CHƯƠNG 1: ĐẠI CƯƠNG

1. Các hệ thống số dùng trong máy tính và các loại mã

1.1. Hệ thập phân (Decimal Number System)

Trong thực tế, ta thường dùng hệ thập phân để biểu diễn các giá trị số. Ở hệ thống này, ta dùng các tổ hợp của các chữ số 0..9 để biểu diễn các giá trị. Một số trong hệ thập phân được biểu diễn theo các số mũ của 10.

VD: Số 5346,72 biểu diễn như sau:
$5346,72 = 5.10^3 + 3.10^2 + 4.10 + 6 + 7.10^{-1} + 2.10^{-2}$

Tuy nhiên, trong các mạch điện tử, việc lưu trữ và phân biệt 10 mức điện áp khác nhau rất khó khăn nhưng việc phân biệt hai mức điện áp thì lại dễ dàng. Do đó, người ta sử dụng hệ nhị phân để biểu diễn các giá trị trong hệ thống số.

1.2. Hệ nhị phân (Binary Number System)

Hệ nhị phân chỉ dùng các chữ số 0 và 1 để biểu diễn các giá trị số. Một số nhị phân (binary digit) thường được gọi là ***bit***. Một chuỗi gồm 4 bit nhị phân gọi là ***nibble***, chuỗi 8 bit gọi là ***byte***, chuỗi 16 bit gọi là ***word*** và chuỗi 32 bit gọi là ***double word***. Chữ số nhị phân bên phải nhất của chuỗi bit gọi là ***bit có ý nghĩa nhỏ nhất*** (***least significant bit – LSB***) và chữ số nhị phân bên trái nhất của chuỗi bit gọi là ***bit có ý nghĩa lớn nhất*** (***most significant bit – MSB***). Một số trong hệ nhị phân được biểu diễn theo số mũ của 2. Ta thường dùng chữ ***B*** cuối chuỗi bit để xác định đó là số nhị phân.

VD: Số 101110.01b biểu diễn giá trị số:
$101110.01b = 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 0 + 0x2^{-1} + 1x2^{-2}$

❖ **Chuyển số nhị phân thành số thập phân:**

Để chuyển một số nhị phân thành một số thập phân, ta chỉ cần nhân các chữ số của số nhị phân với giá trị thập phân của nó và cộng tất cả các giá trị lại.
VD: $1011.11B = 1x2^3 + 0x2^2 + 1x2^1 + 1 + 1x2^{-1} + 1x2^{-2} = 11.75$

❖ **Chuyển số thập phân thành số nhị phân:**

Để chuyển một số thập phân thành số nhị phân, ta dùng 2 phương pháp sau:

- **Phương pháp 1**: Ta lấy số thập phân cần chuyển trừ đi 2^i trong đó 2^i là số lớn nhất nhỏ hơn hay bằng số thập phân cần chuyển. Sau đó, ta lại lấy kết quả này và thực hiện tương tự cho đến 2^0 thì dừng. Trong quá trình thực hiện, ta sẽ ghi lại các giá trị 0 hay 1 cho các bit tuỳ theo trường hợp số thập phân nhỏ hơn 2^i (0) hay lớn hơn 2^i (1).

VD: Xét số 21 thì số 2^i lớn nhất là 2^4

	2^4	2^3	2^2	2^1	2^0
	16	8	4	2	1
21 =	1	0	1	0	1
	5	5	1	1	0

(21 = 10101B)

- **Phương pháp 2**: Lấy số cần chuyển chia cho 2, ta nhớ lại số dư và lấy tiếp thương của kết quả trên chia cho 2 và thực hiện tương tự cho đến khi thương cuối cùng bằng 0. Kết quả chuyển đổi sẽ là chuỗi các bit là các số dư lấy theo thứ tự ngược lại.

VD: Chuyển 227 ra số nhị phân

Số bị chia	Thương	Số dư
227	113	1 (LSB)
113	56	1
56	28	0
28	14	0
14	7	0
7	3	1
3	1	1
1	0	1 (MSB)

(227 = 11100011b)

Để thực hiện chuyển các số thập phân nhỏ hơn 1 sang các số nhị phân, ta làm như sau: lấy số cần chuyển nhân với 2, giữ lại phần nguyên và lại lấy phần lẻ nhân với 2. Quá trình tiếp tục cho đến khi phần lẻ bằng 0 thì dừng. Kết quả chuyển đổi là chuỗi các bit là giá trị các phần nguyên.

VD: Chuyển 0.625 thành số nhị phân

0.625 × 2 = 1.25
0.25 × 2 = 0.5
0.5 × 2 = 1.0
(0.625 = 0.101b)

1.3. Hệ thập lục phân (Hexadecimal Number System)

Như đã biết ở trên, nếu dùng hệ nhị phân thì sẽ cần một số lượng lớn các bit để biểu diễn. Giả sử như số 1024 = 2^{10} sẽ cần 10 bit để biểu diễn. Để rút ngắn kết quả biểu diễn, ta dùng hệ thập lục phân dựa cơ sở trên số mũ của 16. Khi đó, 4 bit trong hệ nhị phân (1 nibble) sẽ biểu diễn bằng 1 chữ số trong hệ thập lục phân (gọi là số hex).

Trong hệ thống này, ta dùng các số 0..9 và các kí tự A..F để biểu diễn cho một giá trị số. Thông thường, ta dùng chữ h ở cuối để xác định đó là số thập lục phân.

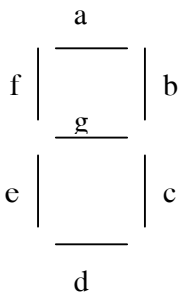
1.4. Mã BCD (Binary Coded Decimal)

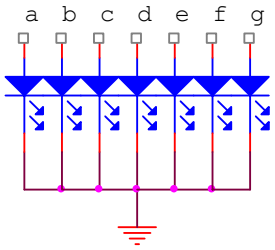
Trong thực tế, đối với một số ứng dụng như đếm tần, đo điện áp, … ngõ ra ở dạng số thập phân, ta dùng mã BCD. Mã BCD dùng 4 bit nhị phân để mã hoá cho một số thập phân 0..9. Như vậy, các số hex A..F không tồn tại trong mã BCD.

VD: Số thập phân 5 2 9
Số BCD 0101 0010 1001

1.5. Mã hiển thị Led 7 đoạn (7-segment display)

Đối với các ứng dụng dùng hiển thị số liệu ra Led 7 đoạn, ta dùng mã hiển thị Led 7 đoạn (bảng 1.1).





Bảng 1.1:

Số thập phân	Số thập lục phân	Số nhị phân	Mã Led 7 đoạn	
			a b c d e f g	Hiển thị
0	0	0000	1 1 1 1 1 1 0	0
1	1	0001	0 1 1 0 0 0 0	1
2	2	0010	1 1 0 1 1 0 1	2
3	3	0011	1 1 1 1 0 1 1	3
4	4	0100	0 1 1 0 0 1 1	4
5	5	0101	1 0 1 1 0 1 1	5
6	6	0110	1 0 1 1 1 1 1	6
7	7	0111	1 1 1 0 0 0 0	7
8	8	1000	1 1 1 1 1 1 1	8
9	9	1001	1 1 1 0 0 1 1	9
10	A	1010	1 1 1 1 1 0 1	A
11	B	1011	0 0 1 1 1 1 1	B
12	C	1100	0 0 0 1 1 0 1	C
13	D	1101	0 1 1 1 1 0 1	D
14	E	1110	1 1 0 1 1 1 1	E
15	F	1111	1 0 0 0 1 1 1	F

2. Các phép toán số học

2.1. Hệ nhị phân

2.1.1. Phép cộng

Phép cộng trong hệ nhị phân cũng thực hiện giống như trong hệ thập phân. Bảng sự thật của phép cộng 2 bit với 1 bit nhớ (carry) như sau:

Bảng 1.2:

Vào			Ra	
A	**B**	**C_{IN}**	**S**	**C_{OUT}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = A \oplus B \oplus C_{IN}$

$C_{OUT} = AB + C_{IN}(A \oplus B)$

VD: 1001 1010

1 +1100 1100

Nhớ 0111 0110

2.1.2. Số bù 2 (2's component)

Trong hệ thống số thông thường, để biểu diễn số âm ta chỉ cần thêm dấu – vào các chữ số. Tuy nhiên, trong hệ thống máy tính, ta không thể biểu diễn được như trên. Phương pháp thông dụng là dùng bit có ý nghĩa lớn nhất (MSB) làm bit dấu (sign bit): nếu MSB = 1 sẽ là số âm còn MSB = 0 là số dương. Khi đó, các bit còn lại sẽ biểu diễn độ lớn (magnitude) của số. Như vậy, nếu ta dùng 8 bit để biểu diễn thì sẽ thu được 256 tổ hợp ứng với các giá trị 0..255 (số không dấu) hay –127.. –0 +0 … +127 (số có dấu).

Để thuận tiện hơn trong việc tính toán số có dấu, ta dùng một dạng biểu diễn đặc biệt là số bù 2. Số bù 2 của một số nhị phân xác định bằng cách lấy đảo các bit rồi cộng thêm 1.

VD: Số 7 biểu diễn là : 0000 0111 có MSB = 0 (biểu diễn số dương)
Số bù 2 là : 1111 1000 + 1 = 1111 1001. Số này sẽ đại diện cho số – 7.

Ta thấy, để thực hiện việc xác định số bù 2 của một số A, cần phải:

- Biểu diễn số A theo mã bù 2 của nó.
- Đảo các bit (tìm số bù 1 của A).
- Cộng thêm 1 vào để nhận được số bù 2.

Khi biểu diễn theo số bù 2, nếu sử dụng 8 bit ta sẽ có các giá trị số thay đổi từ -128..127.

2.1.3. Phép trừ

Phép trừ các số nhị phân cũng được thực hiện tương tự như trong hệ thập phân. Bảng sự thật của phép trừ 2 bit với 1 bit mượn (borrow) như sau:

Bảng 1.3:

Vào			Ra	
A	**B**	**BIN**	**D**	**B_{OUT}**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$S = A \oplus B \oplus BIN$

$BOUT = \overline{A}B + \overline{(A \oplus B)}B_{IN}$

VD:
0110 1101 = 149
- 0011 0001 = 49
0011 1100 = 100

Ngoài cách trừ như trên, ta cũng có thể thực hiện phép trừ thông qua số bù 2 của số trừ.

VD:
0110 1101 → 0110 1101
- 0011 0001 → + 1100 1111
1 (Nhớ)
0011 1100
Số bù 1

1100 1110 + 1 = 1100 1111 (Số bù 2)

Trong phép cộng với số bù 2, ta bỏ qua bit nhớ cuối cùng → kết quả phép cộng số bù 2 là 0011 1100. Đây cũng chính là kết quả phép trừ, bit MSB = 0 cho biết kết quả là số dương.

VD:
77 0100 1101 0100 1101
- 88 - 0101 1000 → + 1010 1000
- 11 1111 0101

Số 88 = 0101 1000 → số bù 1 là 1010 0111 → số bù 2: 1010 1000

Kết quả phép cộng số bù 2 là 1111 0101 có MSB = 1 nên là số âm. Số bù 1 là 0000 1010 → số bù 2: 0000 1011. Kết quả này chính là 11 nên phép trừ sẽ cho kết quả là –11.

Ta thấy, để thực hiện chuyển số bù 2 thành số có dấu thì cần thực hiện:
- Lấy bù các bit để tìm số bù 1.
- Cộng với 1.
- Thêm dấu trừ để xác định là số âm.

2.1.4. Phép nhân

Phép nhân các số nhị phân cũng tương tự như đối với các số thập phân. Chú ý rằng đối với phép nhân nếu nhân 2 số 4 bit sẽ có kết quả là số 8 bit, 2 số 8 bit sẽ có kết quả là số 16 bit, …

VD:

```
  11          1011b
 X 9         1001b
----        -------
  99          1011
             0000
            0000
           1011
          -------
          1100011b
```

Đối với máy tính, phép nhân được thực hiện bằng phương pháp cộng và dịch phải (add-and-right-shift):

- Thành phần dầu tiên của tổng sẽ chính là số bị nhân nếu như LSB của số nhân là 1. Ngược lại, nếu LSB của số nhân bằng 0 thì thành phần này bằng 0.
- Mỗi thành phần thứ i kế tiếp sẽ được tính tương tự với điều kiện là phải dịch trái số bị nhân i bit.
- Kết quả cần tìm chính là tổng các thành phần nói trên.

2.1.5. Phép chia

Phép chia các số nhị phân cũng tương tự như đối với các số thập phân.

VD: 30/5 = 6

```
11110 | 110
110   |-----
-----  101
 011
 000
 ---
  110
  110
  ---
    0
```

Tương tự như đối với phép nhân, ta có thể dùng phép trừ và phép dịch trái cho đến khi không thể thực hiện phép trừ được nữa. Tuy nhiên, để thuận tiện cho tính toán, thay vì dùng phép trừ đối với số chia, ta sẽ thực hiện phép cộng đối với số bù 2 của số chia.

- Đổi số chia ra số bù 2 của nó.
- Lấy số bị chia cộng với số bù 2 của số chia.
  + Nếu kết quả này có bit dấu = 0 thì bit tương ứng của thương = 1.
  + Nếu kết quả này có bit dấu = 1 thì bit tương ứng của thương = 0 và ta phải khôi phục lại giá trị của số bị chia bằng cách cộng kết quả này với số chia.
- Dịch trái kết quả thu được và thực hiện tiếp tục như trên cho đến khi kết quả là 0 hay nhỏ hơn số chia.

2.2. Hệ thập lục phân

2.2.1. Phép cộng

Thực hiện chuyển các số hex cần cộng thành các số nhị phân, tính kết quả trên số nhị phân và sau đó chuyển lại thành số hex.

VD: 7Ah → 0111 1010
3Fh → 0011 1111
B9h ← 1011 1001

Thực hiện cộng trực tiếp trên số hex, nếu kết quả cộng lớn hơn 15 thì sẽ nhớ và trừ cho 16.

VD: 7 A
3 F
10_{10} 25_{10} → B9h

Ah + Fh = $10_{10} + 15_{10} = 25_{10}$ → nhớ 1 và $25_{10} - 16_{10} = 9_{10} = 9h$

7h + 3h = $7_{10} + 3_{10} = 10_{10}$ → cộng số nhớ: $10_{10} + 1_{10} = 11_{10} = Bh$

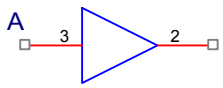
2.2.2. Phép trừ

Thực hiện tương tự như phép cộng.

3. Các thiết bị số cơ bản

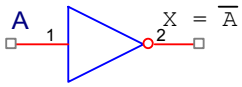
3.1. Cổng đệm (buffer) và các cổng logic (logic gate)

❖ **Cổng đệm:**



A	X
0	0
1	1

❖ **Cổng NOT:**

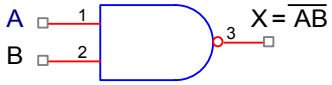


A	X
0	1
1	0

❖ **Cổng AND:**

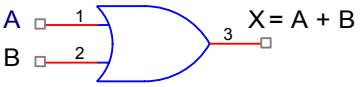
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

❖ **Cổng NAND:**



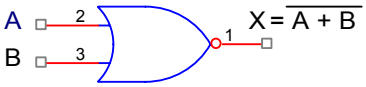
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

❖ **Cổng OR:**



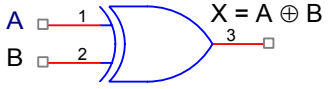
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

❖ **Cổng NOR:**



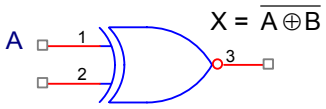
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

❖ **Cổng EX-OR:**



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

❖ **Cổng EX-NOR:**



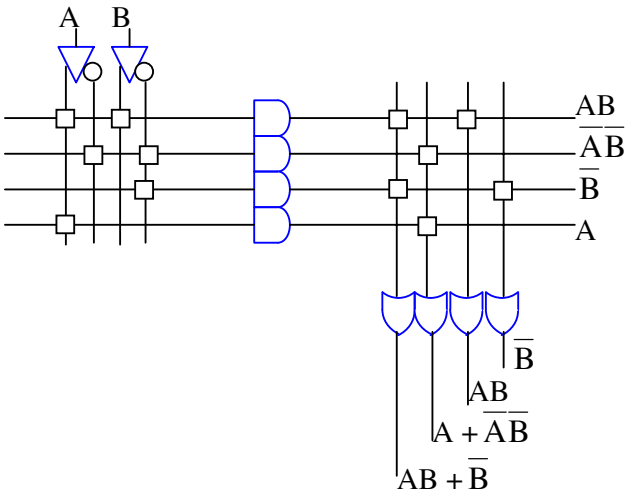
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

3.2. Thiết bị logic lập trình được

Thay vì sử dụng các cổng logic rời rạc, ta có thể dùng các thiết bị logic lập trình được (programmable logic device) như PLA (Programmable Logic Array), PAL (Programmable Array Logic) hay PROM (Programmable Read Only Memory) để liên kết các thiết bị LSI (Large Scale Intergration).

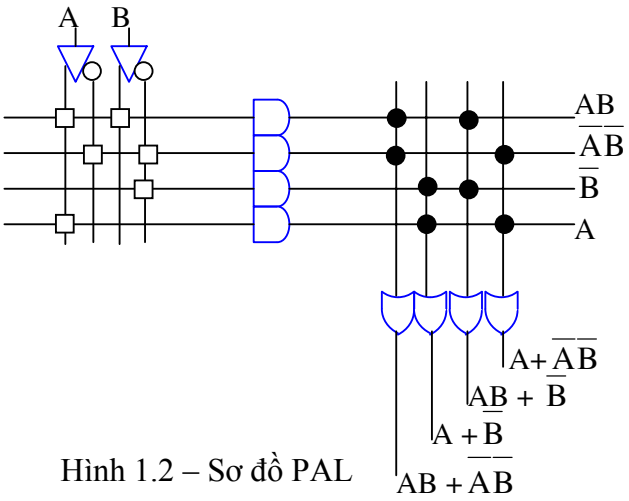
❖ **PLA (hay FPLA – Field PLA):**

Dùng ma trận cổng AND và OR để lập trình bằng cácc phá huỷ các cầu chì. FPLA rất linh động nhưng lại khó lập trình.



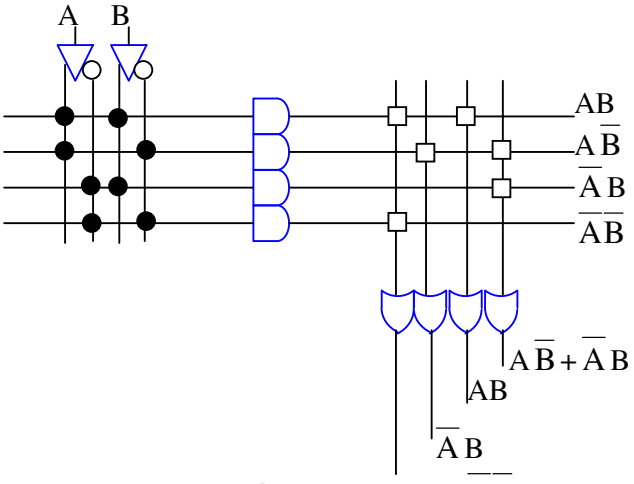
Hình 1.1 – Sơ đồ PLA

❖ **PAL**: ma trận OR đã cố định sẵn và ta chỉ lập trình trên ma trận AND.



Hình 1.2 – Sơ đồ PAL

❖ **PROM**: ma trận AND cố định sẵn và ta chỉ lập trình trên ma trận OR.

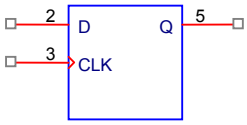


Hình 1.3 – Sơ đồ PROM

3.3. Chốt, flipflop và thanh ghi

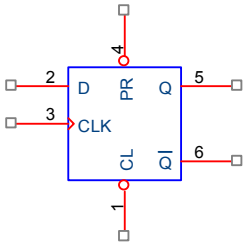
❖ **Chốt (latch):**

Chốt là thiết bị số lưu trữ lại giá trị số tại ngõ ra của nó.



