Increment.
Algorithm: operand = operand + 1
<table><tr><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table>CF - unchanged!</td></tr>
<tr><td>INT</td><td>immediate byte</td><td>Interrupt numbered by immediate byte (0..255).
Algorithm:
Push to stack: flags register, CS , IP
IF = 0
Transfer control to interrupt procedure</td></tr>
<tr><td>INTO</td><td>No operands</td><td>Interrupt 4 if Overflow flag is 1.
Algorithm: if OF = 1 then INT 4</td></tr>
<tr><td>IRET</td><td>No operands</td><td>Interrupt Return.

Algorithm: Pop from stack: IP , CS , flags register
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">popped</td></tr></table></td></tr>
<tr><td>JA</td><td>label</td><td>Short Jump if first operand is Above second operand (as set by CMP instruction). Unsigned.

Algorithm: if (CF = 0) and (ZF = 0) then jump</td></tr>
<tr><td>JAE</td><td>label</td><td>Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned.

Algorithm: if CF = 0 then jump</td></tr>
</table>

JB	label	Short Jump if first operand is Below second operand (as set by CMP instruction). Unsigned. Algorithm: if CF = 1 then jump
JBE	label	Short Jump if first operand is Below or Equal to second operand (as set by CMP instruction). Unsigned. Algorithm: if CF = 1 or ZF = 1 then jump
JC	label	Short Jump if Carry flag is set to 1. Algorithm: if CF = 1 then jump
JCXZ	label	Short Jump if CX register is 0. Algorithm: if CX = 0 then jump
JE	label	Short Jump if first operand is Equal to second operand (as set by CMP instruction). Signed/Unsigned. Algorithm: if ZF = 1 then jump
JG	label	Short Jump if first operand is Greater then second operand (as set by CMP instruction). Signed. Algorithm: if (ZF = 0) and (SF = OF) then jump
JGE	label	Short Jump if first operand is Greater or Equal to second operand (as set by CMP instruction). Signed. Algorithm: if SF = OF then jump
JL	label	Short Jump if first operand is Less then second operand (as set by CMP instruction). Signed. Algorithm: if SF <> OF then jump
JLE	label	Short Jump if first operand is Less or Equal to second operand (as set by CMP instruction). Signed. Algorithm: if SF <> OF or ZF = 1 then jump
JMP	label 4-byte address	Unconditional Jump. Transfers control to another part of the program. *4-byte address* may be entered in this form: 1234h:5678h, first value is a segment second value is an offset Algorithm: always jump
JNA	label	Short Jump if first operand is Not Above second operand (as set by CMP instruction). Unsigned. Algorithm: if CF = 1 or ZF = 1 then jump
JNAE	label	Short Jump if first operand is Not Above and Not Equal to second operand (as set by CMP instruction). Unsigned. Algorithm: if CF = 1 then jump
JNB	label	Short Jump if first operand is Not Below second operand (as set by CMP instruction). Unsigned. Algorithm: if CF = 0 then jump
JNBE	label	Short Jump if first operand is Not Below and Not Equal to second operand (as set by CMP instruction). Unsigned. Algorithm: if (CF = 0) and (ZF = 0) then jump
JNC	label	Short Jump if Carry flag is set to 0. Algorithm: if CF = 0 then jump

JNE	label	Short Jump if first operand is Not Equal to second operand (as set by CMP instruction). Signed/Unsigned. Algorithm: if ZF = 0 then jump
JNG	label	Short Jump if first operand is Not Greater then second operand (as set by CMP instruction). Signed. Algorithm: if (ZF = 1) and (SF <> OF) then jump
JNGE	label	Short Jump if first operand is Not Greater and Not Equal to second operand (as set by CMP instruction). Signed. Algorithm: if SF <> OF then jump
JNL	label	Short Jump if first operand is Not Less then second operand (as set by CMP instruction). Signed. Algorithm: if SF = OF then jump
JNLE	label	Short Jump if first operand is Not Less and Not Equal to second operand (as set by CMP instruction). Signed. Algorithm: if (SF = OF) and (ZF = 0) then jump
JNO	label	Short Jump if Not Overflow. Algorithm: if OF = 0 then jump
JNP	label	Short Jump if No Parity (odd). Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if PF = 0 then jump
JNS	label	Short Jump if Not Signed (if positive). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if SF = 0 then jump
JNZ	label	Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if ZF = 0 then jump
JO	label	Short Jump if Overflow. Algorithm: if OF = 1 then jump
JP	label	Short Jump if Parity (even). Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if PF = 1 then jump
JPE	label	Short Jump if Parity Even. Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if PF = 1 then jump
JPO	label	Short Jump if Parity Odd. Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if PF = 0 then jump

JS	label	Short Jump if Signed (if negative). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if SF = 1 then jump
JZ	label	Short Jump if Zero (equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions. Algorithm: if ZF = 1 then jump
LAHF	No operands	Load AH from 8 low bits of Flags register. Algorithm: AH = flags register AH bit: 7 6 5 4 3 2 1 0 [SF] [ZF] [0] [AF] [0] [PF] [1] [CF] bits 1, 3, 5 are reserved.
LDS	REG, memory	Load memory double word into word register and DS. Algorithm: REG = first word DS = second word
LEA	REG, memory	Load Effective Address. Algorithm: REG = address of memory (offset)
LES	REG, memory	Load memory double word into word register and ES. Algorithm: REG = first word ES = second word
LODSB	No operands	Load byte at DS:[SI] into AL. Update SI. Algorithm: AL = DS:[SI] if DF = 0 then SI = SI + 1 else SI = SI - 1
LODSW	No operands	Load word at DS:[SI] into AX. Update SI. Algorithm: AX = DS:[SI] if DF = 0 then SI = SI + 2 else SI = SI - 2
LOOP	label	Decrease CX, jump to label if CX not zero. Algorithm: CX = CX - 1 if CX <> 0 then jump else no jump, continue
LOOPE	label	Decrease CX, jump to label if CX not zero and Equal (ZF=1) Algorithm: CX = CX - 1 if (CX <> 0) and (ZF = 1) then jump else no jump, continue

LOOPNE	label	Decrease CX, jump to label if CX not zero and Not Equal (ZF = 0). Algorithm: CX = CX - 1 if (CX <> 0) and (ZF = 0) then jump else no jump, continue
LOOPNZ	label	Decrease CX, jump to label if CX not zero and ZF = 0. Algorithm: CX = CX - 1 if (CX <> 0) and (ZF = 0) then jump else no jump, continue
LOOPZ	label	Decrease CX, jump to label if CX not zero and ZF = 1. Algorithm: CX = CX - 1 if (CX <> 0) and (ZF = 1) then jump else no jump, continue
MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate SREG, memory memory, SREG REG, SREG SREG, REG	Copy operand2 to operand1. The MOV instruction cannot: set the value of the CS and IP registers. copy value of one segment register to another segment register (should copy to general register first). copy immediate value to segment register (should copy to general register first). Algorithm: operand1 = operand2

MOVSB	No operands	Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.
		Algorithm:
		ES:[DI] = DS:[SI]
		if DF = 0 then
		SI = SI + 1
		DI = DI + 1
		else
		SI = SI - 1
		DI = DI - 1

MOVSW	No operands	Copy **word** at DS:[SI] to ES:[DI]. Update SI and DI.
		Algorithm:
		ES:[DI] = DS:[SI]
		if DF = 0 then
		SI = SI + 2
		DI = DI + 2
		else
		SI = SI - 2
		DI = DI - 2

<table>
<tr><td>MUL</td><td>REG
memory</td><td>Unsigned multiply.
Algorithm:
when operand is a byte: AX = AL * operand.
when operand is a word: (DX AX) = AX * operand.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>?</td><td>?</td><td>r</td><td>?</td><td>?</td></tr></table>CF=OF=0 when high section of the result is zero.</td></tr>
<tr><td>NEG</td><td>REG
memory</td><td>Negate. Makes operand negative (two's complement).
Algorithm:
Invert all bits of the operand
Add 1 to inverted operand
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>NOP</td><td>No operands</td><td>No Operation.
Algorithm: Do nothing</td></tr>
<tr><td>NOT</td><td>REG
memory</td><td>Invert each bit of the operand.
Algorithm:
if bit is 1 turn it to 0.
if bit is 0 turn it to 1.</td></tr>
<tr><td>OR</td><td>REG, memory
memory, REG
REG, REG
memory,
immediate
REG, immediate</td><td>Logical OR between all bits of two operands. Result is stored in first operand.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td><td>?</td></tr></table></td></tr>
<tr><td>OUT</td><td>im.byte, AL
im.byte, AX
DX, AL
DX, AX</td><td>Output from AL or AX to port.
First operand is a port number. If required to access port number over 255 - DX register should be used.</td></tr>
<tr><td>POP</td><td>REG
SREG
memory</td><td>Get 16 bit value from the stack.
Algorithm:
operand = SS:[SP] (top of the stack)
SP = SP + 2</td></tr>
<tr><td>POPA</td><td>No operands</td><td>Pop all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack.
SP value is ignored, it is Popped but not set to SP register).
Note: this instruction works only on 80186 CPU and later!
Algorithm:
POP DI
POP SI</td></tr>
</table>

<table>
<tr><td></td><td></td><td>POP BP
POP xx (SP value ignored)
POP BX
POP DX
POP CX
POP AX</td></tr>
<tr><td>POPF</td><td>No operands</td><td>Get flags register from the stack.
Algorithm:
flags = SS:[SP] (top of the stack)
SP = SP + 2
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">popped</td></tr></table></td></tr>
<tr><td>PUSH</td><td>REG
SREG
memory
immediate</td><td>Store 16 bit value in the stack.
Note: PUSH immediate works only on 80186 CPU and later!
Algorithm:
SP = SP - 2
SS:[SP] (top of the stack) = operand</td></tr>
<tr><td>PUSHA</td><td>No operands</td><td>Push all general purpose registers AX, CX, DX, BX, SP, BP, SI, DI in the stack.
Original value of SP register (before PUSHA) is used.
Note: this instruction works only on 80186 CPU and later!
Algorithm:
PUSH AX
PUSH CX
PUSH DX
PUSH BX
PUSH SP
PUSH BP
PUSH SI
PUSH DI</td></tr>
<tr><td>PUSHF</td><td>No operands</td><td>Store flags register in the stack.
Algorithm:
SP = SP - 2
SS:[SP] (top of the stack) = flags</td></tr>
<tr><td>RCL</td><td>memory, immediate
REG, immediate

memory, CL
REG, CL</td><td>Rotate operand1 left through Carry Flag. The number of rotates is set by operand2.
When immediate is greater then 1, assembler generates several RCL xx, 1 instructions because 8086 has machine code only for this instruction (the same principle works for all other shift/rotate instructions).
Algorithm:
shift all bits left, the bit that goes off is set to CF and previous value of CF is inserted to the right-most position.
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>OF=0 if first operand keeps original sign.</td></tr>
</table>

RCR	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 right through Carry Flag. The number of rotates is set by operand2. Algorithm: shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left-most position. C O r r *OF=0 if first operand keeps original sign.*

REP	chain instruction	Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.
		Algorithm:
		check_cx:
		if CX <> 0 then
		do following chain instruction
		CX = CX - 1
		go back to check_cx
		else
		exit from REP cycle
		Z
		r

REPE	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Equal), maximum CX times. Algorithm: check_cx: if CX <> 0 then do following chain instruction CX = CX - 1 if ZF = 1 then go back to check_cx else exit from REPE cycle else exit from REPE cycle Z r
REPNE	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times. Algorithm: check_cx: if CX <> 0 then do following chain instruction CX = CX - 1 if ZF = 0 then go back to check_cx else exit from REPNE cycle else exit from REPNE cycle Z r

REPNZ	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Zero), maximum CX times. Algorithm: check_cx if CX <> 0 then do following chain instruction CX = CX - 1 if ZF = 0 then go back to check_cx else exit from REPNZ cycle else exit from REPNZ cycle Z r
REPZ	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Zero), maximum CX times. Algorithm: check_cx if CX <> 0 then do following chain instruction CX = CX - 1 if ZF = 1 then go back to check_cx else exit from REPZ cycle else exit from REPZ cycle Z r
RET	No operands or even immediate	Return from near procedure. Algorithm: Pop from stack → IP if immediate operand is present: SP = SP + operand
RETF	No operands or even immediate	Return from Far procedure. Algorithm: Pop from stack → CS:IP if immediate operand is present: SP = SP + operand
ROL	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 left. The number of rotates is set by operand2. Algorithm: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position. C O r r *OF=0 if first operand keeps original sign.*
ROR	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 right. The number of rotates is set by operand2. Algorithm: shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position. C O r r *OF=0 if first operand keeps original sign.*

<table>
<tr><td>SAHF</td><td>No operands</td><td>Store AH register into low 8 bits of Flags register.
Algorithm: flags register = AH
AH bit: 7 6 5 4 3 2 1 0
[SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
bits 1, 3, 5 are reserved.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>SAL</td><td>memory, immediate
REG, immediate

memory, CL
REG, CL</td><td>Shift Arithmetic operand1 Left. The number of shifts is set by operand2.
Algorithm:
Shift all bits left, the bit that goes off is set to CF.
Zero bit is inserted to the right-most position.
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>OF=0 if first operand keeps original sign.</td></tr>
<tr><td>SAR</td><td>memory, immediate
REG, immediate

memory, CL
REG, CL</td><td>Shift Arithmetic operand1 Right. The number of shifts is set by operand2.
Algorithm:
Shift all bits right, the bit that goes off is set to CF.
The sign bit that is inserted to the left-most position has the same value as before shift.
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>OF=0 if first operand keeps original sign.</td></tr>
<tr><td>SBB</td><td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td><td>Subtract with Borrow.
Algorithm: operand1 = operand1 - operand2 - CF
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>SCASB</td><td>No operands</td><td>Compare bytes: AL from ES:[DI].
Algorithm: ES:[DI] - AL
set flags according to result: OF, SF, ZF, AF, PF, CF
if DF = 0 then DI = DI + 1
else DI = DI - 1
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
<tr><td>SCASW</td><td>No operands</td><td>Compare words: AX from ES:[DI].
Algorithm: ES:[DI] - AX
set flags according to result: OF, SF, ZF, AF, PF, CF
if DF = 0 then DI = DI + 2
else DI = DI - 2
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
</table>

<table>
<tr><td>SHL</td><td>memory, immediate
REG, immediate

memory, CL
REG, CL</td><td>Shift operand1 Left. The number of shifts is set by operand2.
Algorithm:
Shift all bits left, the bit that goes off is set to CF.
Zero bit is inserted to the right-most position.
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>OF=0 if first operand keeps original sign.</td></tr>
<tr><td>SHR</td><td>memory, immediate
REG, immediate

memory, CL
REG, CL</td><td>Shift operand1 Right. The number of shifts is set by operand2.
Algorithm: Shift all bits right, the bit that goes off is set to CF. Zero bit is inserted to the left-most position.
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>OF=0 if first operand keeps original sign.</td></tr>
<tr><td>STC</td><td>No operands</td><td>Set Carry flag.
Algorithm: CF = 1</td></tr>
<tr><td>STD</td><td>No operands</td><td>Set Direction flag. SI and DI will be decremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.
Algorithm: DF = 1</td></tr>
<tr><td>STI</td><td>No operands</td><td>Set Interrupt enable flag. This enables hardware interrupts.
Algorithm: IF = 1</td></tr>
<tr><td>STOSB</td><td>No operands</td><td>Store byte in AL into ES:[DI]. Update DI.
Algorithm:
ES:[DI] = AL
if DF = 0 then DI = DI + 1
else DI = DI - 1</td></tr>
<tr><td>STOSW</td><td>No operands</td><td>Store word in AX into ES:[DI]. Update DI.
Algorithm:
ES:[DI] = AX
if DF = 0 then DI = DI + 2
else DI = DI - 2</td></tr>
<tr><td>SUB</td><td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td><td>Subtract.
Algorithm: operand1 = operand1 - operand2
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td></tr>
</table>

<table>
<tr><td>TEST</td><td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td><td>Logical AND between all bits of two operands for flags only. These flags are effected: ZF, SF, PF. Result is not stored anywhere.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td></tr></table></td></tr>
<tr><td>XCHG</td><td>REG, memory
memory, REG
REG, REG</td><td>Exchange values of two operands.
Algorithm: operand1 < - > operand2</td></tr>
<tr><td>XLATB</td><td>No operands</td><td>Translate byte from table.
Copy value of memory byte at DS:[BX + unsigned AL] to AL register.
Algorithm: AL = DS:[BX + unsigned AL]</td></tr>
<tr><td>XOR</td><td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td><td>Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td><td>?</td></tr></table></td></tr>
</table>

Phụ Lục 3

BẢNG MÃ ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

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 RẼ NHÁNH – VÒNG LẬP 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 lập 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 nổ 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 ly 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 lập
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.
- Qui 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
```

a. Hợp dịch chương trình HELLO.ASM và kiểm tra xem file HELLO.OBJ đã được tạo ra chưa.

b. Liên kết chương trình HELLO, kiểm tra xem file HELLO.EXE đã được tạo ra chưa

c. Chạy chương trình HELLO.EXE, quan sát trên màn hình, chương trình trên làm gì?

d. 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*

a. Dịch chương trình HELLO2.ASM, ghi lại các thông tin về lỗi: Số lỗi, những lỗi gì, trên dòng nào?

b. Kiểm tra xem file HELLO2.OBJ được tạo ra không? Tại sao?

c. Sửa từng lỗi một từ dòng trên xuống, rồi lập lại câu a cho đến khi nào hết lỗi.

d. Liên kết chương trình HELLO2. Kiểm tra xem có file HELLO2.EXE không?

e. Chạy chương trình HELLO2.EXE, so sánh kết quả với **3.1.1.d**

```
DSEG SEGMEMT  ; 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 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
```

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.

a. Dịch sửa lỗi *(nếu có lỗi)* và chạy chương trình.

b. 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.

a. Gán giá trị các biến A = 1000, B = 10, C = 1Fh, D = 30h, E = 300Ah. Dịch và chạy chương trình.

b. Dùng Emu8086 để kiểm tra kết quả của câu a.

c. 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:

4.1. Dùng Emu8086 để khảo sát các lệnh khác trong tập lệnh của Intel-8086.

4.2. 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

b. 16 * 0AF1h

c. 300 * 400

d. 1000 ÷ 100

e. 1000 ÷ 100h

f. 3AB45Eh ÷ 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 nàm 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, sao 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 ← Trái – Phải

Nếu Trái > Phải ⇒ Trái - Phải > 0 : CF = 0 và ZF = 0
Nếu Trái < Phải ⇒ Trái - Phải < 0 : CF = 1 và ZF = 0
Nếu Trái = Phải ⇒ 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 ← 1
MOV AL, 2  ; AL ← 2
CMP AH, AL        ; CF ← 1, ZF ← 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 ← 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 <nhãn> ; (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 JNAE	Nhảy nếu nhỏ hơn (Jump if Below) Nhảy nếu không lớn hơn hoặc bằng	CF = 1
JAE JNB	Nhảy nếu lớn hơn hoặc bằng (Jump if Above or Equal) Nhảy nếu không nhỏ hơn	CF = 0
JBE JNA	Nhảy nếu nhỏ hơn hoặc bằng (Jump if Below or Equal) Nhảy nếu không lớn hơn	CF = 1 và ZF = 1
JA JNBE	Nhảy nếu lớn hơn (Jump if Above) Nhảy nếu không nhỏ hơn hoặc bằng	CF = 0 và ZF = 0
JE JZ	Nhảy nếu bằng (Jump if Equal) Nhảy nếu bằng (Jump if Zero)	ZF = 1
JNE JNZ	Nhảy nếu không bằng (Jump if Not Equal) Nhảy nếu không bằng (Jump if Not Zero)	ZF = 0

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 thuong"*. (Mã ASCII của ký tự HOA <= 'Z'). Soạn thảo và lưu với tên **BAI_3A.ASM**

```
DSEG SEGMENT
     tbao1 DB "Ky tu HOA.$"
     tbao2 DB "Ky tu thuong.$"
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 thuong"
     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 Y ....B 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 kí 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 phận:

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):

```
inchuoi  MACRO    chuoi
         MOV  AH, 9h
         LEA  DX, chuoi
         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
     inchuoi  msg1
     CALL bin_in
     MOV sobin, BL
     inchuoi  xdong
     inchuoi  msg2
     MOV  BL, sobin
     CALL bin_out
     MOV AH, 01
     INT 21h
     MOV AH, 4Ch        ; thoat 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        ; thoat khỏi chương trình
     INT 21h
hex_out  PROC
          MOV CX, 4
     xuat: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 xuat
          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?

```
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 Dec: $"
     xdong DB 10, 13, '$'
     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
     XOR  AX, AX
     MOV  AL, kytu ; Ký tự cần in
     CALL dec_out
     MOV AH, 01
     INT 21h
     MOV AH, 4Ch        ; thoat 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 đẻ 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 chuoi
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 chuoi 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 hiên 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 mà 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        ; mo tap tin da co
     lea dx, tenfile
     mov al, 2     ; thuoc tinh tap tin
     int 21h
     mov thefile, ax    ; cat the file
     mov ah, 3fh   ; doc noi dung file vao vung dem
     mov bx, thefile
     lea dx, buffer
     mov cx, 250        ; so byte can doc tu file da mo
     int 21h
     mov ah, 3eh        ; dong tap tin
     mov bx, thefile
     int 21h
     mov ah, 09h        ; in noi dung cua file ra man hinh
     lea dx, buffer
     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
```

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 thoat trong đoạn chương trình trên có nhiệm vụ gì ?. Thử bỏ lệnh jmp thoat 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 "Chuong tring so sanh oldpass va newpass$"
     oldpass   db "0123456789"
     newpass db "1234567890"
     tbao1 db "Haichuoi giong nhau $"
     tbao2 db "Haichuoi khong giong 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 da nhap dung roi" và thoát, ngược lại thì in ra câu thông báo "Ban da nhap sai roi va vui long nhap lai" 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 kiem 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.

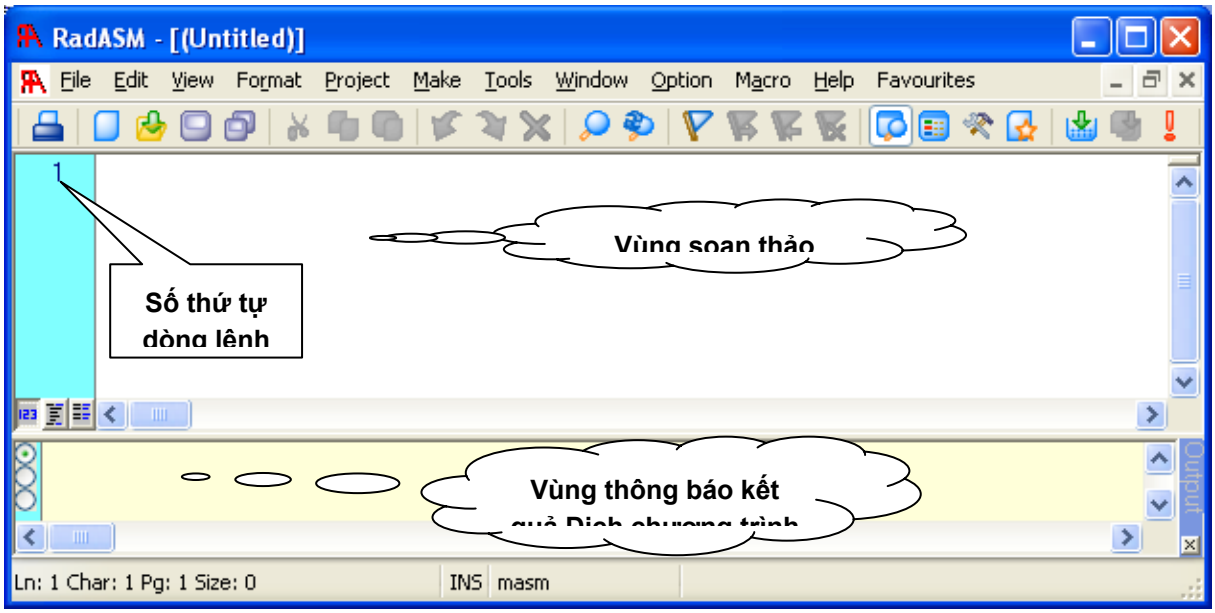
Phụ lục
Môi trường phát triển hợp ngữ RadASM

RadASM là môi trường phát triển Hợp ngữ, được xây dựng để kết hợp những hợp ngữ khác nhau nhau MASM, TASM, HASM …. và chạy trên môi trường Windows. Mỗi hợp ngữ khác nhau khi kếp hợp vào RadASM phải được cấu hình khác nhau. Vì mục đích cung cấp công cụ cho sinh viên sử dụng đơn gian, 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.

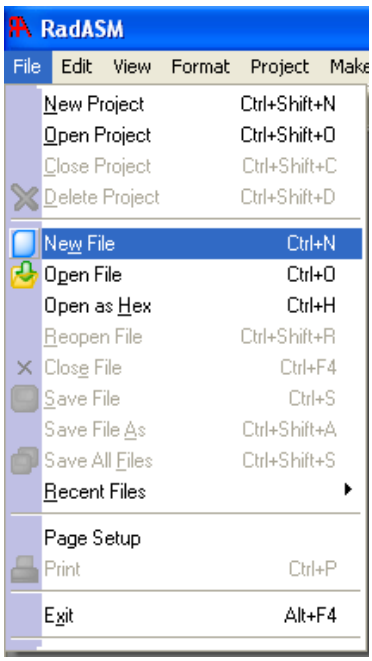
1. Khởi động RadASM

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:

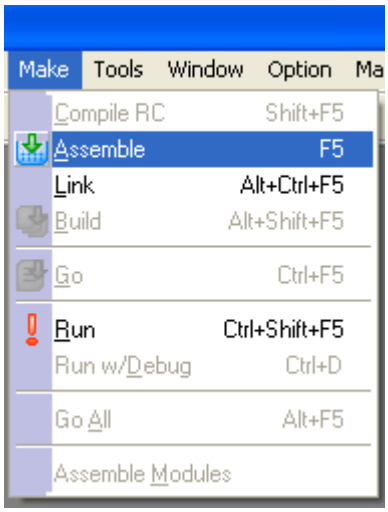
- Vùng soạn thảo dùng để soạn thảo chương trình nguồn ASM
- Vùng thông báo kết quả: Khi dịch chương trình, những kết quả hay lỗi sẽ xuất hiện tại vùng này



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.

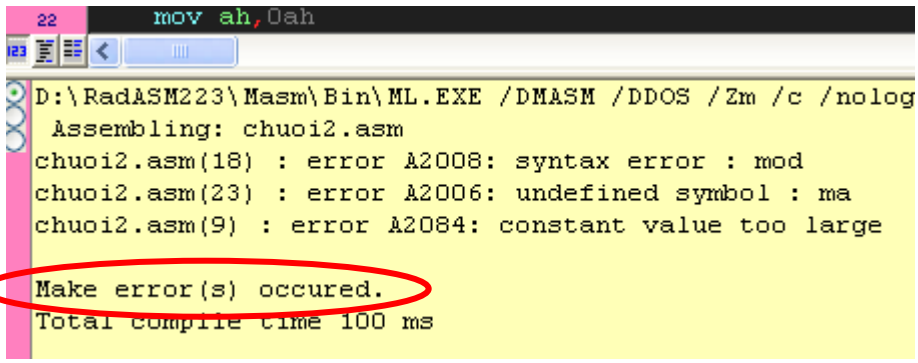
3. Hợp dịch (Assemble)

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

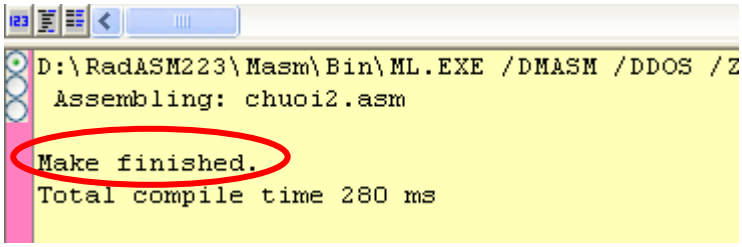
Ví dụ: **chuoi2.asm(18) : error A2008: syntax error : mod**

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



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

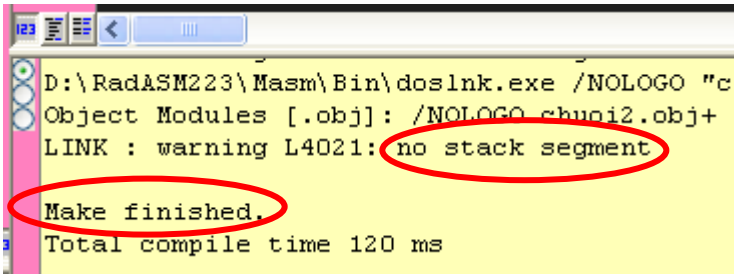


Hình 5: Thông báo kết quả Hợp dịch thành công

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.



Hình 6: Liên kết thành công

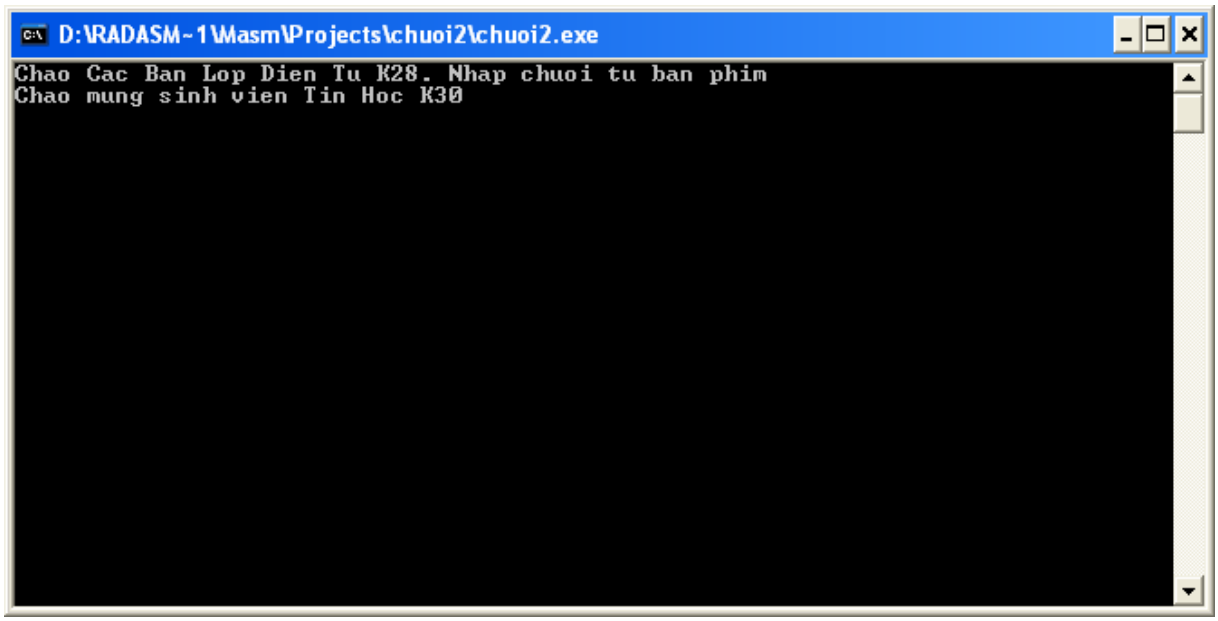
5. Thực thi chương trình EXE

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

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

LẬP TRÌNH HỆ THỐNG VÀ ĐIỀU KHIỂN THIẾT BỊ

(Dùng cho sinh viên hệ đào tạo đại học từ xa)

Lưu hành nội bộ

HÀ NỘI - 2006

LẬP TRÌNH HỆ THỐNG VÀ ĐIỀU KHIỂN THIẾT BỊ

Biên soạn : THS. PHẠM VĂN CƯỜNG

LỜI NÓI ĐẦU

Cuốn giáo trình Lập trình hợp ngữ và điều khiển thiết bị được chia thành 4 chương. Mỗi chương bao gồm các nội dung cơ bản, tóm tắt chương, các câu hỏi và bài tập cho mỗi chương.

Chương 1: trình bày về vấn đề liên quan đến bộ vi xử lý 8088 : kiến trúc, chức năng các thành phần và tập lệnh. Ngoài ra, 1 trong các ngắt được sử dụng phổ biến trong lập trình hệ thống- ngắt 21h của hệ điều hành DOS cũng được giới thiệu trong chương này.

Chương 2: trình bày về các vấn đề liên quan đến lập trình hợp ngữ: cách thức viết và thực hiện một chương trình, cách thức cài đặt các cấu trúc lập trình trong hợp ngữ và các vấn đề liên quan đến chương trình con và macro.

Chương 3: giới thiệu về công cụ gỡ rối debug, chương trình mô phỏng Emu 8086. Liên kết chương trình viết bằng hợp ngữ với chương trình được viết bằng các ngôn ngữ bậc cao như C và Pascal cũng được đề cập ở chương này. Ngoài ra, chương này cò giới thiệu về một số ngắt của BIOS phục vụ thiết bị ngoại vi, chương trình thường trú và chương trình con ngắt.

Chương 4: Trình bày về lập trình phối ghép: lập trình modem, bàn phím và màn hình. Đồng thời chương này cũng giới thiệu về một môi trường RadASM để phát triển các ứng dụng viết bằng hợp ngữ trên Windows.

Do thời gian có hạn và kinh nghiệm còn hạn chế, cuốn giáo trình sẽ không tránh khỏi các sai sót. Tác giả biên soạn rất mong nhận được ý kiến đóng góp từ các độc giả.

Mọi ý kiến góp ý xin gửi về email : pcuongcntt@yahoo.com

Xin chân thành cảm ơn!

Hà Nội, tháng 11/2006

Tác giả

CHƯƠNG TRÌNH ĐÀO TẠO ĐẠI HỌC TỪ XA PTIT

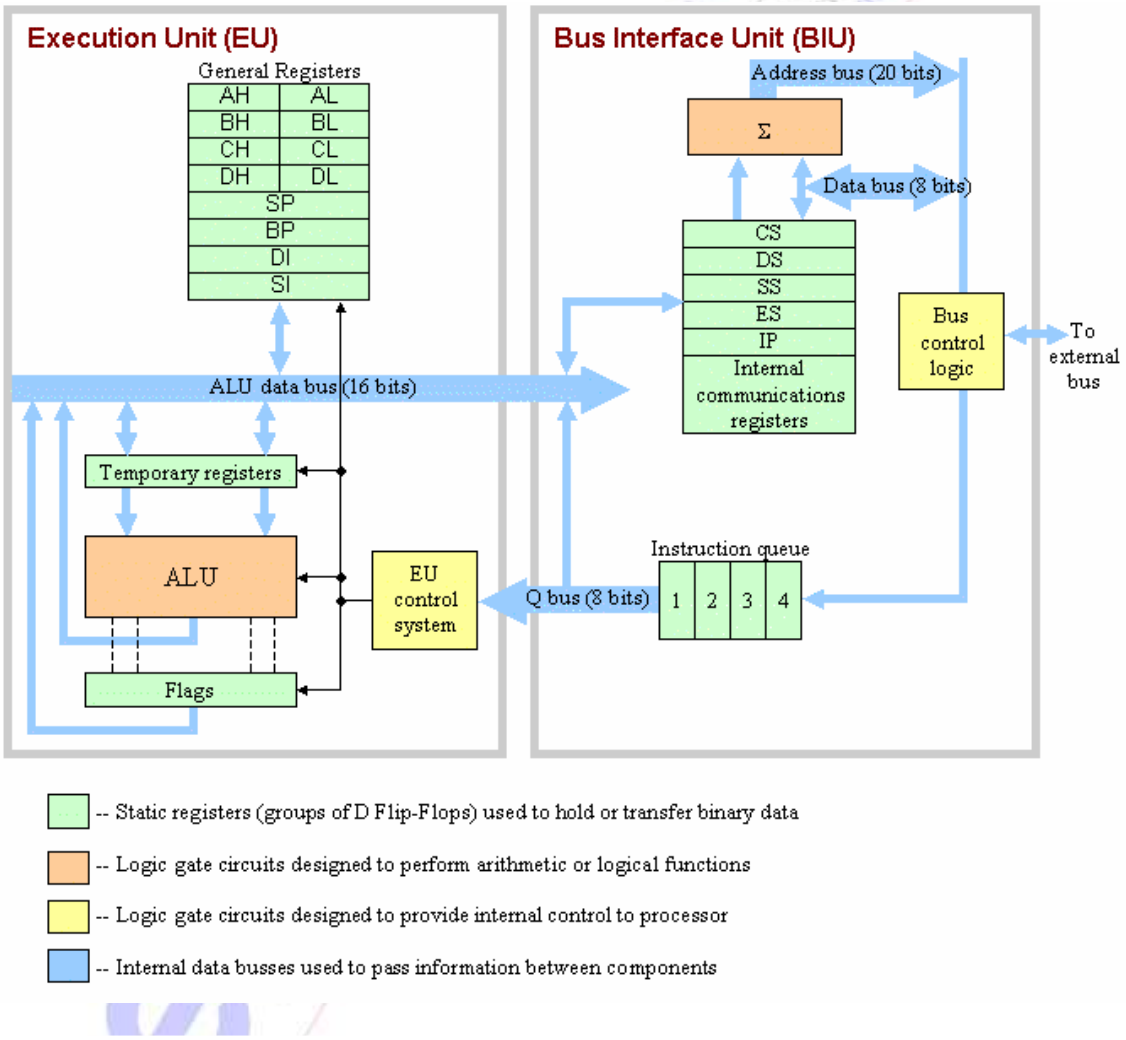
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
Km10 Đường Nguyễn Trãi, Hà Đông-Hà Tây
Tel: (04).5541221, Fax: (04).5540587
Website: http://www.e-ptit.edu.vn; E-mail: dhtx@e-ptit.edu.vn
CHƯƠNG TRÌNH
ĐÀO TẠO ĐẠI HỌC TỪ XA
PTIT

CHƯƠNG 1: GIỚI THIỆU

1.1 CẤU TRÚC BỘ VI XỬ LÝ

Phần này trình bày kiến trúc bên trong của bộ Vi xử lý 8088 và bộ Vi xử lý Pentium IV

1.1.1 Sơ đồ kiến trúc bộ Vi xử lý 8088



Hình 1.1: Kiến trúc bên trong của bộ Vi xử lý 8088

Bộ vi xử lý 8088 được chia làm 2 khối chính: Khối giao diện bus (BIU) và khối thực hiện lệnh (EU).

Các thành phần bên trong của CPU giao tiếp với nhau thông qua các bus trong. Giữa khối giao diện bus và khối thực hiện lệnh được liên hệ với nhau thông qua hàng đợi dữ liệu và hệ thống bus trong.

1.1.2 Chức năng các thành phần

1. Thành phần điều khiển Bus (Bus Control Logic)

Điều khiển các loại tín hiệu trên các bus bao gồm: các tín hiệu trên bus địa chỉ (20 bit), các tín hiệu trên bus dữ liệu (8 bit) và các tín hiệu trên bus điều khiển. Ngoài ra, thành phần này còn làm nhiệm vụ hỗ trợ giao tiếp giữa hệ thống bus trong và bus ngoài. Hệ thống bus ngoài là hệ thống bus kết nối giữa các thành phần của hệ vi xử lý với nhau: CPU, Bộ nhớ trong và Thiết bị vào/ra.

2. Hàng đợi lệnh (Prefetch Queue)

Chứa mã lệnh chờ được xử lý. Hàng đợi lệnh có kích thước 4 byte đối với 8088 và 6 byte đối với 8086. Sở dĩ có điều này là vì hàng đợi lệnh phải có kích thước có thể chứa được ít nhất một lệnh có độ dài bất kỳ (dài nhất) của bộ vi xử lý. Mà tập lệnh của 8086 chứa các lệnh có độ dài từ 1-6 byte.

Hàng đợi lệnh làm việc theo cơ chế FIFO (First In First Out), nghĩa là lệnh nào được đưa vào hàng đợi lệnh trước sẽ được xử lý trước

3. Khối điều khiển (Control Unit)

Khối điều khiển có hai chức năng chính: giải mã lệnh và tạo xung điều khiển . Đầu vào của khối điều khiển là mã lệnh được đọc từ hàng đợi lệnh và đầu ra là các xung điều khiển gửi đến các bộ phận khác nhau bên trong bộ vi xử lý. Quá trình này được thực hiện nhờ hai mạch giải mã lệnh và mạch tạo xung.

4. Khối số học và logic (Arithmetic Logic Unit)

Khối số học và logic có chức năng thực hiện các phép tính toán như phép cộng, trừ… hay các phép logic như AND, OR, NOT. Đầu vào ALU là hai thanh ghi tạm thời chứa dữ liệu của cho phép tính được lấy từ bus dữ liệu. Kết quả đầu ra của ALU được đưa trở lại bus dữ liệu và phản ánh vào thanh ghi cờ (flag register).

5. Các thanh ghi đoạn (Segment registers)

Ta hãy thử xem đoạn chương trình được viết bằng ngôn ngữ C sau:

```
int Cong(int a, int b)
{
   Return (a+b);
}
void main()
{
    int x=3; int  y=4;
    printf("Tong: %d", Cong(x,y));
 }
```

Trong chương trình trên có 2 phần: phần khai báo và phần lệnh của chương trình. Trong phần lệnh có thể có lời gọi chương trình con.

Như vậy để thực hiện được một chương trình (dạng .EXE) thì người ta cần ít nhất 3 đoạn bộ nhớ (segment). Đoạn dành chứa dữ liệu được khai báo, đoạn chứa mã chương trình, đoạn ngăn

xếp phục vụ cho các lời gọi chương trình con. Mỗi đoạn có kích thước 64KB. Khi chương trình được thực hiện, mỗi đoạn bộ nhớ này được trỏ bởi các thanh ghi đoạn. Đó là:

- Thanh ghi đoạn mã CS (Code Segment): trỏ đến đoạn bộ nhớ chứa mã của chương trình.
- Thanh ghi đoạn dữ liệu DS (Data Segment): trỏ đến đoạn bộ nhớ chứa các khai báo của chương trình.
- Thanh ghi đoạn ngăn xếp SS (Stack Segment): trỏ đến đoạn bộ nhớ dành cho stack.
- Ngoài ra, trong nhiều trường hợp người ta sử dụng thêm một đoạn dữ liệu phụ dùng trong trường hợp các dữ liệu cần khai báo vượt quá kích thước cho phép của 1 đoạn (các khai báo mảng, file…). Khi đó thanh ghi đoạn dữ liệu phụ ES (Extra Segment) sẽ trỏ đến đoạn này

6. Các thanh ghi con trỏ và chỉ số (pointers and index registers)

Các thanh ghi con trỏ và chỉ số là các thanh ghi 16 bit. Chúng thường được lưu địa chỉ lệch (offset) và kết hợp với thanh ghi đoạn tương ứng tạo thành cặp thanh ghi chứa địa chỉ xác định của mã lệnh, mục dữ liệu, hoặc mục dữ liệu lưu trong stack. Nhờ vào cặp thanh ghi này, người ta có thể tính đia chỉ vật lý cụ thể theo công thức sau:

*Địa chỉ vật lý = địa chỉ đoạn * 16 + địa chỉ lệch*

Dưới đây là các thanh ghi con trỏ và chỉ số:

- Thanh ghi con trỏ lệnh IP (Instruction Pointer): trỏ vào lệnh kế tiếp sẽ được thực hiện nằm trong đoạn mã do con trỏ CS trỏ tới. Địa chỉ đầy đủ của lệnh là CS:IP.
- Thanh ghi con trỏ cơ sở BP (Base Pointer): trỏ vào một mục dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của mục dữ liệu là CS:IP.
- Thanh ghi con trỏ ngăn xếp SP (Stack Pointer): trỏ vào đỉnh hiện thời ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của đỉnh ngăn xếp là SS:SP.
- Thanh ghi chỉ số nguồn SI (Source Index): trỏ vào một mục dữ liệu trong đoạn DS. Địa chỉ đầy đủ của mục dữ liệu là DS:SI.
- Thanh ghi chỉ số đích DI (Destination Index): trỏ vào một mục dữ liệu trong đoạn DS. Địa chỉ đầy đủ của mục dữ liệu là DS:DI.

7. Các thanh ghi đa năng (Multi-purposed registers)

Bộ xử lý 8088 có 4 thanh ghi đa năng 16 bit đó là: AX, BX, CX và DX. Các thanh ghi này cũng có thể được tách ra thành 2 nửa gồm 8 bít cao (nửa cao) gồm bít thứ 8 đến bít thứ 15 và 8 bít thấp (nửa thấp) gồm các bít thứ 0 đến 7. Các nửa thanh ghi này có thể được sử dụng một cách độc lập để chứa các dữ liệu 8 bít. Đó là các nửa thanh ghi: AH và AL, BH và BL, CH và CL, và DH và DL. Trong đó AH, BH, CH, DH la các nửa cao còn AL,BL, CL, DL là các nửa thấp.

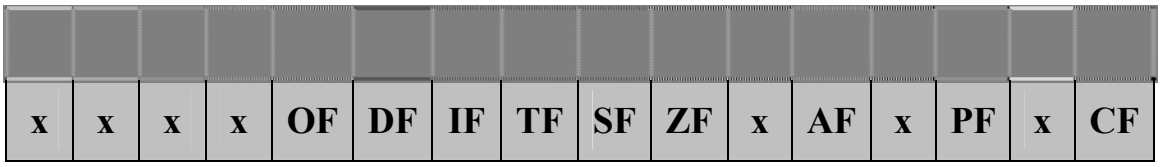
Ngoài chức năng "đa năng", mỗi thanh ghi 16 bít thường được sử dụng trong các tác vụ đặc biệt, giống như tên của chúng:

- AX (Accumulator) thanh chứa: các kết quả của các phép toán thường được lưu vào thanh ghi này. Ngoài ra, AX còn là toán hạng ẩn cho 1 số phép toán như nhân (AX là thừa số) hoặc chia (AX là số bị chia).

- BX (Base) thanh ghi cơ sở: thường được dùng để chứa các địa chỉ cơ sở.
- CX (Count) bộ đếm: CX thường dung để chứa số lần lặp trong trường hợp dùng lệnh LOOP. Ngoài ra, CL còn chứa số lần dịch chuyển, quay trái, quay phải của các toán hạng.
- DX (Data) thanh ghi dữ liệu: DX thường được chứa địa chỉ offset của xâu kí tự khi có các thao tác nhập vào xâu hoặc in xâu. DX (cùng với AX) còn tham gia chứa kết quả của phép nhân các số 16 bit hoặc làm số bị chia cho phép chia các số 16 bit. Ngoài ra, DX còn dùng để chứa địa chỉ của các cổng vào/ra trong trường hợp thực hiện các lệnh IN hoặc OUT.

8. Thanh ghi cờ (flag register)

Thanh ghi cờ là thanh ghi lưu trữ trạng thái của CPU tại mỗi thời điểm. Thanh ghi cờ có 16 bít, trong đó có 7 bít dự trữ cho tương lai (CPU 8088 chưa dùng đến các bít này). Còn lại 9 bít và mỗi bít tương ứng là một cờ. Kết hợp các lệnh nhảy có điều kiện (conditional jump) với các cờ này, người lập trình dễ dàng hơn



Hình 1.2: Cấu trúc của thanh ghi cờ của CPU 8088.

Các bit được đánh dấu x là các cờ chưa được dùng đến.

- *Cờ CF* (Carry Flag): cờ nhớ CF=1 khi có nhớ hoặc trừ có mượn từ bít có trọng số cao nhất (Most Significant Bit). Ngoài ra, cờ CF=1 trong trường hợp khi thao tác với file hoặc thư mục gây ra lỗi như các lỗi tạo, xóa file và thư mục.
- *Cờ PF* (Parity Flag): cờ chẵn lẻ PF=1 khi tổng số các bít bằng 1 trong kết quả của phép tính là một số chẵn.
- *Cờ AF* (Auxiliary Carry Flag): cờ nhớ phụ AF =1 khi có nhớ từ bít thứ 4 sang bít thứ 5 hoặc có mượn từ bít 5 sang bít thứ 4 trong biểu diễn BCD của 1 số.
- *Cờ ZF* (Zero Flag): cờ Zero ZF=1 khi kết quả tính toán bằng 0.
- *Cờ SF* (Sign Flag): cờ dấu SF=1 khi kết quả tính toán là một số âm.
- *Cờ TF* (Trap Flag): cờ bẫy TF=1 khi CPU đang làm việc ở chế độ chạy từng lệnh. Chế độ này được sử dụng cần thiết khi tìm lỗi (defect) và gỡ lỗi (debug) chương trình.
- *Cờ IF* (Interrupt enable Flag): cờ cho phép ngắt IF=1, cho phép tác động đến yêu cầu ngắt che được (maskable interrupts).
- *Cờ DF* (Direction Flag): cờ hướng DF=1 khi CPU xử lý chuỗi kí tự theo thứ tự từ phải sang trái.
- *Cờ OF* (Overflow Flag): cờ tràn OF=1 khi kết quả là một số bù hai vượt ra ngoài giới hạn biểu diễn dành cho nó.

9. Hệ thống bus trong (Internal bus system)

Hệ thống bus bên trong của CPU 8088 bao gồm 3 loại:

- Bus *dữ* liệu: 16 bít, cho phép di chuyển 2 byte dữ liệu tại một thời điểm
- Bus địa chỉ: 20 bít, có thể địa chỉ hóa được 2^{20} bytes và vì thế không gian địa chỉ nhớ của CPU 8088 là 1MB.
- Bus điều khiển: truyền tải các tín hiệu điều khiển như RD, WR …

1.2 MỘT SỐ CHỨC NĂNG CỦA NGẮT 21H

Phần này trình bày các hàm thông dụng của ngắt 21h. Đó là các hàm thao tác vào/ra đối với kí tự, chuỗi ký tự, file, thư mục, kết thúc chương trình và trả lại quyền điều khiển cho Hệ điều hành DOS.

Hàm 01: đọc 1 kí tự (có hiện) từ bàn phím

Input: AH=01

Output: AL= mã ASCII của ký tự

AL=0 nếu gõ vào phím chức năng.

Hàm 02: hiện 1 kí tự lên màn hình

Input: AH=02

DL= mã ASCII của ký tự cần hiển thị

Output:

Hàm 08: đọc 1 kí tự (không hiện) từ bàn phím

Input: AH=08

Output: AL= mã ASCII của ký tự

AL=0 nếu gõ vào phím chức năng.

Hàm 09: hiện xâu kí tự kết thúc bởi '$' lên màn hình

Input: AH = 09

DX = địa chỉ offset của xâu kí tự

Hàm 0Ah: đọc xâu kí tự từ bàn phím

Input: AH = 09

DX = địa chỉ offset của vùng đệm chứa xâu kí tự

Output: DX = địa chỉ offset của xâu kí tự

Hàm 39h: tạo thư mục

Input: AH = 39h

DX = địa chỉ offset của tên thư mục

Output:

- Nếu thành công, thư mục được tạo ra
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 3Ah: xóa thư mục

Input: AH = 3Ah

DX = địa chỉ offset của tên thư mục

Output:

- Nếu thành công, thư mục được xóa
- Nếu không thành công, CF=1 và AX=mã lỗi.

Hàm 3Ch: tạo file

Input: AH = 3Ch
DX = địa chỉ offset của tên file
CX = thuộc tính file

Output:

- Nếu thành công, file được tạo ra, CF=0 và AX= thẻ file (file handle)
- Nếu không thành công, CF=1 và AX= mã lỗi.

Thuộc tính file được định nghĩa như sau:

00h: file bình thường (plain old file)
01h: file chỉ đọc (Read Only)
02h: file ẩn (Hidden from searches)
04h: file hệ thống (system)
08h: thuộc tính cho nhãn đĩa.
10h: thuộc tính cho thư mục con.

Hàm 3Dh: mở file

Input: AH = 3Dh
AL = mode

Output:

- Nếu thành công, file được tạo ra, CF=0 và AX= thẻ file (file handle)
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 3Eh: đóng file

Input: AH = 3Eh
BX = thẻ file

Output:

- Nếu thành công, file được đóng lại và CF=0
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 3Fh: đọc từ file

Input: AH = 3Fh
DS:DX = địa chỉ offset của vùng đệm
CX = số byte cần đọc
BX = thẻ file

Output:

- Nếu thành công, CF=0 và AX= số byte đã đọc được
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 40h: ghi vào file

Input: AH = 40h

DS:DX = địa chỉ offset của vùng đệm

CX = số byte cần ghi

BX = thẻ file

Output:

- Nếu thành công, file được ghi và CF=0.
- Nếu không thành công, CF=1 và AX= mã lỗi.

Hàm 41h: xóa file

Input: AH = 41h

DX = địa chỉ offset của tên file

Output:

- Nếu thành công, file bị xóa
- Nếu không thành công, CF=1 và AX=mã lỗi.

Hàm 4Ch: kết thúc chương trình

Input: AH = 4Ch

Output:

Kết thúc chương trình, trả lại quyền điều khiển cho hệ điều hành.

1.3 GIỚI THIỆU VỀ TẬP LỆNH CỦA 8088

Phần này giới thiệu về một số lệnh thông dụng của bộ vi xử lý 8088. Để tiện dụng cho người học lập trình, các lệnh được chia thành các nhóm lệnh.

1.3.1 Nhóm lệnh di chuyển dữ liệu

1. *Lệnh:* MOV

Chức năng: chuyển giá trị từ toán hạng nguồn vào toán hạng đích

Cú pháp:

MOV	**Dst,src**	**Ví dụ**
	Reg1,reg2	Mov AX,BX
	Reg, data	Mov AH,9Fh
	Mem,reg	Mov [BX],AL
	Reg,mem	Mov CL,[3456h]
	Mem,data	Mov PTR [BX], FFh

Chú ý: Data chỉ nằm ở phía toán hạng nguồn

Hai toán hạng dst và src không thể đồng thời là hai ô nhớ.

2. *Lệnh:* PUSH

Chức năng: chuyển giá trị của toán hạng nguồn vào đỉnh ngăn xếp

Cú pháp:

PUSH	**Src**	**Ví dụ**
	Reg16	push AX
	Mem16	push x
	Segreg	push DS

Chú ý: Toán hạng nguồn luôn có kích thước 16 bít
Toán hạng nguồn không thể là data (hằng số)

3. *Lệnh:* POP
Chức năng: Lấy giá trị của đỉnh ngăn xếp đưa vào toán hạng đích
Cú pháp:

POP	**Dst**	**Ví dụ**
	Reg16	Pop AX
	Mem16	Pop x
	Segreg	Pop DS

Chú ý: Toán hạng nguồn luôn có kích thước 16 bít
Toán hạng đích không thể là data (hằng số)

4. *Lệnh:* PUSHF
Chức năng: chuyển giá trị của thanh ghi cờ vào đỉnh ngăn xếp
Cú pháp:

PUSHF

5. *Lệnh:* POPF
Chức năng: lấy giá trị đỉnh ngăn xếp lưu vào thanh ghi cờ.
Cú pháp:

POPF

Chú ý: hai lệnh PUSHF và POPF được hệ thống tự động gọi khi chương trình có lệnh gọi ngắt hoặc gọi chương trình con.

6. *Lệnh:* XCHG
Chức năng: Hoán vị giá trị giữa toán hạng nguồn và đích

Cú pháp:

XCHG	**Dst,src**	**Ví dụ**
	Reg,Reg	XCHG AX,BX
	Reg,Mem	XCHG AL,[BX]
	Mem,Reg	XCHG [BX],AH

7. *Lệnh:* IN

Chức năng: Đọc giá trị từ 1 cổng vào thanh ghi AL hoặc AX.

Cú pháp:

IN	**AL, địa chỉ cổng (8 bít)**	VD: IN AL,2Eh
IN	**AX, địa chỉ cổng (16 bít)**	VD: IN AX,2EBEh

8. *Lệnh:* OUT

Chức năng: Chuyển giá trị 1 byte hoặc 1 từ từ thanh ghi AL hoặc AX ra cổng.

Cú pháp:

OUT	**địa chỉ cổng (8 bít), AL**	VD: OUT 2Eh,AL
IN	**địa chỉ cổng (16 bít),AX**	VD: OUT 2EBEh,AX

1.3.2 Nhóm các lệnh tính toán số học

Phần này giới thiệu về các lệnh lien quan đến tính toán số học như các lệnh: cộng, trừ, nhân, chia, so sánh. Đồng thời, cũng giải thích sự tác động của các lệnh này lên các bit của thanh ghi cờ.

1. *Lệnh:* ADD

Chức năng: cộng toán hạng nguồn và toán hạng đích, lưu kết quả vào toán hạng đích.

Cú pháp:

ADD	**Dst,src**	**Ví dụ**
	Reg1,reg2	Add AX,BX
	Reg, data	Add AH,19h
	Mem,reg	Add [BX],AL
	Reg,mem	Add CL,[3456h]
	Mem,data	Add [BX], 1Fh

Chú ý:

- Không cộng trực tiếp 2 biến ô nhớ với nhau
- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

2. *Lệnh:* INC

Chức năng: Tăng giá trị của toán hạng đích lên 1.

Cú pháp:

INC	**Dst**	**Ví dụ**
	Reg	Inc CX
	Mem	Inc x

Chú ý:

- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

3. *Lệnh:* SUB

Chức năng: Trừ toán hạng đích cho toán hạng nguồn, lưu kết quả vào toán hạng đích.

Cú pháp:

SUB	**Dst,src**	**Ví dụ**
	Reg1,reg2	Sub AX,BX
	Reg, data	Sub AH,19h
	Mem,reg	Sub [BX],AL
	Reg,mem	Sub CL,[3456h]
	Mem,data	Sub [BX], 1Fh

Chú ý:

- Không trừ trực tiếp 2 biến ô nhớ với nhau
- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

4. *Lệnh:* DEC

Chức năng: Giảm giá trị của toán hạng đích đi 1.

Cú pháp:

DEC	**Dst**	**Ví dụ**
	Reg	DEC CX
	Mem	DEC [BX]

Chú ý:

- Toán hạng đích không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, SF, ZF, AF, PF, CF.

5. *Lệnh:* MUL

Chức năng: Nhân nội dung của toán hạng AX hoặc AL với nội dung của toán hạng nguồn. Giá trị của hai toán hạng đều là dạng không dấu. Kết quả sẽ được cất như sau:

- Nếu là phép nhân hai toán hạng 8 bít thì kết quả sẽ được đặt trong thanh ghi AX.
- Nếu là phép nhân hai toán hạng 16 bít thì kết quả sẽ được đặt trong thanh nghi DX:AX.

Cú pháp:

MUL	**Src**	**Ví dụ**
	Reg	MUL CL
	Mem	MUL x

Chú ý:

- Toán hạng nguồn không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, ZF,CF.

6. *Lệnh:* DIV

Chức năng: Chia giá trị của thanh ghi AX hoặc DX:AX cho nội dung của toán hạng nguồn. Giá trị của hai toán hạng đều là dạng không dấu. Kết quả sẽ được cất như sau:

- Nếu số bị chia là toán hạng 16 bít thì phần thương sẽ được đặt trong thanh ghi AL và phần dư sẽ được đặt trong thanh ghi AH.
- Nếu số bị chia là toán hạng 32 bít thì phần thương sẽ được đặt trong thanh ghi AX và phần dư sẽ được đặt trong thanh ghi DX.

Cú pháp:

DIV	**Src**	**Ví dụ**
	Reg	DIV CL
	Mem	DIV [BX]

Chú ý:

- Toán hạng nguồn không thể là hằng số
- Kết quả có thể tác động đến các cờ: OF, CF.

7. *Lệnh:* CMP

Chức năng: So sánh giá trị của toán hạng đích và toán hạng nguồn. Nội dung của hai toán hạng đều không thay đổi sau lệnh này. Thực chất, lệnh này thực hiện bằng cách lấy toán hạng đích trừ đi toán hạng nguồn. Kết quả phản ánh lên thanh ghi cờ mà không được lưu lại.

Cú pháp:

CMP	**Dst,src**	**Ví dụ**
	Reg1,reg2	Cmp AX,BX
	Reg, data	Cmp AH,9Fh
	Mem,reg	Cmp [BX],AL

Reg,mem	Cmp CL,[3456h]
Mem,data	Cmp [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: OF, SF,ZF,AF,PF,CF.

1.3.3 Nhóm các lệnh thao tác bít

Phần này giới thiệu về các lệnh liên quan đến các lệnh logic, các lệnh dịch chuyển bít, các lệnh quay vòng các bit.

1. *Lệnh:* NOT

- *Chức năng:* Đảo giá trị từng bít một của toán hạng đích.

Cú pháp:

NOT	**Dst**	**Ví dụ**
	Reg	NOT CL
	Mem	NOT [BX]

Chú ý:

- Toán hạng đích không thể là hằng số

2. *Lệnh:* AND

Chức năng: Thực hiện phép VÀ logic giữa hai toán hạng. Kết quả đặt ở trong toán hạng đích.

Cú pháp:

AND	**Dst,src**	**Ví dụ**
	Reg1,reg2	And AX,BX
	Reg, data	And AH,9Fh
	Mem,reg	And [BX],AL
	Reg,mem	And CL,[3456h]
	Mem,data	And [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

3. *Lệnh:* OR

Chức năng: Thực hiện phép HOẶC logic giữa hai toán hạng. Kết quả đặt ở trong toán hạng đích.

Cú pháp:

OR	Dst,src	Ví dụ
	Reg1,reg2	And AX,BX
	Reg, data	And AH,9Fh
	Mem,reg	And [BX],AL
	Reg,mem	And CL,[3456h]
	Mem,data	And [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

4. *Lệnh:* XOR

Chức năng: Thực hiện phép EXCLUSIVE OR logic giữa hai toán hạng (các bít của kết quả có giá trị là 1 nếu hai bít tương ứng của 2 toán hạng là khác nhau). Kết quả đặt ở trong toán hạng đích.

Cú pháp:

XOR	Dst,src	Ví dụ
	Reg1,reg2	Xor AX,BX
	Reg, data	Xor AH,9Fh
	Mem,reg	Xor [BX],AL
	Reg,mem	Xor CL,[3456h]
	Mem,data	Xor [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

5. *Lệnh:* TEST

Chức năng: So sánh nội dung của hai toán hạng bằng cách thực hiện lệnh AND giữa hai toán hạng mà không lưu lại kết quả. Kết quả tác động đến thanh ghi cờ.

Cú pháp:

TEST	Dst,src	Ví dụ
	Reg1,reg2	Test AX,BX
	Reg, data	Test AH,9Fh
	Mem,reg	Test [BX],AL
	Reg,mem	Test CL,[3456h]
	Mem,data	Test [BX], FFh

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Kết quả có thể tác động đến các cờ: SF,ZF, PF.

6. *Lệnh:* SHL/SAL

Chức năng: Dịch trái các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT.

Cú pháp:

SHL/SAL	**Dst,COUNT**	**Ví dụ**
	Reg	SHL AL,CL
	Mem	SHL [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, SHL/SAL Dst,1.
- Kết quả có thể tác động đến các cờ: OF,SF,ZF, PF,CF.

7. *Lệnh:* SHR

Chức năng: Dịch phải các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT.

Cú pháp:

SHR	**Dst,COUNT**	**Ví dụ**
	Reg	SHR AL,CL
	Mem	SHR [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, SHR Dst,1.
- Kết quả có thể tác động đến các cờ: OF,SF,ZF, PF,CF.

8. *Lệnh:* ROR

Chức năng: Quay vòng phải các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT. Trong mỗi lần quay, giá trị bít thấp nhất vừa chuyển vào thanh ghi cờ CF đồng thời chuyển vào bít cao nhất.

Cú pháp:

ROR	**Dst,COUNT**	**Ví dụ**
	Reg	ROR AL,CL
	Mem	ROR [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, ROR Dst,1.
- Kết quả có thể tác động đến các cờ: OF, CF.

9. *Lệnh:* ROL

Chức năng: Quay vòng trái các bít của toán hạng đích đi COUNT lần. Trong đó CL=COUNT. Trong mỗi lần quay, giá trị bít cao nhất vừa chuyển vào thanh ghi cờ CF đồng thời chuyển vào bít thấp nhất.

Cú pháp:

ROL	**Dst,COUNT**	**Ví dụ**
	Reg	ROL AL,CL
	Mem	ROL [BX],CL

Chú ý:

- Hai toán hạng nguồn và đích không thể đồng thời là hằng số hoặc ô nhớ.
- Khi Count=1 thì có thể đặt 1 trực tiếp vào toán hạng, ROL Dst,1.
- Kết quả có thể tác động đến các cờ: OF, CF.

1.3.4 Nhóm các lệnh làm việc với xâu kí tự

1. *Lệnh:* MOVSB(hay MOVSW)

Chức năng: Chuyển một xâu kí tự theo từng byte (hay theo từng từ) từ một vùng nhớ nguồn sang vùng nhớ đích. Trong đó DS:SI trỏ đến xâu kí tự nguồn và ES:DI trỏ đến xâu kí tự đích. Sau mỗi lần chuyển 1 byte (hoặc 1 từ) thì giá trị của SI và DI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

MOVSB Hoặc

MOVSW

2 *Lệnh:* CMPSB (CMPSW)

Chức năng: So sánh hai xâu kí tự theo từng byte (hay theo từng từ) nằm ở hai vùng nhớ. Trong đó, DS:SI và ES:DI trỏ đến hai xâu kí tự. Sau mỗi lần so sánh từng byte (hoặc từng từ) thì giá trị của SI và DI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

CMPSB Hoặc

CMPSW

- Kết quả có thể tác động đến các cờ: OF, SF,ZF,AF,PF, CF.

3 *Lệnh:* LODSB (LODSW)

Chức năng: Chuyển nội dung theo từng byte (hay theo từng từ) của vùng nhớ trỏ bởi DS:SI vào thanh ghi AL (hoặc AX). Sau mỗi lần chuyển từng byte (hoặc từng từ) thì giá trị của SI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

LODSB Hoặc

LODSW

4. *Lệnh:* STOSB (STOSW)

Chức năng: Chuyển nội dung theo từng byte (hay theo từng từ) của thanh ghi AL (hoặc AX) vùng nhớ trỏ bởi ES:DI. Sau mỗi lần chuyển từng byte (hoặc từng từ) thì giá trị của DI tự động tăng lên 1 (hoặc 2) nếu cờ hướng DF=0, hoặc giảm đi 1 (hoặc 2) nếu cờ hướng DF=1.

Cú pháp:

STOSB Hoặc

STOSW

1.3.5 Nhóm các lệnh nhảy

Nhóm các lệnh nhảy bao gồm 4 nhóm nhỏ: các lệnh nhảy không điều kiện, các lệnh nhảy có điều kiện, các lệnh lặp và các lệnh gọi ngắt mềm.

a. Các lệnh nhảy không điều kiện

1. Lệnh CALL

Chức năng: Gọi chương trình con

Cú pháp:

CALL Địa chỉ

Nhãn

Tên chương trình con

Reg

Mem

2. Lệnh RET

Chức năng: Quay trở về chương trình đã gọi chương trình con

Cú pháp:

RET

3. Lệnh JMP

Chức năng: Lệnh nhảy không điều kiện

Cú pháp:

JMP Địa chỉ

Nhãn

Tên chương trình con

Reg

Mem

Chú ý: Bước nhảy của lệnh nhảy này nằm trong một đoạn 64KB.

b. Các lệnh nhảy có điều kiện

Chức năng: Lệnh nhảy không điều kiện

Cú pháp:

Lệnh	Toán hạng	Giải thích
JE/JZ	Nhãn	Nhảy nếu ZF=1 hoặc 2 toán hạng của phép so sánh bằng nhau
JNE/JNZ	Nhãn	Nhảy nếu ZF=0 hoặc 2 toán hạng của phép so sánh khác nhau
JL/JNGE	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn toán hạng bên phải của phép so sánh (CF=1)
JB/JNAE/JC	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn toán hạng bên phải của phép so sánh (SF<>0)
JLE/JNG	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn hoặc bằng toán hạng bên phải của phép so sánh (SF<>OF hoặc ZF=0)
JBE/JNA	Nhãn	Nhảy nếu toán hạng bên trái nhỏ hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ CF=1 hoặc SF=1)
JG/JNLE	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn toán hạng bên phải của phép so sánh.
JA/JNBE	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ CF=0 hoặc ZF=0)
JGE/JNL	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ SF=OF).
JAE/JNB/JNC	Nhãn	Nhảy nếu toán hạng bên trái lớn hơn hoặc bằng toán hạng bên phải của phép so sánh (Cờ CF=0).
JP/JPE	Nhãn	Nhảy nếu cờ parity là chẵn (PF=1)
JNP/JPO	Nhãn	Nhảy nếu cờ parity là lẻ (PF=0)
JO	Nhãn	Nhảy nếu tràn (OF=1)
JNO	Nhãn	Nhảy nếu không tràn (OF=0)
JS	Nhãn	Nhảy nếu cờ dấu =1 (SF=1)
JNS	Nhãn	Nhảy nếu cờ dấu =0 (SF=0)
JCXZ	Nhãn	Nhảy nếu giá trị của thanh ghi CX =0.

Chú ý: Các bước nhảy của các lệnh nhảy có điều kiện không vượt quá 128 byte.

c. *Các lệnh lặp*

Chức năng: Thực hiện vòng lặp cho đến khi điều kiện thỏa mãn.

Cú pháp:

Lệnh	Toán hạng	Giải thích
LOOP	Nhãn	Lặp khối lệnh từ Nhãn đến LOOP cho đến khi giá trị của CX=0. Sau mỗi lần thực hiện vòng lặp giá trị của CX tự động giảm đi 1.
LOOPZ/LOOPE	Nhãn	Lặp khối lệnh từ Nhãn đến LOOPZ hoặc LOOPE cho đến khi giá trị của CX=0 và cờ ZF=1. Sau mỗi lần thực hiện vòng lặp giá trị của CX tự động giảm đi 1.
LOOPNZ/LOOPNE	Nhãn	Lặp khối lệnh từ Nhãn đến LOOPZ hoặc LOOPE cho đến khi giá trị của CX<>0 và cờ ZF=0. Sau mỗi lần thực hiện vòng lặp giá trị của CX tự động giảm đi 1.

d. Các lệnh gọi ngắt mềm.

1. Lệnh: INT

Chức năng: Thực hiện ngắt mềm.

Cú pháp: **INT số hiệu ngắt (dạng hexa)**

Các cờ bị tác động: IF, TF

2. Lệnh: IRET

Chức năng: Trở về chương trình chính (chương trình gọi nó) sau khi thực hiện chương trình con phục vụ ngắt.

Cú pháp: **IRET**

Các cờ bị tác động: OF,SF,ZF,AF,PF,CF.

1.3.6 Các lệnh điều khiển khác

Phần này giới thiệu về một số lệnh điều khiển: thao tác với thanh ghi cờ, lệnh HLT và NOP.

1. Lệnh: CLC

Chức năng: Xóa giá trị cờ CF về 0 (CF=0).

Cú pháp: **CLC**

Các cờ bị tác động: CF.

2. Lệnh: CMC

Chức năng: Đảo giá trị hiện thời của cờ CF.

Cú pháp: **CMC**

Các cờ bị tác động: CF.

3. Lệnh: STC

Chức năng: Đặt cờ CF=1.

Cú pháp: **STC**

Các cờ bị tác động: CF.

4. Lệnh: CLD

Chức năng: Xoá giá trị cờ DF về 0 (DF=0).

Cú pháp: **CLD**

Các cờ bị tác động: DF.

5. Lệnh: STD

Chức năng: Đặt giá trị cờ DF bằng 1 (DF=1).

Cú pháp: **STD**

Các cờ bị tác động: DF.

6. Lệnh: CLI

Chức năng: Xoá giá trị cờ IF về 0 (IF=0). Cấm các ngắt cứng hoạt động, trừ các ngắt không che.

Cú pháp: **CLI**

Các cờ bị tác động: IF.

7. Lệnh: STI

Chức năng: Đặt giá trị cờ IF bằng 1 (IF=1). Cấm các ngắt cứng hoạt động.

Cú pháp: **STI**

Các cờ bị tác động: IF.

8. Lệnh: HLT

Chức năng: dừng máy.

Cú pháp: **HLT**

9 Lệnh: NOP

Chức năng: Không thực hiện gì

Cú pháp: **NOP**

Chú ý: Lệnh NOP rất có ý nghĩa khi CPU thực hiện các chu kỳ đợi và được xen vào một số chu kỳ lệnh trong quá trình thực hiện lệnh theo cơ chế pipeline.

1.4 TÓM TẮT

Chương này đã trang bị cho sinh viên những kiến thức cơ bản để chuẩn bị cho các phần kế tiếp: Lập trình bằng Hợp ngữ. Chương này có ba phần chính:

- Giới thiệu về cấu trúc bộ vi xử lý 8088. Đây là bộ vi xử lý khá đơn giản và dễ hiểu về mặt kiến trúc. Sơ đồ kiến trúc bao gồm hai khối chính: Khối giao diện BUS và khối thực hiện lệnh. Đồng thời, phần này cũng đề cập chi tiết chức năng của các thành phần bên trong bộ vi xử lý.
- Một số chức năng của ngắt 21h. Đây là một ngắt quan trọng nhất của hệ điều hành MS DOS. Ngắt 21h cung cấp nhiều các chức năng khác nhau cho các nhà lập trình hệ thống. Phần này đã giới thiệu 14 chức năng thông dụng của ngắt 21h. Từ các chức năng phục vụ vào ra đối với kí tự, xâu kí tự cho đến các chức năng phục vụ cho thao tác các file và thư mục.
- Tập lệnh của 8088 dạng hợp ngữ. Phần này trình bầy về một số lệnh thông dụng trong tập lệnh của 8088. Để tiện lợi cho người học lập trình ở các phần sau chúng tôi phân chia các lệnh ra thành các nhóm lệnh. Mỗi nhóm lệnh bao gồm một số lệnh thực hiện 1 số chức năng.

1.5 CÂU HỎI VÀ BÀI TẬP

Dưới đây là các câu hỏi dạng lựa chọn. Sinh viên sẽ lựa chọn **một và chỉ một** phương án trả lời **đúng nhất** cho mỗi câu hỏi

Câu 1: Khối giao diện bus (BIU) và khối thực hiện lệnh (EU) giao tiếp với nhau thông qua:

A. Hàng đợi lệnh

B. Hệ thống bus trong

C. Hệ thống bus trong và hàng đợi lệnh

D. Không có liên hệ gì với nhau.

Câu 2: Một trong những chức năng của thành phần điều khiển (CU) là:

A. Tạo xung điều khiển

B. Giải mã địa chỉ

C. Chứa các thanh ghi

D. Chứa các lệnh sắp được xử lý

Câu 3: Các nhóm thanh ghi (16 bít) nào dưới đây có thể được chia làm 2 nửa thanh ghi 8 bít độc lập với nhau:

A. AX,BX,DS

B. SP,IP,CX

C. ES, SS, DS

D. DX,AX,CX

Câu 4: Các nhóm thanh ghi (16 bít) nào dưới đây có thể được chia làm 2 nửa thanh ghi 8 bít độc lập với nhau:

A. AX,BX,DS

B. SP,IP,CX

C. ES, SS, DS

D. DX,AX,CX

Câu 5: Các thanh ghi đoạn nào dưới đây trỏ vào các đoạn dữ liệu của cùng một chương trình .EXE:

A. DS,ES

B. DS,SS

C. SS, SP

D. DS

Câu 6: Lệnh nào dưới đây là sai cú pháp:

A. Mov AL,[BX+1]

B. Mov [BX+1], AL

C. Mov [AL],[BX+1]

D. Mov [CX], 3

Câu 7: Lệnh nào dưới đây là sai cú pháp:

A. Mul [BX+5]
B. Div [BX+5]
C. Mul AL,3
D. Mul DX

Câu 8: Lệnh nào dưới đây thực hiện việc xóa thanh ghi AX.

A. AND AX,AX
B. XOR AX,AX
C. NOT AX
D. MOV AX,[0000H]

Câu 9: Lệnh nào dưới đây thường được sử dụng trước các lệnh nhảy có điều kiện.

A. CMP
B. MOV
C. INT
D. RET

Câu 10 *: Xét đoạn chương trình dưới đây:

```
Mov AH,08
Int 21h
Mov DL,AL
Inc DL
Mov AH,02
Int 21h
```

Đoạn chương trình trên thực hiện công việc nào dưới đây:

A. Nhập một kí tự rồi in kí tự kế tiếp của kí tự đó ra màn hình
B. Nhập một kí tự rồi in kí tự đó ra màn hình
C. Nhập một xâu kí tự rồi in xâu đó ra màn hình
D. Nhập 1 kí tự không hiện lên kí tự đó.

1.6 TÀI LIỆU THAM KHẢO

1. Văn Thế Minh. Kỹ thuật Vi xử lý. Nhà XB Giáo dục 1997.
2. Đặng Thành Phu. Turbo Assembler và Ứng dụng. NXB Khoa học và Kỹ thuật 1998.
3. Nguyễn Minh San. Cẩm nang Lập trình hệ thống (bản dịch). NXB Tổng cục Thống kê.2001.

CHƯƠNG 2: LẬP TRÌNH BẰNG HỢP NGỮ

Chương này tìm hiểu về cách thức lập trình bằng hợp ngữ. Cách khai báo biến, hằng, khung chương trình, các cấu trúc lập trình, chương trình con và macro.

2.1 VIẾT VÀ THỰC HIỆN MỘT CHƯƠNG TRÌNH HỢP NGỮ

2.1.1 Cấu trúc lệnh và khai báo dữ liệu cho chương trình

Một chương trình bao gồm tập hợp các lệnh và các khai báo dữ liệu sử dụng trong chương trình nhằm mục đích giải quyết một vấn đề. Phần này trình bày về cấu trúc một dòng lệnh và các qui tắc khai báo biến, hằng trong một chương trình Hợp ngữ.

b. Cấu trúc dòng lệnh

Dưới Đây là một dòng lệnh đầy đủ của chương trình Hợp ngữ,. Trên thực tế, một dòng lệnh cần tối thiểu hai trường: trường mã lệnh và trường toán hạng. Các trường khác không bắt buộc cần phải đầy đủ.

Nhãn:	**Mã lệnh**	**Toán hạng**	**; Chú giải**

Ví dụ:

CongTiep: Add AL,[BX] ; cộng tiếp nội dung ô nhớ do thanh ghi BX ;trỏ tới vào AL

Các giải thích cho các trường:

Trường	Mô tả
Nhãn	Nhãn có thể là nhãn dung cho lệnh nhảy, tên thủ tục hoặc tên biến. Khi chạy chương trình, các nhãn này sẽ được chương trình dịch gán cho 1 địa chỉ ô nhớ xác định. Nhãn có thể chứa từ 1 đến 32 kí tự, không chứa dấu cách và phải bắt đầu bằng các kí tự a,b,c…,z. Nhãn được kết thúc bằng dấu hai chấm (:)
Mã lệnh	Chứa lệnh thật (Opcode) hoặc giả lệnh (Pseudo-Opcode). Với các lệnh thật thì trường này chứa mã lệnh gợi nhớ (thường là dạng viết ngắn hoặc đầy đủ của một động từ trong tiếng Anh). Trong quá trình chạy chương trình, mã lệnh thật sẽ được chương trình dịch dịch ra mã máy. Đối với các giả lệnh, thì chương trình dịch không dịch chúng ra mã máy.
Toán hạng	Đối với các lệnh thật thì trường này chứa các toán hạng cho lệnh đó. Lệnh thật của 8088 có thể có 0,1 hoặc 2 toán hạng. Toán hạng trong các giả lệnh của 8088 thường chứa các thông tin khác nhau như xác định mô hình bộ nhớ sử dụng, kích thước ngăn xếp…
Chú giải	Trường chú giải được bắt đầu bằng dấu chấm phẩy (;) để ghi những lời giải thích các lệnh của chương trình nhằm giúp cho người đọc chương trình một cách dễ hiểu hơn.

	Khi thực hiện chương trình, phần giải thích (sau dấu ;) sẽ bị bỏ qua. Ngoài ra, người ta cũng dùng trường này để ghi chu giải cho cả một đoạn chương trình, một chương trình con hay thâm chí là lời giới thiệu của bản quyền (copyright message) của người lập trình . Ví dụ: ; This program writen by X ; Last modified: 23/10/2006 ; This program is to delete a file …..

c. *Khai báo biến*

Các biến có thể được khai báo là ba kiểu dữ liệu khác nhau là: biến kiểu byte, biến kiểu từ (2 byte) và biến kiểu từ kép (4 byte).

Biến kiểu byte:

Dạng khai báo:	**Tên biến**	**DB**	**?**
Ví dụ:	X	DB	? ;không có giá trị khởi đầu
	Y	DB	4 ;giá trị khởi đầu là 4

Biến kiểu từ:

Dạng khai báo:	**Tên biến**	**DW**	**?**
Ví dụ:	X	DW	? ;không có giá trị khởi đầu
	Y	DW	4 ; giá trị khởi đầu là 4.

Biến kiểu từ kép:

Dạng khai báo:	**Tên biến**	**DD**	**?**
Ví dụ:	X	DD	? ;không có giá trị khởi đầu
	Y	DD	4Fh ; giá trị khởi đầu là 4Fh

Khai báo biến kiểu mảng

Mảng là một dãy liên tiếp các phần tử có cùng kiểu byte hoặc từ.

Ví dụ:

A1 DB 1,2,3,4,5

Là khai báo một biến mảng tên là A1, gồm 5 byte trong bộ nhớ. Nội dung ô nhớ [A1] có giá trị là 1, [A1+1] có giá trị là 2, [A1+2] có giá trị là 3…

Nếu muốn khai báo một mảng không cần khởi tạo giá trị ban đầu, ta khai báo như sau:

A2 DB 100 DUP(?)

khai báo một mảng có 100 phần tử và các phần tử là kiểu byte.

Khai báo sau là một mảng 100 phần tử kiểu byte và tất cả các phần tử được khởi tạo giá trị 0.

A3 DB 100 DUP(0)

Khai báo biến kiểu xâu kí tự

Xâu kí tự là một mảng mà mỗi phần tử là một kí tự hay mã ASCII của kí tự. Xâu kí tự được kết thúc bởi kí tự '$' (có mã ASCII là 24h).

Ví dụ:

Xau1 DB 'Chào các bạn','$';

XuongDong DB 13,10,'$';xâu chứa các kí tự xuống dòng và về đầu dòng

Xau2 DB 36h,40h,'a','b', '$' ; xâu chứa cả mã ASCII và kí tự.

d. *Khai báo hằng*

Hằng có thể là kiểu số hoặc kiểu kí tự. Cú pháp khai báo hằng như sau:

Tên hằng EQU Giá trị của hằng

Ví dụ:

CR EQU 0Dh ;

Có thể sử dụng hằng để khai báo:

Ví dụ:

Chao EQU 'Chào bạn'

LoiChao DB Chao,'$'

2.1.2 Khung của chương trình Hợp ngữ

Để thực hiện một chương trình dạng mã máy, hệ điều hành cấp phát một số vùng nhớ dành cho chương trình để chứa các mã lệnh, dữ liệu, và ngăn xếp. Phần này trình bày về các giả lệnh điều khiển đoạn dùng cho chương trình, khung của chương trình dạng .COM, khung của chương trình dạng .EXE, và cuối cùng là một số chương trình ví dụ đơn giản.

a. Các giả lệnh điều khiển đoạn (segment directives)

1. Lệnh: MODEL

Chức năng: Khai báo mô hình bộ nhớ cho một module Assembler. Lệnh này thường được đặt sau giả lệnh về khai báo loại CPU (CPU family) và được đặt trước tất cả các giả lệnh điều khiển đoạn khác.

Cú pháp: **.MODEL <Kiểu kích thước bộ nhớ>**

Trong đó kiểu kích thước bộ nhớ là một trong các kiểu sau:

Kiểu kích thước	Mô tả
Tiny (hẹp)	Mã lệnh và dữ liệu được gói vào cùng một đoạn. Kiểu này thường được dùng trong chương trình. COM

Small (nhỏ)	Mã lệnh được gói vào trong một đoạn. Dữ liệu nằm trong một đoạn khác
Medium (trung bình)	Mã lệnh được gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn.
Compact (nén)	Mã lệnh không gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn.
Large (lớn)	Mã lệnh không gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn. Không có mảng dữ liệu được khai báo nào lớn hơn 64KB
Huge (rất lớn)	Mã lệnh không gói vào trong một đoạn. Dữ liệu không gói gọn trong một đoạn. Các mảng dữ liệu được khai báo có thể lớn hơn 64KB.

Ví dụ: .MODEL Small

2. Lệnh: STACK

Chức năng: Khai báo kích thước đoạn ngăn xếp dùng trong chương trình. Đoạn ngăn xếp là một vùng nhớ để lưu các trạng thái hoạt động của chương trình khi có chương trình con.

Cú pháp: **.STACK <Kích thước ngăn xếp>**

Trong đó kích thước ngăn xếp là số byte dành cho ngăn xếp. Nếu không khai báo kích thước ngăn xếp thì chương trình sẽ tự động gán cho giá trị là 1024 (1 KB). Số này là khá lớn, thông thường khoảng 256 byte hay 100h là đủ.

Ví dụ: .STACK 100h

3. Lệnh: DATA

Chức năng: Khai báo đoạn dữ liệu cho chương trình. Đoạn dữ liệu chứa toàn bộ các khai báo hằng, biến của chương trình.

Cú pháp: **.DATA**

Ví dụ: .DATA

```
        CR EQU 0Dh
        LF  EQU 0Ah
   LoiChao   DB 'Chào các bạn','$'
```

4. Lệnh: CODE

Chức năng: Khai báo đoạn mã lệnh chương trình. Đoạn mã chứa các dòng lệnh của chương trình.

Cú pháp: **.CODE**

Ví dụ: .CODE

```
        Mov AH,09
        Mov Dx, Offset LoiChao
      ….
```

b. Khung của chương trình Hợp ngữ để dịch ra dạng .EXE

Dưới đây là khung của một chương trình hợp ngữ mà sau khi được dịch (compiled) và hợp dịch (linked) thì sẽ thành một file thực hiện được dạng .EXE.

Sau khi sử dụng các giả lệnh điều khiển đoạn để khai báo mô hình bộ nhớ, kích thước ngăn xếp, đoạn dữ liệu và bắt đầu đoạn mã lệnh. Tất nhiên người lập trình có thể thay đổi mô hình bộ nhớ (có thể không phải là *Small*) hay kích thước ngăn xếp (một số khác, không phải là 100h) cho phù hợp với mục đích viết chương trình.

Ta dùng nhãn *Start* và *End Start* để đánh dấu điểm bắt đầu và kết thúc đoạn mã lệnh dùng cho chương trình (tất nhiên người lập trình có thể dùng tên nhãn khác để đánh dấu, không bắt buộc phải dùng nhãn *Start*)

```
.MODEL Small
.STACK 100h
.DATA
  ; Các khai báo hằng, biến ở đây
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX
     ; Các lệnh của chương trình chính được viết ở đây
        Mov AH,4Ch
        Int 21h
End Start
; Các chương trình con (nếu có) sẽ được viết ở đây.
```

Hai lệnh:

```
Mov AX,@Data
Mov DS,AX
```

Làm nhiệm vụ cho con trỏ DS trỏ tới đoạn chứa dữ liệu khai báo (Data). Hằng số @Data là tên của đoạn dữ liệu (thực chất hằng số này mang giá trị là địa chỉ của đoạn bộ nhớ cấp phát cho chương trình trong quá trình chạy chương trình). Mà DS không làm việc trực tiếp với hằng số (không thể chuyển giá trị hằng số trực tiếp vào các thanh ghi đoạn), nên thanh ghi AX là biến trung gian để đưa giá trị @Data vào DS.

Hai lệnh cuối của chương trình:

```
Mov AH,4Ch
Int 21h
```

Làm nhiệm vụ kết thúc chương trình .EXE và trả lại quyền điều khiển cho hệ điều hành DOS. Nhắc lại rằng không giống như các hệ điều hành Windows 9x, 2K,XP là các hệ điều hành đa nhiệm. Hệ điều hành DOS là hệ điều hành đơn nhiệm (Single-task). Nghĩa là, tại một thời điểm, chỉ có một chương trình chiếm quyền điều khiển và tài nguyên của hệ thống.

Ví dụ về một chương trình dạng .EXE đơn giản, chương trình Hello World. Chương trình thực hiện việc in ra màn hình một lời chào "Hello World"

```
; Chương trình này in ra màn hình lời chào Hello World
```

```
.MODEL Small
.STACK 100h
.DATA
    Msg db 'Hello World','$'
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX  ; cho DS trỏ đến đoạn Data
        Mov AH,09h ; Hàm 09, in ra 1 xâu kí tự
        Mov DX,Offset Msg ; Dx chứa địa chỉ offset của xâu
        Int 21h ; thực hiện chức năng in xâu
        Mov AH,4Ch ; Trở về và trả quyền điều khiển cho DOS
        Int 21h
End Start
```

c. Khung của chương trình Hợp ngữ để dịch ra dạng .COM

Chương trình .COM ngắn gọn và đơn giản hơn so với các chương trình .EXE. Tất cả các đoạn ngăn xếp, dữ liệu, đoạn mã được gộp vào cùng một đoạn là đoạn mã. Nghĩa là, chương trình .COM được gói gọn trong một đoạn (việc dịch và thực hiện đối với chương trình .COM sẽ nhanh hơn các chương trình .EXE). Với các ứng dụng nhỏ mà mã lệnh và dữ liệu không vượt quá 64KB, ta có thể ghép luôn các đoạn ngăn xếp, dữ liệu và mã lệnh vào cùng với đoạn mã để tạo ra file dạng COM.

Để dịch được ra file dạng .COM, chương trình nguồn phải tuân thủ theo khung dưới đây

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
     ; Các khai báo hằng, biến ở đây
Start:
     ; Các lệnh của chương trình chính được viết ở đây
         Int 20h
     ; Các chương trình con (nếu có) sẽ được viết ở đây.
End Start
```

Khai báo mô hình kích thước sử dụng bộ nhớ luôn là Tiny. Ngoài ra, trong khung này không có lời khai báo đoạn ngăn xếp và đoạn dữ liệu. Lệnh đầu tiên trong đoạn mã là giả lệnh ORG 100h, dùng để gán địa chỉ bắt đầu cho chương trình tại 100h trong đoạn mã. Vùng dung lượng 256 byte đầu tiên được sử dụng cho đoạn mào đầu chương trình (Prefix Segment Program).

Kế tiếp, người ta dung lệnh JMP để nhảy qua phần bộ nhớ được dùng cho khai báo.

Ví dụ về một chương trình .COM đơn giản, chương trình Hello World.

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
         Msg db 'Hello World', '$'
```

```
Start:
        Mov AH,09h ; Hàm 09, in ra 1 xâu kí tự
        Mov DX,Offset Msg ; Dx chứa địa chỉ offset của xâu
        Int 21h ; thực hiện chức năng in xâu
    Int 20h ; kết thúc chương trình, trở về DOS
End Start
```

2.1.3 Tạo, dịch, hợp dịch và thực hiện chương trình Hợp ngữ

Phần này trình bày về các bước để tạo, cách dịch, hợp dịch và thực hiện một chương trình hợp ngữ. Dưới đây là các bước phải được thực hiện tuần tự. Nghĩa là, bước thứ i không thể được thực hiện nếu bước trước nó (bước i-1) chưa được thực hiện thành công.

Bước 1: Soạn chương trình nguồn

Dùng bất kỳ trình soạn thảo nào như Nodepad, Turbo Pascal Editor, Turbo C Editor…để tạo ra file văn bản chương trình. File chương trình phải có phần mở rộng là .ASM.

Bước 2: Dịch

Dùng một trong các chương trình dịch: MASM (Macro Assembler) hoặc TASM (Turbo Assembler) để dịch file .ASM thành file .OBJ. Nếu bước này có lỗi thì ta phải quay lại bước .

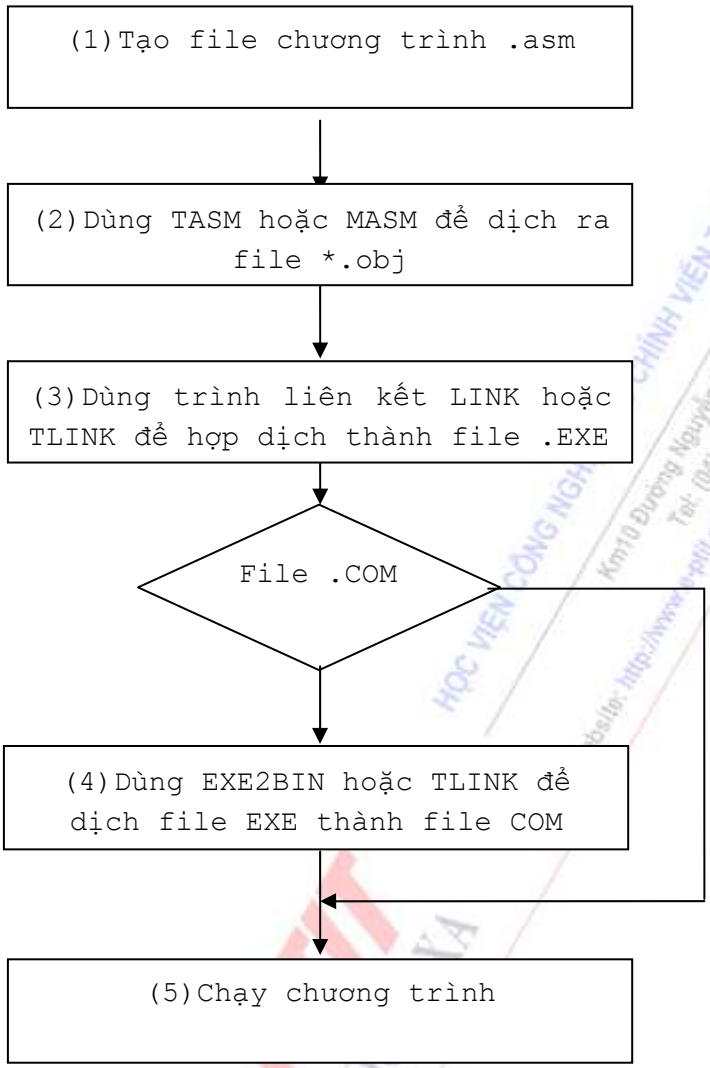
Bước 3: Hợp dịch

Dùng chương trình LINK hoặc TLINK để liên kết một hay nhiều file OBJ lại để thành một file .EXE, file có thể chạy được. Đối với chương trình có cấu trúc là file EXE thì bỏ qua bước

Bước 4: Tạo file .COM

Dùng chương trình EXE2BIN để dịch .EXE thành file .COM.

Bước 5. Thực hiện chương trình vừa tạo.



2.2 CÁC CẤU TRÚC LẬP TRÌNH CƠ BẢN TRONG CHƯƠNG TRÌNH HỢP NGỮ

Phần này trình bày về các cấu trúc lập trình cơ bản được sử dụng trong việc lập trình nói chung và lập trình hợp ngữ nói riêng. Các cấu trúc này thường được sử dụng để điều khiển một lệnh hoặc một khối lệnh. Đó là:

- Cấu trúc tuần tự
- Cấu trúc điều kiện IF-THEN
- Cấu trúc điều kiện rẽ nhánh IF-THEN-ELSE
- Cấu trúc CASE
- Cấu trúc lặp xác định FOR-DO
- Cấu trúc lặp WHILE-DO
- Cấu trúc lặp REPEAT-UNTIL

2.2.1 Cấu trúc tuần tự

Cấu trúc tuần tự có mặt hầu hết tất cả các ngôn ngữ lập trình. Đây là cấu tríc thông dụng, đơn giản nhất, trong đó các lệnh được sắp xếp kế tiếp nhau hết lệnh này đến lệnh khác. Trong quá trình thực hiện chương trình các lệnh tuần tự được xử lý theo thứ tự của chúng. Bắt đầu từ lệnh đầu tiên cho đến khi gặp lệnh cuối cùng của cấu trúc thì công việc cũng được hoàn tất.

Cấu trúc có dạng như sau:

```
lệnh 1
lệnh 2
lệnh 3
….
lệnh n
```

Ví dụ:

Cần tính tổng nội dung các ô nhớ có địa chỉ FFFAh, FFFBh, FFFCh, FFFDh rồi lưu kết quả vào thanh ghi AX.

Để thực hiện việc này ta có thể sử dụng đoạn chương trình sau:

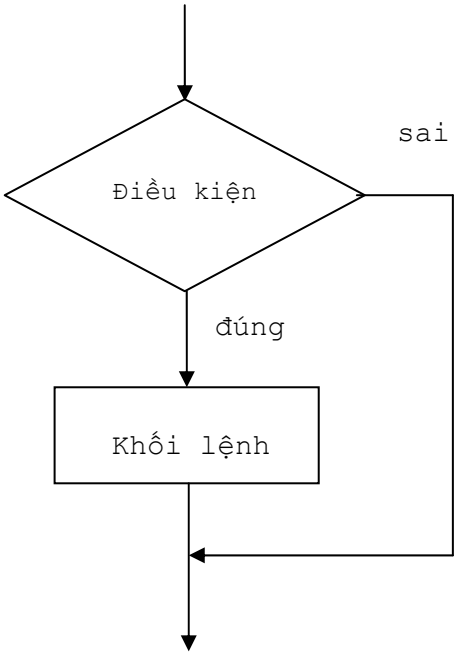
```
Mov BX,FFFAh ; BX trỏ đến FFFAh
Xor AX,AX        ; Tổng =0
Add AL,[BX] ;cộng nội dung ô nhớ có địa chỉ FFFAh vào AX
Inc BX     ;  BX trỏ đến FFFBh
Add AL,[BX]; cộng nội dung ô nhớ có địa chỉ FFFBh vào AX
Inc BX     ;  BX trỏ đến FFFCh
Add AL,[BX]; cộng nội dung ô nhớ có địa chỉ FFFCh vào AX
Inc BX     ;  BX trỏ đến FFFDh
Add AL,[BX]; cộng nội dung ô nhớ có địa chỉ FFFDh vào AX

Xong:         ; ra khỏi cấu trúc
```

2.2.2 Cấu trúc IF… THEN

Đây là cấu trúc điều kiện (conditional statement) mà khối lệnh được thực hiện nếu nó thỏa mãn điều kiện.

Cú pháp: IF <điều kiện> THEN <Khối lệnh>

Cấu trúc này có thể được minh họa bằng sơ đồ khối sau đây:



Khối lệnh có thể gồm 1 hoặc nhiều lệnh. Trong hợp ngữ, để cài đặt cấu trúc IF…THEN người ta thường sử dụng lệnh CMP (so sánh) và đi kèm theo sau là một lệnh nhảy có điều kiện.

Ví dụ:

Viết đoạn chương trình kiểm tra nếu thanh ghi AX>BX thì sẽ tính hiệu AX-BX và lưu kết quả vào thanh ghi AX.

Dưới đây là đoạn chương trình thực hiện công việc đó:

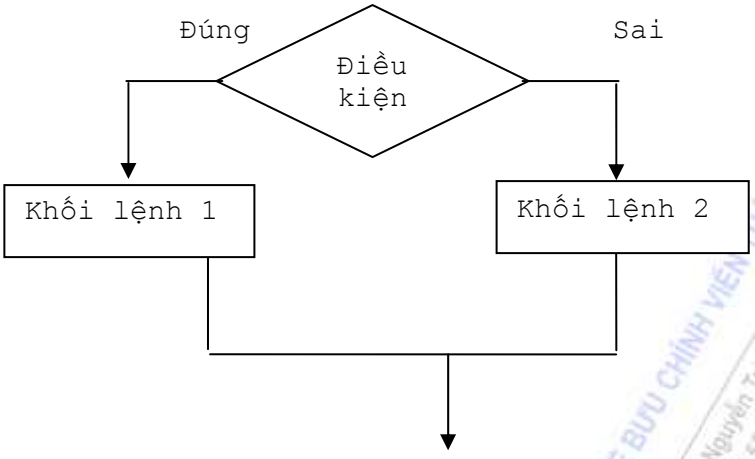
```
        Cmp AX,BX    ; so sánh AX và BX
        Jb Ketthuc   ; nếu AX<BX nhảy đến nhãn Ketthuc
        Sub AX,BX    ;  AX=AX-BX
 Ketthuc:
```

2.2.3 Cấu trúc IF… THEN…ELSE

Đây là dạng phân nhánh của cấu trúc có điều kiện.

Cú pháp: IF <điều kiện> THEN <Khối lệnh 1> ELSE <Khối lệnh 2>

Nếu điều kiện được thỏa mãn thì khối lệnh thứ nhất được thực hiện. Ngược lại, nếu điều kiện là sai thì khối lệnh thứ hai sẽ được thực hiện. Trong mọi trường hợp, một và chỉ một trong hai khối lệnh được thực hiện.



Hình: cấu trúc IF…THEN…ELSE

Trong cài đặt của cấu trúc IF…THEN…ELSE dạng hợp ngữ, thông thường người ta sử dụng một lệnh CMP, một lệnh nhảy có điều kiện và một lệnh nhảy không điều kiện.

Ví dụ: Cho hai số được lưu vào thanh ghi AX và BX, tìm số lớn nhất và lưu kết quả vào thanh ghi DX.

Dưới đây là đoạn chương trình thực hiện công việc đó.

```
        Cmp AX,BX    ; so sánh AX và BX
        JG AXLonHon  ; nếu AX>BX nhảy đến nhãn AXLonhon
        Mov DX,BX    ;  BX>=AX nên DX=AX
       Jmp Ketthuc  ; nhảy đến nhãn Ketthuc
 AXLonHon:
      Mov DX,AX    ;AX>BX nên DX=AX
 Ketthuc:
```

2.2.4 Cấu trúc CASE

Cấu trúc CASE là cấu trúc lựa chọn để thực hiện một khối lệnh giữa nhiều khối lệnh khác.

Cú pháp:

CASE <biểu thức>

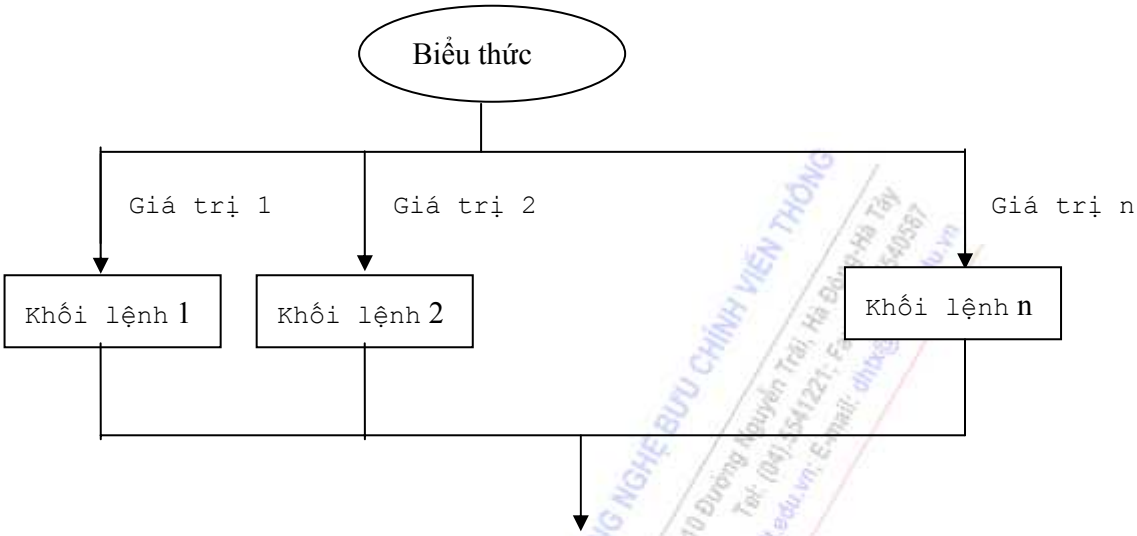
Giá trị 1: Khối lệnh 1

Giá trị 2: Khối lệnh 2

……

Giá trị n: Khối lệnh n

END CASE



Để thực hiện cấu trúc CASE trong chương trình Hợp ngữ, ta phai kết hợp các lệnh CMP, lệnh nhảy có điều kiện, và lệnh nhảy không điều kiện.

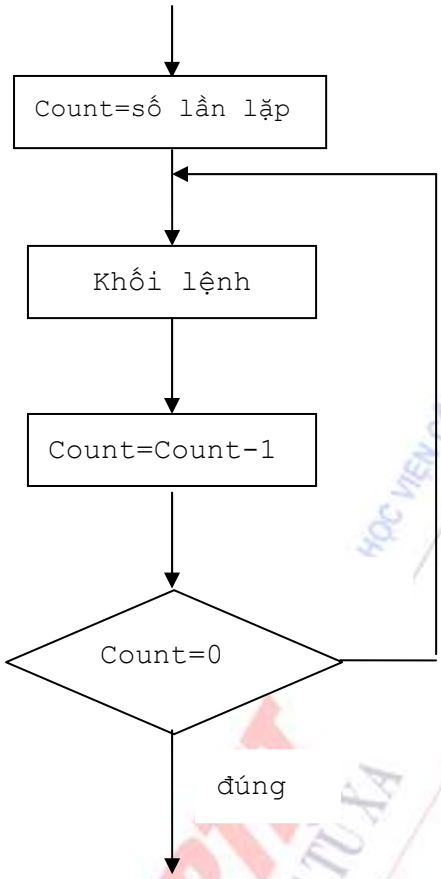
Ví dụ:

```
Am db 'Nhỏ hơn 0','$'
Duong db 'Lớn hơn 0','$'
Khong db 'Bằng không','$'
…
        Sub AX,BX
        Sub AX,CX
        Cmp AX,0
        Je Khong0
        Jb Am0
        Ja Duong0
Khong0:
        Mov AH,9
        Mov DX,offset Khong
        Int 21h
        Jmp Ketthuc
Am0:
        Mov AH,9
        Mov DX,offset Am0
        Int 21h
        Jmp Ketthuc
Duong0:
        Mov AH,9
        Mov DX,offset Duong0
        Int 21h
Ketthuc:
```

Tính hiệu AX-BX-CX và thông báo kết quả là âm, dương hay bằng không.

2.2.5 Cấu trúc lặp FOR-DO

Đây là vòng lặp với số lần lặp đã biết trước. Cú pháp như sau:



FOR Count (=Số lần lặp) DO Khối lệnh

Khối lệnh sẽ được thực hiện Count lần. Để cài đặt cấu trúc này trong hợp ngữ người ta dùng thanh ghi CX để chứa Count và kết hợp với lệnh LOOP để duy trì vòng lặp. Mỗi lần lặp xong thì CX sẽ tự động giảm đi 1.

Ví dụ:

Tính tổng S=1+2+3+….+100

Lưu kết quả vào thanh ghi AX.

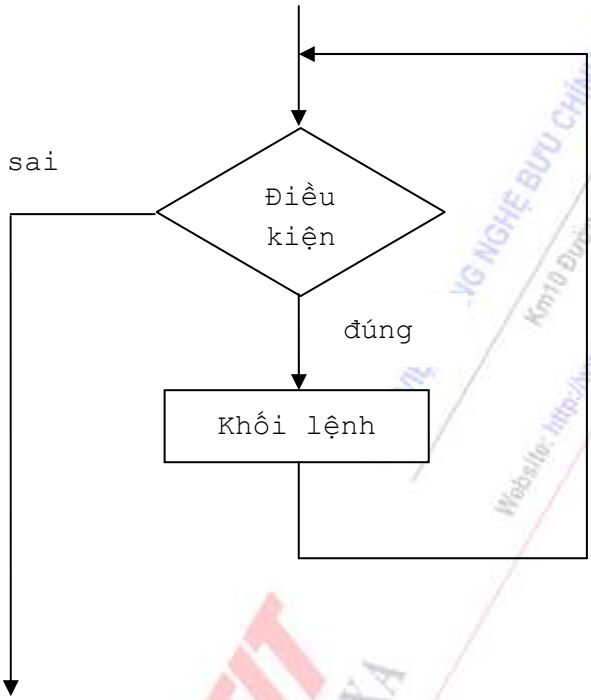
Mỗi lần thực hiện khối lệnh ta sẽ cộng vào AX một số. Lần thực hiện thứ i thì 100-i+1 sẽ được cộng vào AX. Như vậy số lần lặp sẽ là 100. Đoạn chương trình được viết như sau:

```
        Mov CX,100 ; khởi tạo số lần lặp
        Xor AX,AX  ; AX=0 để chứa tổng
Cong:
        Add AX,CX ; Cộng  CX vào AX
        Loop Cong ; Lặp cho đến khi CX=0
                  ; ra khỏi vòng lặp, AX chứa tổng
```

2.2.6 Cấu trúc lặp WHILE-DO

Cú pháp: WHILE điều kiện DO Khối lệnh

Trong cấu trúc lặp WHILE…DO điều kiện được kiểm tra trước khi thực hiện khối lệnh. Nếu điều kiện đúng thì khối lệnh được thực hiện, còn điều kiện sai thì vòng lặp sẽ dừng. Số lần thực hiện khối lệnh chưa được biết trước.



Để cài đặt cấu trúc này trong chương trình hợp ngữ, người ta thường dùng lệnh CMP để kiểm tra điều kiện và kết hợp mới một lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

Ví dụ:

Nhập vào một số nguyên lớn hơn 0 và bé hơn 9 từ bàn phím. Kiểm tra xem có nhập đúng không?. Nếu nhập sai thì yêu cầu phải nhập lại.

Lời giải

Ta biết rằng số 0 có mã ASCII là 30h và số 9 có mã ASCII là 39h. Đoạn chương trình sẽ kiểm tra nếu kí tự gõ vào có mã ASCII bé hơn 30h hoặc lớn hơn 39h thì sẽ yêu cầu người dùng nhập lại kí tự khác.

Dưới đây là đoạn chương trình.

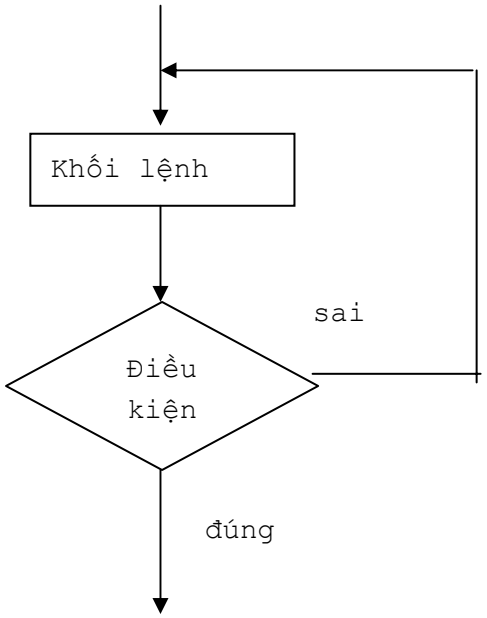
```
Mov AH,01 ; nhập vào 1 kí tự
Nhap:
      Int 21h
      Cmp AL,30h ; kí tự nhập vào <'0'
      Jb Nhap    ; nhập lại
      Cmp AL,39h ;  kí tự nhập vào >'9'
      Ja Nhap    ; nhập lại
```

```
Xong: Sub AL,30h ; Kí tự đã hợp lệ, đổi ra số
```

2.2.7 Cấu trúc lặp REPEAT-UNTIL

Cú pháp: REPEAT Khối lệnh UNTIL Điều kiện

Trong cấu trúc này, khối lệnh được thực hiện ít nhất một lần, sau đó mới kiểm tra điều kiện. Khối lệnh sẽ được lặp đi lặp lại cho đến khi điều kiện thỏa mãn.



Để cài đặt cấu trúc này trong hợp ngữ, người ta thường dùng một lệnh CMP đi kèm với với một lệnh nhảy có điều kiện.

Ví dụ:

Tìm n nhỏ nhất sao cho 1+2+3 +…+n >10000, lưu kết quả (số n) vào BX

Lời giải:

Ta cộng vào tổng (chứa trong AX) các số 1,2,… mỗi lần tính tổng ta đều so sánh tổng đó với 10000, ta cộng cho đến khi AX>10000 thì ta dừng, số hạng cuối cùng của tổng chính là số n cần tìm.

Đoạn chương trình được viết như sau:

```
      Mov CX,1   ; CX=1
      Xor AX,AX ; AX=Tong=0
Cong:
      Add AX,CX ; Cộng CX vào AX
      Cmp AX ,10000 ; Tổng đã > 10000 chưa?
      Ja Xong       ; đã lớn hơn, xong
      Inc CX    ; Tăng CX lên 1
      Jmp Cong  ; Tiếp tục cộng
Xong:
 Mov BX,CX ; lưu n vào BX
```

2.3 CHƯƠNG TRÌNH CON VÀ MACRO

2.3.1 Chương trình con: cơ chế làm việc và cấu trúc

a. Khái niệm:

Chương trình con có mặt hầu hết các ngôn ngữ lập trình. Chương trình con rất có ý nghĩa trong lập trình có cấu trúc. Nó làm cho chương trình trở lên sáng sủa, dễ bảo trì hơn. Bên cạnh đó, nó còn có một ý nghĩa khác: một chương trình con được viết *một* lần nhưng được sử dụng (gọi đến) *nhiều* lần.

Một cách định nghĩa đơn giản: chương trình con là một nhóm các lệnh được gộp lại phục vụ cho việc sử dụng nhiều lần thông qua tên và các tham số của chương trình con đó .

Ví dụ: Một bài toán yêu cầu ta tính tổng của 3 số hạng được nhập từ bàn phím. Thay vì phải viết 3 đoạn chương trình (khá giống nhau) để lần lượt nhập từng số thì ta viết 1 chương trình con nhập vào 1 số rồi gọi nó 3 lần.

b. Cơ chế làm việc của chương trình con

Giả sử có một chương trình (chính) đang thực hiện như sau:

Địa chỉ	Lệnh
1F00	Mov AX,1
1F02	Mov BX,2FFF
1F04	Mov CX,2
1F06	Call TinhTong
1F09	Sub AX,BX
1F0B	Add AX,CX
1F0D	SHR AX,1
1F0F	Jmp Done

Chương trình con TinhTong được lưu ở 1 vùng nhớ khác

Địa chỉ	Lệnh
FFD0	Mov DL,[BX]
FFD2	Add AL,[BX]
FFD4	Inc BX
FFD6	Add AL,[BX]
FFD8	Ret

Khi thực hiện đến lệnh Call TinhTong ở địa chỉ 1F06h, bộ xử lý sẽ thực hiện như sau:

- Lưu địa chỉ của lệnh kế tiếp 1F09h vào ngăn xếp.
- Nạp địa chỉ của lệnh đầu tiên của chương trình con FFD0h vào IP
- Các lệnh của chương trình con được thực hiện cho đến khi gặp lệnh RET, chương trình con kết thúc và trả lại quyền điều khiển cho chương trình chính.
- Địa chỉ 1F09h được lấy ra từ ngăn xếp rồi đặt vào IP. Lệnh tại địa chỉ 1F09h (Sub AX,BX) của chương trình chính sẽ được thực hiện.

c. Cấu trúc của chương trình con

Cấu trúc của một chương trình con có dạng như sau:

<Tên chương trình con> PROC NEAR/FAR

Các lệnh của chương trình con được viết ở đây

RET

<Tên chương trình con> ENDP

Giải thích:

- Lệnh điều khiển PROC được sử dụng để khởi động chương trình con. Nhãn đứng trước toán tử PROC là tên của thủ tục. Sau toán tử PROC có lệnh điều khiển NEAR hoặc FAR để báo cho lệnh RET lấy địa chỉ quay về chương trình của nó trong ngăn xếp.
- Nếu là NEAR thì chương trình con được gọi thì địa chỉ OFFSET (16 bít) được lấy từ ngăn xếp để gán cho thanh ghi IP. Nghĩa là, trong trường hợp này thì chương trình con và chương trình gọi nó ở trên cùng một đoạn (segment)
- Nếu là FAR thì chỉ lấy địa chỉ SEGMENT và OFFSET trong ngăn xếp được lấy ra để gán cho thanh ghi CS và IP. Nghĩa là, trong trường hợp này thì chương trình con và chương trình gọi nó ở hai đoạn khác nhau.

Để lệnh RET có đầy đủ thông tin để biết là phải nạp cả CS:IP hay chỉ nạp IP, có hai cách sau:

- Thay vì sử dụng RET ta sử dụng lệnh RETN (near return) hoặc RETF (far return).
- Dùng hai lệnh điều khiển PROC và ENDP. Lệnh ENDP sẽ đánh dấu kết thúc chương trình con. Nếu đi sau PROC là NEAR thì tất cả các lệnh RET trong chương trình con nằm trong PROC … ENDP đều là RETN nằm chung một đoạn với chương trình gọi đến chương trình con này. Nếu đi sau PROC là FAR thì tất cả các lệnh RET trong chương trình con nằm trong PROC … ENDP đều là RETF (nằm chung một đoạn với chương trình gọi đến chương trình con này).
- Nếu cả chương trình chính và các chương trình con có kích thước nhỏ (tất cả mã lệnh không vượt quá 64KB) thì ta nên sử dụng chúng như là các thủ tục NEAR. Vì như thế chương trình sẽ được thực hiện nhanh hơn.
- Khi sử dụng mô hình bộ nhớ, các giá trị NEAR là FAR cũng được gán ngầm định theo mô hình kích thước bộ nhớ. Chẳng hạn: nếu ta dùng .MODEL Tiny hoặc .MODEL Compact thì chương trình con sẽ được tự động xác định là NEAR. Còn nếu ta dùng

.MODEL Medium, .MODEL Large hoặc .MODEL Huge thì chương trình con sẽ được tự động xác định là FAR.

- Trong trường hợp ta không có khai báo NEAR hoặc FAR sau lệnh PROC thì ngầm định là NEAR.

2.3.2 Truyền tham số

Đây là phần rất quan trọng khi chương trình được thiết kế thành các chương trình con. Chúng được "giao tiếp" với nhau một cách trong suôt, lô gic trong quá trình thực hiện các chức năng của mình để tăng thêm tính tái sử dụng- một tính chất quan trọng của chương trình con. Dữ liệu phải được trao đổi từ chương trình gọi và chương trình con được gọi. Trong các ngôn ngữ bậc cao khác, cấu trúc chương trình con cho phép người lập trình khai báo danh sách tham số. Tuy nhiên, như ta thấy cấu trúc của một chương trình con hợp ngữ ta không thấy đi kèm với một danh sách tham số. Dưới đây ta sẽ xem xét tất cả các các vấn đề liên quan đến việc truyền tham số.

a. Truyền giá trị tham số từ chương trình gọi sang chương trình được gọi

- Truyền tham số thông qua các thanh ghi: đây là cách thức đơn giản và dễ thực hiện nhất, thường được sử dụng đối với các chương trình được viết thuần túy bằng hợp ngữ. Để thực hiện cách truyền tham số này ta chỉ cần đặt một giá trị nào đó vào thanh ghi ở chương trình gọi và sau đó chương trình con được gọi sẽ sử dụng giá trị ở thanh ghi này.
- Truyền tham số thông qua các biến toàn cục: các biến toàn cục được khai báo trong chương trình chính có tác dụng trong toàn bộ chương trình (cả chương trình chính và các chương trình con). Vì vậy ta có thể dùng nó để truyền giá trị giữa chương trình chính và các chương trình con. Cách này khá phổ biến khi ta viết chương trình thuần túy bằng hợp ngữ hoặc phát triển chương trình hỗn hợp bằng hợp ngữ và các ngôn ngữ bậc cao khác.
- Truyền tham số thông qua ngăn xếp: đây là phương pháp khá phức tạp. Tuy nhiên, cách này được sử dụng rất nhiều khi ta viết các module bằng hợp ngữ và các ngôn ngữ bậc cao khác rồi cho chúng liên kết (link) với nhau trong quá trình thực hiện. Cách này sẽ được đề cập chi tiết ở phần sau: *kết nối chương trình hợp ngữ với các chương trình ngôn ngữ bậc cao*.

b. Truyền giá trị tham số từ chương trình được gọi lên chương trình gọi

- Truyền tham số từ chương trình được gọi (chương trình con) lên chương trình gọi (chương trình chính) cũng theo ba cách: thông qua các thanh ghi, biến toàn cục và ngăn xếp. Trong trường hợp liên kết với ngôn ngữ bậc cao thì chương trình con được gọi (được viết bằng hợp ngữ) có thể chuyển giá trị lên chương trình gọi (được viết bằng ngôn ngữ bậc cao) bằng giá trị trả về (returned value). Để làm được điều này trong hợp ngữ thì giá trị trả về của chương trình con được gọi phải tuân thủ các qui cách sau:

  + Nếu giá trị trả về (tên hàm mang giá trị trả về) là 8 bít hoặc 16 bít thì giá trị đó phải được đặt vào thanh ghi AX của hàm trước khi quay về chương trình gọi nó.

  + Nếu giá trị trả về (tên hàm mang giá trị trả về) là 32 bít thì giá trị đó phải được đặt vào thanh ghi DX:AX của hàm trước khi quay về chương trình gọi nó.

- Lưu ý rằng số lượng các thanh ghi của máy tính là có hạn, nên ta không nên dùng quá nhiều thanh ghi cho việc chuyển giao các tham số.

c. Vấn đề bảo vệ các thanh ghi

Khác với lập trình với các ngôn ngữ bậc cao, khi lập trình hợp ngữ người lập trình hợp ngữ còn phải để ý đến việc bảo vệ các thanh ghi trong quá trình gọi các chương trình con. Ở các ngôn ngữ bậc cao, chương trình con không làm thay đổi giá trị của các biến của chương trình chính trừ khi ta chủ tâm làm việc đó. Trong các chương trình được viết bằng hợp ngữ thì ngược lại là rất hay xảy ra trường hợp là các giá trị của các biến trong chương trình chính được nạp vào các thanh ghi mà trong khi đó chương trình con cũng cần các thanh ghi này để thực hiện một công việc nào đó. Và như vậy thì chương trình con khi sử dụng thanh ghi có thể sẽ xóa giá trị trong thanh ghi mà chương trình chính đã đặt vào đó để sử về sau. Do vậy các giá trị đã được lưu vào trong thanh ghi cần phải được bảo vệ khi cần thiết. Có hai cách người ta hay dùng là:

- Sử dụng các lệnh PUSH và POP: Khi bắt đâu một chương trình con, ta nên tiến hành lưu các giá trị của các thanh ghi mà chương trình con sẽ dùng đến vào ngăn xếp nhờ lệnh PUSH và trước khi ra khỏi chương trình con ta phải phục hồi lại các giá trị của các thanh ghi đó từ ngăn xếp nhờ lệnh POP.
- Sử dụng theo một qui ước nhất quán (code convension): qui định sử dụng một số thanh ghi sử dụng cho chương trình chính và tất cả các chương trình con không được sử dụng đến các thanh ghi đó.

2.3.3 Chương trình gồm nhiều module

Đó là chương trình gồm nhiều file, thích hợp cho các chương trình lớn và phức tạp. Chúng được dịch một cách độc lập nhưng được hợp dịch (link) với nhau khi chạy. Sau đây là những ưu điểm của việc viết chương trình gồm nhiều file:

- Cho phép nhiều người lập trình cùng tham gia phát triển một chương trình lớn.
- Dễ dàng cho việc sửa lỗi, khi dịch module nào phát hiện ra lỗi thì chỉ cần sửa và dịch lại module đó.
- Mỗi module thường giải quyết một vấn đề ngắn gọn nên dễ tìm sai sót.

Để chia xẻ các biến toàn cục hoặc các chương trình con được sử dụng chung giữa các module người ta sử dụng các lệnh điều khiển PUBLIC, EXTRN và GLOBAL.

a. Lệnh điều khiển PUBLIC

Chức năng: Lệnh điều khiển PUBLIC chỉ cho chương trình dịch hợp ngữ biết nhãn nào nằm trong module này được phép sử dụng ở các module khác.

Cú pháp: PUBLIC tên nhãn

Khai báo nhãn

Trong đó tên nhãn có thể là:

- Tên chương trình con
- Tên biến
- Tên hằng (theo sau bởi lệnh EQU)

Ví dụ:

Nhãn là tên biến nhớ

```
.DATA
PUBLIC  gTong, gSoHang, gMang, gMangLength
gTong dd ?
gSoHang dw ?
gMangLength EQU 100
gMang db gMangLength DUP(?)
Nhãn là tên của chương trình con
.CODE
PUBLIC  gTinhTong, gTimMax
gTinhTong PROC NEAR
….
gTinhTong ENDP
    ;--------------------------------
gTimMax PROC FAR
….

gTimMax ENDP
   ;-----------------------------------
```

Chú ý: chương trình dịch hợp ngữ không phân biệt chữ hoa hay thường trong các nhãn. Tất cả các chữ đều hiểu như chữ hoa. Nếu muốn có sự phân biệt đó thì:

- dùng tùy chọn /ml khi dịch cho tất cả mọi ký hiệu
- dùng tùy chon /mx khi dịch cho các nhãn được khai báo PUBLIC, EXTRN, hoặc GLOBAL.

b. Lệnh điều khiển EXTRN

Chức năng: Lệnh điều khiển EXTRN báo cho chương trình dịch hợp ngữ biết nhãn nào đã được khai báo PUBLIC ở các module khác được sử dụng trong module này. Nói cách khác các nhãn đã được PUBLIC ở các module khác sẽ được sử dụng trong module này mà không cần khai báo lại nếu chúng được khai báo EXTRN.

Cú pháp: EXTRN tên nhãn: kiểu

Trong đó kiểu có các dạng như sau:

Kiểu	Giải thích
ABS	Giá trị tuyệt đối, dùng để khai báo các nhãn được xác định bởi EQU hoặc =
BYTE	Giá trị nhãn là 1 byte
WORD	Giá trị nhãn là 2 byte
DWORD	Giá trị nhãn là 4 byte
FWORD	Giá trị nhãn là 6 byte

QWORD	Giá trị nhãn là 8 byte
TBYTE	Giá trị nhãn là 10 byte
DATAPTR	Con trỏ NEAR hoặc FAR phụ thuộc vào MODEL của bộ nhớ
NEAR	Chỉ chương trình con dạng khai báo NEAR
FAR	Chỉ chương trình con dạng khai báo FAR
PROC	Xác định nhãn là thủ tục; còn NEAR hoặc FAR phụ thuộc vào lệnh điều khiển .MODEL
UNKNOWN	Cho nhãn không biết kích cỡ

Các kiểu dữ liệu theo sau nhãn được khai báo EXTRN phải xác định đúng, nếu không sẽ gây ra sai sót.

Ví dụ:

Ở module 1 có chương trình con được khai báo như sau:

```
PUBLIC TinhTong
 TinhTong PROC FAR
……
   Ret
 TinhTong ENDP
```

Ở module 2 sử dụng chương trình con TinhTong được khai báo trong module 1.

```
.CODE
EXTRN TinhTong: FAR
…
Call TinhTong
```

Để sử dụng EXTRN để khai báo cho chương trình dịch hợp ngữ biết những nhãn nào đã được khai báo PUBLIC ở phần trước được sử dụng trong module này.

```
.DATA
EXTRN gTong:DWORD, gSoHang:WORD, gMang:BYTE,gMangLength:ABS
EXTRN TinhTong: NEAR, TimMax: FAR
 ….
Call TinhTong
….
Call TimMax
…
```

c. Lệnh điều khiển GLOBAL

Lệnh GLOBAL được hỗ trợ bởi chương trình dịch TASM (Turbo Assembler) của hãng Borland. Lệnh điều khiển này còn có thể thay thế hai lệnh PUBLIC và EXTRN. Nếu ta khai báo GLOBAL cho các nhãn có kèm theo khai báo dạng nhãn thì GLOBAL trong trường hợp này sẽ

thay thế cho PUBLIC, còn khi khai báo nhãn đi sau GLOBAL mà chỉ xác định kiểu nhãn thì GLOBAL sẽ thay thế EXTRN.

Ví dụ:

```
.DATA
GLOBAL gSoHang:WORD, gMang:BYTE
Count DW ?
…
.CODE
GLOBAL TinhTong: NEAR, TimMax: FAR
TimMax PROC FAR
Call TinhTong
…
```

Các nhãn TinhTong, TimMax được khai báo do đó lệnh điều khiển GLOBAL đối với các nhãn này có ý nghĩa như PUBLIC, còn các nhãn gSoHang, gMang chỉ nêu kiểu mà không khai báo thì GLOBAL đối với chúng là EXTRN.

Một trường hợp vô cùng thuận lợi với việc sử dụng lệnh điều khiển GLOBAL là việc sử dụng GLOBAL trng file INCLUDE. Giả sử ta có một tập hợp các nhãn mà ta muốn sử dụng ở tất cả các module của chương trình gồm nhiều module. Ta có thể làm được như vậy nhờ việc gộp tất cả các nhãn vào file INCLUDE và sau đó đưa file này vào các module. Trong trường hợp này ta không thể sử dụng PUBLIC hoặc EXTRN vì EXTRN không thể làm việc được với các nhãn có xác định kích thước khai báo. Còn lệnh PUBLIC chỉ làm việc với các module trong đó các nhãn được khai báo mà không xác định kiểu. Do vậy, chỉ có GLOBAL là thỏ mãn cả hai điều kiện trên.

Ví dụ về một chương trình nằm trên hai file (hai module) khác nhau. Với:

- Module của chương trình chính là main.asm có chức năng xác định địa chỉ OFFSET của hai xâu kí tự, gọi chương trình con làm nhiệm vụ nối hai xâu kí tự đó lại và hiển thị xâu kết quả.
- Module chương trình con là KetNoi.ASM làm nhiệm vụ kết nối hai xâu và xếp vào vùng nhớ kết quả.

Dưới đây là chương trình chính:

```
.MODEL SMALL
.STACK 100h
.DATA
  Xau1 DB "Hello",0
  Xau2 DB "Mr Bin", 13,10,'$',0
  GLOBAL XauKQ:BYTE
  XauKQ DB 50 DUP (?)
.CODE
   EXTRN KetNoi:PROC
Start:
      Mov AX,@Data
      Mov DS,AX
      Mov AX,offset Xau1; AX chứa OFFSET của Xau1
```

```
            Mov BX,offset Xau2; BX chứa OFFSET của Xau2
            Call KetNoi  ; nối hai xâu
            Mov DX,Offset XauKQ ; In ra màn hình
            Mov AH,9
            Int 21h
            Mov AH,4Ch ; Trở về DOS
            Int 21h
      End Start
```

Module của chương trình con KetNoi

```
      .MODEL SMALL
      .STACK 100h
      .DATA
        GLOBAL XauKQ:BYTE
      .CODE
      PUBLIC KetNoi
      KetNoi PROC
      Cld
           Mov DI, SEG XauKQ ; ES:DI trỏ đến xâu kq
           Mov ES,DI
           Mov DI, OFFSET XauKQ
           Mov SI,AX; DS:SI trỏ đến Xau1
Lap1:
           Lodsb   ; lấy 1 kí tự đưa vào AL
           And AL,AL ; cho ZF=1
           Jz DoKetNoi
           Stosb  ; Lưu kí tự từ AL vào xâu
           Jmp Lap
DoKetNoi:
           Mov SI,BX; DS:SI; trỏ đến Xau2
Lap2:
           Lodsb  ; lấy kí tự đưa vào AL
           Stosb ; Cất kí tự vào XauKQ
           And AL,AL  ; cho ZF=1
           Jnz Lap2  ; giá trị khác 0 thì nhảy
           Ret   ; trở về chương trình chính
KetNoi ENDP
END
```

Để chạy hai module trên ta thực hiện theo các thao tác sau:

Dịch hai module một cách tách biệt

```
      TASM Main
      TASM KetNoi
```

Sau đó tiến hành hợp dịchTLINK Main+KetNoi ta sẽ có file Main.exe.

2.3.4 Liên kết thủ tục vào một thư viện

Trong quá trình lập trình, có thể một số thao tác sau ta cần thực hiện:

- Đưa một khối lệnh vào các nơi khác nhau hoặc các module nguồn
- Phân chia các nhãn gán (EQU) và MACRO giữa các phần khác nhau của một chương trình hoặc sử dụng lại chúng trong nhiều chương trình.
- Viết một chương trình dài, xong không muốn chia nhỏ ra nhiều module vì phải dịch từng module rồi liên kết chúng với nhau song chương trình quá to không thể chứa trong một file.

Để giải quyết các vấn đề trên, chương trình dịch của hợp ngữ có lệnh INCLUDE.

Giả sử ta muốn tạo một file INCLUDE chứa khối lệnh mà ta muốn các module khác thêm vào khi dịch.

Cú pháp:

INCLUDE tên file

(file chứa khối lệnh cần được đưa vào vị trí mà lệnh INCLUDE đang đứng)

Cơ chế: Khi chương trình hợp ngữ gặp lệnh INCLUDE thì sẽ tìm đến đường dẫn chứa file đã được xác định sau INCLUDE và đưa toàn bộ khối lệnh mà tệp này chứa xen vào vị trí mà lệnh INCLUDE đang đứng của module chương trình. Hay nói cách khác nội dung của tệp INCLUDE được đặt vào đúng vùng nhớ của chương trình mà lệnh INCLUDE đang được xác định.

Ví dụ:

Giả sử có một chương trình trong file A.ASM có nội dung như sau:

```
...
.CODE
Mov BX,10
Add AX,BX
INCLUDE  B.ASM
Sub AX,CX
```

File B.ASM có nội dung như sau:

```
Mov CX,3
Mov DX,4
```

Kết quả dịch chương trình A.ASM sẽ như sau:

```
.CODE
Mov BX,10
Add AX,BX
Mov CX,3
Mov DX,4
Sub AX,CX
```

Qua ví dụ trên ta thấy trong quá trình dịch chương trình A.ASM, khi đến dòng lệnh INCLUDE B.ASM

Thì chương trình dịch lấy tất cả các lệnh của B.ASM đặt vào vị trí mà lệnh INCLUDE đang đứng. ngoài ra các lệnh INCLUDE còn có tính chất lồng nhau với những mức khác nhau, có nghĩa là trong file INCLUDE có thể gọi 1 file INCLUDE khác.

Dưới đây là cơ chế mà chương trình dịch tìm các file INCLUDE:

- Nếu trong lệnh INCLUDE chỉ rõ tên ổ đĩa, đường dẫn, tên file thì chương trình dịch sẽ tìm theo sự xác định ở trên
- Nếu trong lệnh INCLUDE chỉ xác định tên file thì chương trình dịch tiến hành tìm file này ở thư mục hiện thời. Trường hợp không tìm thấy thì sẽ tìm file đó ở thư mục được chỉ ra trong câu lệnh:

  TASM –i <đường dẫn đến file >
- Nếu không tìm thấy file INCLUDE trong tất cả các trường hợp trên thì chương trình dịch sẽ báo không tìm thấy file INCLUDE.

2.3.5 Macro

Trước khi đi vào tìm hiểu có chế hoạt động, cách viết các macro ta hãy tìm hiểu một số lệnh thường được sử dụng để viết các macro.

a. Các lệnh lặp và điều khiển điều kiện khi dịch

Các lệnh lặp có chức năng là thực hiện một khối lệnh nhiều lần theo số lần lặp (do người lập trình đặt sẵn). Ta hãy xem xét một số lệnh lặp thường hay được sử dụng.

1. Lệnh: REPT

Cú pháp:

```
REPT <số lần lặp>
    Khối lệnh
ENDM   ; kết thúc lệnh lặp
```

Ví dụ:

```
REPT 100
    DB ?
ENDM
```

Khi tiến hành dịch nó sẽ thực hiện khai báo như sau:

```
DB ?
DB ?
 …
DB ?
```

; 100 lần khai báo

Trong trường hợp này giống như khai báo

```
DB 100 Dup(?)
```

Tuy nhiên, hai cách này cũng không hoàn toàn giống nhau.

Ví dụ 2:

```
REPT 100
    DB Count
   Count= Count+1
```

```
ENDM
```

Khi dịch ra sẽ được triển khai như sau:

```
DB 0
DB 1
…
DB 99
```

2.Lệnh: IRP

Chức năng: Lệnh này cho phép lặp một khối lệnh theo số lượng danh sách các tham số với sự thay đổi các giá trị của tham số trong khối lệnh.

Cú pháp:

```
IRP tên tham số <danh sách tham số>
    Khối lệnh
ENDM
```

Ví dụ:

```
IRP varX <0,2,4,6,8,10>
     DB varX
ENDM
```

Khi dịch khối lệnh trên sẽ được dịch như sau:

```
     DB 0
     DB 2
  …
     DB 10
```

3. Lệnh: IF và IFE

Chức năng: Lệnh điều khiển IF báo cho chương trình dịch hợp ngữ biết phải thực hiện việc dịch khối lệnh khi giá trị của biểu thức khác 0.

Cú pháp:

```
IF <biểu thức>
    Khối lệnh
ENDIF
```

Hoặc:

```
IF <biểu thức>
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Ví dụ:

```
IF is8086 ; nếu CPU là 8086 thì không cho phép lưu vào
    Mov AX,EFh ; một hằng số trực tiếp vào ngăn xếp
   Push AX
ELSE
    Push EFh ; nếu không phải thì có thể
ENDIF
```

Lệnh điều khiển IFE giống IF song việc dịch được tiến hành khi giá trị của biểu thức bằng 0

```
IFE 0
    Khối lệnh
ENDIF
```

Luôn dịch khối lệnh bên trong. IFE và ENDIF

4. Lệnh: IFDEF và IFNDEF

Chức năng: Lệnh điều khiển IFDEF báo cho chương trình dịch hợp ngữ biết phải thực hiện khối lệnh khi nhãn đã được khai báo.

Cú pháp:

```
IFDEF Nhãn
    Khối lệnh
ENDIF
```

Hoặc:

```
IF Nhãn
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Lệnh điều khiển IFNDEF giống lệnh trên nhưng với điều kiện ngược lại là chương trình dịch sẽ thực hiện khối lệnh nếu nhãn đã được khai báo.

Ví dụ:

Tránh việc khai báo hai lần giá trị khởi đầu của một biến

```
varX DB 1
….
IF varX
        Display "khai báo trùng hợp biến varX "
 Err
ELSE
 varX DB 10
ENDIF
```

5. Lệnh: IFB và IFNB

Chức năng: Các lệnh này được dùng để kiểm tra việc truyền tham số cho macro. Chúng báo cho chương trình dịch hợp ngữ biết phải thực hiện khối lệnh nếu tham số là rỗng.

Cú pháp:

```
IFB Tham số
    Khối lệnh
ENDIF
```

Hoặc:

```
IFB Tham số
    Khối lệnh 1
```

```
ELSE
    Khối lệnh 2
ENDIF
```

Lệnh điều khiển IFNB giống lệnh trên nhưng với điều kiện ngược lại là chương trình dịch sẽ thực hiện khối lệnh nếu thông số không phải là dấu trống.

6. Lệnh: IFDIF và IFIDN

Chức năng: Lệnh IFDIF báo cho chương trình dịch hợp ngữ biết phải thực hiện khối lệnh nếu hai tham số bằng nhau.

Cú pháp:

```
IFDIF Tham số1, Tham số 2
    Khối lệnh
ENDIF
```

Hoặc:

```
IFDIF Tham số1, Tham số 2
    Khối lệnh 1
ELSE
    Khối lệnh 2
ENDIF
```

Lệnh điều khiển IFIDN giống lệnh IFDIF nhưng với điều kiện ngược lại là chương trình dịch sẽ thực hiện khối lệnh nếu hai tham số phải là khác nhau.

Chú ý: Tên tham số của 2 lệnh trên có phân biệt chữ hoa, chữ thường (Case sensitive)

b. Cơ bản về Macro

Macro bao gồm một tên đại diện và một khối lệnh. Khi chương trình dịch hợp ngữ gặp tên này ở đâu trong chương trình thì khối lệnh sẽ được dịch và đặt khối mã lệnh vào vị trí mà chương trình gọi tên macro.

Về cơ chế thực hiện thì macro giống với file INCLUDE. Nghĩa là, mỗi lần chương trình dịch gặp tên của macro hay tên file INCLUDE thì toàn bộ khối lệnh được định nghĩa trong macro hay file INCLUDE sẽ được dịch và đặt vào vị trí lời gọi.

Tuy nhiên, giữa file INCLUDE và MACRO có một số điểm khác nhau sau đây. Nhìn chung, macro mạnh hơn file INCLUDE vì:

- Macro có thể truyền tham số.
- Macro có thể chứa các nhãn cục bộ.
- Việc dịch Macro diễn ra nhanh hơn vì macro là 1 phần của chương trình (không phải là file) nên không phải đọc từ đĩa ngoài.
- Macro còn sử dụng một số lệnh điều khiển điều kiện hoặc lặp khi thực hiện.

c. Khai báo và sử dụng Macro

Trước khi sử dụng , ta phải khai báo macro theo cú pháp sau:

Tên macro MACRO các tham số

Thân macro

ENDM

Ví dụ:

```
Cong MACRO
Xor AX,AX
Add AX,BX
Add AX,CX
Add AX,DX
ENDM
```

Để sử dụng Macro này ta chỉ cần đưa tên của macro vào vị trí gọi.

```
…
Mov BX,10
Mov CX,100
Mov DX,1000
Cong
…
```

Chú ý:

- Chương trình con thường được sử dụng khi cần tiết kiệm vùng nhớ vì với chương trình con thì các mã lệnh được dịch ra cho một nhiệm vụ nào đó chỉ có một lần và nó sẽ được gọi ở bất kỳ nơi nào trong chương trình khi cần đến. Tuy nhiên Macro được đánh giá là thực hiện nhanh hơn vì nó thực hiện một nhóm lệnh như 1 phần của chương trình gọi nó và không phải sử dụng lệnh CALL và RET.
- Macro linh hoạt hơn so với chương trình con. Chúng sử dụng các lệnh điều khiển để trao đổi giữa tham trị và tham số hình thức. Nếu trong 1 chương trình có nhiều đoạn mã lệnh thực hiện các nhiệm vụ gần giống nhau thì macro rất hiệu quả.

d. Trao đổi tham số

Một macro có thể có 0,1 hoặc nhiều tham số. Macro ở ví dụ trên là macro không có tham số. Ta xem xét một số macro có tham số thông qua các ví dụ.

Ví dụ:

Dùng 1 macro để khai báo một mảng kiểu byte, có độ dài Length và tất cả các phần tử được gán giá trị ban đầu là Value.

```
Mang MACRO Value, Length
     REPT Length
  DB Value
ENDM
```

Để sử dụng Macro ở trên, ta chỉ việc truyền các giá trị cho các tham số. Chẳng hạn, cần khởi tạo 10 phần tử và giá trị khởi tạo là 0 thì ta làm như sau:

```
MangByte LABEL BYTE
Mang 0,10
```

Các giá trị 0 và 10 được truyền vào tham số Value và Length khi Macro được sử dụng. 0 và 10 là tham số thực trong khi Value và Length là tham số hình thức. Khi macro được gọi thì các tham số hình thức được thay bởi tham số thực. Nghĩa là:

```
MangByte LABEL BYTE
     REPT 10
```

```
        DB 0
      ENDM
```

2.4 Chương trình ví dụ

Ví dụ 0: Viết chương trình in ra nhập vào một kí tự nhưng in ra màn hình kí tự kế tiếp. Chẳng hạn, khi nhập vào kí tự 'a' thì mà hình lại hiện ra kí tự 'b'.

Bài giải

Ta sử dụng hàm 08 của ngắt 21h để nhập 1 kí tự không hiện lên màn hình rồi sau đó dung hàm 02 để in kí tự kế tiếp (tăng mã ASCII lên 1) ra màn hình.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov AH,08h ; nhập 1 kí tự không hiện lên màn hình
      Int 21h
      Mov DL,AL ; chuyển mã ASCII của kí tự vào DL
      Inc DL  ; DL chứa kí tự kế tiếp
      Mov AH,02h ; In ra màn hình
      Int 21h ;
      Int 20h ; trở về DOS
End Start
```

Ví dụ 1: Viết chương trình in ra 256 kí tự của bảng mã ASCII

Bài giải

Ta sử dụng một vòng lặp FOR-DO và dùng DL đê chứa mã ASCII của các kí tự trong bảng mã ASCII. CX chứa số kí tự cần in (256). Mỗi kí tự cách nhau bởi 1 dấu cách. Chương trình được viết theo khung của chương trình COM

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov CX,256 ; số kí tự cần in
      Mov DL,0 ; kí tự đầu tiên
      Mov AH,2 ; hàm 2 ngắt 21h in ra 1 kí tự lên màn hình
Tiep:
      Int 21h
      Mov BL,DL ; dùng BL để chứa tạm mã ASCII của kí tự
      Mov DL,32
      Int 21 ; In dấu cách
      Mov DL,BL ; lấy lại kí tự in cuối cùng
      Inc DL ; sang kí tự tiếp theo
      Loop Tiep ; In kí tự kế tiếp
```

```
        Int 20h ; trở về DOS
End Start
```

Ví dụ 2: Viết chương trình nhập vào một dãy các kí tự rồi hiển thị nó theo thứ tự ngược lại.

Bài giải

Ta có thể sử dụng ngăn xếp để giải quyết bài toán này. Mỗi khi có ký tự được nhập vào sẽ được PUSH vào ngăn xếp, sau khi nhập xong (bằng cách gõ Enter) thì các kí tự trong ngăn xếp sẽ được POP ra và hiển thị theo thứ tự ngược lại so với ban đầu.

```
.MODEL small
.STACK 100h
.DATA
        NhapXau db 'Nhap vao day ki tu:  ','$'
        InXau db 'Day ki tu in ra theo thu tu nguoc lai la: ','$'
        xuongdong db 13,10,'$'
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX
        Mov  AH,9
        Mov DX, offset NhapXau
  Int 21h ; in lời mời nhập xâu
  Xor CX,CX ; CX=0
  Mov AH,1  ; Nhap ki tu
DocVao:
      Int 21h
        Cmp AL,13 ; co phai Enter khong?
        JE ThoiDoc ; Neu là Enter, dung lai
        Push AX ; Cho vao ngan xep
        Inc CX ; Tang de dem so ki tu da nhap
        Jmp DocVao
ThoiDoc:
        Mov AH,9
        Mov DX, offset xuongdong
        Int 21h ;xuong dong va ve dau dong
        Mov DX, offset InXau
        Int 21h ; in lời mời in xâu InXau
        Mov AH,2
HienThi:
        Pop DX   ; In tung ki tu trong ngan xep
        Int 21h
        Loop HienThi ; In ra cho den khi CX=0
        Mov AH,4Ch ; Tro ve DOS
        Int 21h
```

```
End Start
```

Ví dụ 3: Nhập vào hai số nguyên x và y (0<=x,y<=9), tính hiệu x-y và in kết quả ra màn hình.

Bài giải:

Bài toán được chia thành 3 phần:

- Nhập x và y
- Tính hiệu x-y
- In kết quả

Một số lưu ý: khi nhập vào bằng hàm 01 của ngắt 21h thì AL sẽ chứa mã ASCII của kí tự vừa nhập. Chẳng hạn, khi ta nhập vào số 3 thì AL=33h (mã ASCII của 3), do vậy để nhận được số thực sự ta phải đem trừ đi 30h. Ngược lại, khi in ra thì đang ở dạng số phải đổi sang mã ASCII bằng cách cộng thêm 30h.

Để thực hiện được phép trừ hai số. Ta tiến hành phép so sánh x và y, nếu x>y ta lấy x trừ đi y, ngược lại ta lấy y trừ đi x và in dấu trừ trước kết quả.

```
.MODEL small
.STACK 100h
.DATA
      stringX db 'x= ','$'
      stringY db 'y= ','$'
      xuongdong db 13,10,'$'
      Hieu db 'x-y = ','$'
      X  db ?
      Y db ?
.CODE
Start:
      Mov AX,@Data
      Mov DS,AX
      Mov  AH,9
      Mov DX, offset stringX
      Int 21h   ; in xâu 'x = '
      Call Nhap  ; gọi chương trình con Nhập
      Mov x,AL
      Mov  AH,9
      Mov DX, offset xuongdong
      Int 21h   ; in xâu xuống dòng và về đầu dòng
      Mov DX, offset xuongdong
      Int 21h   ; in xâu 'y = '
      Call Nhap  ; gọi chương trình con Nhập
      Mov y, AL
      Mov  AH,9
      Mov DX, offset xuongdong
      Int 21h   ; in xâu xuống dòng và về đầu dòng
```

```
        Mov DX, offset xuongdong
        Int 21h   ; in xâu 'x-y = '
        Mov DL,x  ; DL=x
        Cmp DL,y ; so sánh x với y
        Sub DL,y
        Call Inra ; gọi chương trình con Inra
        Jmp Ketthuc ; nhayr
        Jb Behon ; nhảy nếu x<y
        Mov AL,y
        Sub AL,x
        Mov AH,2
        Mov DL,'-' ; In dấu trừ trước kết quả
        Int 21h
        Mov DL,AL
        Call Inra ; gọi chương trình con Inra
Ketthuc:
        Mov AH,4Ch
        Int 21h
End Start
;----------------------------------------
; chương trình con nhập, trả lại số nhập được trong AL
;----------------------------------------
Nhap Proc
        Mov AH,1 ; hàm 01 nhập vào 1 kí tự
Nhaplai:
        Int 21h
        Cmp AL,30h ; nhỏ hơn kí tự '0'
        Jb NhapLai ; nhập lại
        Cmp AL,39h ; lơn hơn kí tự '9'
        Ja NhapLai ; nhập lại
        Sub AL,30h ; đổi mã ASCII sang số
Nhap Endp
;----------------------------------------
;chương trình con In ra 1 số trong khoảng 0..9 trong DL
;-------------------------------------

Inra Proc
        Mov AH,2 ; in ki tự
        Add DL,30h ; đổi sang mã ASCII
        Int 21h
Inra Endp
;-------------------------------------
```

Ví dụ 4: Nhập vào một xâu kí tự rồi in xâu đó ra màn hình

Bài giải:

Đây là một bài tập không khó, tuy nhiên có một số vấn đề mà người học lập trình cần biết trước khi viết chương trình giải bài toán này. Đó là cấu trúc của vùng đệm (buffer) khi lưu trữ xâu kí tự. Chẳng hạn, xâu 'Hello' được lưu trữ trong vùng đệm như sau:

Chứa độ dài lớn nhất của xâu	Chứa độ dài thực của Xâu	Kí tự đầu tiên					Kí tự kết thúc xâu
255	5	H	e	l	l	o	$

Byte đầu tiên của vùng đệm chứa độ dài lớn nhất của xâu, byte thứ 2 chứa độ dài thực. Xâu thực sự được chứa từ byte thứ 3 trở đi. Tuy nhiên, thông thường thì người dùng kết thúc việc nhập bằng phìm Enter có mã ASCII là 13 nhưng kí tự kết thúc xâu lại là $ nên chương trình phải xử lý việc này bằng cách sau:

Lấy địa chỉ offset của xâu đem cộng với nội dung byte thứ hai rồi cộng với 2 thì sẽ trỏ đến byte cuối cùng của xâu đang chứa mã ASCII của Enter (13) rồi thay thế mã này bởi kí tự kết thúc xâu là '$'.

Dưới đây là chương trình hợp ngữ:

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
        XauIn db 'Nhap xau: ','$'
        XauOut db 'Xau vua nhap: ','$'
        Xuongdong db 13,10,24h
        Buffer db 100 dup(?)  ; Khai bao buffer
Start:
        Mov AH,9
        Mov DX, offset XauIn
        Int 21h
        Mov AH,0Ah
        Mov DX, offset Buffer
        Mov BX,DX ; BX va DX cung tro den Buffer
        Mov BYTE PTR[BX],100 ; Do dai lon nhat cua
        Int 21h
        Mov AH,9
        Mov DX, offset Xuongdong ; xuong dong va ve dau dong
        Int 21h
        Mov DX, offset XauOut  ; Xau kq
```

```
        Int 21h
        Mov DX,BX ; BX,DX cung tro den Buffer
        Add BL,[BX+1] ; Cong vao do dai thuc cua xau vao BX
        Add BX,2    ; Tro den byte cuoi cung
        Mov BYTE PTR[BX],'$' ; Thay the byte cuoi cung boi $
        Add DX,2 ; bo qua hai byte dau
        Int 21h  ; in xau ra
        Int 20h ; trở về DOS
End Start
```

Ví dụ 5: Viết chương trình tạo một thư mục với tên thư mục được nhập từ bàn phím.

Bài làm

Trước hết, ta phải nhập vào 1 xâu ký tự tên thư mục. Sau đó sử dụng hàm 39h để tạo thư mục. Kết thúc việc tạo thư mục ta kiểm tra cờ Carry (CF), nếu cờ Carry bằng 1 thì việc tạo thư mục đã bị lỗi, ngược lại là tạo thành công. Lệnh JC (Jump if Carry equals to 1) thực hiện việc đó.

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
        TenThuMuc db 'Nhap ten thu muc: ','$'
        OK db 'Tao thu muc thanh cong','$'
        NotOK db 'Tao thu muc khong thanh cong','$'
        Xuongdong db 13,10,24h
Start:
        Mov AH,9
        Mov DX, offset TenThuMuc
        Int 21h
        Mov AH,0Ah
        Mov DX, offset Buffer
        Mov BX,DX    ; BX va DX cung tro den Buffer
        Mov BYTE PTR[BX],100 ; Do dai lon nhat cua
        Int 21h
        Mov AH,9
        Mov DX, offset Xuongdong ; xuong dong va ve dau dong
        Int 21h
        Mov DX,BX    ; BX,DX cung tro den Buffer
        Add BL,[BX+1] ; Cong vao do dai thuc cua xau vao BX
        Add BX,2      ; Tro den byte cuoi cung
        Mov BYTE PTR[BX],'$' ; Thay the byte cuoi cung boi $
        Add DX,2      ;bo qua hai byte dau, DX=ten thu muc
        Mov AH,39h    ; ham tao thu muc
        Int 21h
        JC Error      ; CF=1, bi loi
        Mov AH,9
```

```
        Mov DX, offset OK ; thanh cong
        Int 21h
        Jmp Ketthuc
Error:
        Mov AH,9
        Mov DX, offset NotOK ; Khong thanh cong
        Int 21h
Ketthuc:
        Int 20h ; trở về DOS
End Start
```

Ví dụ 6: Hãy định nghĩa trước một mảng các số nguyên trỏ bởi biến Mang. Hãy sắp xếp theo chiều tăng dần mảng số nguyên này rồi in kết quả lên màn hình.

Bài làm

Đây là một bài toán hay gặp khi học các ngôn ngữ lập trình. Ta sử dụng thuật giải sắp xếp chèn (INSERTION SORT) để giải bài toán này. Ý tưởng như sau: tìm phần tử lớn nhất của mảng rồi đặt phần tử đó vào cuối dãy, sau đó lại tiếp tục quá trình này với các phần tử còn lại. Tại mỗi lần tìm thì một phần tử được đặt đúng chỗ. Ở bước lặp thứ i có i phần tử được đặt đúng chỗ và ta chỉ cần tìm phần tử lớn nhất trong n-i phần tử còn lại.

Ta tổ chức chương trình này thành một chương trình chính và một chương trình con. Chương trình con Exchange sẽ làm nhiệm vụ hoán đổi vị trí của hai phần tử của dãy.

```
.MODEL small
.STACK 100h
.DATA
        Thongbao db 'Day da sap xep: ','$'
        xuongdong db 13,10,'$'
        Mang  db  8,4,3,1,2,5,1
          Db '$'
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX
        Mov  AH,9
        Mov DX, offset Thongbao
        Int 21h   ; in thongbao
        Mov DX, offset xuongdong
        Int 21h   ; nhay xuong dong
        Mov BX,7  ; BX= so phan tu cua mang
        Mov DX, offset Mang ; DX tro vao mang
        Dec BX ; so vòng lặp bên ngoài
Lap:
        Mov SI,DX ; SI trỏ vào đầu mảng
        Mov CX,BX ;Số lần lặp ở vòng lặp bên trong (tìm max)
        Mov DI, SI ;giả sử phần tử thứ 1 là max
```

```
        Mov AL, [DI] ; AL= max
TimMax:
        Inc SI   ; phần tử kế tiếp
        Cmp [SI],AL ; phần tử mới > max?
        JB Tiep ; không lớn hơn max
        Mov DI,SI ; lớn hơn max, DI trỏ vào max
        Mov AL,[DI]; Đưa max vào AL
Tiep:
        Loop TimMax
        Call DoiCho
        Dec BX
        JNZ Lap
; In Mang
 Mov BX,DX ; BX trỏ đến phần tử đầu tiên
 Mov CX,7 ; in cả 7 phần tử
 Mov AH,2
InMang:
 Mov DL,[BX]
 Add DL,30h
 Int 21h  ; in ra
 Mov DL,32 ; in dấu cách giữa các phần tử cho dễ xem
 Int 21h
 Inc BX  ; sang phần tử kế tiếp
 Loop InMang

Ketthuc:
        Mov AH,4Ch
        Int 21h
End Start
;---------------------------------------
; chương trình con DoiCho
;---------------------------------------
DoiCho Proc
        Push AX
        Mov AL,[SI]
        XCHG AL,[DI]
        Pop AX
        Ret
Nhap Endp
;---------------------------------------
```

2.5 TÓM TẮT

Chương này trình bày các vấn đề cơ bản của lập trình hợp ngữ: tạo và thực hiện một chương trình hợp ngữ, các cấu trúc lập trình cơ bản , chương trình con và Macro.

Trong phần viết và thực hiện một chương trình chúng ta đã tìm hiểu cấu trúc đầy đủ một lệnh hợp ngữ, cách thức khai báo hằng và biến. Phần này cũng trình bày khung của chương trình hợp ngữ, cách tạo, dịch và chạy một chương trình hợp ngữ.

Công cụ quan trọng của các ngôn ngữ lập trình là các cấu trúc lập trình. Hợp ngữ chỉ hỗ trợ các lệnh nhảy (jump) và so sánh (compare) cho phép người lập trình cài đặt các cấu trúc này. Phần các cấu trúc lập trình cơ bản trình bày cú pháp, cách cài đặt các cấu trúc: tuần tự, điều kiện (IF-THEN), điều kiện phân nhánh (IF-THEN-ELSE), lựa chọn (CASE), các cấu trúc lặp xác định trước (FOR-DO) và lặp không xác định trước (WHILE-DO, REPEAT-UNTIL).

Phần cuối cùng là các vấn đề liên quan đến chương trình con. Ngoài các vấn đề cơ bản liên quan đến chương trình con như: cơ chế, cấu trúc của chương trình con, cách thức truyền tham số, phần này cũng đề cập đến việc chia nhỏ một chương trình lớn thành các chương trình con và gói chúng vào các module hay thư viện nhờ các lệnh điều khiển PUBLIC, EXTRN, GLOBAL. Ngoài ra, cùng với chương trình con, Macro là một lựa chọn khi viết chương trình hợp ngữ. Macro không những thực hiện nhanh, mềm dẻo và khá hiệu quả. Chúng ta cũng đã khảo sát về các lệnh điều khiển hay được sử dụng trong các Macro, cách khai báo và sử dụng Macro.

2.6 BÀI TẬP

Bài 1: Viết chương trình in ra màn hình 26 chữ cái ‘A’ …’Z’.

Bài 2: Viết chương trình nhập vào 1 số nguyên n ($0 \le n \le 9$), tính tổng S=1+2+3+…+n rồi in kết quả ra màn hình.

Bài 3: Viết chương trình nhập vào hai số nguyên x và y ($0 \le x,y \le 9$), tính tổng x+y rồi in kết quả ra màn hình. Yêu cầu: Chương trình được tổ chức như sau: gồm 1 chương trình chính, 2 chương trình con hoặc 2 macro, 1 dùng để nhập vào 1 số và 1 dùng để in kết quả .

Bài 4: Viết chương trình nhập vào từ bàn phím hai số nguyên a và b với $0 \le a,b \le 256$. Tính tích của chúng và in kết quả ra màn hình. Yêu cầu chương trình phải được tổ chức thành các chương trình con hoặc macro.

Bài 5: Viết chương trình nhập vào một tên file rồi xóa file đó. In ra màn hình thông báo xóa thành công hay không.

Bài 6: Viết chương trình tạo một file backup từ một file văn bản nguồn. Tên của file văn bản được nhập vào từ bàn phím.

2.7 TÀI LIỆU THAM KHẢO

1. Văn Thế Minh. Kỹ thuật Vi xử lý. Nhà XB Giáo dục 1997.
2. Đặng Thành Phu. Turbo Assembler và Ứng dụng. NXB Khoa học và Kỹ thuật 1998.
3. Nguyễn Minh San. Cẩm nang Lập trình hệ thống (bản dịch). NXB Tổng cục Thống kê.2001.

CHƯƠNG 3. CÁC CÔNG CỤ HỖ TRỢ

Phần này trình bày về trình tiện ích Debug, chương trình mô phỏng Emu8086, sự kết hợp chương trình hợp ngữ với ngôn ngữ bậc cao. Cuối cùng ta xem xét về các chương trình ngắt.

3.1 BỘ GỠ RỐI DEBUG

3.1.1 Tổng quan về Debug

DEBUG là một trình tiện ích trợ giúp người lập trình tác động đến quá trình thực hiện một công việc (task) nào đó. Đồng thời Debug là một chương trình dùng để gỡ lỗi chương trình.

Debug hỗ trợ cho người dùng các nhóm lệnh sau:

Thao tác với bộ nhớ:

- lệnh hiển thị nội dung ô nhớ (lệnh D)
- lệnh sửa nội dung ô nhớ (lệnh E)
- điền thông tin vào một vùng nhớ (lệnh F)
- chuyển nội dung từ vùng nhớ này sang vùng nhớ khác (lệnh M).

Thao tác với các files:

- đặt tên file (lệnh N)
- nạp nội dung một file vào bộ nhớ (lệnh L)
- chạy file dang .COM hoặc .EXE (lệnh G)

Truy cập đến các sector trên đĩa (lệnh L,W)

Soạn thảo và thực hiện một chương trình hợp ngữ (lệnh A,G)

Theo dõi quá trình thực hiện 1 chương trình

- xem, sửa chữa trạng thái thanh ghi (lệnh R)
- chạy từng bước (lệnh T, lệnh P)

Thực hiện một số thao tác vào/ra đối với các thiết bị ngoại vi (lệnh I và O)

Dịch ngược từ mã máy sang hợp ngữ (lệnh U)

Tìm kiếm (lệnh S)

3.1.2 Sử dụng Debug

a. Khởi động Debug

Cách 1: Tại thư mục DOS:

C:\DOS>

gõ debug <Enter>

C:\DOS> debug <Enter>

Dấu nhắc của Debug là dấu –

Cách 2:

Gõ C:\DOS> debug tenfile.exe <Enter>

Khi đó cả trình debug và chương trình người dùng (tenfile.exe) đều được đưa vào bộ nhớ RAM.

b. Một số lưu ý

- Địa chỉ của vùng nhớ (address): được thể hiện dưới dạng SEGMENT:OFFSET, chẳng hạn như:

  DS:0300 hoặc

  9D0:01FF

  Nếu ở đoạn hiện tại, ta có thể chỉ cần dùng địa chỉ offset, ví dụ:

  02FFh
- Khoảng (range): thể hiện địa chỉ một vùng nhớ:

  address L value

  ví dụ: DS:1FF L 10
- Mỗi lệnh của Debug gồm 1 kí tự duy nhất
- Giữa tên lệnh và tham số có ít nhất 1 dấu cách
- Dùng dấu cách giống như dùng dấu phẩy (,)
- Kết thúc 1 lệnh đang được thực hiện bằng Ctrl+C hoặc Ctrl+Break
- Lệnh được thực hiện nếu gõ tên lệnh và gõ enter

3.1.3 Các lệnh của Debug

1. Lệnh: A

Chức năng: Soạn thảo và dịch trực tiếp các lệnh hợp ngữ.

Cú pháp: **A [địa chỉ] <Enter>**

Trong đó **[địa chỉ]** là địa chỉ offset của ô nhớ (dạng hexa) mà ta cần đặt mã lệnh vào.

Ví dụ:

```
-A 1FF0
xxxx:1FF0  mov AH,9
xxxx:1FF2  mov AH,9
```

- Nếu chưa xác định được địa chỉ ban đầu thì lệnh sẽ tự đông đưa vào địa chỉ offset 0100h
- Nếu phát hiện lỗi trong một lệnh, Debug sẽ đưa ra thông báo ERROR và hiện lại địa chỉ ô nhớ có lỗi đó để người sử dụng sửa lại lệnh đó cho đúng
- Muốn trở lại dấu nhắc Debug thì nhấn hai lần Enter
- Lệnh A luôn sử dụng địa chỉ tuyệt đối, nghĩa là luôn đi kèm với 1 địa chỉ chính xác.

2. Lệnh: C

Chức năng: So sánh nội dung của hai vùng nhớ.

Cú pháp: **C khoảng,địa chỉ <Enter>**

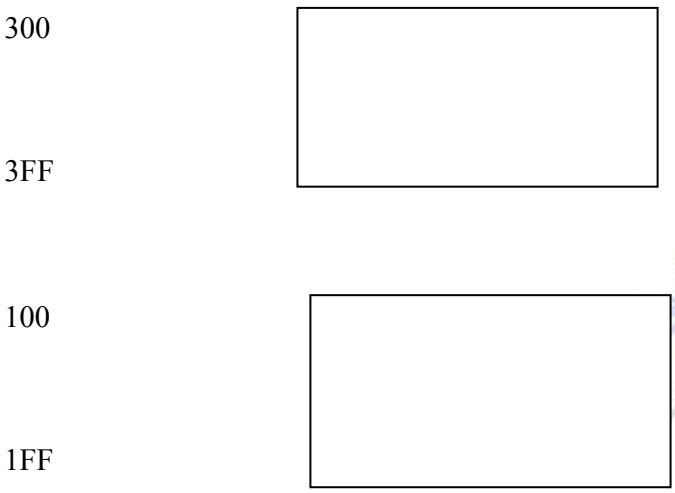
Trong đó khoảng bao gồm địa chỉ đầu tiên và địa chỉ cuối cùng của một vùng nhớ, địa chỉ là 1 địa chỉ offset bắt đầu của 1 vùng nhớ khác.

Ví dụ:

-C 100, 1FF, 500 <Enter>

Hoặc:

-C 100 L 100, 500 <Enter>



3. Lệnh: D

Chức năng: Hiển thị nội dung của một vùng nhớ.

Cú pháp: **D [khoảng |địa chỉ] <Enter>**

Trong đó khoảng bao gồm địa chỉ đầu tiên và địa chỉ cuối cùng của một vùng nhớ, hoặc địa chỉ là 1 địa chỉ offset của 1 ô nhớ khác.

Ví dụ:

-D 100, 1FF <Enter>

Hoặc:

-D 100 L 11 <Enter>

Nếu sử dụng lệnh:

-D <address> <Enter>

Thì nội dung các ô nhớ từ địa chỉ đã cho đến 128 byte kế tiếp sẽ được hiện lên. Nếu tiếp tục

-D <Enter>

Thì nội dung 128 byte kế tiếp theo đó được hiện lên.

4. Lệnh: E

Chức năng: Hiện nội dung ô nhớ và cho phép sửa nội dung ô nhớ.

Cú pháp:

Cách 1*:* **E <địa chỉ> <danh sách các giá trị> <Enter>**

Ví dụ:

-E 01FF:0100 ‘ABC’9A <Enter>

Thì các ô nhớ từ 01FF:0100 đến 01FF:0103 sẽ lần lượt được điền các các giá trị là mã ASCII của A, B, C và giá trị 9A.

Cách 2*:* **E <địa chỉ> <Enter>**

Thì nội dung của ô nhớ có địa chỉ trên sẽ hiện lên, ta có thể thay đổi giá trị mới và sau đó nếu muốn thay đổi nội dung của ô nhớ kế tiếp thì nhấn dấu cách (SPACE), còn nếu muốn dừng lại thì nhấn Enter để trở lại màn hình debug.

Ví dụ:

-E 01FF:0100 9A

Nếu muốn sửa giá trị của ô nhớ này thì đưa vào giá trị mới vào rồi nhấn Enter đển hiện nội dung của ô nhơ kế tiếp 01FF:0101 và cứ tiếp tục như vậy cho đến khi nhấn Enter nếu muốn dừng.

5. Lệnh: F

Chức năng: Nạp các giá trị trong danh sách vào một vùng nhớ.

Cú pháp: **F <khoảng> <danh sách các giá trị> <Enter>**

Ví dụ:

-F 01FF:0100 L 4,‘ABC’9A <Enter>

Thì các ô nhớ từ 01FF:0100 đến 01FF:0103 sẽ lần lượt được điền các các giá trị là mã ASCII của A, B, C và giá trị 9A.

6. Lệnh: G

Chức năng: Cho thực hiện một chương trình đang hiệu chỉnh. Việc thực hiện chương trình sẽ dừng lại khi đạt đến địa chỉ dừng. Sau đó’, hiển thị các thanh ghi và dòng lệnh thực hiện tiếp theo của chương trình

Cú pháp: **G [<địa chỉ đầu>] [<địa chỉ dừng>] <Enter>**

Nếu gõ lệnh **- G [<địa chỉ đầu>] <Enter>**

Thì chương trình sẽ được thực hiện từ địa chỉ đầu cho đên lệnh cuối cùng của chương trình. Nếu là chương trình con thì dừng lại khi gặp lệnh RET, nếu là macro thì dừng lại khi gặp lệnh ENDM.

Nếu gõ lệnh **- G <Enter>**

Thì chương trình sẽ được thực hiện từ địa chỉ đã được nạp vào cặp thanh ghi CS:IP cho đến lệnh cuối cùng của chương trình

7. Lệnh: H

Chức năng: Cộng và trừ hai giá trị hexa và hiển thị kết quả của tổng và hiệu lên màn hình.

Cú pháp: **H <giá trị 1> <giá trị 2> <Enter>**

Ví dụ: - H 10, 0f <Enter>

1F, 01

Kết quả của tổng là 1F và của hiệu là 01

8. Lệnh: I

Chức năng: Đọc và hiển thị giá trị của một cổng lên màn hình.

Cú pháp: **I <địa chỉ của cổng vào> <Enter>**

Ví dụ: - I 1f <Enter>

26

26 là giá trị đọc được từ cổng 1F.

9. Lệnh: L

Chức năng: Chuyển nội dung 1 file hoặc nội dung sector của đĩa vào vùng nhớ.

Cú pháp:

Dạng 1: **L [<địa chỉ>,[, ổ đĩa, sector, số sector]] <Enter>**

Đọc số liệu từ sector đầu của ổ đĩa, với số lượng sector và bắt đầu từ địa chỉ được chỉ ra ở tham số thứ nhất.

Ví dụ 1: - L 01FA:0100 1 0A 30 <Enter>

Đọc số liệu của 48 sector (30h) bắt đầu từ sector 0A của ổ đĩa B và vùng nhớ có địa chỉ 01FA:0100.

(Các ổ đĩa được kí hiệu như sau: 0 là ổ đĩa A, 1 là ổ đĩa B, 2 là ổ đĩa C, 3 là ổ đĩa D).

Nếu tên file được đặt tên bởi lệnh –N <tên file> thì:

- lệnh –L sẽ nạp nội dung của file vào vùng nhớ mặc định CS:0100
- lệnh –L <địa chỉ> sẽ nạp nội dung của file vào vùng nhớ có địa chỉ <địa chỉ>.

Ví dụ 2:

- N cong2so <Enter>
- L <Enter>

Nội dung của file cong2so sẽ được nạp vào vùng nhớ có địa chỉ đầu là CS:0100

Ví dụ 3:

- N cong2so <Enter>
- L 03FF <Enter>

Nội dung của file cong2so sẽ được nạp vào vùng nhớ có địa chỉ đầu là CS:03FF

10. Lệnh: M

Chức năng: Chuyển nội dung từ một vùng nhớ sang một vùng nhớ khác.

Cú pháp: **-M <khoảng>,<địa chỉ> <Enter>**

Nếu trong khoảng và địa chỉ không xác định đoạn thì sẽ lấy DS là địa chỉ đoạn..

Ví dụ nếu sử dung 1 trong 3 lệnh sau:

- M 01FF:0100, 010E, 01FF:0200 <Enter>
- M 01FF:0100 L F, 01FF:0200 <Enter>
- M 01FF:0100 010E 01FF:0200 <Enter>

Thì 15 byte từ vùng nhớ có địa chỉ 01FF:0100 đến 01FF:010E sẽ được chuyển đến vùng nhớ có địa chỉ bắt đầu là: 01FF:0200

11. Lệnh: N

Chức năng: Đặt tên cho file.

Lệnh này thường được đi kèm với các lệnh –L và –W dùng tên file đó .

Cú pháp: **-N <tên file>.EXE (hoặc .COM) <Enter>**

Ví dụ:

- N Tenfilemoi.exe <Enter>
- L <Enter>

12. Lệnh: O

Chức năng: Đưa một byte dữ liệu ra cổng.

Cú pháp: **-O <địa chỉ cổng ra>, <giá trị> <Enter>**

Ví dụ:

- O 02F, 20 <Enter>

13. Lệnh: Q

Chức năng: Thoát khỏi Debug và trở về DOS.

Cú pháp: **-Q <Enter>**

14. Lệnh: R

Chức năng: Hiển thi và sửa đổi nội dung các thanh ghi.

Cú pháp: **-R [thanh ghi | F] <Enter>**

Có các trường hợp sau:

- R <enter> hiển thị và sửa đổi nội dung các thanh ghi.
- RAX <enter> hiển thị nội dung của thanh ghi AX và cho phép sửa nội dung đó, ví dụ: AX 101A. Nếu không muốn thay đổi giá trị thì nhấn Enter, còn muốn sửa giá trị mới thì nhập giá trị mới vào rồi nhấn Enter. Muốn hiển thị nội dung của các thanh ghi kế tiếp (BX, CX, DX) thì nhấn dấu cách (SPACE).
- R F <Enter> hiện và sửa nội dung của thanh ghi cờ.

15. Lệnh: S

Chức năng: Tìm trong vùng nhớ xác định bởi khoảng các kí tự trong danh sách.

Cú pháp: **-S <khoảng>,<danh sách> <Enter>**

Nếu khoảng không xác định thì đoạn ngầm định là thanh ghi DS.

Ví dụ 1:

- S 01FF:0100 0110 20 <Enter>

Hoặc:

- S 01FF:0100 L 10 20 <Enter>

Thì sẽ tìm các ô nhớ có nội dung bằng 20h trong vùng nhớ từ 01FF:0100 đến 01FF:0110. Kết quả là tất cả địa chỉ của các ô nhớ có nội dung bằng 20 thì sẽ được hiển thị, chẳng hạn có 3 ô nhớ có nội dung bằng 20 thì màn hình sẽ liệt kê như sau:

01FF:0100

01FF:0104

01FF:0105

Ví dụ 2:

- S 01FF:0100 0110 'ABC'2E <Enter>

Sẽ tìm 4 ô liên tiếp chứa giá trị là mã ASCII của A, B, C và 2E.

16. Lệnh: T

Chức năng: Thực hiện một hay nhiều lệnh bắt đầu từ địa chỉ CS:IP hoặc từ địa chỉ được chỉ ra ở dấu = , và hiển thị trạng thái toàn bộ các thanh ghi sau mỗi lệnh.

Cú pháp: **-T [= địa chỉ][, số lệnh] <Enter>**

- địa chỉ là địa chỉ của lệnh đầu tiên sẽ thực hiện
- số lệnh được thực hiện trong chế độ này.

Có các trường hợp sau:

- T <enter> lệnh tại địa chỉ CS:IP sẽ được thực hiện.
- T 10 <enter> 10 lệnh bắt đầu từ địa chỉ CS:IP sẽ được thực hiện. Trạng thái của tất cả các thanh ghi sau mỗi lệnh sẽ được hiện ra 1 cách liên tục.
- T=2FF,10 <enter> sẽ thực hiện 16 lệnh bắt đầu từ lệnh tại địa chỉ CS:02FF.

17. Lệnh: P

Chức năng: Giống lệnh T nhưng thực hiện cả 1 chương trình con.

Cú pháp: **-P [= địa chỉ][, số lệnh] <Enter>**

18. Lệnh: U

Chức năng: Dịch ngược các lệnh dưới dạng mã máy nằm trong vùng nhớ sang dạng hợp ngữ và hiển thị địa chỉ, mã máy và lệnh dạng gợi nhớ lên màn hình.

Cú pháp: **-U [khoảng][địa chỉ]<Enter>**

Có các trường hợp sau:

- U <khoảng> <enter> các lệnh nằm trong vùng nhớ sẽ được dịch ngược.
- U <địa chỉ> <enter> dịch ngược bắt đầu từ địa chỉ cho đến 128 byte kế tiếp.
- U <enter> dịch ngược bắt đầu từ địa chỉ CS:IP đến 128 byte kế tiếp.

19. Lệnh: W

Chức năng: Ghi dữ liệu lên đĩa.

Cú pháp: **-W [địa chỉ [,ổ đĩa, sector đầu, số sector] <Enter>**

Dữ liệu trong vùng nhớ bắt đầu từ địa chỉ ghi lên ổ đĩa vào sector đầu tiên cho đến sector cuối do số sector xác định.

Ví dụ:

- N cong2so.exe <enter>
- W 01FF:0200, 1, 2A,5 <Enter>

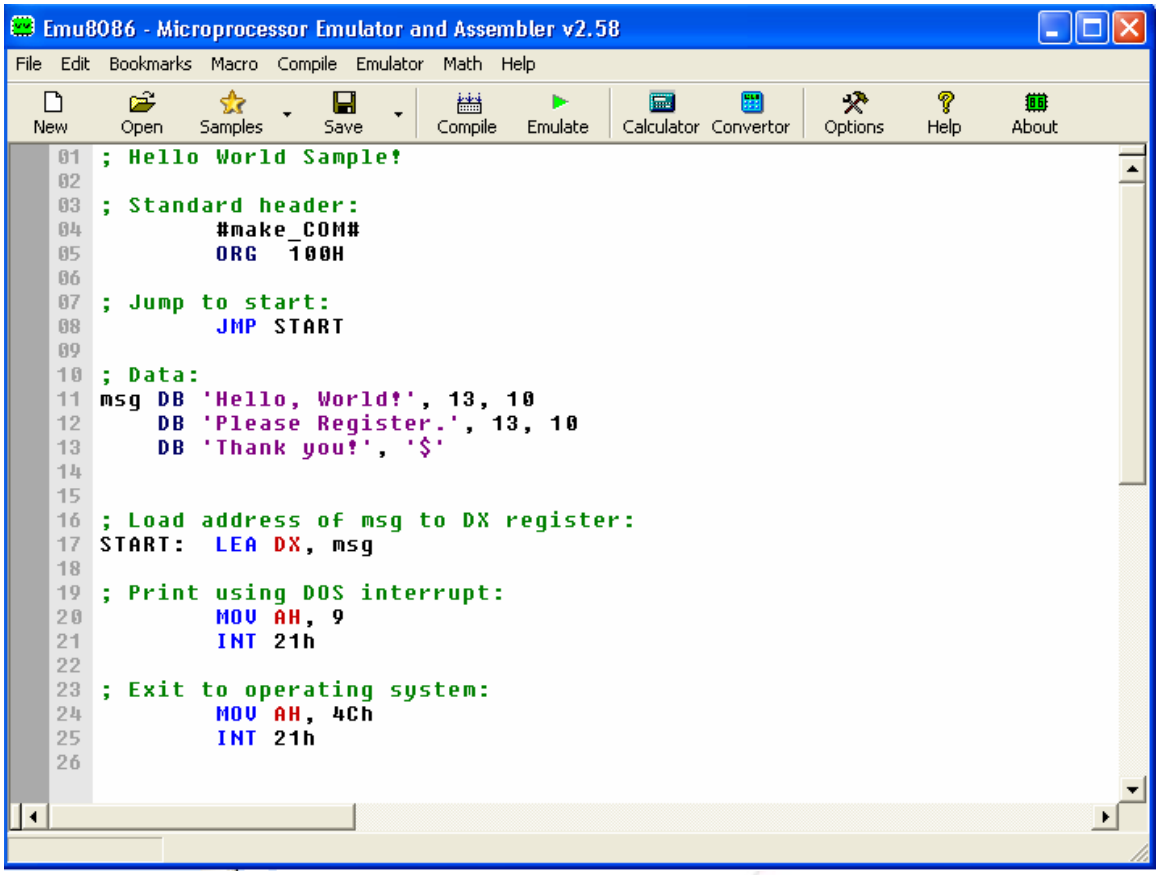
Dữ liệu từ vùng nhớ xác định bởi 01FF:0200 được ghi vào ổ đĩa B từ sector 2A và ghi vào 5 sector với tên file là cong2so.exe.

3.2 CHƯƠNG TRÌNH MÔ PHỎNG EMU8086

Hiện nay, hầu hết các desktop tại các phòng thực hành tại Việt nam chạy hệ điều hành 32-bit như windows XP, NT, 2000… thì người lập trình không thể gọi ngắt bằng chương trình người dùng được. Thay vì gọi ngắt, các hệ điều hành 32-bit cung cấp một tập các hàm giao diện lập trình ứng dụng gọi là API (Application Progammable Interface) cho phép người lập trình gọi hàm. Để người mới học lập trình hệ thống có thể lập trình với các ngắt mà không bị giới hạn bởi phiên bản khác nhau của các hệ điều hành của Microsoft thì cách tốt nhất là học lập trình trên môi trường mô phỏng Emulator 8086. Phần này chúng tôi giới thiệu về phần mềm mô phỏng CPU 8086 của công ty phần mềm Emu8086 Inc., phiên bản 2.58. Các phiên bản mới hơn có thể được download tại địa chỉ của trang web: www.emu8086.com.

3.2.1 Các chức năng soạn thảo, dịch và thực hiện chương trình.

Dưới đây là màn hình cho phép người sử dụng viết một chương trình hợp ngữ hoặc chạy thử một ví dụ có sẵn:



New: tạo một chương trình mới, khi đó người dùng sẽ được hỏi xem sẽ tạo file chương trình dạng nào: COM, EXE, BIN hay BOOT .

Open: mở một file chương trình nguồn hợp ngữ.

Samples: Liệt kê các file chương trình mẫu có sẵn do chương trình mô phỏng cung cấp.

Save: Lưu file chương trình nguồn

Compile: dịch file chương trình nguồn

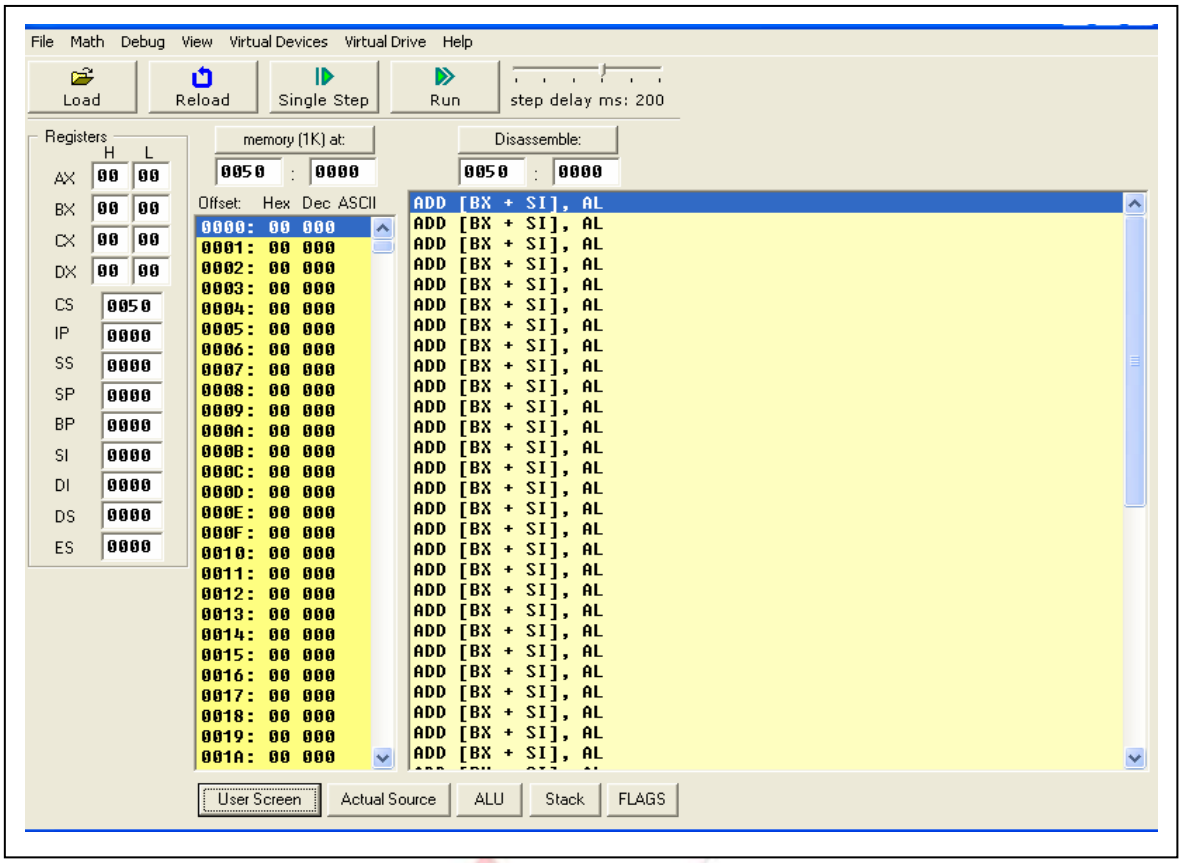
Emulate: cho phép thực hiện chương trình nguồn. Các trạng thái của quá trình thực hiện chương trình được hiển thị trên màn hình mô phỏng dưới đây.

Calculator: người dùng có thể nhập 1 vào một biểu thức với các số là: có dấu, số dạng word hoặc số dạng byte để tính toán. Kết quả tính toán được hiển thị một trong các dạng số thập phân, nhị phân, hexa hoặc số bát phân (cơ số 8).

Convertor: Bộ chuyển đổi gữa các cơ số. Emu8086 hỗ trợ chuyển đổi giữa các cơ số 16 thành cơ số 10 có dấu hoặc không dấu. Chuyển đổi từ cơ số 8 thành thành cơ số 2 (nhị phân). Một mã ASCII gồm 2 số hexa cũng có thể được chuyển đổi thành thập phân hoặc nhị phân.

3.2.2 Chức năng mô phỏng quá trình thực hiện chương trình.

Dưới đây là màn hình mô phỏng trạng thái thực hiện một chương trình.



Hình: Màn hình mô phỏng chương trình

Các chức năng chính:

Load: tải chương trình. Trước khi thực hiện thì chương trình sẽ được tải vào trong bộ nhớ. Chương trình có thể ở dạng các file thực hiện được như EXE, COM, BIN, BOOT hoặc dưới dạng file nguồn ASM.

Reload: người dùng có thể tải lại 1 chương trình.

Single Step: chạy chương trình theo chế độ từng lệnh. Với chế độ này, người dùng có thể quan sát trạng thái các thanh ghi, bộ nhớ trong…

Run: chế độ chạy tất cả các lệnh trong chươn trình.

Trên màn hình, người dùng có thể quan sát trạng thái các thanh ghi và đoạnh bộ nhớ sử dụng cho đoạn mã lệnh của chương trình.

Phần registers mang nội dung của các thanh ghi trong đó các thanh ghi AX,BX,CX và DX được chia làm 2 nửa. phần cao (H) và phần thấp (L). Ngoài ra, ta có thể xem nội dung các thanh ghi đoạn, con trỏ lệnh, ngăn xếp…

Phần bộ nhớ lưu trữ đoạn mã chương trình. Địa chỉ đoạn (dạng hexa) được lưu trong thanh ghi CS. Danh sách địa chỉ offset được hiển thị dưới các dạng hexa và thập phân.

Ngoài ra, người dùng có thể có thể xem:

- kết quả hiển thị lên mà hình (nhắp chuột vào nút User Screen).
- mã nguồn của chương trình (nhắp chuột vào nút Actual Source).
- trạng thái ALU (nhắp chuột vào nút ALU).

- nội dung của ngăn xếp (nhắp chuột vào nút Stack).
- nội dung của thanh ghi cờ (nhắp chuột vào nút FLAG)

3.2.3 Các chương trình mẫu.

Emu8086 cung cấp cho người dùng 54 chương trình mẫu. Chúng rất có ích cho người học lập trình hợp ngữ. Từ các chương trình đơn giản như Hello world cho đến một số chương trình thao tác với một số thiết bị ngoại vi điển hình như màn hình, máy in…Để chạy thử các chương trình mẫu này, người dùng nhắp chuột vào nút Samples/ More Samples để chọn ra một file chương trình để chạy thử. Dưới đây là các giải thích cho 1 một số chương trình mẫu.

Chương trình Calculate SUM. Chương trình tính tổng các phần tử trong một mảng V1 đã được định nghĩa trước và lưu kết quả vào biến V2.

Dưới đây là nội dung chương trình (lời giải thích được dịch ra tiếng Việt):

```
#make_BIN#
; Tính tổng các phần tử trong mảng V1
; Lưu kết quả vào biến V2.
; Số phần tử của mảng:
MOV CX, 5
; AL chứa tổng các phần tử:
MOV AL, 0
; BX là chỉ số của mảng:
MOV BX, 0
; Tính tổng:
Tong:
ADD AL, V1[BX]
; có thể thay đổi giá trị của mảng
; đặt giá trị phần tử bằng chỉ số
MOV V1[BX], BL
; phần tử kế tiếp:
INC BX
; lặp cho đến khi CX=0:
; tính tổng của tất cả các phần tử
LOOP Tong
; lưu kết quả vào biến V2:
MOV V2, AL
HLT
; Khai báo biến:
V1 DB 4, 3, 2, 1, 0
V2 DB 0
```

Ở chương trình trên có một số điểm khác so với các chương trình ta thường thấy. Lệnh đầu tiên của chương trình là #make_BIN#. Chương trình sẽ được viết dưới dạng file binary.

Các biến của chương trình được khai báo ở phần cuối.

Chương trình tính tính tổng hai số nguyên được nhập từ bàn phím nằm trong khoảng [-32768..32767] rồi in kết quả ra mà hình.

Các file chương trình liên quan: calc.asm và emu8086.inc

Trong file emu8086.inc chứa một số chương trình con và macro được gọi từ calc.asm.

Dưới đây là chương trình calc với các lời giải thích đã được viết lại bằng tiếng Việt.