D	CLK	Q
X	0	QN
0	1	0
1	1	1

❖ **Flipflop:**



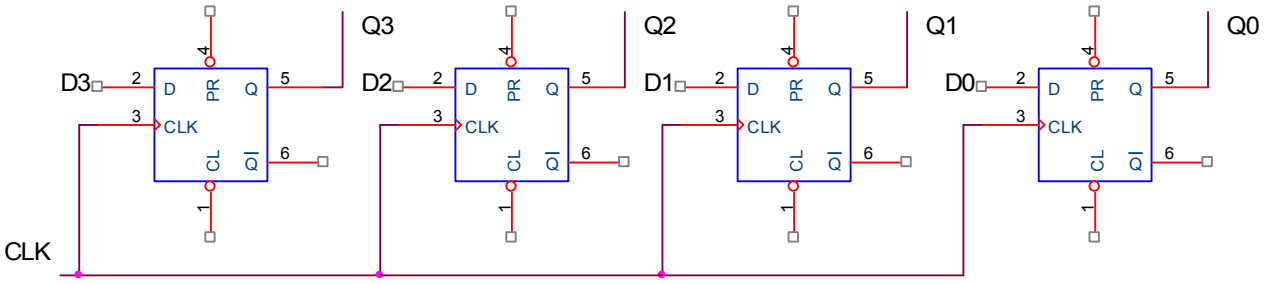
PR	CL	D	CLK	Q	$\overline{Q}$
1	1	1	↑	1	0
1	1	0	↑	0	1
1	1	X	0	Q_N	$\overline{Q_N}$
1	1	X	1	Q_N	$\overline{Q_N}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	.	.

CL: clear PR: Preset CLK: Clock

- Nếu xuất hiện cạnh lên của tín hiệu CLK thì ngõ ra Q sẽ có giá trị theo dữ liệu tại D.
- Nếu PR = 0 thì Q = 1. Nếu CL = 0 thì Q = 0.
- Trạng thái PR = CL = 0 là trạng thái cấm, ngõ ra sẽ không ổn định.

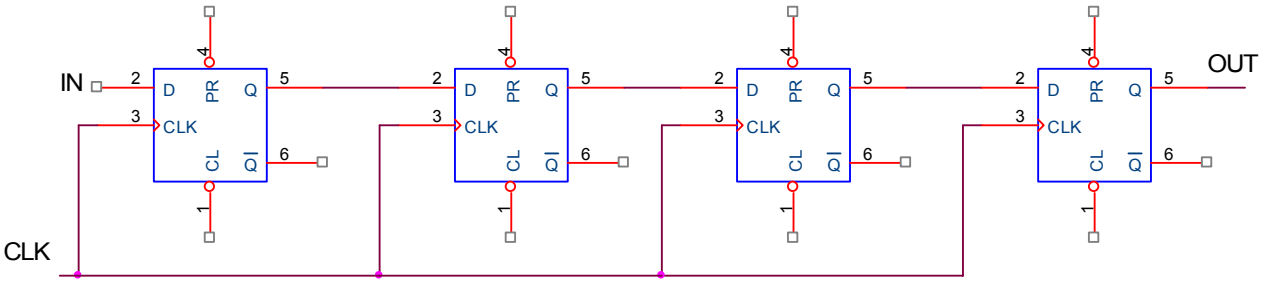
❖ **Thanh ghi (register):**

Thanh ghi là một nhóm các flipflop được kết nối song song để lưu trữ các số nhị phân. Giá trị nhị phân sẽ được đưa vào ngõ vào của các flipflop. Khi có tác động cạnh lên của tín hiệu CLK thì ngõ ra các flipflop sẽ lưu trữ giá trị nhị phân cho đến khi một số nhị phân mới được đưa vào và tác động một cạnh len cho tín hiệu CLK.



Hình 1.4 – Thanh ghi dạng đơn giản

Trong trường hợp các flipflop được kết nối nối tiếp với nhau, ta sẽ có thanh ghi dịch (shift register).



Hình 1.5 – Thanh ghi dịch

3.4. Bộ nhớ

3.4.1. Các kiểu bộ nhớ

❖ **ROM (Read Only Memory):**

Đặc tính chung của ROM là dữ liệu lưu trữ sẽ không bị mất đi dù cho không còn nguồn cung cấp cho ROM (tính nonvolatile – ổn định). Ta chỉ có thể thực hiện tác vụ đọc đối với ROM. ROM có thể được chia thành: ROM che mặt nạ (Masked ROM), PROM (ROM lập trình được), EPROM (ROM có thể xoá bằng tia cực tím) và EEPROM (ROM có thể xoá bằng điện).

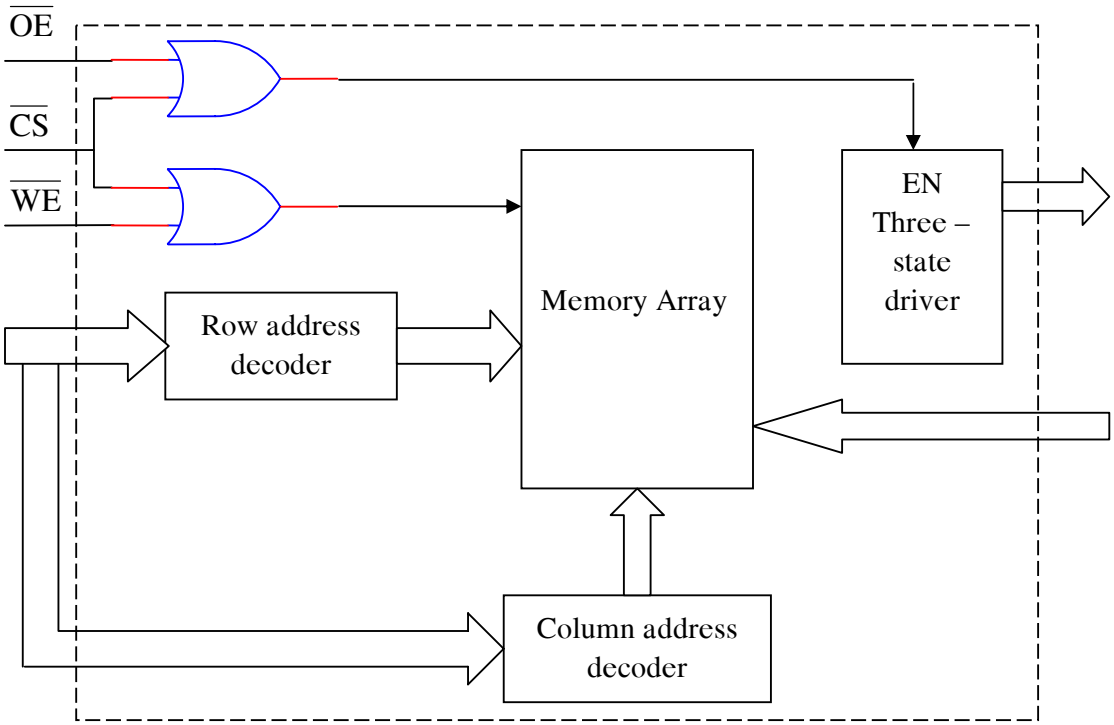
❖ **RAM (Random Access Memory):**

RAM có đặc tính là tất cả nội dung chứa trong RAM sẽ bị mất đi khi không còn nguồn cung cấp cho RAM (tính volatile – không ổn định). Có 2 loại RAM: tĩnh và động.

- **SRAM (Static RAM):** dùng các ma trận flipflop để lưu trữ dữ liệu nên ta có thể ghi các giá trị nhị phân vào RAM bằng cách đưa dữ liệu vào các ngõ vào các flipflop và cấp xung clock cho các flipflop này.

- **DRAM (Dynamic RAM):** tạo ra bằng các cổng transistor và lưu trữ bằng điện tích. Tuy nhiên, do hiện tượng rò rỉ điện tích theo thời gian, ta phải thực hiện nạp điện lại. Quá trình này gọi là làm tươi (refreshing) bộ nhớ. Thuận lợi của DRAM là một số lượng lớn transistor có thể được đặt trên một chip nhớ nên nó có dung lượng cao hơn và nhanh hơn SRAM.

3.4.2. Cấu trúc bên trong của bộ nhớ



Hình 1.6 – Cấu trúc nội một bộ nhớ tiêu biểu

$\overline{CS}$ (Chip Select):cho phép bộ nhớ hoạt động

$\overline{OE}$ (Output Enable): cho phép đọc dữ liệu từ bộ nhớ ra bên ngoài

$\overline{WE}$ (Write Enable): cho phép ghi dữ liệu vào trong bộ nhớ

Row address decoder, Column address decoder: các bộ giải mã hàng và cột để chọn vị trí của memory cell (flipflop hay tụ điện)

Three-state driver: bộ lái ngõ ra 3 trạng thái để đệm ngõ ra

4. Giới thiệu vi xử lý

4.1. Các thế hệ vi xử lý

- ***Thế hệ 1 (1971 – 1973):*** vi xử lý 4 bit, đại diện là 4004, 4040, 8080 (Intel) hay IPM-16 (National Semiconductor).
  + Độ dài word thường là 4 bit (có thể lớn hơn).
  + Chế tạo bằng công nghệ PMOS với mật độ phần tử nhỏ, tốc độ thấp, dòng tải thấp nhưng giá thành rẻ.
  + Tốc độ 10 ÷ 60 µs / lệnh với tần số xung nhịp 0.1 ÷ 0.8 MHz.
  + Tập lệnh đơn giản và phải cần nhiều vi mạch phụ trợ.
- ***Thế hệ 2 (1974 – 1977):*** vi xử lý 8 bit, đại diện là 8080, 8085 (Intel) hay Z80 (Zilog).
  + Tập lệnh phong phú hơn.
  + Địa chỉ có thể đến 64 KB. Một số bộ vi xử lý có thể phân biệt 256 địa chỉ cho thiết bị ngoại vi.
  + Sử dụng công nghệ NMOS hay CMOS.

+ Tốc độ 1 ÷ 8 µs / lệnh với tần số xung nhịp 1 ÷ 5 MHz

- ***Thế hệ 3 (1978 – 1982):*** vi xử lý 16 bit, đại diện là 68000/68010 (Motorola) hay 8086/80286/80386 (Intel)

+ Tập lệnh đa dạng với các lệnh nhân, chia và xử lý chuỗi.

+ Địa chỉ bộ nhớ có thể từ 1 ÷ 16 MB và có thể phân biệt tới 64KB địa chỉ cho ngoại vi

+ Sử dụng công nghệ HMOS.

+ Tốc độ 0.1 ÷ 1 µs / lệnh với tần số xung nhịp 5 ÷ 10 MHz.

- ***Thế hệ 4:*** vi xử lý 32 bit 68020/68030/68040/68060 (Motorola) hay 80386/80486 (Intel) và vi xử lý 32 bit Pentium (Intel)

+ Bus địa chỉ 32 bit, phân biệt 4 GB bộ nhớ.

+ Có thể dùng thêm các bộ đồng xử lý (coprocessor).

+ Có khả năng làm việc với bộ nhớ ảo.

+ Có các cơ chế pipeline, bộ nhớ cache.

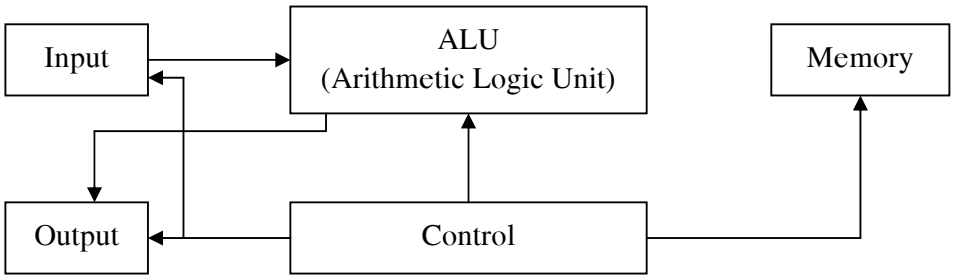
+ Sử dụng công nghệ HCMOS.

4.2. Vi xử lý (µP – microproccessor)

4.2.1. Phân loại vi xử lý

- ***Multi chip***: dùng 2 hay nhiều chip LSI (Large Scale Intergration: tích hợp từ 1000 ÷ 10000 transistor) cho ALU và control.
- ***Microprocessor***: dùng 1 chip LSI/VLSI (Very Large Scale Intergration: tích hợp ÷ 10000 transistor) cho ALU và control.
- ***Single chip microprocessor*** (còn gọi là microcomputer / microcontroller): là 1 chip LSI/VLSI chứa toàn bộ các khối như hình 1.7.

4.2.2. Sơ đồ khối một máy tính cổ điển



Hình 1.7 – Sơ đồ khối một máy tính cổ điển

- *ALU (đơn vị logic số học)*: thực hiện các bài toán cho máy tính bao gồm: +, -, *, /, phép toán logic, …
- *Control (điều khiển)*: điều khiển, kiểm soát các đường dữ liệu giữa các thành phần của máy tính.
- *Memory (bộ nhớ)*: lưu trữ chương trình hay các kết quả trung gian.
- *Input (nhập), Output (Xuất)*: các thiết bị xuất nhập dữ liệu (còn gọi là thiết bị ngoại vi).

4.2.3. Sơ đồ khối của µP

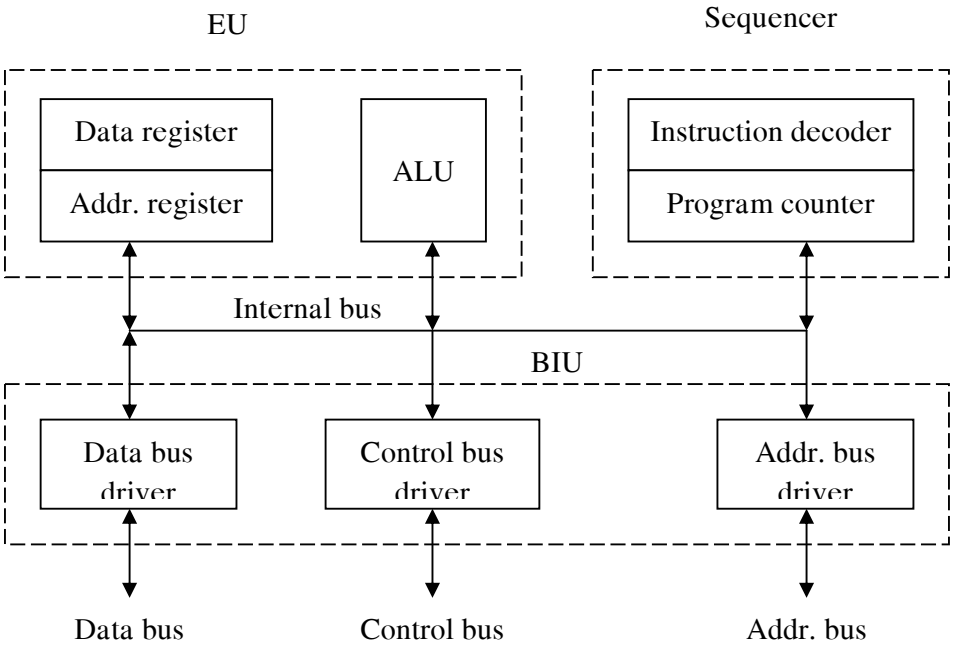
Có 3 khối chức năng: đơn vị thực thi (EU - Execution unit), bộ tuần tự (Sequencer) và đơn vị giao tiếp bus (BIU – Bus interface unit).

- EU: thực hiện các lệnh số học và logic. Các toán hạng được chứa trong các thanh ghi dữ liệu (data register) hay thanh ghi địa chỉ (address register), hay từ bus nội (internal bus).

- Bộ tuần tự: gồm bộ giải mã lệnh (instruction decoder) và bộ đếm chương trình (program counter)

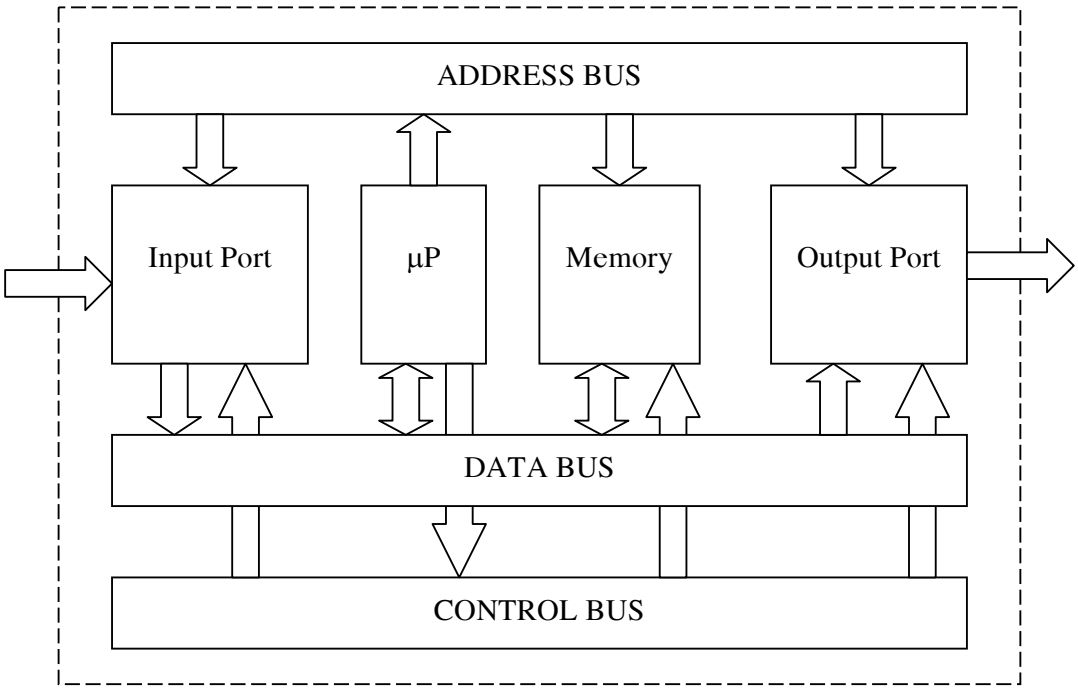
  + Bộ đếm chương trình chứa các lệnh kế tiếp sẽ thực hiện
  + Bộ giải mã sẽ thực hiện các bước cần thiết để thực thi lệnh.

Khi chương trình bắt đầu, bộ đếm chương trình (PC) sẽ ở địa chỉ bắt đầu. Địa chỉ này được chuyển qua bộ nhớ thông qua address bus. Khi tín hiệu Read đưa vào control bus, nội dung bộ nhớ liên quan sẽ đưa vào bộ giải mã lệnh. Bộ giải mã lệnh sẽ khởi động các phép toán cần thiết để thực thi lệnh. Quá trình này đòi hỏi một số chu kỳ máy (machine cycle) tuỳ theo lệnh. Sau khi lệnh đã thực thi, bộ giải mã lệnh sẽ đặt PC đến địa chỉ của lệnh kế.



Hình 1.8 – Sơ đồ khối của vi xử lý

4.2.4. Sơ đồ khối của hệ vi xử lý cơ bản



Hình 1.9 – Sơ đồ khối hệ vi xử lý

Mọi hoạt động cơ bản của một hệ vi xử lý đều giống nhau, không phụ thuộc loại vi xử lý hay quá trình thực hiện. µP sẽ đọc một lệnh từ bộ nhớ (memory), thực thi lệnh và sau đó đọc lệnh kế. Quá trình đọc lệnh gọi là instruction fetch còn quá trình thực hiện tuần tự như trên gọi là fetch – execute sequence. Tuy nhiên có một số µP sẽ nhận một số lệnh rồi mới bắt đầu thực thi.

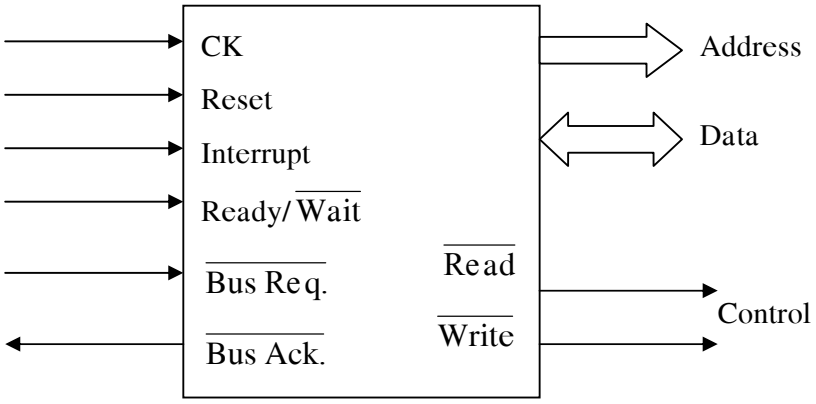
❖ **Các port I/O:**

Các port nhập (input) và xuất (output) dùng để giao tiếp giữa µP và thiết bị ngoại vi (không thể nối trực tiếp với các bus).