```
; Đây là chương trình  nhập vào 2 số nguyên
; trong khoảng [-32535, 32536] từ người dùng
; Tính tổng của chúng
; rồi in kết quả ra màn hình
; chương trình dạng COM
#make_COM#
include 'emu8086.inc'
ORG     100h
; Nhảy qua đoạn khai  báo biến, hằng
JMP     START
; khai báo biến:
num  DW ?
START:
; Nhập vào số thứ nhất:
CALL    PTHIS
DB 13, 10, 'Calculation Range: [-32768..32767]', 13, 10
DB 13, 10, 'Enter first number: ', 0
; Gọi chương trình con scan_num để nhập 1 số, kết quả trả lại
; là một số được lưu trong thanh ghi CX
CALL    scan_num
; Lưu số thứ nhất vào biến num:
MOV     num, CX
; nhập vào số thứ 2:
CALL    PTHIS
msg2 DB 13, 10, 'Enter second number: ', 0
CALL    scan_num
; cộng các số:
ADD     num, CX
JO      overflow
; In kết quả bằng chương trình con PTHIS
CALL    PTHIS
DB 13, 10, 'The sum is: ', 0
MOV     AX, num
CALL    print_num
JMP     exit
; xử lý lỗi tràn:
overflow:
```

```
   PRINTN 'We have overflow!'
exit:
RET
;================================
; Khai báo việc sử dụng các chương trình con
; hoặc macro trong  emu8086.inc
; Chương trình con SCAN_NUM đọc vào 1 số
; từ người dùng và lưu vào thanh ghi CX
DEFINE_SCAN_NUM
; Chương trình con PRINT_NUM in ra
; một số có dấu nằm trong AX
; Chương trình con  PRINT_NUM_UNS  in ra
; một số không dấu nằm trong AX
; do PRINT_NUM gọi đến
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
; Chương trình con  PTHIS in ra giá trị rỗng (NULL)
; xâu được định nghĩa sau lệnh
; CALL PTHIS:
DEFINE_PTHIS
;================================
END
```

Dưới đây là các macro và chương trình con trong file INCLUDE emu8086.inc được gọi đến bởi chương trình trên.

Macro scan_num (trong đó các lời giải thích đã được viết lại bằng tiếng Việt)

```
;*************************************************************
; Đây là macro
; nhận vào một số nguyên có dấu
; và lưu trong thanh ghi CX:
DEFINE_SCAN_NUM         MACRO
; Khai báo các biến cục bộ
LOCAL make_minus, ten, next_digit, set_minus
LOCAL too_big, backspace_checked, too_big2
LOCAL stop_input, not_minus, skip_proc_scan_num
LOCAL remove_not_digit, ok_AE_0, ok_digit, not_cr
JMP     skip_proc_scan_num
SCAN_NUM        PROC    NEAR
        PUSH    DX
        PUSH    AX
        PUSH    SI
        MOV     CX, 0
```

```
        ; reset flag:
        MOV     CS:make_minus, 0
next_digit:
        ; nhập vào 1 kí tự từ bàn phìm
        ; đặt vào trong AL, dùng dịch vụ BIOS phục vụ bàn phím
        MOV     AH, 00h
        INT     16h
        ; và in ra:
        MOV     AH, 0Eh
        INT     10h
        ; Kiểm tra xem có phải là dấu âm:
        CMP     AL, '-'
        JE      set_minus
        ; phím Enter – hoàn thành việc nhập số
        CMP     AL, 13  ; 13 là mã ASCII của phím Enter?
        JNE     not_cr
        JMP     stop_input
not_cr:
        CMP     AL, 8 ; Có nhấn phím 'BACKSPACE'?
        JNE     backspace_checked
        MOV     DX, 0             ; có, thì bỏ đi số cuối cùng
        MOV     AX, CX            ;chia
        DIV     CS:ten            ; chia cho 10.
        MOV     CX, AX
        PUTC    ' '                ; xóa dấu cách
        PUTC    8                  ; backspace again.
        JMP     next_digit
backspace_checked:
        ; chỉ cho phép nhập vào số
        CMP     AL, '0'
        JAE     ok_AE_0
        JMP     remove_not_digit
ok_AE_0:
        CMP     AL, '9'
        JBE     ok_digit
remove_not_digit:
        PUTC    8       ; phím backspace.
        PUTC    ' '    ; xóa nếu kí tự nhập được không phải là số
        PUTC    8      ; phím backspace.
```

```
        JMP     next_digit ; đợi nhập vào chữ số kế tiếp.
ok_digit:
        ; nhân CX với 10
        PUSH    AX
        MOV     AX, CX
        MUL     CS:ten                  ; DX:AX = AX*10
        MOV     CX, AX
        POP     AX
        ; kiểm tra lại nếu số quá lớn
        ;
        CMP     DX, 0
        JNE     too_big
        ; Đổi từ mã ASCII ra số thực sự
        SUB     AL, 30h
        ; add AL to CX:
        MOV     AH, 0
        MOV     DX, CX      ; lưu lại
        ADD     CX, AX
        JC      too_big2    ; nhảy nếu số quá lớn
        JMP     next_digit
set_minus:
        MOV     CS:make_minus, 1
        JMP     next_digit
too_big2:
        MOV     CX, DX   ; khôi phục lại giá trị đã được sao chép
        MOV     DX, 0    ; trước khi sao lưu DX=0
too_big:
        MOV     AX, CX
        DIV     CS:ten  ; Đảo lại chữ số cuối
        MOV     CX, AX
        PUTC    8       ; backspace.
        PUTC    ' '     ; xóa đi số nhập vào cuối cùng.
        PUTC    8       ; backspace again.
        JMP     next_digit ; chờ nhấn Enter hoặc phím xóa lùi.
stop_input:
        ; kiểm tra cờ:
        CMP     CS:make_minus, 0
        JE      not_minus
        NEG     CX
```

```
not_minus:
        POP     SI
        POP     AX
        POP     DX
        RET
make_minus      DB      ?       ; sử dụng biến này như 1 cờ.
ten             DW      10      ; dùng để nhân.
SCAN_NUM        ENDP
skip_proc_scan_num:
```

Dưới đây là Macro DEFINE_PRINT_NUM in ra một số nguyên nằm trong AX (các lời giải thích được viết lại bằng tiếng Việt)

```
;*************************************************************
; Trong macro này định nghĩa chương trình con DEFINE_PRINT_NUM
; để in ra một số nguyên trong AX
; gọi đến chương trình con PRINT_NUM_UNS để in ra một số có dấu
; chương trình con liên quan:
; DEFINE_PRINT_NUM  và DEFINE_PRINT_NUM_UNS !!!
DEFINE_PRINT_NUM        MACRO
; khai báo các nhãn cục bộ
LOCAL not_zero, positive, printed, skip_proc_print_num
JMP     skip_proc_print_num
PRINT_NUM       PROC    NEAR
        PUSH    DX
        PUSH    AX

        CMP     AX, 0
        JNZ     not_zero 1

        PUTC    '0'
        JMP     printed
not_zero:
        ; Kiểm tra dấu của AX,
        CMP     AX, 0
        JNS     positive
        NEG     AX
        PUTC    '-'
positive:
        CALL    PRINT_NUM_UNS
printed:
        POP     AX
        POP     DX
        RET
```

```
PRINT_NUM       ENDP
skip_proc_print_num:
DEFINE_PRINT_NUM        ENDM

;*************************************************************
```

Dưới đây là đoạn chương trình của macro DEFINE_PRINT_NUM_UNS, macro chứa một chương trình con PRINT_NUM_UNS, in ra một số nguyên không dấu.

```
; Macro này định nghĩa một thủ tục in ra màn hình một số nguyên
; không dấu trong AX
; với giá trị  từ 0 đến 65535
DEFINE_PRINT_NUM_UNS    MACRO
;khai báo các nhãn cục bộ
LOCAL begin_print, calc, skip, print_zero, end_print, ten
LOCAL skip_proc_print_num_uns
JMP     skip_proc_print_num_uns
PRINT_NUM_UNS   PROC    NEAR
; cất các giá trị thanh ghi vào ngăn xếp
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        ; Cờ cấm in số 0 trước 1 số
        MOV     CX, 1
        ; két quả của AX/ 10000 luôn nhỏ hơn 9).
        MOV     BX, 10000       ; số chia.
        ; AX =0?
        CMP     AX, 0
        JZ      print_zero
begin_print:
        ; kiểm tra số chia (nếu là 0 thì nhảy đến nhãn end_print:
        CMP     BX,0
        JZ      end_print
      ; tránh in số 0 trước số cần in
        CMP     CX, 0
        JE      calc
        ; nếu AX<BX thì kết quả của phép chia là 0:
```

```
        CMP     AX, BX
        JB      skip
calc:
        MOV     CX, 0   ; thiết lập cờ.
        MOV     DX, 0
        DIV     BX      ; AX = DX:AX / BX   (DX=số dư).
        ; in số cuối cùng
        ; AH =0, bị bỏ qua
        ADD     AL, 30h    ; chuyển sang mã ASCII
        PUTC    AL
        MOV     AX, DX  ; lấy số dư từ phép chia cuối cùng.
skip:
        ; tính BX=BX/10
        PUSH    AX
        MOV     DX, 0
        MOV     AX, BX
        DIV     CS:ten  ; AX = DX:AX / 10   (DX=số dư).
        MOV     BX, AX
        POP     AX
        JMP     begin_print
print_zero:
        PUTC    '0'
end_print:
      ; khôi phục lại giá trị thanh ghi ban đầu
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET
ten             DW      10      ; định nghĩa số chia.
PRINT_NUM_UNS   ENDP
skip_proc_print_num_uns:
DEFINE_PRINT_NUM_UNS    ENDM
;*************************************************************
```

3.3. KẾT NỐI HỢP NGỮ VỚI CÁC NGÔN NGỮ BẬC CAO

Phần này giới thiệu cách thức kết nối một chương trình hợp ngữ với các ngôn ngữ bậc cao như C và Pascal. Việc chuyển đổi một đoạn chương trình từ ngôn ngữ bậc cao sang dạng hợp ngữ sẽ làm cho tốc độ thực hiện của chương trình sẽ được cải thiện đáng kể. Trong nhiều trường hợp, nó còn làm đơn giản cho người lập trình khi viết các đoạn chương trình liên quan đến thao tác phần cứng và các thiết bị ngoại vị thông qua các dịch vụ ở mức BIOS.

3.3.1 Ngôn ngữ C và Hợp ngữ

Để liên kết các đoạn chương trình hợp ngữ vào ngôn ngữ C hoặc Pascal thì người ta thường sử dụng một trong hai cách: sử dụng inline assembly hoặc viết tách biệt các module.

a . Sử dụng inline assembly

Chèn các khối lệnh hợp ngữ vào chương trình được viết bằng ngôn ngữ C. Đây là phương pháp nhanh và đơn giản. Người lập trình chỉ phải thêm từ khóa **asm** đứng trước mỗi lệnh. Với phương pháp này, ta có thể dễ dàng đưa các lệnh của hợp ngữ vào giữa các dòng lệnh của C.

Cú pháp đầy đủ của một dòng lệnh inline-assembly

```
asm  [<Nhãn>:] <lệnh> <các toán hạng>
```

hoặc cũng có thể dùng cả một khối lệnh hợp ngữ được gói bên trong cặp dấu {}. Trong nhiều trường hợp dạng sau được sử dụng thuận tiện hơn. Đặc biệt khi có nhiều hơn 1 lệnh hợp ngữ.

```
asm  {
[<Nhãn 1>:] <lệnh 1> <các toán hạng 1>
[<Nhãn 2>:] <lệnh 2> <các toán hạng 2>
….
[<Nhãn n>:] <lệnh n> <các toán hạng n>
}
```

Mỗi khi chương trình dịch của C gặp từ khóa asm trong dòng lệnh inline assembly thì chương trình dịch sẽ chuyển dòng lệnh hợp ngữ này vào và dịch với việc qui chiếu biến C ra dạng tương ứng của hợp ngữ để thực hiện.

Dưới đây là một ví dụ minh họa cả hai dạng cú pháp trên. Trong ví dụ này in hai xâu kí tự đã được định nghĩa sẵn lên màn hình.

Chương trình được viết theo dạng cú pháp thứ nhất

```
#include <stdio.h>
#include <conio.h>
void main()
{
 char xau1 []=”Hello World $”;
 char xau2 []=”Hello Vietnam $”;
 asm mov dx,offset xau1;
```

```
 asm mov ah,09;
 asm int 21h;
/*xuống dòng */
 asm mov ah,02;
 asm mov dl,13;
 asm int 21h;
/*về đầu dòng */
 asm mov dl,10;
 asm int 21h;
 printf ("%s", xau2);
getch();/*chờ người dùng gõ vào 1 phím*/
}
```

Chương trình được viết theo dạng cú pháp thứ hai

```
#include <stdio.h>
#include <conio.h>
void main()
{
 char xau1 []="Hello World $";
 char xau2 []="Hello Vietnam $";
 asm  {
        mov dx,offset xau
        mov ah,09
        int 21h
        /*xuống dòng */
        mov ah,02
        mov dl,13
        int 21h
        /*về đầu dòng */
        mov dl,10
        int 21h
    }
 printf ("%s", xau2);  /* in xâu 2*/
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Chú ý rằng: mọi lời giải thích sẽ phải tuân thủ theo cách của chương trình C.

Chương trình dịch C khi gặp từ khóa asm thì các biến xau1, xau2 của C sẽ được ánh xạ sang các biến tương ứng của hợp ngữ. Nghĩa là, với từ khóa asm ta có thể đặt câu lệnh hợp ngữ ở bất kỳ đâu trong đoạn mã chương trình chương t rình C.

Qua trình dịch của chương trình C có chứa các dòng lệnh hợp ngữ như sau:

- Chương trình dịch C (turbo C) sẽ dịch file chương trình nguồn (phần mở rộng .C) từ dạng .C sang dạng hợp ngữ (đuôi .asm).
- Chương trình TASM sẽ dịch tiếp file .asm sang file .obj
- Trình liên kết TLINK sẽ thực hiện việc liên kết để tạo file .exe.

Trong trường hợp chương trình chỉ chứa các lện C mà không có inline-assembly thì chương trình dịch sẽ dịch trực tiếp file nguồn C sang file .OBJ.

Các cách truy xuất biến của ngôn ngữ C;

- *Truy xuất trực tiếp:*

Các biến được khai báo trong C được coi như các biến "toàn cục" sử dung chung cho cả C và các inline- assembly. Ví dụ chương trình dưới đây tính tổng 2 số nguyên x và y rồi lưu kết quả vào biến sum.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int x,y, Sum;
 /*Nhập x và y từ bàn phím*/
 printf ("x =  "); scanf("%d",&x);
 printf ("y =  "); scanf("%d",&y);
 asm  {
       mov ax,x
       add ax,y
       mov Sum,ax
 }
 printf ("Tong la: %d", Sum);  /* in tong*/
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

- *Truy xuất gián tiếp qua thanh ghi chỉ số*:

Sử dụng một thanh ghi làm chỉ số của mảng. Ví dụ dưới đây ta tính tổng các phần tử của một mảng gồm 6 số nguyên đã được khai báo trước.

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int Sum;
     int A[]=(3,2,1,5,6,7};
 asm  {
       mov bx,offset A /*bx chỉ số của phần tử đầu tiên */
       xor ax,ax /* ax chứa tổng */
       mov cx,6 /* số phần tử */
```

```
        Cong:
           add al,[bx]
           inc bx
           loop Cong
           mov Sum,ax
     }
     printf ("Tong la: %d", Sum);  /* in tong*/
     getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

- *Truy xuất đến tham số truyền cho hàm:*

Trong cách truy xuất này, ta có thể dùng biến kiểu con trỏ (pointer) làm tham số truyền của hàm.

Ví dụ 1:

Chương trình ví dụ sau in ra 1 xâu kí tự được nhập từ bàn phím và xâu kí tự này được truyền vào một tham số của hàm InXau.

```
#include <stdio.h>
#include <conio.h>
void InXau(char *xau);
/*Chương trình con in ra một xâu kí tự*/
void InXau(char *xau)
{
 asm {
       mov ah,9
       mov dx, offset xau
       int 21h
 }
}
/*chương trình chính*/
void main()
{
 char *s1;
 /*Nhập vào 1 xâu từ bàn phím*/
 printf ("Nhap vao xau:  "); scanf("%s",&s1);
 /*In xau vừa nhập*/
 InXau(s1);
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Ví dụ 2: Viết hàm di chuyển con trỏ màn hình đến vị trí (x,y) trên màn hình (giống lệnh gotoxy(x,y) trong Pascal.

```
#include <stdio.h>
#include <conio.h>
void gotoxy(int x,int y);
```

```
/* Hàm di chuyển con trỏ màn hình đến vị trí x,y trên màn hình
*/
void gotoxy(int x,int y)
{
 asm{
        mov ax,x
        /*hoành độ lưu trong dl */
        mov dl,al
        mov ax,y
        /*tung độ lưu trong dl */
        mov dh,al
        /*đặt vị trí con trỏ*/
        mov ah,02
        mov bh,00
        int 10h  /*ngắt phục vụ màn hình*/
 }
}
/*chương trình chính*/
void main()
{
 int x=50, y=10;
 gotoxy(x,y);
 printf ("(%d,%d)",x,y);
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Ví dụ 3: Các lệnh nhảy có thể được thực hiện bên trong các hàm trong C. Dưới đây là một hàm nhận đầu vào là 1 kí tự ch, hàm sẽ kiểm tra kí tự ch có nằm trong khoảng từ [‘a’…’z’] hay không. Nếu ch thuộc khoảng (đóng) đó thì sẽ đổi kí tự ch từ thường sang hoa.

```
char upcase(char ch)
{
       asm mov al,ch; /*lưu kí tự trong al*/
       asm cmp al,'a';  /*là kí tự đứng trước 'a'*/
       asm jb khongxet;
       asm cmp al,'z'; /*là kí tự đứng sau 'z'*/
       asm ja khongxet;
       asm and al,5fh;
       khongxet:
}
```

- Các kết quả trả về từ hàm

Các kết quả trả về từ hàm được liệt kê trong bảng dưới đây:

Kiểu	Thanh ghi	Dữ liệu (byte)
char	AL	1
short int	AL	1
int	AX	2
unsigned int	AX	2
dword	DX:AX	4
pointer	DX:AX	4

- Lệnh điều khiển #pragma inline

Cú pháp: #pragma inline

Ví dụ: viết chương trình tìm giá trị nhỏ nhất trong 2 số bằng ngôn ngữ C có xen inline assembly

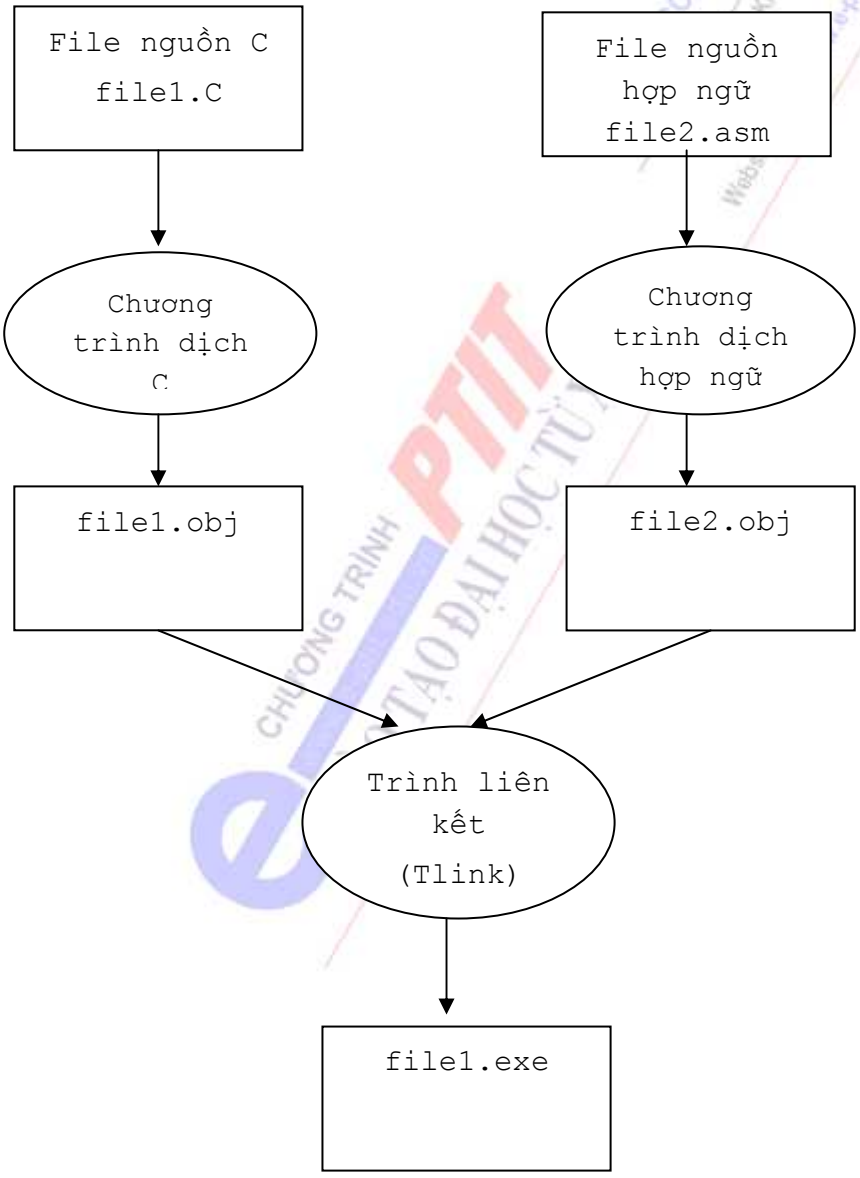
```
#pragma inline
#include <stdio.h>
#include <conio.h>
int min(int x, int y);
/*chương trình chính*/
void main()
{
 int m,n;

 /*Nhập vào 2 số từ bàn phím*/
 printf ("m= "); scanf("%d",&m);
 printf ("n= "); scanf("%d",&n);
 /*In min*/
 printf ("So be la:  %d", min(m,n));
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
int min(int x, int y);
/*Chương trình con tìm min*/
int min(int x, int y)
{
 asm {
       mov ax,m
       cmp ax,n
       jb thoat
       mov ax,n
 thoat:
```

```
        return(_ax);
  }
}
```

b . Viết tách biệt các module hợp ngữ và C

Trong phương pháp trên thì cả lệnh C và hợp ngữ cùng được chứa trong 1 file. Phương pháp trên khá nhanh và hiệu quả đối với các chương trình nhỏ (đoạn mã chương trình bé hơn 64KB). Đối với các chương trình lớn thì các module được tổ chức trong các file khác nhau. Ta có thể viết các module C và hợp ngữ hoàn toàn tách biệt, sau đó tiến hành dịch riêng rẽ từng module sau đó liên kết chúng với nhau trước khi cho chạy. Cuối cùng ta thu được một file thực hiện được (exe) bằng cách trộn các file được viết bằng C và hợp ngữ.

Dưới đây là mô tả cho phương pháp thực hiện này:



Khi ta đã soạn xong chương trình nguồn file1.C và file2.asm thì ta có thể dịch và liên kết bằng lệnh:

tcc file1 file2.asm

Lệnh dịch trên sẽ được thực hiện như sau:

- trình biên dịch turbo C dịch file1.C thành file1.asm
- trình biên dịch tcc sẽ gọi trình biên dịch tasm để dịch file2.asm thành file2.obj
- trình biên dịch tcc sẽ gọi trình liên kết Tlink để liên kết hai file file1.obj và file2.obj thành file1.exe.

Việc viết tách biệt module ra với nhau rất có lợi cho các chương trình có nhiều lệnh hợp ngữ. Không những thuận lợi cho việc bảo trì mà phương pháp này còn tận dụng tối đa khả năng của trình biên dịch hợp ngữ và tránh được các nhược điểm của inline-assembly. Tuy nhiên, để thực hiện được sự liên kết theo cách này thì khi viết các module hợp ngữ người lập trình phải bắt buộc tuân thủ tất cả các qui định của việc liên kết với module C. Đó là các vấn đề liên quan đến segment, chuyển đổi tham số, cách qui chiếu đến các biến của C, và bảo tồn các biến thanh ghi.

- *Các vấn đề cần phải giải quyết khi viết tách các module C và module hợp ngữ:*

1. Module hợp ngữ phải sử dụng sự sắp xếp các đoạn bộ nhớ (segment) tương thích với ngôn ngữ C.

Đây là vấn đề liên quan đến việc khai báo và sử dụng mô hình bộ nhớ và các đoạn bộ nhớ (segment). Dưới đây là một số lệnh điều khiển đơn giản có liên quan đến các qui định của C.

+ Lệnh điều khiển DOSSEG báo cho trình biên dịch TASM sắp xếp các đoạn bộ nhớ (segment) theo thứ tự như qui định của Intel. Ngôn ngữ C và hầu hết các ngôn ngữ bậc cao khác cũng phải sắp xếp các đoạn bộ nhớ theo qui cách này. Như vậy thứ tự sắp xếp các đoạn bộ nhớ trong module hợp ngữ cũng phải tuân thủ theo qui cách này.

+ Lệnh điều khiển MODEL báo cho trình biên dịch TASM biết kích thước mô hình bộ nhớ. Theo sau lệnh . MODEL là các kiểu mô hình bộ nhớ (*Tiny, Small, Compact, Medium, Large và Huge*) giống như việc chọn các tùy chọn trong môi trường C khi dịch. Lệnh điều khiển .MODEL còn được mặc định về dạng (NEAR hoặc FAR) của chương trình con được xây dựng bởi lệnh điều khiển PROC.

+ Các lệnh điều khiển .CODE, .DATA, .FARDATA, và .CONST của nhóm lệnh điều khiển segment đơn giản cũng tạo được những segment tương thích với C.

Ví dụ: Tính tổng của dãy số nguyên sodau + (sodau+1) +(sodau+2) + …+socuoi, với socuoi>sodau. Chương trình được tổ chức làm hai file. File hợp ngữ Tong.asm chứa đoạn chương trình tính tổng còn file ngôn ngữ C InTong.C sẽ chứa đoạn chương trình in kết quả của tổng này.

Module hợp ngữ được viết như sau:

```
.MODEL Small
.DATA
 EXTRN  _sodau: WORD
  XTRN  _socuoi: WORD
 PUBLIC Tong dw ?
.CODE
      PUBLIC _Sum
```

```
_Sum PROC
  Mov CX, _socuoi
  Sub CX, _sodau
  Inc CX ; CX chứa số lượng các số
  Mov BX,_sodau ;  BX chứa số đầu tiên
  Xor AX,AX ; AX chứa tổng
TinhTong:
  Add AX,BX
  Inc BX   ; BX= sốkế tiếp
  Loop TinhTong ; tiep tuc tinh tong neu CX<>0
  Mov Tong,AX
  Ret
_Sum ENDP
 END
```

Hàm _Sum sẽ được chương trình của C gọi từ mô hình dịch Small của turbo C với câu lệnh: Sum().

Dưới đây là module C của file InTong.C

```
extrn int sodau;
extrn int socuoi;
extrn int Sum();
void main()
{
 printf ("So dau:  "); scanf("%d",&sodau);
 printf ("So cuoi "); scanf("%d",&socuoi);
       printf ("Tong la: %d ", Sum());
}
```

Để tạo được file chạy được (đuôi exe) ta thực hiện lệnh sau (giả sử tất cả các file liên quan đều cùng nằm trong 1 thư mục với tcc và turbo C được cài đặt trên ổ C trong thư mục tc).

```
tcc -ms -Ic:\tc\include -Lc:\tc\lib InTong Tong.asm
```

Chú ý:

Nếu muốn liên kết –Sum với mô hình bộ nhớ dạng khác chẳng hạn compact thì ta phải chọn tùy chọn compact trong khi dịch bằng tcc và trong chương trình Tong.asm ta phải sửa lệnh .Model Small thành .Model Compact

Khi muốn sử dụng đoạn bộ nhớ kiểu FAR trong Tong.asm thì ta phải sử dụng lệnh điều khiển .FARDATA.

4. Các khai báo PUBLIC, EXTERNAL và sự tương thích kiểu dữ liệu

Ta đã tìm hiểu về chương trình được tổ chức thành nhiều module và được lưu trữ trên nhiều file khác nhau.. Chương trình được liên kết bằng các module của C và hợp ngữ cũng là chương trình nhiều file, do đó phải thỏa mãn các yêu cầu về khai báo nhãn (tên biến, tên hàm…) giữa các module với nhau, cụ thể là:

Trong các module viết bằng hợp ngữ phải:

Khai báo PUBLIC trước những nhãn (tên biến, tên hàm…) mà các file khác sẽ sử dụng đến bằng cú pháp:

```
PUBLIC _tên nhãn 1, _ tên nhãn 2,…
khai báo nhãn (xác định kích cỡ)
```

Ví dụ:

Với tên nhãn là biến nhớ:

```
PUBLIC _giatri1, _giatri2
giatri1 DB 10
giatri1 DW 1000
```

Với tên nhãn là tên hàm:

```
PUBLIC _Sum
_Sum PROC
   < Các lệnh trong thân hàm>
_Sum ENDP
```

Khai báo EXTRN trước những biến ngoài được file này sẽ sử dụng đến. Cú pháp như sau:

```
EXTRN _tên nhãn 1: kiểu nhãn 1,
      _tên nhãn 2: kiểu nhãn 2,…
```

Ví dụ:

Với tên nhãn là biến nhớ:

```
EXTRN _x1: BYTE, _x2: WORD
```

Với tên nhãn là hàm:

```
EXTRN _Ham: PROC
```

+ Sự tương thích giữa các kiểu về khai báo dữ liệu được cho ở trong bảng sau:

Kiểu khai báo dữ liệu trong C	Kiểu khai báo dữ liệu trong hợp ngữ
Unsigned char	Byte
Char	Byte
Enum	Word
Unsigned short	Word
Short	Word
Unsigned int	Word
Int	Word
Unsigned long	Dword
Long	Dword
Float	Dword
Double	Qword

Long double	Tword
Near	Word
Far	Dword

Ví dụ:chương trình tính giai thừa

Chương trình được tổ chức thành hai module: module C và module hợp ngữ. Mỗi module có một nhiệm vụ như sau:

Module C: đọc số cần tính giai thừa, gọi chương trình con thực hiện việc tính giai thừa và in kết quả ra màn hình. Module này được lưu trong file file1.C

```
#include <stdio.h>
#include <conio.h>
extern GiaiThua();
int number, ketqua;
/*chương trình chính*/
void main()
{
 int m,n;
 /*Nhập vào 1 số từ bàn phím*/
 printf ("Nhap vao 1 so: "); scanf("%d",&number);
 GiaiThua();
 /*In min*/
 printf ("Ket qua la:  %d", ketqua);
 getch(); /*chờ người dùng gõ vào 1 phím*/

}
```

Module hợp ngữ: tính giai thừa. Module này được lưu trong file file2.asm

```
.MODEL Small
.DATA
 EXTRN  _number: WORD,_ketqua: WORD
 temp dw ?
.CODE
      PUBLIC _GiaiThua
      _GiaiThua PROC
        Mov _ketqua, 1 ; ket qua tinh giai thua
        Mov temp, 2 ;bat dau nhan tu 1*2
        Mov CX,_number ; so cac thua so
        Dec CX
 Tinh:
        Mov AX,_ketqua ; AX chua ket qua
        Mul temp  ; nhan ket qua voi số kế tiếp
        Mov _ketqua, AX
```

```
        Inc temp
        Loop Tinh ; tiep tuc tinh giai thua neu CX<>0
        Ret
_GiaiThua ENDP
 END
```

Sau khi soạn xong, có thể tiến hành dịch và liên kết chương trình bằng lệnh:

```
tcc -ms -Ic:\tc\include -Lc:\tc\lib file1 file2.asm
```

Sau khi sửa lỗi, chương trình sẽ được dịc thành file file1.exe.

c . Một số điểm cần lưu ý

Khi viết chương trình C và hợp ngữ liên kết với nhau ta cần chú ý hai điểm:

- Bảo vệ các thanh ghi:

Một chương trình con (hàm hoặc thủ tục) viết bằng hợp ngữ được liên kết với chương trình C phải bảo tồn các thanh ghi đoạn, đó là các thanh ghi: BP,SP,CS,DS và SS. Giá trị của các thanh ghi này phải được lưu vào ngăn xếp bằng các lệnh PUSH trước các lệnh khác trong các chương trình con hoặc macro. Ở phần cuối các chương trình con (trước lệnh ret) thì các lệnh này phải được khôi phục lại bằng các lệnh POP.

- Giá trị trả lại của các hàm:

Giống ngôn ngữ C và các ngôn ngữ khác, các hàm được xây dựng bằng bằng hợp ngữ khi liên kết với C cũng có thể trả về một giá trị (tên hàm mang một giá trị). Xong các giá trị trả về của các hàm được viết bằng hợp ngữ tuân thủ các qui định sau:

Kiểu giá trị trả về	Nơi chứa giá trị trả về
Unsigned char	AX
Char	AX
Enum	AX
Unsigned short	AX
Short	AX
Unsigned int	AX
Int	AX
Unsigned long	DX:AX
Long	DX:AX
Float	Đỉnh ngăn xếp 8087, thanh ghi ST(0)
Double	Đỉnh ngăn xếp 8087, thanh ghi ST(0)
Long double	Đỉnh ngăn xếp 8087, thanh ghi ST(0)
Near	AX
Far	DX:AX

d. Một số ví dụ về truyền tham số giữa các hàm của C và hợp ngữ.

Ví dụ 1: Viết chương trình tính giai thừa với yêu cầu: kết quả của hàm tính giai thừa là một đối số ra của hàm (chứ không phải là giá trị trả lại của hàm [giống ví dụ trong phần b, 2 của mục 3.3.1]).

Module C: đọc số cần tính giai thừa, gọi chương trình con thực hiện việc tính giai thừa. Lấy và in kết quả (từ đối số của hàm) ra màn hình. Module này được lưu trong file file1.C

```
#include <stdio.h>
#include <conio.h>
extern GiaiThua(int number, int near *ketqua);
/*chương trình chính*/
void main()
{
 int n, kq;
 /*Nhập vào n từ bàn phím*/
 printf ("Nhap vao 1 so: "); scanf("%d",&n);
 GiaiThua(n,&kq);
 /*In min*/
 printf ("Ket qua la:  %d", ketqua);
 getch(); /*chờ người dùng gõ vào 1 phím*/
}
```

Module hợp ngữ: tính giai thừa và lưu kết quả vào đối của hàm. Module này được lưu trong file file2.asm

```
.MODEL Small
.DATA
 Gtri DW ?
 temp dw ?
.CODE
      PUBLIC _GiaiThua
      _GiaiThua PROC
        ARG  _number: WORD,_ketqua: WORD
        Push BP    /* bao ve gia tri thanh ghi BP */
        Mov BP,SP  /*BP,SP tro den dau stack*/
        Mov Gtri, 1 ; ket qua tinh giai thua
        Mov temp, 2 ;bat dau nhan tu 1*2
        Mov CX,_number ; so cac thua so
        Dec CX
 Tinh:
        Mov AX, Gtri ; AX chua ket qua
        Mul temp  ; nhan ket qua voi số kế tiếp
        Mov Gtri, AX
        Inc temp
        Loop Tinh ; tiep tuc tinh giai thua neu CX<>0
        Mov BX, _ketqua
```

```
        Mov [BX],AX
        Pop Bp
        Ret
_GiaiThua ENDP
        END
```

Sau khi soạn xong, có thể tiến hành dịch và liên kết chương trình bằng lệnh:

```
tcc –ms –Ic:\tc\include –Lc:\tc\lib file1 file2.asm
```

Sau khi sửa lỗi, chương trình sẽ được dịc thành file file1.exe.

3.3.2 Ngôn ngữ Pascal và Hợp ngữ

Về nguyên lý, thì liên kết giữa hợp ngữ với Pascal giông như việc liên kết giữa hợp ngữ với C. tuy nhiên cũng có một số qui tắc riêng khi thực hiên liên kết giữa hợp ngữ và Pascal.

Cũng như hợp ngữ và C, có hai cách để liên kết giữa hợp ngữ và Pascal là dùng inline assembly và viết tách biệt giữa các module hợp ngữ và module Pascal.

a . Sử dụng inline assembly trong Pascal

Phương pháp này thích hợp cho người lập trình phát triển các chương trình nhỏ. Trong phương pháp này người lập trình sẽ chèn một khối lệnh hợp ngữ vào một chương trình được viết bằng ngôn ngữ Pascal. Đây là phương pháp khá đơn giản và nhanh.

```
 asm
[<Nhãn 1>:] <lệnh 1> <các toán hạng 1>
[<Nhãn 2>:] <lệnh 2> <các toán hạng 2>
….
[<Nhãn n>:] <lệnh n> <các toán hạng n>
end;
```

Khi các chương trình dịch của Pascal gặp từ khóa asm trong dòng lệnh inline-assembly thì chương trình dịch sẽ chuyển khối lệnh hợp ngữ này vào và dịch với việc qui chiếu biến Pascal ra dạng tương ứng của hợp ngữ để thực hiện.

Dưới đây là một ví dụ đơn giản để minh họa:

Ví dụ 1: viết chương trình tìm giá trị nhỏ nhất (min) cho hai số bằng ngôn ngữ Pascal có chèn các dòng lệnh dạng inline-assembly. Giả sử file chương trình là Min.pas

```
Program Min;
Uses crt;
Var
 m,n: integer;
 function min(int x, int y): integer;
  /*Chương trình con tìm min*/
 begin
  asm
                mov ax,x;
                cmp ax,y;
                jb thoat;
                mov ax,y;
```

```
                mov x,ax;
          thoat:
           min=x;
     end;
   end;
Begin
   Clrscr;
  /*Nhập vào 2 số từ bàn phím*/
  write ("m= "); readln(m);
  write ("n= "); readln(n);
  /*In min*/
  write ("So be la:  ", min(m,n));
  readln(); /*chờ người dùng gõ vào 1 phím*/
End.
```

Để thực hiện chương trình ta gõ lệnh:

```
tpc  Min.pas <enter>
```

Chương trình sẽ được dịch ra file exe.

b .Viết tách biệt nhiều module hợp ngữ và Pascal riêng rẽ

Giống như viết tách biệt module và hợp ngữ, khi viết tách biệt giữa Pascal và hợp ngữ cũng phải xử lý một số vấn đề tương tự như: các lệnh điều khiển dịch, các vấn đề liên kết thông tin qua các biến, bảo vệ và khôi phục giá trị các thanh ghi đoạn và sự tương thích về kiểu dữ liệu.

Trong các vấn đề trên, hầu hết các vấn đề đều được giải quyết tương tự như việc đối với C và hợp ngữ. Sự tương thích về kiểu dữ liệu có đôi chút khác biệt, dưới đây là bảng tương thích kiểu dữ liệu giữa hợp ngữ và Pascal.

Kiểu khai báo dữ liệu trong Pascal	Kiểu khai báo dữ liệu trong hợp ngữ
Byte	Byte
Word	Word
Shortint	Byte
Integer	Word
Real	Fword
Single	DWord
Double	QWord
Extended	TByte
Comp	Qword
Pointer	Dword

Ví dụ: Tính tổng các phần tử trong một dãy số nguyên dương khi biết số đầu tiên và số các phần tử cần tính.

Chương trình được thành hai module. Module hợp ngữ có nhiệm vụ tính tổng của các phần tử. Module này được lưu vào file sum.asm

```
.MODEL Small
.DATA
 EXTRN  _sodau: WORD,_sophantu: WORD
.DATA ?
 Tong dw ?
.CODE
       PUBLIC Sum
       Sum PROC
        Mov CX, [sophantu]
        Mov AX, [sodau]
        Mov [tong],AX
 TinhTong:
        Inc AX
        Add [tong],AX
        Loop TinhTong ; tiep tuc tinh tong neu CX<>0
        Mov AX, Tong
        Ret
  Sum ENDP
        END
```

Module Pascal được lưu trong file InTong.pas

```
Program Tong;
Uses crt;
{F+}
Var
 Sodau: integer;
 Sophantu: integer;
 function Sum: integer; external;
 {$I sum.obj}
{F-}
Begin
 Clrscr;
 write ("So dau:  "); readln(sodau);
 write ("So phan tu:  "); readln(sophantu);
       write ("Tong la:  ", Sum);
End.
```

Để tạo ra file chạy (.exe) ta tiến hành qua các bước sau:

- Dịch module hợp ngữ (để tạo file sum.obj)

```
Tasm sum <Enter>
```

- Dịch module Pascal có lên kết với file sum.obj

```
Tpc –ml Intong <enter>
```

3.4 CÁC CHƯƠNG TRÌNH NGẮT

Ngắt là một cơ chế yêu cầu CPU tạm dừng công việc (task) đang thực hiện để thực hiện 1 công việc khác. Nói cụ thế hơn, ngắt yêu cầu CPU tạm dừng chương trình đang thực hiện để thực hiện một chương trình con phục vụ ngắt.

Người ta tạm chia ngắt ra làm hai loại: ngắt cứng và ngắt mềm. Các ngắt mềm được kích hoạt bằng lệnh INT n trong đó n là số hiệu ngắt dưới dạng một số hexa. Ngắt cứng khác vơi ngắt mềm ở chỗ không được kích hoạt bằng một lệnh INT n trong chương trình mà được kích hoạt bằng các tác động của các tín hiệu linh kiện điện tử như bàn phím, ổ đĩa,..

Phần này giới thiệu về các ngắt và các dịch vụ ở mức BIOS và mức hệ điều hành DOS và cách viết chương trình thường trú và chương trình con ngắt.

3.4.1 Ứng dụng các ngắt của BIOS & DOS

Máy tính có 256 ngắt được đánh số hiệu từ 00h đến FFh. Trong đó các ngắt có số hiệu từ 00h đến 1Fh là các ngắt của BIOS, còn các ngắt còn lại từ 20h đến FFh là các ngắt của DOS.

Dưới đây ta sẽ tìm hiểu các ngắt theo từng nhóm ngắt.

a. Các ngắt của BIOS & DOS

Địa chỉ	Số hiệu ngắt	Chức năng
Các ngắt phục vụ hệ thống		
0-3	0	CPU: chia cho 0
4-7	1	CPU: chạy từng bước của DEBUG
8-B	2	CPU: ngắt NMI (hiện thông báo halt)
C-F	3	CPU: thực hiện đến điểm dừng (break point)
10-13	4	CPU: tràn số (overflow)
14-17	5	In nội dung ra mà hình
18-1B	6	Phục vụ liên lạc
1C-1F	7	Dự trữ
Các ngắt cứng		
20-23	8	IRQ0: CLK (18.2 lần/s) nối từ chip 8253
24-27	9	IRQ1: bàn phím
28-2B	A	IRQ2: đầu vào của 8259 thứ 2
2C-2F	B	IRQ3: giao diện nối tiếp

30-33	C	IRQ4:giao diện nối tiếp
34-37	D	IRQ5: thường nối với máy in nối tiếp
38-3B	E	IRQ6: phục vụ đĩa mềm
3C-3F	F	IRQ5: thường nối với máy in song song
Các ngắt thực sự đặc trưng cho BIOS		
40-43	10	Màn hình (I/O video)
44-47	11	Xác định cấu hình
48-4B	12	Cho biết kich cỡ của RAM
4C-4F	13	Thâm nhập đĩa mềm, đĩa cứng.
50-53	14	Giao diện nối tiếp
54-57	15	Giao diện với cassete
58-5B	16	Kiểm tra bàn phím
5C-5F	17	Truy nhập máy in song song
60-63	18	Gọi BASIC trong ROM
64-67	19	Khởi động nóng hệ thống (Ctrl+Alt+Del)
68-6B	1A	Thông báo thời gian
6C-6F	1B	Quản lý phím Ctrl+Break
70-73	1C	Dành cho đồng hồ
74-77	1D	Địa chỉ bảng tham số cho màn hình
78-7B	1E	Cho biết các tham số của đĩa mềm
7C-7F	1F	Địa chỉ các bảng font các kí tự mở rộng
Các ngắt của DOS		
80-83	20	Kết thúc chương trình dạng COM
84-87	21	Các hàm của DOS
88-8B	22	Địa chỉ kết thúc chương trình
8C-8F	23	Địa chỉ thủ tục Ctrl+Break
90-93	24	Báo lỗi đĩa
94-97	25	Đọc đĩa mềm, đĩa cứng
98-9B	26	Ghi đĩa
9C-9F	27	Kết thúc chương trình và thường trú
A0-A3	28	Dành cho các hàm không được DOS cung cấp dữ

		liệu
	29-3F	Dự trữ
	40	BIOS phục vụ đĩa mềm
	41	Địa chỉ của bảng đĩa cứng 1
	42-49	Dự trữ
	4A	Hẹn giờ
	4B-6F	Dự trữ
	70-77	Ngắt cứng của 8259 thứ 2
	78-7F	Dự trữ
	80-F0	Dùng cho bộ thông dịch BASIC
	F1-FF	Dự trữ

b. Cơ chế hoạt động khi một ngắt được kích hoạt

Khi có một yêu cầu ngắt số hiệu N đến chân CPU và nếu yêu cầu ngắt này được CPU đáp ứng Khi đó CPU sẽ thực hiện các công việc sau:

1. Cất nội dung của thanh ghi cờ (FR) vào đỉnh của ngăn xếp. (Bằng việc tự động thực hiện câu lệnh PUSHF).

2.Cấm các ngắt khác tác động vào CPU để CPU chạy ở chế độ bình thường. Đặt các cờ Ì=0 và TF =0 bằng cách thực hiện các lệnh: CLI và CLT.

3. Cất địa chỉ đoạn (segment) của chương trình gọi chương trình ngắt vào ngăn xếp bằng lệnh PUSH CS.

4. Cất địa chỉ lệch (offset) của lệnh kế tiếp của chương trình gọi chương trình ngắt vào ngăn xếp PUSH IP.

5. Lấy địa chỉ mới của chương trình con phục vụ ngắt số hiệu N trong bảng vector ngắt bằng cách lấy địa chỉ offset và segment của ngắt N từ bảng vector ngắt.

IP=[N*4]

CS=[N*4+2]

6. Khi gặt lệnh cuối cùng của chương trình con phục ngắt (lênh IRET). Bộ vi xử lý sẽ quay lại chương trình gọi ngắt tại địa chỉ trả về và khôi phục các giá trị của các thanh ghi từ ngăn xếp bằng các lệnh sau:

POP IP

POP CS

POPF

Giải thích cho mục 5. Ta biết rằng các địa chỉ của chương trình con phục vụ ngắt được lưu vào trong một bảng có kích thước 1K từ địa chỉ 0000h đến 03FFh của bộ nhớ RAM. Bảng vector ngắt lưu địa chỉ của 256 chương trình con phục vụ ngắt và mỗi địa chỉ chiếm 4 byte trong đó 2

byte dành cho địa chỉ đoạn (segment) và 2 byte dành cho địa chỉ lệch (offset). Như bảng ở trên, địa chỉ của chương trình con phục vụ ngắt 0 chiếm byte 0-3, ngắt 1 chiếm byte 4-7 … và chương trình con phục vụ ngắt thứ N sẽ có địa chỉ 4*N. Trong đó 2 byte [4*N] và [4*N+1] là địa chỉ lệch (offset) và 2 byte [4*N+2] và [4*N+3] là địa chỉ đoạn (segment).

c. Các ngắt của BIOS và DOS phục vụ bàn phím

- Ngắt 16h của BIOS

Hàm 0h:

Ý nghĩa: Chờ đọc một kí tự từ bàn phím (nếu có kí tự trong vùng đệm bàn phím thì sẽ nhận được ký tự đó, còn không thì chờ đến khi bàn phím được nhấn.

Đầu vào: AH=0

Int 16h

Đầu ra: Nếu AL<>0 thì

AL chứa mã ASCII của ký tự

AH chứa mã SCAN của ký tự

Nếu AL= 0thì

AL chứa mã bàn phím mở rộng

Hàm 1h:

Ý nghĩa: Kiểm tra xem trong vùng đệm của bàn phím có ký tự hay không (không đợi đến khi ký tự có trong vùng đệm mà trả ngay điều khiển lại cho chương trình)?.

Đầu vào: AH=01

Int 16h

Đầu ra: Nếu ZF=1 không có ký tự trong vùng đệm bàn phím

Nếu ZF=0 thì:

Nếu AL<>0 thì:

AL chứa mã ASCII của ký tự

AH chứa mã SCAN của ký tự

Nếu AL= 0 thì:

AL chứa mã bàn phím mở rộng

Hàm 02h:

Ý nghĩa: Kiểm tra trạng thái một số phím đặc biệt của bàn phím (Insert, Caplock, NumLock, Scroll Lock).

Đầu vào: AH=02

Int 16h

Đầu ra:

AL chứa kết quả các trạng thái hay cờ bàn phím , có ý nghĩa như sau:

7	6	5	4	3	2	1	0
1: chế độ Insert	1: chế độ Cap Lock	1: Num Lock bị ấn	1: Scroll Lock bị ấn	1: Alt bị ấn	1: Ctrl bị ấn	1: Shift trái bị ấn	1: Shift phải bị ấn

Ví dụ: Viết chương trình thiết lập mật khẩu là kí tự A thi khởi động máy

```
.MODEL small
.STACK 100h
.DATA
      matkhau db 'P','$'
      Saimatkhau db 'Sai mat khau ','$'
      Nhapmatkhau db 'Nhap mat khau: ','$'
      xuongdong db 13,10,'$'
.CODE
Start:
      Mov AX,@Data
      Mov DS,AX
Lap:
      Mov  AH,9
      Mov DX, offset Nhapmatkhau
 Int 21h ; in lời mời nhập xâu
 Mov AH,0  ; Nhap ki tu
     Int 16h
      Cmp AL,matkhau ; co phai Enter khong?
      JZ Done ; Neu là Enter, dung lai
      Mov  AH,9
      Mov DX, offset Saimatkhau
      Int 21h ;xuong dong va ve dau dong
      Jmp Lap
Done:
      Mov AH,4Ch ; Tro ve DOS
      Int 21h
End Start
```

Sau khi dịch và hợp dịch chương trình trên ta đặt tên chương trình vào cuoi file autoexec.bat. Khi máy khởi động thf chương trình trên sẽ được tự dộng thực hiện.

- Một số hàm phục vụ bàn phím của ngắt 21h của DOS

Hàm 06h:

Ý nghĩa: Đọc một kí tự từ bàn phím hoặc đưa kí tự ra màn hình. Nếu đọc vào một kí tự thì

Đầu vào: AH=6

Int 21h

DL=0FFh (nếu DL<>0FFh sẽ đưa ra màn hình)

Đầu ra: Nếu ZF=0 thì có kí tự trong vùng đệm bàn phím và:

AL chứa mã ASCII của ký tự

AH chứa mã SCAN của ký tự

Nếu ZF= 1 thì

Vùng đệm bàn phím rỗng

Hàm 07h:

Ý nghĩa: Chờ đọc một kí tự từ bàn phím

Đầu vào: AH=07

Int 21h

Đầu ra: AL chứa mã ASCII của ký tự (AL=0 sẽ không có ký tự nào)

AH chứa mã SCAN của ký tự

Hàm 0Bh:

Ý nghĩa: Đọc trạng thái bộ đệm bàn phím

Đầu vào: AH=0B

Int 21h

Đầu ra: AL =0FFh có kí tự trong bộ đệm

AL =00h không có kí tự trong bộ đệm

Hàm 0Ch:

Ý nghĩa: xóa bộ đệm bàn phím, sau đó gọi hàm vào kí tự có số chức năng đặt trong AL

Đầu vào: AL =số hàm của kí tự.

d. Ngắt của BIOS phục vụ màn hình – ngắt 10h

Màn hình làm việc ở một trong hai chế độ: văn bản (text) và đồ họa (graphics).

Ở chế độ văn bản, các kí tự được trình bày trong các ma trận điểm 5x7 với 25 dòng và 80 cột. Màn hình là hình ảnh của video RAM. Do vậy ở chế độ text một trang màn hình cần tối thiểu là 25 dòng x 80 cột x 2 (1 byte mã ASCII và 1 byte thuộc tính kí tự) =4000 bytes. Byte thuộc tính có dạng như sau:

c	Red	Green	Blue	i	Red	Green	Blue
Nhấp nháy	Màu nền			Đậm nhạt	Màu chữ		

Dưới đây là một số giá trị thường dùng của thuộc tính:

Giá trị	Vỉ màu
00	Không hiển thị
01	Kí tự bình thường
07	Kí tự bình thường
09	In đậm
70	Nghịch ảnh
81	Nhấp nháy
87	Kí tự bình thường và Nhấp nháy
F0	Nghịch ảnh và Nhấp nháy

Dưới đây liệt kê một số chức năng của BIOS về chế độ văn bản của màn hình.

Hàm 00h:

Ý nghĩa: Đặt chế độ cho màn hình

Đầu vào: AH=00

AL = chế độ màn hình

Int 10h

Trong đó chế độ màn hình = 0: 40 x 25 trắng đen.

= 1: 40 x 25 16 màu..

= 2: 80 x 25 trắng đen (card màu).

= 3: 80 x 25 16 màu.

= 7: 80 x 25 trắng đen (card mono).

Hàm 01h:

Ý nghĩa: Đặt kích thước con trỏ

Đầu vào: AH=01

CH = tọa độ hàng

CL = tọa độ cột

Int 10h

Hàm 02h:

Ý nghĩa: Đặt vị trí con trỏ

Đầu vào: AH=02

BH = số trang màn hình

DH=số dòng

DL = số cột

Int 10h

Hàm 03h:

Ý nghĩa: Đọc vị trí con trỏ

Đầu vào: AH=03

BH = số trang màn hình

Int 10h

Đầu ra: DH=số dòng

DL = số cột

CH= tọa độ hàng của con trỏ

CL = tọa độ cột của con trỏ

Hàm 05h:

Ý nghĩa: Đặt trang màn hình hoạt động

Đầu vào: AH=05

BL = số trang màn hình

Int 10h

Hàm 06h:

Ý nghĩa: Cuộn màn hình lên (dùng để xác lập vùng cửa sổ văn bản hình chữ nhật)

Đầu vào: AH=06

AL=số trang để trắng hoặ dòng cuộn (AL=0 để trắng toàn màn hình)

(CH,CL) = tọa độ trên bên trái màn hình

(DH,DL) = tọa độ dưới bên phải màn hình

BH= thuộc tính của vùng để trống của màn hình.

Int 10h

Hàm 07h:

Ý nghĩa: Cuộn màn hình xuống

Đầu vào: AH=07

AL=số trang để trắng hoặ dòng cuộn (AL=0 để trắng toàn màn hình)

(CH,CL) = tọa độ trên bên trái màn hình

(DH,DL) = tọa độ dưới bên phải màn hình

BH= thuộc tính của vùng để trống của màn hình.

Int 10h

Hàm 08h:

Ý nghĩa: Đọc kí tự và thuộc tính của nó tại vị trí con trỏ

Đầu vào: AH=08

BH=số trang

Int 10h

Đầu ra: AL =mã ASCII của kí tự

BL= thuộc tính của kí tự.

Hàm 09h:

Ý nghĩa: Viết các kí tự và thuộc tính vào vị trí con trỏ đang đứng (vị trí con trỏ không đổi). Đưa kí tự ra, đặt màu cho kí tự.

Đầu vào: AH=09

BH= số trang màn hình

CX = số lần kí tự được đưa ra màn hình

AL = mã ASCII của kí tự

BL= thuộc tính của kí tự.

Int 10h

Hàm 0Ah:

Ý nghĩa: Viết các kí tự không có thuộc tính vào vị trí con trỏ đang đứng (vị trí con trỏ chuyển sang phải). Không đặt màu cho kí tự.

Đầu vào: AH=0Ah

BH=số trang màn hình

CX = số lần kí tự được đưa ra màn hình

AL =mã ASCII của kí tự

Int 10h

Hàm 0Eh:

Ý nghĩa: Viết các kí tự theo kiểu teletype ra màn hình (vị trí con trỏ chuyển sang phải).

Đầu vào: AH=0Eh

BH=số trang màn hình

BL = màu của kí tự

AL =mã ASCII của kí tự

Int 10h

Hàm 0Fh:

Ý nghĩa: Lấy kiểu màn hình hiện hành.

Đầu vào: AH=0Fh

Int 10h

Đầu ra:

AH=số cột của màn hình

BH = số trang

AL =chế độ hiện thời của màn hình

Hàm 13h:

Ý nghĩa: Hiển thị một dãy kí tự.

Đầu vào: AH=13h

BH=số trang màn hình

DL= số cột bắt đầu hiển thị

DH= số dòng bắt đầu hiển thị

ES:BP =địa chỉ đầu của vùng nhớ chứa dãy kí tự cần hiển thị

CX=độ dài của dãy kí tự.

Int 10h

Ở chế độ đồ họa có thêm một số hàm sau:

Hàm 0h:

Ý nghĩa: Chọn kiểu màn hình.

Đầu vào: AH=00h

AL = 0Dh: 320 x 200, 16 màu

= 0Eh: 640 x 200 16 màu..

= 0Fh: 640 x 350, trắng đen.

= 10h: 640 x 350 16 màu.

= 11h: 640 x 480 2 màu.

= 12h: 640 x 480 16 màu.

= 13h: 320 x 200 256 màu.

(chỉ với card VGA)

Int 10h

Hàm 0Bh:

Ý nghĩa: Chọn bộ màu.

Đầu vào: AH=0Bh

BH=0: chọn màu cho nền BL=0-15

=1: chọn bộ màu cho điểm.

Int 10h

Hàm 0Ch:

Ý nghĩa: Hiển thị một điểm.

Đầu vào: AH=0Ch

DX=số hàng

CX=số cột

AL = số màu của điểm.

BH=số trang màn hình

Int 10h

Hàm 0Dh:

Ý nghĩa: Đọc thông tin của một điểm.

Đầu vào: AH=0Ch

DX=số hàng

CX=số cột

BH=số trang màn hình

Int 10h

Đầu ra: AL = số màu của điểm.

d. Ngắt của BIOS và DOS phục vụ ổ đĩa – ngắt 13h

Mỗi sector trên đĩa sẽ chứa các đặc trưng (directory) của các file. Mỗi đặc trưng của một file gồm 32 byte chứa các thông tin sau:

Byte	Nội dung
0h-7h	Tên file
8h-0Ah	Phần mở rộng
0Bh	Thuộc tính của file
0Ch-15h	Chưa dùng đến
16h-17h	Giờ của lần thay đổi cuối cùng
18h-19h	Ngày của lần thay đổi cuối cùng
1Ah-1Bh	Chứa số ô của bảng FAT
1Ch-20h	Chứa kích thước file

Byte thuộc tính có cấu trúc như sau:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		=1: lưu trữ	=1: thư mục con	=1: tên nhãn	=1: hệ thống	=1: thuộc tính ẩn	=1: chỉ đọc

Dưới đây là các chức năng của ngắt 13h, ngắt của BIOS phục vụ ổ đĩa.

Hàm 0h:

Ý nghĩa: Reset lại ổ đĩa mềm, chỉ nên gọi hàm này khi gặp lỗi trong khi truy cập đĩa bằng 1 trong 6 chức năng của ngắt 13h.

Đầu vào: AH=0h

Int 13h

Đầu ra: AH= trạng thái lỗi

Hàm 01:

Ý nghĩa: Cho biết trạng thái đĩa

Đầu vào: AH=01h

DL=số ổ đĩa

Int 13h

Đầu ra: AH= trạng thái.

Hàm 02

Ý nghĩa: Đọc một hay nhiều sector

Đầu vào: AH=02h

DL=số ổ đĩa (0-3)

DH=mặt đĩa (0: mặt trên -1:mặt dưới)

CL=sector đầu cần đọc

CH=rãnh chứa sector đầu tiên cần đọc.

AL=số lượng sector cần đọc

ES:BX =địa chỉ vùng nhớ chứa thông tin đọc được.

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Nếu cờ CF=0 thì AL= số sector đọc được.

Hàm 03:

Ý nghĩa: Ghi dữ liệu lên đĩa

Đầu vào: AH=03h

DL=số ổ đĩa (0-3)

DH=mặt đĩa (0: mặt trên -1:mặt dưới)

CL=sector đầu cần đọc

CH=rãnh chứa sector đầu tiên cần đọc.

AL=số lượng sector cần đọc

ES:BX =địa chỉ vùng nhớ cần ghi lên đĩa.

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Nếu cờ CF=0 thì AL= số sector ghi thành công.

Hàm 04:

Ý nghĩa: Kiểm tra CRC (kiểm tra dư thừa vòng); không so sánh dữ liệu trên đĩa với dữ liệu trong vùng nhớ mà chỉ kiểm tra CRC.

Đầu vào: AH=04h

DL=số ổ đĩa (0-3)

DH=mặt đĩa (0: mặt trên -1:mặt dưới)

CL=sector đầu cần đọc

CH=rãnh chứa sector đầu tiên cần đọc.

AL=số lượng sector cần đọc

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Nếu cờ CF=0 thì thành công.

Hàm 05:

Ý nghĩa: Tạo khuôn dạng (format) cho đĩa .

Đầu vào: AH=05h

AL=số lượng sector cần tạo trên 1 rãnh.

CH=số thứ tự của rãnh cần tạo (0-39 hoặc 0-79).

DH=số thứ tự của mặt đĩa (0,1).

ES:BX =trỏ đến một bảng chứa các tham số sau:

Byte 1: rãnh cần tạo khuôn.

Byte 2: mặt đĩa (0-trước, 1-sau).

Byte 3: số thứ tự của sector.

Byte 4: số byte của sector.

Ngoài ra, phải thêm thông tin nằm trên bảng tham số đĩa mềm gồm 11 byte.

Int 13h

Đầu ra: Nếu cờ CF=1 thì AH= mã lỗi.

Hàm 15h:

Ý nghĩa: Xác định loại ổ đĩa.

Đầu vào: AH=15h

DL=số ổ đĩa (0-3)

Int 13h

Đầu ra:

AH=kiểu ổ đĩa

= 0: không có ổ đĩa

= 1: ổ đĩa không phát hiện được sự thay đổi ổ đĩa

= 2: ổ đĩa phát hiện được sự thay đổi ổ đĩa

= 3: ổ đĩa cứng

Hàm 16h:

Ý nghĩa: Kiểm tra có sự thay đổi đĩa hay không.

Đầu vào: AH=16h

DL=số ổ đĩa (0-3)

Đầu ra:

AH= kết quả

= 0: đĩa chưa thay đổi

= 6: đĩa đã thay đổi sau lần truy cập cuối cùng.

3.4.2 Chương trình thường trú và chương trình ngắt

a. Chương trình thường trú

- *Khái niệm về chương trình thường trú*

Chương trình thường trú (Terminate and Stay Resident- TSR) là chương trình có thể chạy "sau" chương trình khác, hỗ trợ khả năng kích hoạt, khả năng nằm lại bộ nhớ sau khi chạy xong. Sau đó khi ta chạy một chương trình khác với một điều kiện nào đó nó sẽ được kích hoạt để hoạt động trở lại.

Với chương trình bình thường khi chạy sẽ được một chương trình tải (Program Loader trong command.com) nạp vào vùng nhớ do DOS cấp phát. Khi chương trình thực hiện xong thì vùng nhớ đã cấp phát cho nó được giải phóng và DOS sẽ đánh dấu lại vùng nhớ này để cấp phát cho chương trình khác. Với chương trình thường trú thì bước cuối cùng không xảy ra, chương trình thường trú làm cho DOS đánh dấu lại miền dành cho DOS và vùng bị nó chiếm, do vậy sau này DOS sẽ không cấp phát vùng nhớ này cho chương trình khác, và như vậy nó được bảo vệ chống bị viết đè bởi chương trình khác, bằng cách này thì chương trình thường trú trở thành một "bộ phận" của DOS.

Chỉ có file dạng COM với cấu trúc nằm gọn trong một đoạn mới dễ dàng trở thành chương trình thường trú. Ngoài hợp ngữ, người ta có thể viết chương trình thường trú trên các ngôn ngữ lập trình bậc cao khác như ngôn ngữ C, Pascal …

- *Viết chương trình thường trú*

Chương trình thường trú được viết giông như chương trình thông thường và thêm một đoạn mã của chương trình thường trú vào vùng nhớ dành cho DOS, đoạn mã đó sẽ không được kích hoạt nếu không được trao điều khiển. Việc thêm một đoạn mã tiếp dau vùng dành cho DOS được thực hiện bằng các chương tình con phục vụ ngắt. Đó là ngắt số 27H hoặc hàm 31H của ngắt 21H.

Có hai cách để làm cho chương trình thường trú được kích hoạt là: dùng ngắt và ấn một tổ hợp phím (hot-key).

Các chương trình thường trú thường sửa nội dung của vector ngắt trong bẳng vector ngắt để làm cho nó trỏ đến địa chỉ của mình trong bộ nhớ, nhờ thế mỗi khi ngắt tương ứng được gọi thì chương trình thường trú lại được trao điều khiển. việc làm này được gọi là chặn vector ngắt. Chẳng hạn, nhiều chương trình POP-UP thường sửa ngắt bàn phím (vector ngắt bàn phím (số 9) nằm tại địa chỉ: 0:0024h) làm cho nó trỏ đến địa chỉ của mình và cất địa chỉ của INT 9h để cho nó làm nhiệm vụ khi càn thiết.

- *Các bước viết chương trình thường trú*

Bước 1: Lấy và đặt lại vector ngắt bằng các dịch vụ của DOS

Ta luôn cần đến các chương trình con phục vụ ngắt đã có của hệ thống để không phải viết lại trong chương trình của mình đoạn chương trình đã có, vì vậy cần lấy nội dung của vector ngắt cũ cất vào miền dữ liệu của chương trình thường trú, khi nào cần sẽ trả lại giá trị này cho vector ngắt mà chương trình thường trú đã thay đổi. Giá trị của một vector ngắt là 2 từ tương ứng với địa chỉ đoạn (CS) và địa chỉ lệch (IP) của chương trình con phục vụ ngắt tương ứng.

+ Lấy vector ngắt (Get Interrupt Vector)

Hàm 35h:

Ý nghĩa: Lấy địa chỉ của một ngắt từ bảng vector ngắt.

Đầu vào: AH=35h

AL=số hiệu vector ngắt

Int 21h

Đầu ra: ES:BX = giá trị của vector ngắt.

Ví dụ: Đoạn chương trình sau lấy vector ngắt của bàn phím (INT 9h)

```
MODEL Tiny
.CODE
```

```
            Org 100h
           Jmp Load_Prog
            ; vùng dữ liệu
            SohieuNgat EQU 9h
            NgatCu DW 2 DUP(0); luu dia chi ngat cu
      Load_Prog PROC
        Mov AH,35
        Mov AL, SohieuNgat
        Int 21h
        Mov NgatCu,BX ; lay dia chi  lech
        Mov NgatCu[2],ES ; lay dia chi doan
        ….
 Load_Prog ENDP
```

+ Đặt giá trị cho một vector ngắt (Set Interrupt Vector)

Ý nghĩa: Đặt lại địa chỉ của vector ngắt .

Đầu vào: AH=25h

AL=địa chỉ vector ngắt

Int 21h

DS:DX = Địa chỉ của chương trình ngắt.

Ví dụ: Đoạn chương trình sau đặt lại địa chỉ của vector ngắt.

```
MODEL Tiny
.CODE
            Org 100h
           Jmp Load_Prog
            ; vùng dữ liệu
            SohieuNgat EQU 9h
            NgatCu DW 2 DUP(0); luu dia chi ngat cu
      Load_Prog PROC
        Mov AH,35h
        Mov AL, SohieuNgat
        Int 21h
        Mov NgatCu,BX ; lay dia chi  lech
        Mov NgatCu[2],ES ; lay dia chi doan
        Mov AH,25
        Mov DX,offset Prog ; Dat vector ngat moi, tro vao PROG
        Int 21h
        ….
 Load_Prog ENDP
```

Sau đó, mỗi lần ngắt bàn phím được kích hoạt thì chương trình PROG của ta sẽ được thực hiện. Tất nhiên sau khi thực hiện chương trình PROG này thì ta phải đặt đặt lại điạc chỉ của vector ngắt bàn phím.

Bước 2: Làm cho chương trình ở lại thường trú

Có thể làm cho chương trình ở lại thường trú bằng cách sử dụng INT 27H hoặc dịch vụ 31H của INT 21H. Trong phần này ta chỉ xem xét việc sử dụng INT 27H.

Nói chung tất cả các chương trình thường trú đều tự nạp ns vào bộ nhớ sau đó tự loại bỏ phần "đuôi" của mình- đó là đoạn mã thực hiện nạp chương trình vào bộ nhớ hay còn được gọi là phần tạm trú. Dưới đây là khung của chương trình thường trú bằng cách chặn ngắt.

```
MODEL Tiny
.CODE
        Org 100h
        Jmp Load_Prog
              ; vùng dữ liệu
        PROG PROC
               ; các lệnh của chương trình được viết ở đây
        PROG ENDP

        Load_PROG PROC
               ; các lệnh của phần thường trú của chương trình được viết ở
        đây
              Mov, DX, offset Load_Prog
              Int 21h
        Load_PROG ENDP
```

b. Chương trình con phục vụ ngắt

Về cơ bản chương trình con phục vụ ngắt (để cho ngắn gọn ta gọi là *chương trình ngắt*) giống như một chương trình dạng COM mà ta đã tìm hiểu từ các phần trước. Tuy nhiên, có một số điểm lưu ý khi ta muốn viết một chương trình con phục vụ ngắt đó là:

- Bảo vệ các thông tin trạng thái và khôi phục lại khi kết thúc chương trình ngắt
- Lệnh cuối cùng của chương trình ngắt là lệnh IRET

Do vậy, dưới đây là "khung" của một chương trình ngắt, nó được sử dụng khi viết một chương trình ngắt.

```
MODEL Tiny
.CODE
      Org 100h
      ; vùng dữ liệu
  SohieuNgat EQU 9h
  NgatCu DW 2 DUP(0); luu dia chi ngat cu
      Jmp Load_Prog
Start:
TenCTN PROC
  Push AX
  Push BX
      Push CX
      Push DX
      Push DI
      Push SI
```

```
        Push DS
        Push ES
        ; Thân chương trình thường trú
        Pop ES
        Pop DS
        Pop SI
        Pop DI
        Pop DX
        Pop CX
        Pop BX
        Pop AX
 IRET
TenCTN ENDP
;------------------------------
Load_Prog PROC
      Mov AH,35
  Mov AL, SohieuNgat
  Int 21h
  Mov NgatCu,BX ; lay dia chi  lech
  Mov NgatCu[2],ES ; lay dia chi doan
  Mov AH,25
      Mov DX,offset TenCTN ; vao PROG
      Int 21h
 Exit:
      Mov DX,offset Load_Prog ; giữ lại cho thường trú
  Int 27h
Load_Prog ENDP
End Start
```

3.5 TÓM TẮT

Chương này đã trình bày về các công cụ lập trình hỗ trợ lập trình bằng hợp ngữ. Ta đã tìm hiểu về bộ gỡ rối Debug-một công cụ quan trọng để giúp người lập trình hệ thống gỡ lỗi cho chương trình. Trong phần Debug ta đã đề cập hai vấn đề chủ yếu: cách sử dụng trình tiện ích Debug và tìm hiểu các lệnh do Debug cung cấp. Tiếp đến là tìm hiểu một chương trình mô phỏng cho bộ xử lý 8086 là Emu8086. Chương trình mô phỏng này không những cung cấp cho người học các cơ chế xử lý lệnh, các trạng thái trong quá trình thực hiện một chương trình…mà còn cung cấp cho người học lập trình hệ thống một môi trường phát triển các chương trình hợp ngữ đơn giản và 52 chương trình mẫu theo nhiều dạng khác nhau.

Phần kết nối chương trình hợp ngữ với các ngôn ngữ bậc cao ta đi sâu vào tìm hiểu co chế kết nối giữa C và hợp ngữ và các vấn đề liên quan. Trên thực tế việc kết hợp cả hai ngôn ngữ C và hợp ngữ để phát triển các chương trình hệ thống là rất thông dụng. Ngoài ra, ta cũng tham khảo cơ chế kết hợp tương tự giữa hợp ngx và Pascal.

Cuối cùng ta đã tìm hiểu về chương trình ngắt và chương trình thường trú. Các dịch vụ của BIOS và DOS. Trong đó ta đã tìm hiểu ký về hai dịch vụ điển hình là: phục vụ cho màn hình và bàn phím. Cuối cùng ta tìm hiểu về cơ chế hoạt động và cách thức viết một chương trình thường trú và chương trình con phục vụ ngắt.

3.6 BÀI TẬP

3.6.1 Câu hỏi trắc nghiệm

1. Để so sánh nội dung của hai vùng nhớ, ta dùng lệnh nào dưới đây của Debug

A. Lệnh A

B. Lệnh C

C. Lệnh D

D. Lệnh E

2. Để cho thực hiện một chương trình , ta dùng lệnh nào dưới đây của Debug

A. Lệnh G

B.Lệnh L

C. Lệnh T

D. Lệnh P

3. Phát biểu nào dưới đây đúng và đầy đủ nhất về chương trình mô phỏng Emu8086.

A. Là một hệ soạn thảo chương trình

B. Hỗ trợ cho người lập trình viết các chương trình con và Macro

C. Mô phỏng quá trình thực hiện chương trình

D. Cung cấp môi trường cho người lập trình viết các chương trình hợp ngữ và Mô phỏng quá trình thực hiện chương trình

4. Phát biểu nào sau đây là đúng nhất cho việc kết hợp giữa hợp ngữ và C bằng cách sử dụng inline assembly:

A. inline assembly chỉ chứa các lệnh MOV, INC, DEC, ADD,SUB

B. Các lệnh inline assembly được viết cùng với các lệnh C trong file chương trình C

C. Các lệnh inline assembly được viết cùng với các lệnh C trong file chương trình hợp ngữ.

D. Trong cùng một chương trình C, các inline assembly phải được viết tách biệt với các lệnh của C.

5. Phát biểu nào sau đây là đúng nhất cho việc kết hợp giữa hợp ngữ và C bằng cách sử dụng viết tách biệt module hợp ngữ và C:

A. Trong các module C chỉ chứa các lệnh C.

B. Trong module hợp ngữ không thể gọi được các hàm viết từ C

C. Trong module C có thể gọi các hàm từ module hợp ngữ và trong module hợp ngữ cũng có thể gọi các hàm từ module C.

D. Các module C và hợp ngữ được dịch và thực hiện độc lập với nhau.

6. Phát biểu nào sau đây là sai đối với việc tương thích kiểu giữa module hợp ngữ và module C:

A. Kiểu unsigned char của C không tương thích với kiểu byte của hợp ngữ.

B. Kiểu char của C không tương thích với kiểu word của hợp ngữ

C. Kiểu short của C tương thích với kiểu word của hợp ngữ

D. Kiểu far * của C tương thích với kiểu dword của hợp ngữ.

7. Phát biểu nào sau đây là sai đối với kiểu giá trị trả lại của hàm và nơi đặt giá trị trả lại trong module hợp ngữ:

A. Kiểu giá trị unsigned long được đặt vào DX:AX.

B. Kiểu giá trị enum được đặt vào AX

C. Kiểu giá trị float được đặt vào đỉnh ngăn xếp 8087 thanh ghi ST(0)

D. Kiểu giá trị unsigned long được đặt vào AX

8. Khi một ngắt được đáp ứng yêu cầu, các lệnh sẽ được thực hiện theo thức tự sau:

A. PUSHF, CLI, PUSH CS, PUSH IP.

B. PUSH IP, PUSH CS, PUSH IP, CLI,.

C. PUSH CS, PUSH IP, CLI, PUSHF .

D. CLI, PUSH CS, PUSH IP, PUSHF

9. Phát biểu nào sau đây là đúng nhất đối chương trình thường trú:

A. Có thể viết chương trình thường trú sử dụng khung của chương trình .EXE.

B. Hỗ trợ khả năng kích hoạt và nằm lại bộ nhớ sau khi chạy xong.

C. Được kích hoạt bởi một tổ hợp phím nóng (hot-key).

D. Hoạt động giống như các chương trình bình thường khác.

10. Phát biểu nào sau đây là đúng nhất đối chương trình thường trú:

A. Vùng nhớ cấp phát cho chương trình thường trú không được giải phóng để cấp phát cho chương trình khác khi nó thực hiện xong .

B. Vùng nhớ cấp phát cho chương trình thường trú được giải phóng để cấp phát cho chương trình khác khi nó thực hiện xong

C. Khi thực hiện lần đầu tiên nó không cần chương trình tải (program loader) tải vào vùng nhớ cấp phát cho nó.

D. Mỗi lần thực hiện chương trình thường trú sẽ được tải vào vùng nhớ được cấp phát cho nó.

3.6.2 Bài tập

1. Dùng trình tiện ích Debug để chạy và gỡ lỗi chương trình ví dụ 1 phần 2.4

2. Viết chương trình nhập vào một số nguyên N (0<N<256), hãy in ra N dưới các dạng: thập phân, nhị phân và hexa. Yêu cầu chương trình được tổ chức như sau:

a. Dùng inline-assembly và C

b. Viết tách biệt module C và hợp ngữ

c. Viết tách biệt module Pascal và hợp ngữ

3. Viết chương trình nhập vào từ phàn phím hai số nguyên dương (0<x,y<256), hãy tính tổng hiệu, tích thương của chúng rồi in ra màn hình. Yêu cầu chương trình được tổ chức như sau:

a. Dùng inline-assembly và C

b. Viết tách biệt module C và hợp ngữ

4. Lập chương trình thực hiện nhiệm vụ copy một file có kích thước tùy ý. Tên file nguồn và đích nhận vào từ bàn phím. Thông báo lỗi ra màn hình. Yêu cầu chương trình được tổ chức như sau:

a. Dùng inline-assembly và C

b. Viết tách biệt module C và hợp ngữ

3.7 TÀI LIỆU THAM KHẢO

1. Đặng Thành Phu. Turbo Assembler và Ứng dụng. Nhà XB Khoa học và Kỹ thuật. 1998.
2. Nguyễn Minh San. Cẩm nang Lập trình Hệ thống (tập 2). Bản dịch. Nhà XB Tài chính Thống Kê. 1996.
3. Nguyễn Đình Việt. Giáo Trình nhập môn Hợp ngữ và Lập trình Hệ thống. Hà nội. 1998.
4. Website: www.emu8086.com

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CHƯƠNG TRÌNH ĐÀO TẠO ĐẠI HỌC TỪ XA PTIT

CHƯƠNG 4: LẬP TRÌNH PHỐI GHÉP

Phần này ta sẽ tìm hiểu về truyền tin nối tiếp và lập trình phối ghép với các thiết bị UART 8250A, bàn phím, màn hình và một công cụ hỗ trợ người lập trình hệ thống trên windows-RadASM.

4.1 TRUYỀN THÔNG TIN NỐI TIẾP VỚI BỘ ĐIỀU HỢP UART

Việc truyền thông tin giữa các thành phần nằm gần nhau như các bộ phận của một hệ vi xử lý đĩa cứng, màn hình, CPU…có thể thực hiện thông qua bus song song mở rộng hoặc qua các mạch phối ghép song song. Trong đó một nhóm bít (8, 16 , 32 hoặc 64) được truyền từ bộ phận này sang bộ phận khác trên 1 tập các đường truyền tín hiệu cáp. Ngược lại, trong trường hợp phải truyền tín hiệu giữa các đối tượng ở xa nhau thì không thể dùng nhiều dây tín hiệu đồng thời vì lý do kinh tế. Phương thức truyền tin nối tiếp phù hợp với yêu cầu thực tế này. Trong phương thức truyền tin nối tiếp thì ở đầu phát, dữ liệu dưới dạng song song được chuyển thành dữ liệu dạng nối tiếp, tín hiệu nối tiếp sau đó được truyền đi liên tiếp theo từng bit trên một đường dây đến đầu thu. Tại đầu thu, tín hiệu nối tiếp sẽ được biến đổi ngược lại thành dạng song song để truyền cho các thành phần đứng gần nhau.

4.1.1 Cơ bản về truyền tin nối tiếp

Có hai kiểu truyền thông nối tiếp: đồng bộ và dị bộ. Trong phương thức đồng bộ thì dữ liệu được truyền đi theo từng bản tin (một đoạn văn bản) với một tốc độ xác định. Túy theo giao thức truyền thông mà mỗi bản tin truyền đi có các cấu trúc khác nhau. Chẳng hạn, nếu dùng giao thức truyền thông tin hệ 2 đồng bộ (BISYNC: Binary Synchronous Communication Protocol) thì cấu trúc một bản tin như sau:

SYN	SYN	SOH	HEADER	STX	TEXT	ETX	BCC
Kí tự đồng bộ	Kí tự đồng bộ	Bắt đầu phần header	phần header	Bắt đầu phần nội dung	Phần nội dung bản tin	Kết thúc phần nội dung	Kí tự kiểm tra khối

Ngược lại, trong phương thức truyền không đồng bộ, dữ liệu được truyền đi theo từng kí tự. Kí tự cần truyền đi được gắn thêm bit đánh dấu ở đầu để đánh dấu bắt đầu kí tự. 1 hoặc 2 bít cuối cùng để đánh dấu kết thúc kí tự. Vì mỗi kí tự được nhận dạng riêng biệt nên nó có thể truyền đi bất cứ khi nào. Dưới đây là cấu trúc của một khung truyền (frame) theo phương pháp không đồng bộ.

Start	D0	D1	D2	D3	D4	D5	D6	Parity	Stop	Stop

Mã kí tự cần truyền

Tùy theo loại mã ta sử dụng để mã hóa kí tự (baudot, ASCII, EBCDIC) mà độ dài cho mã kí tự có thể là 5,6,7,8 bit. Tùy theo hệ thông struyền tin, bên cạch các bít mã dữ liệu còn có thể có bit parity dùng để kiểm tra lỗi khi truyền.

Việc truyền tin dị bộ được thực hiện nhờ một UART (Universal Asynchronous Receiver Transmitter) ở đầu phát và một UART ở đầu thu. Khi có kí tự phát, mạch 8251A sẽ tạo ra khung cho kí tự bằng cách thêm vào các bit start, parity và stop, rồi gửi liên tiếp từng bít ra đường truyền. Bên phía thu, một mạch 8251A khác sẽ nhận kí tự, tháo bỏ khung truyền, kiểm tra parity, rồi chuyển sang dạng song song để CPU đọc.

4.1.2 Các thanh ghi của UART 8250A/16450

Mạch 8250A là một mạch thu phát dị bộ vạn năng (UART) được sử dụng rất phổ biến để phối ghép với cổng thông tin nối tiếp như cổng COM theo chuẩn RS 232C. Dưới đây ta tìm hiểu về các thanh ghi bên trong của UART 8250A.

Dưới đây là bảng liệt kê tổ hợp chân tín hiệu có thể chọn racác thanh ghi bên trong của UART 8250A.

DLA	A2	A1	A0	Chọn ra thanh ghi
0	0	0	0	Thanh ghi đệm thu (RBR), thanh ghi giữ phát (THR)
0	0	0	1	Thanh ghi cho phép tạo yêu cầu ngắt (IER)
1	0	0	0	Thanh ghi cho số chia phần thấp (LSB)
1	0	0	1	Thanh ghi cho số chia phần cao (MSB)
X	0	1	0	Thanh ghi nhận dạng nguồn gốc yêu cầu ngắt (HR)
X	0	1	1	Thanh ghi điều khiển đường truyền (LCR)
X	1	0	0	Thanh ghi điều khiển modem (MCR)
X	1	0	1	Thanh ghi trạng thái đường dây (LSR)
X	1	1	0	Thanh ghi trạng thái modem (MSR)
X	1	1	1	Thanh ghi nháp

Thanh ghi điều khiển đường truyền (Line Control Register)

Thanh ghi này còn có tên khác là thanh ghi định khuôn dạng dữ liệu vì nó quyết định khuôn dạng dữ liệu trên đường truyền. Cấu trúc của thanh ghi LCR được biểu diên như sau:

D7	D6	D5	D4	D3	D2	D1	D0
DLAB	SBCB	SP	EPS	PEN	STB	WLS1	WLS1

- ***Bít D7 (DLAB)***: Bit truy nhập số chia
  1: truy nhập số chia
  0: truy nhập IER,THR và RBR

- ***Bít D6 (SBCB)***: Bit điều khiển gián đoạn
  1: buộc S_{out} =0
  0: không hoạt động
- ***Bít D5 (SP)***: Đảo parity
  1: parity chẵn
  0: không hoạt động
- ***Bít D4 (ESP)***: chọn tạo/kiểm tra parity chẵn
  1: parity chẵn
  0: parity lẻ
- ***Bít D3 (PEN)***: cho phép tạo/kiểm tra parity
  1: cho phép
  0: cấm
- ***Bít D2 (STB)***: số bit stop
  1: 1,5 hoặc 2 bít
  0: 1bít
- ***Bít D1, D0 (WLS1, WLS0)***: Chọn độ dài từ
  00: 5 bít
  01: 6 bít
  10: 7 bít
  11: 8 bít

Thanh ghi đệm giữ phát (Transmitter Holding Register- THR)

Ký tự cần phát đi phải được ghi từ CPU vào thanh ghi này trong khi bit DLAB=0. Sau đó khi truyền 8250A lấy kí tự từ đây, đóng khung cho nó như đã định và đưa từng bít ra chân S_{out}

Thanh ghi đệm thu (Receiver Buffer Register- RBR)

Khi 8250A nhận được một ký tự qua chân S_{in} , nó tháo bỏ khung truyền cho kí tự và giữa kí tự tại thanh ghi đệm thu để chờ CPU đọc vào. CPU chỉ đọc được kí tự trong thanh ghi này khi bít DLAB=0.