Port xuất là một thanh ghi. Khi µP ghi dữ liệu ra địa chỉ của Port thì Port sẽ chứa dữ liệu hiện tại trên data bus. Dữ liệu này sẽ được chốt tại Port cho đến khi µP ghi dữ liệu mới ra Port.

Port nhập là một driver 3 trạng thái. Khi µP đọc vào từ địa chỉ của Port, driver 3 trạng thái lái dữ liệu từ bên ngoài vào data bus. Sau đó, µP đọc dữ liệu từ bus.

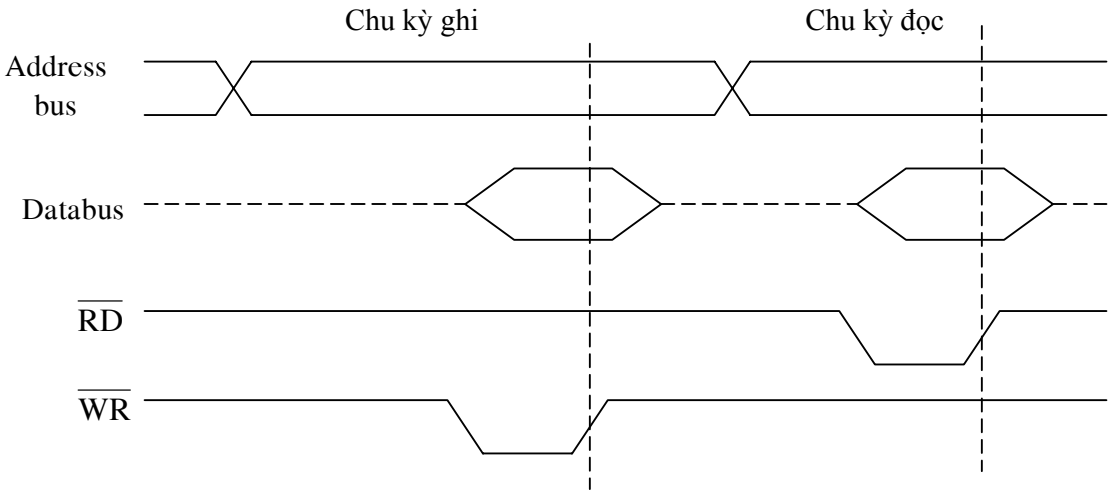
❖ **Các tín hiệu tiêu biểu của một µP:**



Hình 1.10 – Các tín hiệu cơ bản trong µP

Các bus dùng để liên kết các thành phần của hệ thống với µP. µP sẽ chọn một thiết bị cần sử dụng thông qua address bus và đọc hay ghi dữ liệu thông qua data bus. Data bus là bus 2 chiều, dùng chung cho tất cả các quá trình trao đổi dữ liệu. Mỗi chu kỳ bus (bus cycle) là việc thực hiện trao đổi một từ dữ liệu giữa µP và ô nhớ hay thiết bị I/O.

Mỗi chu kỳ bus bắt đầu khi µP xuất một địa chỉ nhằm chọn thiết bị I/O hay chọn một ô nhớ nào đó.



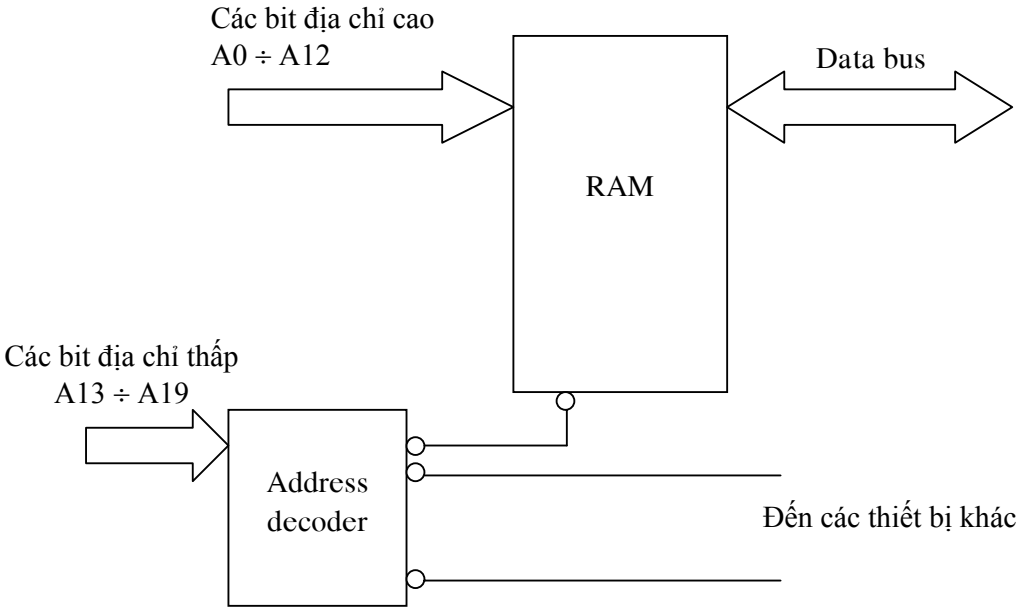
Hình 1.11 – Định thì bus cơ bản

4.3. Giao tiếp với bộ nhớ

4.3.1. Giao tiếp bus cơ bản

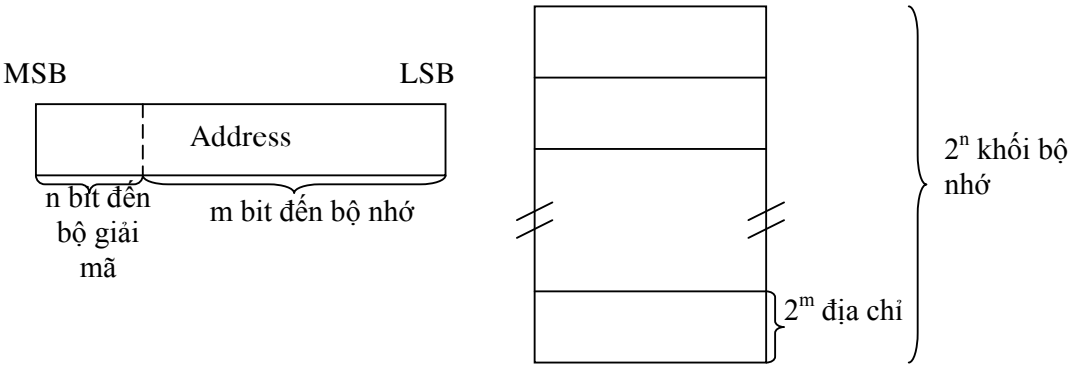
- Các bit địa chỉ thấp (giả sử 13 đường A0 ÷ A12) nối trực tiếp đến chip bộ nhớ (giả sử RAM có dung lượng 8K × 8)
- Các bit địa chỉ cao (giả sử A13 ÷ A19) nối với bộ giải mã địa chỉ (address decoder) tạo tín hiệu cho phép chip bộ nhớ. Do đó, khi thiết kế ta phải xác

định mỗi chip bộ nhớ thuộc vùng địa chỉ nào. Tập hợp các vùng này theo bảng gọi là bảng bộ nhớ (memory map).



Hình 1.12 – Giao tiếp bus cơ bản

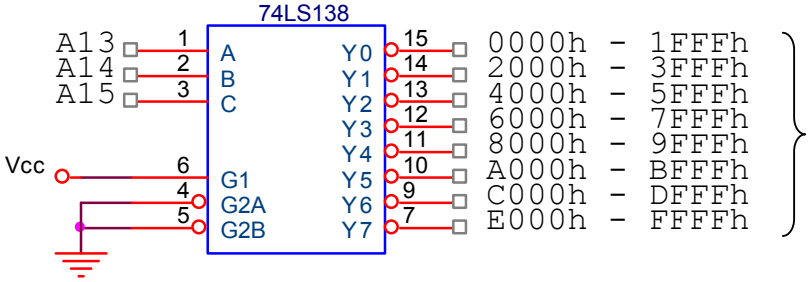
Quan hệ giữa giải mã địa chỉ và bảng bộ nhớ:



Hình 1.13 – Bảng bộ nhớ

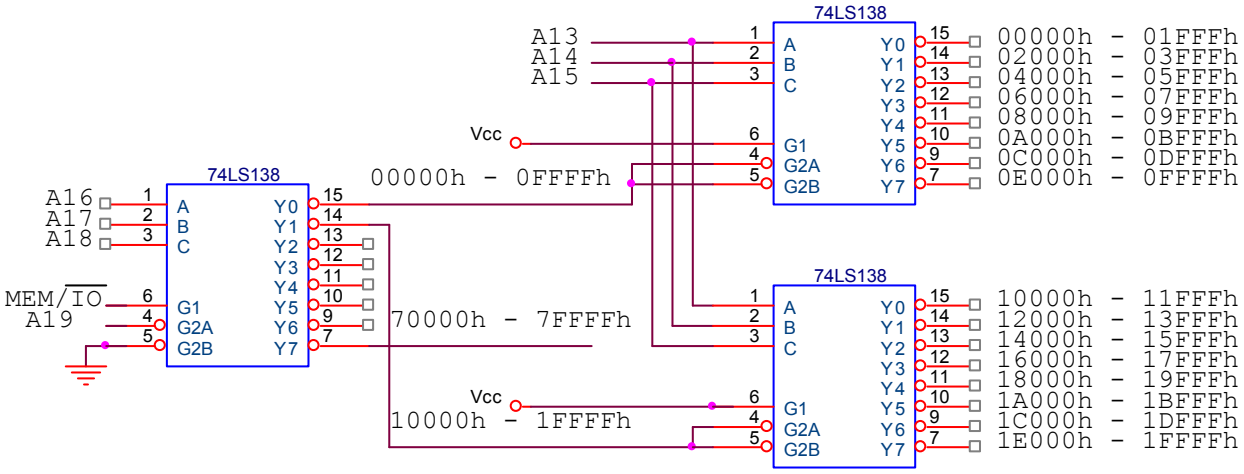
4.3.2. Giải mã địa chỉ

4.3.2.1. Dùng 74LS138



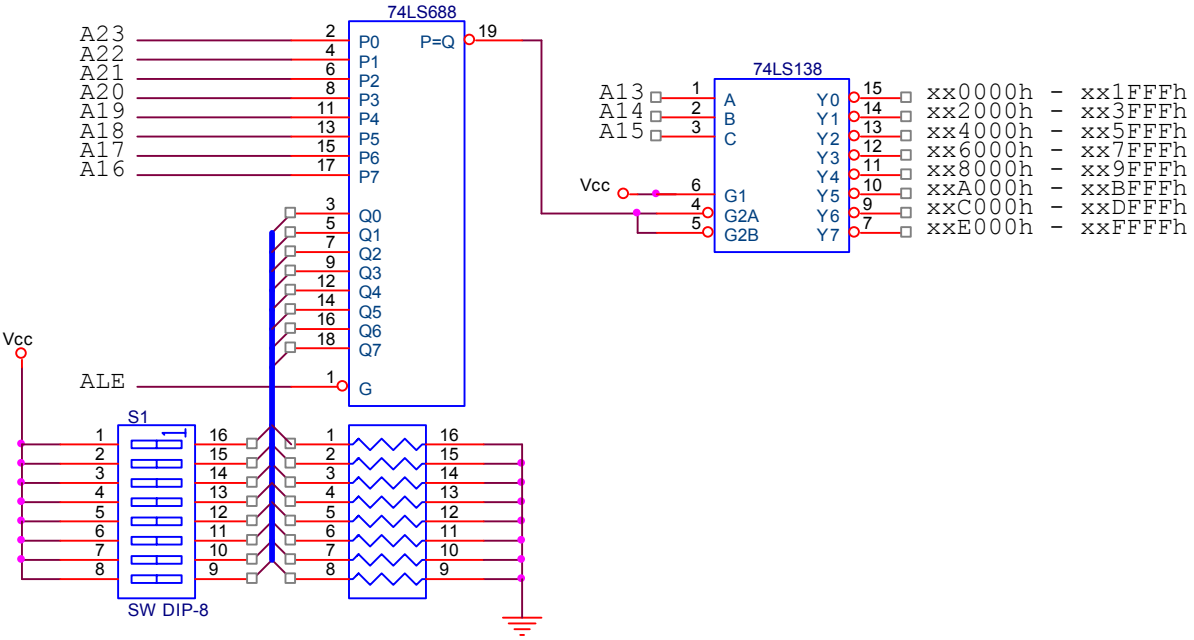
Hình 1.14 – Giải mã địa chỉ dùng 74LS138

4.3.2.2. Dùng nhiều 74LS138



Hình 1.15 – 74LS138 mắc cascaded (xâu chuỗi)

4.3.2.3. Dùng bộ so sánh



Hình 1.16 – Giải mã dùng bộ so sánh

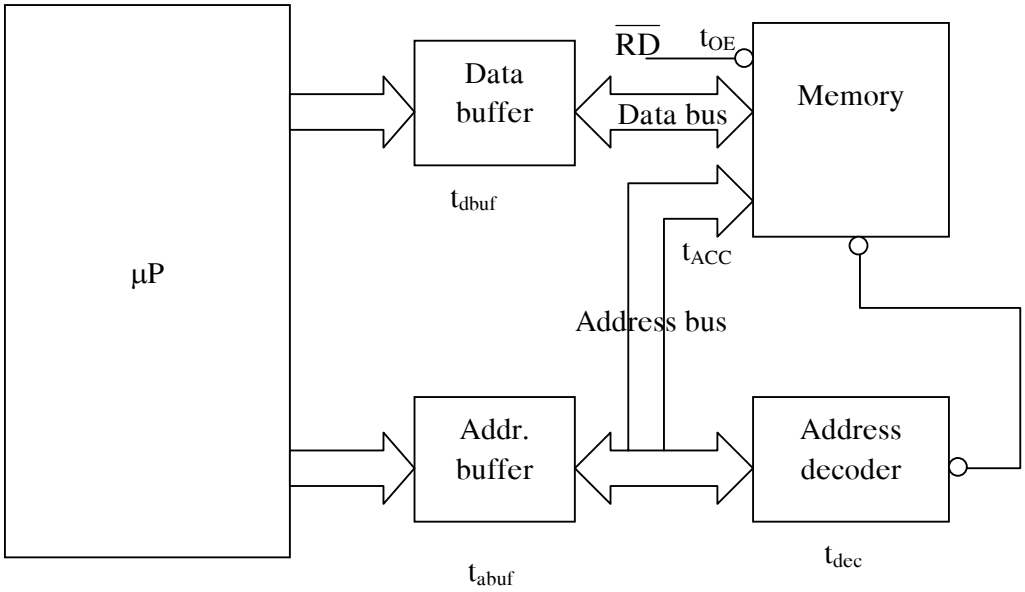
4.3.3. Định thì bộ nhớ

❖ **Thời gian truy xuất (access time):**

- Với chu kỳ đọc: thời gian truy xuất là thời gian tính từ lúc địa chỉ mới xuất hiện ở bộ nhớ cho đến khi có dữ liệu đúng ở ngõ ra của bộ nhớ.
- Với chu kỳ ghi: thời gian truy xuất là thời gian tính từ lúc địa chỉ mới xuất hiện ở bộ nhớ cho đến khi dữ liệu đã đưa vào bộ nhớ.

❖ **Thời gian chu kỳ (cycle time):** là thời gian từ lúc bắt đầu chu kỳ bộ nhớ đến khi bắt đầu chu kỳ kế tiếp.

Ngoài ra, µP có thể sử dụng thêm một số trạng thái chờ khi đọc bộ nhớ.



Hình 1.17 – Các đường trì hoãn trong giao tiếp µP với bộ nhớ

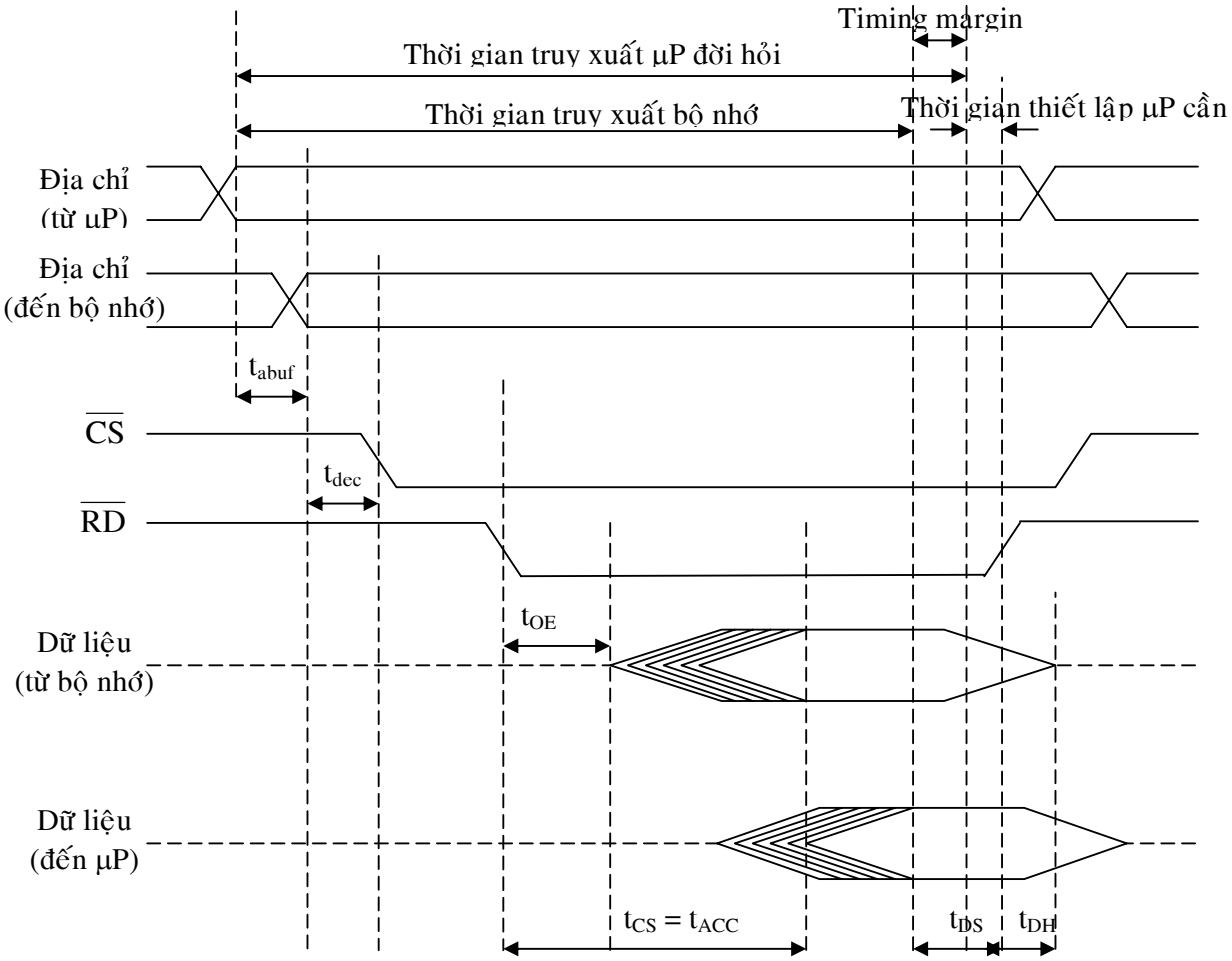
t_{dbuf}: thời gian trì hoãn ở bộ đệm dữ liệu (data buffer)
t_{abuf}: thời gian trì hoãn ở bộ đệm địa chỉ (address buffer)
t_{OE}: thời gian đáp ứng của bộ nhớ với tín hiệu cho phép ngõ ra (ouput enable)
t_{CS}: thời gian bộ nhớ truy xuất từ Chip Select
t_{ACC}: thời gian bộ nhớ truy xuất từ địa chỉ, thông thường tACC = tcs
t_{dec}: thời gian trì hoãn ở bộ giải mã (decoder)

❖ **Định thì đọc bộ nhớ:**

Thời gian truy xuất tổng cộng của hệ thống bộ nhớ chính là tổng thời gian trì hoãn trong các bộ đệm và thời gian truy xuất (access time) bộ nhớ.