Thanh ghi cho phép tạo yêu cầu ngắt (Interrupt Enable Register- IER)

Thanh ghi này dùng để cho phép/ cấm các nguyên nhân gây ra ngắt rkhác nhau. Trong khi mạch 8250A hoạt động có thể tác động đượctới CPU thông qua chân INTRPT của UART. Mỗi bít

trong các bít D3,D2,D1,D0 ở mức cao cho phép các hiện tượng tương ứng với bít đó được đưa ra yêu cầu ngắt đối với CPU.

Dưới đây là cấu trúc của thanh ghi cho phép tạo yêu cầu ngắt.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	MODEM	RLINE	TxEMPTY	RxRDY

- ***Bít D3 (MODEM)*** = 1: cho phép các thay đổi trạng thái của modem gây ra ngắt.
- ***Bít D2 (RLINE)*** = 1: cho phép các tín hiệu trạng thái đường truyền thu gây ra ngắt.
- ***Bít D1 (TxEMPTY)*** = 1: cho phép gây ra ngắt khi đệm giữ phát rỗng.
- ***Bít D0 (RxRDY)*** = 1: ho phép gây ra ngắt khi đệm thu đầy..

Thanh ghi nhận dạng nguồn yêu cầu ngắt (Interrupt Identification Register- IIR)

Thanh ghi nhận dạng ngắt (chỉ để đọc ra) chứa mã mức ưu tiên cao nhất của yêu cầu ngắt tại chân INTRPT của 8250A đang chờ phục vụ. Do vậy khi cần xử lý các yêu cầu ngắt theo kiểu thăm dò, CPU cần đọc bít ID_0 của thanh ghi này để biết là có yêu cầu ngắt và kiểm tra các bít ID_2- ID_1 để xác định được nguồn gốc của yêu cầu ngắt.

Mỗi lần UART được reset thì chỉ có yêu cầu ngắt ở mức ưu tiên số 1 sẽ được phục vụ. Điều này có thể thay đổi bằng cách dùng mặt nạ che đi các yêu cầu ngắt nào đó bằng cách ghi vào thanh ghi IER các bít thích hợp.

Dưới đây là cấu trúc của thanh ghi nhận dạng nguồn yêu cầu ngắt.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	ID_2	ID_1	ID_0

- ***Bít D2, D1 (ID_2, ID_1)***: mã hóa các yêu cầu ngắt có mức ưu tiên cao nhất đang chờ được phục vụ.
- ***Bít D0 (ID_0)*** = 0: có yêu cầu ngắt
  = 1: không có.

Thanh ghi điều khiển modem (Modem Control Register- MCR)

Đây là thanh ghi điều khiển các tín hiệu ra của modem thông qua việc điều khiển các tín hiệu tại chân DTR và RTS của mạnh UART.

Cấu trúc của thanh ghi điều khiển modem như sau:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOOP	OUT2	OUT1	RTS	DTR

- ***Bít D4 (LOOP)*** = 1: Nối vòng cục bộ
  = 0: bình thường.
- ***Bít D3, D2 (OUT2, OUT1)*** = 1: đưa OUTi=1
  = 0: đưa OUTi=0.
- ***Bít D1 (RTS)*** = 1: đưa RTS=1
  = 0: đưa RTS=0.
- ***Bít D0 (DTR)*** = 1: đưa RTS=1
  = 0: đưa RTS=0.

Khi thiết lập D0=DTR=1 ta có thể điều khiển tín hiệu tại chân DTR của mạch 8250A đạt mức tích cực thấp để báo UART sẵn sàng làm việc.

Tương tự, D1=RTS=1 ta có thể điều khiển tín hiệu tại chân RTS của mạch 8250A đạt mức tích cực thấp để báo UART sẵn sàng truyền phát ký tự.

Ngoài ra, ta có thể điều khiển được các đầu ra phụ OUT1 và OUT2. Bằng các bít D2=OUT1=1 và D3=OUT2=1 để điều khiển tín hiệu tại các chân này sao cho OUT1=1 và OUT2=1.

Bít D4=1 cho phép điều khiển mạch 8250A làm việc ở chế độ nối vòng cục bộ để kiểm tra chức năng của UART.

Khi D4=1 thì cấu hình sau được thiết lập:

S_{out} =1

S_{in} =1: không nối với bên ngoài

Các thanh ghi dịch của phần phát với phần thu được nối vòng với nhau

Các chân điều khiển vào của modem không được nối ra ngoài mà được nối ở bên trong mạch với các chân điều khiển ra của modem..

Thanh ghi trạng thái modem (Modem Status Register- MSR)

Thanh ghi này còn được gọi là thanh ghi trạng thái vào từ cổng nối tiếp RS232C vì nó cho biết trạng thái hiện thời của các tín hiệu điều khiển modem từ đường truyền.

Dưới đây là cấu trúc của thanh ghi trạng thái modem:

D7	D6	D5	D4	D3	D2	D1	D0
RLSD	R1	DSR	CTS	RLSD*	RI*	DSR*	CTS*

Bít D7,D6,D5, D4 (RLSD,RI,DSR,CTS): Có giá trị các bít OUT2, OUT1, DTR, RTS trong MCR khi bít LOOP=1.

- ***Bít D3 (RLSD)*** = 1: nếu có sự thay đổi của các tín hiệu tương ứng so với lần đọc trước.
- ***Bít D2 (RI)*** = 1: Nếu RI có biến đổi từ mức thấp lên mức cao

Dấu * đứng trước các chan tín hiệu để chỉ ra rằng trong khi 8250A hoạt động, nếu có sự thay đổi của các tín hiệu đó thì các bít tương ứng sẽ được lập. Với tín hiệu RI thì đó là sự biến đổi từ mức thấp lên mức cao.

Thanh ghi trạng đường truyền (Line Status Register- LSR)

Thanh ghi trạng thái đường truyền cho ta biết được trạng thái của việc truyền tín hiệu trên đường truyền.

Dưới đây là cấu trúc của thanh ghi trạng thái đường truyền:

D7	D6	D5	D4	D3	D2	D1	D0
0	TSRE	THRE	BI	FE	PE	OE	RxDR

- ***Bít D6 (TSRE)***: thanh ghi dịch phát rỗng.

=1: khi 1 ký tự được phát đi, bít này bị xóa khi có một kí tự chuyển từ THR sang TSR

- ***Bít D5 (THRE)***: thanh ghi giữ phát rỗng.

=1: khi ký tự đã được chuyển từ THR sang TSR, bít này bị xóa khi CPU đưa kí tự tới thanh ghi THR.

- ***Bít D4 (BI)***: ngắt gây ra sự gián đoạn khi truyền.

=1: khi tín hiệu đầu vào phần thu ở mức thấp lâu hơn thời gian dành cho một ký tự, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D3 (FE)***: lỗi khung truyền.

=1: báo có lỗi về khung truyền, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D2 (PE)***: lỗi parity.

=1: báo có lỗi parity, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D1 (OR)***: lỗi do nhận tín dữ liệu bị ghi đè.

=1: có hiện tượng ghi đè do CPU chưa kip đọc thanh ghi đệm thu, bit này bị xóa khi CPU đọc thanh ghi LSR.

- ***Bít D0 (RxDR)***: sẵn sàng nhận dữ liệu.

=1: đã nhận được một kí tự và đăth nó trong thanh ghi đệm thu (RBR), bit này bị xóa khi CPU đọc thanh ghi RBR.

4.1.3 Lập trình cho UART 8250A/16450

Như ta đã biết truyền thông nối tiếp giữa hai máy tính được tực hiện qua cổng COM và theo chuẩn RS 232C. Phần này sẽ giới thiệu về lập trình truyền thông qua các cổng COM và ngắt BIOS phục vụ vào/ra của các cổng COM.

a. Địa chỉ các cổng COM và ngắt 14h-dịch vụ BIOS cho cổng COM

Bảng địa chỉ cổng COM:

Cổng	Địa chỉ I/O	Yêu cầu ngắt
COM1	3F8-3FF	IRQ 4
COM2	2F8-2FF	IRQ 3
COM3	338-33F	IRQ 5
COM4	238-23F	IRQ 5

Các cổng COM3 và COM4 có thể không cần sử dụng mức ngắt như trên. Chúng có thể sử dụng các mức ngắt IRQ2, IRQ5, IRQ7 hoặc IRQ9.

Dưới đây ta sẽ tìm hiểu về các chức năng của ngắt 14h –dịch vụ BIOS dành cho các cổng COM.

Hàm 0h:

Ý nghĩa: Khởi tạo cổng COM.

Đầu vào: AH=0

DX=số hiệu cổng COM (0-3)

AL= tham số khởi tạo trong đó định dạng của AL như sau:

D7	D6	D5	D4	D3	D2	D1	D0
Tốc độ truyền (bits/giây) 000: 110 bits/ giây 001: 150 bits/ giây 010: 300 bits/ giây 011: 600 bits/ giây 100: 1200 bits/ giây 101: 2400 bits/ giây 110: 4800 bits/ giây 111: 9600 bits/ giây			*Bit chẵn /lẻ* x0: không chẵn lẻ 01: lẻ 11: chẵn		*Độ dài bit stop* 0: 1 bít 1: 2 bít	*Độ dài mã kí tự:* 00: 5 kí tự 01: 6 kí tự 10: 7 kí tự 11: 8 kí tự	

Int 14h

Đầu ra: AH = trạng thái của cổng vừa khởi tạo.

Hàm 1h:

Ý nghĩa: Ghi một kí tự ra cổng COM

Đầu vào: AH=1

DX=số hiệu cổng COM (0-3)

AL= Mã ASCII của kí tự

Int 14h

Đầu ra: AH = trạng thái lỗi của cổng. Bít 7=1: có lỗi, bít 7=0 không có lỗi.

Hàm 2h:

Ý nghĩa: Nhận một kí tự ra cổng COM

Đầu vào: AH=2

DX=số hiệu cổng COM (0-3)

Đầu ra: AL= Mã ASCII của kí tự nhận được. AH = trạng thái lỗi của cổng. Bít 7=1: có lỗi, bít 7=0 không có lỗi.

Hàm 3h:

Ý nghĩa: Đọc trạng thái cổng COM

Đầu vào: AH=3

DX=số hiệu cổng COM (0-3)

Đầu ra: AH = trạng thái cổng COM và AL-= trạng thái modem. Chi tiết như sau.

Dạng thức của AH:

Bit 7	Lỗi qua thời gian (timeout)
Bit 6	Thanh ghi phát rỗng
Bit 5	Thanh ghi nhận rỗng
Bit 4	Cho phép ngắt
Bit 3	Lỗi khung truyền
Bit 2	Lỗi parity
Bit 1	Lỗi đường truyền
Bit 0	Dữ liệu đã có trong bộ đệm

Dạng thức của AL:

Bit 7	Phát hiện vật mang dữ liệu (Data Carrier Detect)
Bit 6	Chỉ thị chuông (Ringing Indicator)
Bit 5	Sẵn sàng thiết lập dữ liệu (Data Set Ready)

Bit 4	Xóa gửi (Clear To Send)
Bit 3	Phát hiện sóng mang dữ liệu Delta(Delta Data Carrier Detect)
Bit 2	Chỉ thị chuông ở đuôi (Tailing Edge Ring Indicator)
Bit 1	Sẵn sàng thiết lập dữ liệu Delta (Delta Data Set Ready)
Bit 0	Xóa gửi Delta (Delta Data Set Ready)

b. Lập trình cho UART 8250A

Ví dụ 1: Khởi tạo chế độ làm việc cho cổng COM3 với các thông số: 6 bít mã kí tự truyền , tốc độ truyền 2400 bits/ giây, parity chẵn, một bít stop.

Dạng thức của AL

D7	D6	D5	D4	D3	D2	D1	D0
101: 2400 bits/ giây			*Bit chẵn /lẻ* 11: chẵn		*Độ dài bit stop* 0: 1 bít	*Độ dài mã kí tự:* 01: 6 kí tự	

AL=1011 1001=B9h

Đoạn mã chương trình sẽ được viết như sau:

```
Mov AH,0 ; khởi tạo
Mov DX,2 ; cổng COM3 có số hiệu 2
Mov AL,B9h
Int 14h
```

Ví dụ 2: Khởi tạo chế độ làm việc cho cổng COM1 với các thông số: 7 bít mã kí tự truyền , tốc độ truyền 4800 bits/ giây, parity chẵn, một bít stop không điều khiển gán đoạn ở S_{out}.

Địa chỉ cổng của thanh ghi điều khiển đường truyền là 3FB (tính từ địa chỉ cơ sở 3F8)

Giá trị của của thanh ghi điều khiển đường truyền là:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	1	0	1	0

Giá trị này bằng 1AH

Địa chỉ cổng của thanh ghi số chia LSB là 3F8 với DLAB=1

Địa chỉ cổng của thanh ghi số chia MSB là 3F9 với DLAB=1

Giả thiết dùng xung đồng hồ tần số 1,8432 Mhz ở đầu vào của UART và ta muốn có tốc độ truyền 4.800 bits/giây, ta cần tính số chia để ghi giá trị số chia vào thanh ghi số chia

Số chia = 1.843200/(4.800x16)=24

Đoạn mã chương trình sẽ được viết như sau:

```
Mov AL,80 ; đưa vào LCR để tạo ra DLAB=1
Mov DX,3FBH ; địa chỉ LCR
Out DX,AL
Mov AL,24 ; đưa vào LSB của số chia
Out DX,AL
Mov AL,0 ; đưa vào MSB của số chia
Mov DX,3F9H ; địa chỉ MSB của số chia
Out DX,AL
Mov AL,1AH ; qui định khuôn dạng dữ liệu
Mov DX,3FBH ; địa chỉ LCR
Out DX,AL
…
```

4.2 MỘT SỐ LẬP TRÌNH PHỐI GHÉP CƠ BẢN

Phần này giới thiệu về một số lập trình phối ghép cơ bản với bàn phím, màn hình. Chúng ta sẽ sử dụng những chức năng và dịch vụ do BIOS và DOS cung cấp để lập trình với các thiết bị này. Chi tiết về các chức năng đã được trình bày ở phần 3.4.1.

4.2.1 Lập trình phối ghép với bàn phím

a. Cơ bản về bàn phím

Trong số các thiết bị thu nhận tín hiệu: bàn phím, chuột, cần điều khiển, bút quang, màn hình cảm biến, bảng vẽ vector, thiết bị quét, máy ảnh số, thiết bị nhận dạng ảnh, tiếng nói… thì bàn phím là một trong những thiết bị vào thông dụng nhất. Bàn phím bao gồm một tập hợp các phím mà mỗi phím hoạt động như một công tắc cảm biến lực chuyển lực nhấn thành một đại lượng điện. Bộ vi điều khiển 8042 của máy tính và bộ vi điều khiển của bàn phím liên lạc với nhau tuần tự và đồng bộ qua hai đầu dây dẫn: dây dữ liệu và dây đồng hồ. hai vi điều khiển này làm việc việc với nhau theo nguyên tắc chủ/tớ (master/slaver) trong đó 8042 là chu còn 8048 là tớ.

Khi có một thao tác phím, bàn phím phát ra một tín hiệu điện (IRQ1) gửi đến CPU. CPU sẽ tạm dừng công việc và gọi ngắt Int 9 đọc cổng bàn phím (địa chỉ 60H) để nhận mã Scan của tác động phím vừa xảy ra. Ngoài ra, nó sẽ đọc cờ bàn phím đó là một word ở địa chỉ 40:0017 để kết hợp với mã scan sinh ra mã ASCII tương ứng với tác động phím ở trên. Cặp hai byte gồm mã Scan và mã ASCII sẽ được Int 9h đưa vào vùng đệm bàn phím theo thứ tự mã ASCII đứng trước mã Scan. Chúng nằm ở đó cho tới khi được ngắt 16h phục vụ.

Trong trường hợp ngắt 9h phát hiện thấy từ cổng 60H các mã Scan ứng với tổ hợp phím đặc biệt, ví dụ Cltr_Alt_Del, PrintScreen, Ctrl_Numlock, Ctrl_Break, SysReq thì nó sẽ hiểu đây là các lệnh khiến ROM-BIOS phải thực hiện ngay chứ không lưu lại các tổ hợp phím này trong bộ đệm bàn phím .

- Khi gặp Cltr_Alt_Del, ROM BIOS gọi ngắt 19h (Boot strap loader)
- Khi gặp PrtScr, ROM BIOS gọi ngắt 5h (hard copy)

- Khi gặp Cltr_Numlock, ROM BIOS thi hành một vòng lặp vô hạn cho đến khi một phím khác được nhấn.
- Khi gặp Cltr_Break, ROM BIOS gọi ngắt 1Bh – kết thúc chương trình đang thực hiện, trả điều khiển lại cho DOS.
- Khi gặp SysReq, ROM BIOS gọi ngắt 15h với giá trị 8500H trong AX, khi nhả phím này int 15h cũng được gọi nhưng với AX=8501. Dịch vụ 85H của ngắt 15H chỉ chứa một lệnh IRET, không gây tác động gì.

b. Bộ đệm bàn phím và các thao tác

Mỗi phím trên bàn phím được bộ xử lý của bàn phím gán cho một mã Scan đặc trưng cho vị trí của phím trên bàn phím và cho trạng thái của phím (nhấn, không nhấn, nhấn chưa nhả..). Bộ đệm bàn phím (Keyboard buffer) là một miền nhớ 16 word trong RAM. Thuộc vùng dữ liệu của BIOS. Dưới đây là một phần vùng dữ liệu của BIOS (DTA) liên quan đến bộ đệm và trạng thái bàn phím:

Địa chỉ	Nội dung
40:0000-40:0006	Địa chỉ các bộ phối ghép RS232 (1-4)
40:0008-40:000F	Địa chỉ các bộ phối ghép máy in (1-4)
40:0010	Cờ thiết bị (do ngắt 11h trả lại)
40:0012	Dấu hiệu kiểm tra của nhà sản xuất
40:0013	Dung lượng nhớ tính theo đơn vị KB
40:0015	Bộ nhớ của kệnh vào/ra
40:0017	Cờ bàn phím
40:0019	Các số vào bằng phím alt
40:001A	Vị trí Head của vùng đệm
40:001C	Vị trí Tail của vùng đệm
40:001E-40:003D	Bộ đệm bàn phím

Vùng đệm bàn phím được tổ chức theo một hàng đợi quay vòng (circular queue) trong đó thao tác đọc và ghi vào bộ đệm là độc lập với nhau, điều này làm cho việc ghi tác động phím vào và đọc ra theo kiểu mã phím (ASCII+Scan) nào được đưa vào bộ đệm trước thì sẽ được lấy ra trước.

Con trỏ Head lưu trữ địa chỉ dành cho thao tác đọc, đó là vị trí sẽ đọc ký tự tiếp theo ra khỏi bộ đệm bàn phím. Sau mỗi thao tác đọc Head được tăng lên 1 word. Word tại địa chỉ 40:001A trong vùng DTA chứa địa chỉ của Head.

Con trỏ Tail lưu trữ địa chỉ sẽ ghi tác động phím tiếp theo vào bộ đệm bàn phím. Sau mỗi thao tác ghi Tail được tăng lên 1. Word tại địa chỉ 40:001C chứa địa chỉ của Tail.

Khi cả Head và Tail cùng trỏ tới word cuối cùng của bộ đệm và nếu có một tác động phím nữa xảy ra thì cả Head va Tail cùng trỏ tới đầu vùng đệm (40:001E)

Cứ mỗi phím được đọc ra thì Head lại tiến gần đến Tail, khi tất cả vùng đệm đã được đọc hết thì Head sẽ đuổi kip Tail và cả hai cùng trỏ tới cùng một địa chỉ, lúc đó bộ đệm là rỗng.

Khi có lời gọi ngắt int 9 thì chương trình của ta chiếm quyền điều khiển, nó sẽ gọi đến chương trình xử lý ngắt bàn phím cũ để nhận tác động phím và đặt cặp byte mã ASCII và Scan vào bộ đệm bàn phím. Đồng thời gán cho DS địa chỉ đoạn của vùng dữ liệu BIOS và kiểm tra xem Tail có trong bộ đệm bàn phím không, word nằm trước Tail sẽ tương ứng với phím vừa mới nhận vào. Đọc byte mã Scan vào thanh ghi DH và byte mã ASCII vào DL. Sau đó, kiểm tra word trong DX có phải là hot-key hay không, nếu không phải sẽ nhảy đến kết thúc.

c. Phương pháp lập trình bàn phím và một số chương trình mẫu

Có hai cách lập trình bàn phím:

- Lập trình hệ thống, truy nhập lập trình qua các bộ vi điều khiển 8042, 8044 qua các cổng 60H, 61H và 64H.
- Lập trình ứng dụng, dùng các hàm BIOS (int 9h, Int 16H) hoặc các hàm 0Ah, 0Bh, 0Ch của int 21h.

Cổng 60H là cổng A của 8255A và là nơi để CPU và bàn phím trao đổi thông tin bao gồm các lệnh (thường là của chương trình con ngắt) điều khiển bàn phím hoặc ký tự từ bàn phím vào. Đối với người lập trình thì cổng 60H được coi như thanh ghi mã lệnh của bàn phím.

Mã lệnh EDH là mã lệnh tắt, bật đèn

Ví dụ 1: Viết chương trình tự động bật phím CapLock (sau khi chạy chương trình này phím Caplock sẽ được bật lên).

Ta hãy xem cấu trúc của cờ bàn phím (chi tiết xem phần 3.4.1):

7	6	5	4	3	2	1	0
1: chế độ Insert	1: chế độ Cap Lock	1: Num Lock bị ấn	1: Scroll Lock bị ấn	1: Alt bị ấn	1: Ctrl bị ấn	1: Shift trái bị ấn	1: Shift phải bị ấn

Cờ bàn phím là byte ở tại địa chỉ 40:0017 trong vùng đệm bàn phím. Để đèn của phím CapLock là ON thì giá trị của cờ bàn phím được đặt bằng: 01000000 =40h. ta chỉ cần gán giá trị 40h vào byte có địa chỉ 40:0017 là xong.

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
Start:
        Mov AX,40h ; AX chứa địa chỉ đoạn dữ liệu DTA
        Mov DS, AX ; cho DS trỏ tới vùng DTA
        Mov BX,0017h ; DS:BX chứa địa chỉ của byte chứa cờ
        ; trạng thái bàn phím
        Mov byte PTR[BX] ,40h ; đặt cờ bàn phím bằng 40h
```

```
        Int 20h ; trở về DOS
End Start
```

Ví dụ 2: Viết chương trình tự động bật phím CapLock , NumLock, ScollLock (sau khi chạy chương trình này phím Caplock, NumLock, ScollLock sẽ được bật lên). Yêu cầu chương trình thực hiện qua các cổng 60H, 61H và 64H.

Chương trình sẽ kiểm tra trạng thái nhấn phím hay chưa.

Cấu trúc của byte điều khiển các đèn LED trên bàn phím như sau:

7	6	5	4	3	2	1	0
					Cap Lock	Num Lock	Scroll Lock

Để các đèn Caplock, NumLock, ScollLock sẽ được bật lên thì ta phải gửi giá trị 00000111 =07h ra cổng 60H.

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
Start:
        Mov AL,EDHh ; lệnh tắt/bật đèn
        Out 60H, AL ; đưa ra cổng bàn phím
Kiemtra:
        In AL,64h  ;Kiểm tra trạng thái bàn phím
        Test AL,02h ; Nếu bộ đệm bàn phím bằng đầy
        Jnz Kiemtra ; kiểm tra lại
        Mov Al,07 ; Nếu bộ đệm trống sẽ bật đèn
        Out 60H, AL ; đưa ra mã bật đèn ra cổng bàn phím
        Int 20h ; trở về DOS
End Start
```

Ví dụ 3: Viết chương trình cấm bàn phím hoạt động. Biết rằng lệnh khóa ADH là lệnh cấm bàn phím.

Kiểm tra trạng thái bàn phím trước khi cấm.

```
.MODEL Tiny
.CODE
        Org 100h
        Jmp Start
Start:
Kiemtra:
        In AL,64H ; đọc trạng thái
        Test AL,02h ; Nếu bộ đệm bàn phím bằng đầy
        Jnz Kiemtra ; kiểm tra lại
```

```
        Mov Al,ADh ; Nếu bộ đệm trống sẽ bật đèn
        Out 64H, AL ; đưa ra cấm bàn phím ra cổng 64h
        Int 20h ; trở về DOS
End Start
```

Ví dụ 4: Viết chương trình xử lý bàn phím đơn giản. Chương trình kiểm tra các phím chữ cái và phân biệt chữ hoa, chữ thường.

```
.MODEL small
.STACK 100h
.DATA
        Table db  16 dup(0)
              db  'qwertyuiop',0,0,0,0 ; hang tren
          db  'asdfghjkl',0,0,0,0 ; hang giua
          db  'zxcvbnm'; hang duoi
          db 16 dup(0) ; vung danh cho chu hoa
          db 'QERTYUIOP',0,0,0,0 ; chu hoa cho hang tren
          db 'ASDFGHJKL',0,0,0,0,0 ; chu hoa cho hang giua
          db 'ZXCVBNM'; chu hoa cho hang duoi
.CODE
Start:
        Mov AX,@Data
        Mov DS,AX
        Cli; xoa ngat
        Push DS ;
        Mov AX, seg TryKB ; DS:AX tro den Checkbanphim
        Mov DS,AX
        Mov DX, offset TryKB ; Checkbanphim
        Mov Al,9
        Mov Ah,25H  ; dat lai dia chi vector ngat
        Int 21h
        Pop DS
        Sti ; cho phep ngat
        ;---------------------------------
        ;---------  Chuong trinh xu ly ngat ban phim--------
        TryKB proc far
        Push AX
        Push BX
        Push CX
        Push DI
        Push ES
        ; nhan ma Scan va tra loi ban phim
        In Al,60h ; nhan ma Scan
        Mov AH,AL ; chuyen vao AH
        Push AX ; dua vao stack
```

```
      In AL,61h ; doc cong PB cua 8255A
      Or AL,10000000b ; dua bit 7 len bit 1
      Out 61h,AL ; dua ra cong PB
      ; Cho ES tro den doan du lieu
      Mov AX,40H ; dua AX den cuoi bo nho
      Mov ES,AX
      Pop AX  ; AL= ma scan
; kiem tra phim shift
 Cmp Al,42 ; co nhan phim shift ko?
     Jnz checkkey ; neu khong thi kiem tra tiep
      Mov BL,1
      Or ES:[17h],BL đat bit 1 của byte trang thai =1
      Jmp Thoat ; thoat khoi
PhimKetiep:
 Test AL,10000000b; ma nha phim
 Jnz Thoat
 Mov BL,ES:[17h]  ; neu ko doc dc trang thai phim shift
 Test BL,00000011b; nhan phim shift?
 Jz DoiMa ; neu ko doi lai ma
    Add AL,100 ;doi ra ki tu hoa(dia chi 100 byte ke tiep)
DoiMa:
 Mov BX, offset table ;
 Xlat table ; doi ma scan sang ma ASCII
 Cmp Al,0 ; tra ve 0?
 Jz Thoat
; Kiem tra do dem ban phim da day cua?
 Mov BX, 1AH  ; nap con tro bo dem ban phim
 Mov CX,ES:[BX]
 Mov DI,ES:[BX]+2
 Cmp CX,60
 Jz Tieptuc
 Add, CX,2
      Cmp CX,DI
      Jz Thoat
; Bo dem chua day, nap the ki tu vao
Tieptuc:
      Mov ES:[DI],AL
 Cmp DI,60H
      Jnz Naptiep
      Mov DI,28 ; dua dia chi hien tai ve 28+2=30
Naptiep:
      Add DI,2
      Mov ES:[BX],DI
      ; ket thuc
```

```
Thoat:
 Pop ES
 Pop DI
 Pop CX
 Pop BX
 Pop AX
    Mov AL,20h ; tra lai ngat
 Out 20h,AL
 IRET
     TryKB endp
     ;-------------------------------------------
End Start
```

4.2.2 Lập trình phối ghép với màn hình

a. Cơ bản về màn hình

Bộ nhớ Video

Màn hình của máy tính hiện đại ngày nay đều sử dụng bộ nhớ video (bộ nhớ màn hình) để chứa nội dung hình ảnh được hiển thị và các thông tin liên quan. Bộ nhớ video còn được gọi là bộ đệm khung (frame buffer). Từ thế hệ Pentium, bộ vi xử lý còn có cổng ga tốc đồ họa AGP (Accelerated Graphics Port) Cổng này cho phépbộ vi xử lý dồ họa truy cập trực tiếp vào bộ nhớ hệ thống cho các phép tính đồ họa nhưng vẫn có bộ nhớ video riêng để lưu trữ nội dung các điểm ảnh màn hình. Phương pháp này cho phép sử dụng bộ nhớ hệ thống mềm dẻo hơn mà không làm ảnh hưởng tới tốc độ máy tính. Cổng AGP ngày nay trở thành chuẩn trong các máy tính hiện đại.

Dưới đây là bảng dung lượng bộ nhớ video và khả năng hiển thị màn hình.

Dung lượng bộ nhớ	Kích thước màn hình	Chiều sâu mầu	Số màu
1MB	1024x768 800x600	8-bit 16-bit	256 65,536
2MB	1024x768 800x600 1280x1024	8-bit 16-bit 24-bit	256 65,536 16.7 million
4MB	1024x768	24-bit	16.7 million
6MB	1280x1024	24-bit	16.7 million
8MB	1600x1200	32-bit	16.7 million

Bộ chuyển đổi số/tương tự

Tín hiệu ra màn hình không phải dạng số mà là dạng tín hiệu tương tự. Tín hiệu ra phải đi qua một bộ biến đổi bộ nhớ tương tự/ số (RAMDAC- RAM Digital Analog Converter) đọc nội dung bộ nhơ video rồi chuyển sang tín hiệu tương tự. Qua trình này lặp lại theo tần số làm tươi màn hình.

Bộ xử lý Video

Bộ xử lý video biên dịch và thực hiện lệnh đồ họa thành từng điểm ảnh cụ thể để đưa ra RAMDAC. Bộ vi xử lý video có cấu trúc hoàn chỉnh và phức tạp như một bộ vi xử lý thực sự. Nhờ có bộ xử lý video nên đã giảm tính toán cho từng điểm ảnh để hiển thị đối với bộ xử lý. Thay vào đó CPU chỉ cần truyền tham số căn bản của đối tượng cần hiển thị sang bộ xử lý video. Bộ xử lý video sẽ tính từng điểm ảnh cần hiển thị từ các tham số nhận được, giải phòng CPU khỏi nhiệm vụ này.

Đồ họa ba chiều

Để phần mềm bắt kịp được với phần cứng trong quá trình phát triển của hiển thị đồ họa ba chiều, các hệ điều hành cung cấp một thư viện giao diện lập trình ứng dụng (Application Programming Interface). Giống như BIOS, API cung cấp các lệnh đồ họa chuẩn ra các chương trình xử lý video thay vì lập trình trực tiếp nó. Phần mềm điều khiển sẽ phải biên dịch tiếp nhưng lệnh này sang lệnh máy.

b. Ví dụ về Lập trình phối ghép màn hình

Dưới đây là một số ví dụ về lập trình màn hình có sử dụng các chức năng và dịch vụ do BIOS cung cấp đã trình bày ở phần 3.4.1.

Ví dụ 1: In ra màn hình 256 kí tự của bảng mã ASCII, mỗi kí tự có một thuộc tính khác nhau trong chế độ text.

```
.MODEL Tiny
.CODE
      Org 100h
      Jmp Start
Start:
      Mov AL,0
      Mov BL,0 ; thuoc tính 0 với kí tu co ma ASCII =0
      Mov DL,0 ; DL= so cot, bat dau tu cot 0
      Mov DH,4 ; Bat dau tu hang thu 4
      Mov CX, 255 ; in 256 kí tự
InTiep:
   Push AX
   Mov AH,2
       Int 10h  ; dat vi tri con tro
       Pop AX
       Mov AH,9 ; in ki tu trong AL voi thuoc
                ; thuoc tinh trong BL
     Push AX
   Push CX
   Mov CX,1 ; in 1 ki tu
   Int 10h
  Pop CX
  Pop AX
  Inc AL ; ki tu ke tiep
  Inc BL ; thuoc tinh ke tiep
  Inc DL ; sang cot ben canh
```

```
       Cmp DL,79 ; Cot cuoi cung?
       Jb TiepTuc
       Mov Dl,0 ; xuong dong ke tiep
       Inc DH
     TiepTuc:
        Loop InTiep
       Int 20h
     End Start
```

Ví dụ 2: Vẽ một hình chữ nhật trong chế độ đồ họa VGA

```
.MODEL Tiny
.CODE
       Org 100h
       Jmp Start
       TopRow dw 100
       TopCol dw 100
       BotRow dw 400
       BotCol dw 600
       Color db 4 ; mau do
Start:
       Mov AH,0h ; thiet lap che do do hoa
       Mov AL,12h ; VGA mode
       Int 10h
       Mov CX, BotCol
       Sub CX,TopCol
       Mov SI,TopRow
       Mov DI,TopCol
       ; Ve cach tren
TopSide:
     Call DrawPixel  ; goi chuong trinh con ve diem anh
       Inc DI  ;
       Loop TopSide
       Mov CX, BotRow
       Sub CX, TopRow
       Mov SI, TopRow
       ; Ve cach  2 canh ben
Side:
       Mov DI, TopCol
  Call DrawPixel  ; goi chuong trinh con ve diem anh
       Mov DI, BotCol
  Call DrawPixel  ; goi chuong trinh con ve diem anh
       Inc SI
       Loop Side
       Mov CX, BotCol
```

```
        Sub CX,TopCol
        Mov SI, BotRow
        Mov DI, TopCol
        ; Ve cach duoi
BotSide:
      Call DrawPixel  ; goi chuong trinh con ve diem anh
        Inc DI  ;
        Loop BotSide
        Mov AH, 1 ; Cho 1 phim go vao
        Int 21h
        Mov AH,0
        Mov AL,2
        Int 10h
        Int 20h
        ; chuong trinh con ve 1 diem anh
        DrawPixel Proc
        Push AX
        Push CX
        Push DX
        Mov DX, SI
        Mov CX,DI
        Mov AL, Color
        Mov AH,0CH
        Int 10h
        Pop DX
        Pop CX
        Pop AX
        Ret
        DrawPixel Endp
End Start
```

Ví dụ 3: Viết một điểm ảnh trực tiếp vào bộ nhớ hiển thị (không thông qua RAM).

Đây là cách nhanh nhất, tuy nhiên thủ tục viết vào bộ nhớ hiển thị phụ thuộc vào loai card điều khiển màn hình cụ thể.

Chương trình sau được thực hiện trong chế độ đồ họa (640x200x2).

```
VeDiemAnh Proc Near
; DX=hàng, CX= cột
; ES=B800H
; Giả sử chế độ đồ họa đã được khởi tạo như sau:
; AH=0, AL=6, Int 10h

        Xor BX,BX ; BX=0
        SHR DX,1 ; kiem ta xem diem anh o dong chan hay le
        Add BX, 8*1024 ; o dong chan
```

```
TinhTiep:
 Mov AX,DX ; DX =so hang
 Mov CL,2
      SHL DX,CX ; DX=DX*4
      Add DX,AX ; DX=5*DX
      Mov AX,CX
      Mov CL,4
      SHL DX,CL ; DX=DX*16
      Add BX,DX ; BX=offset cua dong chua diem anh
      Mov DX,AX
      AND DX,7 ; DX=DX Mod 8 = vi tri byte tren dong
      Mov CL,3 ; no chua diem anh
      SHR AX,CL; AX=AX div 8= vi tri byte tren dong
      Add, BX,AX ; [BX] la byte chua diem anh
      NEG DL
      Add DL,7
      Mov CL,DL
      Mov AL,1
      SHL AL,CL
      OR ES:[BX], AL
      Ret
VeDiemAnh Endp
```

4.3 LẬP TRÌNH HỢP NGỮ TRONG WINDOWS

Phần này giới thiệu về hợp ngữ ở mức cao (High-Level Assembly) hay HLA và các công cụ sử dụng để lập trình trên windows.

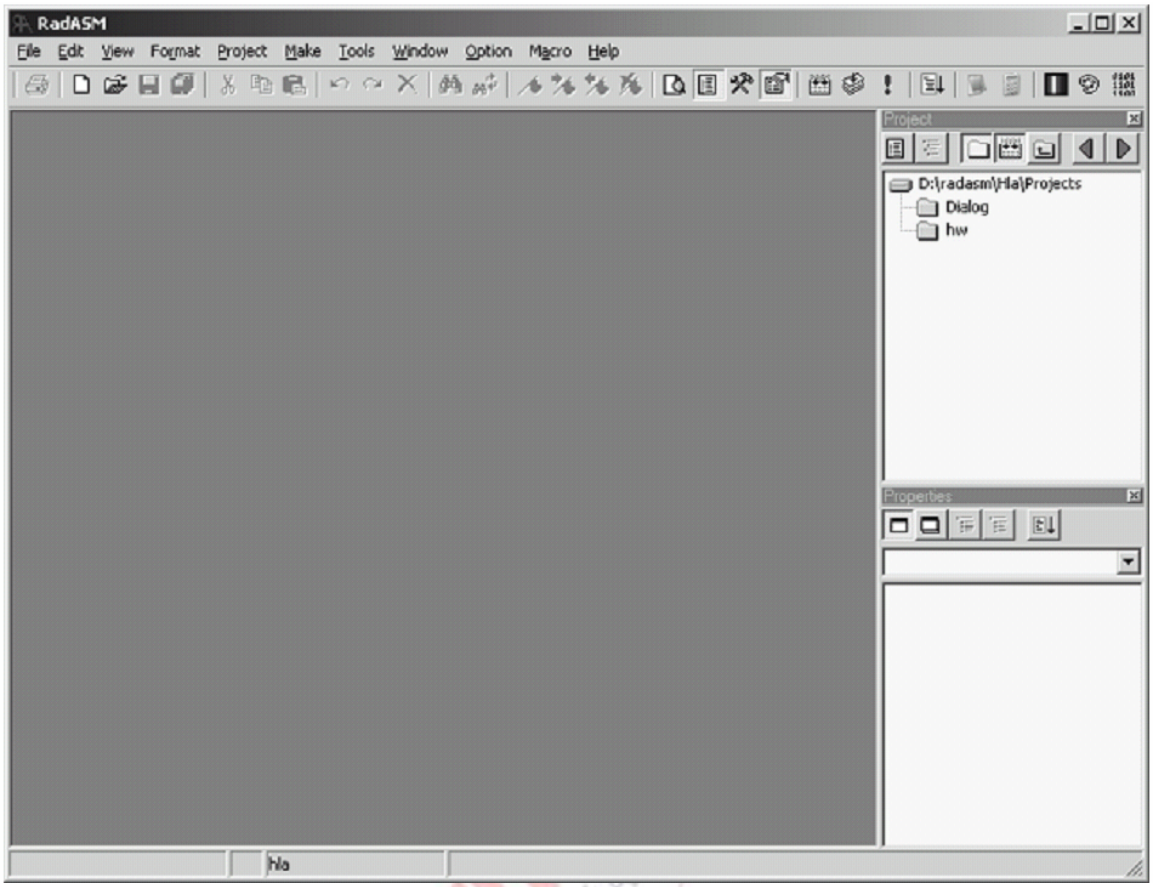
4.3.1 Công cụ hỗ trợ lập trình hợp ngữ trên windows

Dưới đây là công cụ để hỗ trợ người lập trình viết chương trình bằng hợp ngữ bậc cao (HLA) trên windows.

- Ngôn ngữ HLA
- Thư viện chuẩn HLA
- RadASM: môi trường phát triển ứng dụng cho HLA.
- Bộ công cụ gỡ rối OllyDbg (Olly Debugger)
- Chương trình xử lý đầu ra để output ra các file PE/COFF (như MASM, FASM)
- Một bộ liên kết (linker) có thể kết nối được các file output từ bộ dịch.(MS link)
- Một trình biên dịch tài nguyên (resource compiler) như rc.exe của Microsoft.
- Một số tiện ích cho phép xây dựng ứng dụng (như nmake của Microsoft hoặc make của Borland)
- Các module thư viện Win32.

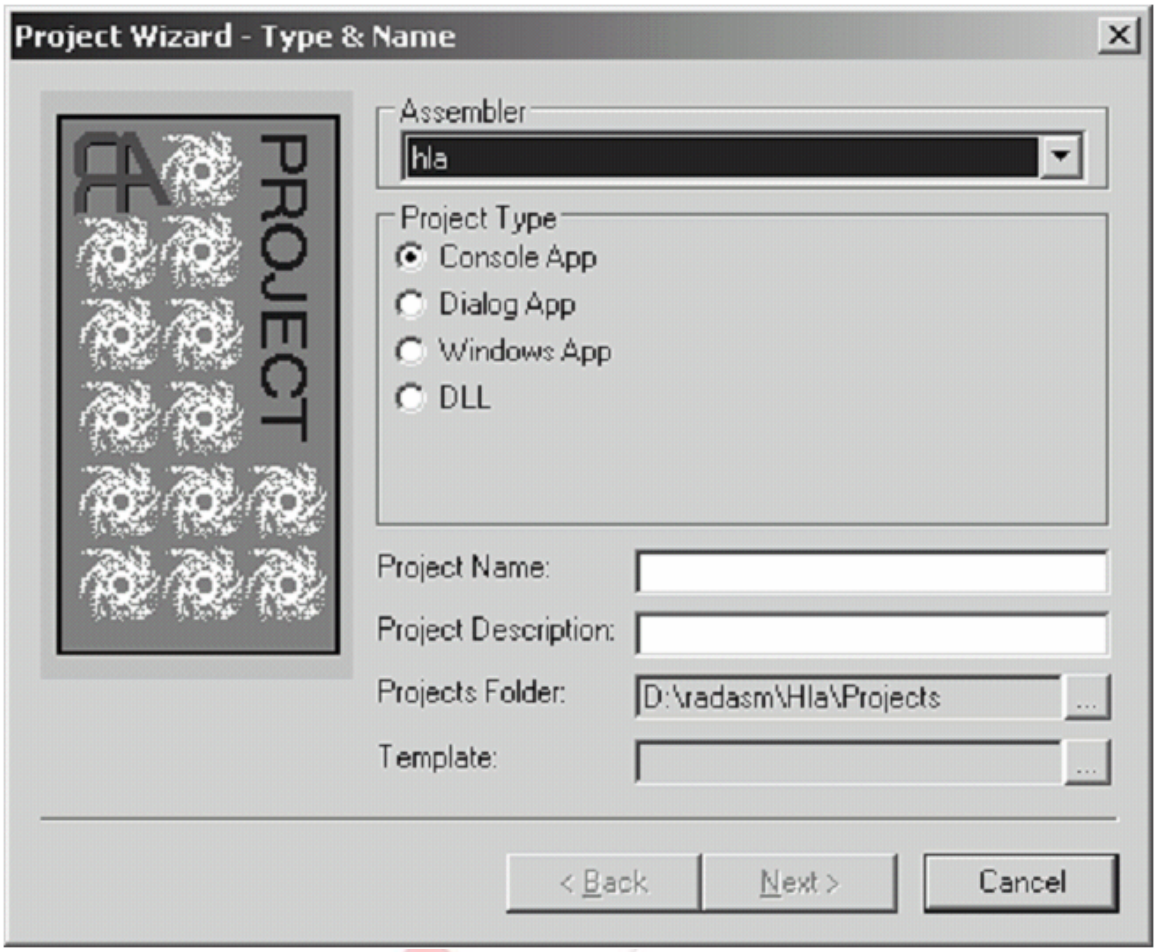
4.3.2 Sử dụng công cụ phát triển RadASM

Khi khởi động, cửa sổ chính màn hình RadASM như sau:



Nó bao gồm menu chính, một cửa số dành cho các file thuộc về một dự án va cửa sổ thuộc tính của các file/ các đối tượng của dự án.

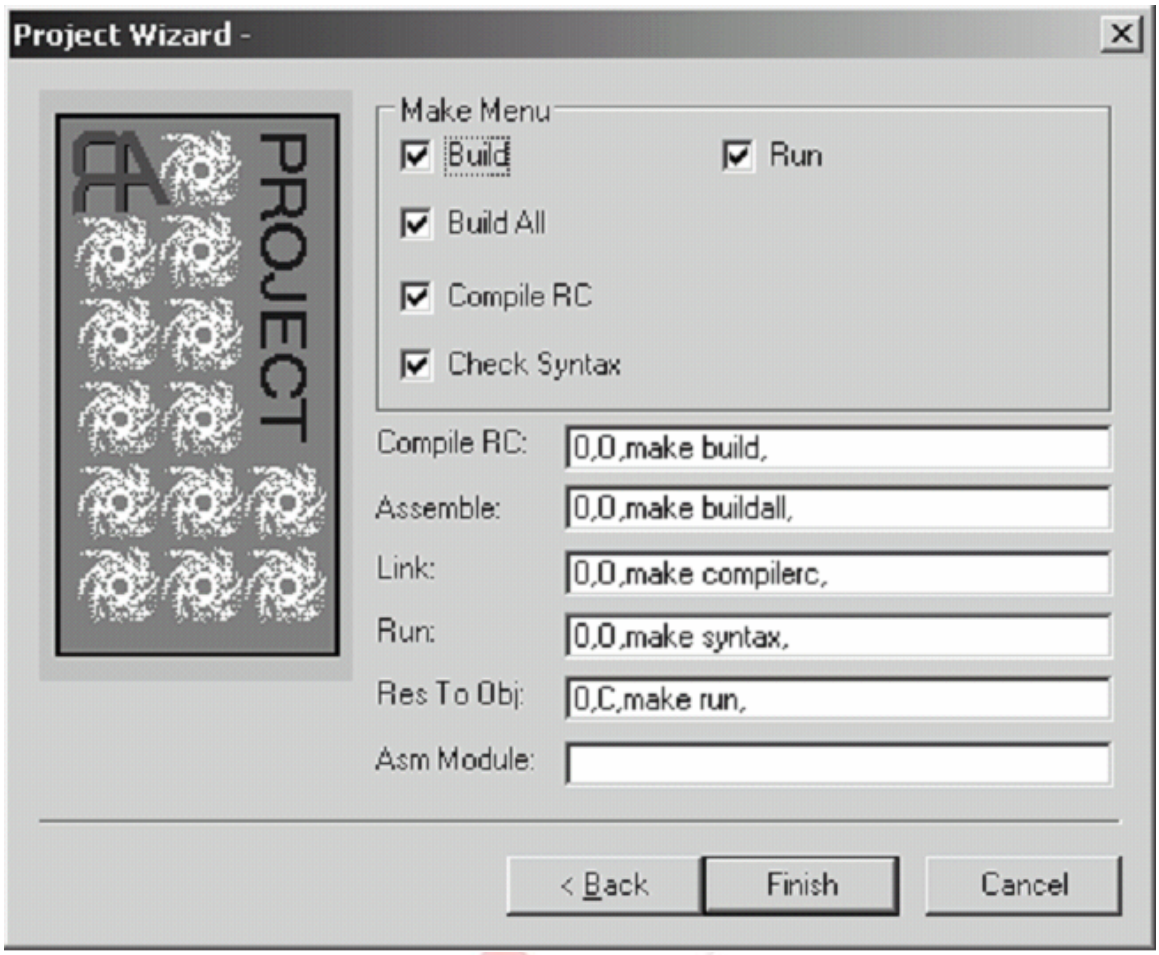
Để thực hiện tạo một Project mới, người dùng vào menu File, chọn new project. Khi đó màn hình thông tin của project mới sẽ được hiện ra như hình sau:



Người lập trình có thể chọn một trong số các mẫu ứng dụng của project đã có sẵn là một trong bốn loại: ứng dụng Console (input và output của chương trình đều hiển thị trên cửa số console), ứng dụng kiểu hộp thoại (chương trình [input và output] và người dùng tương tác với nhau qua các hộp thoại), ứng dụng windows thông thường, tạo các hàm để dịch ra thư viện liên kết động (DLL)

Tiếp đến, người lập trình đưa vào tên project, các mô tả của project, thư mục chứa project và template sử dụng cho project.

Sau khi soạn xong chương trình thì người lập trình tiến hành dịch, debug… rồi chạy chương trình. Cửa sổ dưới đây giúp người lập trình thực hiện việc đó:



Tùy theo từng giai đoạn phát triển ứng dụng Người lập trình của thể thực hiện các thao tác sau để dịch, kểm tra lỗi hoặc thực hiện chương trình.

4.4 TÓM TẮT

Chương này đã trình bày về phương thức truyền thông tin nối tiếp với bộ điều hợp UART 8250A. Trong phần này, ta tìm hiểu về cơ chế truyền thông tin nối tiếp kiểu đồng bộ, dị bộ, các thanh ghi bên trong của UART 8250A và lập trình cho UART 8250A. Tiếp theo là phần lập trình phối ghép cho bàn phím: nguyên tắc hoạt động của bàn phím, bộ đệm bàn phím, vùng dữ liệu, sử dụng các dịch vụ của BIOS để lập trình điều khiển bàn phím thông qua một số ví dụ đơn giản mang tính minh họa. Tương tự, ta cũng tìm hiểu về cách lập trình điều khiển màn hình thông qua hai phương pháp: sử dụng dịch vụ của BIOS và truy nhập trực tiếp vào bộ nhớ màn hình. Tuy nhiên, cách thứ hai tương đối khó vi nó phụ thuộc vào loại card điều khiển màn hinh và người lập trình phải cài đặt một ánh xạ từ bộ nhớ RAM sang bộ nhớ màn hình, nên cách này được xem như một sự tham khảo cho người lập trình hệ thống.

Phần cuối cùng là phần giới thiệu trình hợp ngữ trong Windows thông qua công cụ RadASM. Ngoài một số đối tượng, chức năng được môi trường phát triển cho ứng dụng RadASM cung cấp dới dạng các thư viện, việc viết các ứng dụng bằng hợp ngữ trên windows cũng khá giống môi trường khác. Nên tác giả không đề cập kỹ ở cuốn sách này. Đây là phần chỉ mang tính chất giới thiệu cho độc giả.

4.5 BÀI TẬP

4.5.1 Câu hỏi trắc nghiệm

1. Lý do chính của việc truyền thông nối tiếp giữa hai đối tượng cần truyền tin cho nhau là:

A. Truyền thông nối tiếp là công nghệ mới

B. Truyền thông giữa hai đối tượng truyền/nhận thong tin ở khoảng cách xa nhau.

C. Truyền thông nối tiếp là nhanh hơn truyền thông song song

D. Truyền thông nối tiếp chính xác hơn song song.

2. Phát biểu nào sau đây là đúng nhất:

A. Truyền tin dị bộ truyền theo từng kí tự còn truyền đồng bộ truyền theo nhiều kí tự.

B. Truyền tin dị bộ là kiểu truyền tin song song còn truyền đồng bộ là truyền tin nối tiếp.

C. Truyền tin đồng bộ là truyền song song còn truyền tin dị bộ là kiểu truyền tin nối tiếp

D. Cách mã hóa kí tự trong hai kiểu truyền tin đồng bộ và dị bộ luôn luôn khác nhau..

3. Thanh ghi nào dưới đây quyết định khuôn dạng dữ liệu khi truyền:

A. Thanh ghi trạng thái modem.

B. Thanh ghi trạng thái đường truyền.

C. Thanh ghi nháp

D. Thanh ghi điều khiển đường truyền..

4. Phát biểu nào sau đây là đúng nhất cho một khung truyền kiểu dị bộ:

A. Dành ra ít nhất là 6 bít để mã hóa dữ liệu.

B. Dành ra ít nhất là 7 bít để mã hóa dữ liệu.

C. Có ít nhất 1 bít Start

D. Có ít nhất là 2 bít Stop.

5. Phát biểu nào sau đây là đúng nhất về bít parity trong khung truyền kiểu dị bộ:

A. Trong một khung truyền có ít nhất 1 bít parity.

B Trong một khung truyền có thể không có bít parity nào.

C. Trong một khung truyền có nhiều nhất 2 bít parity.

D. Bít parity lưu trạng.thái của khung truyền.

6. Khi người dùng nhấn phím Ctrl_NumLock, thì máy tính sẽ:

A. Lưu mã Scan của hai phím Ctrl và NumLock vào bộ đệm bàn phím.

B Thực hiện một vòng lặp mà không lưu lại gì vào trong vùng đệm bàn phím.

C. Lưu mã ASCII của hai phím Ctrl và NumLock vào bộ đệm bàn phím.

D. Lưu cả mã Scan và mã ASCII của hai phím Ctrl và NumLock vào bộ đệm bàn phím.

7. Tại một thời điểm, bộ đệm bàn phím có thể chứa được tối đa:

A. 16 mã phím.

B 8 mã phím

C. 32 mã phím

D. 24 mã phím

8. Nếu con trỏ Head bằng con trỏ Tail thì:

A. Bộ đệm bàn phím đầy.

B Có 8 mã phím

C. Có 16 mã phím

D. Bộ đệm bàn phím rỗng

4.5.2 Bài tập Lập trình

1. Hãy viết chương trình hợp ngữ tự động bật tổ hợp các phím CapLock, Insert, NumLock

2. hãy lập trình để khởi tạo chế độ làm việc một mạch UART 8250A cho cổng COM2 với các thông số:

a. 6 bít mã kí tự truyền , tốc độ truyền 1200 bits/ giây, parity lẻ, hai bít stop.

b. 7 bít mã kí tự truyền , tốc độ truyền 9600 bits/ giây, parity chẵn 1.5 bít stop.

3. Sử dụng các dịch vụ ngắt của BIOS phục vụ cho màn hình (ngắt 10h) trong chế độ dồ họa

để vẽ ra màn hình các hình sau:

a. Tam giác

b. Hình vuông

c. Hình tròn

d. Hình Parabol

4.6 TÀI LIỆU THAM KHẢO

1. Nguyễn Nam Trung. Cấu trúc máy Vi tính và Thiết bị ngoại vi. Nhà XB Khoa học Kỹ thuật.2000.

2. Hồ Khánh Lâm. Giáo trình Kỹ thuật Vi xử lý Tập 1&2. Nhà XB Bưu điện. 2006.

3. Đặng Thành Phu. Turbo Assembler và Ứng dụng. Nhà XB Khoa học và Kỹ thuật. 1998.

4.Nguyễn Minh San. Cẩm nang Lập trình Hệ thống (tập 2). Bản dịch. Nhà XB Tài chính Thống Kê. 1996.

5.Nguyễn Đình Việt. Giáo Trình nhập môn Hợp ngữ và Lập trình Hệ thống. Hà nội. 1998.

TÀI LIỆU THAM KHẢO

1. Đặng Thành Phu. Turbo Assembler và Ứng dụng. Nhà XB Khoa học và Kỹ thuật. 1998.

2. Nguyễn Nam Trung. Cấu trúc máy Vi tính và Thiết bị ngoại vi. Nhà XB Khoa học Kỹ thuật.2000.

3. Hồ Khánh Lâm. Giáo trình Kỹ thuật Vi xử lý Tập 1&2. Nhà XB Bưu điện. 2006.

4.Nguyễn Minh San. Cẩm nang Lập trình Hệ thống (tập 1&2)- bản dịch. Nhà XB Tài chính Thống Kê. 1996.

5.Nguyễn Đình Việt. Giáo Trình nhập môn Hợp ngữ và Lập trình Hệ thống. Hà nội. 1998.

6. webstie : www.emu8086.com

7. Randall Hyde. Entire Windows Assembly Programming. University California Riverside, USA. 2003.

8. Văn Thế Minh. Kỹ thuật Vi xử lý. Nhà XB Giáo dục. 1997.

CHƯƠNG TRÌNH PTIT ĐÀO TẠO ĐẠI HỌC TỪ XA

MỤC LỤC

LỜI NÓI ĐẦU....1
CHƯƠNG 1: GIỚI THIỆU....3
1.1 CẤU TRÚC BỘ VI XỬ LÝ....3
1.1.1 Sơ đồ kiến trúc bộ Vi xử lý 8088....3
1.1.2 Chức năng các thành phần....4
1.2 MỘT SỐ CHỨC NĂNG CỦA NGẮT 21H....7
1.3 GIỚI THIỆU VỀ TẬP LỆNH CỦA 8088....9
1.3.1 Nhóm lệnh di chuyển dữ liệu....9
1.3.2 Nhóm các lệnh tính toán số học....11
1.3.3 Nhóm các lệnh thao tác bít....14
1.3.4 Nhóm các lệnh làm việc với xâu kí tự....17
1.3.5 Nhóm các lệnh nhảy....18
1.3.6 Các lệnh điều khiển khác....20
1.4 TÓM TẮT....21
1.5 CÂU HỎI VÀ BÀI TẬP....22
1.6 TÀI LIỆU THAM KHẢO....23
CHƯƠNG 2: LẬP TRÌNH BẰNG HỢP NGỮ....24
2.1 VIẾT VÀ THỰC HIỆN MỘT CHƯƠNG TRÌNH HỢP NGỮ....24
2.1.1 Cấu trúc lệnh và khai báo dữ liệu cho chương trình....24
2.1.2 Khung của chương trình Hợp ngữ....26
2.1.3 Tạo, dịch, hợp dịch và thực hiện chương trình Hợp ngữ....30
2.2 CÁC CẤU TRÚC LẬP TRÌNH CƠ BẢN TRONG CHƯƠNG TRÌNH HỢP NGỮ.31
2.2.1 Cấu trúc tuần tự....32
2.2.2 Cấu trúc IF… THEN....32
2.2.3 Cấu trúc IF… THEN…ELSE....33
2.2.4 Cấu trúc CASE....34
2.2.5 Cấu trúc lặp FOR-DO....36
2.2.6 Cấu trúc lặp WHILE-DO....37
2.2.7 Cấu trúc lặp REPEAT-UNTIL....38
2.3 CHƯƠNG TRÌNH CON VÀ MACRO....39
2.3.1 Chương trình con: cơ chế làm việc và cấu trúc....39
2.3.2 Truyền tham số....41
2.3.3 Chương trình gồm nhiều module....42
2.3.4 Liên kết thủ tục vào một thư viện....47

2.3.5 Macro 48
2.4 Chương trình ví dụ 53
2.5 TÓM TẮT 60
2.6 BÀI TẬP 61
2.7 TÀI LIỆU THAM KHẢO 61
CHƯƠNG 3. CÁC CÔNG CỤ HỖ TRỢ 62
3.1 BỘ GỠ RỐI DEBUG 62
3.1.1 Tổng quan về Debug 62
3.1.2 Sử dụng Debug 62
3.1.3 Các lệnh của Debug 63
3.2 CHƯƠNG TRÌNH MÔ PHỎNG EMU8086 69
3.2.1 Các chức năng soạn thảo, dịch và thực hiện chương trình. 69
3.2.2 Chức năng mô phỏng quá trình thực hiện chương trình 70
3.2.3 Các chương trình mẫu. 72
3.3. KẾT NỐI HỢP NGỮ VỚI CÁC NGÔN NGỮ BẬC CAO 80
3.3.1 Ngôn ngữ C và Hợp ngữ 80
3.3.2 Ngôn ngữ Pascal và Hợp ngữ 93
3.4 CÁC CHƯƠNG TRÌNH NGẮT 96
3.4.1 Ứng dụng các ngắt của BIOS & DOS 96
3.4.2 Chương trình thường trú và chương trình ngắt 108
3.5 TÓM TẮT 112
3.6 BÀI TẬP 113
3.6.1 Câu hỏi trắc nghiệm 113
3.6.2 Bài tập 114
3.7 TÀI LIỆU THAM KHẢO 115
CHƯƠNG 4: LẬP TRÌNH PHỐI GHÉP 116
4.1 TRUYỀN THÔNG TIN NỐI TIẾP VỚI BỘ ĐIỀU HỢP UART 116
4.1.1 Cơ bản về truyền tin nối tiếp 116
4.1.2 Các thanh ghi của UART 8250A/16450 117
4.1.3 Lập trình cho UART 8250A/16450 121
4.2 MỘT SỐ LẬP TRÌNH PHỐI GHÉP CƠ BẢN 125
4.2.1 Lập trình phối ghép với bàn phím 125
4.2.2 Lập trình phối ghép với màn hình 131
4.3 LẬP TRÌNH HỢP NGỮ TRONG WINDOWS 135
4.3.1 Công cụ hỗ trợ lập trình hợp ngữ trên windows 135
4.3.2 Sử dụng công cụ phát triển RadASM 136
4.4 TÓM TẮT 138

4.5 BÀI TẬP 139
4.5.1 Câu hỏi trắc nghiệm 139
4.5.2 Bài tập Lập trình 140
4.6 TÀI LIỆU THAM KHẢO 140
TÀI LIỆU THAM KHẢO 141
MỤC LỤC 142

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
Km10 Đường Nguyễn Trãi, Hà Đông, Hà Nội
Tel: (04).5541221, Fax: (04).5540587
Website: http://www.e-ptit.edu.vn; E-mail: dhtx@e-ptit.edu.vn
CHƯƠNG TRÌNH ĐÀO TẠO ĐẠI HỌC TỪ XA PTIT

LẬP TRÌNH HỆ THỐNG VÀ ĐIỀU KHIỂN THIẾT BỊ

Mã số: 492LHD350

Chịu trách nhiệm bản thảo

TRUNG TÂM ĐÀO TẠO BƯU CHÍNH VIỄN THÔNG 1

CHƯƠNG TRÌNH ĐÀO TẠO ĐẠI HỌC TỪ XA PTIT
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

TRƯỜNG ĐHDL DUY TÂN
KHOA TÀI CHÍNH NGÂN HÀNG

GIÁO TRÌNH

LẬP TRÌNH HỆ THỐNG MÁY TÍNH

Giáo trình he thong
download form www.geosoftvn.com
Tu sach MÁY TÍNH

Chương 1
KIẾN TRÚC VÀ HOẠT ĐỘNG CỦA HỆ VI XỬ LÝ / MÁY TÍNH

1. Cấu trúc luận lý

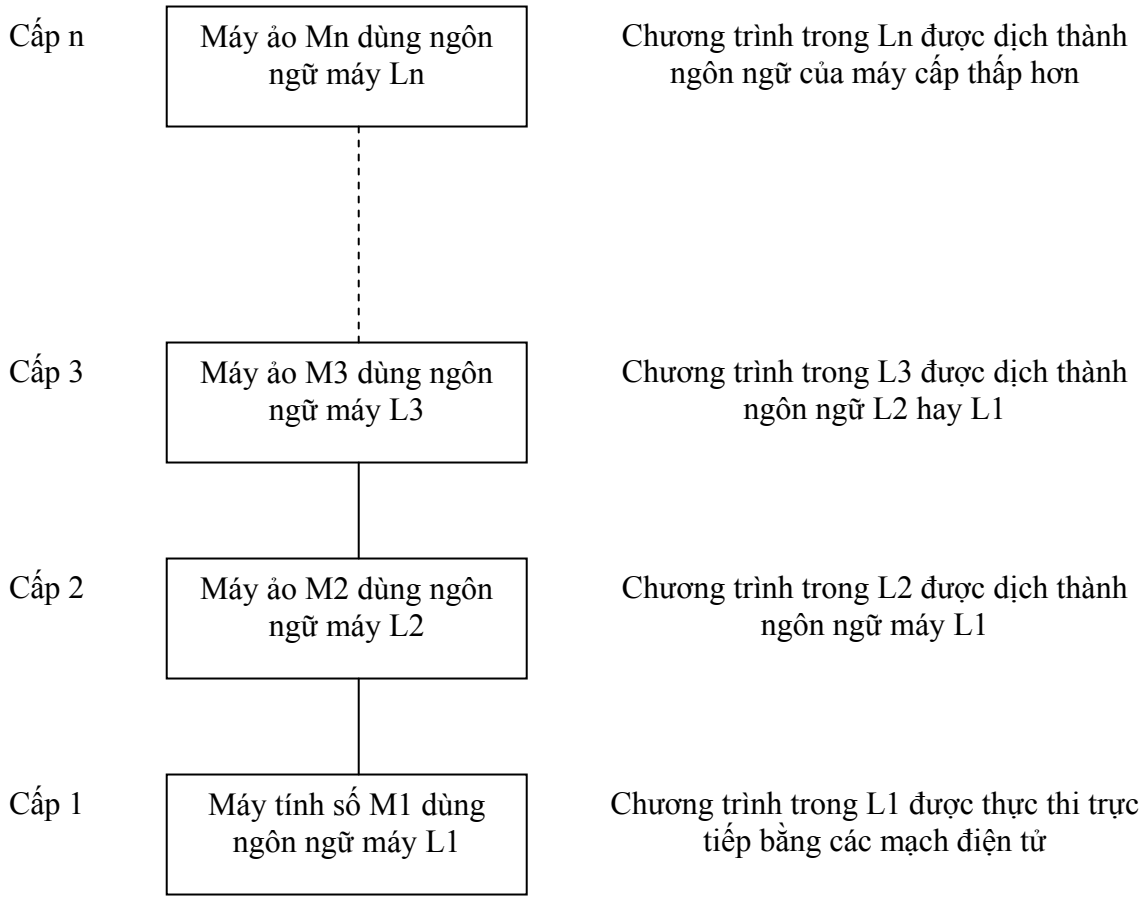
Máy tính số (Digital computer) là máy giải quyết các vấn đề bằng cách thực hiện các chỉ thị do con người cung cấp. Chuỗi các chỉ thị này gọi là chương trình (program). Các mạch điện tử trong một máy tính số sẽ thực hiện một số giới hạn các chỉ thị đơn giản cho trước. Tập hợp các chỉ thị này gọi là tập lệnh của máy tính. Tất cả các chương trình muốn thực thi đều phải được biến đổi sang tập lệnh trước khi được thi hành. Các lệnh cơ bản là:

- Cộng 2 số.
- So sánh với 0.
- Di chuyển dữ liệu.

Tập lệnh của máy tính tạo thành một ngôn ngữ giúp con người có thể tác động lên máy tính, ngôn ngữ này gọi là ngôn ngữ máy (machine language). Tuy nhiên, hầu hết các ngôn ngữ máy đều đơn giản nên để thực hiện một yêu cầu nào đó, người thiết kế phải thực hiện một công việc phức tạp. Đó là chuyển các yêu cầu này thành các chỉ thị có chứa trong tập lệnh của máy. Vấn đề này có thể giải quyết bằng cách thiết kế một tập lệnh mới thích hợp cho con người hơn tập lệnh đã cài đặt sẵn trong máy (built-in). Ngôn ngữ máy sẽ được gọi là ngôn ngữ cấp 1 (L1) và ngôn ngữ vừa được hình thành gọi là ngôn ngữ cấp 2 (L2).

Tuy nhiên, trong thực tế, để có thể thực hiện được, các ngôn ngữ L1 và L2 không được khác nhau nhiều. Như vậy, ngôn ngữ L2 cũng không thật sự giúp ích nhiều cho người thiết kế. Do đó, một tập lệnh kế tiếp được hình thành sẽ hướng về con người nhiều hơn là máy tính, tập lệnh này sẽ tạo thành một ngôn ngữ và ta gọi là ngôn ngữ L3. Ta có thể viết các chương trình trong L3 như là đã tồn tại máy tính sử dụng ngôn ngữ L3 (máy ảo L3). Các chương trình này sẽ được dịch sang ngôn ngữ L2 và được thực thi bằng một chương trình dịch L2.

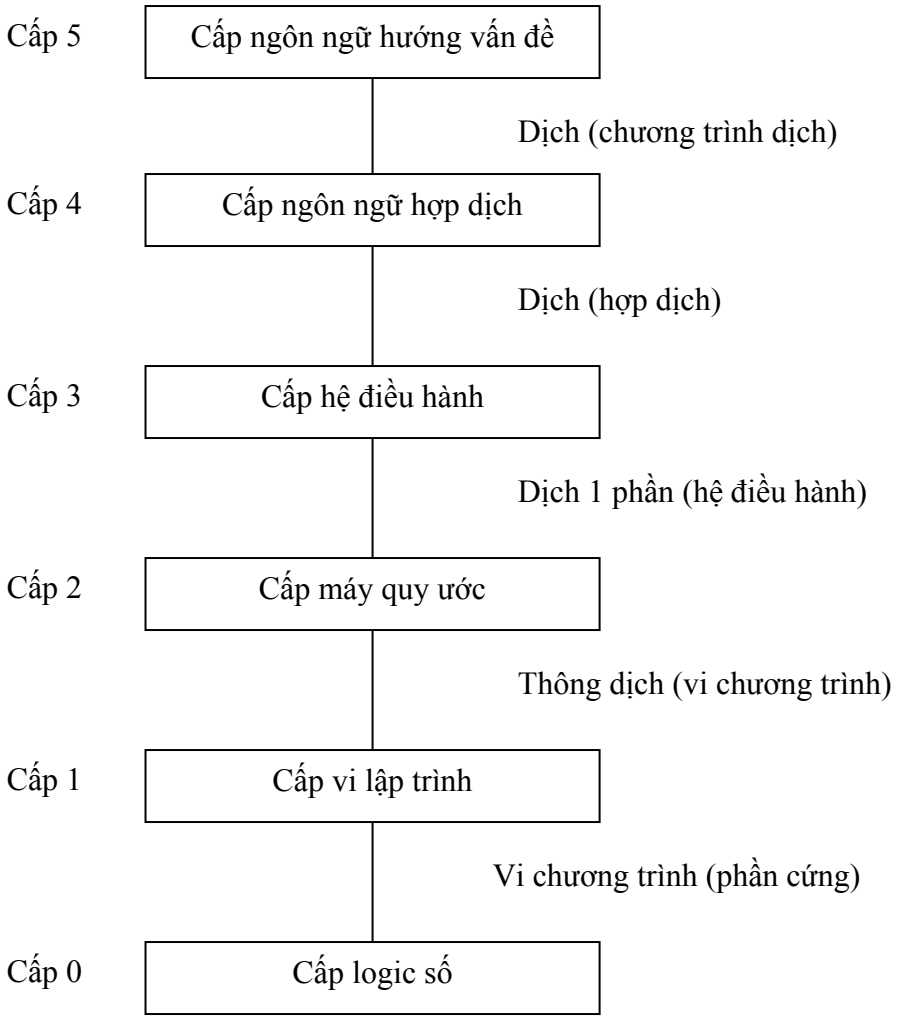
Việc xây dựng toàn bộ chuỗi các ngôn ngữ, mỗi ngôn ngữ được tạo ra sẽ thích hợp hơn ngôn ngữ trước đó sẽ có thể tiếp tục cho đến khi nhận được ngôn ngữ thích hợp nhất. Sơ đồ một máy ảo n cấp có thể biểu diễn như sau:



Hình 1.1. Máy ảo n cấp

Một máy tính số có n cấp có thể xem như có n-1 máy ảo khác nhau, mỗi máy ảo có một ngôn ngữ máy riêng. Các chương trình viết trên các máy ảo này không thể thực thi trực tiếp mà phải dịch thành các ngôn ngữ máy cấp thấp hơn. Chỉ có máy thật dùng ngôn ngữ máy L1 mới có thể thực thi trực tiếp bằng các mạch điện tử. Một lập trình viên sử dụng máy ảo cấp n không cần biết tất cả các trình dịch này. Chương trình trong máy ảo cấp n sẽ được thực thi bằng cách dịch thành ngôn ngữ máy cấp thấp hơn và ngôn ngữ máy này sẽ được dịch thành ngôn ngữ máy thấp hơn nữa hay dịch trực tiếp thành ngôn ngữ máy L1 và thực thi trực tiếp trên các mạch điện tử.

Về cơ bản, máy tính gồm có 6 cấp:



Hình 1.2 – Các cấp trên máy tính số

Cấp 0 chính là phần cứng của máy tính. Các mạch điện tử của cấp này sẽ thực thi các chương trình ngôn ngữ máy của cấp 1. Trong cấp logic số, đối tượng quan tâm là các cổng logic. Các cổng này được xây dựng từ một nhóm các transistor.

Cấp 1 là cấp ngôn ngữ máy thật sự. Cấp này có một chương trình gọi là vi chương trình (microprogram), vi chương trình có nhiệm vụ thông dịch các chỉ thị của cấp 2. Hầu hết các lệnh trong cấp này là di chuyển dữ liệu từ phần này đến phần khác của máy hay thực hiện việc một số kiểm tra đơn giản.

Mỗi máy cấp 1 có một hay nhiều vi chương trình chạy trên chúng. Mỗi vi chương trình xác định một ngôn ngữ cấp 2. Các máy cấp 2 đều có nhiều điểm chung ngay cả các máy cấp 2 của các hãng sản xuất khác nhau. Các lệnh trên máy cấp 2 được thực thi bằng cách thông dịch bởi vi chương trình mà không phải thực thi trực tiếp bằng phần cứng.

Cấp thứ 3 thường là cấp hỗn hợp. Hầu hết các lệnh trong ngôn ngữ của cấp máy này cũng có trong ngôn ngữ cấp 2 và đồng thời có thêm một tập lệnh mới, một tổ chức bộ

nhớ khác và khả năng chạy 2 hay nhiều chương trình song song. Các lệnh mới thêm vào sẽ được thực thi bằng một trình thông dịch chạy trên cấp 2, gọi là hệ điều hành. Nhiều lệnh cấp 3 được thực thi trực tiếp do vi chương trình và một số lệnh khác được thông dịch bằng hệ điều hành (do đó, cấp này là cấp hỗn hợp).

Cấp 4 thật sự là dạng tượng trưng cho một trong các ngôn ngữ. Cấp này cung cấp một phương pháp viết chương trình cho các cấp 1, 2, 3 dễ dàng hơn. Các chương trình viết bằng hợp ngữ được dịch sang các ngôn ngữ của cấp 1, 2, 3 và sau đó được thông dịch bằng các máy ảo hay thực tương ứng.

Cấp 5 bao gồm các ngôn ngữ được thiết kế cho người lập trình nhằm giải quyết một vấn đề cụ thể. Các ngôn ngữ này được gọi là cấp cao. Một số ngôn ngữ cấp cao như Basic, C, Cobol, Fortran, Lisp, Prolog, Pascal và các ngôn ngữ lập trình hướng đối tượng như C++, J++, … Các chương trình viết bằng các ngôn ngữ này thường được dịch sang cấp 3 hay 4 bằng các trình biên dịch (compiler).

2. Giao tiếp ngoại vi

Ta phân biệt tất cả 3 phương pháp xuất / nhập dữ liệu:

- Nhập / xuất bằng cách hỏi trạng thái của thiết bị ngoại vi (polling)
- Nhập / xuất bằng ngắt (interrupt).
- Nhập / xuất bằng cách truy xuất trực tiếp vào bộ nhớ dùng các phần cứng phụ trợ (DMA).

2.1. Nhập / xuất dữ liệu bằng cách hỏi vòng (polling)

Ta biết rằng vấn đề điều khiển nhập / xuất dữ liệu sẽ rất đơn giản trong trường hợp thiết bị ngoại vi lúc nào cũng có thể làm việc với µP. Ta có thể ví dụ như bộ hiển thị Led 7 đoạn lúc nào cũng sẵn sàng hiển thị dữ liệu khi mà µP gởi dữ liệu ra. Tuy nhiên, trong thực tế, không phải lúc nào µP cũng làm việc với các thiết bị ngoại vi có tính năng như trên. Ví dụ như khi làm việc với một máy in, µP yêu cầu in nhưng máy in không sẵn sàng (giả sử như hết giấy, kẹt giấy, …). Khi đó, µP phải kiểm tra xem một thiết bị mà nó cần giao tiếp có sẵn sàng hay không nếu thiết bị sẵn sàng thì mới thực hiện trao đổi dữ liệu. Để kiểm tra các thiết bị ngoại vi, µP phải sử dụng các tín hiệu bắt tay (handshake) xác định tuần tự từng thiết bị, xem thiết bị nào có yêu cầu trao đổi dữ liệu. Các tín hiệu này lấy từ các mạch giao tiếp do người thiết kế tạo ra.

Giả sử hệ thống có 2 thiết bị ngoại vi, nếu thiết bị 1 có dữ liệu cần truyền đến µP thì nó sẽ gởi 1 xung để chốt dữ liệu đồng thời tạo tín hiệu sẵn sàng cho thiết bị. Khi µP kiểm tra tín hiệu sẵn sàng của thiết bị 1 thì nó sẽ đọc dữ liệu vào từ mạch chốt và xoá tín hiệu sẵn sàng.

Trong trường hợp µP muốn gởi dữ liệu ra thiết bị 2, nó sẽ đọc tín hiệu sẵn sàng của thiết bị 2, nếu thiết bị 2 có thể nhận dữ liệu thì µP sẽ gởi dữ liệu ra mạch chốt và thiết bị 2 sẽ đọc dữ liệu vào.

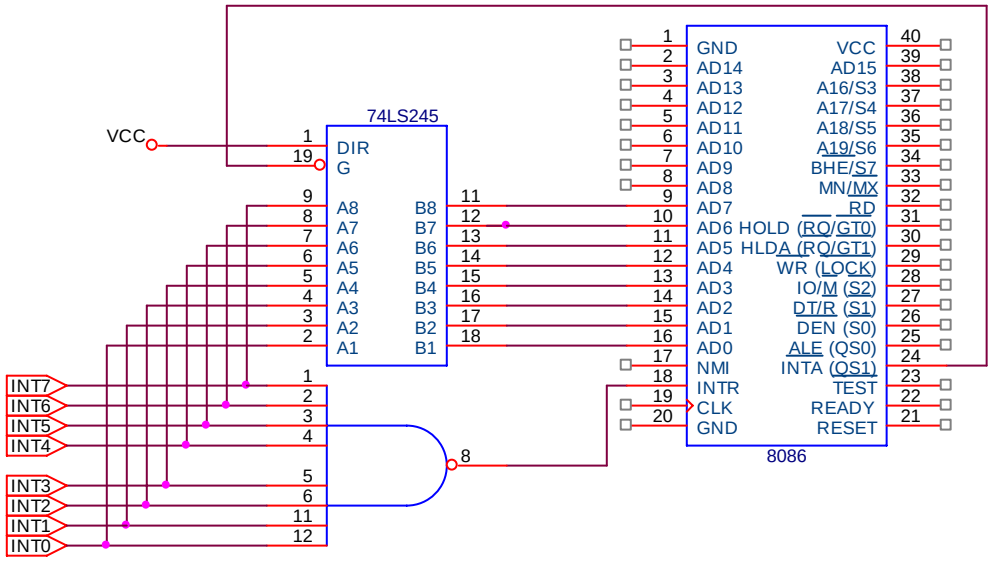
2.2. Ngắt và xử lý ngắt

Trong cách thức thực hiện trao đổi dữ liệu bằng cách hỏi vòng như trên, trước khi tiến hành thực hiện thì µP phải kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi. Tuy nhiên trong thực tế ta cần phải tận dụng khả năng của µP để làm các công việc khác mà không phải tốn thời gian kiểm tra thiết bị, chỉ khi nào có yêu cầu trao đổi dữ liệu thì mới tạm dừng công việc hiện tại. Cách làm việc như vậy gọi là ngắt µP, khi có một ngắt xảy ra thì ta phải thực hiện gọi các chương trình phục vụ ngắt tại các địa chỉ xác định của µP. Các tín hiệu ngắt từ thiết bị ngoại vi đưa vào µP thông qua các chân NMI hay INTR.

2.2.1. Các loại ngắt

❖ **Ngắt cứng**: là các yêu cầu ngắt từ các chân NMI hay INTR.

Ngắt cứng NMI là ngắt không che được còn ngắt cứng INTR có thể che được. Các lệnh CLI (Clear Interrupt) và STI (Set Interrupt) chỉ ảnh hưởng đến việc µP có chấp nhận yêu cầu ngắt tại chân INTR hay không. Yêu cầu ngắt tại chân INTR có thể có các kiểu ngắt từ 00h ÷ FFh. Kiểu ngắt này sẽ được đưa vào bus dữ liệu để µP xác định kiểu ngắt (dùng cho các thiết bị ngoại vi khác nhau).



Hình 1.3 – Kết nối ngắt đơn giản

❖ **Ngắt mềm**: là các ngắt thực hiện bằng phần mềm tác động do người sử dụng.

2.2.2. Đáp ứng của µP khi có yêu cầu ngắt

Khi có yêu cầu ngắt đến µP và nếu được phép ngắt, µP sẽ thực hiện các công việc sau:

- [SP] ← SP – 2, [SP] ← FR (Flag Register): cất thanh ghi cờ vào stack.
- IF ← 0, TF ← 0: không cho thực hiện các ngắt khác.
- SP ← SP – 2, [SP] ← CS: cất địa chỉ đoạn mã vào stack.
- SP ← SP – 2, [SP] ← IP: cất địa chỉ trở về sau khi phục vụ ngắt

- IP ← [Số_hiệu_ngắt*4], CS ← [Số_hiệu_ngắt*4 + 2]: lấy lệnh tại địa chỉ phục vụ ngắt tương ứng
- Sau khi kết thúc chương trình con phục vụ ngắt (khi gặp lệnh IRET):
  + IP ← [SP], SP ← SP + 2

  + CS ← [SP], SP ← SP + 2: lấy lại địa chỉ trước khi gọi chương trình phục vụ ngắt

  + FR ← [SP], SP ← SP + 2: lấy lại giá trị thanh ghi cờ

2.2.3. Xử lý ưu tiên ngắt

Như ta đã biết ở trên, khi μP đang thực hiện lệnh, nếu có ngắt xảy ra thì μP sẽ tạm ngừng chương trình và thực thi chương trình con phục vụ ngắt. Trong thực tế sẽ có trường hợp có nhiều yêu cầu ngắt khác nhau cùng một lúc, khi đó μP sẽ phục vụ cho ngắt theo thứ tự ưu tiên với nguyên tắc là ngắt nào có mức ưu tiên cao nhất thì sẽ phục vụ cho ngắt đó trước.

Các mức ưu tiên của các ngắt (từ mức thấp nhất đến mức cao nhất):

- Ngắt thực hiện chạy từng lệnh (INT 1)
- Ngắt che được INTR
- Ngắt không che được NMI
- Ngắt nội bộ (INT 0: xảy ra do phép chia số 0, ngắt mềm)

2.3. Nhập / xuất dữ liệu bằng DMA (Direct Memory Access)

Trong các phương thức trao đổi dữ liệu như hai phần trên đã trình bày thì việc trao đổi dữ liệu giữa thiết bị ngoại vi và hệ thống thường theo trình tự sau: ***từ ngoại vi đến vi xử lý rồi đi vào bộ nhớ*** **hay** ***từ bộ nhớ đến vi xử lý rồi ghi ra ngoại vi***. Trong thực tế sẽ có trường hợp ta cần thực hiện trao đổi dữ liệu ngay giữa ngoại vi và bộ nhớ. Khi đó người ta đưa ra cơ chế truy xuất bộ nhớ trực tiếp (DMA). Để thực hiện được vấn đề này, các hệ vi xử lý thông thường dùng thêm các mạch chuyên dụng để điều khiển quá trình truy xuất bộ nhớ trực tiếp (DMAC – Direct Memory Access Controller). Có tất cả 3 cơ chế hoạt động:

- ***Tận dụng thời gian CPU không dùng bus:***

Ta phải dùng thêm mạch phát hiện các chu kỳ xử lý nội của CPU và tận dụng các chu kỳ này để thực hiện trao đổi dữ liệu.

- ***Treo CPU để trao đổi từng byte:***

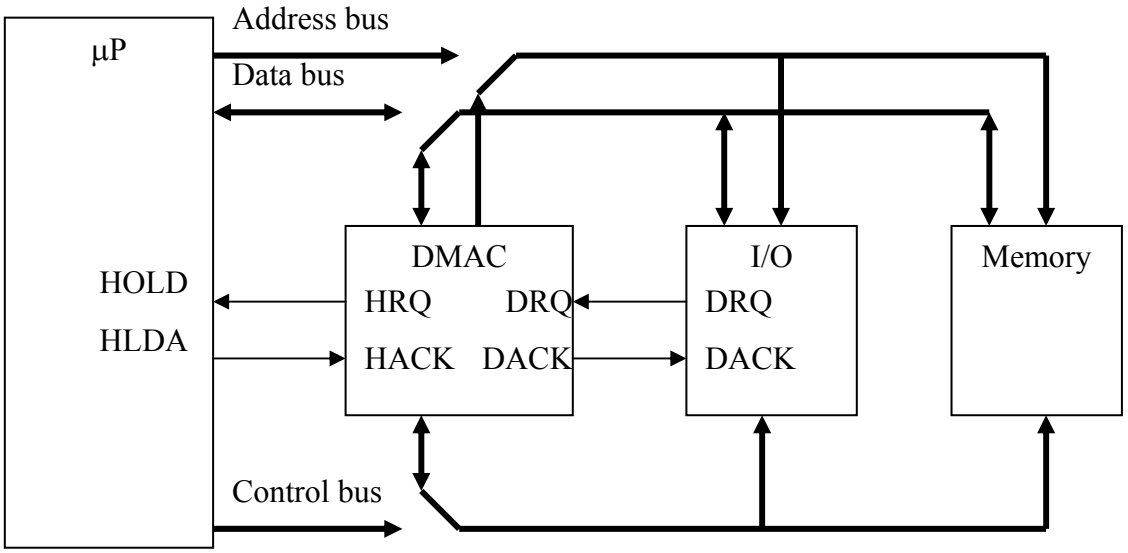
CPU không bị treo trong khoảng thời gian dài mà chỉ bị treo trong thời gian ngắn đủ để trao đổi 1 byte dữ liệu giữa bộ nhớ và ngoại vi. Do đó, công việc của CPU không bị gián đoạn mà chỉ bị chậm đi.

- ***Treo CPU một khoảng thời gian để trao đổi một khối dữ liệu:***

Trong cơ chế này, CPU bị treo trong suốt quá trình trao đổi dữ liệu.

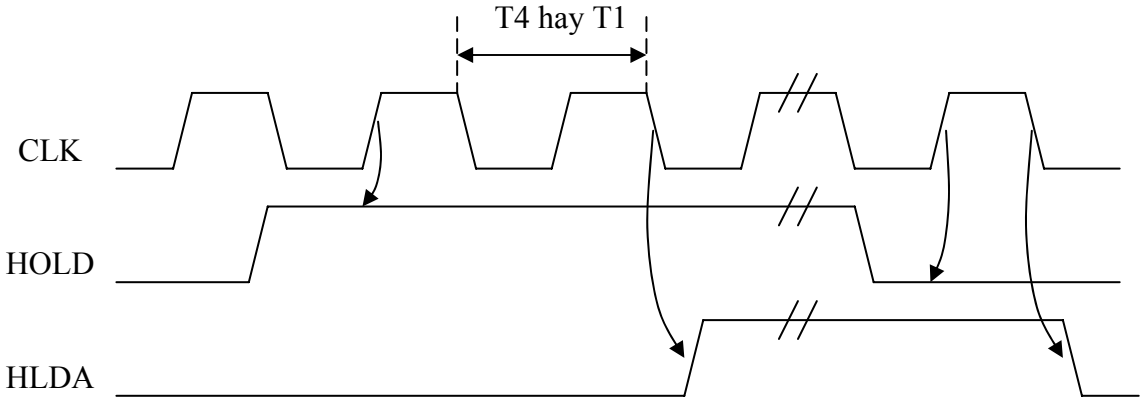
- CPU ghi từ lệnh và từ chế độ làm việc vào DMAC.
- Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu, nó gởi tín hiệu DRQ = 1 (DMA Request) đến DMAC.

- DMAC gởi tín hiệu HRQ (Hold Request) đến chân HOLD của CPU để yêu cầu treo CPU. Tín hiệu này sẽ giữ ở mức cao cho đến hết quá trình trao đổi dữ liệu.
- Sau khi nhận yêu cầu treo, CPU sẽ thực hiện hết chu kỳ bus của m?nh rồi treo các bus và gởi tín hiệu HLDA (Hold Acknowledge) để báo cho DMAC biết có thể sử dụng các bus.
- DMAC chuyển dữ liệu từ bộ nhớ đến ngoại vi bằng cách: đưa địa chỉ byte đầu tiên ra bus địa chỉ và đưa tín hiệu $\overline{\text{MEMR}}$ để đọc 1 byte từ bộ nhớ, kế tiếp DMAC đưa tín hiệu $\overline{\text{IOW}}$ để ghi dữ liệu ra ngoại vi. Sau đó, DMAC giảm số byte cần truyền, cập nhật địa chỉ bộ nhớ và lặp lại quá trình cho đến khi hết byte cần truyền.



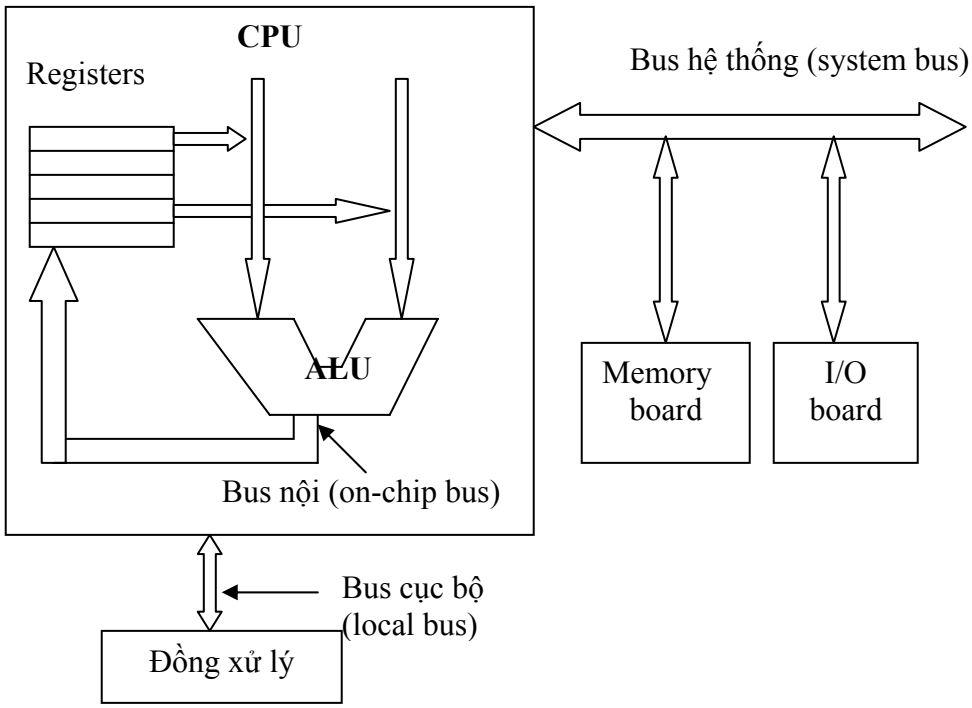
Hình 1.4 – Giao tiếp DMAC với hệ vi xử lý

Hai tín hiệu dùng để yêu cầu treo và chấp nhận yêu cầu treo CPU dùng cho cơ chế DMA là HOLD và HLDA có thể mô tả như sau:



Hình 1.5 – Tín hiệu HOLD và HLDA

3. Bus



Hình 1.6 - Các bus trong một hệ thống máy tính

Bus là đường truyền tín hiệu điện nối các thiết bị khác nhau trong một hệ thống máy tính. Bus thường có từ 50 đến 100 dây dẫn được gắn trên mainboard, trên các dây này có các đầu nối đưa ra, các đầu này được sắp xếp và cách nhau những khoảng quy định để có thể cắm vào đó những I/O board hay board bộ nhớ (bus hệ thống – system bus).

Cũng có những bus dùng cho mục đích chuyên biệt, thí dụ nối 1 vi xử lý với 1 hay nhiều vi xử lý khác hoặc nối với bộ nhớ cục bộ (local bus).

Trong vi xử lý cũng có một số bus để nối các thành phần bên trong của bộ vi xử lý với nhau. Người thiết kế chip vi xử lý có thể tuỳ ý lựa chọn loại bus bên trong nó, còn với các bus liên hệ bên ngoài cần phải xác định rõ các quy tắc làm việc cũng như các đặc điểm kỹ thuật về điện và cơ khí của bus để người thiết kế mainboard có thể ghép nối chip vi xử lý với các thiết bị khác. Nói cách khác, các bus này phải tuân theo 1 chuẩn nào đó. Tập các quy tắc của chuẩn còn được gọi là giao thức bus (bus protocol)

Thường có nhiều thiết bị nối với bus, một số thiết bị là tích cực (active) có thể đòi hỏi truyền thông trên bus, trong khi đó có các thiết bị thụ động chờ yêu cầu từ các thiết bị khác. Các thiết bị tích cực được gọi là chủ (master) còn thiết bị thụ động là tớ (slave).

Ví dụ: Khi CPU ra lệnh cho bộ điều khiển đĩa đọc/ghi một khối dữ liệu thì CPU là master còn bộ điều khiển đĩa là slave. Tuy nhiên, bộ điều khiển đĩa ra lệnh cho bộ nhớ nhận dữ liệu thì nó lại giữ vai trò master.

3.1. Bus Driver và Bus Receiver

Tín hiệu điện trong máy tính phát ra thường không đủ để điều khiển bus, nhất là khi bus khá dài và có nhiều thiết bị nối với nó. Chính vì thế mà hầu hết các bus master được nối với bus thông qua 1 chip gọi là ***bus driver***, về cơ bản nó là một bộ khuếch đại tín hiệu số. Tương tự như vậy, hầu hết các slave được nối với bus thông qua ***bus receiver***. Đối với các thiết bị khi thì đóng vai trò master, khi thì đóng vai trò slave, người ta sử dụng 1 chip kết hợp gọi là ***transceiver***. Các chip này đóng vai trò ghép nối và là các thiết bị 3 trạng thái, cho phép nó có thể ở trạng thái thứ 3 – hở mạch (thả nổi).

Giống như vi xử lý, bus có các đường địa chỉ, đường số liệu và đường điều khiển. Tuy nhiên, không nhất thiết có ánh xạ 1 – 1 giữa các tín hiệu ở các chân ra của vi xử lý và các đường dây của bus. Thí dụ: một số chíp vi xử lý có 3 chân ra, truyền ra các tín hiệu báo chíp vi xử lý đang thực hiện các thao tác $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$, $\overline{\text{IOW}}$ hay thao tác khác. Một bus điển hình thường có 4 đường trên.

Các vấn đề quan trọng nhất liên quan đến thiết kế bus là: xung clock bus (sự phân chia thời gian, hay còn gọi là **bus blocking**), cơ chế phân xử bus (bus arbitration), xử lý ngắt và xử lý lỗi.

Các bus có thể được chia theo giao thức truyền thông thành hai loại riêng biệt là bus đồng bộ và bus không đồng bộ phụ thuộc vào việc sử dụng clock bus.

3.2. Bus đồng bộ (Synchronous bus)

Mỗi chu kỳ bus bắt đầu bằng việc xuất địa chỉ bộ nhớ hoặc I/O port (chu kỳ xung nhịp T1). Bus điều khiển có 4 tín hiệu tác động mức thấp là $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ và $\overline{\text{IOW}}$.

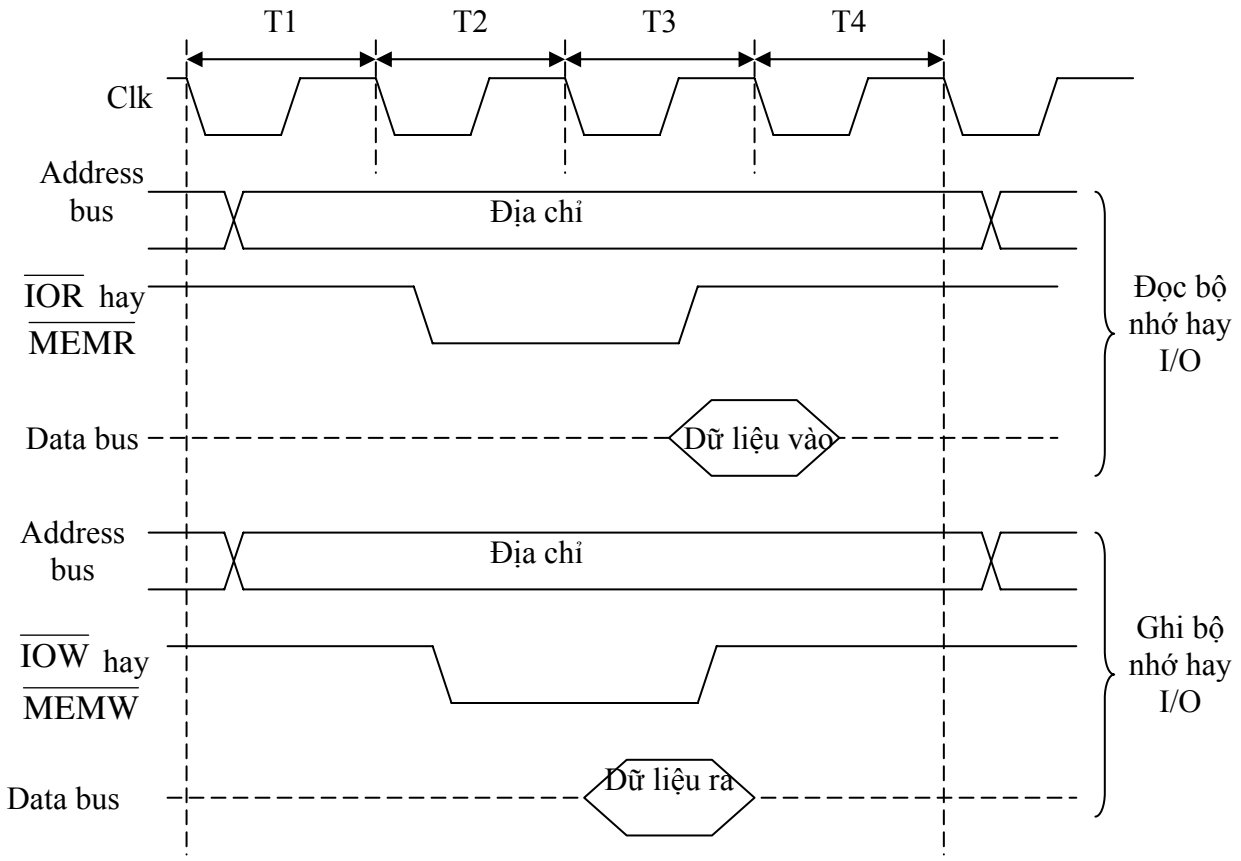
Các chuỗi sự kiện xảy ra trong một chu kỳ bus đọc bộ nhớ:

T1: µP xuất địa chỉ bộ nhớ 20 bit. Các đường dữ liệu không hoạt động và các đường điều khiển bị cấm

T2: Đường điều khiển $\overline{\text{MEMR}}$ xuống mức thấp. Đơn vị bộ nhớ ghi nhận chu kỳ bus này là quá trình đọc bộ nhớ và đặt byte hay word có địa chỉ đó lên data bus.

T3: µP đặt cấu hình để các đường data bus là nhập. Trạng thái này chủ yếu để bộ nhớ có thời gian tìm kiếm byte hay word dữ liệu

T4: µP đợi dữ liệu trên data bus. Do đó, nó thực hiện chốt data bus và giải phóng các đường điều khiển đọc bộ nhớ. Quá trình này sẽ kết thúc chu kỳ bus.



Hình 1.7 – Định thì chu kỳ bus đồng bộ

Trong một chu kỳ bus, µP có thể thực hiện đọc I/O, ghi I/O, đọc bộ nhớ hay ghi bộ nhớ. Các đường address bus và control bus dùng để xác định địa chỉ bộ nhớ hay I/O và hướng truyền dữ liệu trên data bus.

Chú ý rằng µP điều khiển tất cả các quá trình trên nên bộ nhớ bắt buộc phải cung cấp được dữ liệu vào lúc $\overline{MEMR}$ lên mức cao trong trạng thái T4. Nếu không, µP sẽ đọc dữ liệu ngẫu nhiên không mong muốn trên data bus. Để giải quyết vấn đề này, ta có thể dùng thêm các trạng thái chờ (wait state).

❖ **Truyền theo khối:**

Ngoài các chu kỳ đọc/ghi, một số bus truyền dữ liệu đồng bộ còn hỗ trợ truyền dữ liệu theo khối. Khi bắt đầu thao tác đọc khối, bus master báo cho slave biết số byte cần được truyền đi, thí dụ truyền con số này đi trong chu kỳ T1, sau đó đáng lẽ truyền đi 1 byte, slave đưa ra trong mỗi chu kỳ 1 byte cho tới khi đủ số byte được thông báo. Như vậy, khi đọc dữ liệu theo khối, n byte dữ liệu cần n+2 chu kỳ clock chứ không phải 3n chu kỳ.

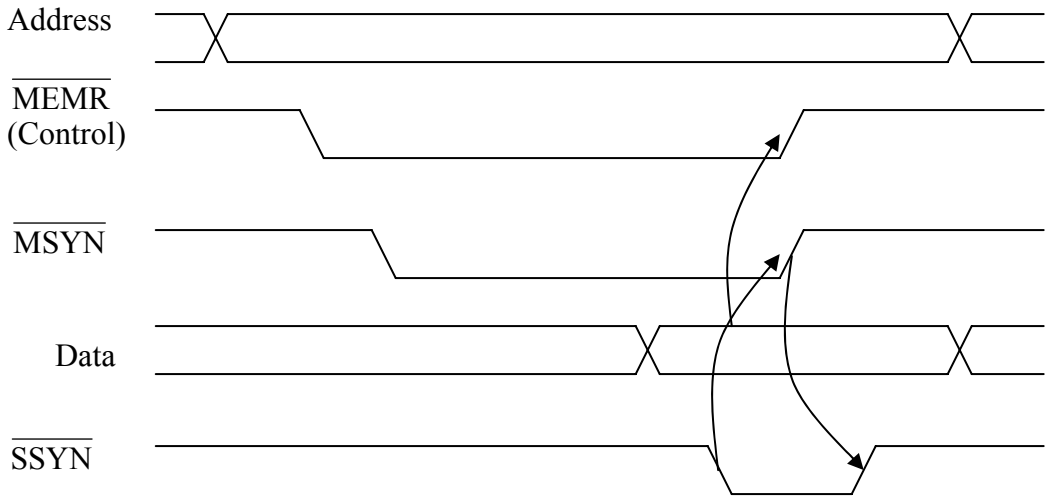
Một cách khác để cho truyền dữ liệu nhanh hơn là giảm chu kỳ. Tuy nhiên, giảm chu kỳ bus dẫn đến khó khăn về mặt kỹ thuật, các tín hiệu truyền trên các đường khác nhau không phải luôn có cùng tốc độ, dẫn đến hiệu ứng *bus skew*. Điều quan trọng là thời

gian chu kỳ phải dài hơn so với skew để tránh việc những khoảng thời gian được số hoá lại trở thành các đại lượng biến thiên liên tục.

3.3. Bus bất đồng bộ(Asynchronous bus)

Bus bất đồng bộ không sử dụng xung clock đồng bộ, chu kỳ của nó có thể kéo dài tuỳ ý và có thể khác nhau đối với các cặp thiết bị khác nhau. Làm việc với các bus đồng bộ dễ dàng hơn do nó được định thời một cách gián đoạn , tuy vậy chính đặc điểm này cũng dẫn đên nhược điểm. Mọi công việc được tiến hành trong khoảng thời gian là bội số của xung clock, nếu 1 thao tác nào đó của vi xử lý hay bộ nhớ hoàn thành trong 3.1 chu kỳ thì nó cũng sẽ phải kéo dài trong 4 chu kỳ. Khi đã chọn chu kỳ bus và đã xây dựng bộ nhớ, I/O card cho bus này thì khó có thể tận dụng những tiến bộ của công nghệ. Chẳng hạn sau khi đã xây bus với sự định thời như trên, công nghệ mới đưa ra các vi xử lý và bộ nhớ có thời gian chu kỳ là 100ns chứ không còn là 750ns như cũ, thì chúng vẫn chạy với tốc độ thấp như các vi xử lý, bộ nhớ loại cũ, bởi vì giao thức bus đòi hỏi bộ nhớ phải đưa được dữ liệu ra và ổn định trước thời điểm cạnh âm của T3. Nếu có nhiều thiết bị khác nhau cùng nối với 1 bus, trong đó có thể có một số thiết bị hoạt động nhanh hơn hơn các thiết bị khác thì cần phải đặt bus hoạt động phù hợp với thiết bị có tốc độ thấp nhất.

Bus bất đồng bộ ra đời nhằm khắc phục những nhược điểm của bus đồng bộ. Trước hết master phát ra địa chỉ nhớ mà nó muốn truy cập, sau đó phát tín hiệu $\overline{\text{MEMR}}$ tích cực để xác định cần truy xuất bộ nhớ và yêu cầu quá trình truy xuất là READ để xác định chiều truyền dữ liệu. Tín hiệu $\overline{\text{MEMR}}$ được đưa ra sau tín hiệu địa chỉ một khoảng thời gian phụ thuộc tốc độ hoạt động của master. Sau khi 2 tín hiệu này đã ổn định, master sẽ phát ra tín hiệu $\overline{\text{MSYN}}$ (master synchrization) ở mức tích cực để báo cho slave biết rằng các tín hiệu cần thiết đã sẵn sàng trên bus, slave có thể nhận lấy. Khi slave nhận được tín hiệu này, nó sẽ thực hiện công việc với tốc độ nhanh nhất có thể được, đưa dữ liệu của ô nhớ được yêu cầu lên bus dữ liệu. Khi hoàn thành slave sẽ phát tín hiệu $\overline{\text{SSYN}}$ (slave synchronization) tích cực.



Hình 1.8 – Định thì chu kỳ bus bất đồng bộ

Master nhận được tín hiệu $\overline{SSYN}$ tích cực thì xác định được dữ liệu của slave đã sẵn sàng nên thực hiện việc chốt dữ liệu, sau đó đảo các đường địa chỉ cũng như các tín hiệu $\overline{MEMR}$ và $\overline{MSYN}$. Khi slave nhận được tín hiệu $\overline{MSYN}$ không tích cực, nó xác định kết thúc chu kỳ và đảo tín hiệu $\overline{SSYN}$ làm bus trở lại trạng thái ban đầu, mọi tín hiệu đều không tích cực, chờ bus master mới.

Trên giản đồ thời gian của bus bất đồng bộ, ta sử dụng mũi tên để thể hiện nguyên nhân và kết quả. $\overline{MSYN}$ tích cực dẫn đến việc truyền dữ liệu ra bus dữ liệu và đồng thời cũng dẫn đến việc slave phát ra tín hiệu $\overline{SSYN}$ tích cực, đến lượt mình tín hiệu $\overline{SSYN}$ lại gây ra sự đảo mức của các đường địa chỉ, $\overline{MEMR}$ và $\overline{MSYN}$. Cuối cùng sự đảo mức của $\overline{MSYN}$ lại gây ra sự đảo mức tín hiệu $\overline{SSYN}$ và kết thúc chu kỳ. Tập các tín hiệu phối hợp với nhau như vậy được gọi là bắt tay toàn phần (full handshake), chủ yếu gồm 4 tín hiệu sau:

- $\overline{MSYN}$ tích cực.
- $\overline{SSYN}$ tích cực để đáp lại tín hiệu $\overline{MSYN}$.
- $\overline{MSYN}$ được đảo để đáp lại tín hiệu $\overline{SSYN}$ (tích cực).
- $\overline{SSYN}$ được đảo để đáp lại tín hiệu $\overline{MSYN}$ không tích cực.

Ta có thể nhận thấy bắt tay toàn phần là độc lập thời gian, mỗi sự kiện được gây ra bởi 1 sự kiện trước đó chứ không phải bởi xung clock. Nếu 1 cặp master-slave nào đó hoạt động chậm thì cặp master-slave kế tiếp không hề bị ảnh hưởng.

Tuy ưu điểm của bus bất đồng bộ rất rõ ràng, nhưng trong thực tế phần lớn các bus đang sử dụng là loại đồng bộ. Nguyên nhân là các hệ thống sử dụng bus đồng bộ dễ thiết kế hơn. Vi xử lý chỉ cần chuyển các mức tín hiệu cần thiết sang trạng thái tích cực là bộ nhớ đáp ứng ngay, không cần tín hiệu phản hồi. Chỉ cần các chọn phù hợp thì mọi hoạt động đều trôi chảy, không cần phải bắt tay.

3.4. Xử lý ngắt

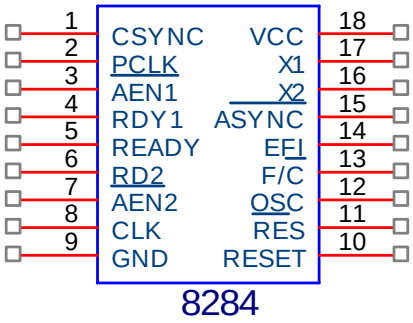
Ở trên, ta chỉ khảo sát các chu kỳ bus thông thường, trong đó master nhận hay gởi thông tin từ / đến slave. Một ứng dụng quan trọng nữa của bus là dùng để xử lý ngắt. Khi CPU ra lệnh cho thiết bị I/O làm một việc gì đó, nó thường chờ đợi tín hiệu ngắt do thiết bị I/O phát ra khi hoàn thành công việc được CPU yêu cầu. Khi nhận được tín hiệu ngắt, CPU sẽ đáp ứng ngay, có thể nhận dữ liệu do thiết bị I/O truyền về, hay gởi tiếp dữ liệu ra thiết bị I/O, hay CPU sẽ sử dụng bus cho một thao tác khác…. Như vậy chính ngắt phát ra tín hiệu yêu cầu sử dụng bus.

Vì có thể nhiều thiết bị ngoại vi cùng phát ra ngắt, cho nên cần có 1 cơ chế phân xử giống như đối với các bus thông thường. Giải pháp thường dùng là gán các mức độ ưu tiên cho các thiết bị và sử dụng 1 arbiter tập trung để trao quyền ưu tiên cho các thiết bị quan trọng thường xuyên được sử dụng. Hiện trên thị trường có những chip điều khiển ngắt được tiêu chuẩn hóa và được sử dụng rộng rãi là chip 8259A. Có thể nối 8 chip điều khiển I/O tới các đầu IRx (Interrupt request) của 8259A. Khi có 1 thiết bị nào đó muốn ngắt, nó đặt mức tích cực lên chân Irx, 8259A nhận được tín hiệu tích cực ở 1 hay một số

đầu vào Irx thì sẽ đặt mức tích cực lên đầu dây INT. Tín hiệu INT sẽ truyền trực tiếp đến chân Interrupt của CPU. Khi CPU có thể xử lý được ngắt, nó gởi lại 1 tín hiệu chấp nhận ngắt cho 8259A. Lúc này, CPU chờ 8259A chỉ ra I/O nào yêu cầu ngắt, bằng cách gởi số hiệu của I/O đó lên bus dữ liệu (D0-D7) để đi đến CPU. Sau đó, phần cứng CPU sẽ sử dụng con số đó để tính chỉ số trong 1 bảng con trỏ -bảng vector ngắt (interrupt vector) để tìm địa chỉ chương trình con, cho chạy chương trình này để phục vụ ngắt. Các chương trình con này gọi là chương trình con xử lý ngắt.

4. Các chip hỗ trợ cho bộ xử lý trung tâm

4.1. Mạch tạo xung clock 8284



Hình 1.9 – Mạch tạo xung clock 8284

PCLK (Peripheral Clock): xung clock f = $f_X/6$ (f_X là tần số thạch anh) với chu kỳ bổn phận 50%.

CSYNC (Clock Synchronisation): ngõ vào xung đồng bộ chung khi hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải nối GND.

$\overline{AEN1}$, $\overline{AEN2}$ (Address Enable): cho phép chọn các chân tương ứng RDY1, RDY2 báo hiệu trạng thái sẵn sàng của bộ nhớ hay thiết bị ngoại vi.

RDY1, RDY2 (Bus ready): kết hợp với $\overline{AEN1}$, $\overline{AEN2}$ tạo các chu kỳ đợi ở CPU

READY: nối đến chân READY của µP.

CLK (Clock): xung clock f = $f_X/3$, nối với chân CLK của CPU.

RESET: nối với chân RESET của CPU, là tín hiệu khởi động lại toàn hệ thống.

$\overline{RES}$ (Reset Input): chân khởi động cho 8284, được nối với mạch RC để tự khởi động khi bật nguồn.

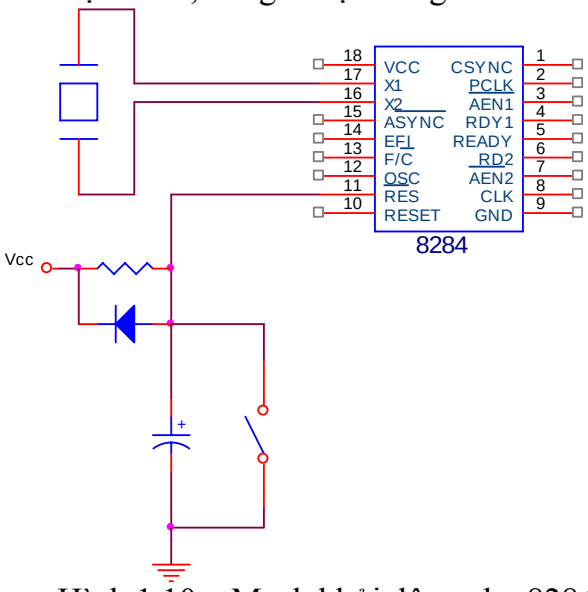
OSC: ngõ ra xung clock có tần số f_X.

$F/\overline{C}$ (Frequency / Crystal): chọn nguồn tín hiệu chuẩn cho 8284, nếu ở mức cao thì chọn tần số xung clock bên ngoài, ngược lại thì dùng xung clock từ thạch anh.

EFI (External Frequency Input): xung clock từ bộ dao động ngoài.

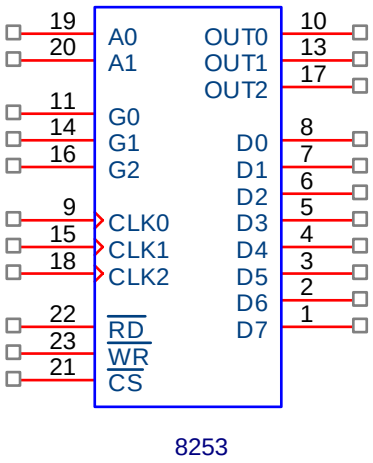
$\overline{ASYNC}$: chọn chế độ làm việc cho tín hiệu RDY. Nếu $\overline{ASYNC}$ = 1, tín hiệu RDY có ảnh hưởng đến tín hiệu READY cho đến khi có xung âm của xung clock. Ngược lại thì RDY chỉ ảnh hưởng khi xuất hiện xung âm.

X1,X2: ngõ vào của thạch anh, dùng để tạo xung chuẩn cho hệ thống.

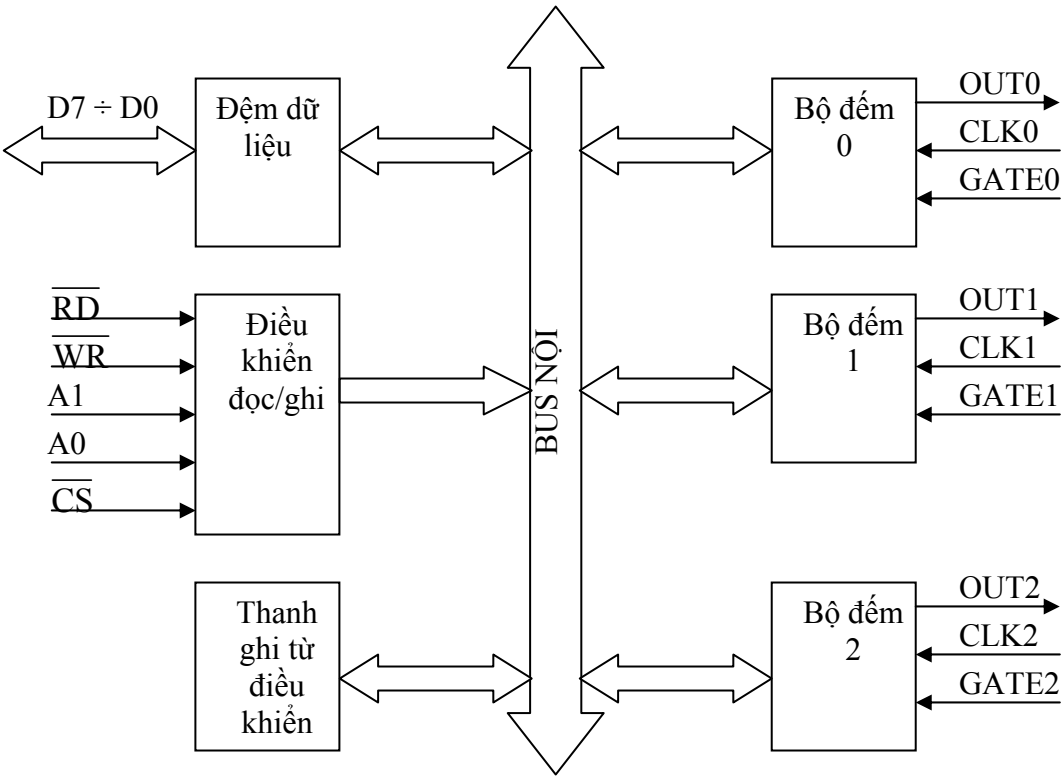


Hình 1.10 – Mạch khởi động cho 8284

4.2. Mạch định thời PIT – 8253 / 8254 (Programmable Interval Timer)



Hình 1.11 – Sơ đồ chân của PIT 8253



Hình 1.12 – Sơ đồ khối của PIT 8253

D7 ÷ D0: bus dữ liệu

CLK0 ÷ CLK2: ngõ vào xung clock cho các bộ đếm

OUT0 ÷ OUT2: ngõ ra bộ đếm

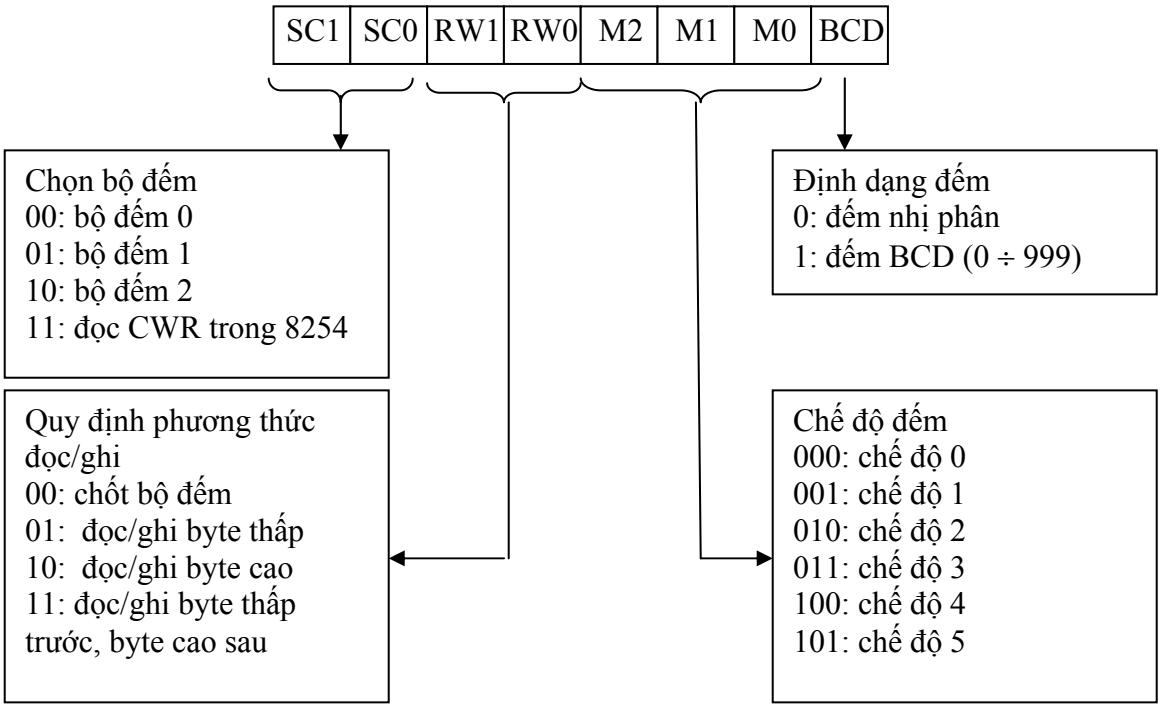
$\overline{RD}$, $\overline{WR}$: cho phép CPU đọc / ghi dữ liệu từ / đến các thanh ghi của 8253

A1, A0: giải mã chọn bộ đếm hay thanh ghi điều khiển, thường được nối với bus địa chỉ của CPU

A1	**A0**	**Chọn**
0	0	Bộ đếm 0
0	1	Bộ đếm 1
1	0	Bộ đếm 2
1	1	Thanh ghi từ điều khiển

G0 ÷ G2 (Gate): cho phép hay cấm các bộ đếm hoạt động (=1: cho phép, =0: cấm).

PIT 8253 có tất cả 5 chế độ đếm tùy thuộc vào giá trị trong thanh ghi điều khiển.



Hình 1.13 – Dạng từ điều khiển của 8253

PIT 8253 có 3 bộ đếm lùi 16 bit có thể lập trình và độc lập với nhau. Mỗi bộ đếm có tín hiệu xung clock riêng (8254 tương tự như 8253 nhưng có thêm lệnh đọc thanh ghi từ điều khiển CWR).

❖ **Các chế độ đếm:**

Chế độ 0 (Interrupt on Terminal Count): tín hiệu ngõ ra ở mức thấp cho tới khi bộ đếm tràn thì sẽ chuyển lên mức cao.

Chế độ 1 (Programmable Monoflop): tín hiệu ngõ ra chuyển xuống mức thấp tại cạnh âm của xung clock đầu tiên và sẽ chuyển lên mức cao khi bộ đếm kết thúc.

Chế độ 2 (Rate Generator): tín hiệu ngõ ra xuống mức thấp trong chu kỳ đầu tiên và sau đó chuyển lên mức cao trong các chu kỳ còn lại.

Chế độ 3 (Square-Wave Generator): tương tự như chế độ 2 nhưng xung ngõ ra là sóng vuông khi giá trị đếm chẵn và sẽ thêm một chu kỳ ở mức cao khi giá trị đếm lẻ.

Chế độ 4 (Software-triggered Pulse): giống như chế độ 2 nhưng xung Gate không khởi động quá trình đếm mà sẽ đếm ngay khi số đếm ban đầu được nạp. Ngõ ra ở mức cao để đếm và xuống mức thấp trong chu kỳ xung đếm. Sau đó, ngõ ra sẽ trở lại mức cao.

Chế độ 5 (Hardware-triggered Pulse): giống như chế độ 2 nhưng xung Gate không khởi động quá trình đếm mà được khởi động bằng cạnh dương của xung clock ngõ vào. Ngõ ra ở mức cao và xuống mức thấp sau một chu kỳ clock khi quá trình đếm kết thúc.

❖ **Ba chức năng của 8253 trong PC:**

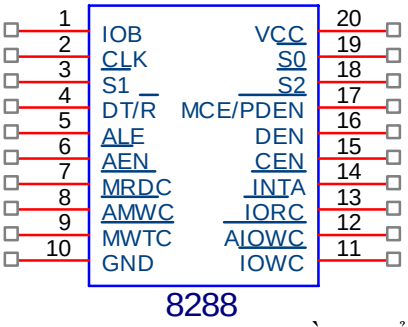
Cập nhật đồng hồ hệ thống: bộ đếm 0 của PIT phát tuần hoàn một ngắt cứng qua IRQ0 của 8259 để CPU có thể thay đổi đồng hồ hệ thống. Bộ đếm hoạt động trong chế độ 2. Ngõ vào được cấp xung clock tần số 1.19318 MHz. G0 = 1 để bộ đếm luôn được phép đếm. Giá trị ban đầu được nạp là 0 cho phép PIT phát ra xung chính xác với tần số:1.19318/65536 = 18.206Hz. Cạnh dương của mỗi xung này sẽ tạo ra một ngắt cứng trong 8259. Yêu cầu này sẽ dẫn tới ngắt 08h để cập nhật đồng hồ hệ thống 18.206 lần trong 1 giây.

Làm tươi bộ nhớ: PIT nối với chip DMAC dùng làm tươi bộ nhớ DRAM. Bộ đếm 1 sẽ định kỳ kích hoạt kênh 0 của DMAC-8237A để tiến hành 1 chu trình đọc giả làm tươi bộ nhớ. Bộ nhớ 1 hoạt động trong chế độ 3 phát sóng vuông với giá trị nạp ban đầu là 18. Do đó sóng vuông được phát ra có tần số 1,19318 MHz/18 = 66288 Hz (chu kỳ bằng 0.015s). Như vậy cứ sau 15 ms cạnh dương của sóng vuông này sẽ tạo 1 chu kỳ đọc giả để làm tươi bộ nhớ.

Phát sóng âm với tần số biến đổi ra loa của PC: Bộ đếm 2 của PIT được dùng để phát sóng âm ra loa của PC.

4.3. Mạch điều khiển bus 8288

Mạch điều khiển bus 8288 lấy một số tín hiệu điều khiển của CPU và cung cấp các tín hiệu điều khiển cần thiết cho hệ vi xử lý.



Hình 1.14 – Mạch điều khiển bus 8288

IOB (Input / Output Bus Mode): điều khiển để 8288 làm việc ở các chế độ bus khác nhau.

CLK (Clock): ngõ vào lấy từ xung clock hệ thống (từ 8284) và dùng để đồng bộ toàn bộ các xung điều khiển đi ra từ mạch 8288.

DT/$\overline{R}$ (Data Transmit/Receive): CPU truyền (1) hay nhận (0) dữ liệu.

ALE (Address Latch Enable): tín hiệu cho phép chốt địa chỉ, tín hiệu này thường được nối với chân G của 74573 để điều khiển chốt địa chỉ.

$\overline{AEN}$ (Address Enable): chờ thời gian trễ khoảng 150 ns sẽ tạo các tín hiệu điều khiển ở đầu ra của 8288 để đảm bảo rằng địa chỉ sử dụng đã hợp lệ.

$\overline{S2}$, $\overline{S1}$, $\overline{S0}$: các tín hiệu trạng thái lấy trực tiếp từ CPU. Tuỳ theo các giá trị nhận được mà 8288 sẽ đưa các tín hiệu theo bảng:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	**Tạo tín hiệu**
0	**0**	**0**	$\overline{INTA}$
0	**0**	**1**	$\overline{IORC}$
0	**1**	**0**	$\overline{IOWC}$, $\overline{AIOWC}$
0	**1**	**1**	**Không**
1	**0**	**0**	$\overline{MRDC}$
1	**0**	**1**	$\overline{MRDC}$
1	**1**	**0**	$\overline{MWTC}$, $\overline{AMWC}$
1	**1**	**1**	**Không**

$\overline{MRDC}$ (Memory Read Command): điều khiển đọc bộ nhớ

$\overline{MWTC}$ (Memory Write Command): điều khiển ghi bộ nhớ

$\overline{AMWC}$ (Advanced MWTC),: giống như $\overline{MWTC}$ nhưng hoạt động sớm hơn một chút dùng cho các bộ nhớ chậm đáp ứng kịp tốc độ CPU.

$\overline{IOWC}$ (I/O Write Command): điều khiển ghi ngoại vi

$\overline{AIOWC}$ (Advanced IOWC),: giống như $\overline{IOWC}$ nhưng hoạt động sớm hơn một chút dùng cho các ngoại vi chậm đáp ứng kịp tốc độ CPU.

$\overline{IORC}$ (I/O Read Command): điều khiển đọc ngoại vi

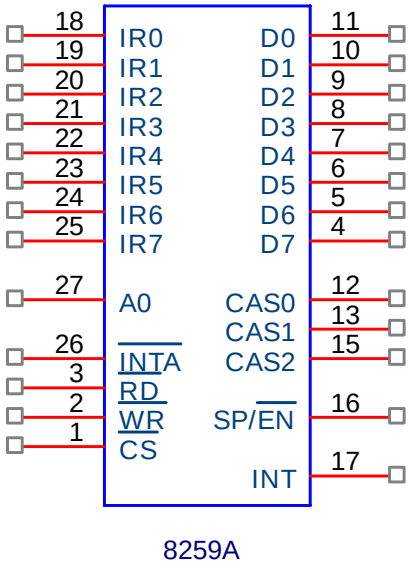
$\overline{INTA}$ (Interrupt Acknowledge): ngõ ra thông báo CPU chấp nhận yêu cầu ngắt của thiết bị ngoại vi

CEN (Command Enable): cho phép đưa ra tín hiệu DEN và các tín hiệu điều khiển khác của 8288.

DEN (Data Enable): điều khiển bus dữ liệu thành bus cục bộ hay bus hệ thống.

MCE / ***$\overline{PDEN}$*** (Master Cascade Enable / Peripheral Data Enable): định chế độ làm việc cho mạch điều khiển ngắt PIC 8259 để nó làm việc ở chế độ master.

4.4. Chip điều khiển ngắt ưu tiên PIC 8259A (Priority Interrupt Controller)



Hình 1.15 – Sơ đồ chân của 8259A

Trong trường hợp nhiều yêu cầu ngắt cần phải phục vụ, ta thường dùng vi mạch 8259A để giải quyết vấn đề ưu tiên. 8259A có thể giải quyết được 8 yêu cầu ngắt với 8 mức ưu tiên khác nhau.

❖ **Các khối chức năng:**

IRR (thanh ghi yêu cầu ngắt): lưu trữ các yêu cầu ngắt tại ngõ vào

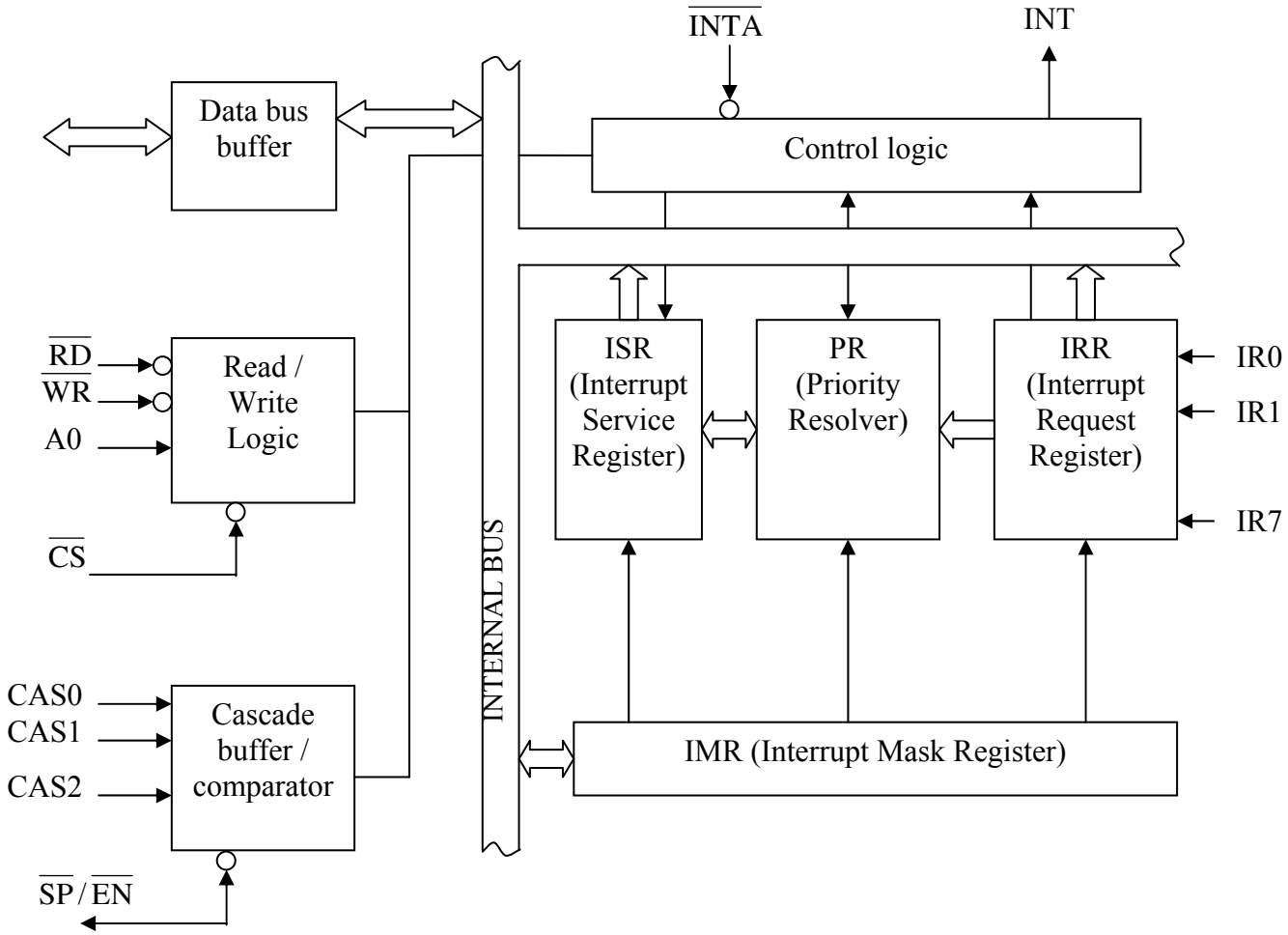
ISR (thanh ghi phục vụ ngắt): lưu trữ các yêu cầu ngắt đang phục vụ

IMR (thanh ghi mặt nạ ngắt): lưu trữ mặt nạ của các yêu cầu ngắt tại ngõ vào

Control logic (logic điều khiển): gởi yêu cầu ngắt tới chân INTR của CPU khi có tín hiệu ngắt tại ngõ vào của 8259A và nhận trả lời chấp nhận yêu cầu ngắt hay không $\overline{\text{INTA}}$ từ CPU để đưa kiểu ngắt vào CPU.

Data bus buffer (đệm bus dữ liệu): giao tiếp giữa 8259A với bus dữ liệu của CPU.

Cascade buffer / comparator (đệm nối tầng và so sánh): lưu trữ và so sánh số hiệu của các kiểu ngắt trong trường hợp dùng nhiều mạch 8259A.



Hình 1.16 – Sơ đồ khối của PIC 8259A

❖ **Các tín hiệu điều khiển:**

CAS0 ÷ 2 (In, Out): các ngõ vào chọn mạch 8259A tớ (slave) từ mạch 8259A chủ (master) trong trường hợp dùng nhiều mạch 8259A để tăng yêu cầu ngắt.

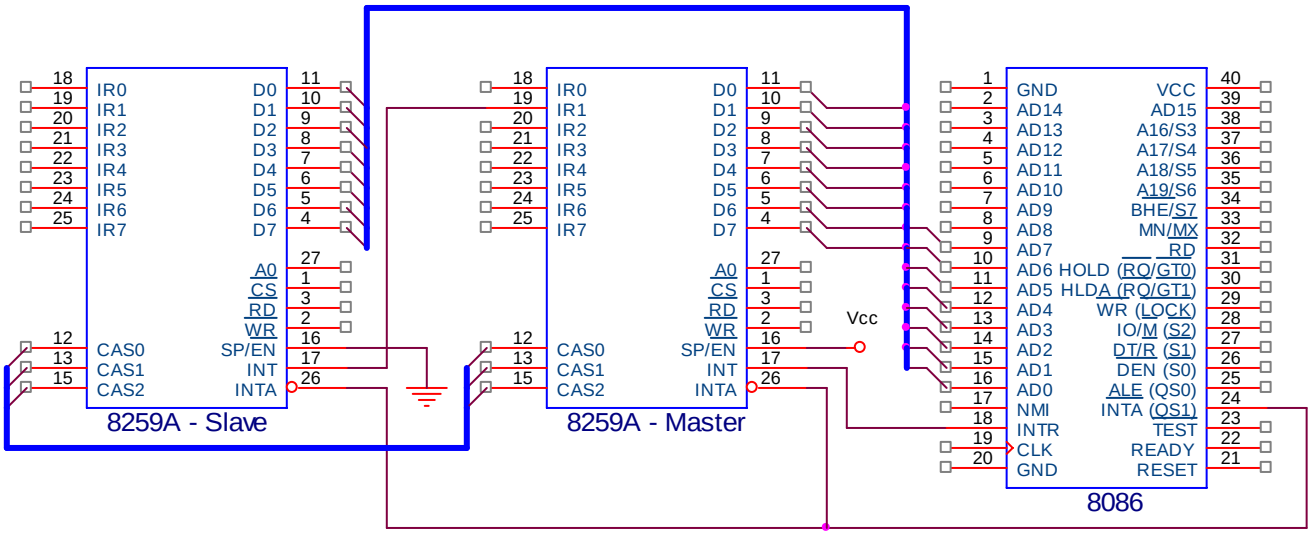
$\overline{SP}$ / $\overline{EN}$ (In, Out) (Slave Program / Enable Buffer): nếu 8259A hoạt động ở chế độ không dùng đệm dữ liệu thì tín hiệu này dùng để xác định mạch 8259A là mạch chủ ($\overline{SP}$ = 1) hay tớ ($\overline{SP}$ = 0). Nếu 8259A hoạt động ở chế độ có đệm dữ liệu thì tín hiệu này dùng để cho phép giao tiếp giữa 8259A và CPU, khi đó mạch 8259A là master hay slave phải dựa vào từ lệnh khởi động ICW4.

INT (Out): tín hiệu yêu cầu ngắt đưa đến CPU (chân INTR).

$\overline{INTA}$ (In): nhận trả lời chấp nhận ngắt hay không từ CPU (chân $\overline{INTA}$)

A0: cho phép chọn các từ điều khiển của 8259A.

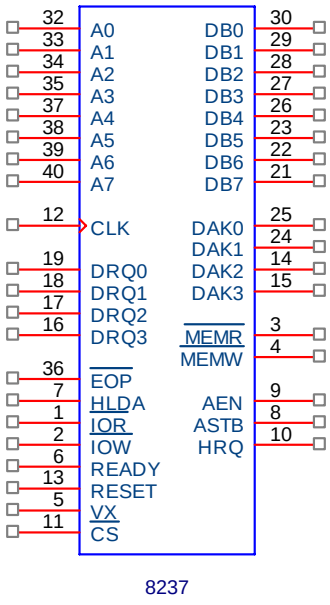
8259A cho phép xử lý 8 ngắt với 8 mức ưu tiên khác nhau. Trong trường hợp hệ thống có số lượng ngắt lớn hơn thì có thể mắc nhiều 8259A liên tầng.

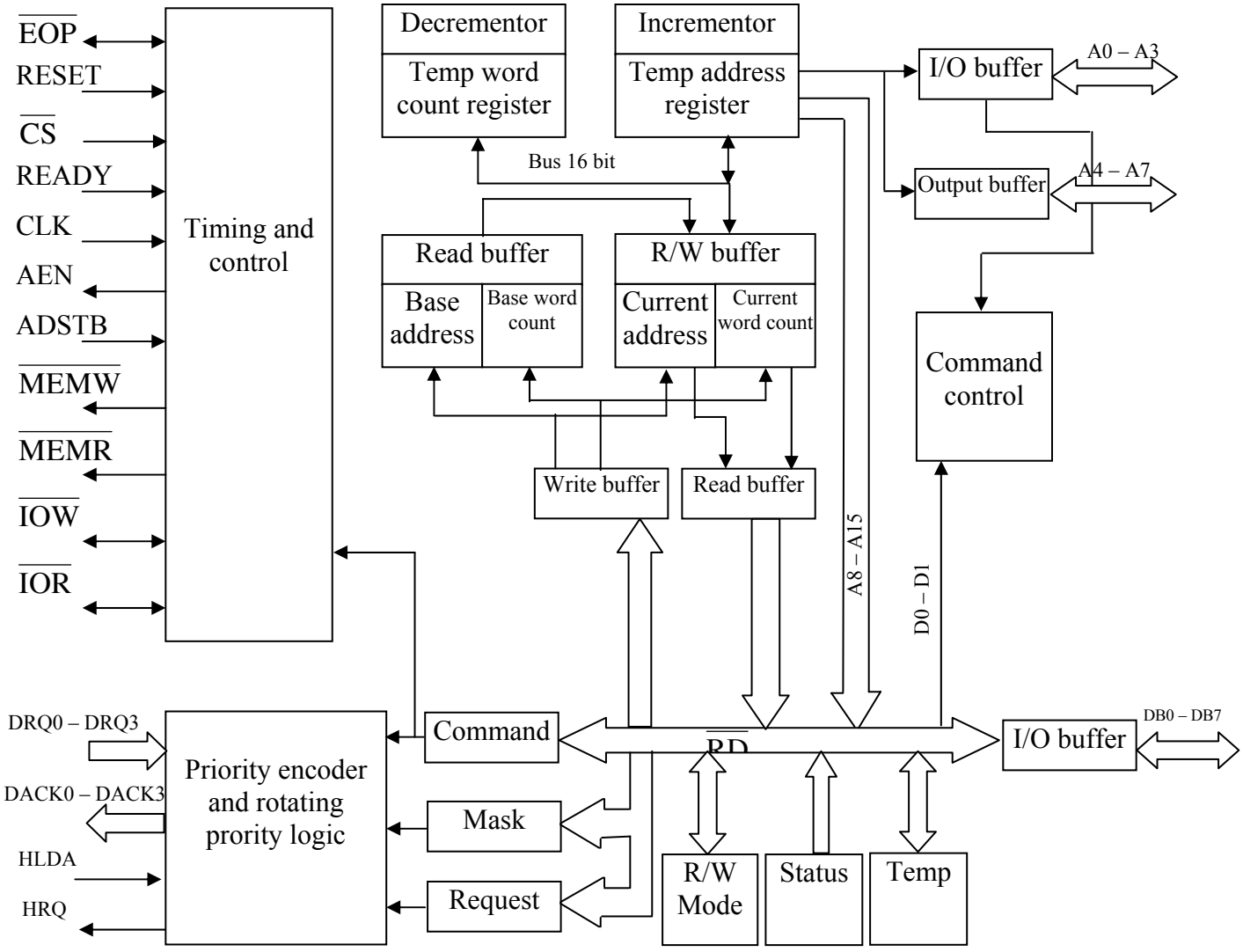


Hình 1.17 – 8259A mắc liên tầng

4.5. Chip điều khiển truy nhập bộ nhớ trực tiếp DMAC 8237 (Direct Memory Access Controller)

DMAC 8237 có thể thực hiện truyền dữ liệu theo 3 kiểu: kiểu đọc (từ bộ nhớ ra thiết bị ngoại vi), kiểu ghi (từ thiết bị ngoại vi đến bộ nhớ) và kiểu kiểm tra.





Hình 1.18 – Sơ đồ chân và sơ đồ khối của DMAC 8237A

❖ **Khối Timing and Control (định thời và điều khiển):**

Tạo các tín hiệu định thời và điều khiển cho bus ngoài (external bus). Các tín hiệu này được đồng bộ với xung clock đưa vào DMAC (tần số xung clock tối đa là 5 MHz).

❖ **Khối Priority encoder and rotating priority logic (mã hóa ưu tiên và quay mức ưu tiên):**

DMAC 8237A có 2 mô hình ưu tiên: mô hình ***ưu tiên cố định*** (***fixed priority***) và mô hình ***ưu tiên quay*** (***rotating priority***). Trong mô hình ưu tiên cố định, kênh 0 sẽ có mức ưu tiên cao nhất còn kênh 3 có mức ưu tiên thấp nhất. Còn đối với mô hình ưu tiên quay thì mức ưu tiên khi khởi động giống như mô hình ưu tiên cố định nhưng khi yêu cầu DMA tại một kênh nào đó được phục vụ thì sẽ được đặt xuống mức ưu tiên thấp nhất.

❖ **Khối Command Control (điều khiển lệnh):**

Giải mã các thanh ghi lệnh (xác định thanh ghi sẽ được truy xuất và loại hoạt động cần thực hiện).

❖ Các thanh ghi:

DMAC 8237A có tất cả 12 loại thanh ghi nội khác nhau:

Tên	Kích thước (bit)	Số lượng
Thanh ghi địa chỉ cơ sở (Base Address Register)	16	4
Thanh ghi đếm từ cơ sở (Base Word Count Register)	16	4
Thanh ghi địa chỉ hiện hành (Current Address Register)	16	4
Thanh ghi đếm từ hiện hành (Current Word Count Register)	16	4
Thanh ghi địa chỉ tạm (Temporary Address Register)	16	1
Thanh ghi đếm từ tạm (Temporary Word Count Register)	16	1
Thanh ghi trạng thái (Status Register)	8	1
Thanh ghi lệnh (Command Register)	8	1
Thanh ghi tạm (Temporary Register)	8	1
Thanh ghi chế độ (Mode Register)	6	4
Thanh ghi mặt nạ (Mask Register)	4	1
Thanh ghi yêu cầu (Request Register)	4	1

❖ Chức năng các chân của 8237A:

$\overline{CLK}$ (Input): tín hiệu xung clock của mạch. Tín hiệu này thường được lấy từ 8284 sau khi qua cổng đảo.

$\overline{CS}$ (Input): thường được nối với bộ giải mã địa chỉ.

RESET (Input): khởi động 8237A, được nối với ngõ RESET của 8284. Khi Reset thì thanh ghi mặt nạ được lập còn các phần sau bị xóa:

+ Thanh ghi lệnh
+ Thanh ghi trạng thái
+ Thanh ghi yêu cầu
+ Thanh ghi tạm
+ Flip-flop đầu/cuối (First/Last flip-flop)

READY (Input): nối với READY của CPU để tạo chu kỳ đợi khi truy xuất các thiết bị ngoại vi hay bộ nhớ chậm.

HLDA (Hold Acknowledge)(Input): tín hiệu chấp nhận yêu cầu treo từ CPU

$DRQ_0 - DRQ_3$ (DMA Request)(Input): các tín hiệu yêu cầu treo từ thiết bị ngoại vi.

DB0 – DB7 (Input, Output): nối đến bus địa chỉ và dữ liệu của CPU

$\overline{IOR}$, $\overline{IOW}$ (Input, Output): sử dụng trong các chu kỳ đọc và ghi

$\overline{EOP}$ (End Of Process)(Input,Output): bắt buộc DMAC kết thúc quá trình DMA nếu là ngõ vào hay dùng để báo cho một kênh biết là dữ liệu đã chuyển xong (Terminal count – TC), thường dùng như yêu cầu ngắt để CPU kết thúc quá trình DMA.

A0 – A3 (Input, Output): chọn các thanh ghi trong 8237A khi lập trình hay dùng để chứa 4 bit địa chỉ thấp.

A4 – A7 (Output): chứa 4 bit địa chỉ

HRQ (Hold Request)(Output): tín hiệu yêu cầu treo đến CPU

$DACK_0 – DACK_3$ (DMA Acknowledge)(Output): tín hiệu trả lời yêu cầu DMA cho các kênh.

AEN (Output): cho phép lấy địa chỉ vùng nhớ cần trao đổi

ADSTB (Address Strobe)(Output): chốt các bit địa chỉ cao A8 – A15 chứa trong các chân DB0 – DB7

$\overline{MEMR}$, $\overline{MEMW}$ (Output): dùng để đọc / ghi bộ nhớ.

❖ **Các thanh ghi nội:**

Các thanh ghi nội trong DMAC 8237A được truy xuất nhờ các bit địa chỉ thấp A0 – A3.

Bit địa chỉ				**Địa chỉ**	**Chọn chức năng**	**R/W?**
A3	**A2**	**A1**	**A0**			
0	0	0	0	X0	Thanh ghi địa chỉ bộ nhớ kênh 0	R/W
0	0	0	1	X1	Thanh ghi đếm từ kênh 0	R/W
0	0	1	0	X2	Thanh ghi địa chỉ bộ nhớ kênh 1	R/W
0	0	1	1	X3	Thanh ghi đếm từ kênh 1	R/W
0	1	0	0	X4	Thanh ghi địa chỉ bộ nhớ kênh 2	R/W
0	1	0	1	X5	Thanh ghi đếm từ kênh 2	R/W
0	1	1	0	X6	Thanh ghi địa chỉ bộ nhớ kênh 3	R/W
0	1	1	1	X7	Thanh ghi đếm từ kênh 3	R/W
1	0	0	0	X8	Thanh ghi trạng thái / lệnh	R/W
1	0	0	1	X9	Thanh ghi yêu cầu	W
1	0	1	0	XA	Thanh ghi mặt nạ cho một kênh	W
1	0	1	1	XB	Thanh ghi chế độ	W
1	1	0	0	XC	Xóa flip-flop đầu/cuối	W
1	1	0	1	XD	Xóa toàn bộ các thanh ghi / đọc thanh ghi tạm	W/R
1	1	1	0	XE	Xóa thanh ghi mặt nạ	W
1	1	1	1	XF	Thanh ghi mặt nạ	W

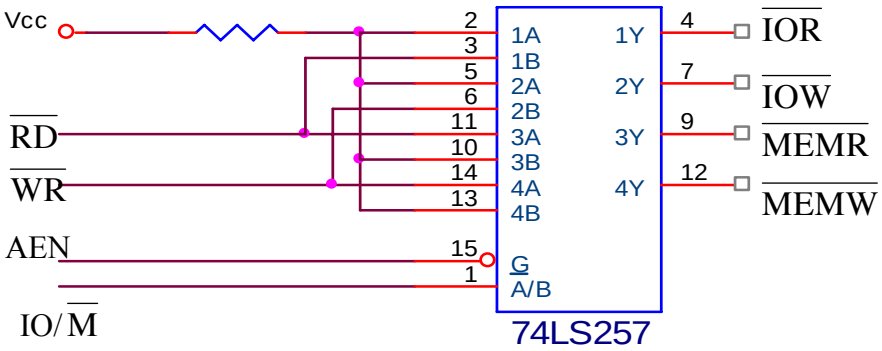
Địa chỉ các thanh ghi nội dùng ghi / đọc địa chỉ:

Kênh	$\overline{IOR}$	$\overline{IOW}$	**A3**	**A2**	**A1**	**A0**	**Thanh ghi**	**R/W?**
0	1	0	0	0	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	0	0	Địa chỉ hiện hành	R
	1	0	0	0	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	0	1	Bộ đếm hiện hành	R
1	1	0	0	0	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	1	0	Địa chỉ hiện hành	R
	1	0	0	0	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	1	1	Bộ đếm hiện hành	R
2	1	0	0	1	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	0	0	Địa chỉ hiện hành	R
	1	0	0	1	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	0	1	Bộ đếm hiện hành	R
3	1	0	0	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	1	0	Địa chỉ hiện hành	R
	1	0	0	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	1	1	Bộ đếm hiện hành	R

Địa chỉ các thanh ghi trạng thái và điều khiển:

$\overline{IOR}$ `	$\overline{IOW}$	**A3**	**A2**	**A1**	**A0**	**Thanh ghi**
1	0	1	0	0	0	Ghi thanh ghi lệnh
0	1	1	0	0	0	Đọc thanh ghi trạng thái
1	0	1	0	0	1	Ghi thanh ghi yêu cầu
1	0	1	0	1	0	Ghi thanh ghi mặt nạ
1	0	1	0	1	1	Ghi thanh ghi chế độ
1	0	1	1	0	0	Xóa flip-flop đầu/cuối
1	0	1	1	0	1	Xóa tất cả các thanh ghi nội
0	1	1	1	0	1	
1	0	1	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành
0	1	1	1	1	0	Địa chỉ hiện hành
1	0	1	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành
0	1	1	1	1	1	Bộ đếm hiện hành

Mạch 8273A-5 chứa 4 kênh trao đổi dữ liệu DMA với mức ưu tiên lập trình được. 8237A-5 có tốc độ truyền 1 MBps cho mỗi kênh và 1 kênh có thể truyền 1 mảng có độ dài 64 KB. Để có thể sử dụng mạch DMAC 8237A, ta cần tạo tín hiệu điều khiển như sau:



Hình 1.19 – Tín hiệu điều khiển cho hệ thống làm việc với DMAC 8237A

Tín hiệu AEN từ 8237A dùng để cấm các tín hiệu điều khiển từ CPU khi DMAC đã nắm quyền điều khiển bus.

4.6. Chip điều khiển màn hình CRTC 6845 (Cathode Ray Tube Controller)

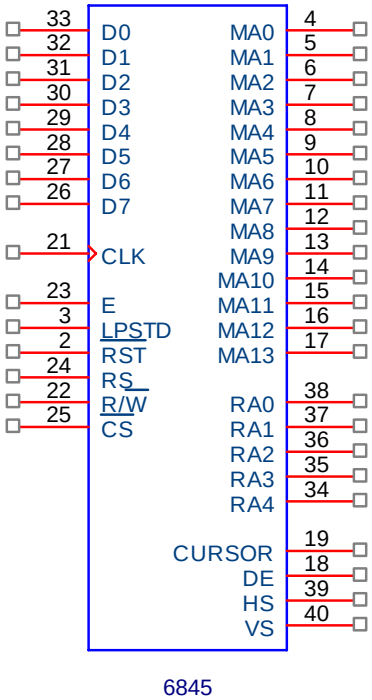
RST (Reset): khởi động lại 6845.

MA0 ÷ MA13 (Memory Address): 14 địa chỉ nhớ cho RAM màn hình.

DE (Display Enable): cho phép (=1) hay không (=0) các tín hiệu điều khiển và địa chỉ vùng hiện lên màn hình.

LPSTD (Light Pen Strobe): lưu trữ địa chỉ hiện hành của RAM màn hình trong thanh ghi bút sáng. CPU đọc thanh ghi và xác định vị trí bút sáng trên màn hình.

CURSOR: vị trí con trỏ đã quét (=1) hay chưa (=0).



Hình 1.20 – Sơ đồ chân của 6845

VS (Vertical Synchronization): ngõ ra tín hiệu đồng bộ quét dọc

HS (Horizontal Synchronization): ngõ ra tín hiệu đồng bộ quét ngang

RA0 ÷RA4 (Row Address): phân định hàng quét của ký tự trong chế độ văn bản (32 hàng quét). Trong chế độ đồ họa, chúng kết hợp với MA0 ÷ MA13 tạo các địa chỉ cho các bank RAM màn hình.

D0 ÷ D7: đường dữ liệu.

$\overline{CS}$: chọn chip.

RS (Regigter Select): chọn thanh ghi địa chỉ (=0) hay thanh ghi dữ liệu (=1).

E: xung âm kích hoạt bus dữ liệu và dùng như xung clock cho 6845 đọc / ghi dữ liệu vào các thanh ghi bên trong.

$R/\overline{W}$: đọc / ghi dữ liệu vào các thanh ghi.

CLK: dùng đồng bộ với tín hiệu của màn hình và thường bằng tốc độ hiện ký tự trên màn hình.

4.7. Chip đồng xử lý toán học 8087/80287/80387 (Mathematical co-processor)

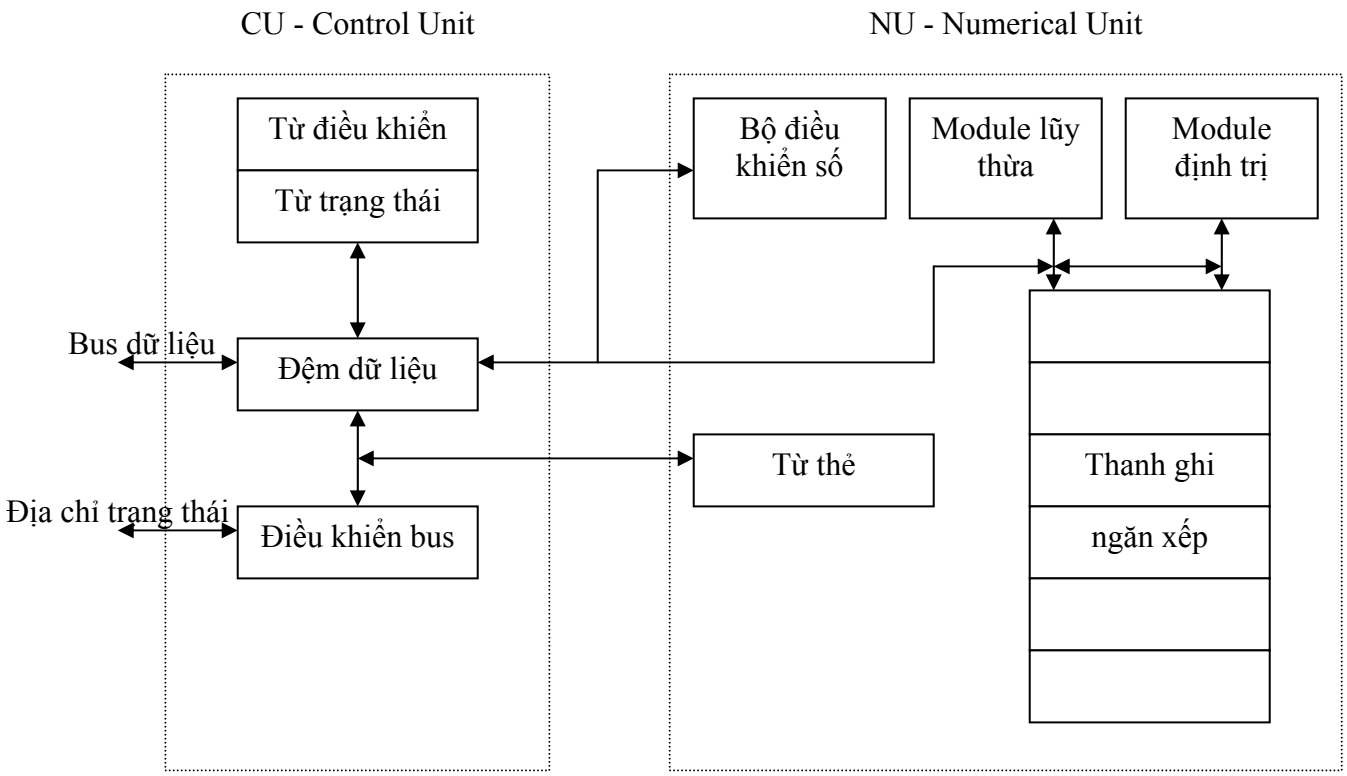
Các bộ đồng xử lý toán 80x87 hỗ trợ CPU trong việc tính toán các biểu thức dùng dấu chấm động như cộng, trừ, nhân, chia các số dấu chấm động, căn thức, logarit, … Chúng cho phép xử lý các phép toán này nhanh hơn nhiều so với CPU. Thời gian xử lý giữa 8087 và 8086 như sau (dùng xung clock 8 MHz):

Phép toán	**8087 [μs]**	**8086 [μs]**
Cộng / trừ	10.6	1000
Nhân	11.9	1000
Chia	24.4	2000
Căn bậc hai	22.5	12250
Tang	56.3	8125
Lũy thừa	62.5	10680
Lưu trữ	13.1	750

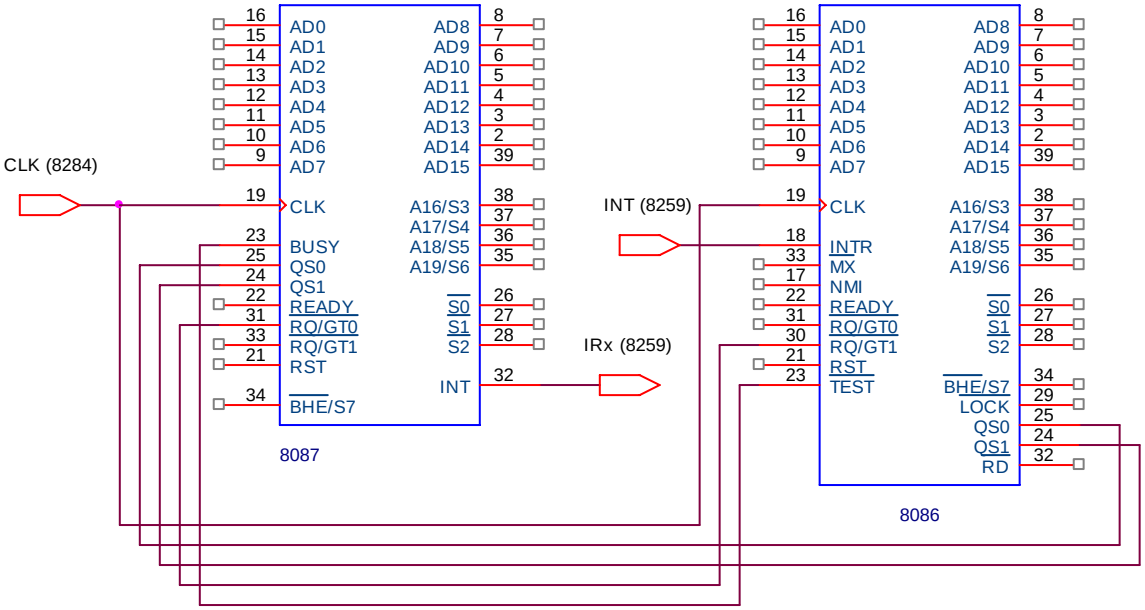
❖ **8087:**

8087 gồm một đơn vị điều khiển (CU – Control Unit) dùng để điều khiển bus và một đơn vị số học (NU – Numerical Unit) để thực hiện các phép toán dấu chấm động trong các mạch tính lũy thừa (exponent module) và mạch tính phần định trị (mantissa module). Khác với 8086, thay vì dùng các thanh ghi rời rạc là một ngăn xếp thanh ghi.

Đơn vị điều khiển nhận và giải mã lệnh, dọc và ghi các toán hạng, chạy các lệnh điều khiển riêng của 8087. Do đó, CU có thể đồng bộ với CPU trong khi NU đang thực hiện các công việc tính toán. CU bao gồm bộ điều khiển bus, bộ đệm dữ liệu và hàng lệnh.



Hình 1.21 – Sơ đồ khối của 8087



Hình 1.22 – Sơ đồ kết nối 8087 và CPU 8086

Ngăn xếp thanh ghi có tất cả 8 thanh ghi từ R0 ÷ R7, mỗi thanh ghi dài 80 bit trong đó bit 79 là bit dấu, bit 64 ÷ 78 dùng cho số mũ và phần còn lại là phần định trị. Dữ liệu truyền giữa các thanh ghi này được thực hiện rất nhanh do 8087 có độ rộng bus dữ liệu là 84 bit và không cần phải biến đổi định dạng.

Ngay sau khi reset PC, bộ đồng xử lý kiểm tra xem nó có được nối với PC hay không bằng các đường $\overline{BHE}$/S7. 8087 sẽ điều chỉnh độ dài của hàng lệnh cho phù hợp với CPU (nếu dùng 8086 thì độ dài là 6 byte).

8087 có một thanh ghi trạng thái là thanh ghi từ thẻ (tag word) gồm các cặp bit Tag0 ÷ Tag7 để lưu trữ các thông tin liên quan đến nội dung của các thanh ghi R0 ÷ R7 để cho phép thực hiện một số tác vụ nhanh hơn. Mỗi thanh ghi từ thẻ có 2 bit xác định 4 giá trị khác nhau của các thanh ghi Ri.

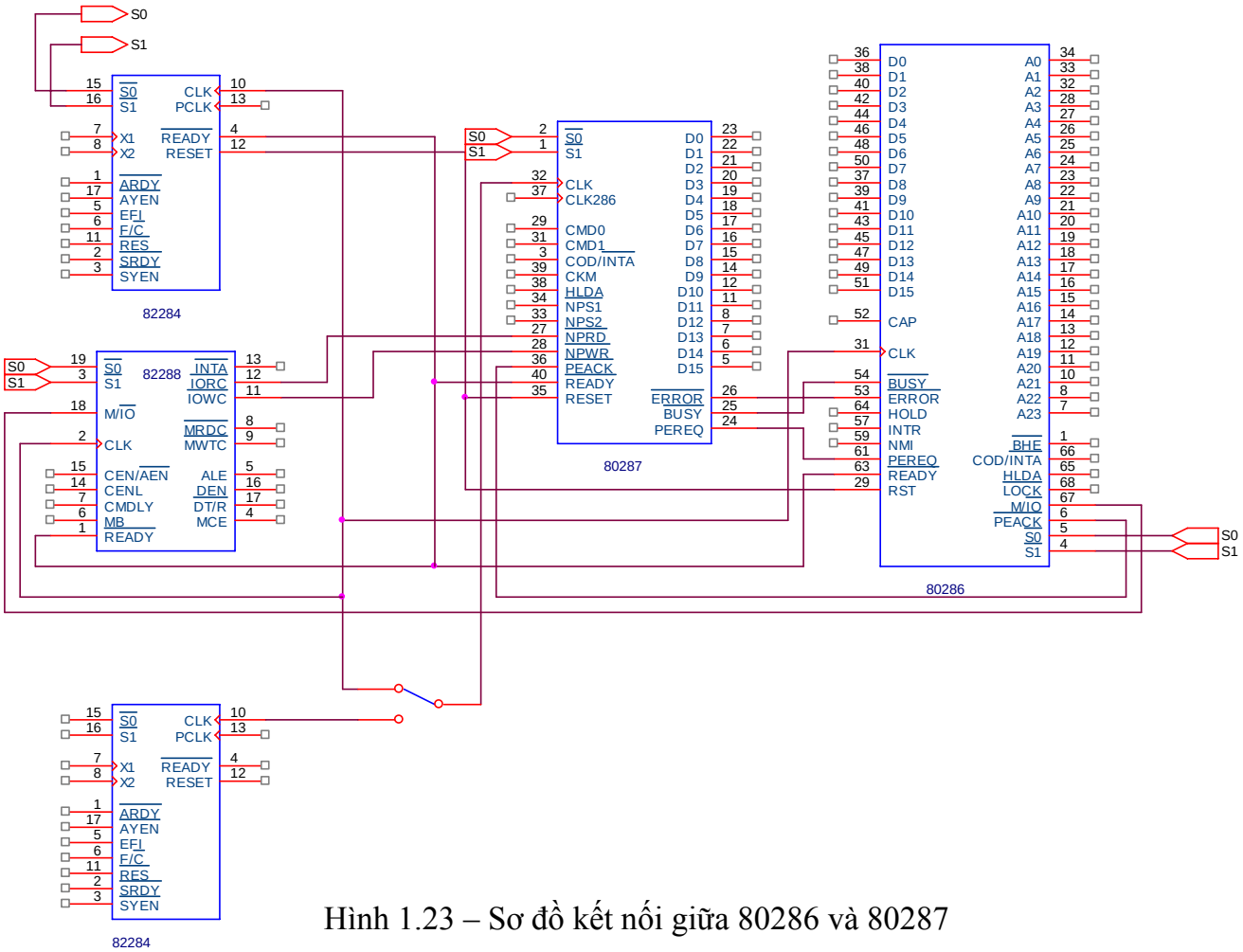
Tag = 00: xác định

Tag = 01: zero

Tag = 10: NAN, giá trị bất thường

Tag = 11: rỗng

❖ **80287:**



Hình 1.23 – Sơ đồ kết nối giữa 80286 và 80287

Do 80286 có chế độ mạch bảo vệ nên mạch ghép nối giữa 80286 và 80287 được thiết kế khác 8087 ở đơn vị điều khiển CU. Bộ đồng xử lý ở đây không thực hiện truy xuất bộ nhớ trực tiếp. Để truy xuất được bộ nhớ, 80287 không những cần một đơc vị định địa chỉ đơn giản của nó mà còn phải được tăng cường thêm chức năng quản lý bộ nhớ của 80286. Cấu trúc bên trong của 80287 cũng tương tự như 8087, chỉ có đơn vị bus thay đổi cho phù hợp với 80286. Khác vơi 8087, 80287 hoạt động không đồng bộ với CPU nên có thể dùng xung clock riêng.

❖ **80387:**

Ưu điểm của 80387 so với 80287 là có thể thực hiện các phép toán số học nhanh hơn. No có bus dữ liệu 32 bit như CPU và sử dụng công nghệ CMOS nên công suất tiêu thụ thấp hơn.

5. Bộ thanh ghi

μP 8086/8088 có tất cả 14 thanh ghi nội. Các thanh ghi này có thể phân loại như sau:

- Thanh ghi dữ liệu (data register)
- Thanh ghi chỉ số và con trỏ (index & pointer register)
- Thanh ghi đoạn (segment register)
- Thanh ghi trạng thái và điều khiển (status & control register)

5.1. Các thanh ghi dữ liệu

Các thanh ghi dữ liệu gồm có các thanh ghi 16 bit AX, BX, CX và DX trong đó nửa cao và nửa thấp của mỗi thanh ghi có thể định địa chỉ một cách độc lập. Các nửa thanh ghi này (8 bit) có tên là AH và AL, BH và BL, CH và CL, DH và DL.

Các thanh ghi này được sử dụng trong các phép toán số học và logic hay trong quá trình chuyển dữ liệu.

Bảng 2.8:

Thanh ghi	Sử dụng trong
AX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word) IN (nhập word) OUT (xuất word) CWD Các phép toán xử lý chuỗi (string)
AL	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) IN (nhập byte) OUT (xuất byte) XLAT AAA, AAD, AAM, AAS (các phép toán ASCII) CBW (đổi sang word)

	DAA, DAS (số thập phân) Các phép toán xử lý chuỗi (string)
AH	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) CBW (đổi sang word)
BX	XLAT
CX	LOOP, LOOPE, LOOPNE Các phép toán string với tiếp dầu ngữ REP
CL	RCR, RCL, ROR, ROL (quay với số đếm byte) SHR, SAR, SAL (dịch với số đếm byte)
DX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word)

AX (ACC – Accumulator): thanh ghi tích luỹ

BX (Base): thanh ghi cơ sở

CX (Count): đếm

DX (Data): thanh ghi dữ liệu

5.2. Các thanh ghi chỉ số và con trỏ

Bao gồm các thanh ghi 16 bit SP, BP, SI và DI, thường chứa các giá trị offset (độ lệch) cho các phần tử định địa chỉ trong một phân đoạn (segment). Chúng có thể được sử dụng trong các phép toán số học và logic. Hai thanh ghi con trỏ (SP – Stack Pointer và BP – Base Pointer) cho phép truy xuất dễ dàng đến các phần tử đang ở trong ngăn xếp (stack) hiện hành. Các thanh ghi chỉ số (SI – Source Index và DI – Destination Index) được dùng để truy xuất các phần tử trong các đoạn dữ liệu và doạn thêm (extra segment). Thông thường, các thanh ghi con trỏ liên hệ đến đoạn stack hiện hành và các thanh ghi chỉ số liên hệ đến doạn dữ liệu hiện hành. SI và DI dùng trong các phép toán chuỗi.

5.3. Các thanh ghi đoạn

Bao gồm các thanh ghi 16 bit CS (Code segment), DS (Data segment), SS (stack segment) và ES (extra segment), dùng để định địa chỉ vùng nhớ 1 MB bằng cách chia thành 16 đoạn 64 KB.

Tất cả các lệnh phải ở trong đoạn mã hiện hành, được định địa chỉ thông qua thanh ghi CS. Offset (độ lệch) của mã được xác định bằng thanh ghi IP. Dữ liệu chương trình thường được đặt ở đoạn dữ liệu, định vị thông qua thanh ghi DS. Stack định vị thông qua thanh ghi SS. Thanh ghi đoạn thêm có thể sử dụng để định địa chỉ các toán hạng, dữ liệu, bộ nhớ và các phần tử khác ngoài đoạn dữ liệu và stack hiện hành.

5.4. Các thanh ghi điều khiển và trạng thái

Thanh ghi con trỏ lệnh IP (Instruction Pointer) giống như bộ đếm chương trình (Program Counter). Thanh ghi điều khiển này do BIU quản lý nhằm lưu trữ offset từ bắt đầu đoạn mã đến lệnh thực thi kế tiếp. Ta không thể xử lý trực tiếp trên thanh ghi IP.

Thanh ghi cờ (Flag register) hay từ trạng thái 16 bit chứa 3 bit điều khiển (TF, IF và DF) và 6 bit trạng thái (OF, SF, ZF, AF, PF và CF) còn các bit còn lại mà 8086/8088 không sử dụng thì không thể truy xuất được.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- OF (Overflow - tràn): OF = 1 xác định tràn số học, xảy ra khi kết quả vượt ra ngoài phạm vi biểu diễn
- DF (Direction- hướng): xác định hướng chuyển string, DF = 1 khi μP làm việc với string theo thứ tự từ phải sang trái.
- IF (Interrupt - ngắt): cho phép hay cấm các interrupt có mặt nạ
- TF (Trap - bẫy): đặt μP vào chế độ từng bước, dùng cho các chương trình gỡ rối (debugger).
- SF (Sign - dấu): dùng để chỉ các kết quả số học là số dương (SF = 0) hay âm (SF = 1).
- ZF (Zero): = 1 nếu kết quả của phép toán trước là 0.
- AF (Auxiliary – nhớ phụ): dùng trong các số thập phân để chỉ nhớ từ nửa byte thấp hay mượn từ nửa byte cao.
- PF (Parity): PF = 1 nếu kết quả của phép toán là có tổng số bit 1 là chẵn (dùng để kiểm tra lỗi truyền dữ liệu)
- CF (Carry): CF = 1 nếu có nhớ hay mượn từ bit cao nhất của kết quả. Cờ này cũng dùng cho các lệnh quay.

Chương 2
NGẮT VÀ SỰ KIỆN

1. Khái niệm

Ngắt (interrupt) là quá trình dừng chương trình chính đang chạy để ưu tiên thực hiện một chương trình khác, chương trình này được gọi là chương trình phục vụ ngắt (ISR – Interrupt Service Routine). ISR hoàn toàn giống với một chương trình bình thường trên máy tính, nghĩa là nó có khả năng truy xuất đến tất cả các lệnh ngôn ngữ máy của µP. Tuy nhiên cuối ISR sẽ kết thúc bằng lệnh IRET (Interrupt Return) để µP tiếp tục thực hiện lệnh đã kết thúc trước đây.

Các nguyên nhân dẫn đến ngắt là:

- Bản thân chương trình đang thực hiện bị lỗi, ví dụ như: chia cho 0, …
- Do tác động của thiết bị ngoại vi, ví dụ như: thực hiện lệnh in nhưng máy in lỗi, ghi dữ liệu vào đĩa nhưng không có đĩa, …
- Do lập trình viên chủ động gọi các ngắt có sẵn.

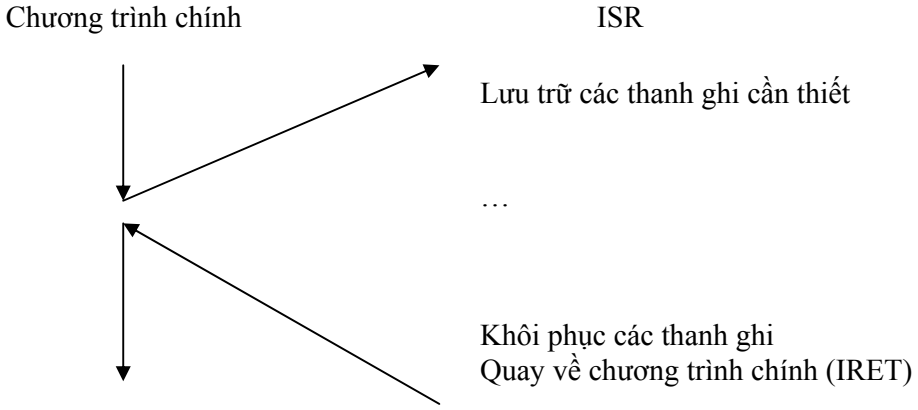
Một cách đơn giản, chúng ta có thể xem ngắt như là quá trình gọi chương trình con nhưng các chương trình con này được tạo ra sẵn trong máy tính và quá trình gọi này có thể xảy ra tại thời điểm không xác định trước.

Sự kiện (Event) là một tác động lên một đối tượng trong môi trường Windows. Khi có một sự kiện xảy ra, Windows sẽ gởi thông điệp (message) đến đối tượng. Các sự kiện thường xảy ra là:

- Sự kiện chuột: Click, Double Click, …
- Sự kiện bàn phím: nhấn phím, nhả phím, …
- Sự kiện cửa sổ: Activate, Load, Unload, …

2. Các loại ngắt và bảng vector ngắt

Quá trình ngắt có thể mô tả như sau:

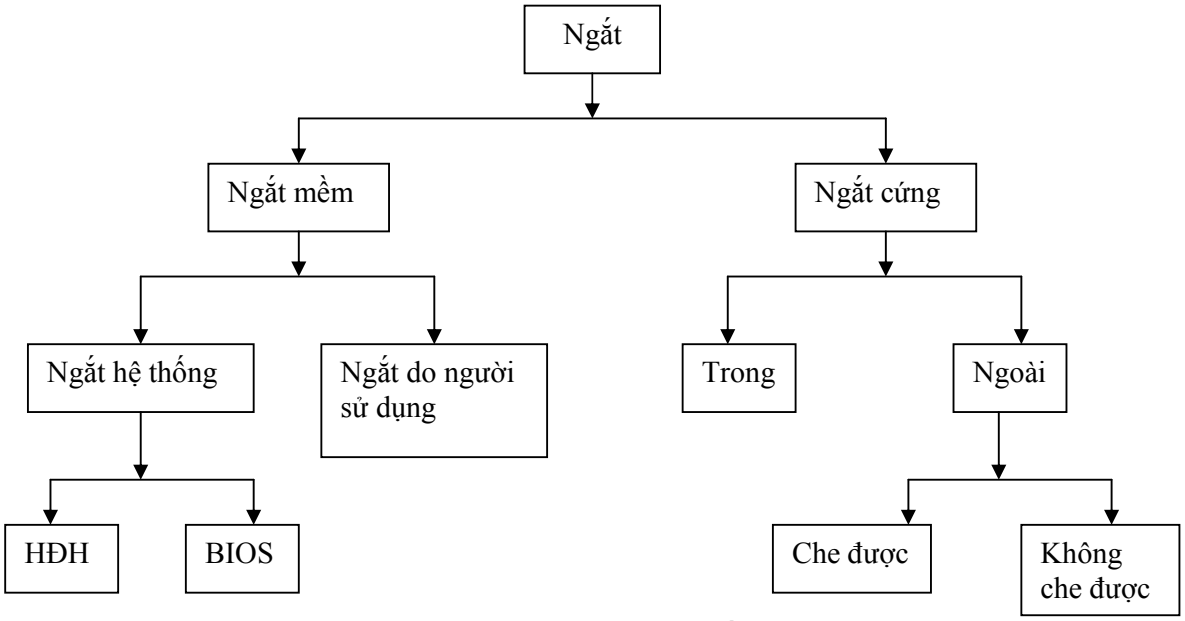


Hình 2.1 – Quá trình thực hiện ngắt

Trong các quá trình ngắt, ta phân biệt thành 2 loại: ngắt cứng và ngắt mềm. Ngắt mềm là ngắt được gọi bằng một lệnh trong chương trình ngôn ngữ máy. Ngắt mềm được thục hiện trên hợp ngữ thông qua lệnh INT. Đối với các ngôn ngữ bậc cao hơn, vẫn cho phép thực hiện gọi ngắt nhưng phải được biên dịch thành lệnh INT trong hợp ngữ rồi mời thực hiện.

Khác với ngắt mềm, ngắt cứng không được khởi động bên trong máy tính mà do các linh kiện điện tử tác đông lên hệ thống. Ngắt cứng cũng được chia thành 2 loại: ngắt che được và ngắt không che được. Ngắt che được là ngắt có thể cho phép hay không cho phép thực thi bằng phần mềm thông qua cờ ngắt IF (Interrupt Flag): lệnh CLI (Clear Interrupt Flag) sẽ cấm ngắt và lệnh STI (Set Interrupt Flag) sẽ cho phép các ngắt này hoạt động.

Các loại ngắt khác nhau có thể mô tả như sau:



Hình 2.2 – Các loại ngắt

Khi thực hiện lệnh gọi một ngắt nào đó, chương trình con phục vụ cho ngắt sẽ được gọi. Để thực hiện các ngắt tương ứng, địa chỉ thực hiện các chương trình con phục vụ ngắt được đặt trong một bảng, gọi là bảng vector ngắt.

Bảng vector ngắt gồm có 256 phần tử, mỗi phần tử gồm 4 byte ứng với 256 ngắt (từ ngắt 0 đến ngắt 0FFh). Mỗi phần tử trong bảng vector ngắt chứa 2 địa chỉ: địa chỉ thanh ghi đoạn đưa vào CS và địa chỉ offset của chương trình phục vụ ngắt đưa vào IP.

Bảng vector ngắt có thể mô tả như sau:

Địa chỉ	Địa chỉ ISR	Số thứ tự ngắt
0000h:0000h	CS	0
0000h:0002h	IP	
0000h:0004h	CS	1
0000h:0006h	IP	
0000h:0008h	CS	2
0000h:000Ah	IP	
…	…	…
0000h:03FCh	CS	255
0000h:03FEh	IP	

Khi có một quá trình ngắt xảy ra, CPU sẽ tìm địa chỉ bắt đầu của chương trình ngắt được chứa trong bảng vector ngắt theo số thự tự ngắt. Do một phần tử trong bảng vector ngắt chiếm 4 byte nên để tìm giá trị địa chỉ trong bảng vector ngắt, ta chỉ cần nhân số thứ tự ngắt với 4.