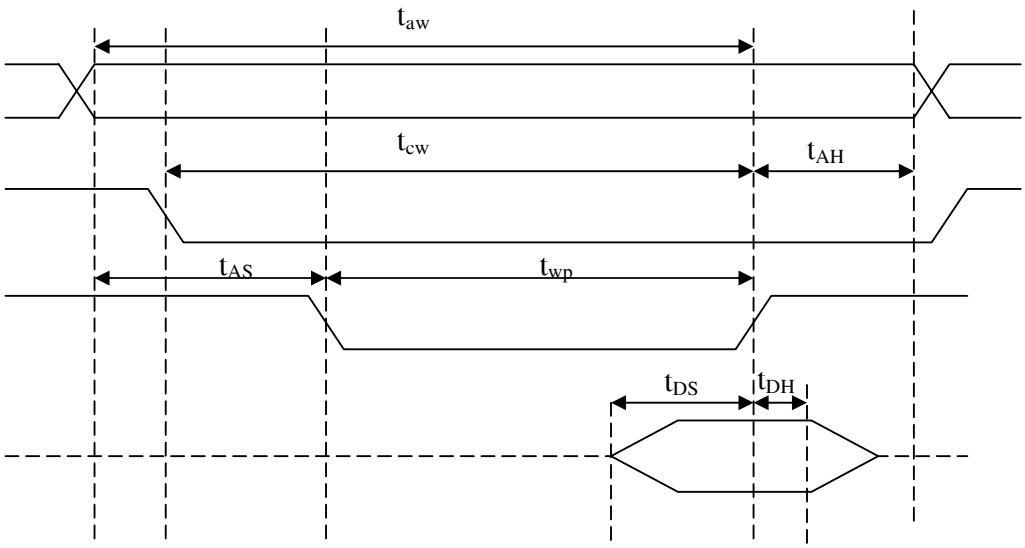
Hiệu giữa thời gian truy xuất cần thiết bởi µP với thời gian truy xuất thật sự của bộ nhớ gọi là biên định thì (timing margin).

t_{DS} (Data Setup): thời gian thiết lập dữ liệu cung cấp bởi hệ thống bộ nhớ
t_{DH} (Data Hold): thời gian giữ dữ liệu cung cấp bởi hệ thống bộ nhớ



Hình 1.18 – Định thì đọc bộ nhớ

❖ **Định thì ghi bộ nhớ:**



Hình 1.19 – Định thì ghi bộ nhớ

t_{aw}: thời gian truy xuất ghi (access write)

t_{wp}: độ rộng xung ghi tối thiểu (write pulse)

t_{AS}: thời gian địa chỉ hợp lệ trước khi $\overline{WR} = 0$

Thông thường, ta không quan tâm đến địa chỉ cho đến khi xác nhận $\overline{CS}$ nên thường $t_{cw} = t_{aw}$.

CHƯƠNG 2: TỔ CHỨC HỆ THỐNG VI XỬ LÝ

1. Giới thiệu

Tất cả các máy vi tính IBM họ PC hoặc các máy vi tính tương thích IBM đều sử dụng µP Intel họ iAPX. Bảng 2.1 liệt kê các đặc tính cơ bản của một số µP của Intel trong đó 80486 chứa một bộ điều khiển cache tích hợp và 8 KB RAM tĩnh, Pentium chứa cache 16 KB RAM tĩnh.

Bảng 2.1: Kiến trúc các µP của Intel 8 bit, 16 bit và 32 bit

ĐẶC TÍNH	8080	8086	8088	80186	80188	80286	80386	386SX	486/Pentium
Bus địa chỉ (số bit)	8	16	8	16	8	16	32	16	32
Đường dữ liệu nội (số bit)	8	16	16	16	16	16	32	32	32/64
Tốc độ (MHz)	2,2.6, 6.3	5,8,10	5,8	8,10, 12.5	8,10, 12.5	6,8,10, 12.5,20	16,20,25, 33	16	25-66
Thanh ghi đến thanh ghi (µs/word)	1.3	0.3	0.38	0.2	0.3	0.125	0.125	0.125	0.04
Đáp ứng interrupt (µs)	7.3	6.1	8.6	3.36	6.2	2.52	3.5	2.52	3.5
Địa chỉ bộ nhớ	64K	1M	1M	1M	1M	16M	4G	4G	4G
Cách định địa chỉ	5	24	24	24	24	24	28	28	28
Coprocessor	0	8087	8087	8087	8087	80287	80287/ 80387	80287/ 80387	On chip
Số thanh ghi đa dụng	6	8	8	8	8	8	8	8	8
Số thanh ghi đoạn	0	4	4	4	4	4	6	6	6
Điều khiển interrupt	8259-A	8259 -A	8259-A	On chip	On chip	8259-A	8259-A	82335	µPLD
Timer – counter	8253	8253 /54	8253/ 54	On chip	On chip	8253/ 54	8253/5 4	8253/5 4	On chip

2. µP 8086/8088

2.1. Mô tả

2.1.1. Định thì chu kỳ bus

Mỗi chu kỳ bus bắt đầu bằng việc xuất địa chỉ bộ nhớ hoặc I/O port (chu kỳ xung nhịp T1). Với 8086 thì địa chỉ này có thể là địa chỉ bộ nhớ 20 bit, địa chỉ I/O gián tiếp 16 bit (thanh ghi DX) hay địa chỉ I/O trực tiếp 8 bit.

Bus điều khiển có 4 tín hiệu tác động mức thấp là $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ và $\overline{IOW}$.

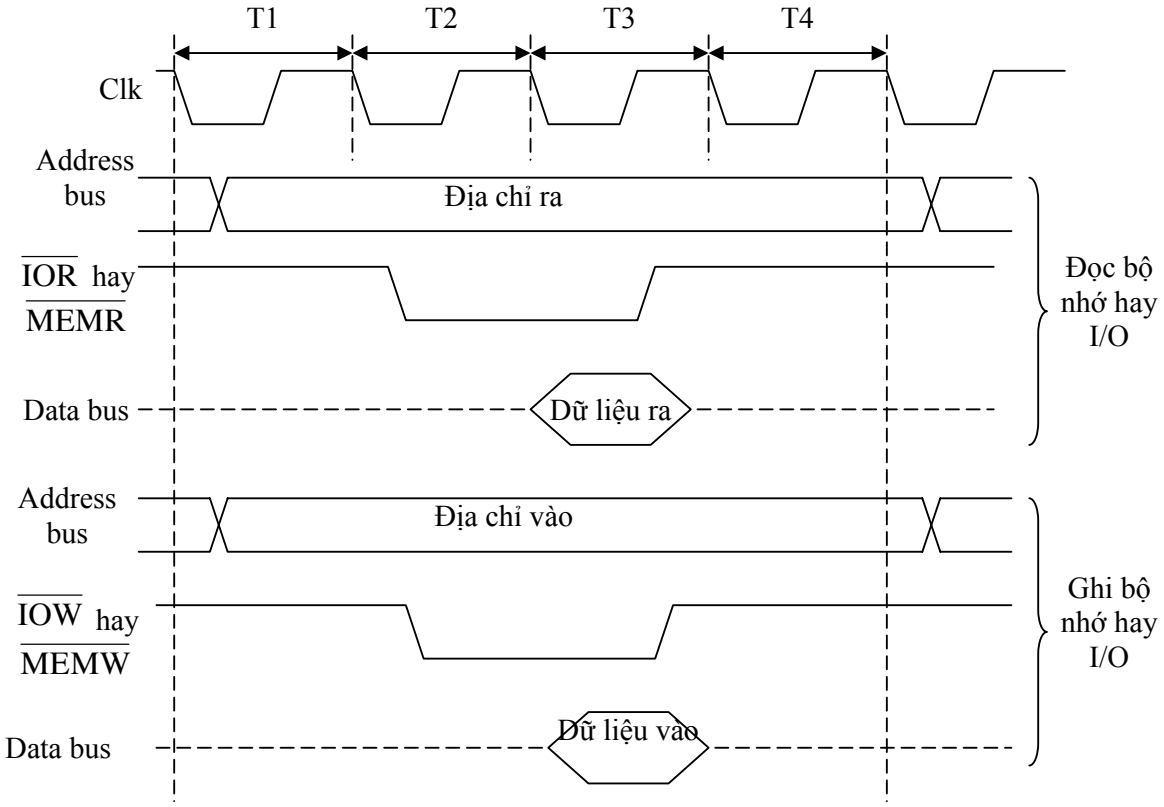
Các chuỗi sự kiện xảy ra trong một chu kỳ bus đọc bộ nhớ:

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 $\overline{MEMR}$ 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.

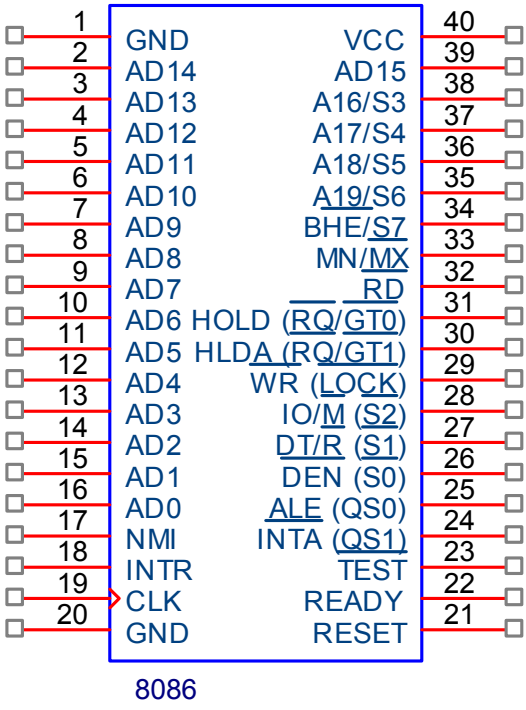


Hình 2.1 – Định thì chu kỳ bus

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).

2.1.2. Mô tả chân



Hình 2.2 – Sơ đồ chân của 8086

8086 có bus địa chỉ 20 bit, bus dữ liệu 16 bit, 3 chân nguồn và 17 chân dùng cho các chức năng điều khiển. Tuy nhiên, ta có thể dùng kỹ thuật ghép kênh thời gian (time multiplexing) để cho phép một chân có nhiều chức năng nên các chân sẽ được phân ra:

- 16 chân dữ liệu và địa chỉ (AD0 ÷ AD15): các chân này sẽ là các đường địa chỉ trong trạng thái T1 và dữ liệu trong các trạng thái T2 – T4.
- 4 chân địa chỉ và trạng thái
- 3 chân nguồn
- 17 chân định thì và điều khiển

8086 có thể hoạt động ở chế độ tối thiểu (minimum mode) hay chế độ tối đa (maximum mode). Chế độ tối thiểu chỉ dùng cho các hệ thống µP đơn giản còn chế độ tối đa dùng cho các hệ thống phúc tạp hơn giao tiếp với các bộ nhớ và I/O riêng.

❖ **Các tín hiệu chung cho cả hai chế độ tối đa và tối thiểu:**

Bảng 2.2:

Chân	Chức năng	Loại
AD15 ÷ AD0	Bus dữ liệu / địa chỉ	2 chiều, 3 trạng thái
A19/S6 ÷ A16/S3	Địa chỉ / trạng thái	Ngõ ra 3 trạng thái
$\overline{MX}$	Điều khiển chế độ	Ngõ vào
$\overline{RD}$	Điều khiển đọc	Ngõ ra 3 trạng thái
$\overline{TEST}$	Chờ kiểm tra điều khiển	Ngõ vào
READY	Chờ trạng thái điều khiển	Ngõ vào
RESET	Reset hệ thống	Ngõ vào
NMI	Yêu cầu ngắt không thể che	Ngõ vào
INTR	Yêu cầu ngắt	Ngõ vào
CLK	Xung nhịp hệ thống	Ngõ vào
VCC	+5V	Ngõ vào
GND	GND	Ngõ vào

❖ **Các tín hiệu chỉ dùng trong chế độ tối thiểu:**

Bảng 2.3:

Chân	Chức năng	Loại
HOLD	Yêu cầu giữ	Ngõ vào
HLDA	Ghi nhận giữ	Ngõ vào
$\overline{WR}$	Điều khiển ghi	Ngõ ra 3 trạng thái
IO/$\overline{M}$	Điều khiển I/O và bộ nhớ	Ngõ ra 3 trạng thái
DT/$\overline{R}$	Truyền / nhận dữ liệu	Ngõ ra 3 trạng thái
$\overline{DEN}$	Cho phép dữ liệu	Ngõ ra 3 trạng thái
$\overline{BHE}$/S7	Đường trạng thái	Ngõ ra 3 trạng thái
ALE	Cho phép chốt địa chỉ	Ngõ ra
$\overline{INTA}$	Ghi nhận ngắt	Ngõ ra

❖ **Các tín hiệu chỉ dùng trong chế độ tối đa:**

Bảng 2.4:

Chân	Chức năng	Loại
$\overline{RQ}$/$\overline{GT1,0}$	Yêu cầu / cấp bus	2 chiều
$\overline{LOCK}$	Điều khiển khóa ưu tiên bus	Ngõ ra 3 trạng thái
$\overline{S2} \div \overline{S0}$	Trạng thái chu kỳ bus	Ngõ ra 3 trạng thái
QS1, QS2	Trạng thái hàng lệnh	Ngõ ra

❖ **Trạng thái bus:**

Bảng 2.5:

Ngõ vào trạng thái			Chu kỳ CPU
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	Ghi nhận ngắt
0	0	1	Đọc I/O port
0	1	0	Ghi I/O port
0	1	1	Ngừng
1	0	0	Nhận lệnh
1	0	1	Đọc bộ nhớ
1	1	0	Ghi bộ nhớ
1	1	1	Thụ động

❖ **Trạng thái hàng lệnh:**

Bảng 2.6:

QS1	QS0	Trạng thái hàng lệnh
0	0	Không hoạt động
0	1	Lấy byte đầu tiên của lệnh
1	0	Hàng rỗng
1	1	Lấy byte kế tiếp

❖ **Nguồn cung cấp và xung nhịp (VCC, GND và CLK):**

- 8086 sử dụng nguồn cấp điện +5V và có 2 chân đất.
- Dòng điện cực đại là 340 mA (10 mA cho loại CMOS).
- Xung nhịp dùng dạng xung chữ nhật có chu kỳ với thời gian cạnh lên và xuống nhỏ hơn 10 ns.
- Tiêu hao công suất và tần số xung nhịp cực đại:

❖ **Các chân trạng thái trong chế độ tối đa (S0, S1 và S2 - status):**

Các chân này sử dụng bởi bộ điều khiển bus 8288 để tạo các tín hiệu điều khiển như bảng 2.5.

❖ **Các chân điều khiển bus (HOLD, HLDA, $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$, $\overline{LOCK}$):**

Chế độ tối thiểu:

- HOLD (giữ): ngõ vào tác động mức cao làm cho µP hở mạch tất cả các bus của nó, tách µP khỏi bộ nhớ của nó và I/O để cho phép thiết bị khác xử lý

bus hệ thống. Quá trình này gọi là truy xuất bộ nhớ trực tiếp (DMA – Direct Memory Access).

- HLDA (Hold acknowledge): ghi nhận yêu cầu DMA đối với bộ điều khiển DMA.

Chế độ tối đa:

- $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$ (Request / Grant): các chân này dùng cả hai chức năng vào (nhận yêu cầu) và ra (chấp nhận yêu cầu). Khi một thiết bị muốn lấy điều khiển của bus cục bộ, nó sẽ phát yêu cầu bằng cách đưa tín hiệu mức thấp vào chân yêu cầu. Sau khi nhận yêu cầu, 8086 sẽ ở trạng thái HOLD và gởi tín hiệu chấp nhận ra chân này. Ở đây, chân $\overline{RQ}/\overline{GT0}$ có độ ưu tiên cao hơn chân $\overline{RQ}/\overline{GT1}$.
- $\overline{LOCK}$: báo cho các thiết bị khác biết không thể lấy điều khiển của bus cục bộ.

❖ **Các chân ngắt (NMI, INTR và $\overline{INTA}$):**

INTR và NMI là các yêu cầu ngắt khởi động bằng phần cứng, làm việc chính xác như các ngắt mềm. NMI (Non-Maskable Interrupt) là ngõ vào tác động cạnh lên. NMI là ngắt không thể che được và luôn được phục vụ, thường dùng cho các sự kiện như hư nguồn hay các lỗi bộ nhớ. INTR tác động mức cao và có thể bị che bằng cách xoá cờ IF trong thanh ghi cờ (xem 2.3.4) bằng lệnh CLI.

Khi NMI tích cực, điều khiển sẽ được chuyển đến địa chỉ chứa trong các vị trí 00008h ÷ 0000Bh. Khi INTR tích cực, chu kỳ ghi nhận ngắt (interrupt acknowledge cycle) được thực hiện. Quá trình này giống như chu kỳ đọc bộ nhớ ngoại trừ $\overline{INTA}$ tích cực thay vì $\overline{RD}$. Thiết bị tạo ngắt sẽ đặt một giá trị 8 bit vào data bus và chuyển điều khiển đến vị trí giá trị × 4 đến giá trị × 4 + 3.

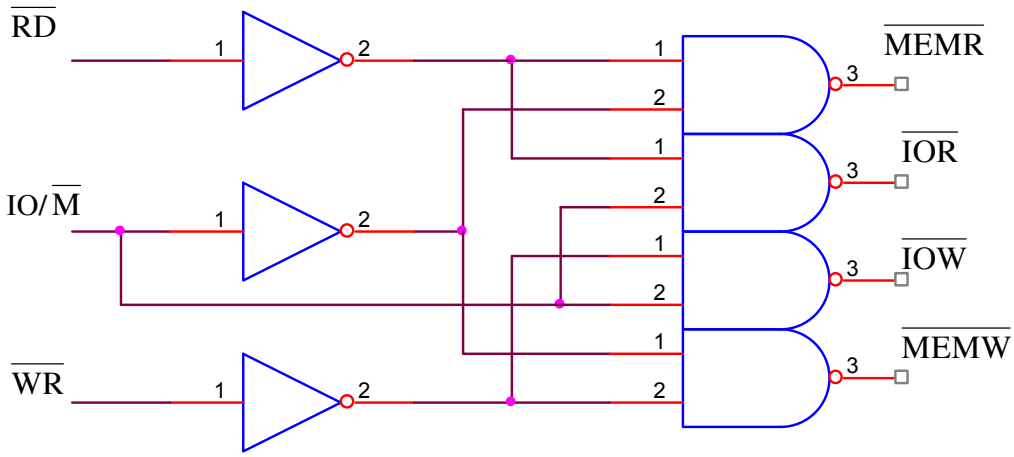
❖ **Chân RESET**: hoạt động khi có xung tác động mức cao, dùng để khởi động lại (P. Sau khi khởi động, (P sẽ đọc lệnh tại địa chỉ FFFF0h. RESET được sử dụng khi hệ thống có sự cố.

❖ **Các chân điều khiển bus (READY, $\overline{RD}$, ALE, $\overline{DEN}$, DT/$\overline{R}$, $\overline{WR}$ và IO/$\overline{M}$):**

Trong các chân điều khiển này, chỉ có hai chân READY và $\overline{RD}$ làm việc ở chế độ tối đa.

- Chân READY: ngõ vào READY được lấy mẫu ở cạnh lên của xung nhịp T2. Nếu chân này ở mức thấp (không sẵn sàng) thì sẽ thêm vào một chu kỳ T3 nữa. Chu trình này sẽ tiếp tục cho đến khi nào chân READY lên mức cao. Ngõ vào này thường được điều khiển bởi thiết bị bộ nhớ chậm, không thể cung cấp dữ liệu kịp thời cho µP.
- Chân IO/$\overline{M}$ (IO/Memory – Xuất nhập /Bộ nhớ): xác định chu kỳ bus hiện hành làm việc với bộ nhớ (mức thấp) hay I/O (mức cao).