Danh sách các ngắt mô tả như sau:

STT	Địa chỉ	Chức năng
00h	0000h – 0003h	CPU: chia cho 0
01h	0004h – 0007h	CPU: thực hiện từng lệnh
02h	0008h – 000Bh	CPU: Lỗi RAM
03h	000Ch – 000Fh	CPU: thực hiện đến điểm dừng
04h	0010h – 0013h	CPU: tràn số
05h	0014h – 0017h	In trang màn hình (Print Screen)
06h, 07h	0018h – 001Fh	Dành riêng
08h	0020h – 0023h	IRQ0: ngắt đồng hồ (18.2 lần / giây)
09h	0024h – 0027h	IRQ1: ngắt bàn phím
0Ah	0028h – 002Bh	IRQ2: Dành riêng
0Bh	002Ch – 002Fh	IRQ3: Giao tiếp nối tiếp 1
0Ch	0030h – 0033h	IRQ4: Giao tiếp nối tiếp 2
0Dh	0034h – 0037h	IRQ5: Đĩa cứng

0Eh	0038h – 003Bh	IRQ6: Đĩa mềm
0Fh	003Ch – 003Fh	IRQ7: Máy in
10h	0040h – 0043h	BIOS: màn hình
11h	0044h – 0047h	BIOS: xác định cấu hình máy tính
12h	0048h – 004Bh	BIOS: xác định kích thước RAM
13h	004Ch – 004Fh	BIOS: truy nhập đĩa cứng / đĩa mềm
14h	0050h – 0053h	BIOS: truy nhập giao tiếp nối tiếp
15h	0054h – 0057h	BIOS: truy nhập cassette hay mở rộng ngắt
16h	0058h – 005Bh	BIOS: kiểm tra bàn phím
17h	005Ch – 005Fh	BIOS: truy nhập máy in
18h	0060h – 0063h	Chương trình xâm nhập ROM BASIC
19h	0064h – 0067h	BIOS: khởi động hệ thống (khi nhấn Ctrl-Alt-Del)
1Ah	0068h – 006Bh	BIOS: đọc / ghi ngày / giờ
1Bh	006Ch – 006Fh	Nhấn phím Break
1Ch	0070h – 0073h	Gọi sau INT 08h
1Dh	0074h – 0077h	Địa chỉ bảng tham số màn hình
1Eh	0078h – 007Bh	Địa chỉ bảng tham số đĩa mềm
1Fh	007Ch – 007Fh	Địa chỉ bảng font có ký tự mở rộng
20h	0080h – 0083h	DOS: kết thúc chương trình
21h	0084h – 0087h	DOS: gọi các hàm của DOS
22h	0088h – 008Bh	Địa chỉ kết thúc chương trình
23h	008Ch – 008Fh	Nhấn Ctrl-Break
24h	0090h – 0093h	Địa chỉ chương trình xử lý lỗi
25h	0094h – 0097h	DOS: đọc đĩa mềm / đĩa cứng
26h	0098h – 009Bh	DOS: ghi đĩa mềm / đĩa cứng
27h	009Ch – 009Fh	DOS: kết thúc chương trình và thường trú
28h – 3Fh	00A0h – 00FFh	Dành riêng cho DOS
40h	0100h – 0103h	BIOS: phục vụ đĩa mềm
41h	0104h – 0107h	Địa chỉ bảng tham số đĩa cứng 1
42h – 45h	0108h – 0117h	Dành riêng
46h	0118h – 011Bh	Địa chỉ của bảng tham số đĩa cứng 2
47h – 49h	011Ch – 0127h	Dành cho user

4Ah	0128h – 012Bh	Hẹn giờ
4Bh – 67h	012Ch – 019Fh	Dành cho user
68h – 6Fh	01A0h – 01BFh	Không dùng
70h	01C0h – 01C3h	IRQ8: đồng hồ thời gian thực
71h	01C4h – 01C7h	IRQ9
72h	01C8h – 01CBh	IRQ10
73h	01CCh – 01CFh	IRQ11
74h	01D0h – 01D3h	IRQ12
75h	01D4h – 01D7h	IRQ13: từ 80x87
76h	01D8h – 01DBh	IRQ14: đĩa cứng
77h	01DCh – 01DFh	IRQ15
78h – 7Fh	01E0h – 01FFh	Dành riêng
80h – F0h	0200h – 03C3h	Dùng cho bộ thông dịch BASIC
F1h – FFh	03C4h – 03FFh	Không sử dụng

3. Gọi ngắt và chặn ngắt

Quá trình gọi ngắt từ hợp ngữ đơn giản là thực hiện lệnh ***INT STT_ngắt*** sau khi nạp các tham số cần thiết cho ngắt. Khi thực hiện lệnh gọi ngắt, CPU sẽ tìm kiếm trong bảng vector ngắt địa chỉ của chương trình phục vụ ngắt. Người sử dụng cũng có thể xây dựng một chương trình cơ sở như các chương trình xử lý ngắt. Sau đó, các chương trình khác có thể gọi ngắt ra để sử dụng. Một chương trình có thể gọi chương trình con loại này mà không cần biết địa chỉ của nó.

Như vậy, nếu muốn thay đổi ngắt, ta có thể thay đổi nội dung trong bảng vector ngắt để chỉ đến chương trình phục vụ do người sử dụng tự xây dựng và chương trình này sẽ được thực hiện khi ngắt được gọi. Để làm điều này, ta chỉ cần tìm vị trí của vector ngắt tương ứng trong bảng và thay đổi giá trị của nó. Điều này thực hiện được do bảng vector ngắt đặt trong RAM và được nạp mỗi khi khởi động hệ thống.

Quá trình lấy và gán địa chỉ của chương trình con phục vụ ngắt có thể thực hiện thông qua ngắt 21h bằng các hàm sau:

Hàm 35h: lấy địa chỉ của ngắt

Vào: AL = số thứ tự ngắt

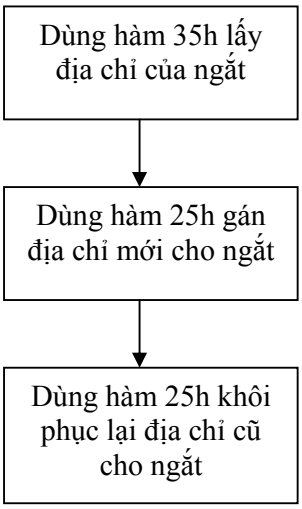
Ra: ES:BX = địa chỉ của chương trình phục vụ ngắt

Hàm 25h: gán địa chỉ của một chương trình phục vụ cho một ngắt

Vào: AL = số thứ tự ngắt, DS:DX

Ra: không có

Để thực hiện chặn một ngắt, ta cần thực hiện như sau:



Hình 2.2 – Quá trình chặn một ngắt

Một ví dụ cho quá trình chặn ngắt như sau:

```
;*****************************************************************************
;*
;*              LAB6-2.ASM - Assembler Laboratory ZMiTAC
*
;*
;*  Sample program that converts lowercase to uppercase when key pressed
*
;*
;*****************************************************************************
.MODEL SMALL
.STACK 100h
.CODE
;*****************************************************************************
;*      Variables
;*****************************************************************************
old_proc        dd 0                    ; address of original interrupt handler
dot_flag        db 0                    ; dot flag

;*****************************************************************************
;*      09h interrupt handler
;*****************************************************************************
segment_kb      EQU 40h                 ; beggining of keyboard data segment
wsk_kb          EQU 1Ch                 ; offset of pointer to keyboard buffer
kb_buf_begin    EQU 80h                 ; offset of address of begining of the
buffer
kb_buf_end      EQU 82h                 ; offset of address of end of the buffer

keys    PROC FAR

;-----------------------------------------------------------------------------
;       Calling of original interrupt handler
;-----------------------------------------------------------------------------
```

```
        int 60h
;-------------------------------------------------------------------------------
;       Prepare registers
;-------------------------------------------------------------------------------
        push ax                         ; push registers on the stack
        push bx
        push dx
        push ds
        mov ax,segment_kb               ; address of keyboard data segment to DS
        mov ds,ax

;-------------------------------------------------------------------------------
;       Read the character and check ASCII code
;-------------------------------------------------------------------------------
        mov bx,ds:[wsk_kb]              ; actual pointer to BX
        mov ax,ds:[kb_buf_begin]        ; buffer beggining to AX
        cmp bx,ax                       ; is the beggining of the buffer ?
        jne mid_buf
        mov bx,ds:[kb_buf_end]          ; last character is at the end of the
buffer
mid_buf:
        mov ax,ds:[bx-2]                ; read last character

        cmp al,'.'                      ; compare with dot
        je dot_found                    ; if dot
        cmp al,'Z'                      ; compare with 'Z'
        ja check_lowercase              ; if above check lowercase
        cmp al,'A'                      ; compare with  'A'
        jb keys_end                     ; end if less

        mov dot_flag,0                  ; uppercase - clear flag
        jmp keys_end                    ; return

check_lowercase:
        cmp al,'z'                      ; compare with 'z'
        ja keys_end                     ; end if above
        cmp al,'a'                      ; compare with 'a'
        jb keys_end                     ; end if less
        cmp dot_flag,0                  ; was dot pressed?
        je keys_end                             ; end if not

;-------------------------------------------------------------------------------
;       Change lowercase to uppercase
;-------------------------------------------------------------------------------
        sub al,'a'-'A'                  ; sub difference between cases
        mov ds:[bx-2],ax
        mov dot_flag,0                  ; uppercase - clear flag
        jmp keys_end                            ; return

dot_found:
        mov dot_flag,1                  ; set flag
        jmp keys_end                    ; return

;-------------------------------------------------------------------------------
;       Pop registers and return from interrupt
;-------------------------------------------------------------------------------
keys_end:
        pop ds
        pop dx
        pop bx
        pop ax
```

```
        iret
keys    ENDP

;*************************************************************************
;*      Main program
;*************************************************************************
;-------------------------------------------------------------------------
;       Get interrupt
;-------------------------------------------------------------------------

start   proc
        mov ah,35h                      ; function 35h - read handler address
        mov al,09h                      ; of interrupt 09h
        int 21h
        mov word ptr old_proc,bx        ; store 32-bit address
        mov word ptr old_proc+2,es      ; of original interrupt handler
        push cs
        pop ds                          ; handler code segment to DS
        mov dx,offset keys              ; offset of handler address to DX
        mov ah,25h                      ; function 25h - set new handler
        mov al,09h                      ; of interrupt 09h
        int 21h
        mov dx,word ptr old_proc+2
        mov ds,dx
        mov dx,word ptr old_proc
        mov ah,25h                      ; function 25h - set new adress
        mov al,60h                      ; of original interrupt handler
        int 21h                         ; 60h instead of 09h

;-------------------------------------------------------------------------
;       Main loop
;-------------------------------------------------------------------------
looping:mov ah,08h                      ; function 08h - read character
        int 21h                         ; ASCII code is returned in AL
        cmp al,1Bh                      ; ESC
        je ending                       ; if ESC end of the loop
        mov dl,al                       ; not ESC - move char to DL
        mov ah,02h                      ; function 02h - display character
        int 21h                         ; ASCII code of char in DL
        jmp looping

;-------------------------------------------------------------------------
ending: mov dx,word ptr old_proc+2
        mov ds,dx
        mov dx,word ptr old_proc
        mov ah,25h                      ; function 25h - set old handler
        mov al,09h                      ; of interrupt 09h
        int 21h

        mov ah,4Ch                      ; end of the program
        int 21h
start   endp
END     start
```

4. Tạo và bắt các sự kiện

Trong hệ điều hành Windows, hook (câu móc) là cơ chế cho phép một hàm chặn một sự kiện (thông điệp, chuột, phím nhấn) trước khi đưa đến đối tượng cần xử lý. Hàm này cho phép thay đổi hoặc thậm chí cấm sự kiện xảy ra. Chúng được gọi là hàm lọc (filter) và được

phân loại dựa theo loại sự kiện bị chặn. Để gọi được hàm lọc, ta cần phải thực hiện quá trình gắn (attach) vào quá trình câu móc (như câu móc bàn phím). Việc gắn một hay nhiều hàm lọc vào một quá trình câu móc được gọi là thiết lập (setting) một quá trình câu móc.

Nếu một quá trình câu móc có nhiều hơn một hàm lọc, Windows sẽ duy trì một chuỗi các hàm lọc trong đó hàm được cài đặt vào gần nhất sẽ nằm ở đầu chuỗi và hàm cài đặt lâu nhất sẽ nằm ở cuối chuỗi. Nếu sự kiện xảy ra làm khởi động quá trình câu móc, Windows sẽ gọi hàm lọc đầu tiên trong chuỗi. Quá trình câu móc vào một sự kiện được sử dụng bằng hàm **SetWindowsHookEx** và hàm **UnhookWindowsHookEx** dùng để xóa bỏ hàm lọc khỏi quá trình.

Cơ chế câu móc cung cấp các khả năng mạnh mẽ cho một ứng dụng Windows. Các ứng dụng này có thể dùng quá trình câu móc để:

- Xử lý và thay đổi các thông điệp gởi đến các dialog box, message box, scroll bar và menu của một ứng dụng (WH_MSGFILTER).
- Xử lý và thay đổi các thông điệp gởi đến các dialog box, message box, scroll bar và menu của hệ thống (WH_SYSMSGFILTER).
- Xử lý và thay đổi các thông điệp của hệ thống bất cứ khi nào hàm **GetMessage** hay **PeekMessage** được gọi (WH_GETMESSAGE).
- Xử lý và thay đổi các thông điệp của hệ thống bất cứ khi nào hàm **SendMessage** được gọi (WH_CALLWNDPROC).
- Ghi hay thực hiện lại các sự kiện bàn phím và chuột (WH_JOURNALRECORD, WH_JOURNALPLAYBACK).
- Xử lý, sửa đổi hay cấm sự kiện chuột (WH_MOUSE).
- Xử lý, sửa đổi hay cấm sự kiện bàn phím (WH_KEYBOARD).
- Đáp ứng với các hoạt động nào đó của hệ thống, có khả năng phát triển CBT (computer-based training) cho ứng dụng (WH_CBT).
- Cấm các hàm lọc khác (WH_DEBUG).

Các ứng dụng thường dùng quá trình câu móc để:

- Dùng phím F1 để hỗ trợ cho menu, dialog box và message box (WH_MSGFILTER).
- Lưu lại quá trình thực hiện khi nhấn phím hay chuột (thường dùng cho macro). Ví dụ như Windows Recorder sử dụng hook để hỗ trợ chức năng record và playback (WH_JOURNALRECORD, WH_JOURNALPLAYBACK).
- Quản lý thông điệp để xác nhận thông điệp được gởi tới cửa sổ hay được tạo ra (WH_GETMESAGE, WH_CALLWNDPROC).
- Mô phỏng ngõ vào chuột và bàn phím (WH_JOURNALPLAYBACK). Quá trình câu móc là phương pháp tin cậy để thực hiện hoạt động này. Nếu ta thực hiện mô phỏng bằng cách gởi thông điệp, Windows sẽ không thực hiện cập nhật trạng thái của bàn phím hay chuột dẫn đến các hoạt động không mong muốn. Nếu quá trình câu móc thực hiện điều này, nó sẽ được xử lý giống như sự kiện chuột hay bàn

phím xảy ra. Ví dụ như Microsoft Excel dùng hook để thực hiện macro SENDKEYS.

- Cung cấp khả năng CBT cho ứng dụng thực hiện trên môi trường Windows (WH_CBT) làm cho quá trình phát triển ứng dụng CBT dễ dàng hơn.

Phạm vi sử dụng: Một trong những đặc trưng của Win32 Hook là cho phép chỉ định quá trình câu móc là hệ thống hay ở dạng luồng (thread). Hook hệ thống cho phép tác động đến các cửa sổ khác trong hệ thống còn hook luồng chỉ cho phép tác động đến cửa sổ hiện hành.

Cách thức sử dụng quá trình câu móc:

Để sử dụng quá trình câu móc, ta cần phải biết:

- Làm thế nào dùng hàm câu móc của Windows để thêm vào hay xóa bỏ một hàm lọc trong chuỗi hàm xử lý của một quá trình câu móc.
- Cần phải thực hiện các hoạt động gì để cài đặt một hàm lọc.
- Có thể thực hiện được hàm câu móc nào và chúng có thể làm được gì, gởi những thông tin nào.

4.1. Hàm câu móc của Windows

Các ứng dụng trên nền Windows sử dụng các hàm **SetWindowsHookEx**, **UnhookWindowsHookEx** và **CallNextHookEx** để quản lý chuỗi hàm lọc trong một quá trình câu móc. Trước phiên bản 3.1, Windows thực hiện quản lý bằng các hàm **SetWindowsHook**, **UnhookWindowsHook** và **DefHookProc**. Mặc dù các hàm này cũng có khả năng thực hiện được trên nền Win32 nhưng sẽ có một số đặc trưng không sử dụng được như các phiên bản mới (**Ex**).

4.1.1. Hàm SetWindowsHookEx

Dùng để thêm một hàm lọc vào một quá trình câu móc.

Khai báo:

```
Public Declare Function SetWindowsHook Lib "user32" Alias "SetWindowsHookA" (ByVal nFilterType As Long, ByVal pfnFilterProc As Long) As Long
Public Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" (ByVal idHook As Long, ByVal lpfn As Long, ByVal hmod As Long, ByVal dwThreadId As Long) As Long
```

Hàm SetWindowsHookEx gồm có 4 tham số:

- **idHook**: xác định loại hàm câu móc sẽ cài đặt. Thông số này gồm các giá trị sau:

WH_KEYBOARD: cài đặt hàm câu móc quản lý thông điệp gởi đi khi nhấn phím (ngoại trừ tổ hợp Ctrl – Alt – Del).

WH_MOUSE: cài đặt hàm câu móc quản lý thông điệp khi điều khiển chuột.

WH_CALLWNDPROC: cài đặt hàm câu móc quản lý thông điệp trước khi hệ thống gởi đến cửa sổ, chỉ cho phép xử lý thông điệp mà không được thay đổi thông điệp.

WH_CALLWNDPROCRET: cài đặt hàm câu móc quản lý thông điệp sau khi cửa sổ đã xử lý. Loại này cho phép thay đổi giá trị trả về của thông điệp.

WH_MSGFILTER: cài đặt hàm câu móc quản lý các thông điệp được tạo ra giống như có một sự kiện của dialog box, message box, menu hay scroll bar.

WH_GETMESSAGE: cài đặt hàm câu móc quản lý các thông điệp được gởi tới hàng đợi.

WH_CBT: cài đặt hàm câu móc để nhận thông báo từ ứng dụng CBT.

WH_DEBUG: cài đặt hàm câu móc để gỡ rối một hàm câu móc khác.

WH_FOREGROUNDIDLE: cài đặt hàm câu móc trong đó hàm này được gọi khi luồng (thread) foreground của ứng dụng rảnh (idle). Quá trình này thường sử dụng để thực thi các tác vụ có độ ưu tiên thấp khi luồng ưu tiên rảnh.

WH_JOURNALPLAYBACK: cài đặt hàm câu móc để gởi các thông điệp đã được lưu bằng hàm câu móc WH_JOURNALRECORD.

WH_JOURNALRECORD: cài đặt hàm câu móc lưu lại các thông điệp đã gởi đến hàng đợi.

WH_KEYBOARD_LL: cài đặt hàm câu móc quản lý sự kiện bàn phím ở mức thấp (dùng cho Windows NT/2000/XP).

WH_MOUSE_LL: cài đặt hàm câu móc quản lý sự kiện chuột ở mức thấp (dùng cho Windows NT/2000/XP).

WH_SHELL: cài đặt hàm câu móc cho một ứng dụng shell.

WH_SYSMSGFILTER: cài đặt hàm câu móc quản lý các thông điệp được tạo ra giống như có một sự kiện của dialog box, message box, menu hay scroll bar. Hàm này quản lý cho tất cả ứng dụng trong cùng một desktop.

- **lpfn**:

Con trỏ chỉ đến địa chỉ của hàm lọc. Nếu tham số *dwThreadId* = 0 hay chỉ đến một luồng được tạo bởi một tiến trình (process) khác, tham số *lpfn* phải chỉ đến một hàm câu móc trong một thư viện liên kết động (DLL). Ngược lại, *lpfn* chỉ đến hàm câu móc chứa trong bản thân tiến trình hiện hành.

- **hMod**:

handle chỉ đến DLL chứa hàm xử lý xác định bằng tham số *lpfn*. Tham số *hMod* phải đặt là NULL nếu hàm câu móc nằm trong tiến trình hiện hành

- **dwThreadId**:

Xác định ID của luồng thực hiện quá trình câu móc. Nếu *dwThreadId* = 0, hàm câu móc sẽ tác động đến tất cả các luồng. Ứng dụng có thể dùng hàm **GetCurrentThreadId** để xác định ID của luồng hiện hành.

Phạm vi thực hiện của hàm câu móc mô tả như sau:

Hook	Phạm vi
WH_CALLWNDPROC	Luồng hay hệ thống
WH_CBT	Luồng hay hệ thống
WH_DEBUG	Luồng hay hệ thống
WH_GETMESSAGE	Luồng hay hệ thống
WH_JOURNALRECORD	Hệ thống
WH_JOURNALPLAYBACK	Hệ thống
WH_FOREGROUNDIDLE	Luồng hay hệ thống
WH_SHELL	Luồng hay hệ thống
WH_KEYBOARD	Luồng hay hệ thống
WH_MOUSE	Luồng hay hệ thống
WH_MSGFILTER	Luồng hay hệ thống
WH_SYSMSGFILTER	Hệ thống

Hàm **SetWindowsHookEx** trả về handle của quá trình câu móc đã cài đặt và trả về NULL nếu quá trình cài đặt không thành công. Handle này được dùng để xóa quá trình câu móc khi sử dụng hàm **UnhookWindowsHookEx**. Các thông báo lỗi khi quá trình câu móc không thành công là:

- ERROR_INVALID_HOOK_FILTER: mã câu móc sai
- ERROR_INVALID_FILTER_PROC: hàm lọc sai
- ERROR_HOOK_NEEDS_HMOD: một quá trình câu móc toàn cục sử dụng tham số hMod = NULL hay chỉ đến một luồng không tồn tại.
- ERROR_GLOBAL_ONLY_HOOK: một quá trình câu móc chỉ dùng được cho hệ thống nhưng được cài đặt cho một luồng xác định.
- ERROR_INVALID_PARAMETER: ID của luồng sai.
- ERROR_JOURNAL_HOOK_SET: Cài đặt thêm một quá trình câu móc dạng nhật ký (WH_JOURNALRECORD và WH_JOURNALPLAYBACK) trong khi một quá trình dạng này đang tồn tại (tại một thời điểm chỉ cho phép một quá trình dạng nhật ký).
- ERROR_MOD_NOT_FOUND: Tham số hMod chỉ đến một hàm không xác định được.
- Khác: không cho phép do bảo mật của hệ thống hay bộ nhớ tràn.

4.1.2. Hàm UnhookWindowsHookEx:

Dùng để xoá một hàm lọc ra khỏi chuỗi xử lý một quá trình câu móc. Hàm này lấy handle của quá trình câu móc trả về từ lệnh gọi hàm SetWindowsHookEx và luôn trả về giá trị TRUE.

Khai báo:

```
Public Declare Function UnhookWindowsHook Lib "user32" Alias "UnhookWindowsHook" (ByVal nCode As Long, ByVal pfnFilterProc As Long) As Long
Public Declare Function UnhookWindowsHookEx Lib "user32" Alias "UnhookWindowsHookEx" (ByVal hHook As Long) As Long
```

4.1.3. Hàm CallNextHookEx:

Dùng để chuyển thông tin câu móc đến hàm câu móc kế tiếp trong chuỗi xử lý.

```
Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long
```

- **hHook**: handle của quá trình câu móc, là giá trị trả về từ lệnh gọi hàm SetWindowsHookEx. Thông thường Windows bỏ qua giá trị này.
- **nCode**: mã của quá trình câu móc, hàm câu móc dùng mã này để xác định phương pháp xử lý thông tin.
- **wParam**: xác định tham số được xử lý bởi hàm câu móc.
- **lParam**: giống như wParam.

Khi một quá trình câu móc khởi động, Windows gọi hàm đầu tiên trong chuỗi hàm lọc và kết thúc quản lý quá trình, các hàm lọc phía sau sẽ không xử lý. Để thực hiện các hàm ở phía sau trong chuỗi hàm, Windows cung cấp hàm CallNextHookEx cho phép gọi một hàm kế tiếp trong chuỗi hàm lọc. Như vậy, nếu một hàm lọc nào đó không thực hiện hàm CallNextHookEx thì các hàm lọc ở phía sau sẽ không thực hiện.

Một ví dụ sử dụng các hàm xử lý câu móc như sau:

```
'Ch•a trong m•t file module
Public Const WH_KEYBOARD = 2
Public Const VK_SHIFT = &H10
Public Const VK_CONTROL = &H11
Public Const VK_MENU = &H12
Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long
Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer 'Xác ••nh tr•ng thái c•a m•t phím (Bit15)
```

```
Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" (ByVal idHook As Long, ByVal lpfn As Long, ByVal hmod As Long, ByVal dwThreadId As Long) As Long
Declare Function UnhookWindowsHookEx Lib "user32" (ByVal hHook As Long) As Long
Public hHook As Long
Public Function KeyboardProc(ByVal idHook As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
If idHook < 0 Then
'G•i hàm x• lý k• ti•p
KeyboardProc = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
Else
'N•u nh•n Shift-C
If (GetKeyState(VK_SHIFT) And &H8000) And wParam = Asc("C") Then
'thì hi•n th• k•t qu•
Form1.Print "Shift-C pressed ..."
End If
If (GetKeyState(VK_CONTROL) And &H8000) And wParam = Asc("C") Then
Form1.Print "Ctrl-C pressed ..."
End If
If (GetKeyState(VK_MENU) And &H8000) And wParam = Asc("C") Then
Form1.Print "Alt-C pressed ..."
End If
'G•i hàm x• lý k• ti•p
KeyboardProc = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
End If
End Function
-----------------------------------------------
'Ch•a trong form
Private Sub Form_Load()
'••t quá trình câu móc
hHook = SetWindowsHookEx(WH_KEYBOARD, AddressOf KeyboardProc, App.hInstance, App.ThreadID)
End Sub
Private Sub Form_Unload(Cancel As Integer)
```

```
'Xoá quá trình câu móc
UnhookWindowsHookEx hHook
End Sub
```

4.2. Hàm lọc

Hàm lọc thường có dạng như sau:

```
Function FilterFunc (ByVal nCode As Integer, ByVal wParam As Long, ByVal lParam As Long)
```

Hàm lọc nhận 3 tham số:

- nCode: mã của quá trình câu móc, là một số nguyên xác định hàm lọc, ví dụ như loại sự kiện làm khởi động quá trình câu móc. Mã này được xác định khi hàm lọc xử lý sự kiện hay gọi hàm **DefHookProc**. Nếu mã câu móc < 0 thì hàm lọc sẽ không xử lý sự kiện mà sẽ gọi hàm **DefHookProc** để truyền 3 tham số còn lại cho hàm lọc kế tiếp trong chuỗi hàm lọc bằng hàm **CallNextHookEx**.
- Tham số thứ hai wParam và thứ ba lParam chứa các thông tin cần thiết cho hàm lọc. Mỗi quá trình câu móc dùng các giá trị wParam và lParam khác nhau. Ví dụ như, quá trình câu móc bàn phím WH_KEYBOARD chứa mã phím nhấn trong wParam và trạng thái bàn phím trong lParam. Hay quá trình câu móc WH_MSGFILTER chứa giá trị NULL trong wParam và một con trỏ chỉ đến thông điệp chứa trong lParam.

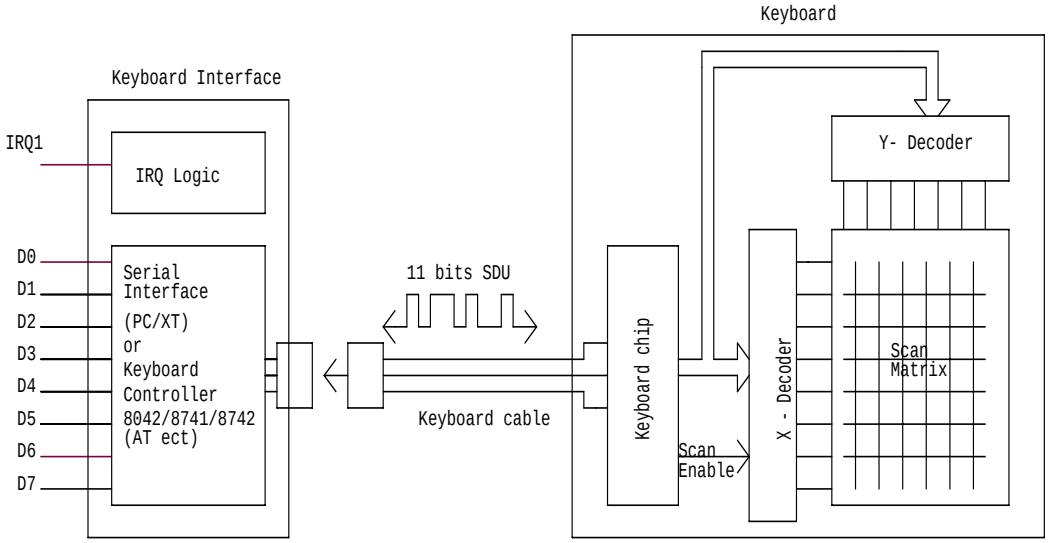
Hàm lọc dùng trong DLL:

Đối với các quá trình câu móc cục bộ, hàm lọc có thể đặt ngay trong mã lệnh của ứng dụng nhưng đối với các quá trình câu móc hệ thống, hàm lọc phải được đặt trong một DLL. Chỉ có quá trình câu móc dạng nhật ký (WH_JOURNALRECORD và WH_JOURNALPLAYBACK) là ngoại lệ. Hàm lọc của quá trình câu móc hệ thống phải chia sẻ dữ liệu cho tiến trình thực hiện quá trình câu móc. Các biến toàn cục sử dụng trong DLL phải được xác định rõ hay phải đặt trong vùng dữ liệu chia sẻ.

Chương 3
GIAO TIẾP THIẾT BỊ CHUẨN

1. Giao tiếp bàn phím

1.1. Nguyên lý hoạt động



Hình 3.1 - Sơ đồ nguyên lý và các ghép nối của bàn phím

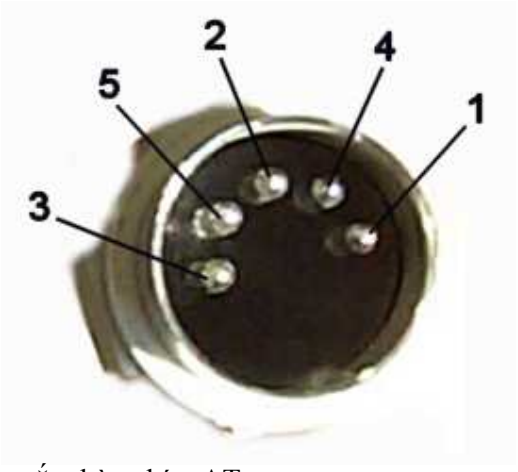
Chip xử lý bàn phím liên tục kiểm tra trạng thái của ma trận quét (scan matrix) để xác định công tắc tại các tọa độ X, Y đang được đóng hay mở và ghi một mã tương ứng vào bộ đệm bên trong bàn phím. Sau đó mã này sẽ được truyền nối tiếp tới mạch ghép nối bàn phím trong PC. Cấu trúc của SDU (Serial Data Unit) cho việc truyền số liệu:

0 10

STRT	DB_0	DB_1	DB_2	DB_3	DB_4	DB_5	DB_6	DB_7	PAR	STOP

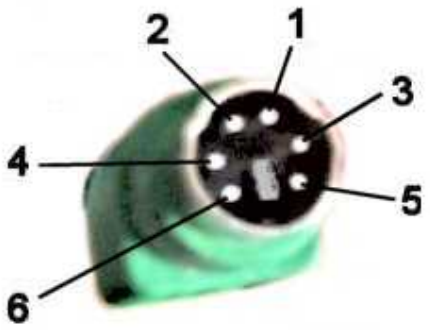
STRT: bit start (luôn bằng 0)
DB_0 - DB_7: bit số liệu từ 0 đến 7.
PAR: bit parity (luôn lẻ)
STOP: bit stop (luôn bằng 1).

Chân 1: clock
Chân 2: dữ liệu
Chân 3: Reset
Chân 4: GND
Chân 5: Vcc



Hình 3.2 – Đầu cắm bàn phím AT

Chân 1: dữ liệu
Chân 2: không dùng
Chân 3: GND
Chân 4: Vcc
Chân 5: clock
Chân 6: không dùng



Hình 3.3 – Đầu cắm bàn phím PS/2

Mỗi phím nhấn sẽ được gán cho 1 mã quét (scan code) gồm 1 byte. Nếu 1 phím được nhấn thì bàn phím phát ra 1 mã make code tương ứng với mã quét truyền tới mạch ghép nối bàn phím của PC. Ngắt cứng INT 09h được phát ra qua IRQ1.

Chương trình xử lý ngắt sẽ xử lý mã này tuỳ theo phím SHIFT có được nhấn hay không. Ví dụ: nhấn phím SHIFT trước, không rời tay và sau đó nhấn ‘C’:

make code được truyền - 42(SHIFT) - 46 (‘C’).

Nếu rời tay nhấn phím SHIFT thì bàn phím sẽ phát ra break code và mã này được truyền như make code. Mã này giống như mã quét nhưng bit 7 được đặt lên 1, do vậy nó tương đương với make code cộng với 128. Tuỳ theo break code, chương trình con xử lý ngắt sẽ xác định trạng thái nhấn hay rời của các phím. Thí dụ, phím SHIFT và ‘C’ được rời theo thứ tự ngược lại với thí dụ trên:

break code được truyền 174 (bằng 46 cộng 128 tương ứng với ‘C’) và 170 (bằng 42 cộng 128 tương ứng với SHIFT).

Phần cứng và phần mềm xử lý bàn phím còn giải quyết các vấn đề vật lý sau:

- Nhấn và nhả phím nhưng không được phát hiện.

- Khử nhiễu rung cơ khí và phân biệt 1 phím được nhấn nhiều lần hay được nhấn chỉ 1 lần nhưng được giữ trong một khoảng thời gian dài.

1.2. Lập trình giao tiếp qua các cổng

Bàn phím cũng là một thiết bị ngoại vi nên về nguyên tắc có thể truy xuất nó qua các cổng vào ra.

❖ ***Các thanh ghi và các port:***

Sử dụng 2 địa chỉ port 60h và 64h có thể truy xuất bộ đệm vào, bộ đệm ra và thanh ghi điều khiển của bàn phím.

Port	Thanh ghi	R/W
60h	Đệm ngõ ra	R
60h	Đệm ngõ vào	W
64h	Thanh ghi điều khiển	W
64h	Thanh ghi trạng thái	R

Thanh ghi trạng thái xác định trạng thái hiện tại của bộ điều khiển bàn phím. Thanh ghi này chỉ đọc (read only) và đọc bằng lệnh IN tại port 64h.

7							0
PARE	TIM	AUXB	KEYL	C/D	SYSF	INPB	OUTB

PARE: Lỗi chẵn lẻ của byte cuối cùng được vào từ bàn phím; 1 = có lỗi chẵn lẻ, 0 = không có.

TIM: Lỗi quá thời gian (time-out); 1 = có lỗi, 0 = không có.

AUXB: Đệm ra cho thiết bị phụ (chỉ có ở máy PS/2); 1 = giữ số liệu cho thiết bị, 0 = giữ số liệu cho bàn phím.

KEYL: Trạng thái khóa bàn phím; 1 = không khóa, 0 = khóa.

C/D: Lệnh/dữ liệu; 1 = Ghi qua port 64h, 0 = Ghi qua port 60h.

SYSF: cờ hệ thống; 1 = tự kiểm tra thành công, 0 = reset khi cấp điện

INPB: Trạng thái đệm vào; 1 = dữ liệu CPU trong bộ đệm vào, 0 = đệm vào rỗng.

OUTB: Trạng thái đệm ra; 1 = dữ liệu bộ điều khiển bàn phím trong bộ đệm ra, 0 = đệm ra rỗng.

Thanh ghi điều khiển

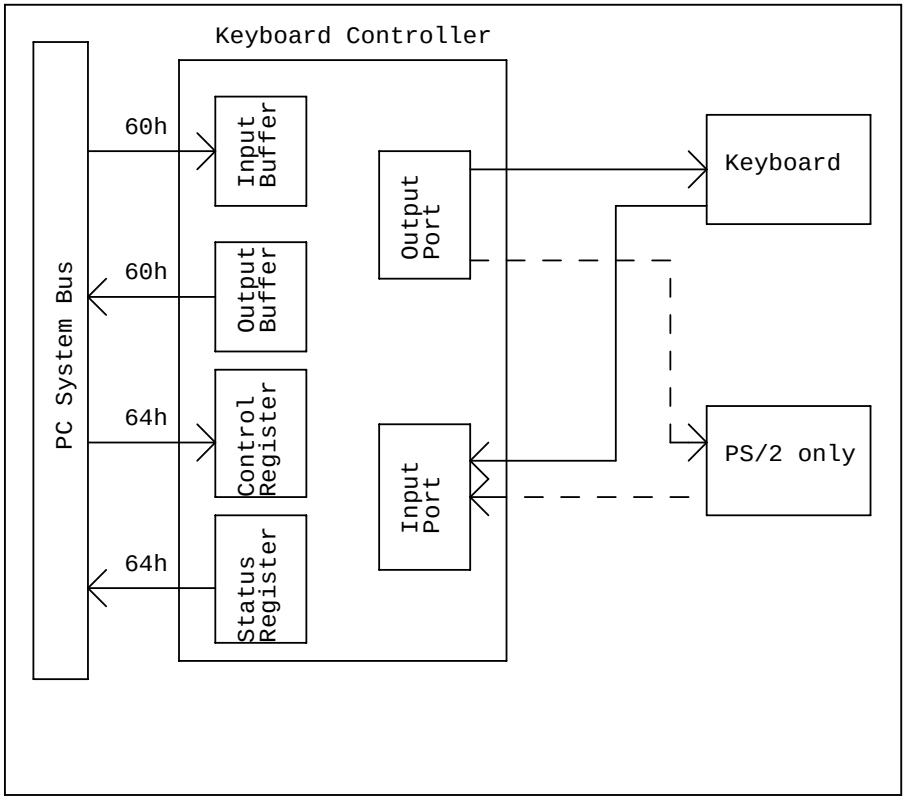
Các lệnh cho bộ điều khiển bàn phím:

Mã	Mô tả
A7h	Cấm thiết bị phụ
A8h	Cho phép thiết bị phụ
A9h	Kiểm tra giao tiếp thiết bị phụ và lưu mã kiểm tra vào bộ đệm ra 00h: không lỗi 01h: CLK ở mức thấp 02h: CLK ở mức cao 03h: DATA ở mức thấp

	04h: DATA ở mức cao FFh: lỗi khác
AAh	Tự kiểm tra (ghi 55h vào bộ đệm ra nếu không lỗi
ABh	Kiểm tra giao tiếp bàn phím và lưu mã kiểm tra vào bộ đệm ra
ADh	Cấm bàn phím
AEh	Cho phép bàn phím
C0h	Đọc cổng vào và truyền dữ liệu đến bộ đệm ra
C1h	Đọc các bit 3 – 0 của cổng vào và truyền đến các bit 3- 0 của thanh ghi trạng thái cho đến khi INPB = 1
C2h	Đọc các bit 7 – 4 của cổng vào và truyền đến các bit 7- 4 của thanh ghi trạng thái cho đến khi INPB = 1
D0h	Đọc cổng ra
D1h	Ghi cổng ra
D2h	Ghi vào bộ đệm ra và xoá AUXB
D3h	Ghi vào bộ đệm ra và set AUXB
D4h	Ghi byte dữ liệu tiếp theo vào thiết bị phụ

Khóa bàn phím:

```
Start:
IN AL, 64h          ; đọc byte trạng thái
TEST AL, 02h        ; kiểm tra bộ đệm có đầy hay không
JNZ start
OUT 64h, 0ADh       ; khóa bàn phím
```



Hình 3.4 - Bộ điều khiển bàn phím

❖ ***Các lệnh cho bàn phím:***

Mã	Lệnh	Mô tả
EDh	Bật/tắt LED	Bật/tắt các đèn led của bàn phím
EEh	Echo	Trả về byte EEh
F0h	Đặt/nhận dạng mã quét	Đặt 1 trong 3 tập mã quét và nhận diện các mã quét tập mã quét hiện tại.
F2h	Nhận diện bàn phím	Nhận diện ACK = AT, ACK+abh+41h=MF II.
F3h	Đặt tốc độ lặp lại/trễ	Đặt tốc độ lặp lại và thời gian trễ của bàn phím
F4h	Enable	Cho phép bàn phím hoạt động
F5h	Chuẩn/không cho phép	Đặt giá trị chuẩn và cấm bàn phím.
F6h	Chuẩn/cho phép	Đặt giá trị chuẩn và cho phép bàn phím.
FEh	Resend	Bàn phím truyền ký tự cuối cùng một lần nữa tới bộ điều khiển bàn phím
FFh	Reset	Chạy reset bên trong bàn phím

Thí dụ: lệnh bật đèn led cho phím NUMCLOCK, tắt tất cả các đèn khác.

```
MOV AL,0EDh
OUT  60H, AL
WAIT:
IN  AL, 64H          ; đọc thanh ghi trạng thái
JNZ   WAIT
MOV AL,02h
OUT  60H, AL         ; bật đèn cho numclock
```

Cấu trúc của byte chỉ thị như sau:

7					2	1	0
0	0	0	0	0	CPL	NUM	SCR

CPL: 1 = bật đèn Caps Lock; 0 = tắt
NUM: 1 = bật đèn Num Lock; 0 = tắt
SCR: 1 = bật đèn Scroll Lock; 0 = tắt

1.3. Lập trình giao tiếp qua các hàm của DOS, BIOS

BIOS ghi các ký tự do việc nhấn các phím vào bộ đệm tạm thời được gọi là bộ đệm bàn phím (keyboard buffer), có địa chỉ 40h:1Eh, gồm 32 byte và kết thúc ở địa chỉ 40h:3Dh. Mỗi ký tự được lưu trữ bằng 2 byte, byte cao là mã quét, và byte thấp là mã ASCII. Chương trình xử lý ngắt sẽ xác định mã ASCII từ mã quét bằng bảng biến đổi và ghi cả 2 mã vào bộ đệm bàn phím. Bộ đệm bàn phím được tổ chức như bộ đệm vòng (ring buffer) và được quản lý bởi 2 con trỏ. Các giá trị con trỏ được lưu trữ trong vùng dữ liệu của BIOS ở địa chỉ 40h:1Ah và 40h:1Ch. Con trỏ ghi (40h:1Ch) cho biết vị trí còn trống kế tiếp để ghi ký tự nhập, con trỏ đọc (40h:1Ah) cho biết vị trí ký tự đầu tiên sẽ đọc. Từ đó, bộ đệm bàn phím rỗng khi con trỏ ghi và con trỏ đọc trùng nhau → bộ đệm chỉ chứa được 15 ký tự.

Các hàm của ngắt 16h:

Hàm 0h - đọc ký tự từ bàn phím, nếu không nhấn thì sẽ chờ

Ra: AH = scancode, AL = mã ASCII. Nếu phím nhấn là các phím đặc biệt thì AL = 0

Hàm 1h - ZF = 1 nếu không có ký tự trong bộ đệm. Giá trị trả về giống như hàm 00h nhưng không xoá ký tự ra khỏi bộ đệm

Hàm 2h - Trả về trạng thái của các phím, kết quả chứa trong AL

7	6	5	4	3	2	1	0
INS	CAPS LOCK	NUM LOCK	SCROLL LOCK	ALT	CTRL	LEFT SHIFT	RIGHT SHIFT

Hàm 10h - Giống hàm 00h nhưng trả về mã mở rộng

Hàm 11h - Giống hàm 01h nhưng trả về mã mở rộng

Hàm 12h - Giống hàm 02h nhưng AH chứa thêm các thông tin

7	6	5	4	3	2	1	0
SYS REQ	CAPS LOCK	NUM LOCK	SCROLL LOCK	RIGHT ALT	RIGHT CTRL	LEFT ALT	LEFT CTRL

Các thí dụ:

- Giả sử phím ‘c’ đã được nhấn.

```
MOV AH,00h
INT 16h
```

Kết quả: AH = 2Eh (mã quét cho phím ‘a’); AL = 63h (ASCII cho ‘c’)

- Giả sử phím ‘HOME’ đã được nhấn.

```
MOV AH,00h
INT 16h
```

Kết quả: AH = 47h (mã quét cho phím ‘HOME’)
AL = 0 (các phím chức năng và điều khiển không có mã ASCII)

- Giả sử phím ‘HOME’ đã được nhấn.

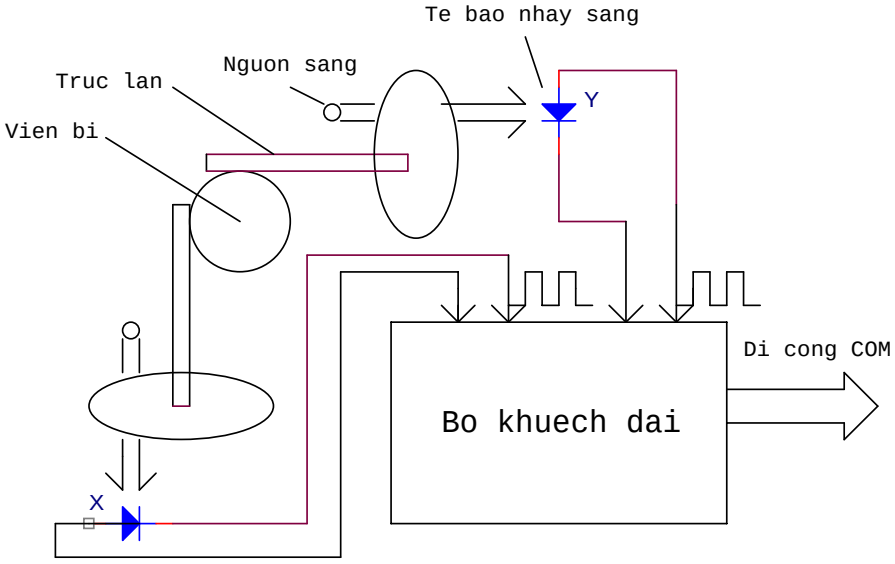
```
MOV AH,10h
INT 16h
```

Kết quả: AH = 47h (mã quét cho phím ‘HOME’)
AL = E0h

2. Giao tiếp chuột

2.1. Cấu tạo

Cấu tạo của chuột rất đơn giản, phần trung tâm là 1 viên bi thép được phủ keo hoặc nhựa được quay khi dịch chuyển chuột. Chuyển động này được truyền tới 2 thanh nhỏ được đặt vuông góc với nhau. Các thanh này sẽ biến chuyển động của chuột theo 2 hướng X,Y thành sự quay tương ứng của 2 đĩa gắn với chúng. Trên 2 đĩa có những lỗ nhỏ liên tục đóng và ngắt 2 chùm sáng tới các sensor nhạy sáng để tạo ra các xung điện. Số các xung điện tỷ lệ với lượng chuyển động của chuột theo các hướng X,Y và số xung trên 1 sec biểu hiện tốc độ của chuyển động chuột. Kèm theo đó có 2 hay 3 phím bấm.



Hình 3.5 - Sơ đồ cấu tạo của chuột

2.2. Mạch ghép nối và chương trình điều khiển chuột

Hầu hết chuột được nối với PC qua cổng nối tiếp, qua đó chuột cũng được cấp nguồn nuôi từ PC. Khi dịch chuyển hoặc nhấn, nhả các phím chuột, nó sẽ phát ra một gói dữ liệu tới mạch giao tiếp và mạch sẽ phát ra 1 ngắt. Phần mềm điều khiển chuột làm các nhiệm vụ: chuyển ngắt tới mạch giao tiếp nối tiếp xác định, đọc dữ liệu và cập nhật các giá trị bên trong liên quan tới trạng thái của bàn phím cũng như vị trí của chuột. Hơn nữa, nó còn cung cấp 1 giao tiếp mềm qua ngắt 33h để định các giá trị bên trong này cũng như làm dịch chuyển con trỏ chuột trên màn hình tương ứng với vị trí của chuột.

Có thể chọn kiểu con trỏ chuột cứng hoặc mềm trong chế độ văn bản hay con trỏ chuột đồ hoạ trong chế độ đồ họa. Các hàm 09h và 0Ah trong ngắt 33h cho phép định nghĩa loại và dạng con trỏ chuột.

2.3. Chương trình với con trỏ

Ngắt 33h cho phép xác định vị trí, số lần click chuột và hình dạng con trỏ (số thứ tự hàm chứa trong AX).

Hàm	Ý nghĩa	Tham số
0	Reset chuột	Ra: AX = 0: nếu có, = 1: không BX = số nút nhấn
1	Hiển thị con trỏ	
2	Ẩn con trỏ	
3	Nhận vị trí con trỏ và trạng thái nút	Ra: BX: trạng thái nút (D0: nút trái, D1: nút phải, D2: nút giữa) (= 0: nhả, = 1: nhấn) CX: vị trí ngang DX: vị trí dọc
4	Đặt vị trí con trỏ	Vào: CX: vị trí ngang DX: vị trí dọc
5	Trạng thái nút và số lần nhấn từ khi gọi	Vào: BX = nút kiểm tra (=0: trái, =1: phải) Ra: AX = trạng thái nút BX = số lần nhấn CX: vị trí ngang DX: vị trí dọc lần nhấn cuối
6	Giống hàm 05h nhưng kiểm tra số lần nhả	
7	Giới hạn dịch chuyển ngang của con trỏ	Vào: CX = cột trái DX = cột phải
8	Giới hạn dịch chuyển dọc của con trỏ	Vào: CX = dòng dưới DX = dòng trên
9	Xác định hình dạng con trỏ đồ hoạ	Vào: BX = vị trí ngang CX = vị trí dọc ES:DX: địa chỉ mặt nạ màn hình và con trỏ
A	Xác định hình dạng con trỏ văn bản	Vào: BX = 0: con trỏ phần mềm CX = mặt nạ màn hình DX = mặt nạ con trỏ BX = 1: con trỏ phần cứng CX = dòng bắt đầu DX = dòng kết thúc

Chú ý rằng toạ độ con trỏ xác định theo pixel với độ phân giải 640x200 trong khi chế độ văn bản sử dụng toạ độ ký tự 80x25 nên để chuyển sang toạ độ ký tự thì phải chia cho 8. Con trỏ chuột hiển thị trên màn hình đồ hoạ bằng cách thực hiện:

Từ mới = (từ cũ AND mặt nạ màn hình) XOR mặt nạ con trỏ

Nếu ta đặt mặt nạ màn hình là 0 thì ký tự màn hình tại đó sẽ bị xoá.

VD: Con trỏ chuột mềm nhấp nháy và chứa ký tự 'A'

```
MOV AH,0Ah
MOV BX,0
MOV CX,0 ; m•t n• màn hình = 0
MOV DH,8Bh; = 10001011b → màu n•n Gray, màu ký t• Cyan
MOV DL,'A'
INT 33h
```

VD: Con trỏ chuột cứng có các đường quét 3 và 8

```
MOV AH,0Ah
MOV BX,1
MOV CX,03h
MOV DX,08h
INT 33h
```

3. Giao tiếp màn hình

3.1. Card màn hình

Để hiện các hình ảnh, ký tự, hay hình vẽ trên màn hình, PC phải thông qua mạch ghép nối màn hình (graphics adapter). Board mạch này thường được cắm trên khe cắm mở rộng của PC.

Trong chế độ văn bản (text mode), các ký tự được xác định bởi mã ASCII, trong đó có cả các thông tin về thuộc tính của ký tự, thí dụ ký tự được hiện theo cách nhấp nháy hay đảo màu đen trắng ….ROM ký tự (character rom) lưu trữ các hình mẫu điểm ảnh của các ký tự tương ứng để máy phát ký tự biến đổi các mã ký tự đó thành 1 chuỗi các bit điểm ảnh (pixel bit) và chuyển chúng tới thanh ghi dịch (shift register). Máy phát tín hiệu sẽ sử dụng các bit điểm ảnh này cùng với các thông tin thuộc tính từ RAM video và các tín hiệu đồng bộ từ CRTC để phát ra các tín hiệu cần thiết cho monitor.

Trong chế độ đồ họa (graphics mode), thông tin trong RAM video được sử dụng trực tiếp cho việc phát ra các ký tự. Lúc này các thông tin về thuộc tính cũng không cần nữa. Chỉ từ các giá trị bit trong thanh ghi dịch, máy phát tín hiệu sẽ phát các tín hiệu về độ sáng và màu cho monitor.

Mỗi ký tự được biểu diễn bởi 1 từ 2 byte trong RAM video. Byte thấp chứa mã ký tự, byte cao chứa thuộc tính. Cấu trúc của một từ nhớ video như sau:

15	14	13	12	11	10	9	8
BLNK	BAK_2	BAK_1	BAK_0	INT	FOR_2	FOR_1	FOR_0

7	6	5	4	3	2	1	0
CHR_7	CHR_6	CHR_5	CHR_4	CHR_3	CHR_2	CHR_1	CHR_0

BLNK: Nhấp nháy; 1 = bật, 0 = tắt
INT: Cường độ sáng ; 1 = cao, 0 = bình thường
$CHR_7 \ldots CHR_{0:}$ Mã ký tự.

Bảng màu quy định như sau:

Mã hex	Màu	Mã hex	Màu	Mã hex	Màu	Mã hex	Màu
0	Black	4	Red	8	Dark Gray	C	Light Red
1	Blue	5	Magenta	9	Light Blue	D	Light Magenta
2	Green	6	Brown	A	Light Green	E	Yellow
3	Cyan	7	Light Gray	B	Light Cyan	F	White

Trong chế độ văn bản, 6845 liên tục xuất các địa chỉ cho RAM video qua MA0-MA13. Ký tự ở góc tận cùng phía trên bên trái màn hình có địa chỉ thấp nhất mà 6845 sẽ cung cấp ngay sau khi quét dọc ngược. Logic ghép nối định địa chỉ cho RAM video bằng việc lấy ra mã ký tự cùng với thuộc tính. Mã ký tự dùng cho máy phát ký tự như là chỉ số thứ nhất trong ROM ký tự. Lúc này, 6845 định địa chỉ hàng quét đầu tiên của ma trận ký tự, địa chỉ hàng bằng 0. Các bit của ma trận điểm ảnh bây giờ sẽ được truyền đồng bộ với tần số video từ thanh ghi dịch tới máy phát tín hiệu. Nếu máy phát tín hiệu nhận được giá trị 1 từ thanh ghi dịch, nó sẽ phát tín hiệu video tương ứng với màu của ký tự. Nếu nhận được 0 nó sẽ cấp tín hiệu tương ứng với màu nền. Vậy dòng quét thứ nhất được hiện phù hợp với các ma trận điểm ảnh của các ký tự trong hàng ký tự thứ nhất. Khi tia điện tử đạt tới cuối dòng quét, 6845 kích hoạt lối ra HS (Horizontal Synchronization) để tạo ra quá trình quét ngược và đồng bộ ngang. Tia điện tử quay trở về bắt đầu quét dòng tiếp. Sau mỗi dòng quét, 6845 tăng giá trị RA0-RA4 lên 1. Địa chỉ dòng này hình thành một giá trị offset bên trong ma trận điểm ảnh cho ký tự được hiện. Dựa trên mỗi dòng quét như vậy, một dòng các điểm ảnh của ký tự trong hàng ký tự được hiện ra. Điều này có nghĩa là với ma trận 9x14 điểm ảnh cho 1 ký tự, hàng ký tự thứ nhất đã được hiện sau 14 dòng quét. Khi địa chỉ RA0-RA4 trở về giá trị 0, 6845 sẽ cấp 1 địa chỉ MA0-MA13 mới và hàng ký tự thứ hai sẽ được hiện ra cũng như vậy. Ở cuối dòng quét cuối cùng, 6845 sẽ reset địa chỉ MA0-MA13 và RA0-RA4 và cho phép lối ra VS (Vertical Synchronization) phát ra tín hiệu quét ngược cùng tín hiệu đồng bộ dọc.

Mỗi ký tự có chiều cao cực đại ứng với 32 dòng vì có 5 đường địa chỉ RA0-RA4, còn bộ nhớ video trong trường hợp này được tới 16K từ vì có địa chỉ MA0-MA13 là 14 bit. Trong chế độ đồ họa, chúng kết hợp với nhau để tạo thành địa chỉ 19 bit, lúc đó 6845 có thể định địa chỉ cho bộ nhớ video lên tới 512K từ. Trong trường hợp này, các byte trong RAM video không được dịch thành mã ký tự và thuộc tính nữa mà trực tiếp xác định cường độ sáng và màu của điểm ảnh. Đa số các RAM video được chia thành vài băng được định địa chỉ bởi RA0-RA4. Các đường MA0-MA13 sẽ định địa chỉ offset bên trong mỗi băng. Dữ liệu trong RAM video lúc này được trực tiếp truyền tới thanh ghi dịch và máy phát tín hiệu. ROM ký tự và máy phát ký tự không làm việc.

RAM video được tổ chức khác nhau tuỳ theo chế độ hoạt động và bản mạch ghép nối. Thí dụ, với RAM video 128 KB, có thể địa chỉ hóa toàn bộ bộ nhớ màn hình qua CPU như bộ nhớ chính. Nhưng nếu kích thước RAM video lớn hơn thì làm như vậy sẽ đè lên vùng ROM mở rộng ở địa chỉ C0000h. Do đó, card EGA và VGA với trên 128 KB nhớ được tăng cường thêm 1 chuyển mạch mềm (soft-switch) cho phép thâm nhập các cửa sổ

128 KB khác nhau vào RAM video lớn hơn nhiều. Các chuyển mạch này được quy định bởi riêng các nhà sản xuất board mạch.

3.2. Chế độ văn bản

RAM video được coi như một dãy từ tuyến tính, từ đầu tiên được gán cho ký tự góc trên tận cùng bên trái màn hình gọi là hàng 1 cột 1. Từ thứ 2 là hàng 1, cột 2, …. Số từ tuỳ thuộc vào độ phân giải của kiểu hiện ký tự.

Thí dụ: độ phân giải chuẩn 25 hàng, 80 ký tự đòi hỏi 2000 từ nhớ 2 byte. Như vậy, tổng cộng cần 4 KB bộ nhớ RAM video. Trong khi đó với card có độ phân giải cao SVGA 60 hàng, 132 ký tự cần đến 15840 byte. Do đó RAM video thường được chia thành vài trang. Kích thước của mỗi trang tuỳ thuộc vào chế độ hiện của màn hình và số trang cực đại, phụ thuộc cả vào kích thước của RAM video. 6845 có thể được lập trình sao cho địa chỉ khởi phát của MA0-MA13 sau quét ngược dọc là khác 00h. Nếu địa chỉ khởi phát là bắt đầu của 1 trang thì có thể quản lý RAM video theo vài trang tách biệt nhau, nếu CPU thay đổi nội dung của 1 trang mà trang đó hiện đang không hiện thì màn hình cũng không thay đổi. Do đó, cần phân biệt trang nhớ đang được kích hoạt (đang hiện) và trang đang được xử lý.

Đoạn chương trình ghi ký tự 'A' có cường độ sáng cao vào góc trên bên trái với màu số 7 và màu nền số 0. Trang thứ nhất và là duy nhất bắt đầu ở địa chỉ B0000h.

```
MOV AX, 0B000h
MOV ES, AX
MOV AH, 0F8h
MOV AL, 41h
MOV ES:[00H],AX
```

3.3. Chế độ đồ hoạ

Tổ chức trong chế độ này phức tạp hơn. Ví dụ: với card Hercules, RAM video được chia thành 4 băng trên 1 trang . Băng thứ nhất: đảm bảo các điểm ảnh cho các dòng 0, 4, 8, …, 344; băng thứ hai cho các dòng 1, 5, 9, …, 345; băng thứ 3 cho các dòng 2, 6, 10, …., 346; và băng thứ 4 cho các dòng 3, 7, 11, …, 347. 64 KB được chia thành 2 trang 32 KB. Độ phân giải trong chế độ đồ họa là 720 x 348 điểm ảnh, mỗi điểm ảnh được biểu diễn bởi 1 bit. Do vậy, một dòng cần 90 byte (720 điểm ảnh / 8 điểm ảnh trên 1 byte). Địa chỉ của byte chứa điểm ảnh thuộc đường i và cột j trong trang k là:

B0000h+8000h*k+2000h*(i mod 4)+ 90*int(i/4)+int(j/8)

B0000h là đoạn video, 8000h là kích thước của trang, 2000h*(i mod 4) là offset của băng chứa byte đó, 90*int(i/4) là offset của dòng i trong băng và int(j/8) là offset của cột j trong băng.

3.4. Truy xuất màn hình qua DOS và BIOS

3.4.1. Truy xuất qua DOS

Các hàm của int 21h có thể hiện các ký tự trên màn hình nhưng không can thiệp được vào màu:

- Hàm 02h: ra màn hình.
- Hàm 09h: ra một chuỗi.
- Hàm 40h: ghi file/ thiết bị

Các lệnh **copy, type** và **print** trong command.com cho phép hiện text trên màn hình. DOS gộp chung bàn phím và monitor thành 1 thiết bị mang tên CON (console). Ghi CON là truyền số liệu tới monitor, còn đọc CON là nhận ký tự từ bàn phím. Ví dụ: để hiện nội dung của file output.txt lên màn hình của monitor sẽ có các cách sau:

- copy output.txt con
- type output.txt > con
- print output.txt /D:con

3.4.2. Truy xuất qua BIOS

BIOS thâm nhập monitor bằng int 10h với nhiều chức năng hơn DOS, như đặt chế độ hiện hình, quản lý tự động các trang, phân biệt các điểm trên màn hình nhờ các tọa độ,…

- ***Hàm 00h***: định chế độ đồ hoạ:

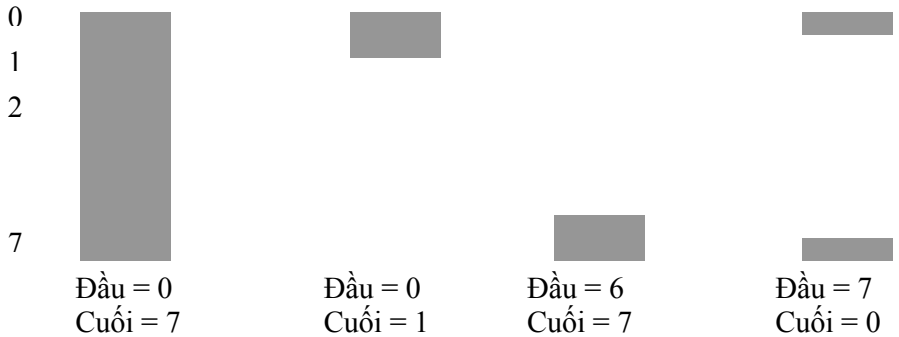
Vào:

AL = chế độ

Mode	Type	Max Colors	Size	Resolution	Max Pages	Base Addr
00	Text	16	40x25	- -	8	B8000h
01	Text	16	40x25	- -	8	B8000h
02	Text	16	80x25	- -	4,8	B8000h
03	Text	16	80x25	- -	4,8	B8000h
04	Graphics	4	40x25	320x200	1	B8000h
05	Graphics	4	40x25	320x200	1	B8000h
06	Graphics	2	80x25	640x200	1	B8000h
07	Text	Mono	80x25	- -	1,8	B0000h
08	Graphics	16	20x25	? ?	1	B0000h
09	Graphics	16	40x25	? ?	1	B0000h
0A	Graphics	4	80x25	? ?	1	B0000h
0B	- -	-	- -	- -	-	- -
0C	- -	-	- -	- -	-	- -
0D	Graphics	16	40x25	320x200	8	A0000h
0E	Graphics	16	80x25	640x200	4	A0000h
0F	Graphics	Mono	80x25	640x350	2	A0000h
10	Graphics	16	80x25	640x350	2	A0000h
11	Graphics	2	80x25	640x480	1	A0000h
12	Graphics	16	80x25	640x480	1	A0000h
13	Graphics	256	40x25	320x200	1	A0000h

- ***Hàm 01h***: xác định hình dạng con trỏ

Vào:



CH = dòng đầu con trỏ

CL = dòng cuối con trỏ

- ***Hàm 02h***: xác định vị trí con trỏ

Vào:

DH = hàng, DL = cột (bắt đầu từ toạ độ 0,0)

BH = trang (= 0 trong chế độ đồ hoạ)

- ***Hàm 03h***: lấy vị trí và hình dạng con trỏ

Vào: BH = trang

Ra:

DH = hàng, DL = cột

CH, CL = tham số xác định hình dạng con trỏ

- ***Hàm 05h***: chọn trang màn hình

AL = số trang (0..7)

- ***Hàm 06h***: cuộn cửa sổ lên

Vào:

AL = số hàng cuộn (= 0: cuộn toàn màn hình)

BH = thuộc tính các hàng trống xuất hiện khi cuộn

CH, CL = hàng, cột của góc phía trên bên trái

DH, DL = hàng, cột của góc phía dưới bên phải

- ***Hàm 07h***: cuộn cửa sổ xuống

Giống hàm 06h

- ***Hàm 08h***: đọc ký tự và thuộc tính ký tự tại vị trí con trỏ

Vào: BH = trang

Ra:

AH = thuộc tính

AL = mã ASCII

- ***Hàm 09h***: ghi ký tự và thuộc tính ký tự tại vị trí con trỏ

Vào:

BH = trang

BL = thuộc tính (theo bảng trang 61)

AL = mã ASCII

CX = số ký tự cần xuất

- ***Hàm 0Ah***: ghi ký tự tại vị trí con trỏ

Vào:

BH = trang

AL = mã ASCII

CX = số ký tự cần xuất

- ***Hàm 0Bh***: chọn bảng màu (dùng cho chế độ đồ hoạ)

Vào: BH = 0, BL = chọn màu nền (theo bảng trang 61)

BH = 1, BL = số bảng màu

Palette	Pixel	Color	Palette	Pixel	Color
0	0	Black	1	0	Black
0	1	Green	1	1	Cyan
0	2	Red	1	2	Magenta
0	3	Brown	1	3	White

- ***Hàm 0Ch***: hiện một điểm trên màn hình (dùng cho chế độ đồ hoạ)

AL = giá trị pixel (0 – 3), nếu AL.7 = 1 thì giá trị màu là phép toán XOR vứi giá trị màu hiện hành

CX = cột, DX = hàng

- ***Hàm 0Dh***: đọc một điểm trên màn hình (dùng cho chế độ đồ hoạ)

Vào: CX = cột, DX = hàng

Ra: AL = giá trị pixel

- ***Hàm 0Fh***: xác định chế độ màn hình hiện hành

Ra:

AL = chế độ

AH = số cột

BH = số trang

- ***Hàm 13h***: xuất chuỗi

Vào:

AL = chế độ xuất (bit0 = 1: cập nhật vị trí con trỏ sau khi xuất, bit1 = 1: chuỗi có chứa thuộc tính ký tự)

BH = trang

BL = thuộc tính (theo bảng trang 61)

CX = số ký tự cần xuất

CX = số ký tự cần xuất

DH, DL = hàng, cột xuất chuỗi

ES:BP = chuỗi in (nếu AL.1 = 1 thì chuỗi có dạng *'Kýtự',thuộc tính,'Kýtự',thuộc tính,'Kýtự',thuộc tính…*)

❖ **Những thường trình đồ họa:**

BIOS trên main board có sẵn những hàm dùng cho thâm nhập MDA và CGA. BIOS của riêng EGA và VGA có những hàm mở rộng tương ứng trong khi vẫn giữ nguyên định dạng gọi.

Một trong những hàm quan trọng nhất của int 10h là hàm 00h dùng để đặt chế độ hiện hình. Để thay đổi chế độ hiện hình cần phải làm rất nhiều bước chương trình phức tạp để nạp các thanh ghi của chip 6845. Trong khi đó, hàm 00h làm cho ta tất cả các công việc này.

Thí dụ: tạo kiểu 6 với độ phân giải 640*200 trên CGA.

```
Mov ah, 00h      ; hàm 00h
Mov al, 06h      ; chế độ 6
Int 10h          ; gọi ngắt
```

Các card EGA/VGA có riêng BIOS của chúng. Trong quá trình khởi động PC, nó sẽ chặn int 10h lại và chạy chương trình BIOS riêng. Thường trình cũ (của BIOS trên board mach chính) được thay địa chỉ tới int 42h. Tất cả các lệnh gọi int 10h sẽ được BIOS của EGA/VGA thay địa chỉ tới int 42h nếu board mạch EGA/VGA đang chạy các kiểu hiện tương thích với MDA hay CGA. Có các kiểu hoạt động từ 0 đến 7.

BIOS của EGA/VGA dùng vùng 40:84h tới 40:88h để lưu số liệu BIOS và các thông số của EGA/VGA. Nó có các hàm mới với các hàm phụ sau:

- Hàm 10h: truy xuất các thanh ghi màu và bảng màu
- Hàm 11h: cài đặt các bảng định nghĩa ký tự mới
- Hàm 12h: đặt cấu hình hệ con video
- Hàm 1Bh: thông tin về trạng thái và chức năng của BIOS video (chỉ có ở VGA)
- Hàm 1Ch: trạng thái save/restore của video (chỉ có ở VGA)

Sau đây là chức năng của các hàm và thí dụ sử dụng chúng:

- Hàm 10h, hàm phụ 03h – xoá/đặt thuộc tính

Ví dụ: Xoá thuộc tính nhấp nháy:

```
Mov ah, 10h      ; dùng hàm 10h
Mov al, 03h      ; dùng hàm phụ 03h
Mov bl, 00h      ; xoá thuộc tính nhấp nháy
Int 10h          ; gọi ngắt
```

- Hàm 11h – ghép nối với máy phát ký tự

Ví dụ: Nạp bảng định nghĩa ký tự 8*14 không cần chương trình CRTC:

```
Mov ah, 11h      ; dùng hàm 11h
Mov al, 01h      ; nạp bảng ký tự từ Rom Bios vào Ram máy phát ký tự.
Mov bl, 03h      ; gán số 3 cho bảng
```

```
Int 10h          ; gọi ngắt
```

- Hàm 12h, hàm phụ 20h – chọn thường trình in màn hình. Dùng hàm phụ này có thể thay thế thường trình chuẩn cho INT 05h bằng thường trình có thể dùng cho các độ phân giải mới của EGA/VGA.

Ví dụ: Cho phép thường trình mới in màn hình:

```
Mov ah, 12h      ; dùng hàm 12h
Mov bl, 20h      ; dùng hàm phụ 20h
```

Ấn PRINT hoặc SHIFT+PRINT để gọi thường trình in đã được lắp đặt.

❖ Truy xuất trực tiếp bộ nhớ video:

Để vẽ 1 điểm trên màn hình, BIOS phải làm nhiều thủ tục nhưng nếu muốn vẽ toàn bộ 1 cửa sổ hình hay lưu trữ thì phải truy xuất trực tiếp RAM video.

- Với board đơn sắc MDA trong kiểu hiện văn bản số 7, 4 KB RAM được tổ chức như 1 dãy (array) gồm 2000 từ nhớ kề nhau (mỗi từ là mã thuộc tính: ký tự) tạo nên 25 dòng, 80 cột. RAM video bắt đầu ở đoạn B0000h, trong đó ký tự góc trên cùng bên trái là từ thứ nhất trong RAM video. Như vậy mỗi dòng có 160 byte (A0h). Địa chỉ của từ nhớ ứng với ký tự ở dòng i, cột j (i = 0-24, j = 0-79) được tính theo công thức sau:

  Address (i,j) = B0000h +A0h*i +02h*j.

- Với board EGA, ở kiểu hiện văn bản từ 0 đến 3 mã ký tự được lưu trữ trong lớp nhớ 0 cùng với thuộc tính trong lớp 1 của RAM video. Mạch logic chuyển địa chỉ trên board thực hiện sự kết hợp nhất định nào đó sao cho tổ chức và cấu trúc của RAM video cũng như cách tính địa chỉ vẫn tương đồng với cách của CPU. Trong chế độ đồ họa từ 13 đến 16, RAM video bắt đầu từ địa chỉ đoạn A000h. Các điểm ảnh được xếp kề cận nhau trong bộ nhớ và mỗi điểm ảnh đòi hỏi 4 bit, các bit này được phân ra ở 4 lớp nhớ. Như vậy địa chỉ của 1 trong 4 bit này trên 1 điểm ảnh không chỉ gồm đoạn video và offset mà còn thêm vào số lớp nhớ nữa.

Để hiện 1 điểm ảnh với 1 trong 16 màu, không phải chỉ tính địa chỉ bit mà còn phải thâm nhập 4 lớp nhớ. Muốn vậy, phải dùng thanh ghi mặt nạ bản đồ (map mask register). Thanh ghi này được định địa chỉ qua cổng chỉ số 3C4h với địa chỉ 02h và có thể được ghi qua cổng số liệu 3C5h. Cấu trúc của thanh ghi mặt nạ bản đố như sau:

7	6	5	4	3	2	1	0
Res	Res	Res	Res	LY_3	LY_2	LY_1	LY_0

LY_3-LY_0: Thâm nhập ghi tới các lớp từ 0→3;
1 = cho phép; 0 = không cho phép

Res : Dự trữ

Ví dụ: Đặt bit 0 của byte ở địa chỉ A000:0000h cho lớ 0, 1, 3.

```
Mov AX, 0A000h      ; nạp đọan video vào AX
Mov ES, AX          ; truyền đọan video vào ES
Mov BX, 0000h       ; nạp offset 0000h vào BX
Out 3C4h, 02h       ; chỉ số 2 → thanh ghi mặt nạ bản đồ
Out  3C5h, 0Bh      ; ghi 0000 1011b vào thanh ghi mặt nạ bản đồ
                          (cho phép lớp 0, 1, 3)
Mov 3C5h, 0Bh       ; đặt bit 0 trong các lớp 0, 1 và 3
```

Để lưu trữ nội dung màn hình cần phải đọc các giá trị bit của 4 lớp khi dùng thanh ghi chọn bản đồ đọc (read map select register). Nó được định địa chỉ với chỉ số 04h qua cổng chỉ số 3CEh, và có thể được ghi qua cổng số liệu 3CFh. Cấu trúc của thanh ghi này:

7	6	5	4	3	2	1	0
res	res	res	res	res	res	LY_1	LY_0

LY_1-LY_0: cho phép thâm nhập đọc với:

00 = lớp 0
01 = lớp 1
10 = lớp 2
11 = lớp 3

res : dự trữ

Ví dụ: đọc byte ở địa chỉ A000:0000h cho lớp 2:

```
Mov AX, A000h       ; nạp đọan video vào AX
Mov ES, AX          ; truyền đọan video vào ES
Mov BX, 0000h       ; nạp offset vào BX
Out 3Ceh, 04h       ; chỉ số 4 → thanh ghi chọn bản đồ đọc
Out 3CFh, 02h       ; ghi 0000 0010b vào thanh ghi chọn bản đồ đọc
                          (cho phép lớp 2)
Mov  AL, [ES:BX]    ; nạp byte trong lớp 2 vào AL.
```

Chú ý rằng 4 bit tại 4 lớp đại diện cho 1 điểm ảnh nên trong kiểu hiện 16 EGA có độ phân giải cao nhất mỗi dòng cần 80byte (640 điểm ảnh / 8 điểm ảnh trên 1 byte); mỗi trang màn hình gồm 32 KB. Địa chỉ byte của điểm ảnh ở dòng i, cột j trang k (i=0-349, j=0-639, k=0-1) là:

$$\text{Address}(i,j,k) = A0000h + 8000h*k + 50h*j + \text{int}(i/8).$$

Với board VGA, các chế độ hiện văn bản từ 0 đến 3 và 7 cũng như các chế độ đồ họa từ 4 đến 6 và 13 đến 16 của CGA. EGA và MDA đều chạy được trên nó.

Trong chế độ văn bản, mã ký tự được lưu trữ trong lớp nhớ 0 cùng với thuộc tính trong lớp 1 của RAM video VGA. Quá trình chuyển hóa địa chỉ cũng giống như EGA nhưng khác ở chổ nó vẫn đảm bảo chế độ văn bản 7 với độ phân giải 720x400, ma trận điểm ảnh 9x16. Trong chế độ đồ họa $4 \div 6$ và $13 \div 19$, mọi tổ chức, cấu trúc cũng như cách tính

địa chỉ tương tự như CGA và EGA. VGA được tăng cường 3 kiểu hiện hình mới từ 17 đến 19.

Kiểu 17 tương thích với board đồ họa của máy PS/2 kiểu 30 là MCGA (multi colour graphics array). Các điểm ảnh chỉ gồm 1 bit (2 màu) được định vị chỉ trên lớp 0. Thí dụ, trong VGA kiểu 17 với 80 byte trên 1 dòng (640 điểm ảnh / 8 điểm ảnh trên 1 byte). Mỗi trang màn hình gồm 40 KB. Địa chỉ của byte ở dòng i, cột j (i= 0-479), j=0-639) như sau:

Address (i,j) = A0000h+50h*j+int (i/8)

Kiểu 18, 4 bit của điểm ảnh được phân trong 4 lớp nhớ như ở EGA. Trong kiểu VGA phân giải cao với 16 màu khác nhau, 80 byte trên 1 dòng (640 điểm ảnh / 8 điểm ảnh trên 1 byte), mỗi trang màn hình gồm 40 KB (A0000h byte); địa chỉ của mỗi byte ở dòng i, cột j (i=0-479; j = 0-639) bằng:

Address (i,j) = A0000h + 50h*j + int (i/8).

Kiểu 19 với 256 màu cho 1 điểm ảnh thì RAM video lại được tổ chức rất đơn giản như 1 dãy tuyến tính, trong đó 1 byte tương ứng với 1 điểm ảnh. Giá trị của byte phân định màu của điểm ảnh. Kiểu này đòi hỏi 320 byte (140h) trên 1 dòng (320 điểm ảnh / 1 điểm ảnh trên 1 byte). Một trang màn hình gồm 64 KB (10000h) nhưng chỉ có 64000 byte được sử dụng. Địa chỉ của điểm ảnh trong dòng i, cột j (i = 0-199, j=0-319) là:

Address (i,j) = A0000h + 140h*j + i

PHỤ LỤC CHƯƠNG 3

```
TITLE   DISPLAYING MOUSE POSITION
CURSOR   MACRO ROW,COLUMN
         MOV  AH,02H
         MOV  BH,00
         MOV  DH,ROW
         MOV  DL,COLUMN
         INT     10H
         ENDM
DISPLAY  MACRO STRING
         MOV  AH,09H
         MOV  DX,OFFSET STRING
         INT     21H
         ENDM
.MODEL SMALL
.STACK
.DATA
MESSAGE_1    DB 'PRESS ANY KEY$'
MESSAGE_2    DB 'THE MOUSE CURSOR IS LOCATED AT $'
POS_HO       DB ?,?, ' AND $'
POS_VE       DB ?,?,'$'
OLDVIDEO     DB ?            ;current video mode
NEWVIDEO     DB 0EH          ;new video mode
.CODE
MAIN  PROC
     MOV  AX,@DATA
     MOV  DS,AX
     MOV  AH,0FH          ;get current video mode
     INT  10H
     MOV  OLDVIDEO,AL     ;save it
     MOV  AX,0600H       ;clear screen
     MOV  BH,07
     MOV  CX,0
     MOV  DX,184FH
     INT  10H
     MOV  AH,00H         ;set new video mode
     MOV  AL,NEWVIDEO
     INT  10H
     MOV  AX,0                   ;initialize mouse
     INT  33H
     MOV  AX,01        ;show mouse cursor
     INT  33H
     CURSOR 20,20
     DISPLAY MESSAGE_1
AGAIN:
     MOV  AX,03H         ;get mouse location
     INT  33H
     MOV  AX,CX   ;get the hor. pixel position
     CALL CONVERT
     MOV  POS_HO,AL  ;save the LSD
     MOV  POS_HO+1,AH  ;save  the MSD
```

```
          MOV  AX,DX   ;get the vert. pixel position
          CALL CONVERT
          MOV  POS_VE,AL  ;save
          MOV  POS_VE+1,AH
          CURSOR 5,20
          DISPLAY MESSAGE_2
          DISPLAY POS_HO
          DISPLAY POS_VE
          MOV  AH,01               ;check for key press
          INT  16H
          JZ   AGAIN ;if no key press
          MOV  AH,02               ;hide mouse
          INT  33H
          MOV  AH,0             ;restore original video mode
          MOV  AL,OLDVIDEO         ;load original video mode
          MOV  AH,0 ;restore original video mode
          INT     10H
          MOV     AH,4CH               ;go back to DOS
          INT     21H
     MAIN     ENDP
     ;--------------------
     ;divide pixels position by 8 and convert to ASCII to make
it displayable
     ;ax=pixels position (it is in hex)
     ;on return ax= two ASCII digits
     CONVERT  PROC
     SHR  AX,1          ;divide
     SHR  AX,1          ;by 8
     SHR  AX,1          ;to get screen position by character
     MOV  BL,10
     MOV  AH,0
     DIV  BL
     ADD  AX,3030H      ;make it ASCII
     RET                   ;return with AX=two ASCII digits
     CONVERT  ENDP
     END     MAIN
     ---------------------------------------------------------------
```

```
     ;THIS PROGRAM WAITS FOR THE MOUSE PRESS COUNT AND
     ;DISPLAYS IT WHEN ANY KEY IS PRESSED.
     .MODEL   SMALL
     .STACK
     .DATA
     MESSAGE_1  DB 'PRESS LEFT BUTTON A NUMBER OF TIMES:LESS
THAN 99.$'
     MESSAGE_2   DB 'TO FIND OUT HOW MANY TIMES, PRESS ANY
KEY$'
     MESSAGE_3  DB 'YOU PRESSED IT $'
     P_COUNT    DB ?,?, ' TIMES $'
     MESSAGE_4  DB 'NOW PRESS ANY KEY TO GO BACK TO DOS$'
     OLDVIDEO   DB ?
     NEWVIDEO   DB 12H
```

```
.CODE
MAIN    PROC
 MOV  AX,@DATA
 MOV  DS,AX
 MOV  AH,0FH  ;get current video mode
 INT  10H
 MOV  OLDVIDEO,AL  ;save it
 MOV  AX,0600H  ;clear screen
 MOV  BH,07
 MOV  CX,0
 MOV  DX,184FH
 INT  10H
 MOV  AH,00H  ;set new video mode
 MOV  AL,NEWVIDEO
 INT  10H
 MOV  AX,0   ;initialize mouse
 INT  33H
 MOV  AX,01  ;show mouse cursor
 INT  33H
 CURSOR 2,1
 DISPLAY MESSAGE_1
 CURSOR 4,1
 DISPLAY MESSAGE_2
 MOV  AH,07  ;wait for key press
 INT  21H
 MOV  AX,05H  ;get mouse press count
 MOV  BX,0  ;check press count for left button
 INT  33H
 MOV  AX,BX  ;BX=button press count
 MOV  BL,10
 DIV  BL
 ADD  AX,3030H
 MOV  P_COUNT,AL  ;save the number
 MOV  P_COUNT+1,AH
 CURSOR 10,2
 DISPLAY MESSAGE_3
 DISPLAY P_COUNT
 CURSOR 20,2
 DISPLAY MESSAGE_4
 MOV  AH,07          ;wait for a key press to get out
 INT  21H
 MOV  AH,02          ;hide mouse
 INT  33H
 MOV  AH,0           ;restore original video mode
 MOV  AL,OLDVIDEO  ;load original vide mode
 INT  10H
 MOV  AH,4CH         ;go back to DOS
 INT  21H
MAIN  ENDP
 END MAIN
;-------------------------------------------------

.MODEL SMALL
```

```
.STACK 100h
.DATA
mask_mon DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
         DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
         DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
         DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
mask_p   DB   80h,0,0E0h,0,0F8h,0,0FEh,0
         DB   0D8h,0,0Ch,0,6,0,3,0
         DB   0,0,0,0,0,0,0,0
         DB   0,0,0,0,0,0,0,0
.CODE
     main PROC
          mov ax,@data
          mov ds,ax
          mov es,ax

          mov ah,0
          mov al,6
          int 10h

          mov ax,0
          int 33h

          mov ax,1
          int 33h

          mov ah,08h
          int 21h

          mov ax,9
          mov bx,0
          mov cx,0
          lea dx,mask_mon
          int 33h

          mov ah,08h
          int 21h

          mov ah,4Ch
          int 21h
     main ENDP
     END  main
```

1000000000000000
1110000000000000
1111100000000000
1111111000000000
1101100000000000
0000110000000000
0000011000000000
0000001100000000

0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000

Keyboard Scan Codes: Set 1

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1E	9E		9	0A	8A		[	1A	9A
B	30	B0		`	29	89		INSERT	E0,52	E0,D2
C	2E	AE		-	0C	8C		HOME	E0,47	E0,97
D	20	A0		=	0D	8D		PG UP	E0,49	E0,C9
E	12	92		\	2B	AB		DELETE	E0,53	E0,D3
F	21	A1		BKSP	0E	8E		END	E0,4F	E0,CF
G	22	A2		SPACE	39	B9		PG DN	E0,51	E0,D1
H	23	A3		TAB	0F	8F		U ARROW	E0,48	E0,C8
I	17	97		CAPS	3A	BA		L ARROW	E0,4B	E0,CB
J	24	A4		L SHFT	2A	AA		D ARROW	E0,50	E0,D0
K	25	A5		L CTRL	1D	9D		R ARROW	E0,4D	E0,CD
L	26	A6		L GUI	E0,5B	E0,DB		NUM	45	C5
M	32	B2		L ALT	38	B8		KP /	E0,35	E0,B5
N	31	B1		R SHFT	36	B6		KP *	37	B7
O	18	98		R CTRL	E0,1D	E0,9D		KP -	4A	CA
P	19	99		R GUI	E0,5C	E0,DC		KP +	4E	CE
Q	10	19		R ALT	E0,38	E0,B8		KP EN	E0,1C	E0,9C
R	13	93		APPS	E0,5D	E0,DD		KP .	53	D3
S	1F	9F		ENTER	1C	9C		KP 0	52	D2
T	14	94		ESC	01	81		KP 1	4F	CF
U	16	96		F1	3B	BB		KP 2	50	D0
V	2F	AF		F2	3C	BC		KP 3	51	D1
W	11	91		F3	3D	BD		KP 4	4B	CB
X	2D	AD		F4	3E	BE		KP 5	4C	CC
Y	15	95		F5	3F	BF		KP 6	4D	CD
Z	2C	AC		F6	40	C0		KP 7	47	C7

0	0B	8B		F7	41	C1		KP 8	48	C8
1	02	82		F8	42	C2		KP 9	49	C9
2	03	83		F9	43	C3		]	1B	9B
3	04	84		F10	44	C4		;	27	A7
4	05	85		F11	57	D7		'	28	A8
5	06	86		F12	58	D8		,	33	B3
6	07	87		PRNT SCRN	E0,2A, E0,37	E0,B7, E0,AA		.	34	B4
7	08	88		SCROLL	46	C6		/	35	B5
8	09	89		PAUSE	E1,1D,45 E1,9D,C5	-NONE-				

Keyboard Scan Codes: Set 2

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B

X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

AT Keyboard Scan Codes (Set 3)

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	F0,54
B	32	F0,32		`	0E	F0,0E		INSERT	67	F0,67
C	21	F0,21		-	4E	F0,4E		HOME	6E	F0,6E
D	23	F0,23		=	55	F0,55		PG UP	6F	F0,6F
E	24	F0,24		\	5C	F0,5C		DELETE	64	F0,64
F	2B	F0,2B		BKSP	66	F0,66		END	65	F0,65
G	34	F0,34		SPACE	29	F0,29		PG DN	6D	F0,6D
H	33	F0,33		TAB	0D	F0,0D		U ARROW	63	F0,63
I	43	F0,48		CAPS	14	F0,14		L ARROW	61	F0,61
J	3B	F0,3B		L SHFT	12	F0,12		D ARROW	60	F0,60
K	42	F0,42		L CTRL	11	F0,11		R ARROW	6A	F0,6A
L	4B	F0,4B		L WIN	8B	F0,8B		NUM	76	F0,76
M	3A	F0,3A		L ALT	19	F0,19		KP /	4A	F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7E	F0,7E
O	44	F0,44		R CTRL	58	F0,58		KP -	4E	F0,4E
P	4D	F0,4D		R WIN	8C	F0,8C		KP +	7C	F0,7C
Q	15	F0,15		R ALT	39	F0,39		KP EN	79	F0,79
R	2D	F0,2D		APPS	8D	F0,8D		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	08	F0,08		KP 1	69	F0,69

U	3C	F0,3C		F1	07	F0,07		KP 2	72	F0,72
V	2A	F0,2A		F2	0F	F0,0F		KP 3	7A	F0,7A
W	1D	F0,1D		F3	17	F0,17		KP 4	6B	F0,6B
X	22	F0,22		F4	1F	F0,1F		KP 5	73	F0,73
Y	35	F0,35		F5	27	F0,27		KP 6	74	F0,74
Z	1A	F0,1A		F6	2F	F0,2F		KP 7	6C	F0,6C
0	45	F0,45		F7	37	F0,37		KP 8	75	F0,75
1	16	F0,16		F8	3F	F0,3F		KP 9	7D	F0,7D
2	1E	F0,1E		F9	47	F0,47		]	5B	F0,5B
3	26	F0,26		F10	4F	F0,4F		;	4C	F0,4C
4	25	F0,25		F11	56	F0,56		'	52	F0,52
5	2E	F0,2E		F12	5E	F0,5E		,	41	F0,41
6	36	F0,36		PRNT SCRN	57	F0,57		.	49	F0,49
	3D	F0,3D		SCROLL	5F	F0,5F		/	4A	F0,4A
8	3E	F0,3E		PAUSE	62	F0,62				

Chương 4
GIAO TIẾP CỔNG NỐI TIẾP

1. Cấu trúc cổng nối tiếp

Cổng nối tiếp được sử dụng để truyền dữ liệu hai chiều giữa máy tính và ngoại vi, có các ưu điểm sau:

- Khoảng cách truyền xa hơn truyền song song.
- Số dây kết nối ít.
- Có thể truyền không dây dùng hồng ngoại.
- Có thể ghép nối với vi điều khiển hay PLC (Programmable Logic Device).
- Cho phép nối mạng.
- Có thể tháo lắp thiết bị trong lúc máy tính đang làm việc.
- Có thể cung cấp nguồn cho các mạch điện đơn giản

Các thiết bị ghép nối chia thành 2 loại: DTE (Data Terminal Equipment) và DCE (Data Communication Equipment). DCE là các thiết bị trung gian như MODEM còn DTE là các thiết bị tiếp nhận hay truyền dữ liệu như máy tính, PLC, vi điều khiển, … Việc trao đổi tín hiệu thông thường qua 2 chân RxD (nhận) và TxD (truyền). Các tín hiệu còn lại có chức năng hỗ trợ để thiết lập và điều khiển quá trình truyền, được gọi là các tín hiệu bắt tay (handshake). Ưu điểm của quá trình truyền dùng tín hiệu bắt tay là có thể kiểm soát đường truyền.

Tín hiệu truyền theo chuẩn RS-232 của EIA (Electronics Industry Associations). Chuẩn RS-232 quy định mức logic 1 ứng với điện áp từ -3V đến -25V (mark), mức logic 0 ứng với điện áp từ 3V đến 25V (space) và có khả năng cung cấp dòng từ 10 mA đến 20 mA. Ngoài ra, tất cả các ngõ ra đều có đặc tính chống chập mạch.

Chuẩn RS-232 cho phép truyền tín hiệu với tốc độ đến 20.000 bps nhưng nếu cáp truyền đủ ngắn có thể lên đến 115.200 bps.

Các phương thức nối giữa DTE và DCE:

- Đơn công (simplex connection): dữ liệu chỉ được truyền theo 1 hướng.

- Bán song công (half-duplex): dữ liệu truyền theo 2 hướng, nhưng mỗi thời điểm chỉ được truyền theo 1 hướng.