- Chân $\overline{RD}$ (Read): tín hiệu tác động mức thấp chỉ chiều truyền dữ liệu từ bộ nhớ hay I/O đến µP. Ta có thể kết hợp với tín hiệu này với IO/$\overline{M}$ để tạo các tín hiệu $\overline{MEMR}$ và $\overline{IOR}$. Nó được xuất ra trong trạng thái T2 và lấy đi trong trạng thái T4. Thiết bị bộ nhớ hay I/O giả sử là đã đặt byte hay word vào các đường dữ liệu khi $\overline{RD}$ trở về mức cao.
- Chân $\overline{WR}$ (Write): tín hiệu này ngược với $\overline{RD}$, nó xác định chiều truyền dữ liệu từ µP đến I/O hay bộ nhớ.

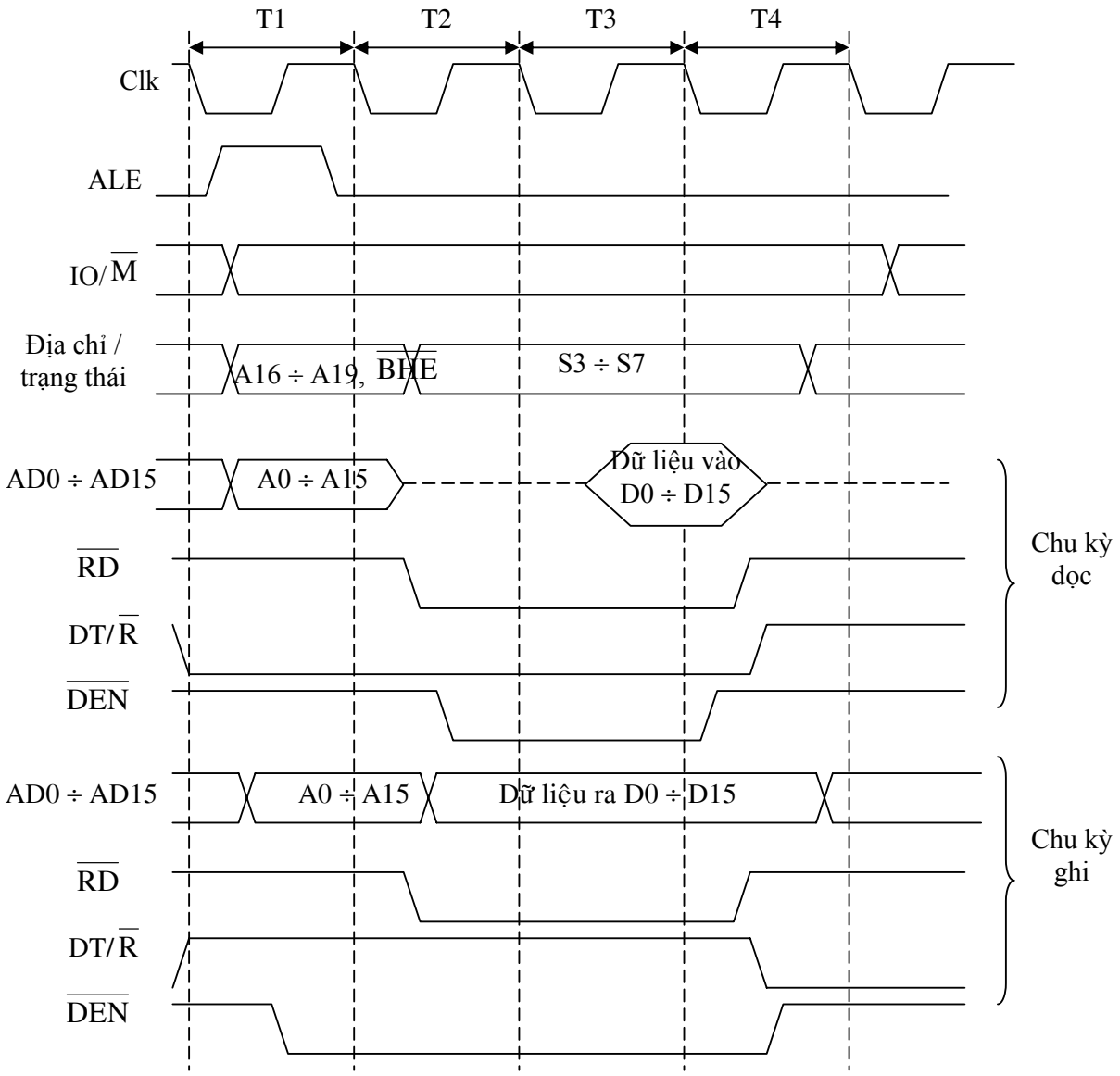


Hình 2.3 – Tạo tín hiệu điều khiển bộ nhớ và I/O

- Chân ALE (Address Latch Enable - cho phép chốt địa chỉ): tín hiệu ra trên chân này có thể dùng để phân kênh các đường địa chỉ, dữ liệu và trạng thái trên AD0 ÷ AD15, A16/S3 ÷ A19/S6 và $\overline{BHE}$/S7. Mọi chu kỳ bắt đầu với xung ALE trong trạng thái T1. Địa chỉ 20 bit được bảo đảm sẽ hợp lệ khi ALE chuyển từ mức cao xuống mức thấp.

- Chân $\overline{DEN}$ (Data Enable – cho phép dữ liệu): tín hiệu này được dùng với DT/$\overline{R}$ để cho phép nối các bộ đệm hai chiều vào data bus. Nó ngăn ngừa sự tranh chấp bus bằng cách cấm các bộ đệm dữ liệu cho đến trạng thái T2 khi các đường dữ liệu / địa chỉ không còn lưu trữ địa chỉ của bộ nhớ hay I/O.

- Chân DT/$\overline{R}$ (Data transmit/receive – truyền/nhận dữ liệu): dùng để điều khiển chiều của luồng dữ liệu qua các bộ đệm (nếu có) vào bus dữ liệu của hệ thống. Khi ở mức thấp, nó chỉ thực hiện tác vụ đọc và khi ở mức cao nó chỉ thực hiện tác vụ ghi.



Hình 2.4 – Các chu kỳ đọc và ghi của 8086

❖ **Các chân trạng thái (AD16/S3 ÷ AD19/S6 và $\overline{BHE}$/S7):**

5 tín hiệu trạng thái này được xuất ra trong các trạng thái T2 ÷ T4, dùng cho các mục đích kiểm tra. Bit S7 là bit trạng thái dư (không dùng), bit S6 luôn bằng 0, S5 mô tả trạng thái của cờ ngắt IF còn S3, S4 dùng để xác định đoạn đang sử dụng:

Bảng 2.7:

S4	S3	Đoạn
0	0	Thêm
0	1	Stack
1	0	Mã (hay không)
1	1	Dữ liệu

Tín hiệu $\overline{BHE}$/S7 (Bus High Enable) chỉ được xuất trong trạng thái T1. Khi chân này ở mức thấp, nó sẽ chỉ AD8 ÷ AD15 liên quan đến việc truyền dữ liệu. Quá trình này có thể xảy ra đối với các truy xuất bộ nhớ, I/O hay truy xuất 1 byte dữ liệu từ địa chỉ lẻ.

❖ **Bus dữ liệu (AD0 ÷ AD15):**

16 chân này tạo thành bus dữ liệu hai chiều. Các đường này chỉ hợp lệ trong các trạng thái T2 ÷ T4. Trong trạng thái T1, chúng giữ 16 bit thấp của địa chỉ bộ nhớ hoặc I/O.

❖ **Bus địa chỉ (AD0 ÷ AD15 và AD16/S3 ÷ AD19/S6):**

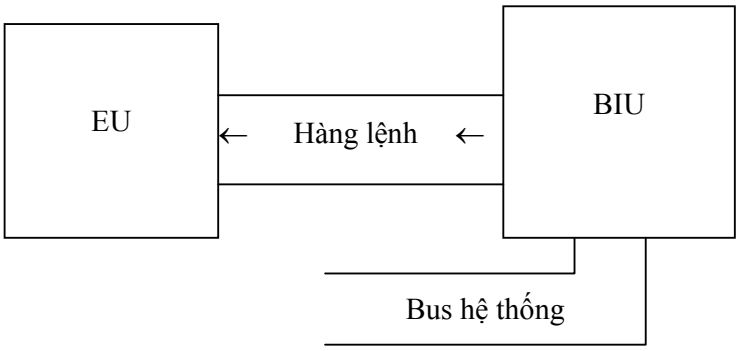
20 chân này tương ứng với bus địa chỉ 20 bit và cho phép µP truy xuất 1 MB vị trí bộ nhớ. Các đường ra này chỉ hợp lệ trong trạng thái T1, chuyển thành các đường dữ liệu và trạng thái trong trạng thái T2 ÷ T4.

❖ **Chọn chế độ $\overline{MX}$:**

Chân này dùng để chọn chế độ hoạt động cho 8086, nếu ở mức cao thì sẽ hoạt động ở chế độ tối thiểu còn ở mức thấp thì sẽ hoạt động ở chế độ tối đa.

2.2. Kiến trúc nội

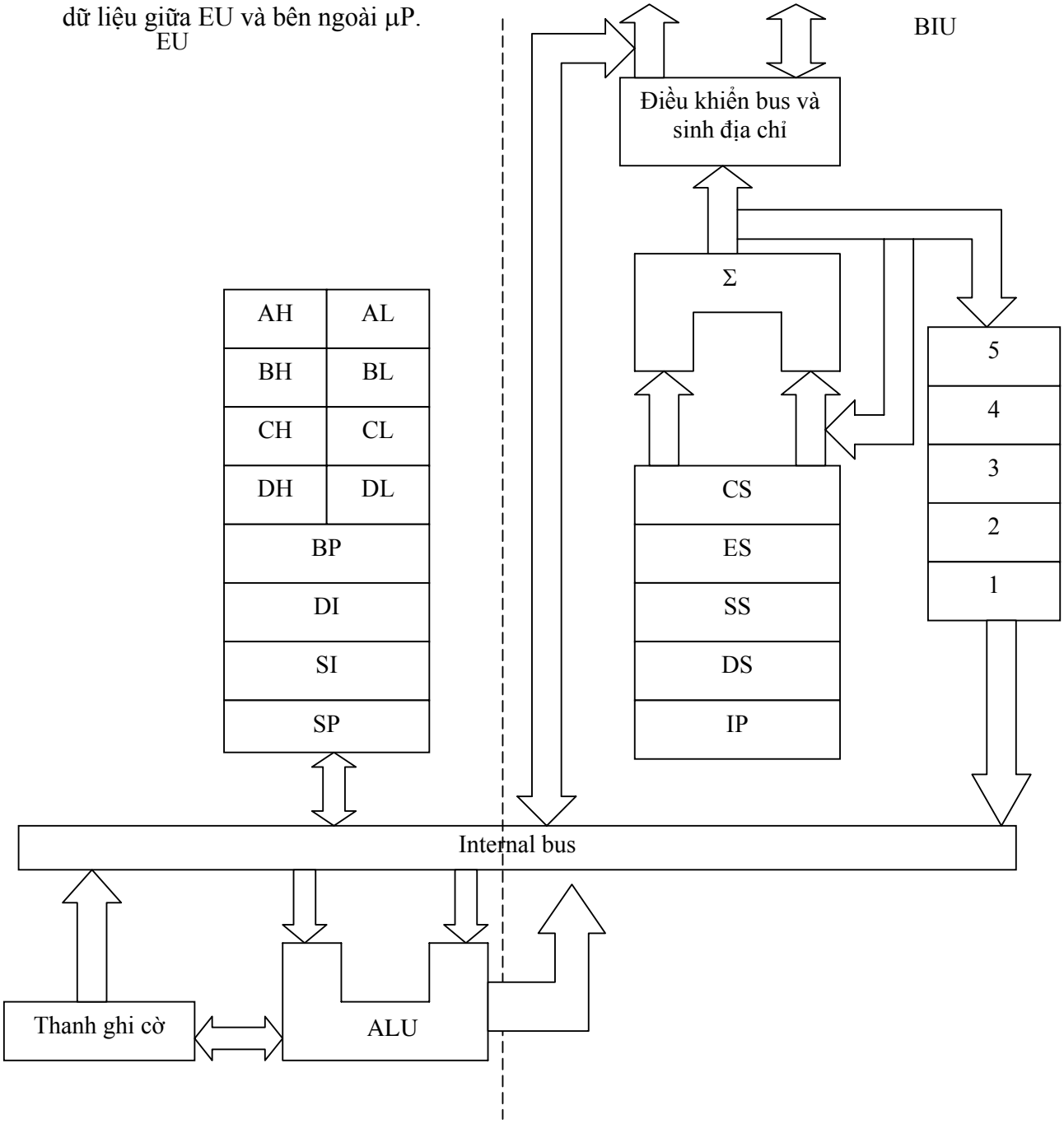
µP có khả năng thực hiện các tác vụ dữ liệu theo tập lệnh bên trong. Một lệnh được ghi nhận bằng mã đã được định nghĩa trước, gọi là mã lệnh (opcode). Trước khi thực thi một lệnh, µP phải nhận được mã lệnh từ bộ nhớ chương trình của nó. Quá trình xử lý này gọi là chu kỳ nhận lệnh (fetch cycle). Một khi các mã được nhận và được giải mã thì mạch bên trong µP có thể tiến hành thực thi (execute) mã lệnh.



Hình 2.5 – Kiến trúc nội của µP 8086

BIU (Bus Interface Unit – đơn vị giao tiếp bus) nhận các mã lệnh từ bộ nhớ và đặt chúng vào hàng chờ lệnh. EU (Execute Unit – đơn vị thực thi) sẽ giải mã và thực hiện các lệnh trong hàng. Chú ý rằng các đơn vị EU và BIU làm việc độc lập với nhau nên BIU có khả năng đang nhận một lệnh mới trong khi EU đang thực thi lệnh trước đó. Khi EU đã thực hiện xong lệnh, nó sẽ lấy mã lệnh kế tiếp trong hàng đợi lệnh (instruction queue).

Kiến trúc nội của µP 8086 ở hình 2.2. Nó có 2 bộ xử lý riêng: BIU và EU. BIU cung cấp các chức năng phần cứng, bao gồm tạo các địa chỉ bộ nhớ và I/O để chuyển dữ liệu giữa EU và bên ngoài µP.



Hình 2.6 – Kiến trúc nội của 8086

EU nhận các mã lệnh chương trình và dữ liệu từ BIU, thực thi các lệnh này và chứa các kết quả trong các thanh ghi. Ngoài ra, dữ liệu cũng có thể chứa trong một vị trí bộ nhớ hay được ghi vào thiết bị xuất. Chú ý rằng EU không có bus hệ thống nên phải thực hiện nhận và xuất tất cả các dữ liệu của nó thông qua BIU.

Sự khác biệt giữa µP 8086 và 8088 là BIU. Trong 8088, đường bus dữ liệu là 8 bit trong khi của 8086 là 16 bit. Ngoài ra hàng lệnh của 8088 dài 4 byte trong khi của 8086 là 6 byte.

Tuy nhiên do EU giữa hai loại µP này giống nhau nên ***các chương trình viết cho 8086 có thể chạy được trên 8088 mà không cần thay đổi gì cả***.

Quá trình nhận lệnh và thực thi lệnh:

1/ BIU xuất nội dung của thanh ghi con trỏ lệnh IP (Instruction Pointer) ra bus địa chỉ để chọn byte hay word đọc vào BIU.

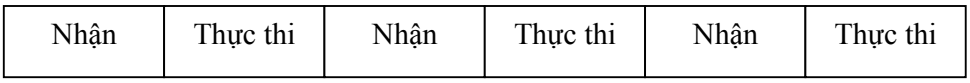
2/ Thanh ghi IP được tăng thêm 1 để chuẩn bị nhận lệnh kế.

3/ Khi lệnh ở trong BIU, nó được đưa sang hàng lệnh (queue). Đây là một thanh ghi lưu trữ dạng FIFO (First In First Out – Vào trước ra trước), dùng cơ chế xử lý xen kẽ liên tục các dòng mã lệnh (kỹ thuật đường ống – pipelining).

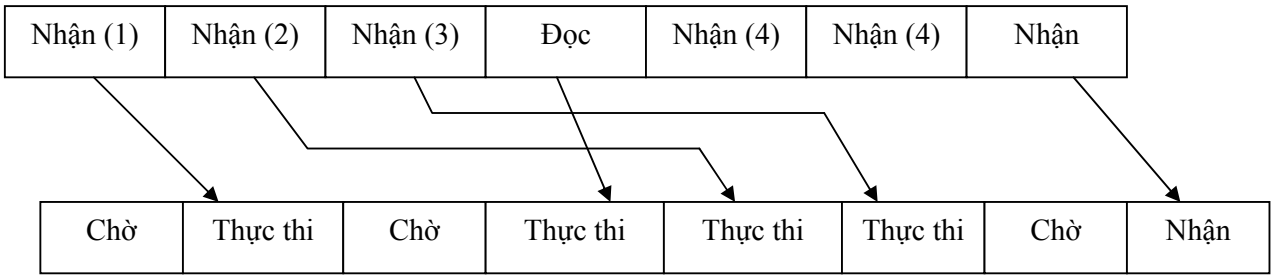
4/ Giả sử ban đầu hàng lệnh trống, EU sẽ không làm gì cả cho đến khi bắt đầu xuất hiện một lệnh trong hàng, EU sẽ lấy lệnh ra khỏi hàng và bắt đầu thực thi lệnh đó.

5/ Trong khi EU đang thực thi lệnh, BIU tiến hành nhận lệnh mới. Tuỳ theo thời gian thực thi lệnh mà BIU có thể đưa vào hàng lệnh nhiều lệnh mới trước khi EU thực hiện lệnh xong và tiếp tục lấy lệnh mới.

BIU được lập trình để có thể nhận một lệnh mới bất kỳ lúc nào hàng lệnh có chỗ cho 1 byte (8088) hay 2 byte (8086). Lợi ích của phương pháp xử lý theo cơ chế pipeline là EU có thể thực thi các lệnh gần như liên tục thay vì phải đợi BIU nhận thêm lệnh mới.



(a)



(b)

(1): lệnh thực thi không cần dữ liệu trong hàng
(2): lệnh thực thi cần dữ liệu trong hàng
(3): lệnh nhảy
(4): các lệnh bị bỏ qua do lệnh nhảy

Hình 2.7

(a) µP thông thường dùng chu kỳ nhận và thực thi lệnh tuần tự
(b) kiến trúc dạng pipeline của 8086/8088 cho phép thực thi các lệnh mà không bị trễ do quá trình nhận lệnh

Có 3 điều kiện làm cho EU ở chế độ chờ:

- Điều kiện thứ nhất xảy ra khi lệnh cần truy xuất đến một vị trí bộ nhớ không ở trong hàng. BIU phải treo quá trình nhận lệnh và xuất ra địa chỉ của ô nhớ này. Sau khi truy xuất bộ nhớ, EU có thể tiếp tục quá trình thực thi lệnh từ hàng lệnh và BIU có thể tiếp tục đưa các lệnh vào hàng.

- Điều kiện thứ hai xảy ra khi lệnh được thực thi là lệnh nhảy (jump). Trong trường hợp này, thay vì dùng địa chỉ lệnh kế tiếp, ta phải chuyển đến địa chỉ mới (không tuần tự). Tuy nhiên, BIU vẫn luôn đặt các lệnh theo tuần tự và do đó sẽ lưu các lệnh không sử dụng. Trong khi nhận lệnh kế tiếp tại địa chỉ do lệnh jump chỉ đến, EU phải đợi và tất cả các byte trong hàng phải bỏ.

- Điều kiện thứ ba có thể làm BIU treo quá trình nhận lệnh đó là khi thực thi các lệnh có thời gian thực thi lớn. Giả sử như lệnh AAM (ASCII Adjust for Multiplication) cần 83 chu kỳ xung nhịp để hoàn tất trong khi đó với 4 chu kỳ xung nhịp cho quá trình nhận lệnh thì hàng sẽ bị đầy. Như vậy BIU phải đợi cho đến khi lệnh được thực hiện xong và EU nhận mã lệnh từ hàng thì mới có thể tiếp tục quá trình nhận lệnh.

2.3. Các 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)

2.3.1. 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

2.3.2. 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.

2.3.3. 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.

2.3.4. 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.

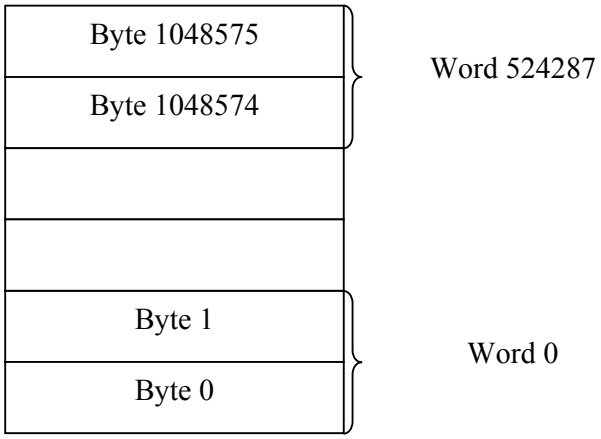
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- 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.

2.4. Phân đoạn bộ nhớ

Ta biết rằng dù 8086 là µP 16 bit (có bus dữ liệu 16 bit) nhưng vẫn dùng bộ nhớ theo các byte. Điều này cho phép µP làm việc với byte cũng như word, nó rất quan trọng trong giao tiếp với các thiết bị I/O như máy in, thiết bị đầu cuối và modem (chúng được thiết kế để chuyển dữ liệu mã hoá ASCII 7 hay 8 bit). Ngoài ra, nhiều mã lệnh của 8086/8088 có chiều dài 1 byte nên cần phải truy xuất được các byte riêng biệt để có thể xử lý các lệnh này.

8086/8088 có bus địa chỉ 20 bit nên có thể cho phép truy xuất 2^{20} = 1048576 địa chỉ bộ nhớ khác nhau.

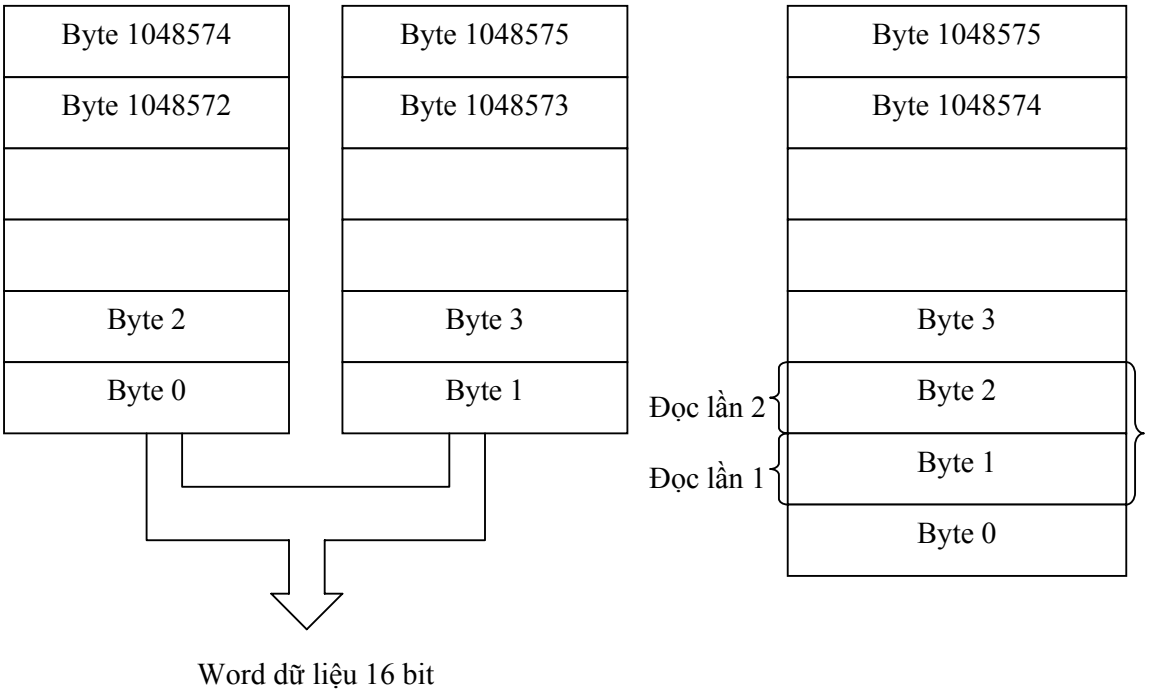


Hình 2.8 – Vùng nhớ của 8086/8088 có 1048576 byte hay 524288 word

Để thực hiện đọc 16 bit từ bộ nhớ, 8086 sẽ thực hiện đọc đồng thời byte có địa chỉ lẻ và byte có địa chỉ chẵn. Do đó, 8086 tổ chức bộ nhớ thành các bank chẵn và lẻ. Theo hình 2.8, ta có thể thấy rằng các word luôn bắt đầu tại địa chỉ chẵn nhưng ta vẫn

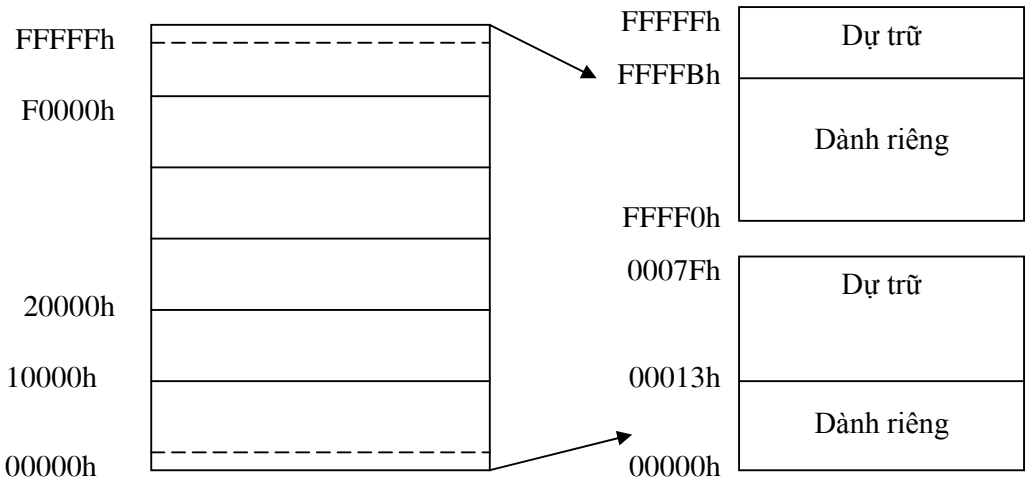
có thể đọc word có địa chỉ lẻ bằng cách thực hiện 2 chu kỳ đọc bộ nhớ: một chu kỳ đọc byte thấp và một chu kỳ đọc byte cao. Điều này sẽ làm chậm tốc độ xử lý.

Đối với 8088 thì do bus dữ liệu 8 bit nên dù word có địa chỉ chẵn hay lẻ, nó cũng cần phải thực hiện 2 chu kỳ đọc hay ghi bộ nhớ và giao tiếp với bộ nhớ như một bank.



Hình 2.9 – Đọc word địa chỉ chẵn và địa chỉ lẻ

Ngoài ra bộ nhớ cũng chia thành 16 khối, mỗi khối có kích thước 64 KB, bắt đầu ở địa chỉ 00000h và kết thúc ở FFFFFh. Địa chỉ bắt đầu mỗi khối sẽ tăng lên 1 ở số hex có ý nghĩa nhiều nhất khi thay đổi từ khối này sang khối kia. Ví dụ như khối 00000h → 10000h → 20000h …



Hình 2.10 – Bảng bộ nhớ cho 8086/8088

❖ **Các thanh ghi phân đoạn:**

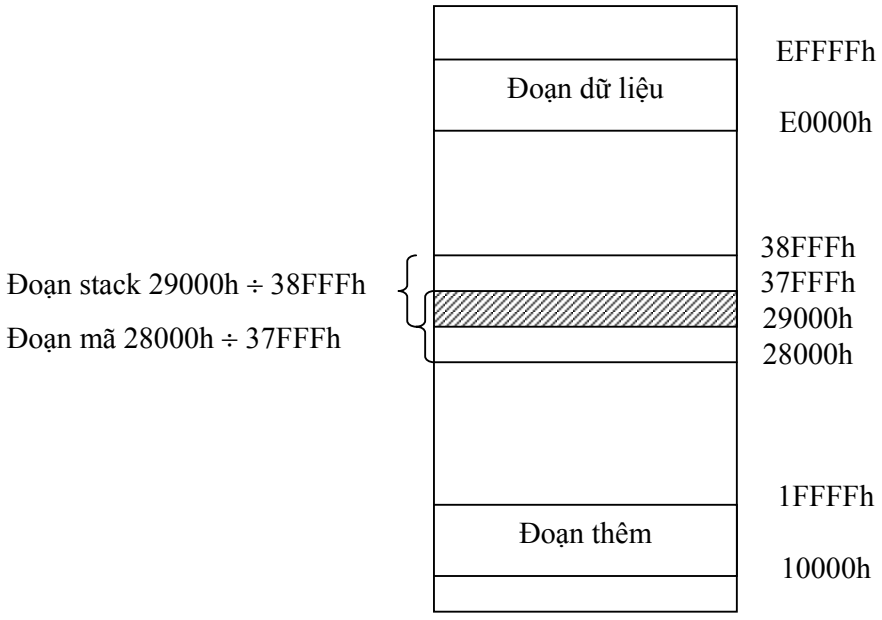
8086/8088 định nghĩa 4 khối bộ nhớ 64KB: đoạn mã (code segment) giữ các mã lệnh chương trình, đoạn ngăn xếp (stack segment) lưu các địa chỉ sẽ trả về từ các chương trình con (subroutine) hay trình phục vụ ngắt (interrupt subroutine), đoạn dữ liệu (data segment) lưu trữ dữ liệu cho chương trình và đoạn thêm (extra segment) thường dùng cho các dữ liệu dùng chung.

Các thanh ghi đoạn (CS, DS, SS và ES) dùng để chỉ vị trí nền của mỗi đoạn. Các thanh ghi này có 16 bit trong khi địa chỉ bộ nhớ là 20 bit nên để xác dịnh vị trí bộ nhớ, ta sẽ thêm 4 bit 0 vào các bit thấp của thanh ghi đoạn. Giả sử như thanh ghi CS chứa giá trị 1111h thì nó sẽ chỉ tới địa chỉ nền là 11110h. Chú ý rằng địa chỉ bắt đầu một đoạn không thể tuỳ ý mà phải bắt đầu tại một địa chỉ chia hết cho 16. Nghĩa là 4 bit thấp phải là 0. Ta cũng chú ý rằng 4 đoạn có thể không tách rời nhau mà chồng lấp lên nhau và ta cũng có thể cho 4 giá trị của các thanh ghi đoạn bằng nhau nghĩa là 4 đoạn này trùng nhau.

VD: Thanh ghi DS có giá trị là 1000h thì địa chỉ nền là 10000h. Địa chỉ kết thúc tìm được bằng cách cộng địa chỉ nền với giá trị FFFFh (64K) → địa chỉ kết thúc là 10000h + FFFFh = 1FFFFh. Như vậy đoạn dữ liệu có địa chỉ từ 10000h = 1FFFFh.

Các vị trí bộ nhớ không được định nghĩa trong các đoạn hiện hành không thể truy xuất được. Muốn truy xuất đến các vị trí đó, ta phải định nghĩa lại một trong các thanh ghi đoạn sau cho đoạn phải chứa vị trí đó. Như vậy, tại một thời điểm bất kỳ ta chỉ có thể truy xuất tối đa 4 × 64 KB = 256 KB bộ nhớ. Nội dung của các thanh ghi đoạn chỉ có thể xác định thông qua phần mềm.

VD: Giả sử các thanh ghi đoạn có các giá trị CS = 2800h, DS = E000h, SS = 2900h và ES = 1000h. Ta có vị trí các đoạn trong bảng bộ nhớ như sau:



Hình 2.11 – Vị trí các phân đoạn theo giá trị các thanh ghi đoạn

❖ **Địa chỉ logic và địa chỉ vật lý:**

Các địa chỉ trong một đoạn thay đổi từ 0000h ÷ FFFFh, tương ứng với chiều dài đoạn là 64 KB. Một địa chỉ trong một đoạn được gọi là ***địa chỉ logic hay offset***. Ví dụ như địa chỉ logic 0010h của đoạn mã trong hình 2.11 sẽ có địa chỉ thật sự là 28000h + 0010h = 28010h. Địa chỉ này gọi là ***địa chỉ vật lý***.

Như vậy, địa chỉ vật lý chính là địa chỉ thật sự xuất hiện ở bus địa chỉ, nó có chiều dài 20 bit còn địa chỉ logic là độ lệch (offset) từ vị trí 0 của một đoạn cho trước.

VD: Giả sử xét các đoạn như hình 2.11. Địa chỉ vật lý tương ứng với địa chỉ logic 1000h trong đoạn stack là:

29000h + 1000h = 2A000h

Địa chỉ vật lý tương ứng với địa chỉ logic 2000h trong đoạn mã là:

28000h + 2000h = 2A000h

Ta thấy rằng có thể địa chỉ vật lý trùng nhau khi địa chỉ logic khác nhau nghĩa là một địa chỉ vật lý có thể có nhiều địa chỉ logic khác nhau.

Để chỉ địa chỉ logic 1000h trong đoạn mã, ta dùng ký hiệu CS:1000h. Tương tự như vậy cho các đoạn khác, nghĩa là địa chỉ logic 1111h trong đoạn dữ liệu sẽ là DS:1111h.

Mọi lệnh tham chiếu bộ nhớ sẽ có một thanh ghi đoạn mặc nhiên. Thanh ghi IP cung cấp địa chỉ offset khi truy xuất đến đoạn mã và BP cho đoạn stack. Ví dụ như IP = 1000h và CS = 2000h thì BIU sẽ truy xuất đến địa chỉ 20000h + 1000h = 21000h và nhận byte tại vị trí này.

Bảng 2.9:

Tham chiếu bộ nhớ	Đoạn mặc nhiên	Đoạn khác	Offset
Nhận lệnh	CS	Không	IP
Tác vụ stack	SS	Không	SP
Dữ liệu tổng quát	DS	CS,ES,SS	Địa chỉ hiệu dụng
Nguồn của string	DS	CS,ES,SS	SI
Đích của string	ES	Không	DI
BX dùng làm con trỏ	DS	CS,ES,SS	Địa chỉ hiệu dụng
BP dùng làm con trỏ	SS	CS,ES,SS	Địa chỉ hiệu dụng

VD: Ta sử dụng lệnh MOV [BP],AL với BP = 2C00h. Ở đây BP dùng làm con trỏ nên dùng đoạn stack. Giả sử các phân đoạn như hình 2.11 thì địa chỉ vật lý sẽ là 29000h + 2C00h = 2BC00h

❖ **Định nghĩa các vị trí bộ nhớ:**

Thông thường ít khi nào ta cần biết đến địa chỉ vật lý của một vị trí bộ nhớ mà ta chỉ quan tâm đến địa chỉ logic của nó mà thôi. Lý do là vì địa chỉ vật lý còn phải phụ thuộc vào nội dung của các thanh ghi đoạn ngay cả khi địa chỉ logic giữ không đổi như đã xét ở trên.

Khi viết các chương trình hợp ngữ, thường gán cho các địa chỉ logic bằng các nhãn (label) hay các tên (name). Ví dụ:

```
DATA      SEGMENT
SAMPLEB   DB    ?
DATA      ENDS
```

sẽ gán nhãn SAMPLE cho byte ở địa chỉ logic 0 trong đoạn dữ liệu. Các phát biểu này không phải là các lệnh µP mà chỉ là các lệnh giả (pseudo instruction) dùng cho các chương trình dịch.

Toán tử DATA SEGMENT báo cho chương trình dịch biết các lệnh theo sau sẽ nằm trong đoạn dữ liệu. Toán tử DB (Define Byte) gán cho nhãn SAMPLEB 1 byte trong đoạn dữ liệu. Ký hiệu ? xác định rằng không cần định nghĩa nội dung của byte đó. Do SAMPLEB là dòng đầu tiên nên nó sẽ có địa chỉ logic là 0. Phát biểu DATA ENDS kết thúc đoạn dữ liệu (ở đây chỉ dịnh nghĩa 1 byte). Trong trường hợp muốn định nghĩa 1 word, ta dùng toán tử DW (Define Word).

VD:

```
SAMPLEW   DW    1000h
```

Phát biểu này định nghĩa nhãn SAMPLEW ứng với vị trí word và nội dung của vị trí này là 1000h.

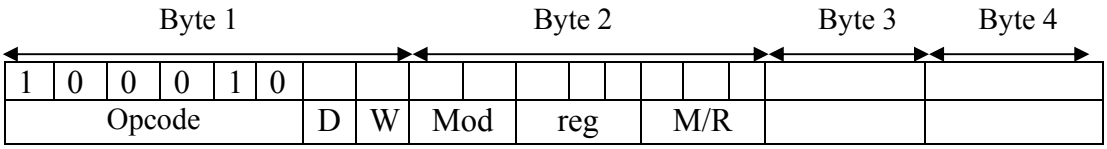
Ngoài ra, ta có thể dùng các toán tử DD định nghĩa từ kép (double word), DQ định nghĩa từ bộ bốn (8 byte) và DT định nghĩa 10 byte.

3. Cách mã hoá lệnh

Lệnh của bộ vi xử lý sẽ biểu diễn bằng các ký tự dưới dạng gợi nhớ (mnemonic) để có thể dễ dàng sử dụng. Đối với vi xử lý thì các lệnh được biểu diễn bằng các mã lệnh (opcode) nên sau khi nhận lệnh vi xử lý phải thực hiện giải mã lệnh rồi mới thực thi nó. Một lệnh vi xử lý có thể dài 1 byte hay nhiều byte. Nếu ta dùng 1 byte để mã hoá thì sẽ mã hoá được 256 lệnh khác nhau. Tuy nhiên do một lệnh không phải chỉ có một cách thực hiện nên ta không thể thực hiện đơn giản như trên.

Để tìm hiểu cách mã hoá lệnh, ta xét lệnh **MOV des,src** dùng để chuyển dữ liệu giữa hai thanh ghi hay một ô nhớ và một thanh ghi.

Lệnh MOV mã hoá như sau:



Để mã hóa lệnh MOV, ta cần dùng ít nhất là 2 byte trong đó 6 bit dùng cho mã lệnh.

Bit D xác định hướng truyền của dữ liệu, D = 0 xác định dữ liệu sẽ đi từ thanh ghi cho bởi 3 bit Reg, D = 1 xác định dữ liệu sẽ đi đến thanh ghi cho bởi 3 bit Reg.

Bit W xác định sẽ truyền 1 byte (W = 0) hay 1 word (W = 1).

3 bit Reg dùng để chọn thanh ghi sử dụng:

Bảng 2.10:

Mã	Thanh ghi	
	W = 1	W = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	DI	CH
110	BP	DH
111	SI	BH

2 bit mod và 3 bit R/M (Register / Memory) dùng để xác định chế độ địa chỉ cho các toán hạng của lệnh.