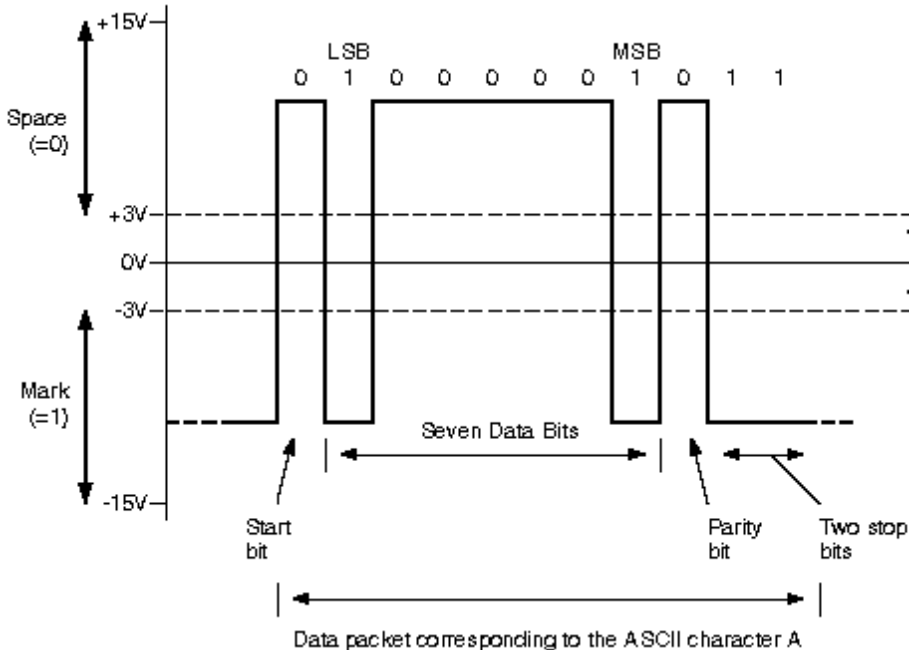
- Song công (full-duplex): số liệu được truyền đồng thời theo 2 hướng.

Định dạng của khung truyền dữ liệu theo chuẩn RS-232 như sau:

	Start	D0	D1	D2	D3	D4	D5	D6	D7	P	Stop
	0										1

Khi không truyền dữ liệu, đường truyền sẽ ở trạng thái mark (điện áp -10V). Khi bắt đầu truyền, DTE sẽ đưa ra xung Start (space: 10V) và sau đó lần lượt truyền từ D0 đến D7

và Parity, cuối cùng là xung Stop (mark: -10V) để khôi phục trạng thái đường truyền. Dạng tín hiệu truyền mô tả như sau (truyền ký tự A):



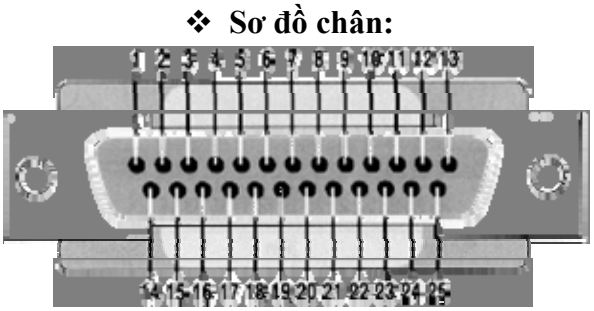
Hình 4.1 – Tín hiệu truyền của ký tự 'A'

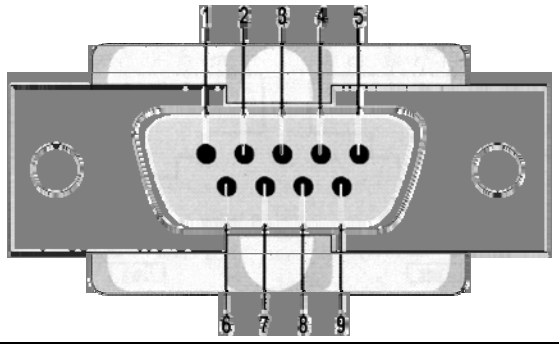
Các đặc tính kỹ thuật của chuẩn RS-232 như sau:

Chiều dài cable cực đại	15m
Tốc độ dữ liệu cực đại	20 Kbps
Điện áp ngõ ra cực đại	± 25V
Điện áp ngõ ra có tải	± 5V đến ± 15V
Trở kháng tải	3K đến 7K
Điện áp ngõ vào	± 15V
Độ nhạy ngõ vào	± 3V
Trở kháng ngõ vào	3K đến 7K

Các tốc độ truyền dữ liệu thông dụng trong cổng nối tiếp là: 1200 bps, 4800 bps, 9600 bps và 19200 bps.

❖ **Sơ đồ chân:**





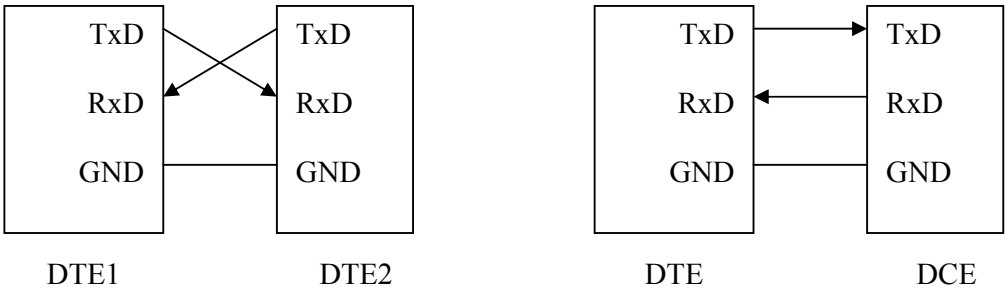
Hình 4.2 – Sơ đồ chân cổng nối tiếp

Cổng COM có hai dạng: đầu nối DB25 (25 chân) và đầu nối DB9 (9 chân) mô tả như hình 4.2. Ý nghĩa của các chân mô tả như sau:

D25	D9	Tín hiệu	Hướng truyền	Mô tả
1	-	-	-	Protected ground: nối đất bảo vệ
2	3	TxD	DTE→DCE	Transmitted data: dữ liệu truyền
3	2	RxD	DCE→DTE	Received data: dữ liệu nhận
4	7	RTS	DTE→DCE	Request to send: DTE yêu cầu truyền dữ liệu
5	8	CTS	DCE→DTE	Clear to send: DCE sẵn sàng nhận dữ liệu
6	6	DSR	DCE→DTE	Data set ready: DCE sẵn sàng làm việc
7	5	GND	-	Ground: nối đất (0V)
8	1	DCD	DCE→DTE	Data carier detect: DCE phát hiện sóng mang
20	4	DTR	DTE→DCE	Data terminal ready: DTE sẵn sàng làm việc
22	9	RI	DCE→DTE	Ring indicator: báo chuông
23	-	DSRD	DCE→DTE	Data signal rate detector: dò tốc độ truyền
24	-	TSET	DTE→DCE	Transmit Signal Element Timing: tín hiệu định thời truyền đi từ DTE
15	-	TSET	DCE→DTE	Transmitter Signal Element Timing: tín hiệu định thời truyền từ DCE để truyền dữ liệu
17	-	RSET	DCE→DTE	Receiver Signal Element Timing: tín hiệu định thời truyền từ DCE để truyền dữ liệu
18	-	LL		Local Loopback: kiểm tra cổng
21	-	RL	DCE→DTE	Remote Loopback: Tạo ra bởi DCE khi tín hiệu nhận từ DCE lỗi
14	-	STxD	DTE→DCE	Secondary Transmitted Data
16	-	SRxD	DCE→DTE	Secondary Received Data
19	-	SRTS	DTE→DCE	Secondary Request To Send
13	-	SCTS	DCE→DTE	Secondary Clear To Send
12	-	SDSRD	DCE→DTE	Secondary Received Line Signal Detector
25	-	TM		Test Mode
9	-			Dành riêng cho chế độ test
10	-			Dành riêng cho chế độ test
11				Không dùng

2. Truyền thông giữa hai nút

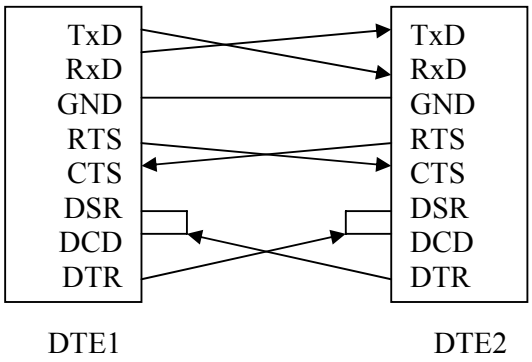
Các sơ đồ khi kết nối dùng cổng nối tiếp:



Hình 4.3 – Kết nối đơn giản trong truyền thông nối tiếp

Khi thực hiện kết nối như trên, quá trình truyền phải bảo đảm tốc độ ở đầu phát và thu giống nhau. Khi có dữ liệu đến DTE, dữ liệu này sẽ được đưa vào bộ đệm và tạo ngắt.

Ngoài ra, khi thực hiện kết nối giữa hai DTE, ta còn dùng sơ đồ sau:



Hình 4.4 – Kết nối trong truyền thông nối tiếp dùng tín hiệu bắt tay

Khi DTE1 cần truyền dữ liệu thì cho DTR tích cực → tác động lên DSR của DTE2 cho biết sẵn sàng nhận dữ liệu và cho biết đã nhận được sóng mang của MODEM (ảo). Sau đó, DTE1 tích cực chân RTS để tác động đến chân CTS của DTE2 cho biết DTE1 có thể nhận dữ liệu. Khi thực hiện kết nối giữa DTE và DCE, do tốc độ truyền khác nhau nên phải thực hiện điều khiển lưu lượng. Quá trinh điều khiển này có thể thực hiện bằng phần mềm hay phần cứng. Quá trình điều khiển bằng phần mềm thực hiện bằng hai ký tự Xon và Xoff. Ký tự Xon được DCE gởi đi khi rảnh (có thể nhận dữ liệu). Nếu DCE bận thì sẽ gởi ký tự Xoff. Quá trình điều khiển bằng phần cứng dùng hai chân RTS và CTS. Nếu DTE muốn truyền dữ liệu thì sẽ gởi RTS để yêu cầu truyền, DCE nếu có khả năng nhận dữ liệu (đang rảnh) thì gởi lại CTS.

3. Truy xuất trực tiếp thông qua cổng

Các cổng nối tiếp trong máy tính được đánh số là COM1, COM2, COM3, COM4 với các địa chỉ như sau:

Tên	Địa chỉ	Ngắt	Vị trí chứa địa chỉ
COM1	3F8h	4	0000h:0400h
COM2	2F8h	3	0000h:0402h
COM3	3E8h	4	0000h:0404h
COM4	2E8h	3	0000h:0406h

Giao tiếp nối tiếp trong máy tính sử dụng vi mạch UART với các thanh ghi cho trong bảng sau:

Offset	DLAB	R/W	Tên	Chức năng
0	0	W	THR	Transmitter Holding Register (đệm truyền)
	0	R	RBR	Receiver Buffer Register (đệm thu)
	1	R/W	BRDL	Baud Rate Divisor Latch (số chia byte thấp)
1	0	R/W	IER	Interrupt Enable Register (cho phép ngắt)
	1	R/W	BRDH	Số chia byte cao
2		R	IIR	Interrupt Identification Register (nhận dạng ngắt)
		W	FCR	FIFO Control Register
3		R/W	LCR	Line Control Register (điều khiển đường dây)
4		R/W	MCR	Modem Control Register (điều khiển MODEM)
5		R	LSR	Line Status Register (trạng thái đường dây)
6		R	MSR	Modem Status Register (trạng thái MODEM)
7		R/W		Scratch Register (thanh ghi tạm)

Các thanh ghi này có thể truy xuất trực tiếp kết hợp với địa chỉ cổng (ví dụ như thanh ghi cho phép ngắt của COM1 có địa chỉ là $BA_{COM1} + 1 = 3F9h$.

❖ IIR (Interrupt Identification):

IIR xác định mức ưu tiên và nguồn gốc của yêu cầu ngắt mà UART đang chờ phục vụ. Khi cần xử lý ngắt, CPU thực hiện đọc các bit tương ứng để xác định nguồn gốc của ngắt. Định dạng của IIR như sau:

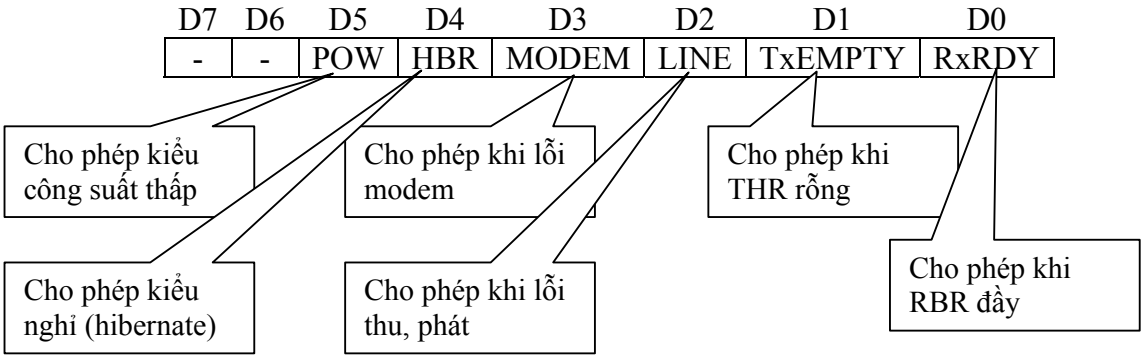
D7 D6	D5	D4	D3	D2 D1	D0
00: không có FIFO 11: cho phép FIFO	Cho phép FIFO 64 byte (trong 16750)	-	1: ngắt time-out (trong 16550)	Xác định nguồn gốc ngắt	0: có ngắt 1: không ngắt

D2	D1	Ưu tiên	Tên	Nguồn	D2 – D0 bị xoá khi
0	0	4	Đường truyền	Lỗi khung, thu đè, lỗi parity, gián đoạn khi thu	Đọc LSR
0	1	3	Đệm thu	Đệm thu đầy	Đọc RBR
1	0	2	Đệm phát	Đệm phát rỗng	Đọc IIR, ghi THR
1	1	1	Modem	CTS, DSR, RI, RLSD	Đọc MSR

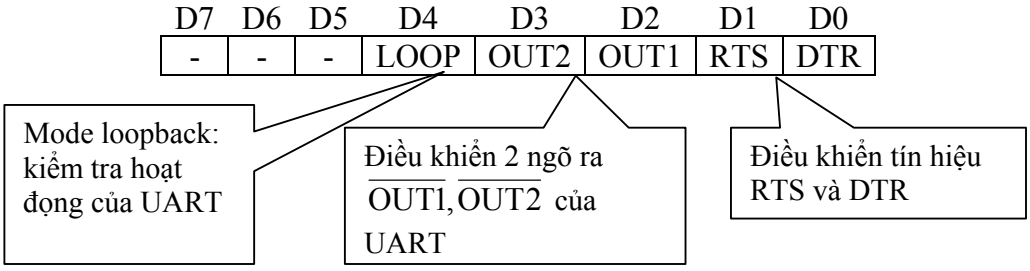
(mức 1 ưu tiên cao nhất)

❖ **IER (Interrupt Enable Register):**

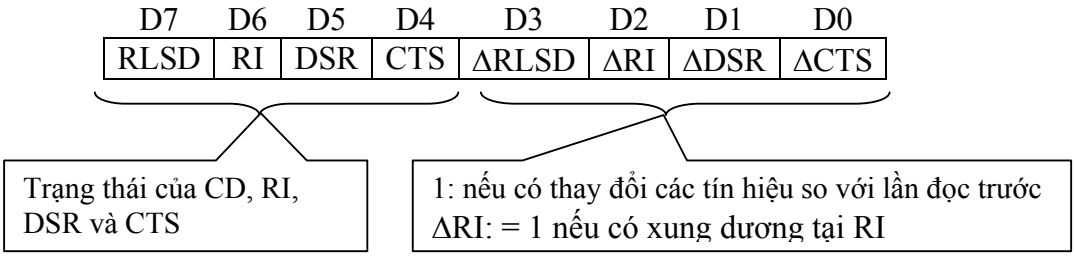
IER cho phép hay cấm các nguyên nhân ngắt khác nhau (1: cho phép, 0: cầm ngắt)



❖ **MCR (Modem Control Register):**



❖ **MSR (Modem Status Register):**



❖ **LSR (Line Status Register):**

D7	D6	D5	D4	D3	D2	D1	D0
FIE	TSRE	THRE	BI	FE	PE	OE	RxDR

FIE: FIFO Error – sai trong FIFO

TSRE: Transmitter Shift Register Empty – thanh ghi dịch rỗng (=1 khi đã phát 1 ký tự và bị xoá khi có 1 ký tự chuyển đến từ THR.

THRE: Transmitter Holding Register Empty (=1 khi có 1 ký tự đã chuyển từ THR – TSR và bị xoá khi CPU đưa ký tự tới THR).

BI: Break Interrupt (=1 khicó sự gián đoạn khi truyền, nghĩa là tồn tại mức logic 0 trong khoảng thời gian dài hơn khoảng thời gian truyền 1 byte và bị xoá khi CPU đọc LSR)

FE: Frame Error (=1 khi có lỗi khung truyền và bị xoá khi CPU đọc LSR)

PE: Parity Error (=1 khi có lỗi parity và bị xoá khi CPU đọc LSR)

OE: Overrun Error (=1 khi có lỗi thu đè, nghĩa là CPU không đọc kịp dữ liệu làm cho quá trình ghi chồng lên RBR xảy ra và bị xoá khi CPU đọc LSR)

RxDR: Receiver Data Ready (=1 khi đã nhận 1 ký tự và đưa vào RBR và bị xoá khi CPU đọc RBR).

❖ **LCR (Line Control Register):**

D7	D6	D5	D4	D3	D2	D1	D0
DLAB	SBCB	PS2	PS1	PS0	STB	WLS1	WLS0

DLAB (Divisor Latch Access Bit) = 0: truy xuất RBR, THR, IER, = 1 cho phép đặt bộ chia tần trong UART để cho phép đạt tốc độ truyền mong muốn.

UART dùng dao động thạch anh với tần số 1.8432 MHz đưa qua bộ chia 16 thành tần số 115,200 Hz. Khi đó, tuỳ theo giá trị trong BRDL và BRDH, ta sẽ có tốc độ mong muốn. Ví dụ như đường truyền có tốc độ truyền 2,400 bps có giá trị chia 115,200 / 2,400 = 48d = 0030h → BRDL = 30h, BRDH = 00h.

Một số giá trị thông dụng xác định tốc độ truyền cho như sau:

Tốc độ (bps)	**BRDH**	**BRDL**
1,200	00h	60h
2,400	00h	30h
4,800	00h	18h
9,600	00h	0Ch
19,200	00h	06h
38,400	00h	03h
57,600	00h	02h
115,200	00h	01h

SBCB (Set Break Control Bit) =1: cho phép truyền tín hiệu Break (=0) trong khoảng thời gian lớn hơn một khung

PS (Parity Select):

PS2	**PS1**	**PS0**	**Mô tả**
X	X	0	Không kiểm tra
0	0	1	Kiểm tra lẻ
0	1	1	Kiểm tra chẵn
1	0	1	Parity là mark
1	1	1	Parity là space

STB (Stop Bit) = 0: 1 bit stop, =1: 1.5 bit stop (khi dùng 5 bit dữ liệu) hay 2 bit stop (khi dùng 6, 7, 8 bit dữ liệu).

WLS (Word Length Select):

WLS1	WLS0	Độ dài dữ liệu
0	0	5 bit
0	1	6 bit
1	0	7 bit
1	1	8 bit

Một ví dụ khi lập trình trực tiếp trên cổng như sau:

```
.MODEL SMALL
.STACK 100h
.DATA
     Com1      EQU 3F8h
     Com_int   EQU 08h
     Buffer    DB   251 DUP(?)
     Bufferin  DB   0
     Bufferout      DB   0
     Char      DB   ?
     Seg_com   DW   ?    ; Vector ng•t c•
     Off_com   DW   ?
     Mask_int  DB   ?
     Msg       DB   'Press any key to exit$'
.CODE
Main PROC
     MOV  AX,@DATA
     MOV  DS,AX

     MOV  AH,35h
     MOV  AL,Com_int
     INT  21h
     MOV  Seg_com,ES     ; L•u vector ng•t c•
     MOV  Off_com,BX

     PUSH DS
     MOV  BX,CS
     MOV  DS,BX
     LEA  DX,Com_ISR
     MOV  AH,35h         ;Gán vector ng•t m•i
     MOV  AL,Com_int
     INT  21h
     POP  DS

     MOV  DX,Com1+3      ; ••a ch• LCR
     MOV  AL,80h         ; Set DLAB = 1 cho phép ••nh t•c
     OUT  DX,AL          ; •• truy•n d• li•u
```

```
        MOV  DX,Com1        ; G•i byte th•p
        MOV  AL,0Ch
        OUT  DX,AL

        MOV  DX,Com1+1
        MOV  AL,00h         ; G•i byte cao → 000Ch: xác ••nh
        OUT  DX,AL          ; t•c •• truy•n 9600bps

        MOV  DX,Com1+3      ; LCR = 0000 0011B
        MOV  AL,03h         ; DLAB = 0, SBCB = 0 → c•m Break
        OUT  DX,AL          ; PS = 000 → no parity
                            ; STB = 0 → 1 stop bit
                            ; WLS = 11 → 8 bit d• li•u

        MOV  DX,Com1+4      ; Tác ••ng ••n DTR và RTS
        MOV  AL,03h         ; MCR = 0000 0011b → DTR=RTS = 1
        OUT  DX,AL          ; → ngõ DTR và RTS c•a c•ng n•i
                            ; ti•p = 0

        MOV  DX,21h         ; Ki•m tra tr•ng thái ng•t
        IN   AL,DX          ; D7 – D0 xác ••nh các IRQi
        MOV  Mask_int,AL    ; =0: cho phép, =1: c•m

        AND  AL,0EFh        ; = 1110 1111b → cho phép IRQ4
        OUT  DX,AL          ; → cho phép COM1

        MOV  AL,01h         ; IER = 0000 0001b → cho phép
        MOV  DX,Com1+1      ; ng•t khi RBR ••y
        OUT  DX,AL

        MOV  AH,09h
        LEA  Dx,Msg
        INT  21h

Lap:
        MOV  AH,0Bh
        INT  21h
        CMP  AL,0FFh
        JE   Exit

        MOV  AL,bufferin
        CMP  AL,bufferout
        JE   Lap
        MOV  AL,buffer[bufferout]
        MOV  char,AL
        INC  bufferout
        MOV  AL,bufferout
        CMP  AL,251
        JNE  Next
        MOV  bufferout,0
Next:
```

```
        MOV DL,char         ; Xu•t giá tr• ra màn hình
        MOV  AH,02h
        INT  21h

        MOV  AL,char        ; Xu•t ra c•ng n•i ti•p
        MOV  DX,Com1
        OUT  DX,AL
        JMP  Lap

Exit:
        MOV  AL,Mask_int
        OUT  21h,AL         ; Khôi ph•c tr•ng thái ng•t

        MOV  DX,Off_com
        MOV  BX,Seg_com
        MOV  DS,BX
        MOV  AH,35h         ;Khôi ph•c vector ng•t
        MOV  AL,Com_int
        INT  21h


        MOV  AH,4Ch
        INT  21h
Main ENDP

Com_ISR  PROC
        MOV  DX,Com1+5      ; ••c n•i dung LSR
        IN   AL,DX
        AND  AL,1           ; N•u D0 = 1 thì có d• li•u
        JZ   exit_ISR

        MOV  DX,Com1
        IN   AL,DX
        MOV  buffer[bufferin],AL
        INC  bufferin
        MOV  AL,bufferin
        CMP  AL,251
        JNE  Exit_ISR
        MOV  bufferin,0
Exit_ISR:
        MOV  AL,20h         ; Báo cho PIC k•t thúc ng•t
        OUT  20h,AL
        IRET
Com_ISR  ENDP
END Main
```

4. Truyền thông nối tiếp dùng ActiveX

4.1. Mô tả

Việc truyền thông nối tiếp trên Windows được thực hiện thông qua một ActiveX có sẵn là Microsoft Comm Control.. ActiveX này dược lưu trữ trong file MSCOMM32.OCX. Quá trình này có hai khả năng thực hiện điều khiển trao đổi thông tin:

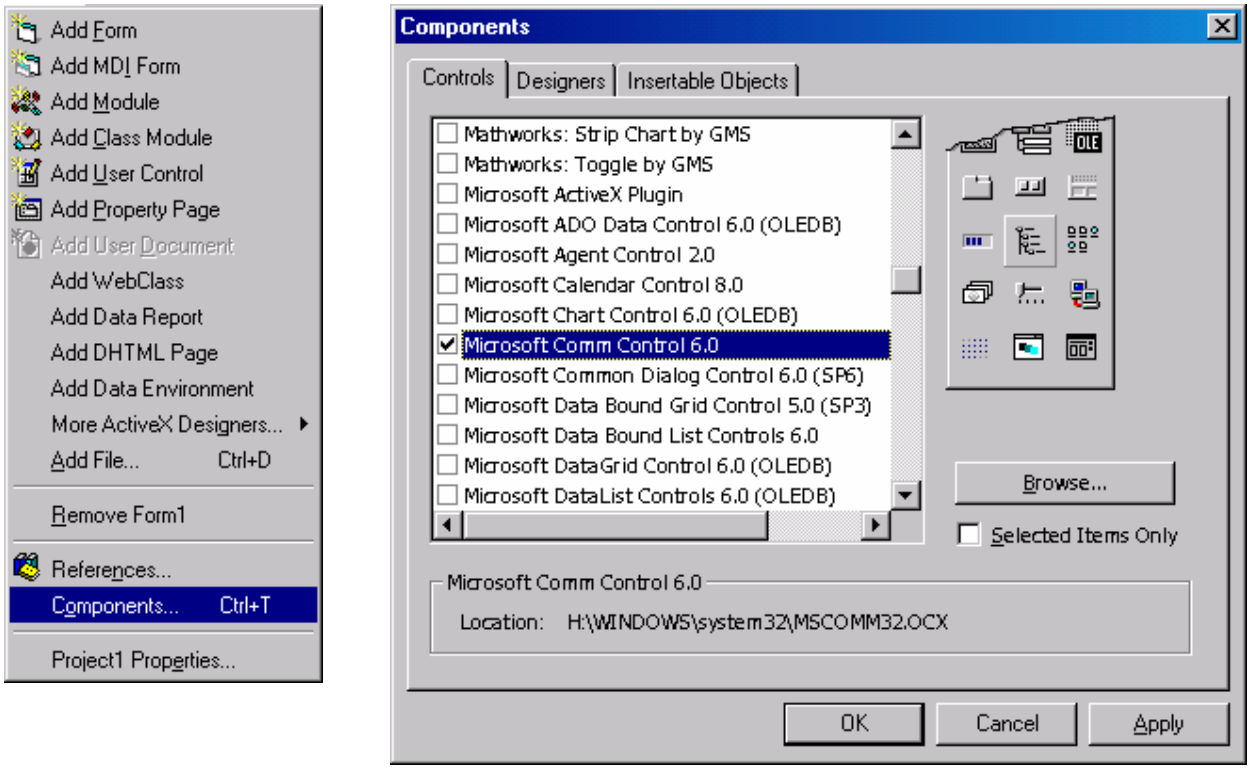
- **Điều khiển sự kiện:**

Truyền thông điều khiển sự kiện là phương pháp tốt nhất trong quá trình điều khiển việc trao đổi thông tin. Quá trình điều khiển thực hiện thông qua sự kiện OnComm.

- **Hỏi vòng:**

Quá trinh điều khiển bằng phương pháp hỏi vòng thực hiện thông qua kiểm tra các giá trị của thuộc tính CommEvent sau một chu kỳ nào đó để xác định xem có sự kiện nào xảy ra hay không. Thông thường phương pháp này sử dụng cho các chương trình nhỏ.

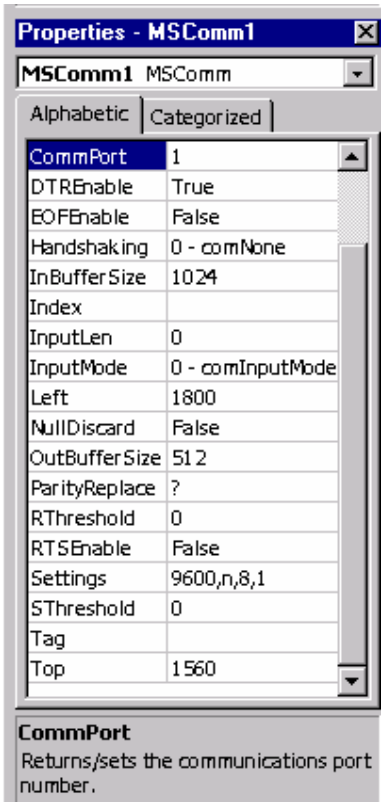
ActiveX MsComm được bổ sung vào một Visual Basic Project thông qua menu **Project > Components**:



Hình 4.5 – Bổ sung đối tượng MsComm vào VBP

Biểu tượng của MsComm: và các thuộc tính cơ bản mô tả như sau:

Thuộc tính	Mô tả
CommPort	Số thứ tự cổng truyền thông
Input	Nhận ký tự từ bộ đệm
Output	Xuất ký tự ra cổng nối tiếp
PortOpen	Mở / đóng cổng
Settings	Xác định các tham số truyền



Hình 4.6 – Các thuộc tính của đối tượng MSComm

4.2. Các thuộc tính

❖ **Settings:**

Xác định các tham số cho cổng nối tiếp. Cú pháp:

```
MSComm1.Settings = ParamString
```

MSComm1: tên đối tượng

ParamString: là một chuỗi có dạng như sau: "BBBB,P,D,S"

BBBB: tốc độ truyền dữ liệu (bps) trong đó các giá trị hợp lệ là:

110	2400	38400
300	9600 (măc định)	56000
600	14400	188000
1200	19200	256000

P: kiểm tra chẵn lẻ, với các giá trị:

Giá trị	Mô tả
O	Odd (kiểm tra lẻ)
E	Even (kiểm tra chẵn)
M	Mark (luôn bằng 1)
S	Space (luôn bằng 0)
N	Không kiểm tra

D: số bit dữ liệu (4, 5, 6, 7 hay 8), mặc định là 8 bit

S: số bit stop (1, 1.5, 2)

VD:

`MSComm1.Settings = "9600,O,8,1"` sẽ xác định tốc độ truyền 9600bps, kiểm tra parity chẵn với 1 bit stop và 8 bit dữ liệu.

❖ **CommPort:**

Xác định số thứ tự của cổng truyền thông, cú pháp:

`MSComm1.CommPort = PortNumber`

PortNumber là giá trị nằm trong khoảng từ 1 → 99, mặc định là 1.

VD:

`MSComm1.CommPort = 1` xác định sử dụng COM1

❖ **PortOpen:**

Đặt trạng thái hay kiểm tra trạng thái đóng / mở của cổng nối tiếp. Nếu dùng thuộc tính này để mở cổng nối tiếp thì phải sử dụng trước 2 thuộc tính Settings và CommPort. Cú pháp:

`MSComm1.PortOpen = True | False`

Giá trị xác định là True sẽ thực hiện mở cổng và False để đóng cổng đồng thời xoá nội dung của các bộ đệm truyền, nhận.

VD: Mở cổng COM1 với tốc độ truyền 9600 bps

```
MSComm1.Settings = "9600,N,8,1"
MSComm1.CommPort = 1
MSComm1.PortOpen = True
```

❖ **Các thuộc tính nhận dữ liệu:**

Input: nhận một chuỗi ký tự và xoá khỏi bộ đệm. Cú pháp:

`InputString = MSComm1.Input`

Thuộc tính này kết hợp với InputLen để xác định số ký tự đọc vào. Nếu InputLen = 0 thì sẽ đọc toàn bộ dữ liệu có trong bộ đệm.

InBufferCount: số ký tự có trong bộ đệm nhận. Cú pháp:

`Count = MSComm1.InBufferCount`

Thuộc tính này cùng dược dùng để xoá bộ đệm nhận bằng cách gán giá trị 0.

`MSComm1.InBufferCount = 0`

InBufferSize: đặt và xác định kích thước bộ đệm nhận (tính bằng byte). Cú pháp:

`MSComm1.InBufferCount = NumByte`

Giá trị măc định là 1024 byte. Kích thước bộ đệm này phải đủ lớn để tránh tình trạng mất dữ liệu.

VD: Đọc toàn bộ nội dung trong bộ đệm nhận nếu có dữ liệu

```
MSComm1.InputLen = 0
If MSComm1.InBufferCount <> 0 Then
     InputString = MSComm1.Input
End If
```

❖ **Các thuộc tính xuất dữ liệu:**

Bao gồm các thuộc tính **Output**, **OutBufferCount** và **OutBufferSize**, chức năng của các thuộc tính này giống như các thuộc tính nhập.

❖ **CDTimeout:**

Đặt và xác định khoảng thời gian lớn nhất (tính bằng ms) từ lúc phát hiện sóng mang cho đến lúc có dữ liệu. Nếu quá khoảng thời gian này mà vẫn chưa có dữ liệu thì sẽ gán thuộc tính CommEvent là CDTO (Carrier Detect Timeout Error) và tạo sự kiện OnComm. Cú pháp:

```
MSComm1.CDTimeout = NumTime
```

❖ **DSRTimeout:**

Xác định thời gian chờ tín hiệu DSR trước khi xảy ra sự kiện OnComm.

❖ **CTSTimeout:**

Đặt và xác định khoảng thời gian lớn nhất (tính bằng ms) đợi tín hiệu CTS trước khi đặt thuộc tính CommEvent là CTSTO và tạo sự kiện OnComm. Cú pháp:

```
MSComm1.CTSTimeout = NumTime
```

❖ **CTSHolding:**

Xác định đã có tín hiệu CTS hay chưa, tín hiệu này dùng cho quá trình bắt tay bằng phần cứng (cho biết DCE sẵn sàng nhận dữ liệu), trả về giá trị True hay False.

❖ **DSRHolding:**

Xác định trạng thái DSR (báo hiệu sự tồn tại của DCE), trả về giá trị True hay False.

❖ **CDHolding:**

Xác định trạng thái CD, trả về giá trị True hay False.

❖ **DTREnable:**

Đặt hay xoá tín hiệu DTR để báo sự tồn tại của DTE. Cú pháp:

```
MSComm1.DTREnable = True | False
```

❖ **RTSEnable:**

Đặt hay xoá tín hiệu RTS để yêu cầu truyền dữ liệu đến DTE. Cú pháp:

```
MSComm1.RTSEnable = True | False
```

❖ **NullDiscard:**

Cho phép nhận các ký tự NULL (rỗng) hay không (= True: cấm). Cú pháp:

```
MSComm1.NullDiscard = True | False
```

❖ **SThreshold:**

Số byte trong bộ đệm truyền làm phát sinh sự kiện OnComm. Nếu giá trị này bằng 0 thì sẽ không tạo sự kiện OnComm. Cú pháp:

```
MSComm1.SThreshold = NumChar
```

❖ **HandShaking:**

Chọn giao thức bắt tay khi thực hiện truyền dữ liệu. Cú pháp:

```
MSComm1.HandShaking = Protocol
```

Các giao thức truyền bao gồm:

Protocol	Giá trị	Mô tả
ComNone	0	Không băt tay (mặc định)
ComXon/Xoff	1	Bắt tay phần mềm (Xon/Xoff)
ComRTS	2	Bắt tay phần cứng (RTS/CTS)
ComRTSXon/Xoff	3	Bắt tay phần cứng và phàn mềm

❖ **CommEvent:**

Trả lại các lỗi truyền thonog hay sự kiện xảy ra tại cổng nối tiếp

Các sự kiện:

Sự kiện	Giá trị	Mô tả
ComEvSend	1	Đã truyền ký tự
ComEvReceive	2	Khi có ký tự trong bộ đệm nhận
ComEvCTS	3	Có thay đổi trên CTS (Clear To Send)
ComEvDSR	4	Có thay đổi trên DSR (Data Set Ready)
ComEvCD	5	Có thay đổi trên CD (Carrier Detect)
ComEvRing	6	Phát hiện chuông
ComEvEOF	7	Nhận ký tự kết thúc file

Các lỗi truyền thông:

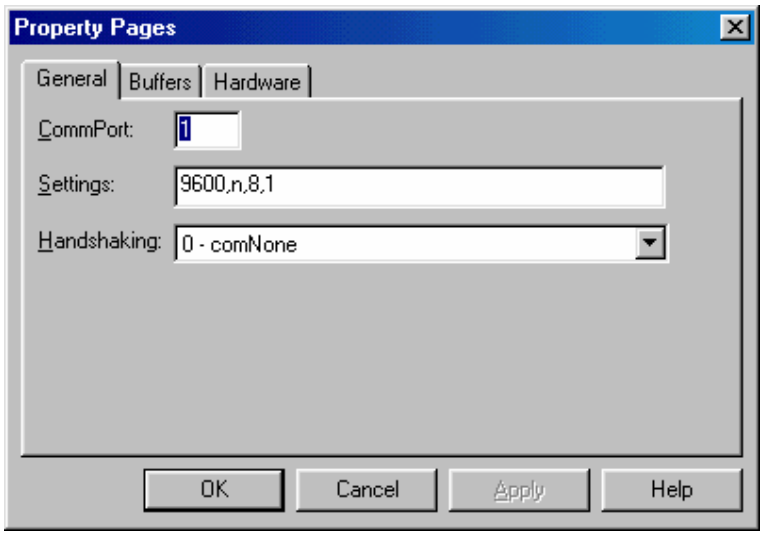
Lỗi	Giá trị	Mô tả
ComBreak	1001	Nhận tín hiệu Break
ComCTSTO	1002	Carrier Detect Timeout
ComFrame	1004	Lỗi khung
ComOver	1006	Phần cứng không đọc ký tự trước khi gởi ký tự kế
ComCDTO	1007	Carrier Detect Timeout
ComRxOver	1008	Tràn bộ đệm nhận
ComRxParity	1009	Lỗi parity
ComTxFull	1010	Tràn bộ đệm truyền

4.3. Sự kiện OnComm

Sự kiện OnComm xảy ra bất cứ khi nào giá trị của thuộc tính CommEvent thay đổi. Các thuộc tính RThreshold và SThreshold = 0 sẽ cấm sự kiện OnComm khi thực hiện nhận hay gởi dữ liệu. Thông thường, SThreshold = 0 và RThreshold = 1.

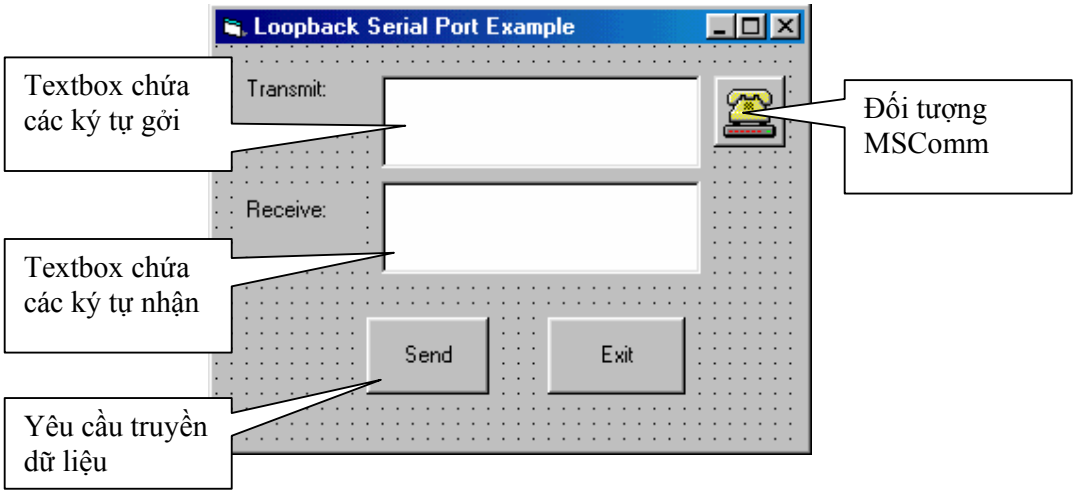
Một chương trình truyền nhận đơn giản thực hiện bằng cách nối chân TxD với RxD của cổng COM1 (loopback). Phương pháp này dùng để kiểm tra cổng nối tiếp.

Thuộc tính cơ bản của cổng nối tiếp:



Hình 4.7 – Các thuộc tính cơ bản của MSComm

Cửa sổ chương trình thực thi:



Hình 4.8 – Cửa sổ chương trình loopback

Chương trình nguồn:

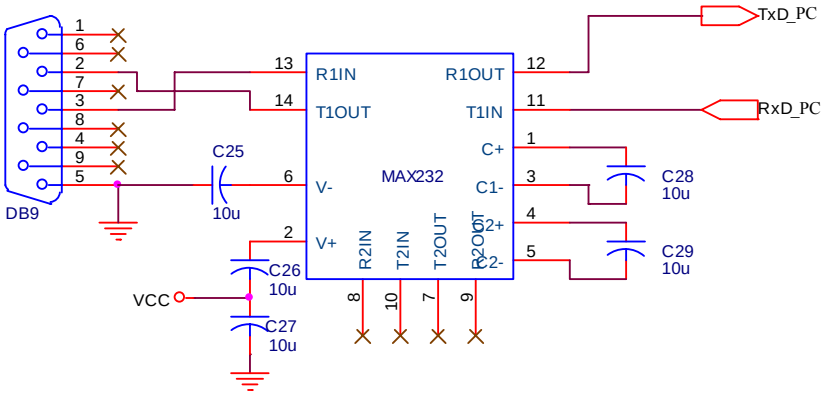
```
VERSION 5.00
Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
```

```
      Begin VB.Form Form1
         Caption              =     "Loopback  Serial  Port
Example"
         ClientHeight    =   3195
         ClientLeft      =   60
         ClientTop       =   345
         ClientWidth     =   4680
         LinkTopic       =   "Form1"
         ScaleHeight     =   3195
         ScaleWidth      =   4680
         StartUpPosition =   3  'Windows Default
         Begin VB.CommandButton cmdExit
            Caption         =   "Exit"
            Height          =   615
            Left            =   2640
            TabIndex        =   5
            Top             =   2160
            Width           =   1095
         End
         Begin VB.CommandButton cmdSend
            Caption         =   "Send"
            Height          =   615
            Left            =   1200
            TabIndex        =   4
            Top             =   2160
            Width           =   975
         End
         Begin VB.TextBox txtReceive
            Height          =   735
            Left            =   1320
            Locked          =   -1  'True
            TabIndex        =   3
            Top             =   1080
            Width           =   2535
         End
         Begin VB.TextBox txtTransmit
            Height          =   735
            Left            =   1320
            TabIndex        =   0
            Top             =   240
            Width           =   2535
         End
         Begin MSCommLib.MSComm MSComm1
            Left            =   3960
            Top             =   240
            _ExtentX        =   1005
            _ExtentY        =   1005
```

```
      _Version        =   393216
      DTREnable       =   -1  'True
      RThreshold      =   1
   End
   Begin VB.Label Label2
      Caption         =   "Receive:"
      Height          =   375
      Left            =   240
      TabIndex        =   2
      Top             =   1200
      Width           =   855
   End
   Begin VB.Label Label1
      Caption         =   "Transmit:"
      Height          =   375
      Left            =   240
      TabIndex        =   1
      Top             =   240
      Width           =   975
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub cmdExit_Click()
MSComm1.PortOpen = False    'Đóng cổng
End
End Sub
Private Sub cmdSend_Click()
MSComm1.Output = Trim(txtTransmit.Text)'Gởi dữ liệu
End Sub
Private Sub Form_Load()
MSComm1.CommPort = 1             'COM1
MSComm1.Settings = "9600,n,8,1"  'Tốc độ 9600bps
MSComm1.PortOpen = True          ' Mở cổng
End Sub
Private Sub MSComm1_OnComm()
If (MSComm1.CommEvent = comEvReceive) Then
txtReceive.Text = txtReceive.Text + MSComm1.Input
End If
End Sub
```

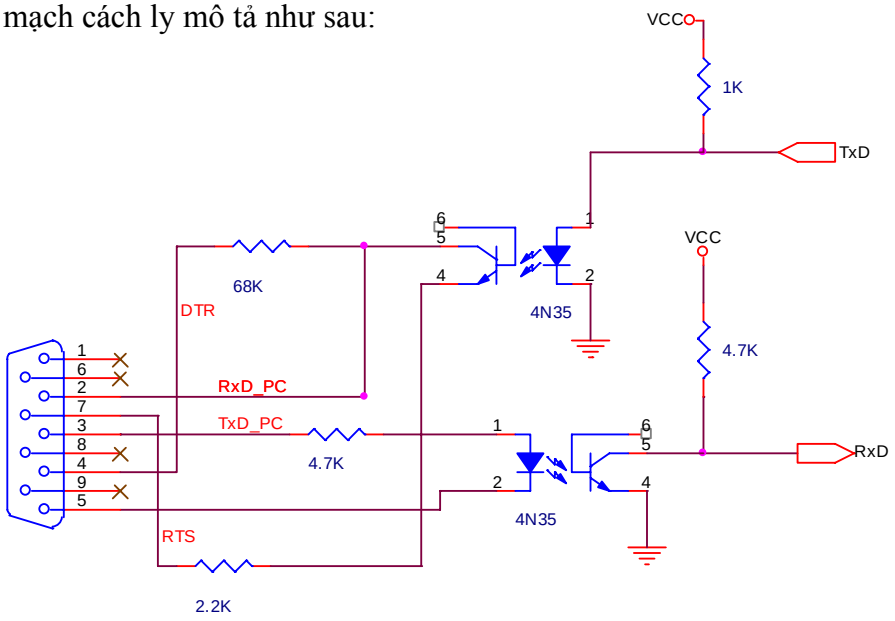
5. Giao tiếp với vi điều khiển

Khi thực hiện giao tiếp với vi điều khiển, ta phải dùng thêm mạch chuyển mức logic từ TTL → 232 và ngược lại. Các vi mạch thường sử dụng là MAX232 của Maxim hay DS275 của Dallas. Mạch chuyển mức logic mô tả như sau:



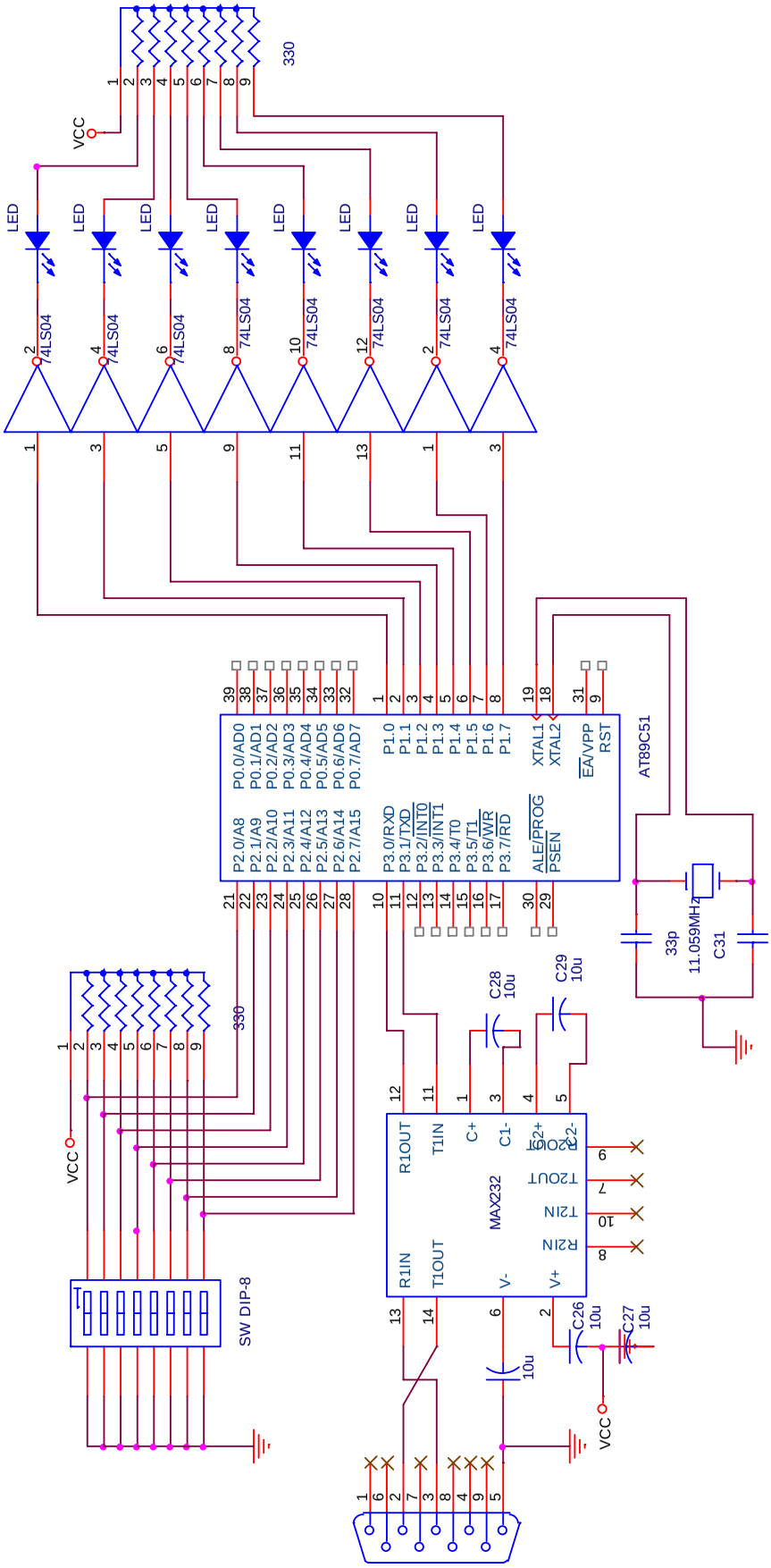
Hình 4.9 – Mạch chuyển mức logic TTL ↔ RS232

Tuy nhiên, khi sử dụng mạch chuyển mức logic dùng các vi mạch thì đòi hỏi phải dùng chung GND giữa máy tính và vi mạch → có khả năng làm hỏng cổng nối tiếp khi xảy ra hiện tượng chập mạch ở mạch ngoài. Do đó, ta có thể dùng thêm opto 4N35 để cách ly về điện. Sơ đồ mạch cách ly mô tả như sau:



Hình 4.10 – Mạch chuyển mức logic TTL ↔ RS232 cách ly

Khi giao tiếp, vi điều khiển chính là một DTE nên sẽ nối RxD của máy tính với TxD của vi điều khiển và ngược lại. Mạch kết nối đơn giản giữa vi điều khiển và máy tính như sau:

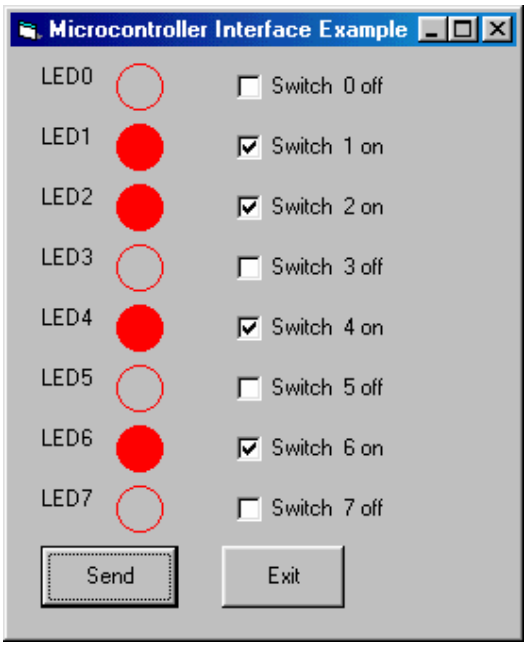


Hình 4.11 – Kết nối với vi điều khiển

Chương trình nguồn cho vi điều khiển AT89C51:

```
     MOV  TMOD,#20h
     MOV  SCON,#52h; Truyền 8 bit dữ liệu, no parity
     MOV  TH1,#(-3); Tốc độ truyền 9600 bps
     MOV  TL1,#(-3)
     SETB TR1

Receive:
     JNB  RI,Transmit    ; Có dữ liệu hay không
     CLR  RI
     MOV  A,SBUF         ; Nếu có thì xuất ra LED
     MOV  P1,A
Transmit:
     JNB  TI,Receive     ; Đã truyền xong chưa
     CLR  TI
     MOV  A,P2           ; Nếu xong thì truyền trạng thái
     MOV  SBUF,A         ; của công tắc SW DIP-8
     JMP  Receive
```

Giao diện của chương trình trên máy tính:



Hình 4.12 – Chương trình giao tiếp với vi điều khiển

Chương trình nguồn:

```
     VERSION 5.00
     Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
```

```
      Begin VB.Form Form1
         Caption            =    "Microcontroller Interface
Example"
         ClientHeight    =   4665
         ClientLeft      =   60
         ClientTop       =   345
         ClientWidth     =   4020
         LinkTopic       =   "Form1"
         ScaleHeight     =   4665
         ScaleWidth      =   4020
         StartUpPosition =   3  'Windows Default
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   7
            Left            =   1800
            TabIndex        =   17
            Top             =   3480
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   6
            Left            =   1800
            TabIndex        =   16
            Top             =   3000
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   5
            Left            =   1800
            TabIndex        =   15
            Top             =   2520
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   4
            Left            =   1800
            TabIndex        =   14
            Top             =   2040
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   3
            Left            =   1800
```

```
   TabIndex        =   13
   Top             =   1560
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   2
   Left            =   1800
   TabIndex        =   12
   Top             =   1080
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   1
   Left            =   1800
   TabIndex        =   11
   Top             =   600
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   0
   Left            =   1800
   TabIndex        =   10
   Top             =   120
   Width           =   1575
End
Begin VB.CommandButton cmdExit
   Caption         =   "Exit"
   Height          =   495
   Left            =   1680
   TabIndex        =   9
   Top             =   3960
   Width           =   975
End
Begin MSCommLib.MSComm MSComm1
   Left            =   3360
   Top             =   3960
   _ExtentX        =   1005
   _ExtentY        =   1005
   _Version        =   393216
   DTREnable       =   -1  'True
   RThreshold      =   1
End
Begin VB.CommandButton cmdSend
   Caption         =   "Send"
```

```
      Height          =   495
      Left            =   240
      TabIndex        =   8
      Top             =   3960
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED7"
      Height          =   375
      Index           =   7
      Left            =   240
      TabIndex        =   7
      Top             =   3480
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED6"
      Height          =   375
      Index           =   6
      Left            =   240
      TabIndex        =   6
      Top             =   3000
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED5"
      Height          =   375
      Index           =   5
      Left            =   240
      TabIndex        =   5
      Top             =   2520
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED4"
      Height          =   375
      Index           =   4
      Left            =   240
      TabIndex        =   4
      Top             =   2040
      Width           =   975
   End
   Begin VB.Label lblLED
```

```
   BackStyle       =   0  'Transparent
   Caption         =   "LED3"
   Height          =   375
   Index           =   3
   Left            =   240
   TabIndex        =   3
   Top             =   1560
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED2"
   Height          =   375
   Index           =   2
   Left            =   240
   TabIndex        =   2
   Top             =   1080
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED1"
   Height          =   375
   Index           =   1
   Left            =   240
   TabIndex        =   1
   Top             =   600
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED0"
   Height          =   375
   Index           =   0
   Left            =   240
   TabIndex        =   0
   Top             =   120
   Width           =   975
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   7
   Left            =   840
   Shape           =   3  'Circle
```

```
   Top             =   3480
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   6
   Left            =   840
   Shape           =   3  'Circle
   Top             =   3000
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   5
   Left            =   840
   Shape           =   3  'Circle
   Top             =   2520
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   4
   Left            =   840
   Shape           =   3  'Circle
   Top             =   2040
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   3
   Left            =   840
   Shape           =   3  'Circle
   Top             =   1560
   Width           =   375
End
```

```
      Begin VB.Shape shpLED
         BorderColor     =   &H000000FF&
         FillColor       =   &H000000FF&
         FillStyle       =   0  'Solid
         Height          =   375
         Index           =   2
         Left            =   840
         Shape           =   3  'Circle
         Top             =   1080
         Width           =   375
      End
      Begin VB.Shape shpLED
         BorderColor     =   &H000000FF&
         FillColor       =   &H000000FF&
         FillStyle       =   0  'Solid
         Height          =   375
         Index           =   1
         Left            =   840
         Shape           =   3  'Circle
         Top             =   600
         Width           =   375
      End
      Begin VB.Shape shpLED
         BorderColor     =   &H000000FF&
         FillColor       =   &H000000FF&
         FillStyle       =   0  'Solid
         Height          =   375
         Index           =   0
         Left            =   840
         Shape           =   3  'Circle
         Top             =   120
         Width           =   375
      End
   End
   Attribute VB_Name = "Form1"
   Attribute VB_GlobalNameSpace = False
   Attribute VB_Creatable = False
   Attribute VB_PredeclaredId = True
   Attribute VB_Exposed = False
   Private Sub cmdExit_Click()
   If MSComm1.PortOpen Then
       MSComm1.PortOpen = False
   End If
   End
   End Sub

   Private Sub cmdSend_Click()
```

```
      Dim t As Integer
      Dim i As Integer
      t = 0
      For i = 0 To 7
         t = t + (2 ^ i) * (1 - shpLED(i).FillStyle)
      Next i
      MSComm1.Output = Chr(t)
      End Sub

      Private Sub Form_Load()
      MSComm1.Settings = "9600,N,8,1"
      MSComm1.CommPort = 1
      MSComm1.PortOpen = True
      End Sub

      Private Sub lblLED_Click(Index As Integer)
shpLED(Index).FillStyle = 1 - shpLED(Index).FillStyle
      End Sub

      Private Sub MSComm1_OnComm()
      Dim t As String
      Dim n As Integer
      Dim i As Integer
      If MSComm1.CommEvent = comEvReceive Then
          n = Asc(MSComm1.Input)
          For i = 0 To 7
              chkSW(i).Value = n Mod 2
              If chkSW(i).Value = 0 Then
                  chkSW(i).Caption = "Switch " & Str(i) &
" off"
              Else
                  chkSW(i).Caption = "Switch " & Str(i) &
" on"
              End If
              n = Fix(n / 2)
          Next i
      End If
      End Sub
```

6. Giao tiếp với MODEM

6.1. Giao tiếp

Quá trình trao đổi dữ liệu giữa máy tính và Modem được thực hiện theo cơ chế bắt tay phần cứng hay phần mềm.

- **Bắt tay phần cứng:** máy tính muốn truyền dữ liệu thì cho RTS = 1 và chờ Modem trả lời bằng tín hiệu CTS. Ngược lại, Modem muốn truyền dữ liệu thì cho DSR = 1 và chờ tín hiệu DTR từ máy tính.
- **Bắt tay phần mềm:** dùng ký tự Xon (Ctrl-S) và Xoff (Ctrl-Q) để bắt đầu truyền hay kết thúc truyền.

Các giao thức truyền dữ liệu trên Modem:

- **XModem:** chia thành khối 128 byte, mỗi khối chèn thêm CRC 4 byte.
- **YModem:** khối 1024 byte.
- **ZModem:** khối có kích thước thay đổi tuỳ theo đường truyền.

Quy tắc truyền lệnh trên Modem:

- Mỗi dòng lệnh của modem bắt đầu bằng ký tự AT, ngoại trừ lệnh A/ và +++.
- Dòng lệnh có thể chứa nhiều lệnh.
- Kết thúc lệnh bằng ký tự Enter (mã ASCII là 13) ngoại trừ lệnh A/ và +++.
- Dòng lệnh cuối cùng được lưu trong modem. Có thể dùng lệnh A/ để thực hiện lại lệnh này.
- Thông báo kết quả thực hiện lệnh của modem có thể ở dạng từ chữ hay số(giá trị mặc định là chữ). Có thể sử dụng lệnh V để lựa chọn dạng thông báo là chữ hay số.
- Để hoạt động đúng, modem cần có các thông số xác định. Nếu không có sự thay đổi cần thiết, modem hoạt động theo giá trị mặc định(default). Nếu thông số trong lệnh bị bỏ qua, giá trị thông số mặc định là 0.

6.2. Các lệnh cơ bản của Modem

Lệnh	Mô tả
+++	Chuyển Modem sang chế độ lệnh
A/	Lặp lại lệnh trước
A	Cho phép kết nối và phát tín hiệu sóng mang. Modem sẽ báo tín hiệu CONNECT nếu thu được tín hiêu sóng mang từ modem đầu cuối. Nếu không thu được sóng mang, modem sẽ gác máy và thông báo NO CARRIER
DPn	Quay số điện thoại n dạng xung
DTn	Quay số điện thoại n dạng tone
H0	Gác máy
H1	Nhấc máy
O0	Chuyển về chế độ dữ liệu
O1	Chế độ điều chỉnh Modem
Q0	Cho phép Modem gởi thông báo đến DTE (mặc định)
Q1	Cấm Modem gởi thông báo
Q2	Gởi thông báo khi Modem chủ động kết nối, không gởi khi Modem nhận cuộc gọi
V0	Nhận thông báo dạng số

V1	Nhận thông báo dạng ký tự (mặc định)
Sn = V	Nạp giá trị V vào thanh ghi Sn S0 = V: chờ V hồi chuông trước khi trả lời, V = 0 – 255 (mặc định V = 0: không trả lời) S6 = V: chờ V giây trước khi quay số (mặc định V = 2) S7 = V: chờ V giây kể từ lúc gọi đến lúc nhận được tín hiệu, nếu không sẽ thông báo lỗi
Sn?	Đọc nội dung thanh ghi Sn
Z0	Reset Modem về cấu hình 0
Z1	Reset Modem về cấu hình 1
L0, L1, L2, L3	Âm lượng loa Modem
M0	Tắt loa
M1	Mở loa cho đến khi nhận dược sóng mang (măc định)
M2	Mở loa
M3	Tắt loa khi quay số và nhận sóng mang

6.3. Các thanh ghi thông dụng trên modem

Thanh ghi S0: xác định số hồi chuông nhận được mà sau đó modem sẽ trả lời một cách tự động. Giá trị trong thanh ghi này có thể thay đổi trong khoảng từ 0-255. mặc định giá trị là 0 (không trả lời).

Thanh ghi S1: Thanh ghi S1 chỉ có tác dụng khi thanh ghi S0 khác 0, dùng để đếm số hồi chuông thu được.

Thanh ghi S2: xác định giá trị thập phân của các ký tự (mã ASSCII) được dùng làm ký tự thoát, Giá trị mặc định là 43(+)

Thanh ghi S3: xác định ký tự được dùng để kết thúc một dòng lệnh, mặc nhiên là 13 (tương ứng là Enter)

Thanh ghi S4: xác định ký tự xuống dòng sau ký tự kết thúc, giá trị mặc nhiên là 10 (line feed)

Thanh ghi S5: xác định phím xoá lui, giá trị mặc nhiên là 8 (backspace)

Thanh ghi S6: xác định thời gian đợi sau khi truy cập đường điện thoại và trước khi tiến hành quoay digit đầu tiên trong một lệnh quay số. Đây là thời gian trì hoãn cho phép để dial tone cung cấp từ đường truyền. Giá trị mặc nhiên và tối thiểu là 2s.

Thanh ghi S7: xác định thời gian mà modem đợi tín hiệu sóng mang trước khi gác máy. Giá trị mặc định là 30s.

Thanh ghi S8: xác định thời gian tạm dừng cho mỗi dấu phẩy ',' trong chuỗi lệnh quay số. Giá trị mặc định là 2s

Thanh ghi S9: xác định thời gian mà tín hiệu sóng mang phải hiện diện để modem có thể nhận biết được, giá trị mặc định là 600ms. Giá trị này nếu quá lớn sẽ gây lỗi trong dữ liệu truyền.

Thanh ghi S10: xác định thời gian cho phép tín hiệu sóng mang có thể biến mất trong chốc lát nào đó mà không cắt cuộc nối. Ổn định trong khoảng 100-25500ms, giá trị mặc nhiên tùy vào khả năng chống nhiễu của từng modem, thường là 700ms.

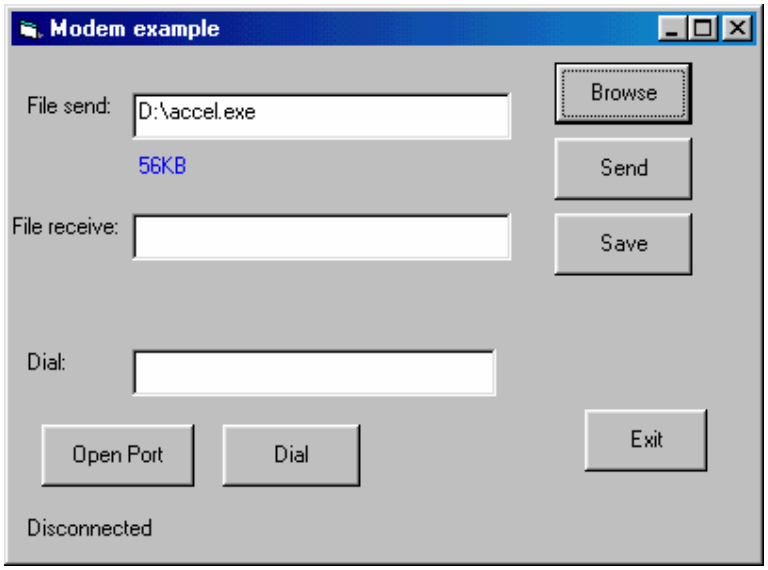
Thanh ghi S11: xác định tốc độ quay số khi sử dụng phương pháp quay số tone, giá trị mặc nhiên tùy vào modem, thường vào khoảng 70ms.

Thanh ghi S12: xác định thời gian an toàn khi truy nhập vào ký tự thoát (+++). Nếu giá trị nhỏ quá có thể nhập không kịp, giá trị lớn quá so với tốc độ nhập cũng không thể thoát được.

6.4. Các thông báo của Modem

Dạng ký tự	Dạng số	Ý nghĩa
OK	0	Lệnh thành công
CONNECT	1	Kết nối 300 bps
RING	2	Có tín hiệu chuông
NO CARRIER	3	Không có sóng mang
ERROR	4	Lỗi: nhận lệnh không giá trị, sai kiểm tra, hàng lệnh quá dài
CONNECT 1200	5	Kết nối 1200bps
NO DIAL TONE	6	Không có âm hiệu mời quay số
BUSY	7	Máy bận
NO ANSWER	8	Không có tín hiệu trả lời
CONNECT 2400	10	Kết nối 2400bps
CONNECT 4800	11	Kết nối 4800bps
CONNECT 9600	12	Kết nối 9600bps
CONNECT 14400	13	Kết nối 14400bps
CONNECT 19200	14	Kết nối 19200bps
CONNECT 16800	15	Kết nối 16800bps
CONNECT 57600	18	Kết nối 57600bps
CONNECT 7200	24	Kết nối 7200bps
CONNECT 12000	25	Kết nối 12000bps
CONNECT 28800	32	Kết nối 28800bps
CONNECT 115200	33	Kết nối 115200bps
CARRIER 300	40	Phát hiện sóng mang
CARRIER 9600	50	Phát hiện sóng mang
CARRIER 28800	58	Phát hiện sóng mang

Ví dụ lập trình điều khiển Modem như sau:



Hình 4.13 – Giao tiếp và điều khiển Modem

Chương trình nguồn:

```
     VERSION 5.00
     Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
     Object          =          "{F9043C88-F6F2-101A-A3C9-
08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
     Begin VB.Form frmModem
        Caption         =   "Modem example"
        ClientHeight    =   4065
        ClientLeft      =   60
        ClientTop       =   345
        ClientWidth     =   5925
        LinkTopic       =   "Form1"
        ScaleHeight     =   4065
        ScaleWidth      =   5925
        StartUpPosition =   3  'Windows Default
        Begin VB.CommandButton cmdSave
           Caption         =   "Save"
           Height          =   495
           Left            =   4320
           TabIndex        =   14
           Top             =   1320
           Width           =   1095
        End
        Begin VB.TextBox txtReceive
           Height          =   375
           Left            =   960
```

```
   TabIndex        =   12
   Top             =   1320
   Width           =   3015
End
Begin VB.Timer Timer1
   Enabled         =   0   'False
   Interval        =   1000
   Left            =   4920
   Top             =   2400
End
Begin VB.CommandButton cmdExit
   Caption         =   "Exit"
   Height          =   495
   Left            =   4560
   TabIndex        =   10
   Top             =   2880
   Width           =   975
End
Begin VB.TextBox txtDial
   Height          =   375
   Left            =   960
   TabIndex        =   7
   Top             =   2400
   Width           =   2895
End
Begin VB.CommandButton cmdDial
   Caption         =   "Dial"
   Height          =   495
   Left            =   1680
   TabIndex        =   5
   Top             =   3000
   Width           =   1095
End
Begin VB.CommandButton cmdSend
   Caption         =   "Send"
   Height          =   495
   Left            =   4320
   TabIndex        =   4
   Top             =   720
   Width           =   1095
End
Begin VB.CommandButton cmdOpen
   Caption         =   "Open Port"
   Height          =   495
   Left            =   240
   TabIndex        =   3
   Top             =   3000
```

```
      Width           =   1215
   End
   Begin VB.CommandButton cmdBrowse
      Caption         =   "Browse"
      Height          =   495
      Left            =   4320
      TabIndex        =   1
      Top             =   120
      Width           =   1095
   End
   Begin MSComDlg.CommonDialog diagSend
      Left            =   4200
      Top             =   3120
      _ExtentX        =   847
      _ExtentY        =   847
      _Version        =   393216
   End
   Begin VB.TextBox txtSend
      Height          =   375
      Left            =   960
      TabIndex        =   0
      Top             =   360
      Width           =   3015
   End
   Begin MSCommLib.MSComm MSComm1
      Left            =   5160
      Top             =   3000
      _ExtentX        =   1005
      _ExtentY        =   1005
      _Version        =   393216
      DTREnable       =   -1  'True
      Handshaking     =   2
      NullDiscard     =   -1  'True
      RThreshold      =   1
      RTSEnable       =   -1  'True
   End
   Begin VB.Label Label3
      Caption         =   "File receive:"
      Height          =   375
      Left            =   0
      TabIndex        =   13
      Top             =   1320
      Width           =   855
   End
   Begin VB.Label lblReceive
      Caption          =   "Receive file !!! Select
file name."
```

```
      ForeColor       =   &H000000FF&
      Height          =   375
      Left            =   840
      TabIndex        =   11
      Top             =   1920
      Visible         =   0   'False
      Width           =   2895
   End
   Begin VB.Label lblStatus
      Caption         =   "Disconnected"
      Height          =   375
      Left            =   120
      TabIndex        =   9
      Top             =   3720
      Width           =   5775
   End
   Begin VB.Label Label2
      Caption         =   "Dial:"
      Height          =   375
      Left            =   120
      TabIndex        =   8
      Top             =   2400
      Width           =   735
   End
   Begin VB.Label Label1
      Caption         =   "File send:"
      Height          =   375
      Left            =   120
      TabIndex        =   6
      Top             =   360
      Width           =   735
   End
   Begin VB.Label lblSize
      ForeColor       =   &H00FF0000&
      Height          =   375
      Left            =   960
      TabIndex        =   2
      Top             =   840
      Width           =   1815
   End
End
Attribute VB_Name = "frmModem"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Connected As Boolean
```

```
    Private SendFlag As Boolean
    Private ReceiveFlag As Boolean
    Private FileReceive As Integer
    Private CRFlag As Boolean
    Private Sub cmdBrowse_Click()
    On Error GoTo Loi
    diagSend.FileName = ""
    diagSend.Filter = "All files(*.*)|*.*"
    diagSend.InitDir = App.Path
    diagSend.ShowOpen
    txtSend.Text = diagSend.FileName
    lblSize.Caption = Str(Round(FileLen(txtSend.Text) /
1024, 2)) + "KB"
    Exit Sub
    Loi:
    lblSize.Caption = "0 KB"
    txtSend.Text = ""
    End Sub

    Private Sub cmdDial_Click()
    If Not MSComm1.PortOpen Then
        MsgBox "Comm Port Closed. Open first!!!",
vbOKOnly + vbCritical, "Error"
    ElseIf Trim(txtDial.Text) = "" Then
        MsgBox "Enter phone's number!!!", vbOKOnly +
vbCritical, "Error"
    Else
        If cmdDial.Caption = "Dial" Then
            MSComm1.Output = "ATDT" & Trim(txtDial.Text)
+ vbCr
            cmdDial.Caption = "Hang up"
            lblStatus.Caption = "Dialing ..."
        Else
            MSComm1.Output = "ATH1" + vbCr
            cmdDial.Caption = "Dial"
            lblStatus.Caption = "Hang up"
        End If
    End If

    End Sub

    Private Sub cmdExit_Click()
    If MSComm1.PortOpen Then
        MSComm1.PortOpen = False
    End If
    End
    End Sub
```

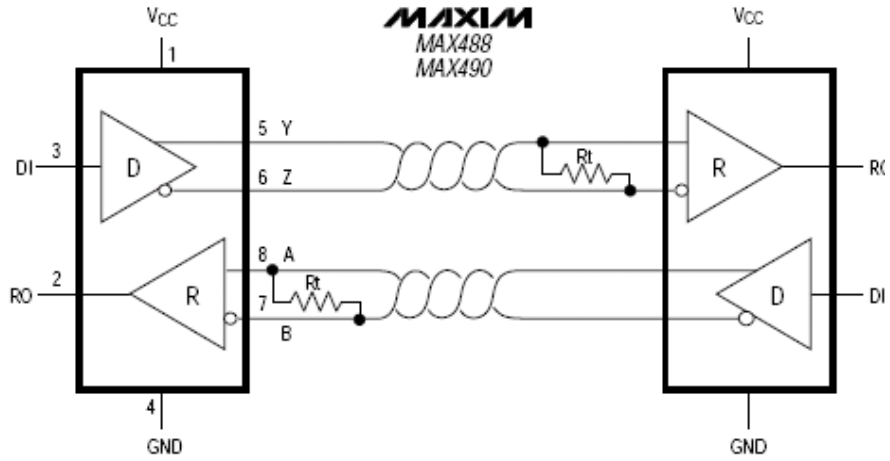
```
Private Sub cmdOpen_Click()
MSComm1.PortOpen = Not MSComm1.PortOpen
If MSComm1.PortOpen Then
    cmdOpen.Caption = "Close Port"
    MSComm1.Output = "ATS0=5" + vbCr
    Call Form_Load
Else
    cmdOpen.Caption = "Open Port"
    lblStatus.Caption = "Disconnected"
End If
End Sub

Private Sub cmdSave_Click()
            FileReceive = FreeFile
            ReceiveFlag = True
            Timer1.Enabled = False
            Do
                diagSend.FileName = ""
                diagSend.ShowSave
                If Trim(diagSend.FileName) = "" Then
                    MsgBox "File name error!!",
vbCritical + vbOKOnly, "Error"
                End If
            Loop While Trim(diagSend.FileName) = ""
            txtReceive.Text = diagSend.FileName
            MSComm1.Output = "RECEIVE" + vbCr
            Open Trim(txtReceive.Text) For Output As
#FileReceive
End Sub

Private Sub cmdSend_Click()
Dim FileNum As Integer
Dim Buffer As String
If Not MSComm1.PortOpen Then
    MsgBox "Comm Port Closed. Open first!!!",
vbOKOnly + vbCritical, "Error"
ElseIf Not Connected Then
    MsgBox "Not connected!!!", vbOKOnly +
vbCritical, "Error"
ElseIf Trim(txtSend.Text) = "" Then
    MsgBox "Select a file to send!!!", vbOKOnly +
vbCritical, "Error"
Else
    MSComm1.Output = "SEND" + vbCr
    Do
        DoEvents
```

```
            Loop While Not SendFlag
            FileNum = FreeFile
            Open Trim(txtSend.Text) For Input As #FileNum
            Do
                Input #FileNum, Buffer
                If Right(Buffer, 1) <> vbCr Then Buffer =
    Buffer + vbCrLf
                MSComm1.Output = Buffer
            Loop While Not EOF(FileNum)
            MSComm1.Output = "END FILE"
            Close #FileNum
            SendFlag = False
        End If
        End Sub

        Private Sub Form_Load()
        Connected = False
        SendFlag = False
        ReceiveFlag = False
        CRFlag = False
        End Sub

        Private Sub MSComm1_OnComm()
        Dim Buffer As String
        Dim Buffer1 As String
        Dim Buff As String
        Dim i As Integer
        Select Case MSComm1.CommEvent
            Case comEvRing
                lblStatus.Caption = "Ringing..."
            Case comEvCD
                If MSComm1.CDHolding Then
                    lblStatus.Caption = "Connected"
                    Connected = True
                Else
                    lblStatus.Caption = "Disconnected"
                    Connected = False
                End If
            Case comEvReceive
                Buffer = MSComm1.Input
                If InStr(Buffer, "SEND") Then
                    Timer1.Enabled = True
                    Exit Sub
                End If
                If InStr(Buffer, "RECEIVE") Then
                    SendFlag = True
                    Timer1.Enabled = False
```

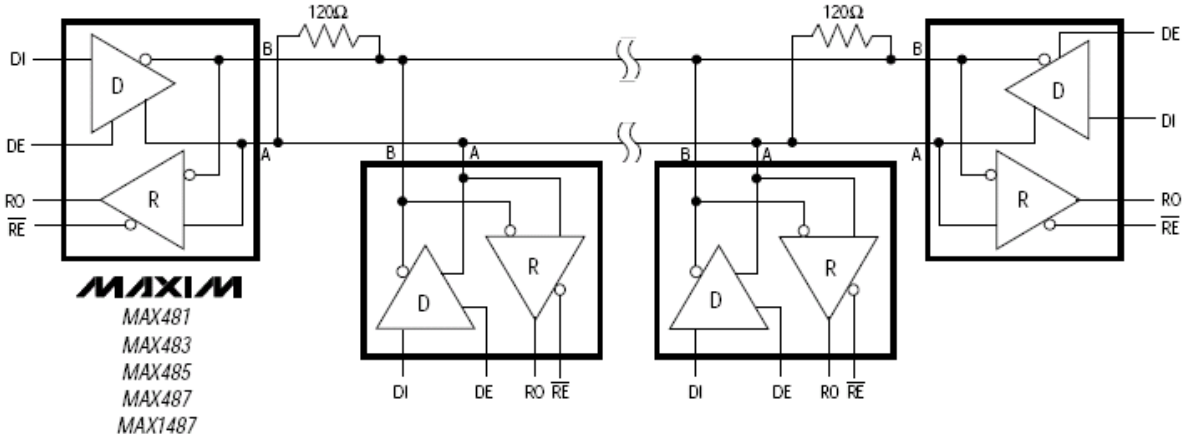
```
                    Buffer = ""
                    Exit Sub
                End If
                If InStr(Buffer, "CONNECT") Then
                    Connected = True
                    lblStatus.Caption = "Connected"
                    Exit Sub
                End If
                If ReceiveFlag Then
                    Buffer1 = ""
                    For i = 1 To Len(Buffer)
                        Buff = Mid$(Buffer, i, 1)
                        If Buff = Chr$(13) Then
                            CRFlag = True
                            Buff = ""
                        ElseIf Buff = Chr$(10) Then
                            CRFlag = False
                            Buff = ""
                        If Not CRFlag Then
                            Buffer1 = Buffer1 + Buff
                        End If
                    Next i
                    Print #FileReceive, Buffer1
                End If
                If InStr(Buffer, "END FILE") Then
                    Close #FileReceive
                    Call Form_Load
                End If
            Case comEvEOF
                lblStatus = "Disconnected"
                Connected = False
        End Select
        End Sub

        Private Sub Timer1_Timer()
        lblReceive.Visible = Not lblReceive.Visible
        End Sub

        Private Sub txtSend_LostFocus()
        On Error GoTo Loi
        lblSize.Caption = Str(Round(FileLen(txtSend.Text) /
1024, 2)) + "KB"
        Exit Sub
        Loi:
        lblSize.Caption = "0 KB"
        txtSend.Text = ""
        End Sub
```

7. Mạng 485

Chuẩn RS232 dùng đường truyền không cân bằng vì các tín hiệu lấy chuẩn là GND chung nên dễ bị ảnh hưởng của nhiễu làm tốc độ và khoảng cách truyền bị giới hạn. Khi muốn tăng khoảng cách truyền, môt phương pháp có thể sử dụng là dùng 2 dây truyền vi sai vì lúc này 2 dây có cùng đặc tính nên sẽ loại trừ được nhiễu chung. Hai chuẩn được sử dụng là RS422 và RS485 nhưng thông thường sử dụng RS485. Điện áp vi sai yêu cầu phải lớn hơn 200mV. Nếu $V_{AB} > 200$ mV thì tương ứng với logic 1 và $V_{AB} < -200$ mV tương ứng với logic 0. Chuẩn RS485 sử dụng hai điện trở kết thúc là 120 Ω tại hai đầu xa nhất của đường truyền và sử dụng dây xoắn đôi.



Hình 4.13 – Chuẩn giao tiếp RS422



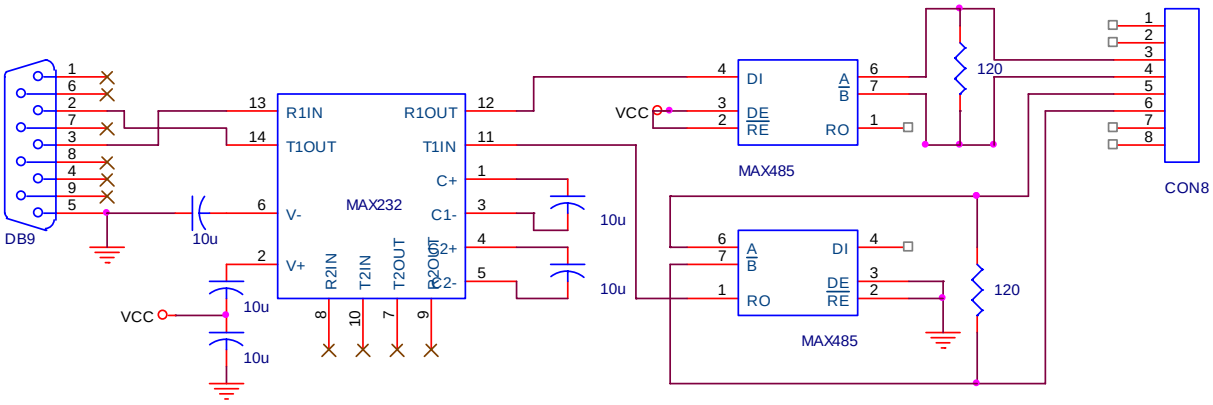
Hình 4.14 – Chuẩn giao tiếp RS485

Các đặc tính kỹ thuật:

Đặc tính	RS422	RS485
Số thiết bị truyền	1	32
Số thiết bị nhận	10	32

Chiều dài cable cực đại	1200m	1200m
Tốc độ truyền cực đại (từ 12 – 1200m)	10Mps – 100Kbps	10Mps – 100Kbps
Điện áp cực đại tại ngõ ra thiết bị truyền	-0.25V ÷ 6V	-7V ÷ 12V
Điện áp ngõ vào thiết bị nhận	-10V ÷ 10V	-7V ÷ 12V

Đối với chuẩn RS232, khoảng cách truyền không cho phép đi xa nên khi muốn thực hiện truyền ở khoảng cách xa thì phải chuyển từ RS232 sang chuẩn RS485 để truyền đi và sau đó chuyển từ RS485 sang RS232 để máy tính có thể nhận dạng được. Sơ đồ mạch chuyển đổi từ RS232 sang RS485 và ngược lại mô tả như sau:



Hình 4.15 – Chuyển đổi từ RS323 sang RS485 và ngược lại

Chương 5
GIAO TIẾP CỔNG SONG SONG

1. Cấu trúc cổng song song

Cổng song song gồm có 4 đường điều khiển, 5 đường trạng thái và 8 đường dữ liệu bao gồm 5 chế độ hoạt động:

- Chế độ tương thích (compatibility).
- Chế độ nibble.
- Chế độ byte.
- Chế độ EPP (Enhanced Parallel Port).
- Chế độ ECP (Extended Capabilities Port).

3 chế độ đầu tiên sử dụng port song song chuẩn (SPP – Standard Parallel Port) trong khi đó chế độ 4, 5 cần thêm phần cứng để cho phép hoạt động ở tốc độ cao hơn. Sơ đồ chân của máy in như sau:

Chân	Tín hiệu	Mô tả
1	$\overline{\text{STR}}$ (Out)	Mức tín hiệu thấp, truyền dữ liệu tới máy in
2	D0	Bit dữ liệu 0
3	D1	Bit dữ liệu 1
4	D2	Bit dữ liệu 2
5	D3	Bit dữ liệu 3
6	D4	Bit dữ liệu 4
7	D5	Bit dữ liệu 5
8	D6	Bit dữ liệu 6
9	D7	Bit dữ liệu 7
10	$\overline{\text{ACK}}$ (In)	Mức thấp: máy in đã nhận 1 ký tự và có khả năng nhận nữa
11	BUSY (In)	Mức cao: ký tự đã được nhận; bộ đệm máy in đầy; khởi động máy in; máy in ở trạng thái off-line.
12	PAPER EMPTY (In)	Mức cao: hết giấy
13	SELECT (In)	Mức cao: máy in ở trạng thái online
14	$\overline{\text{AUTOFEED}}$ (Out)	Tự động xuống dòng; mức thấp: máy in xuống dòng tự động
15	$\overline{\text{ERROR}}$ (In)	Mức thấp: hết giấy; máy in ở offline; lỗi máy in
16	$\overline{\text{INIT}}$ (Out)	Mức thấp: khởi động máy in
17	$\overline{\text{SELECTIN}}$ (Out)	Mức thấp: chọn máy in
18-25	GROUND	0V

Cổng song song có ba thanh ghi có thể truyền dữ liệu và điều khiển máy in. Địa chỉ cơ sở của các thanh ghi cho tất cả cổng LPT (line printer) từ LPT1 đến LPT4 được lưu trữ trong vùng dữ liệu của BIOS. Thanh ghi dữ liệu được định vị ở offset 00h, thanh ghi trang

thái ở 01h, và thanh ghi điều khiển ở 02h. Thông thường, địa chỉ cơ sở của LPT1 là 378h, LPT2 là 278h, do đó địa chỉ của thanh ghi trạng thái là 379h hoặc 279h và địa chỉ thanh ghi điều khiển là 37Ah hoặc 27Ah. Tuy nhiên trong một số trường hợp, địa chỉ của cổng song song có thể khác do quá trình khởi động của BIOS. BIOS sẽ lưu trữ các địa chỉ này như sau:

Địa chỉ	Chức năng
0000h:0408h	Địa chỉ cơ sở của LPT1
0000h:040Ah	Địa chỉ cơ sở của LPT2
0000h:040Ch	Địa chỉ cơ sở của LPT3

Định dạng các thanh ghi như sau:

Thanh ghi dữ liệu (hai chiều):

	7	6	5	4	3	2	1	0
Tín hiệu máy in	D7	D6	D5	D4	D3	D2	D1	D0
Chân số	9	8	7	6	5	4	3	2

Thanh ghi trạng thái máy in (chỉ đọc):

	7	6	5	4	3	2	1	0
Tín hiệu máy in	BUSY	$\overline{\text{ACK}}$	PAPER EMPTY	SELECT	$\overline{\text{ERROR}}$	$\overline{\text{IRQ}}$	x	x
Số chân cắm	11	10	12	13	15	-	-	-

Thanh ghi điều khiển máy in:

	7	6	5	4	3	2	1	0
Tín hiệu máy in	x	x	DIR	IRQ Enable	$\overline{\text{SELECTIN}}$	$\overline{\text{INIT}}$	$\overline{\text{AUTOFEED}}$	$\overline{\text{STROBE}}$
Số chân cắm	-	-	-	-	17	16	14	1

x: không sử dụng

IRQ Enable: yêu cầu ngắt cứng; 1 = cho phép; 0 = không cho phép

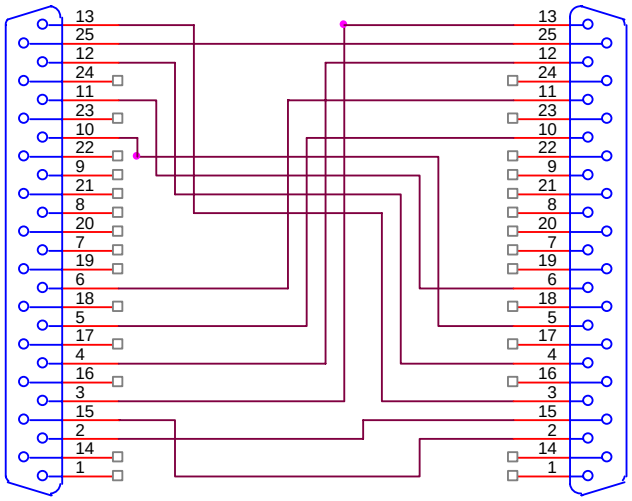
Chú ý rằng chân BUSY được nối với cổng đảo trước khi đưa vào thanh ghi trạng thái, các bit $\overline{\text{SELECTIN}}$, $\overline{\text{AUTOFEED}}$ và $\overline{\text{STROBE}}$ được đưa qua cổng đảo trước khi đưa ra các chân của cổng máy in.

Thông thường tốc độ xử lý dữ liệu của các thiết bị ngoại vi như máy in chậm hơn PC nhiều nên các đường $\overline{\text{ACK}}$, BUSY và $\overline{\text{STR}}$ được sử dụng cho kỹ thuật bắt tay. Khởi đầu, PC đặt dữ liệu lên bus sau đó kích hoạt đường $\overline{\text{STR}}$ xuống mức thấp để thông tin cho máy in biết rằng dữ liệu đã ổn định trên bus. Khi máy in xử lý xong dữ liệu, nó sẽ trả lại tín hiệu $\overline{ACK}$ xuống mức thấp để ghi nhận. PC đợi cho đến khi đường BUSY từ máy in xuống thấp (máy in không bận) thì sẽ đưa tiếp dữ liệu lên bus.

2. Giao tiếp với thiết bị ngoại vi

2.1. Giao tiếp với máy tính

Quá trình giao tiếp với cổng song song dùng 2 chế độ: chế độ chuẩn SPP và chế độ mở rộng. Việc giao tiếp ở chế độ chuẩn mô tả như sau:

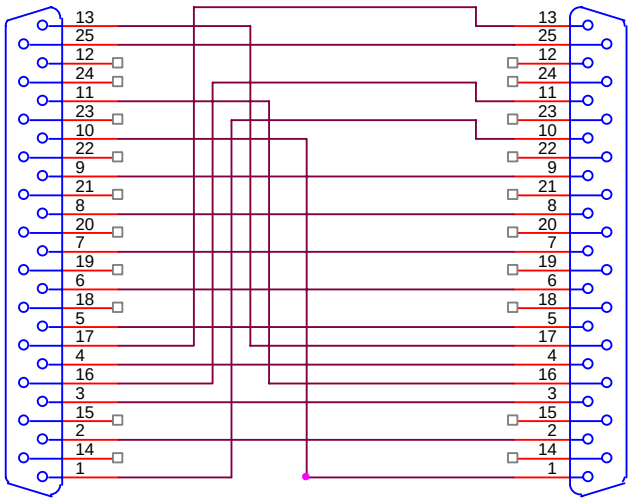


Hình 5.1 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ chuẩn

Sơ đồ chân kết nối mô tả như sau:

PC1		PC2	
Chức năng	**Chân**	**Chân**	**Chức năng**
D0	2	15	$\overline{\text{ERROR}}$
D1	3	13	SELECT
D2	4	12	PAPER EMPTY
D3	5	10	$\overline{\text{ACK}}$
D4	6	11	BUSY
BUSY	11	6	D4
$\overline{\text{ACK}}$	10	5	D3
PAPER EMPTY	12	4	D2
SELECT	13	3	D1
$\overline{\text{ERROR}}$	15	2	D0
GND	25	25	GND

Ngoài ra, việc kết nối giữa 2 máy tính sử dụng cổng song song có thể dùng chế độ mở rộng, chế độ này cho phép giao tiếp với tốc độ cao hơn.