Bảng 2.11:

MOD / R/M	00	01	10	11	
				W = 1	W = 0
000	[BX]+[SI]	[BX]+[SI]+addr8	[BX]+[SI]+addr16	AX	AL
001	[BX]+[DI]	[BX]+[DI]+addr8	[BX]+[DI]+addr16	CX	CL
010	[BP]+[SI]	[BP]+[SI] +addr8	[BP]+[SI] +addr16	DX	DL
011	[BP]+[DI]	[BP]+[DI] +addr8	[BP]+[DI] +addr16	BX	BL
100	[SI]	[SI] +addr8	[SI] +addr16	SP	AH
101	[DI]	[DI] +addr8	[DI] +addr16	BP	CH
110	addr16	[BP] +addr8	[BP] +addr16	SI	DH
111	[BX]	[BX] +addr8	[BX] +addr16	DI	BH

Tổng quát, 8086/8088 có khoảng 300 tác vụ có thể có trong tập lệnh của nó. Mỗi lệnh kéo dài từ 1 đến 6 byte. Từ ví dụ trên, ta thấy mã lệnh có các vùng:

- Vùng mã lệnh (opcode): chứa mã lệnh của lệnh sẽ thực thi
- Vùng thanh ghi (reg): chứa các thanh ghi sẽ thực hiện (bảng 2.10)
- Vùng chế độ (mod): (bảng 2.11)

00: toán hạng bộ nhớ trực tiếp nếu R/M = 110, ngược lại là toán hạng gián tiếp
01: toán hạng gián tiếp, độ dời 8 bit
10: toán hạng gián tiếp, độ dời 16 bit
11: sử dụng 2 thanh ghi, vùng R/M sẽ là vùng Reg

- Vùng thanh ghi / bộ nhớ R/M (Reg/Mem): (bảng 2.11)

4. Các cách định địa chỉ

Bảng 2.12:

Cách định địa chỉ	Mã đối tượng	Ví dụ			
		Từ gợi nhớ	Đoạn truy xuất	Hoạt động	Mô tả
Tức thời	B80010	MOV AX,1000h	Mã	AH ← 10h AL ← 00h	(1)
Thanh ghi	8BD1	MOV DX,CX	Trong µP	DX ← CX	(2)
Trực tiếp	8A260010	MOV AH,[1000h]	Döõ lieäu	AH ← [1000h]	(3)
Gián tiếp thanh ghi	8B04 FF25 FE4600 FF0F	MOV AX,[SI] JMP [DI] INC BYTE PTR [BP] DEC WORD PTR [BX]	Dữ liệu Dữ liệu Stack Dữ liệu	AL ← [SI]; AH ←[SI+1] IP←[DI+1:DI] [BP]←[BP]+1 [BX+1:BX]← [BX+1:BX]-1	(4)
Có chỉ số	8B4406 FF6506	MOV AX,[SI+6] JMP [DI+6]	Dữ liệu Dữ liệu	AL ← [SI+6]; AH ←[SI+7] IP←[DI+7:DI+6]	(5)
Có nền	8B4602 FF6702	MOV AX,[BP+2] JMP [BP+2]	Stack Dữ liệu	AL←[BP+2]; AH ←[BP+3] IP←[BX+3:BX+6]	(6)
Có nền và có chỉ số	8B00 FF21 FE02 FF0B	MOV AX,[BX+SI] JMP [BX+DI] INC BYTE PTR [BP+SI] DEC WORD PTR [BP+DI]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI];AH←[BX+SI+1] IP←[BX+DI+1:BX+DI] [BP+SI]←[BP+SI]+1 [BP+DI+1:BP+DI]← [BP+DI+1:BP+DI]-1	(7)
Có nền và có chỉ số với độ dời	8B4005 FF6105 FE4205 FF4B05	MOV AX,[BX+SI+5] JMP [BX+DI+5] INC BYTE PTR [BP+SI+5] DEC WORD PTR [BP+DI+5]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI+5] AH←[BX+SI+1] IP←[BX+DI+6:BX+DI+5] [BP+SI+5]←[BP+SI+5]+1 [BP+DI+6:BP+DI+5]← [BP+DI+6:BP+DI+5]-1	(8)
String	A4	MOVSB	Thêm, dữ liệu	[ES:DI] ← [DS:DI] Nếu DF = 0 thì SI ← SI + 1; DI ← DI + 1 Nếu DF = 1 thì SI ← SI - 1; DI ← DI - 1	(9)

- BYTE PTR và WORD PTR tránh lầm giữa truy xuất byte và word.
- Độ dời được cộng vào thanh ghi con trỏ hay nền là số nhị phân dạng bù 2.
- (1): nguồn dữ liệu trong lệnh
- (2): đích và nguồn là các thanh ghi của µP
- (3): địa chỉ bộ nhớ cung cấp trong lệnh
- (4): địa chỉ bộ nhớ cung cấp trong thanh ghi con trỏ hay chỉ số
- (5): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số cộng với độ dời trong lệnh
- (6): địa chỉ bộ nhớ là tổng của thanh ghi BX hay BP cộng với độ dời trong lệnh
- (7): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số và thanh ghi nền

- (8): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số, thanh ghi nền và độ dời trong lệnh
- (9): địa chỉ nguồn bộ nhớ là thanh ghi SI trong đoạn dữ liệu và địa chỉ đích bộ nhớ là thanh ghi DI trong đoạn thêm

4.1. Định địa chỉ tức thời

Các lệnh dùng cách định địa chỉ tức thời lấy dữ liệu trong lệnh làm một phần của lệnh. Trong cách này, dữ liệu sẽ được chứa trong đoạn mã thay vì trong đoạn dữ liệu. Dữ liệu cho lệnh MOV AX,1000h được cung cấp tức thời sau mã lệnh B8. Chú ý rằng trong mã đối tượng byte dữ liệu cao đi sau byte dữ liệu thấp.

Cách định địa chỉ tức thời thường dùng để nạp một thanh ghi hay vị trí bộ nhớ với các dữ liệu ban đầu. Sau đó, các lệnh kế tiếp sẽ làm việc với các dữ liệu này. Tuy nhiên, cách định địa chỉ này không sử dụng được cho các thanh ghi đoạn.

4.2. Định địa chỉ thanh ghi

Một số lệnh chỉ làm công việc chuyển dữ liệu giữa các thanh ghi của µP. Ví dụ như MOV DX,CX sẽ chuyển dữ liệu từ thanh ghi CX vào thanh ghi DX. Ở đây ta không cần thực hiện tham chiếu bộ nhớ.

Ta có thể kết hợp cách định địa chỉ tức thời và định địa chỉ thanh ghi để nạp dữ liệu cho các thanh ghi đoạn.

VD:

```
MOV  AX, 1000h
MOV CS,AX
```

Sau khi thực hiện 2 lệnh này, giá trị của thanh ghi CS sẽ là 1000h.

4.3. Định địa chỉ trực tiếp

Ngoài 2 cách định địa chỉ trên, tất cả các cách định địa chỉ còn lại cho trong bảng 2.6 đều cần phải truy xuất đến bộ nhớ với ít nhất một toán hạng. Trong cách định địa chỉ trực tiếp, địa chỉ bộ nhớ được cung cấp trực tiếp như là một phần của lệnh. Ví dụ như lệnh MOV AH,[1000h] sẽ đưa nội dung chứa trong ô nhớ DS:1000h vào thanh ghi AH hay lệnh MOV [2000h],AX sẽ đưa nội dung chứa trong AX vào 2 ô nhớ liên tiếp DS:2000h và DS:2001h

4.4. Định địa chỉ truy xuất bộ nhớ gián tiếp

Các cách định địa chỉ trực tiếp sẽ thuận lợi cho các truy xuất bộ nhớ không thường xuyên. Tuy nhiên, nếu một ô nhớ cần phải truy xuất nhiều lần trong một chương trình thì quá trình nhận địa chỉ (2 byte) sẽ phải thực hiện nhiều lần. Điều này sẽ không hiệu quả.

Để giải quyết vấn đề này, ta thực hiện lưu trữ địa chỉ của ô nhớ cần truy xuất trong một thanh ghi con trỏ, chỉ số hay thanh ghi cơ sở (BX, BP, SI hay DI). Ngoài ra, ta có thể sử dụng độ dời bù 2 bằng cách cộng vào các thanh ghi để dời đi so với vị trí được các thanh ghi chỉ đến.

Bảng 2.13:

Cách định địa chỉ	Địa chỉ hiệu dụng (EA – Effective Address)		
	Độ dời	Thanh ghi nền	Thanh ghi chỉ số
Gián tiếp thanh ghi	Không	BX hay BP	Không
	Không	Không	SI hay DI
Có chỉ số	-128 ÷ 127	Không	SI hay DI
Có nền	-128 ÷ 127	BX hay BP	Không
Có nền và chỉ số	Không	BX hay BP	SI hay DI
Có nền và chỉ số với độ dời	-128 ÷ 127	BX hay BP	SI hay DI

Như vậy, một độ dời có thể được cộng vào thanh ghi nền và kết quả này được cộng tiếp vào thanh ghi chỉ số. Địa chỉ thu được gọi là địa chỉ hiệu dụng EA.

Ngoài ra ta cũng có thể viết cách định địa chỉ gián tiếp như sau:

MOV AX,table[SI]

Trong đó ***table*** là nhãn gán cho một vị trí ô nhớ nào đó. Lệnh này sẽ truy xuất phần tử thứ SI trong dãy ***table*** (giả sử SI = 2 thì sẽ truy xuất phần tử thứ 2). Ta cũng có thể viết lệnh trên như sau:

MOV AX,[table + SI]

Chú ý rằng các đoạn mặc định cho các cách định địa chỉ gián tiếp là đoạn stack khi dùng BP, là đoạn dữ liệu khi dùng BX, SI hay DI.

VD: Lệnh:

```
MOV  AH,10h          thực hiện định địa chỉ tức thời
MOV  AX,[BP + 10]    thực hiện định địa chỉ có nền
MOV  AH,[BP + SI]    thực hiện định địa chỉ có nền và có chỉ số
```

4.5. Định địa chỉ chuỗi

Chuỗi là một dãy liên tục các byte hay word lưu trữ trong bộ nhớ dưới dạng các ký tự ASCII. 8086/8088 có các lệnh dùng để xử lý chuỗi, các lệnh này sử dụng cặp thanh ghi DS:SI để chỉ nguồn chuỗi ký tự và ES:DI để chỉ đích chuỗi. Lệnh MOVSB sẽ chuyển byte dữ liệu nguồn đến vị trí đích trong đó SI và DI sẽ tăng hay giảm tuỳ theo giá trị của DF (xem 2.3.4 và bảng 2.13)

4.6. Thay đổi thanh ghi đoạn mặc định

Như đã nói ở phần trên, khi sử dụng các lệnh định địa chỉ thanh ghi, ta chỉ cần dùng các thanh ghi để xác định độ lệch còn các thanh ghi đoạn thì được hiểu mặc định. Ví dụ như ta dùng lệnh MOV AH,[BP] thì sẽ đưa dữ liệu tại ô nhớ SS:BP vào thanh ghi AH. Trong trường hợp không muốn dùng thanh ghi đoạn mặc định, ta có thể thay đổi bằng cách thêm tên thanh ghi đoạn vào để loại bỏ thanh ghi đoạn mặc định. Ví dụ lệnh MOV AH,CS:[BP] sẽ đưa dữ liệu tại CS:[BP] vào AH.

CHƯƠNG 3: LẬP TRÌNH HỢP NGỮ

1. Các tập tin .EXE và .COM

DOS chỉ có thể thi hành được các tập tin dạng .COM và .EXE. Tập tin .COM thường dùng để xây dựng cho các chương trình nhỏ còn .EXE dùng cho các chương trình lớn.

1.1. Tập tin .COM

- Tập tin .COM chỉ có một đoạn nên kích thước tối đa của một tập tin loại này là 64 KB.
- Tập tin .COM được nạp vào bộ nhớ và thực thi nhanh hơn tập tin .EXE nhưng chỉ áp dụng được cho các chương trình nhỏ.
- Chỉ có thể gọi các chương trình con dạng near.

Khi thực hiện tập tin .COM, DOS định vị bộ nhớ và tạo vùng nhớ dài 256 byte ở vị trí 0000h, vùng này gọi là PSP (Program Segment Prefix), nó sẽ chứa các thông tin cần thiết cho DOS. Sau đó, các mã lệnh trong tập tin sẽ được nạp vào sau PSP ở vị trí 100h và đưa giá trị 0 vào stack. Như vậy, kích thước tối đa thực sự của tập tin .COM là 64 KB – 256 byte PSP – 2 byte stack.

Tất cả các thanh ghi đoạn đều chỉ đến PSP và thanh ghi con trỏ lệnh IP chỉ đến 100h, thanh ghi SP có giá trị 0FFFEh.

1.2. Tập tin .EXE

- Nằm trong nhiều đoạn khác nhau, kích thước thông thường lớn hơn 64 KB.
- Có thể gọi được các chương trình con dạng near hay far.
- Tập tin .EXE chứa một header ở đầu tập tin để chứa các thông tin điều khiển cho tập tin.

2. Khung của một chương trình hợp ngữ

Khung của một chương trình hợp ngữ có dạng như sau:

```
TITLE       Chương trình hợp ngữ
.MODEL      Kiểu kích thước bộ nhớ     ; Khai báo quy mô sử
                                       ; dụng bộ nhớ
.STACK      Kích thước                 ; Khai báo dung lượng
                                       ; đoạn stack
.DATA                                  ; Khai báo đoạn dữ  liệu
msg   DB    'Hello$'
.CODE                                  ; Khai báo đoạn mã
main  PROC
…
CALL        Subname                    ; Gọi chương trình con
…
main  ENDP

Subname     PROC                       ; Định nghĩa chương
                                       ; trình con
…
```


```
RET
Subname     ENDP
END   main
```

❖ **Quy mô sử dụng bộ nhớ:**

Bảng 3.1:

Loại	Mô tả
Tiny	Mã lệnh và dữ liệu nằm trong một đoạn
Small	Mã lệnh trong một đoạn, dữ liệu trong một đoạn
Medium	Mã lệnh không nằm trong một đoạn, dữ liệu trong một đoạn
Compact	Mã lệnh trong một đoạn, dữ liệu không nằm trong một đoạn
Large	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và không có mảng nào lớn hơn 64KB
Huge	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và các mảng có thể lớn hơn 64KB

Thông thường, các ứng dụng đơn giản chỉ đòi hỏi mã chương trình không quá 64 KB và dữ liệu cũng không lớn hơn 64 KB nên ta sử dụng ở dạng Small:

```
.MODEL      SMALL
```

❖ **Khai báo kích thước stack:**

Khai báo stack dùng để dành ra một vùng nhớ dùng làm stack (chủ yếu phục vụ cho chương trình con), thông thường ta chọn khoảng 256 byte là đủ để sử dụng (nếu không khai báo thì chương trình dịch tự động cho kích thước stack là 1 KB):

```
.STACK      256
```

❖ **Khai báo đoạn dữ liệu:**

Đoạn dữ liệu dùng để chứa các biến và hằng sử dụng trong chương trình.

❖ **Khai báo đoạn mã:**

Đoạn mã dùng chứa các mã lệnh của chương trình. Đoạn mã bắt đầu bằng một chương trình chính và có thể có các lệnh gọi chương trình con (CALL).

Một chương trình chính hay chương trình con bắt đầu bằng lệnh PROC và kết thúc bằng lệnh ENDP (đây là các lệnh giả của chương trình dịch). Trong chương trình con, ta sử dụng thêm lệnh RET để trả về địa chỉ lệnh trước khi gọi chương trình con.

3. Cú pháp của các lệnh trong chương trình hợp ngữ

Một dòng lệnh trong chương trình hợp ngữ gồm có các trường (field) sau (không nhất thiết phải đầy đủ tất cả các trường):

Tên	Lệnh	Toán hạng	Chú thích
A:	MOV	AH,10h	; Đưa giá trị 10h vào thanh ghi AH
Main	PROC		

Trường tên chứa nhãn, tên biến hay tên thủ tục. Các tên nhãn có thể chứa tối đa 31 ký tự, không chứa ký tự trắng (space) và không được bắt đầu bằng số (A: hay Main:). Các nhãn được kết thúc bằng dấu '**:**'.

Trường lệnh chứa các lệnh sẽ thực hiện. Các lệnh này có thể là các lệnh thật (MOV) hay các lệnh giả (PROC). Các lệnh thật sẽ được dịch ra mã máy.

Trường toán hạng chứa các toán hạng cần thiết cho lệnh (AH,10h).

Trường chú thích phải được bắt đầu bằng dấu '**;**'. Trường này chỉ dùng cho người lập trình để ghi các lời giải thích cho chương trình. Chương trình dịch sẽ bỏ qua các lệnh nằm phía sau dấu **;**.

3.1. Khai báo dữ liệu

Khi khai báo dữ liệu trong chương trình, nếu sử dụng số nhị phân, ta phải dùng thêm chữ **B** ở cuối, nếu sử dụng số thập lục phân thì phải dùng chữ **H** ở cuối. ***Chú ý rằng đối với số thập lục phân, nếu bắt đầu bằng chữ A..F thì phải thêm vào số 0 ở phía trước.***

Ví dụ:

```
1011b       ; Số nhị phân
1111        ; Số thập phân
1011h       ; Số thập lục phân
```

3.2. Khai báo biến

Khai báo biến nhằm để chương trình dịch cung cấp một địa chỉ xác định trong bộ nhớ. Ta dùng các lệnh giả sau để định nghĩa các biến ứng với các kiểu dữ liệu khác nhau: DB (define byte), DW (define word) và DD (define double word).

VD:

```
A1      DB      1       ; Định nghĩa biến A1 dài 1 byte (chương
                        ; trình dịch sẽ dùng 1 byte trong bộ nhớ để
                        ; lưu trữ A1), trị ban đầu A1 = 1
A2      DB      ?       ; Biến A2 kiểu byte, không có giá trị gán
                        ; ban đầu
A3      DB      'A'     ; Biến kiểu ký tự
A4      DW      1       ; Định nghĩa biến A4 dài 2 byte, giá trị ban
                        ; đầu A4 = 1, ta cũng có thể dùng dấu ? để
                        ; xác định biến không cần khởi tạo giá trị ban đầu
A5      DD      1       ; Biến A5 dài 4 byte
A6      DB      1,2,3   ; Định nghĩa biến mảng (array) gồm có 3
                        ; phần tử, mỗi phần tử dài 1 byte (nghĩa là
                        ; sẽ dùng 3 byte lưu trữ) với các giá trị ban
                        ; đầu của các phần tử lần lượt là 1,2,3
A7      DB      10      DUP(0)
                        ; Khai báo biến mảng gồm 10 phần tử, mỗi
                        ; phần tử có chiều dài 1 byte với giá trị gán
                        ; ban đầu là 0
```


```
A8    DB    10    DUP(?)
                  ; Khai báo biến mảng gồm 10 phần tử, mỗi
                  ; phần tử có chiều dài 1 byte, không cần
                  ; gán giá trị ban đầu
```

Ngoài ra ta có thể dùng các toán tử DUP lồng vào nhau khi khai báo biến mảng. Giả sử ta cần khai báo mảng A9 có các giá trị gán ban đầu 1,2,3,1,1,3,2,2,1,1,3,2,2. Ta có thể thực hiện như sau:

```
      A9    DB    1,2,3,1,1,3,2,2,1,1,3,2,2
Hay:  A9    DB    1,2,3,2 DUP(1,1,3,2,2)
Hay:  A9    DB    1,2,3,2 DUP(2 DUP(1),3,2 DUP(2))
```

Đối với các biến có nhiều hơn 1 byte, byte thấp sẽ chứa ở ô nhớ có địa chỉ thấp và byte cao sẽ chứa ở ô nhớ có địa chỉ cao.

VD:

```
A10   DW    1234h
```

Biến A10 giả sử bắt đầu lưu tại địa chỉ 1000h thì ô nhớ 1000h chứa giá trị 34h còn ô nhớ 1001h chứa giá trị 12h.

Đối với biến kiểu chuỗi (string), thực chất là một mảng các ký tự, ta có thể khai báo như sau:

```
      A11   DB    'ABCD'
Hay   A11   DB    65h,66h,67h,68h
```

Sau lệnh khai báo này thì ô nhớ 1000h (giả sử biến A11 lưu trữ tại địa chỉ 1000h) chứa 'A', 1001h chứa 'B', 1002h chứa 'C' và 1003h chứa 'D'.

3.3. Khai báo hằng

Các hằng khai báo trong chương trình hợp ngữ bằng lệnh giả EQU để chương trình dễ hiểu hơn. Hằng có thể ở dạng số, ký tự hay chuỗi.

VD:

```
A12   EQU   10
A13   EQU   'AAA'
```

Sau khi sử dụng khai báo này, nếu ta dùng lệnh:

```
MOV  AH,A12
```

thì AH = 10h

```
A14   DB    'B',A13
```

thì khai báo chuỗi A14 với giá trị gán ban đầu là 'BAAA'.