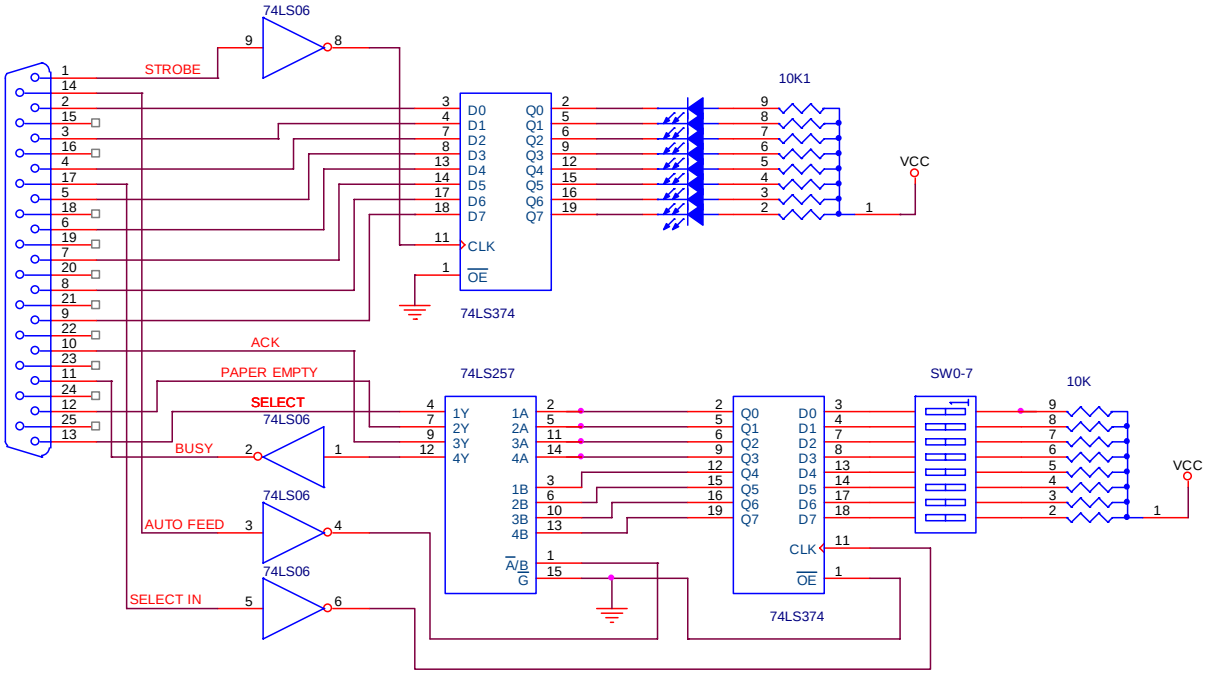
Hình 5.2 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ mở rộng

Sơ đồ chân kết nối mô tả như sau:

PC1		**PC2**	
Chức năng	**Chân**	**Chân**	**Chức năng**
D0	2	2	D0
D1	3	3	D1
D2	4	4	D2
D3	5	5	D3
D4	6	6	D4
D5	7	7	D5
D6	8	8	D6
D7	9	9	D7
SELECT	13	17	$\overline{\text{SELECTIN}}$
BUSY	11	16	$\overline{\text{INIT}}$
$\overline{\text{ACK}}$	10	1	$\overline{\text{STROBE}}$
$\overline{\text{SELECTIN}}$	17	13	SELECT
$\overline{\text{INIT}}$	16	11	BUSY
$\overline{\text{STROBE}}$	1	10	$\overline{\text{ACK}}$

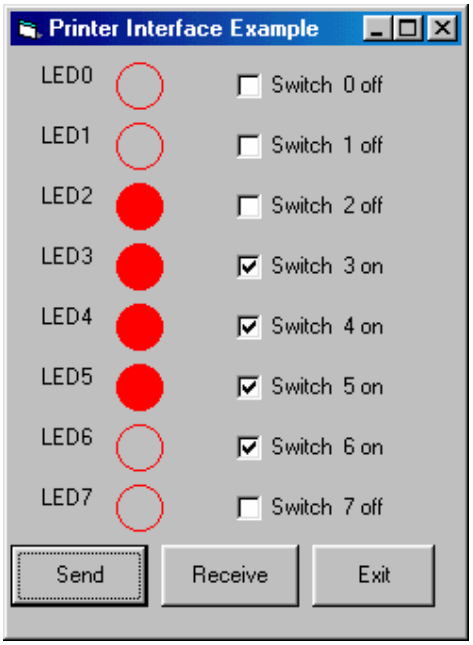
2.2. Giao tiếp thiết bị khác

Quá trình giao tiếp với các thiết bị ngoại vi có thể thực hiện thông qua chế độ chuẩn. Để đọc dữ liệu, có thể dùng một IC ghép kênh 2→1 74LS257 và dùng 4 bit trạng thái của cổng song song còn xuất dữ liệu thì sử dụng 8 đường dữ liệu D0 – D7.



Hình 5.3 – Mạch giao tiếp đơn giản thông qua cổng máy in

Giao diện:



Hình 5.4 – Giao diện của chương trình giao tiếp với cổng máy in

Chương trình giao tiếp trên VB sử dụng thư viện liên kết động để trao đổi dữ liệu với cổng máy in. Thư viện IO.DLL bao gồm các hàm sau:

- Hàm PortOut: xuất 1 byte ra cổng

```
    Private Declare Sub PortOut Lib "IO.DLL" (ByVal Port
As Integer, ByVal Data As Byte)
```

Port: địa chỉ cổng, Data: dữ liệu xuất

- Hàm PortWordOut: xuất 1 word ra cổng

```
    Private Declare Sub PortWordOut Lib "IO.DLL" (ByVal
Port As Integer, ByVal Data As Integer)
```

- Hàm PortDWordOut: xuất 1 double word ra cổng

```
    Private Declare Sub PortDWordOut Lib "IO.DLL" (ByVal
Port As Integer, ByVal Data As Long)
```

- Hàm PortIn: nhập 1 byte từ cổng, trả về giá trị nhập

```
    Private Declare Function PortIn Lib "IO.DLL" (ByVal
Port As Integer) As Byte
```

- Hàm PortWordIn: nhập 1 word từ cổng

```
    Private Declare Function PortWordIn Lib "IO.DLL"
(ByVal Port As Integer) As Integer
```

- Hàm PortDWordIn: nhập 1 double word từ cổng

```
    Private Declare Function PortDWordIn Lib "IO.DLL"
(ByVal Port As Integer) As Long
```

Chương trình nguồn:

```
    VERSION 5.00
    Begin VB.Form Form1
       Caption         =   "Printer Interface Example"
       ClientHeight    =   4665
       ClientLeft      =   60
       ClientTop       =   345
       ClientWidth     =   3585
       LinkTopic       =   "Form1"
       ScaleHeight     =   4665
       ScaleWidth      =   3585
       StartUpPosition =   3  'Windows Default
       Begin VB.CommandButton cmdReceive
          Caption         =   "Receive"
          Height          =   495
          Left            =   1200
          TabIndex        =   18
          Top             =   3960
          Width           =   1095
```

```
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   7
   Left            =   1800
   TabIndex        =   17
   Top             =   3480
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   6
   Left            =   1800
   TabIndex        =   16
   Top             =   3000
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   5
   Left            =   1800
   TabIndex        =   15
   Top             =   2520
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   4
   Left            =   1800
   TabIndex        =   14
   Top             =   2040
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   3
   Left            =   1800
```

```
      TabIndex        =   13
      Top             =   1560
      Width           =   1575
   End
   Begin VB.CheckBox chkSW
      Height          =   375
      Index           =   2
      Left            =   1800
      TabIndex        =   12
      Top             =   1080
      Width           =   1575
   End
   Begin VB.CheckBox chkSW
      Height          =   375
      Index           =   1
      Left            =   1800
      TabIndex        =   11
      Top             =   600
      Width           =   1575
   End
   Begin VB.CheckBox chkSW
      Height          =   375
      Index           =   0
      Left            =   1800
      TabIndex        =   10
      Top             =   120
      Width           =   1575
   End
   Begin VB.CommandButton cmdExit
      Caption         =   "Exit"
      Height          =   495
      Left            =   2400
      TabIndex        =   9
      Top             =   3960
      Width           =   975
   End
   Begin VB.CommandButton cmdSend
```

```
      Caption         =   "Send"
      Height          =   495
      Left            =   0
      TabIndex        =   8
      Top             =   3960
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED7"
      Height          =   375
      Index           =   7
      Left            =   240
      TabIndex        =   7
      Top             =   3480
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED6"
      Height          =   375
      Index           =   6
      Left            =   240
      TabIndex        =   6
      Top             =   3000
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED5"
      Height          =   375
      Index           =   5
      Left            =   240
      TabIndex        =   5
      Top             =   2520
      Width           =   975
   End
```

```
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED4"
   Height          =   375
   Index           =   4
   Left            =   240
   TabIndex        =   4
   Top             =   2040
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED3"
   Height          =   375
   Index           =   3
   Left            =   240
   TabIndex        =   3
   Top             =   1560
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED2"
   Height          =   375
   Index           =   2
   Left            =   240
   TabIndex        =   2
   Top             =   1080
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED1"
   Height          =   375
   Index           =   1
   Left            =   240
   TabIndex        =   1
```

```
      Top             =   600
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED0"
      Height          =   375
      Index           =   0
      Left            =   240
      TabIndex        =   0
      Top             =   120
      Width           =   975
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   7
      Left            =   840
      Shape           =   3  'Circle
      Top             =   3480
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   6
      Left            =   840
      Shape           =   3  'Circle
      Top             =   3000
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
```

```
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   5
      Left            =   840
      Shape           =   3  'Circle
      Top             =   2520
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   4
      Left            =   840
      Shape           =   3  'Circle
      Top             =   2040
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   3
      Left            =   840
      Shape           =   3  'Circle
      Top             =   1560
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   2
```

```
            Left            =   840
            Shape           =   3  'Circle
            Top             =   1080
            Width           =   375
         End
         Begin VB.Shape shpLED
            BorderColor     =   &H000000FF&
            FillColor       =   &H000000FF&
            FillStyle       =   0  'Solid
            Height          =   375
            Index           =   1
            Left            =   840
            Shape           =   3  'Circle
            Top             =   600
            Width           =   375
         End
         Begin VB.Shape shpLED
            BorderColor     =   &H000000FF&
            FillColor       =   &H000000FF&
            FillStyle       =   0  'Solid
            Height          =   375
            Index           =   0
            Left            =   840
            Shape           =   3  'Circle
            Top             =   120
            Width           =   375
         End
      End
      Attribute VB_Name = "Form1"
      Attribute VB_GlobalNameSpace = False
      Attribute VB_Creatable = False
      Attribute VB_PredeclaredId = True
      Attribute VB_Exposed = False
      'IO.DLL
      Private Declare Sub PortOut Lib "IO.DLL" (ByVal Port
As Integer, ByVal Data As Byte)
```

```
    Private Declare Function PortIn Lib "IO.DLL" (ByVal
Port As Integer) As Byte
    'Variable
    Private BA_LPT As Integer

    Private Sub cmdExit_Click()
    End
    End Sub

    Private Sub cmdReceive_Click()
    Dim n As Integer
    Dim n1 As Integer
    Dim i As Integer

    PortOut BA_LPT + 2, &H8 'SELECTIN = 1
    PortOut BA_LPT + 2, 0 'SELECTIN = 0
    n1 = PortIn(BA_LPT + 1) 'Doc 4 bit thap
    n1 = n1 / &H10          'Dich phai 4 bit
    PortOut BA_LPT + 2, 2   'AUTOFEED=1
    n = PortIn(BA_LPT + 1)  'Doc 4 bit cao
    n = n And &HF0
    n = n + n1
        For i = 0 To 7
            chkSW(i).Value = n Mod 2
            If chkSW(i).Value = 0 Then
                chkSW(i).Caption = "Switch " & Str(i) &
" off"
            Else
                chkSW(i).Caption = "Switch " & Str(i) &
" on"
            End If
            n = Fix(n / 2)
        Next i
    End Sub

    Private Sub cmdSend_Click()
    Dim t As Integer
    Dim i As Integer
```

```
    Dim s As String
    t = 0
    For i = 0 To 7
       t = t + (2 ^ i) * (1 - shpLED(i).FillStyle)
    Next i
    PortOut BA_LPT, t
    PortOut BA_LPT, 1   'STROBE = 1
    PortOut BA_LPT, 0   'STROBE = 0
    End Sub

    Private Sub Form_Load()
    BA_LPT = &H378
    PortOut BA_LPT + 2, 0
    End Sub

    Private Sub lblLED_Click(Index As Integer)
    shpLED(Index).FillStyle = 1 -
shpLED(Index).FillStyle
    End Sub
```

LẬP TRÌNH HỆ THỐNG CHAT ĐƠN GIẢN BẰNG WINSOCK TRONG MÔI TRƯỜNG LẬP TRÌNH VISUAL C++

Bài thực hành 2

LẬP TRÌNH HỆ THỐNG CHAT ĐƠN GIẢN BẰNG WINSOCK TRONG MÔI TRƯỜNG LẬP TRÌNH VISUAL C++

Nội dung

4.1 Giới thiệu môi trường lập trình Visual C++ 6.0

4.2 Lập trình Winsock trong VC++

4.3 Thiết kế ứng dụng mạng MiniChat

4.4 Hiện thực chương trình MiniChatServer

4.5 Hiện thực chương trình MiniChatClient

Giới thiệu môi trường lập trình Visual C++ 6.0 (VC++)

- Là môi trường lập trình C++ cho phép thiết kế trực quan giao diện.
- Các ứng dụng được tổ chức theo dạng project, một project chứa các file khác nhau về mã chương trình, giao diện, các file header…
- Có nhiều loại ứng dụng trong VC++. Chương này giới thiệu về ứng dụng MFC

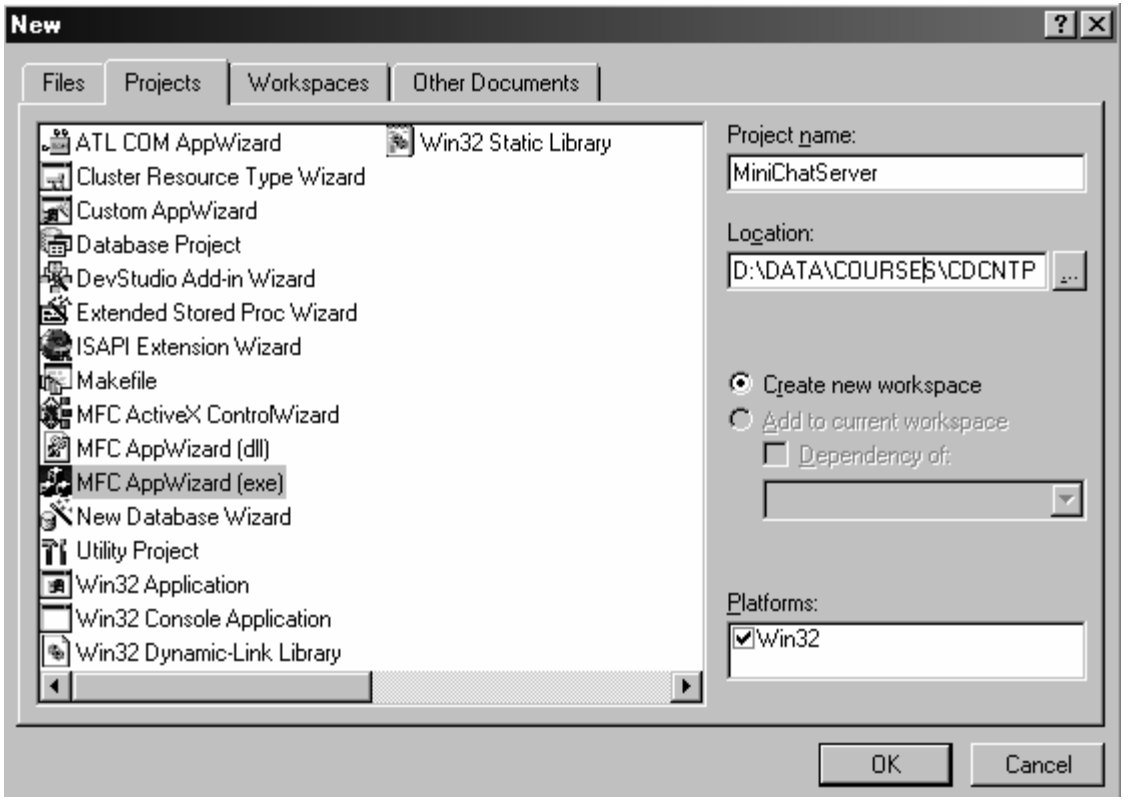
Tạo mới một project

Dùng menu File → New. Hộp thoại như bên dưới xuất hiện

Ở tag projecst, chọn loại ứng dụng là MFC AppWizard (exe).

Ở phần location, chọn thư mục để chứa project.

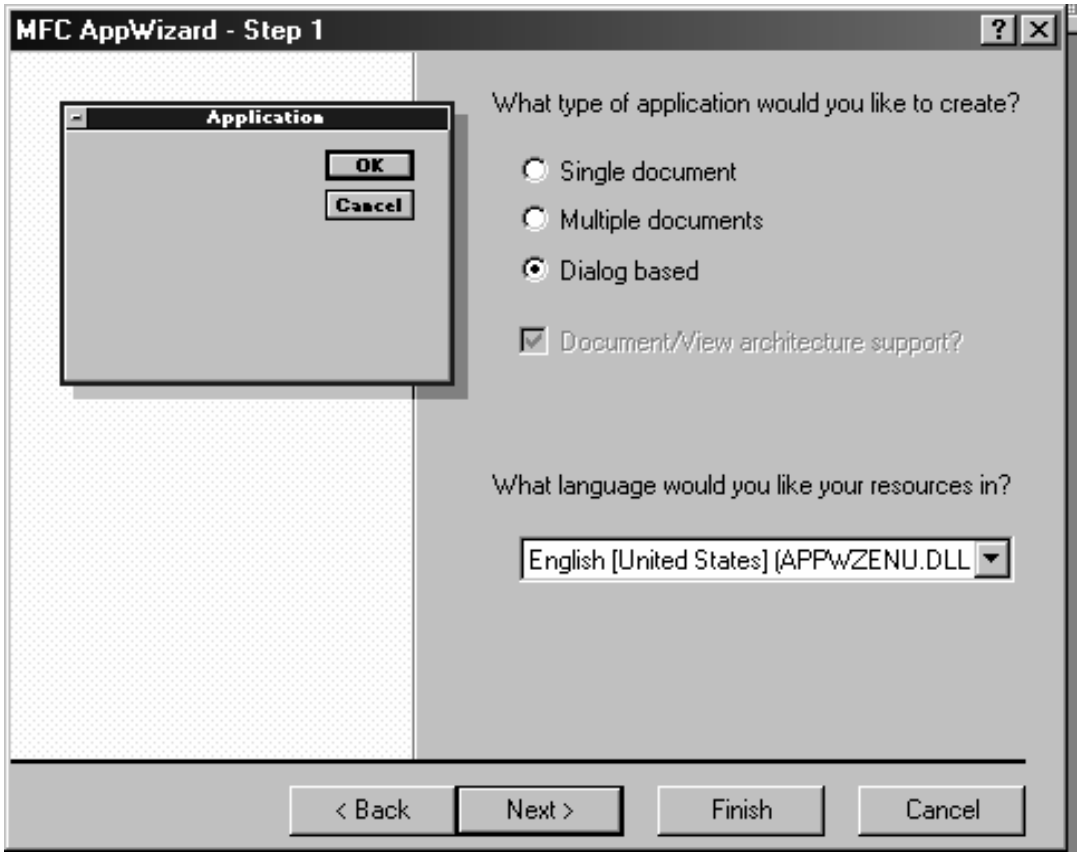
Gõ tên project và chọn OK



Hiệu chỉnh các thông số

Bước thứ nhất chọn loại ứng dụng, chọn dạng Dialog based như hình bên.

Nhấn button Next để tiếp tục

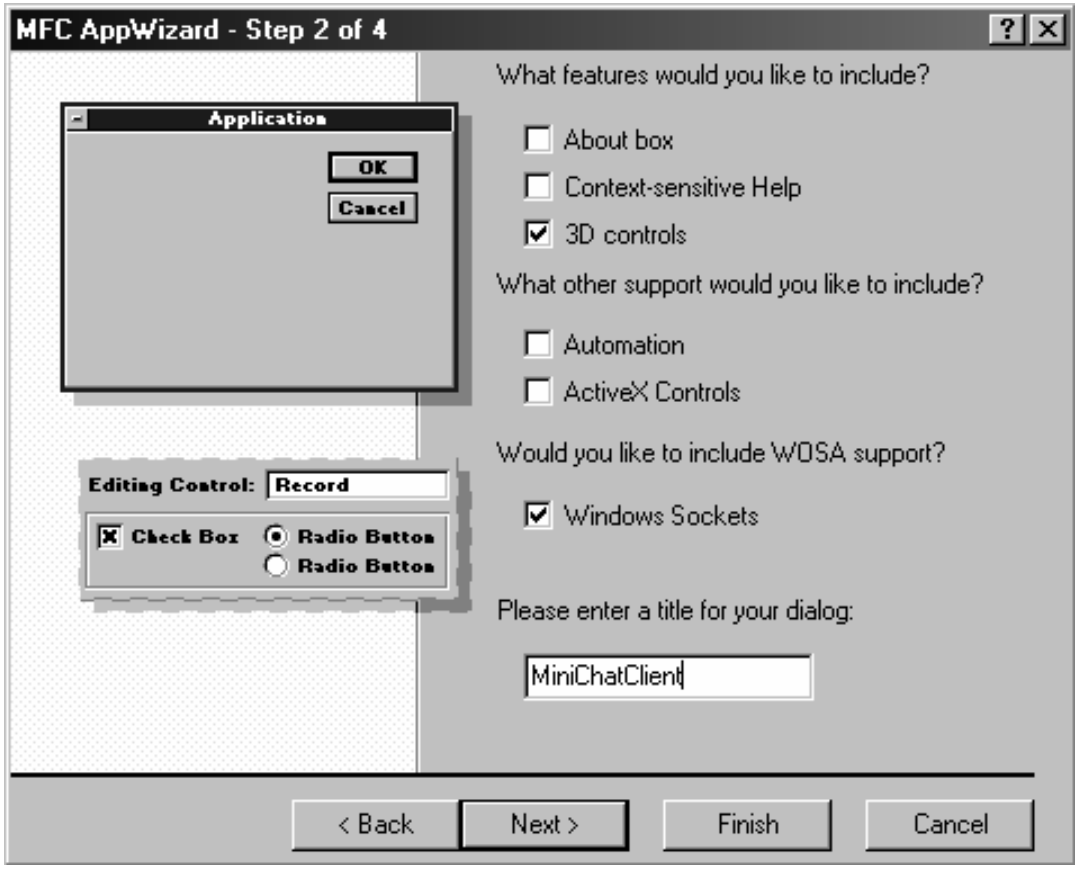


Hiệu chỉnh các thông số

Bước 2, chọn các đặc tính của ứng dụng như hình

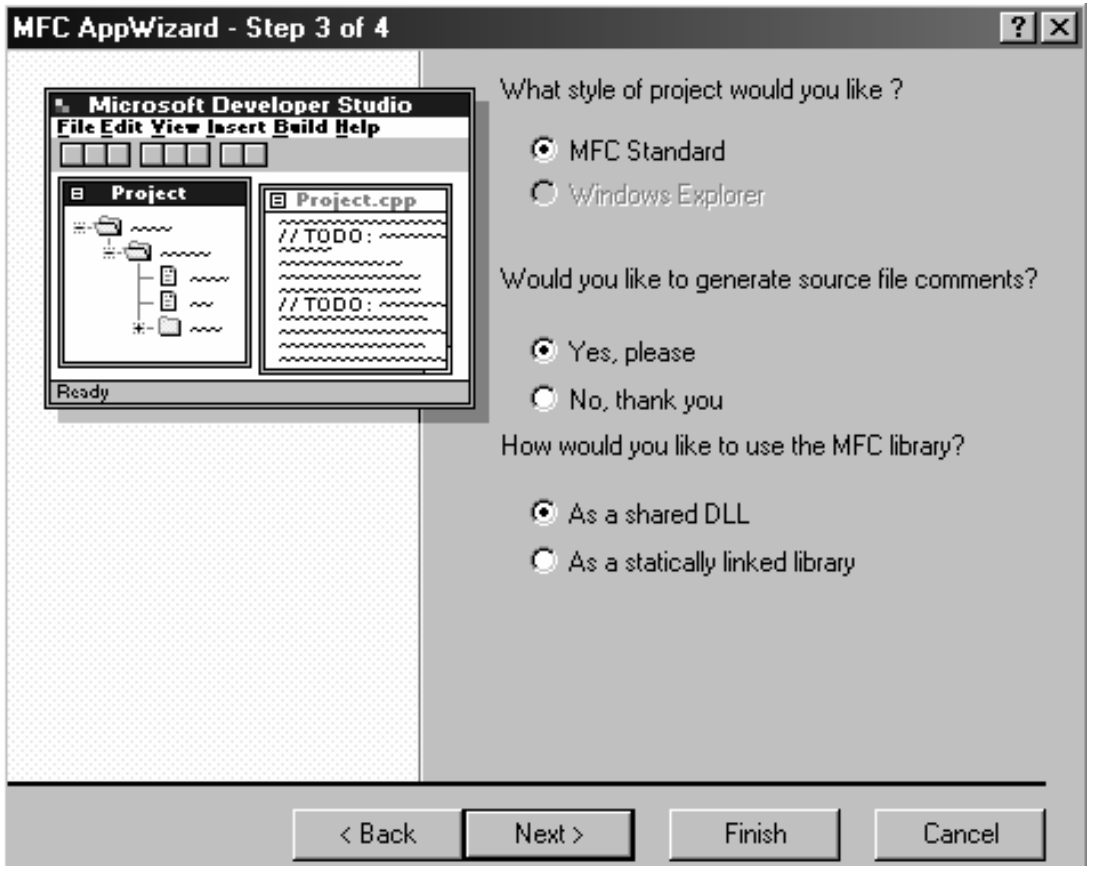
+ Phải chọn checkbox Windows Sockets

Nhấn button Next để tiếp tục



Hiệu chỉnh các thông số

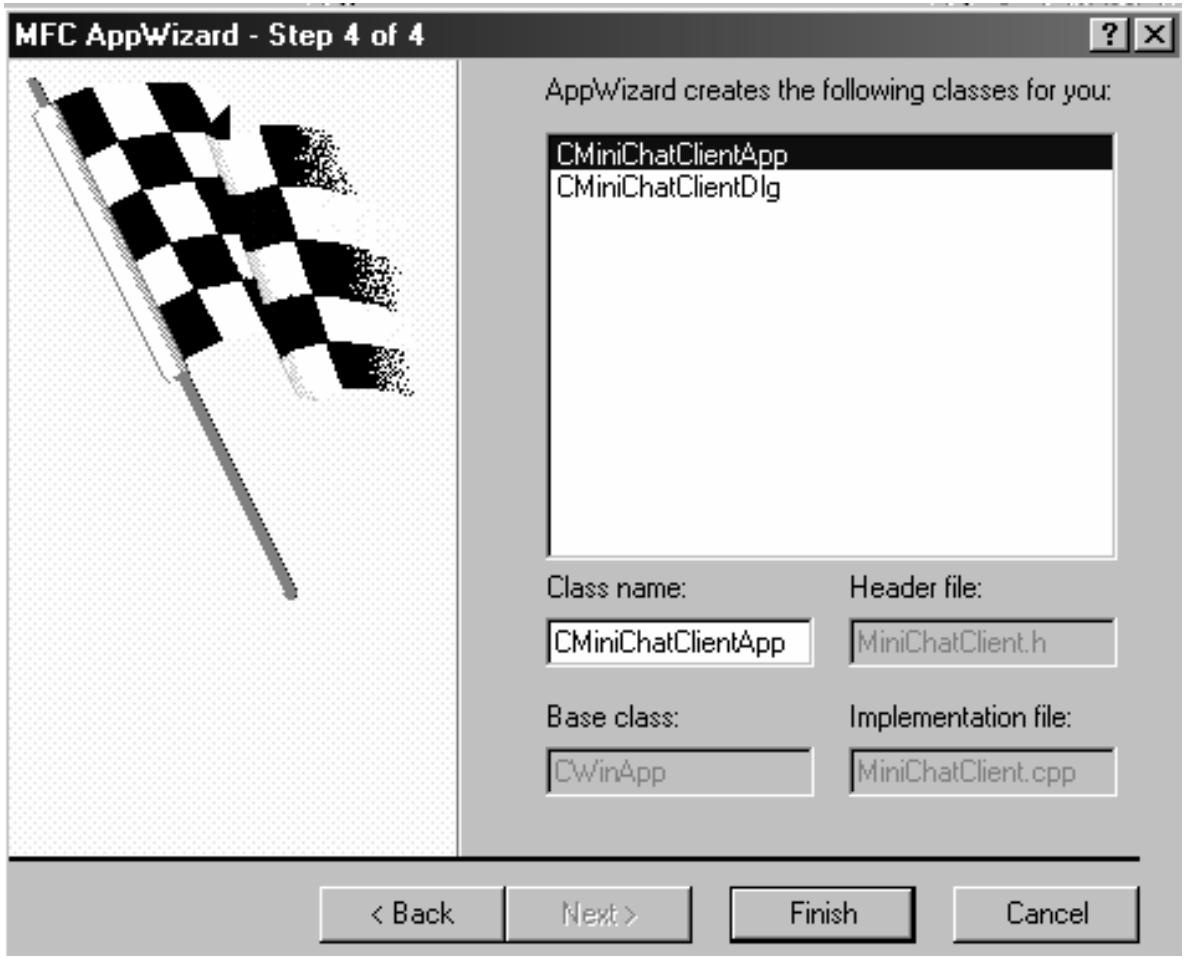
Bước 3, chọn các chức năng hỗ trợ như hình vẽ
Nhấn button Next
để tiếp tục



Hiệu chỉnh các thông số

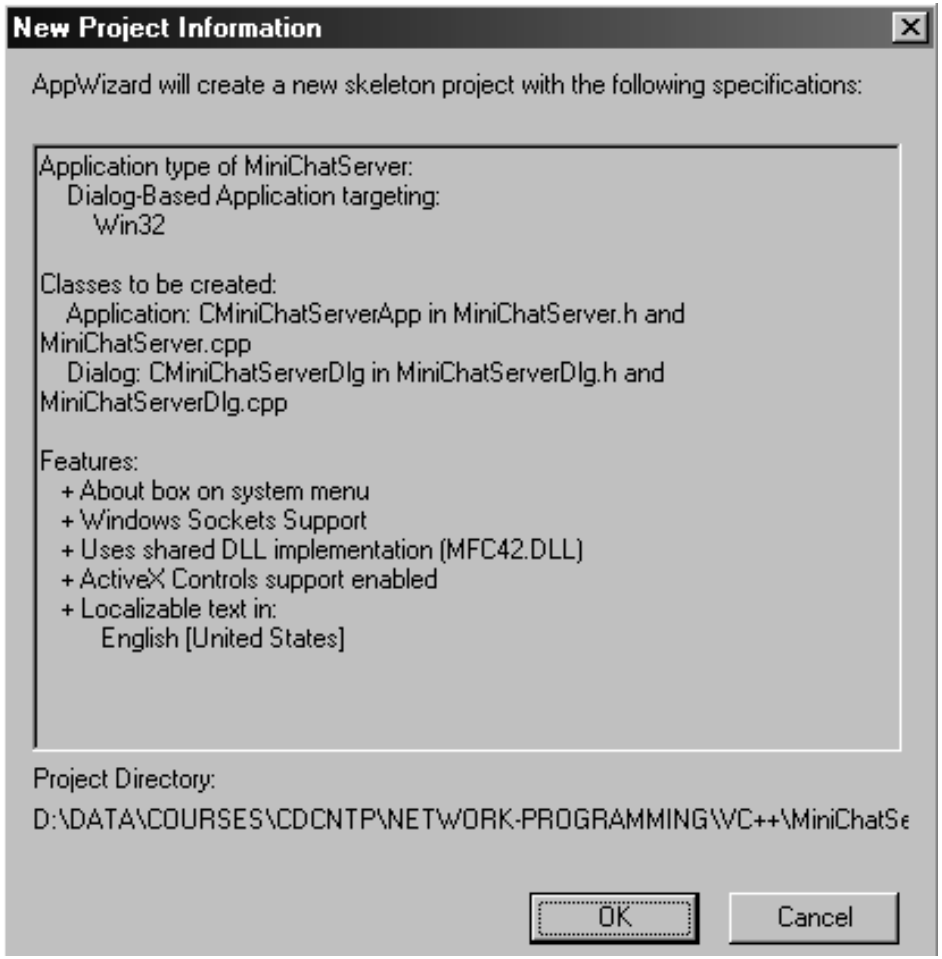
Bước 4: xác nhận các thông số đã chọn. Có thể qua lại các bước trước đó để hiệu chỉnh bằng button Back.

Chọn button Finish để kết thúc

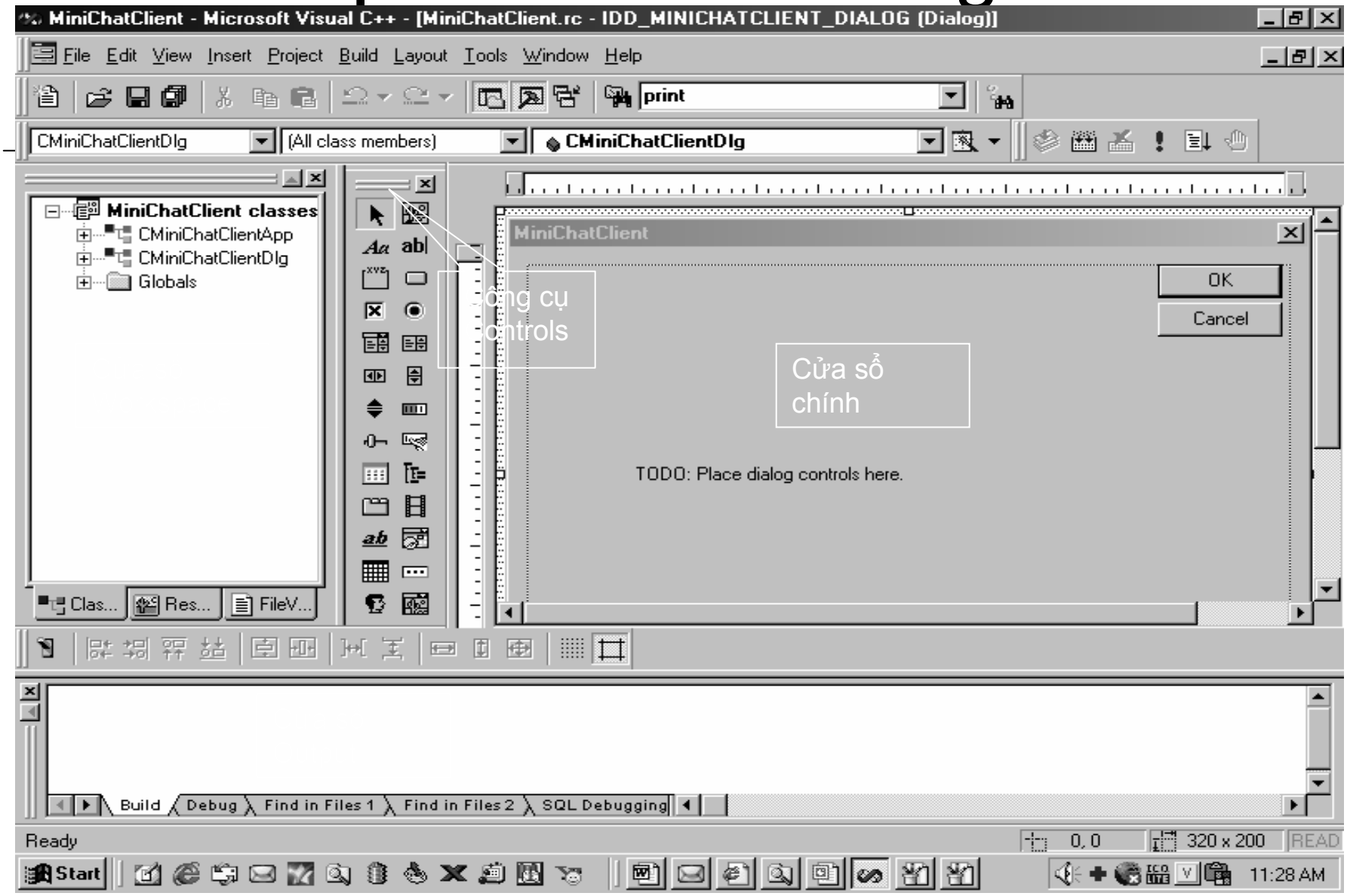


Hiệu chỉnh các thông số

Bước cuối cùng: xác nhận và chọn OK để bắt đầu lập trình

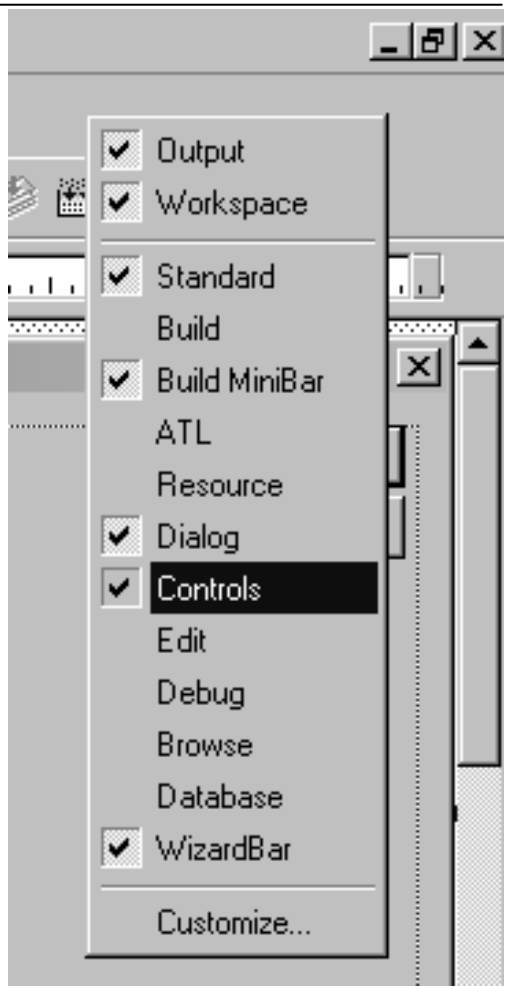


Giao diện của môi trường VC++



Thiết kế giao diện

- Để thiết kế giao diện, ta cần dùng thanh công cụ Controls (right-click vào các thanh công cụ, chọn Controls như hình bên)
- Các đối tượng giao diện thường dùng:
  - Static Text
  - Edit Box
  - Button
  - Listbox



Vẽ các đối tượng giao diện

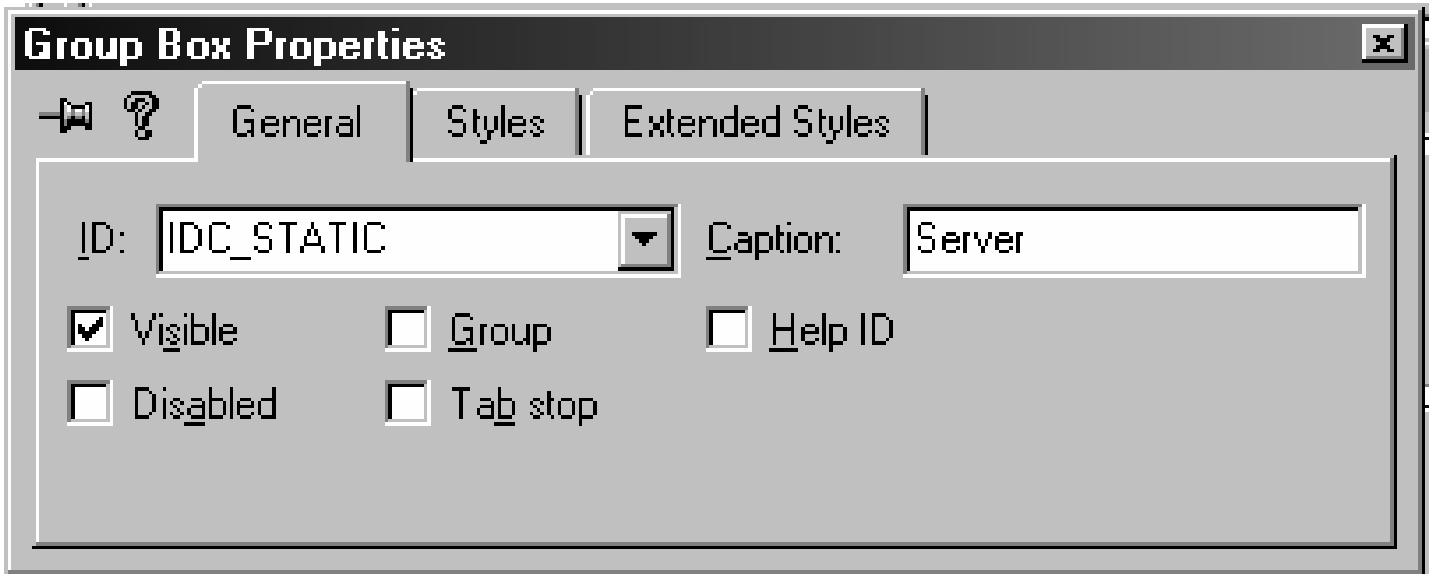
- Mở Dialog cần vẽ các đối tượng giao diện (Ở cửa sổ Workspace, chọn chế độ ResourseView, click chọn thư mục dialog, chọn Dialog tương ứng)
- Muốn vẽ đối tượng giao diện nào click vào đối tượng giao diện đó, đưa trỏ chuột vào Dialog để vẽ (dùng cơ chế Drag chuột, vừa nhấn chuột trái vừa kéo)

Thiết lập thuộc tính cho các đối tượng giao diện

- Right-click vào đối tượng giao diện và chọn Properties
- ID là thuộc tính tên nhận dạng của đối tượng giao diện
- Tuỳ mỗi loại đối tượng giao diện có các thuộc tính riêng

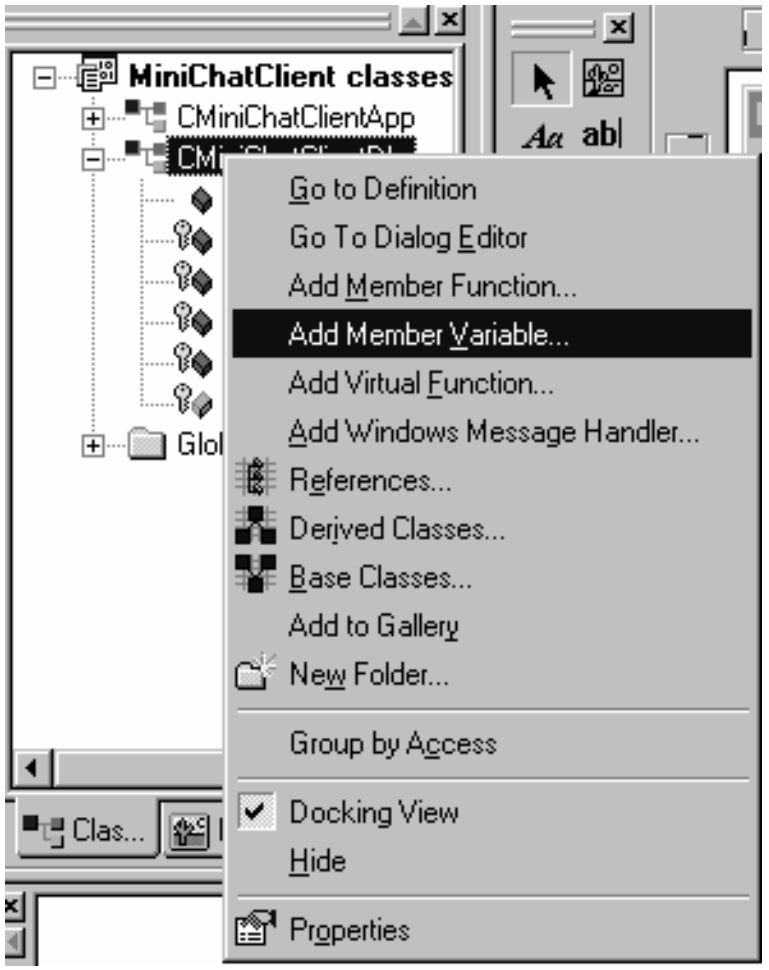
Thiết lập thuộc tính cho các đối tượng giao diện

- Thiết lập caption (Nội dung hiển thị lên phần tử giao diện) cho đối tượng giao diện Button và Static Text như hình bên dưới



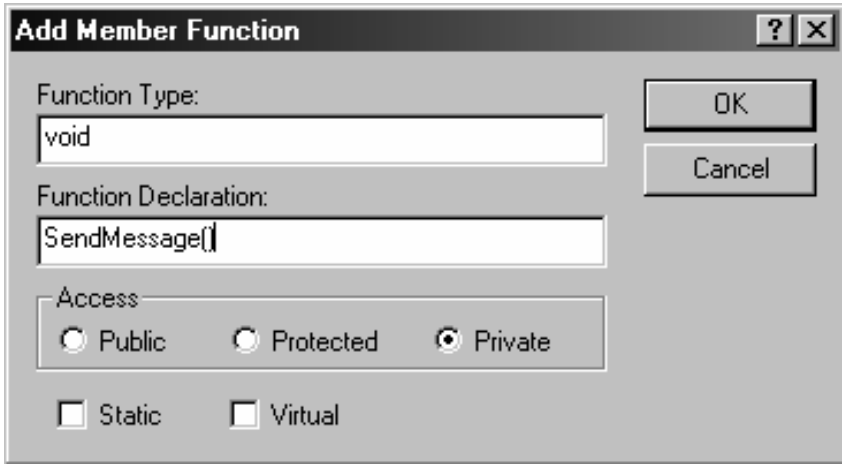
Khai báo biến và định nghĩa hàm

- Trong cửa sổ workspace, chọn tab ClassView, right-click vào class C*Dlg, menu hiển thị như hình vẽ bên
- Chọn chức năng Add Member Variable
- Chức năng này cũng dùng tương tự cho việc định nghĩa hàm



Khai báo biến và định nghĩa hàm

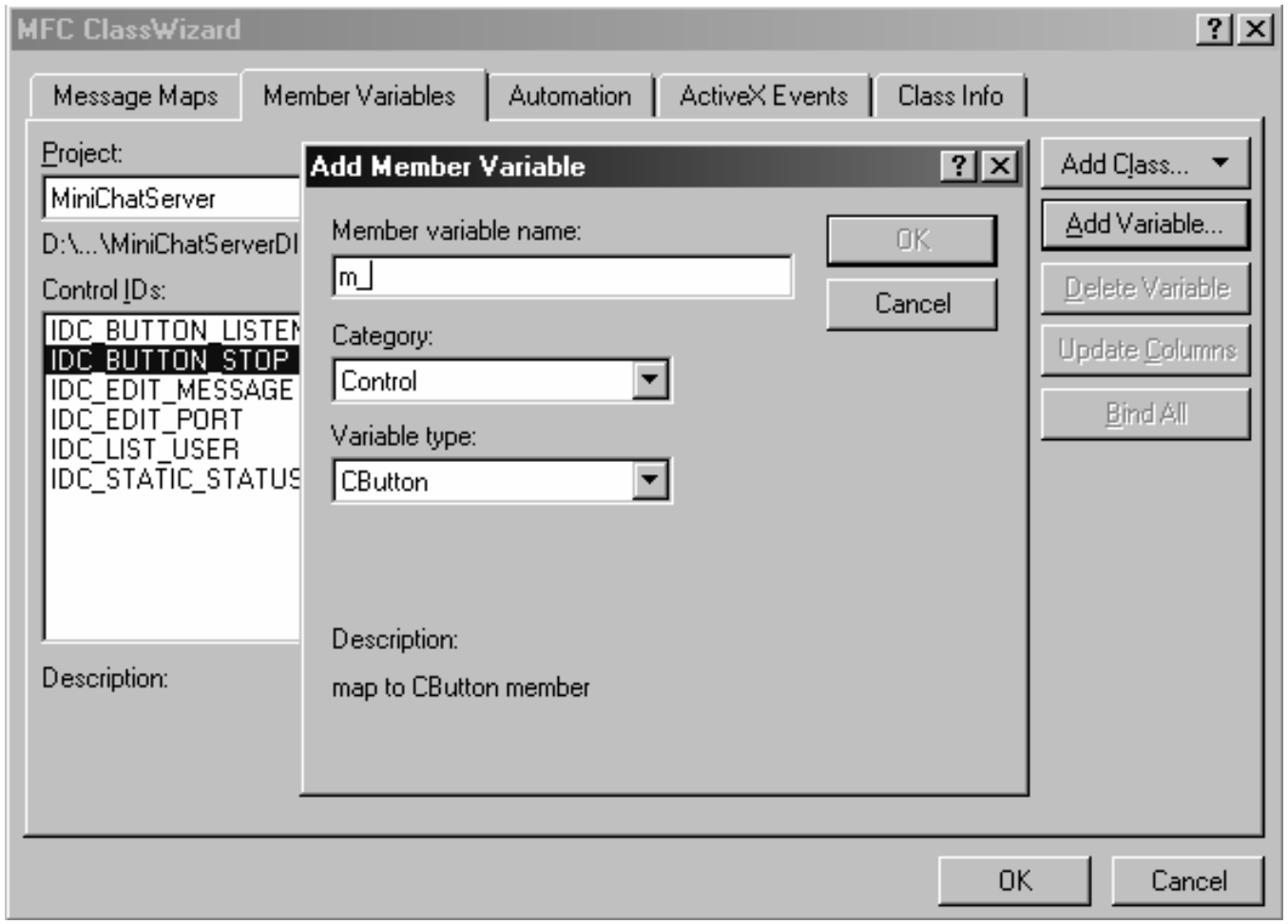
- Khai báo biến như hình trên: đánh kiểu biến, tên biến và tầm vực của biến rồi nhấn OK
- Định nghĩa hàm như hình bên dưới: kiểu trả về, tên hàm và các thông số, tầm vực truy xuất



Gán biến cho đối tượng giao diện

- Mỗi đối tượng giao diện đều có thể truy xuất thông qua biến được định nghĩa
- Chọn menu View -> ClassWinzard -> Member Variables
- Chọn đối tượng giao diện tương ứng (nhờ vào ID đã đặt), click button Add Variable)
- Đặt tên biến, loại biến (Control hoặc Value) và kiểu dữ liệu

Gán biến cho đối tượng giao diện



Thiết lập - lấy giá trị phần tử giao diện Edit Box và Static Text

- Thiết lập:
  - Gán giá trị cho biến tương ứng.
  - Dùng lệnh: UpdateData(FALSE);
- Lấy giá trị:
  - Dùng lệnh: UpdateData(TRUE);
  - Giá trị được truyền cho biến tương ứng của phần tử giao diện

Ví dụ:

```
m_mes=m_mes+"Accepted a connection!\r\n";
UpdateData(FALSE);
```

Thêm - loại giá trị cho phần tử giao diện Listbox

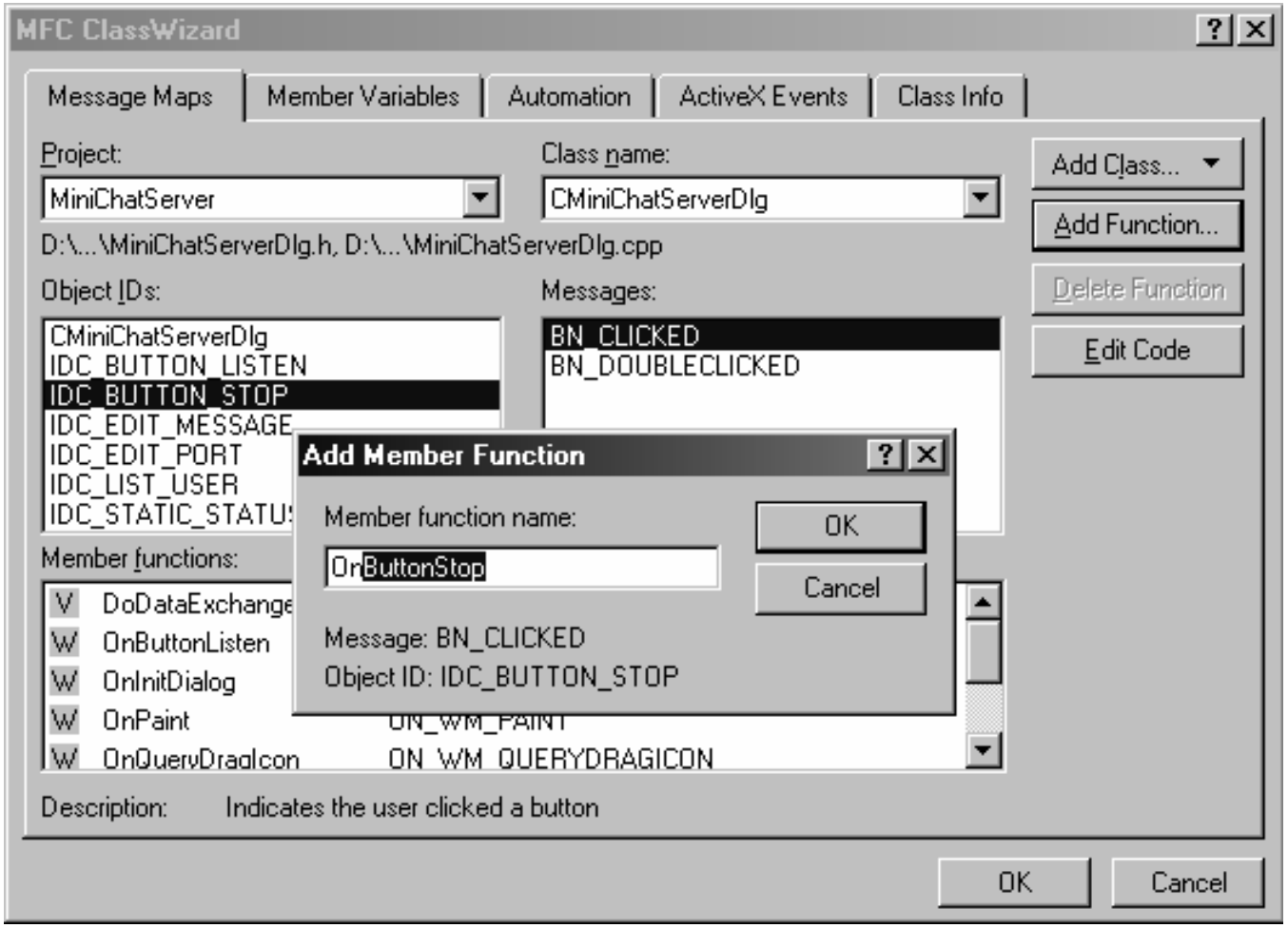
- Thêm vào ListBox:
  - Dùng phương thức AddString(String) của đối tượng điều khiển ListBox
- Loại phần tử ra khỏi ListBox:
  - Dùng phương thức RemoveString(int index) của đối tượng điều khiển
- Lấy index của một phần tử nào, ta cần phải quản lý danh sách của Listbox

Tạo hàm xử lý sự kiện cho button

- Khi người sử dụng click chuột vào button nào trên giao diện, hệ thống sẽ sinh ra sự kiện BN_CLICKED cho đối tượng đó.
- Người lập trình phải viết mã để xử lý sự kiện đó.
- Để tạo hàm xử lý sự kiện, ta có thể double-click vào button, VC++ sẽ đề nghị tên hàm, nhấn OK để bắt đầu viết mã

Tạo hàm xử lý sự kiện cho button

Có thể dùng ClassWinzard, chọn đối tượng Button, chọn message BN_CLICKED và nhấn Add Function



Lập trình Winsock trong VC++

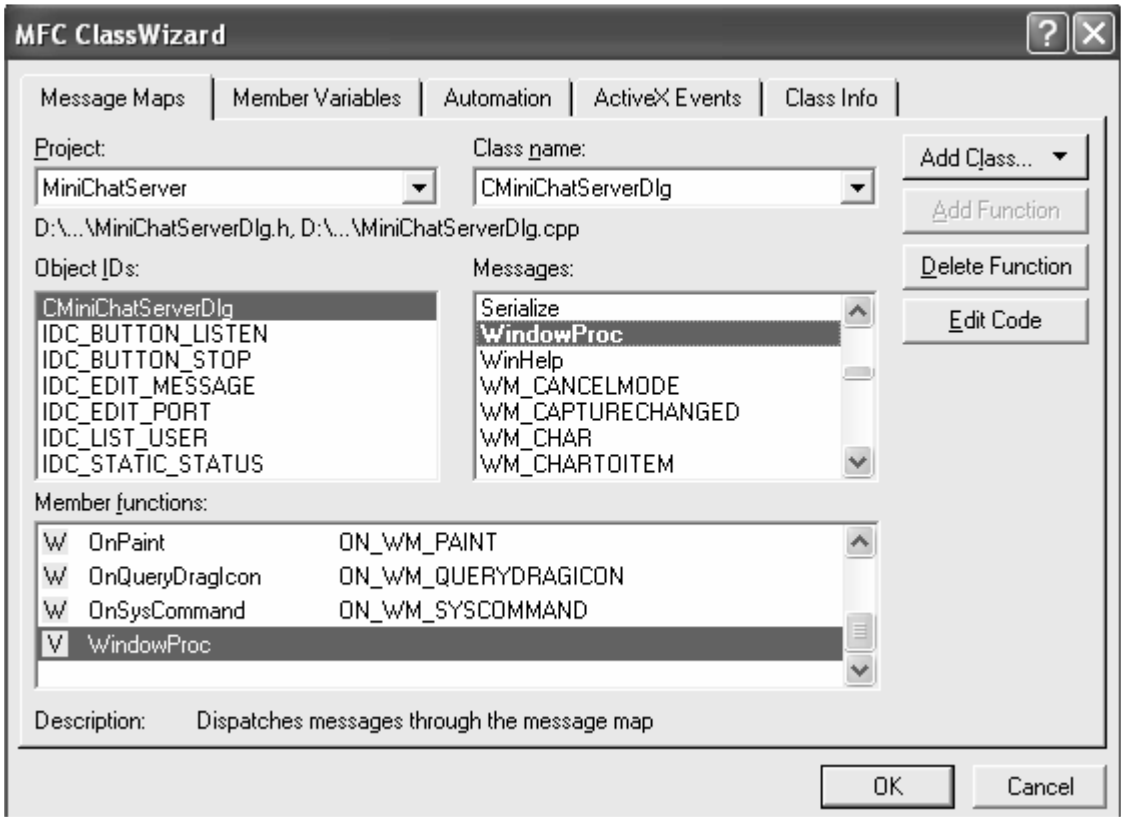
- Phân tích và thiết kế giao diện cần thiết cho ứng dụng mạng.
- Thiết lập các biến dữ liệu tương ứng với các phần tử giao diện Listbox, Edit box, Static Text
- Đặt các biến dùng cho lập trình socket
- Viết mã lệnh trình tự các hàm như đã trình bày ở chương 3

Lập trình Winsock trong VC++

- Hàm được gọi đầu tiên khi ứng dụng khởi tạo là OnInitDialog(): chúng ta có thể viết hàm khởi tạo socket, bind, listen, accept trong hàm này
- Có thể tạo các button để xử lý gọi các hàm nêu trên.
- Nên tạo các hàm để xử lý sự kiện và các lệnh tương ứng

Lập trình Winsock trong VC++

- Tạo hàm **WindowProc** để xử lý các sự kiện mạng:
  - Cửa sổ ClassWinzard, chọn ID là C*Dlg, Messages là WindowProc và click button Add Function
  - Click button Edit code để viết mã



Lập trình Winsock trong VC++

- Một số messages cần quan tâm khi lập trình mạng trong VC++
  - WM_CLOSE: xảy ra khi người sử dụng đóng chương trình
  - WM_KEYUP: xảy ra khi người sử dụng nhả một phím, có thể dùng để detect phím Enter
- Việc sử lý các messages này cũng cần phải tạo hàm xử lý tương ứng như slide trước

Thiết kế ứng dụng mạng MiniChat

- Ứng dụng MiniChat có hai chương trình MiniChatServer và MiniChatClient
- Trong hệ thống, chỉ có một chương trình server và nhiều chương trình client đang chạy.
- Chương trình client gởi dữ liệu đến cho chương trình server để yêu cầu thông tin hoặc gởi thông tin => Định nghĩa các loại dữ liệu gởi

Định nghĩa các loại dữ liệu gởi

- Dữ liệu gởi của client cho server:
  - Tham gia vào chat room: LOGIN:nickname*
    - nickname là tên của người sử dụng dùng để chat, không được trùng với các nickname khác
  - Gởi message cho toàn bộ chat room: PUBLIC: nicknamesender:message*
  - Gởi message cho riêng một user: PRIVATE:nicknamesender:nicknamereciever: message*
  - Thoát khỏi chat room: QUIT*

Định nghĩa các loại dữ liệu gởi

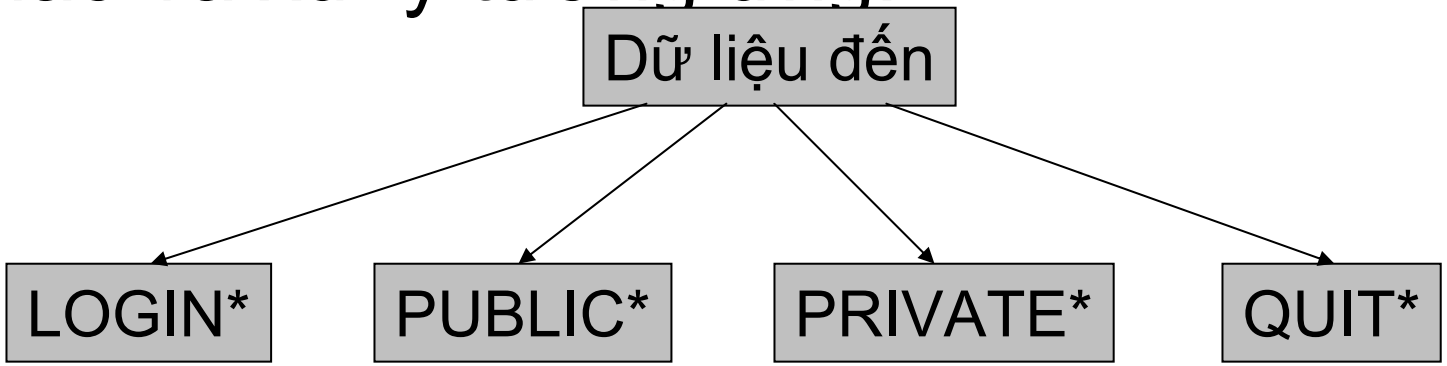
- Dữ liệu gởi từ server cho client:
  - Danh sách các user (nickname) có trong chatroom: LIST[:nickname]+*
    - Ví dụ: LIST:cuc:mai:lan:dao*
  - Message cho toàn bộ user: PUBLIC:nicknamesender:message*
  - Message cho riêng một user: PRIVATE:nicknamesender:message*

Định nghĩa các loại dữ liệu gởi

- Dữ liệu gởi từ server cho client:
  - Một user login vào: USERL:nickname*
  - Đã xử lý yêu cầu đúng: +OK
  - Các lỗi:
    - -100: Unknown command
    - -101: Not login
    - -102: Nickname existed
    - -103: Nickname not exist
    - -104: Cannot send the message
    - -105: Not accept null nickname
    - -106: Login already

Thiết kế sơ đồ chức năng của ứng dụng MiniChatServer

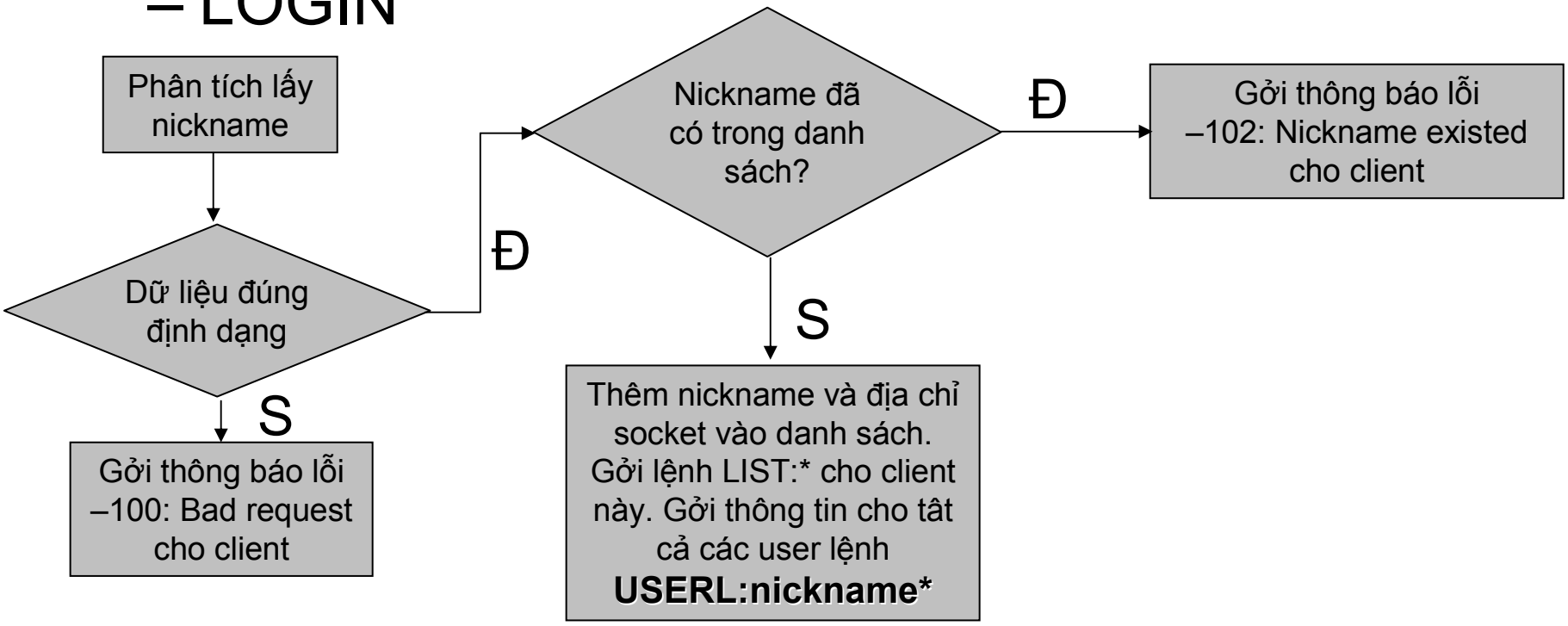
- Chương trình **server** mở socket và lắng nghe kết nối từ các client. Khi có dữ liệu đến, server phân tích dữ liệu thuộc dạng nào và xử lý tương ứng:



  - Nếu không thuộc các định dạng trên thì gởi lện **–100: Bad request** cho client

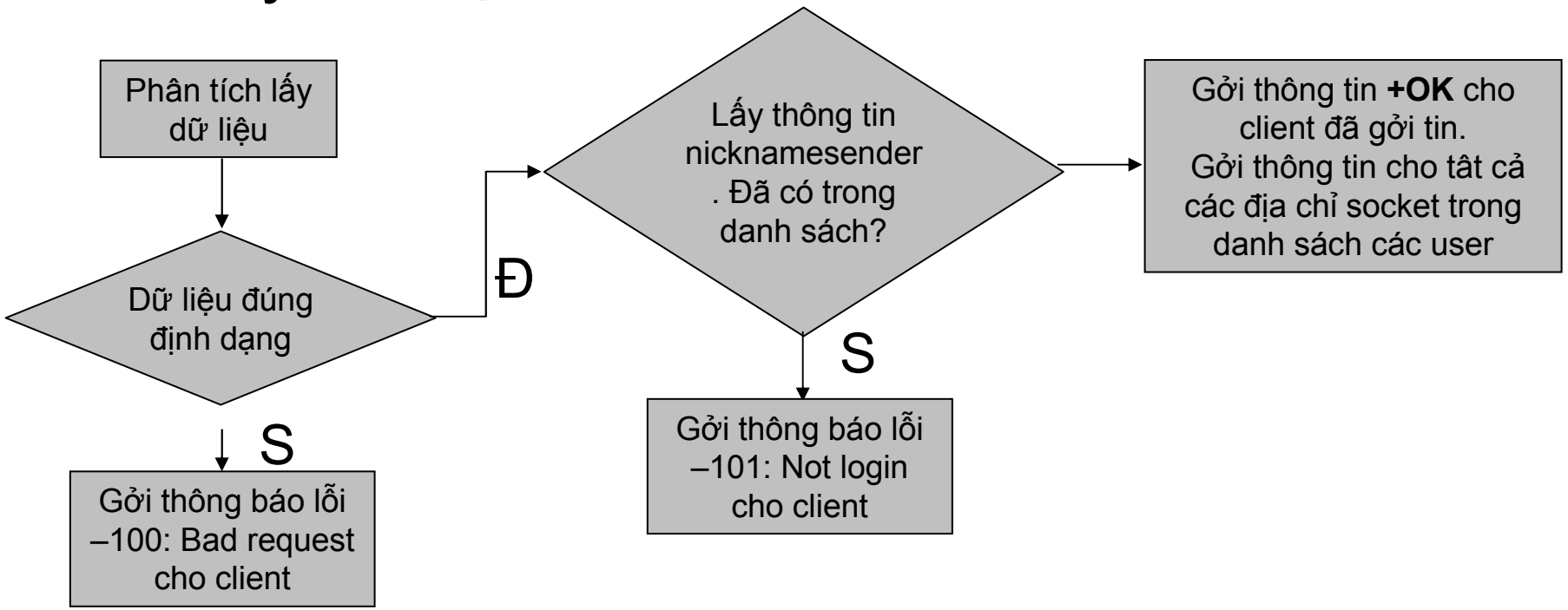
Thiết kế sơ đồ chức năng của ứng dụng MiniChatServer

- Với mỗi loại dữ liệu sẽ xử lý tương ứng:
  - LOGIN*



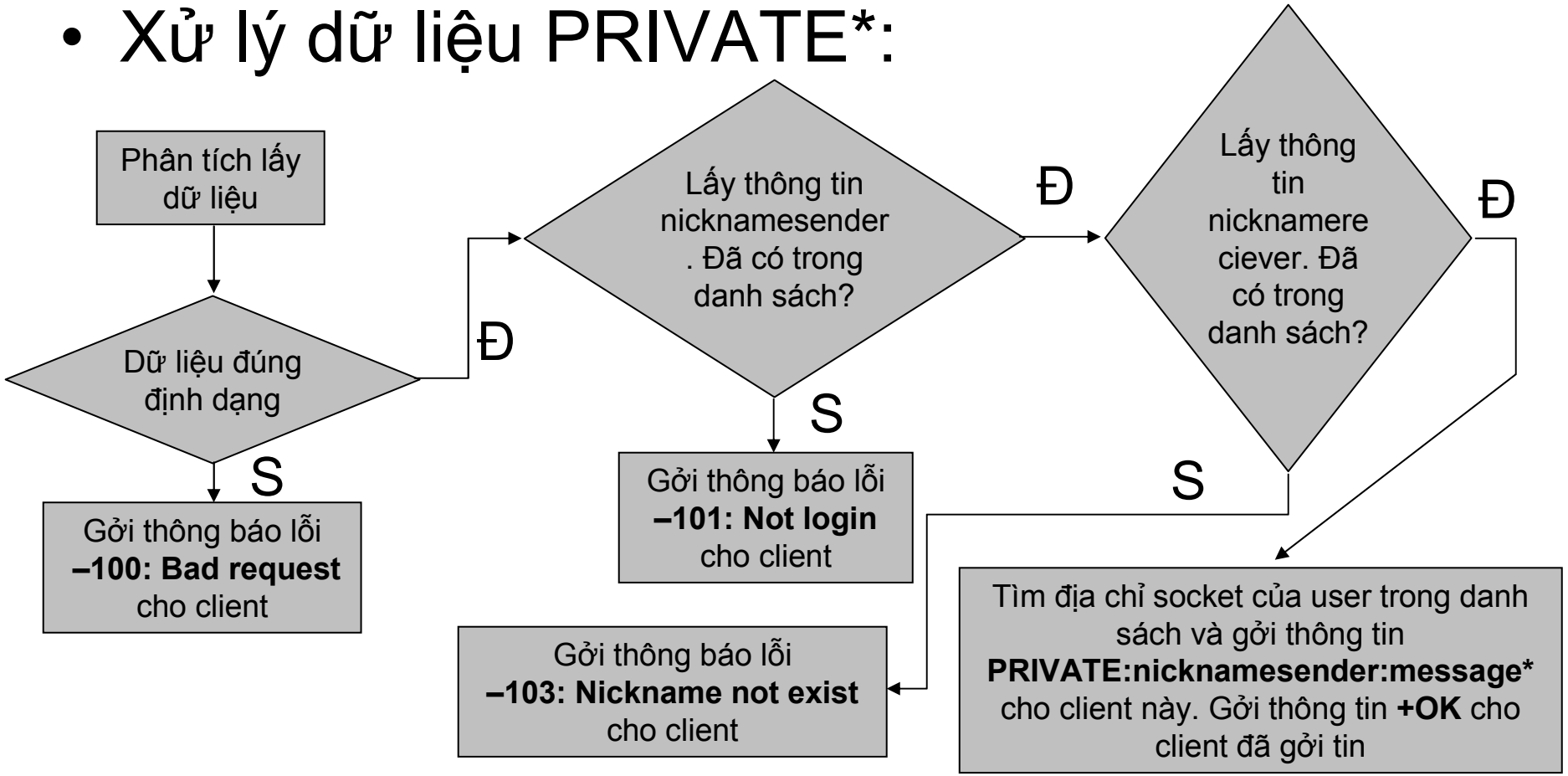
Thiết kế sơ đồ chức năng của ứng dụng MiniChatServer

- Xử lý dữ liệu PUBLIC*:



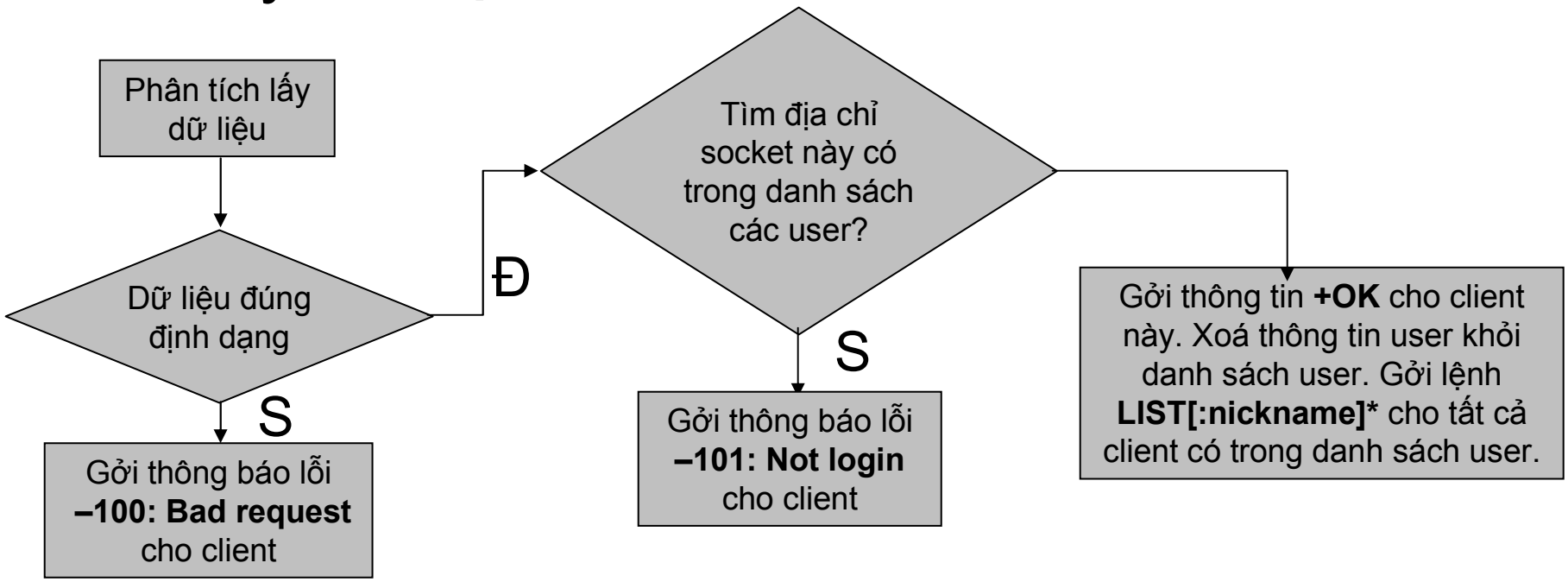
Thiết kế sơ đồ chức năng của ứng dụng MiniChatServer

- Xử lý dữ liệu PRIVATE*:



Thiết kế sơ đồ chức năng của ứng dụng MiniChatServer

- Xử lý dữ liệu QUIT*:



Thiết kế sơ đồ chức năng của ứng dụng MiniChatClient

- Chương trình **client** tạo socket, lấy các thông tin từ giao diện của người sử dụng để kết nối đến server.
- Nếu kết nối thành công, lấy thông tin nickname để gởi dữ liệu LOGIN:nickname* đến server và chờ nhận dữ liệu về. Dữ liệu về có hai dạng:
  - nếu bắt đầu bằng ký hiệu '-' có nghĩa là bị lỗi, phân tích lỗi tương ứng để thông báo cho user
  - Nếu là LIST* có nghĩa là đăng nhập thành công, phân tích danh sách các nickname để hiển thị cho user

Thiết kế sơ đồ chức năng của ứng dụng MiniChatClient

- User gởi message vào chat room, có hai dạng:
  - Chỉ gởi cho một user: tạo lệnh PRIVATE* và gởi cho server. Chờ nhận dữ liệu về:
    - +OK: tiếp tục các quá trình khác
    - -xxx: lỗi giao thức, phân tích lỗi và thông báo
  - Gởi cho toàn bộ chat room: tạo lệnh PUBLIC* và gởi cho server. Chờ nhận dữ liệu về:
    - +OK: tiếp tục quá trình khác
    - -xxx: lỗi giao thức, phân tích lỗi và thông báo

Thiết kế sơ đồ chức năng của ứng dụng MiniChatClient

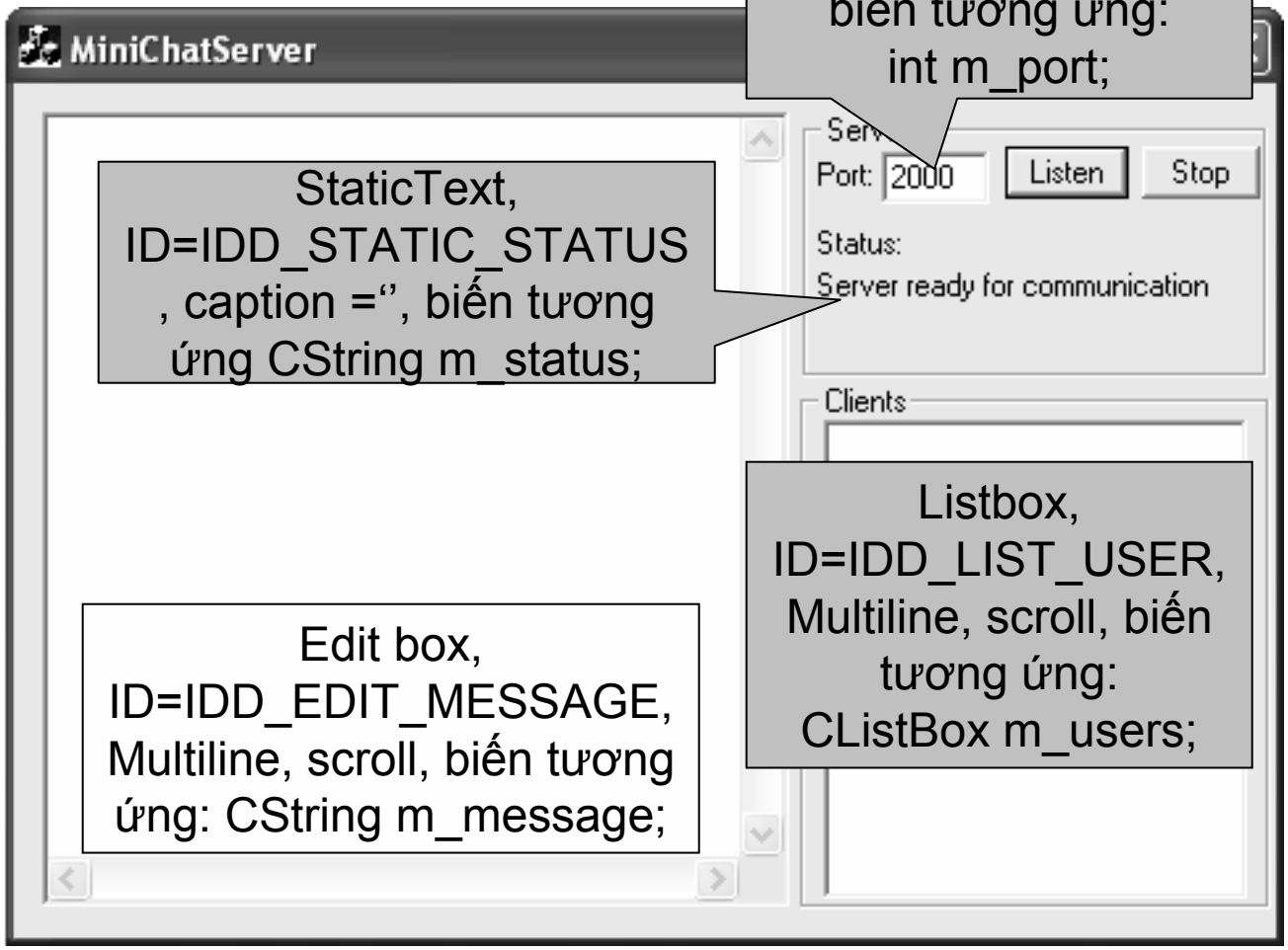
- User thoát khỏi chat room hoặc tắt chương trình
  - Gởi lệnh QUIT* cho server và chờ nhận dữ liệu về:
    - +OK: tiếp tục các quá trình khác
    - -xxx: lỗi giao thức, phân tích lỗi và thông báo
- Nhận dữ liệu bất kỳ từ server: phân tích dữ liệu thuộc một trong các dạng: USERL*, USERQ*, PRIVATE*, PUBLIC*, nếu không thuộc các dạng này thì thông báo lỗi. Đối với mỗi dạng dữ liệu sẽ được sử lý như slide kế

Thiết kế sơ đồ chức năng của ứng dụng MiniChatClient

- Nếu là lệnh USERL*:
  - Phân tích lấy nickname
  - Thêm nickname vào danh sách các user
- Nếu là lệnh USERQ*:
  - Phân tích lấy nickname
  - Xoá nickname khỏi danh sách các user, nếu có lỗi thì báo lỗi cho user
- Nếu là lệnh PRIVATE* hoặc PUBLIC*
  - Hiển thị thông tin này cho user

Hiện thực chương trình MiniChatServer

Thiết kế giao diện như hình bên và đặt các biến tương ứng cho các phần tử giao diện



Hiện thực chương trình MiniChatServer

- Định nghĩa kiểu dữ liệu record để lưu danh sách các user (đầu file C*Dlg.h):

```
typedef struct T_UserRecord {
    char name[20];
    SOCKET socket;
    int state;
    struct T_UserRecord* next;
} T_UserRecord;
```

- Khai báo các biến: (dùng chức năng Add Member Variable)

```
SOCKET ServerSocket; char temp_message[128];
T_UserRecord *UserRecordList; T_UserRecord *tempUserRecord;
```

Hiện thực chương trình MiniChatServer

- Khai báo các hằng số (file Resource.h)
  - #define MSG_LENGTH 256
  - #define WSA_ACCEPT 1006
  - #define WSA_RDCLOSE 1007
  - #define CONNECTED 2000
  - #define LOGIN 2001
  - #define CHAT 2002
- Lập trình theo các bước sau:
  - Tạo hàm xử lý sự kiện khi người dùng click chuột vào Button Listen, hàm OnButtonListen(): lần lượt gọi các lệnh socket, bind, listen, và WSASyncSelect

Hiện thực chương trình MiniChatServer

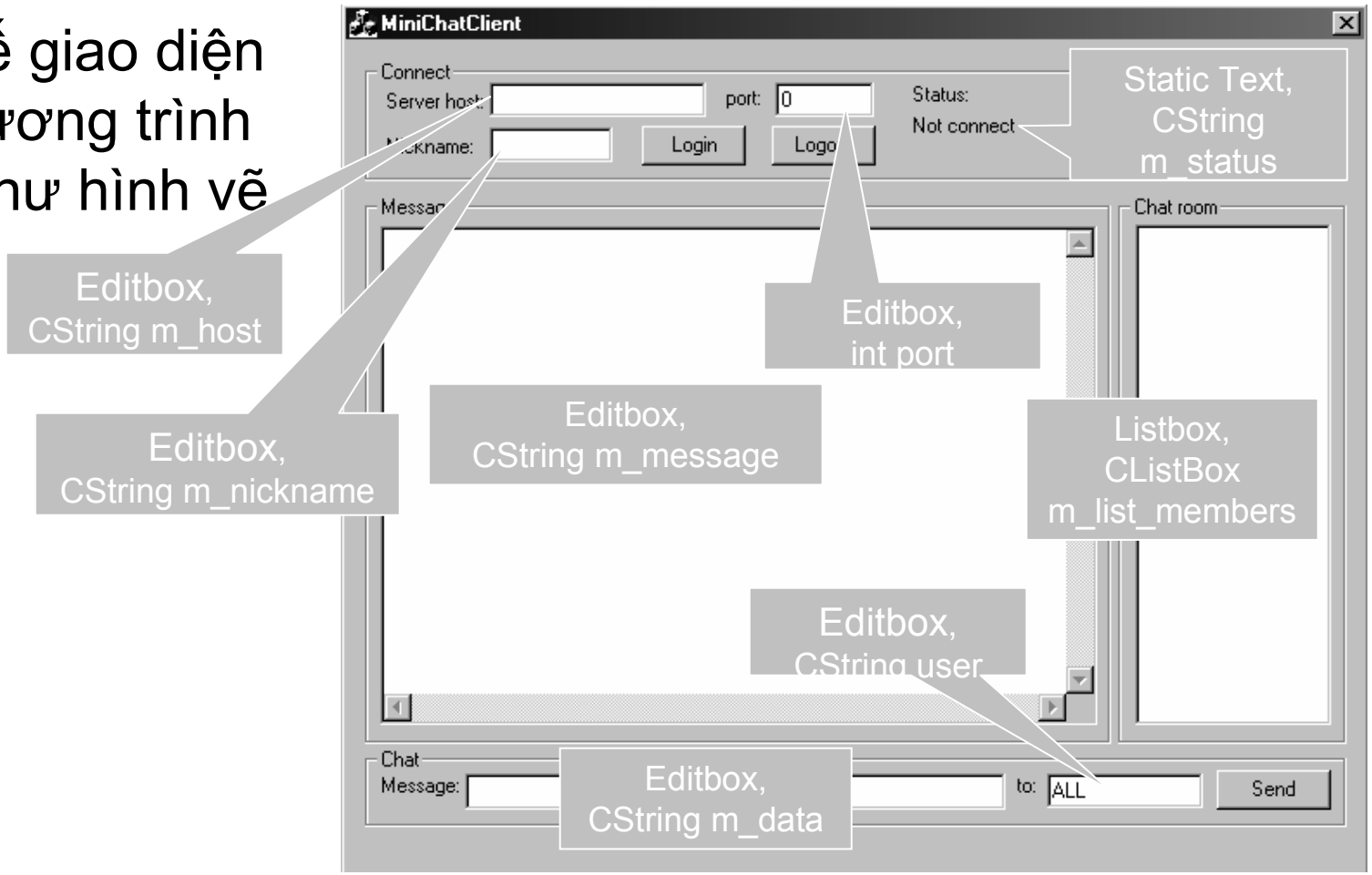
- Trình tự:
  - Tạo hàm WindowProc() và viết mã lệnh:
    - Tùy theo loại message sẽ gọi hàm tương ứng để xử lý (hai loại message định nghĩa là WSA_ACCEPT, WSA_RDCLOSE).
  - Định nghĩa và viết mã lệnh hàm OnAccept(…): xử lý sự kiện FD_ACCEPT, message WSA_ACCEPT khi có yêu cầu kết nối
  - Định nghĩa và viết mã lệnh hàm Process(…): xử lý sự kiện FD_READ, message WSA_RDCLOSE khi có dữ liệu gởi đến từ client

Hiện thực chương trình MiniChatServer

- Trình tự:
  - Trong hàm Process(): phân tích định dạng của dữ liệu đến xử lý tương ứng:
    - LOGIN*: hàm Login(…)
    - PUBLIC*: hàm SendPublic(…)
    - PRIVATE*: hàm SendPrivate(…)
    - QUIT*: hàm Logout(…)
  - Định nghĩa và viết mã lệnh cho lần lượt các hàm trên theo sơ đồ khối đã thiết kế

Hiện thực chương trình MiniChatClient

Thiết kế giao diện của chương trình client như hình vẽ



Hiện thực chương trình MiniChatClient

- Khai báo các biến: (dùng chức năng Add Member Variable)

```
SOCKET ClientSocket;
char temp_message[128];
int chat_status;
CString data;
```

- Khai báo các hằng số (file Resource.h)
  - #define NOTLOGIN 2000
  - #define LOGIN 2001
  - #define QUIT 2002
  - #define CHAT 2003
  - #define WSA_RDREAD 3000

Hiện thực chương trình MiniChatClient

- Lập trình theo các bước sau:
  - Tạo hàm xử lý sự kiện khi người dùng click chuột vào Button Login, hàm OnButtonLogin(): lần lượt gọi các lệnh socket, connect, và WSASyncSelect để chờ nhận sự kiện mạng
  - Tạo hàm WindowProc() và viết mã lệnh:
    - Chương trình client chỉ có một message (WSA_RDREAD) cho hai sự kiện FR_READ và FD_CLOSE, với mỗi sự kiện ta thực hiện lệnh gọi hàm tương ứng
  - Định nghĩa và viết mã lệnh hàm Process(…): xử lý sự kiện FD_READ, message WSA_RDREAD khi có dữ liệu từ server gởi đến

Hiện thực chương trình MiniChatClient

- Trình tự:
  - Định nghĩa và viết mã lệnh lần lượt các hàm Login, ResponseLogin, Send, Communicate, DisplayUserList
  - Tạo hàm xử lý sự kiện :OnSelchangeListMember, OnButtonSend, OnButtonLogout