4. Các toán tử trong hợp ngữ

❖ **Toán tử số học:**

Bảng 3.2:

Toán tử	Cú pháp	Mô tả
+	+bt	Số dương
-	-bt	Số âm
*	bt1*bt2	Nhân
/	bt1/bt2	Chia
mod	bt1 mod bt2	Lấy phần dư
+	bt1 + bt2	Cộng
-	bt1 – bt2	Trừ
shl	bt shl n	Dịch trái n bit
shr	bt shr n	Dịch phải n bit

Trong đó bt, bt1, bt2 là các biểu thức hằng, n là số nguyên.

```
VD:  MOV  AH,(8+1)*7/3          ; AH ← 21
     MOV  AH, 00010001b shr 2   ; AH ← 0000 0100b
     MOV  AH,00010001b shl 2    ; AH ← 0100 0100b
     MOV  AH,100 mod 3          ; AH ← 1
```

❖ **Toán tử logic:**

Bao gồm các toán tử AND, OR, NOT, XOR

```
VD:  MOV  AH,10 OR 4 AND 2      ; AH = 10
     MOV  AH, 0F0h AND 7Fh      ; AH = 70h
```

❖ **Toán tử quan hệ:**

Các toán tử quan hệ so sánh 2 biểu thức, cho giá trị true (-1) nếu điều kiện thoả và false (0) nếu không thoả.

Bảng 3.3:

Toán tử	Cú pháp	Mô tả
EQ	bt1 EQ bt2	Bằng
NE	bt1 NE bt2	Không bằng
LT	bt1 LT bt2	Nhỏ hơn
LE	bt1 LE bt2	Nhỏ hơn hay bằng
GT	bt1 GT bt2	Lớn hơn
GE	bt1 GE bt2	Lớn hơn hay bằng

❖ **Các toán tử cung cấp thông tin:**

➢ **Toán tử SEG:**

SEG bt

Toán tử SEG xác định địa chỉ đoạn của biểu thức ***bt***. ***bt*** có thể là biến, nhãn, hay các toán hạng bộ nhớ.

➢ **Toán tử OFFSET:**

OFFSET bt

Toán tử OFFSET xác định địa chỉ offset của biểu thức ***bt***. ***bt*** có thể là biến, nhãn, hay các toán hạng bộ nhớ.

```
VD:   MOV  AX,SEG A          ; Nạp địa chỉ đoạn và địa chỉ offset
      MOV  DS,AX             ; của biến A vào cặp thanh ghi
      MOV  AX,OFFSET A       ; DS:AX
```

➢ **Toán tử chỉ số [ ]: (index operator)**

Toán tử chỉ số thường dùng với toán hạng trực tiếp và gián tiếp.

➢ **Toán tử (:) (segment override operator)**

Segment:bt

Toán tử **:** quy định cách tính địa chỉ đối với segment được chỉ. ***Segment*** là các thanh ghi đoạn CS, DS, ES, SS.

Chú ý rằng khi sử dụng toán tử : kết hợp với toán tử [ ] thì ***segment:*** phải đặt ngoài toán tử [ ].

VD: Cách viết [CS:BX] là sai, ta phải viết CS:[BX]

➢ **Toán tử TYPE:**

TYPE bt

Trả về giá trị biểu thị dạng của biểu thức ***bt***.

- Nếu ***bt*** là biến thì sẽ trả về 1 nếu biến có kiểu byte, 2 nếu biến có kiểu word, 4 nếu biến có kiểu double word.
- Nếu ***bt*** là nhãn thì trả về 0FFFFh nếu bt là near và 0FFFEh nếu bt là far.
- Nếu ***bt*** là hằng thì trả về 0.

➢ **Toán tử LENGTH:**

LENGTH bt

Trả về số các đơn vị cấp cho biến ***bt***

➢ **Toán tử SIZE:**

SIZE bt

Trả về tổng số các byte cung cấp cho biến ***bt***

```
VD:   A      DD    100    DUP(?)
      MOV  AX,LENGTH A     ; AX = 100
      MOV  AX,SIZE A       ; AX = 400
```


❖ **Các toán tử thuộc tính:**

➢ **Toán tử PTR:**

Loai PTR bt

Toán tử này cho phép thay đổi dạng của biểu thức ***bt***.

- Nếu ***bt*** là biến hay toán hạng bộ nhớ thì ***Loai*** là byte, word hay dword.
- Nếu ***bt*** là nhãn thì ***Loai*** là near hay far.

```
VD:  A     DW   100   DUP(?)
     B     DD   ?
     MOV  AH,BYTE PTR A        ; Đưa byte đầu tiên trong mảng A
                               ; vào thanh ghi AH
     MOV  AX,WORD PTR B        ; Đưa 2 byte thấp trong biến B
                               ; vào thanh ghi AX
```

➢ **Toán tử HIGH, LOW:**

HIGH bt

LOW bt

Cho giá trị của byte cao và thấp của biểu thức ***bt***, ***bt*** phải là một hằng.

```
VD:  A     EQU  1234h
     MOV  AH,HIGH A            ; AH ← 12h
     MOV  AH,LOW A             ; AH ← 34h
```

5. Các cách định địa chỉ trong hợp ngữ

❖ **Toán hạng trực tiếp:**

Toán hạng trực tiếp là một biểu thức hằng xác định. Các hằng số có thể ở dạng thập phân (có dấu và không dấu), nhị phân, thập lục phân, các hằng số định nghĩa bằng lệnh EQU, …

```
VD:  MOV  AH,10
     MOV  AH,1010b
     MOV  AH,0Ah
     MOV  AH,A12
     MOV  AX,OFFSET msg
     MOV  AX,SEG msg
```

❖ **Toán hạng thanh ghi:**

Các thanh ghi có thể sử dụng trong phép định địa chỉ thanh ghi là AH, BH, CH, DH, AL, BL, CL, DL, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, ES, SS.

❖ **Toán hạng bộ nhớ:**

➢ **Trực tiếp:**

Toán hạng này xác định dữ liệu lưu trong bộ nhớ tại một địa chỉ xác định khi dịch, địa chỉ này là một biểu thức hằng (có thể kết hợp với toán tử chỉ số [ ] hay toán tử +, -, :). Thanh ghi đoạn mặc định là thanh ghi DS nhưng ta có thể dùng toán tử : để chỉ thanh ghi đoạn khác.

```
VD:  A    DW   1000h
     B    DB   100   DUP(0)
     MOV  AX,A              ; Chuyển nội dung của biến A vào
     MOV  AX,[A]            ; thanh ghi AX
     MOV  AH,B              ; Truy xuất phần tử đầu tiên của
     MOV  AH,B[0]           ; mảng B
     MOV  AH,B + 1          ; Truy xuất phần tử thứ hai của
     MOV  AH,B[1]           ; mảng B
     MOV  AH,B  + 5         ; Truy xuất phần tử thứ 6 của
     MOV  AH,B[5]           ; mảng B
```

Chú ý rằng lệnh MOV AX,[1000h] sẽ chuyển giá trị 1000h vào thanh ghi AX. Nếu muốn chuyển nội dung tại ô nhớ 1000h vào thanh ghi AX thì phải dùng lệnh MOV AX,DS:[1000h] hay MOV AX,DS:1000h

- **Gián tiếp:**

Toán hạng bộ nhớ gián tiếp cho phép dùng các thanh ghi BX, BP, SI, DI để chỉ các giá trị trong bộ nhớ.

```
VD:  MOV  BX,2
     MOV  SI,3
     MOV  AH,B[BX]          ; Chuyển phần tử thứ 3 của mảng B
                            ; vào thanh ghi AH
     MOV  AH,B[BX+1]        ; Chuyển phần tử thứ 4 của mảng B
     MOV  AH,B[BX]+1        ; vào thanh ghi AH (BX + 1 = 3)
     MOV  AH,B[BX+SI]       ; Chuyển phần tử thứ 6 của mảng B
     MOV  AH,B[BX][SI]      ; vào thanh ghi AH
     MOV  AH,[B+BX+SI]      ; BX + SI = 5
     MOV  AH,[B][BX][SI]
     MOV  AH,B[BX+SI+5]     ; Chuyển phần tử thứ 11 của mảng B
     MOV  AH,B[BX][SI]+5    ; vào thanh ghi AH
     MOV  AH,[B+BX+SI+5]    ; BX + SI + 5 = 10
```

6. Tạo và thực thi chương trình hợp ngữ

Ta có thể tạo và thực thi một chương trình hợp ngữ trên một máy PC theo các bước sau:

- Dùng một chương trình soạn thảo văn bản ***không định dạng*** (như NC) tạo một tập tin chứa chương trình hợp ngữ (gán phần mở rộng của tập tin này là .ASM, giả sử là TEMP.ASM).
- - Dùng chương trình TASM.EXE (Turbo Assembler) để dịch ra mã máy dạng .OBJ: **TASM TEMP**
- Sau khi dịch xong, ta sẽ được file TEMP.OBJ chứa các mã máy của chương trình. Để chuyển thành file thực thi, ta dùng chương trình TLINK.EXE để chuyển thành tập tin .EXE: **TLINK TEMP**
- - Nếu tập tin thực thi ở dạng .COM thì ta dùng thêm chương trình EXE2BIN.EXE: **EXE2BIN TEMP TEMP.COM**

7. Tập lệnh hợp ngữ

7.1. Nhóm lệnh chuyển dữ liệu

7.1.1. Nhóm lệnh chuyển dữ liệu đa dụng

❖ Lệnh **MOV dst,src**: chuyển nội dung toán hạng src vào toán hạng dst. Toán hạng nguồn src có thể là thanh ghi (reg), bộ nhớ (mem) hay giá trị tức thời (immed); toán hạng đích dst có thể là reg hay mem.

Lệnh MOV có thể có các trường hợp sau:

Reg8 ← reg8	MOV AL,AH
Reg16 ← reg16	MOV AX,BX
Mem8 ← reg8	MOV [BX],AL
Reg8 ← mem8	MOV AL,[BX]
Mem16 ← reg16	MOV [BX],AX
Reg16 ← mem16	MOV AX,[BX]
Reg8 ← immed8	MOV AL,04h
Mem8 ← immed8	MOV mem[BX],01h
Reg16 ← immed16	MOV AL,0F104h
Mem16 ← immed16	MOV mem[BX],0101h
SegReg ← reg16	MOV DS,AX
SegReg ← mem16	MOV DS,mem
Reg16 ← segreg	MOV AX,DS
Mem16 ← segreg	MOV [BX],DS

- Lệnh MOV không ảnh hưởng đến các cờ.
- Không thể chuyển trực tiếp dữ liệu giữa hai ô nhớ mà phải thông qua một thanh ghi

```
MOV  AX,mem1
MOV  mem2,AX
```

- Không thể chuyển giá trị trực tiếp vào thanh ghi đoạn

```
MOV  AX,1010h
MOV  DS,AX
```

- Không thể chuyển trực tiếp giữa 2 thanh ghi đoạn
- Không thể dùng thanh ghi CS làm toán hạng đích

❖ Lệnh **XCHG dst,src**: (Exchange) hoán chuyển nội dung 2 toán hạng. Toán hạng chỉ có thể là reg hay mem.

- Lệnh XCHG không ảnh hưởng đến các cờ
- Không thể dùng cho các thanh ghi đoạn

❖ Lệnh **PUSH src**: cất nội dung một thanh ghi vào stack. Toán hạng là reg16

❖ Lệnh **POP dst**: lấy dữ liệu 16 bit từ stack đưa vào toán hạng dst.

Ta có thể dùng nhiều lệnh PUSH để cất dữ liệu vào stack nhưng khi dùng lệnh POP để lấy dữ liệu ra thì phải dùng theo thứ tự ngược lại.

```
PUSH    AX
PUSH    BX
```


```
PUSH    CX
…
POP     CX
POP     BX
POP     AX
```

❖ Lệnh **XLAT [src]**: chuyển nội dung của ô nhớ 8 bit vào thanh ghi AL. Địa chỉ ô nhớ xác định bằng cặp thanh ghi DS:BX (nếu không chỉ ra src) hay src, địa chỉ offset chứa trong thanh ghi AL.

Lệnh XLAT tương đương với các lệnh:

```
MOV  AH,0
MOV  SI,AX
MOV  AL,[BX+SI]
```

7.1.2. Nhóm lệnh chuyển địa chỉ

❖ Lệnh **LEA reg16,mem16**: (Load Effective Address) chuyển địa chỉ offset của toán hạng bộ nhớ vào thanh ghi reg16.

Lệnh này sẽ tương đương với **MOV reg16, OFFSET mem16**

❖ Lệnh **LDS reg16,mem32**: (Load pointer using DS) chuyển nội dung bộ nhớ toán hạng mem32 vào cặp thanh ghi DS:reg16.

Lệnh LDS AX,mem tương đương với:

```
MOV  AX,mem
MOV  BX,mem+2
MOV  DS,BX
```

❖ Lệnh **LES reg16,mem32**: (Load pointer using ES) giống như lệnh LDS nhưng dùng cho thanh ghi ES

7.1.3. Nhóm lệnh chuyển cờ hiệu

❖ Lệnh **LAHF**: (Load AH from flag) chuyển các cờ SF, ZF, AF, PF và CF vào các bit 7,6,4,2 và 0 của thanh ghi AH (3 bit còn lại không đổi)

❖ Lệnh **SAHF**: (Store AH into flag) chuyển các bit 7,6,4,2 và 0 của thanh ghi AH vào các cờ SF, ZF, AF, PF và CF.

❖ Lệnh **PUSHF**: chuyển thanh ghi cờ vào stack

❖ Lệnh **POPF**: lấy dữ liệu từ stack chuyển vào thanh ghi cờ

7.1.4. Nhóm lệnh chuyển dữ liệu qua cổng

Mỗi I/O port giao tiếp với CPU sẽ có một địa chỉ 16 bit cho nó. CPU gởi hay nhận dữ liệu từ cổng bằng cách chỉ đến địa chỉ cổng đó. Tuỳ theo chức năng mà cổng có thể: chỉ đọc dữ liệu (input port), chỉ ghi dữ liệu (output port) hay có thể đọc và ghi dữ liệu (input/output port).

❖ Lệnh **IN**: đọc dữ liệu từ cổng và đưa vào thanh ghi AL

```
IN      AL,port8
IN      AL,DX
```

Nếu địa chỉ port chỉ có 8 bit thì có thể đưa giá trị trực tiếp vào, nếu là 16 bit thì phải thông qua thanh ghi AX.

❖ Lệnh **OUT**: ghi dữ liệu trong thanh ghi AL ra cổng

```
OUT   port8,AL
OUT   DX,AL
```

VD:
```
MOV  AL,3
OUT  61h,AL          ; Gởi giá trị 03h ra cổng 61h
MOV  AL,1
MOV  DX,03F8h        ; Xuất ra cổng máy in
OUT  DX,AL
MOV  DX,03F8h
IN   AL,DX           ; Đọc dữ liệu từ cổng máy in
```

7.2. Nhóm lệnh chuyển điều khiển

7.2.1. Lệnh nhảy không điều kiện JMP

JMP label
JMP reg/mem

Lệnh JMP dùng để chuyển điều khiển chương trình từ vị trí này sang vị trí khác (thay đổi nội dung cặp thanh ghi CS:IP).

7.2.2. Lệnh nhảy có điều kiện

Lệnh nhảy có điều kiện chỉ sử dụng cho các nhãn nằm trong khoảng từ –127 đến 128 byte so với vị trí của lệnh.

❖ Lệnh **JA label**: (Jump if Above)
Nếu CF = 0 và ZF = 0 thì JMP label

❖ Lệnh **JAE label**: (Jump if Above or Equal)
Nếu CF = 0 thì JMP label

❖ Lệnh **JB label**: (Jump if Below)
Nếu CF = 1 thì JMP label

❖ Lệnh **JBE label**: (Jump if Below or Equal)
Nếu CF = 1 hoặc ZF = 1 thì JMP label

❖ Lệnh **JNA label**: (Jump if Not Above)
Giống lệnh JBE

❖ Lệnh **JNAE label**: (Jump if Not Above or Equal)
Giống lệnh JB

❖ Lệnh **JNB label**: (Jump if Not Below)
Giống lệnh JAE

❖ Lệnh **JNBE label**: (Jump if Not Below or Equal)
Giống lệnh JA

❖ Lệnh **JG label**: (Jump if Greater)
Nếu SF = OF và ZF = 0 thì JMP label

❖ Lệnh **JGE label**: (Jump if Greater or Equal)
Nếu SF = OF thì JMP label

❖ Lệnh **JL label**: (Jump if Less)
Nếu SF <> OF thì JMP label

❖ Lệnh **JLE label**: (Jump if Less or Equal)
Nếu CF <> OF hoặc ZF = 1 thì JMP label

❖ Lệnh **JNG label**: (Jump if Not Greater)
Giống lệnh JLE

❖ Lệnh **JNGE label**: (Jump if Not Greater or Equal)
Giống lệnh JL

❖ Lệnh **JNL label**: (Jump if Not Less)
Giống lệnh JGE
❖ Lệnh **JNLE label**: (Jump if Not Less or Equal)
Giống lệnh JG

❖ Lệnh **JC label**: (Jump if Carry)
Giống lệnh JB

❖ Lệnh **JNC label**: (Jump if Not Carry)
Giống lệnh JNB

❖ Lệnh **JZ label**: (Jump if Zero)
Nếu ZF = 1 thì JMP label

❖ Lệnh **JE label**: (Jump if Equal)
Giống lệnh JZ

❖ Lệnh **JNZ label**: (Jump if Not Zero)
Nếu ZF = 0 thì JMP label

❖ Lệnh **JNE label**: (Jump if Equal)
Giống lệnh JNZ

❖ Lệnh **JS label**: (Jump on Sign)
Nếu SF = 1 thì JMP label

❖ Lệnh **JNS label**: (Jump if No Sign)
Nếu SF = 0 thì JMP label

❖ Lệnh **JO label**: (Jump on Overflow)
Nếu OF = 1 thì JMP label

❖ Lệnh **JNO label**: (Jump if No Overflow)
Nếu OF = 0 thì JMP label

❖ Lệnh **JP label**: (Jump on Parity)
Nếu PF = 1 thì JMP label

❖ Lệnh **JNP label**: (Jump if No Parity)
Nếu PF = 0 thì JMP label

❖ Lệnh **JCXZ label**: (Jump if CX Zero)
Nếu CX = 1 thì JMP label

7.2.3. Lệnh so sánh

CMP left(reg/mem), right(reg/mem/immed)

Lệnh CMP dùng để so sánh nội dung 2 toán hạng, kết quả chứa vào thanh ghi cờ và không làm thay đổi nội dung các toán hạng.

VD: Đoạn chương trình so sánh 2 số A và B: A >B thì nhảy đến label1, A = B thì nhảy đến label2, A < B thì nhảy đến label3.

```
MOV  AX,A
CMP  AX,B
JG   label1
JL   label2
JMP  label3
```

7.2.4. Các lệnh vòng lặp

❖ Lệnh **LOOP**:
LOOP label
Mô tả:
CX = CX – 1
Nếu CX <> 0 thì JMP label

❖ Lệnh **LOOPE**:
LOOPE label
Mô tả:
CX = CX – 1
Nếu (ZF = 1) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPZ**:
Giống lệnh LOOPE

❖ Lệnh **LOOPNE**:
LOOPNE label
Mô tả:
CX = CX – 1
Nếu (ZF = 0) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPNZ**:
Giống lệnh LOOPNE

7.2.5. Lệnh liên quan đến chương trình con

❖ Lệnh **CALL**:
Lệnh CALL dùng để gọi một chương trình con, có thể là near hay far.

```
CALL    label     ; Gọi chương trình con tại vị trí xác định
                  ; bởi nhãn label
CALL    reg/mem   ; Gọi chương trình con tại vị trí xác định
                  ; trong reg/mem
```

❖ Lệnh **RET**: (return)
RET [n]
RETN [n]
RETF [n]

Lệnh RET dùng để kết thúc chương trình con, điều khiển sẽ được đưa về địa chỉ trước khi gọi chương trình con. RETN để kết thúc chương trình con dạng near và RETF dùng để kết thúc chương trình con dạng far.

Trong trường hợp lệnh RET có hằng số n theo sau thì sẽ cộng với thanh ghi SP giá trị n (n phải là số chẵn). Lệnh này dùng để loại bỏ một số tham số chương trình con sử dụng ra khỏi stack.

7.3. Nhóm lệnh xử lý số học

7.3.1. Xử lý phép cộng

❖ Lệnh **ADD dst,src**:
dst ← dst + src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst là reg hay mem.

- Không thể cộng trực tiếp 2 thanh ghi đoạn
- Lệnh ADD ảnh hưởng đến các cờ sau:
  + Cờ CF: = 1 khi kết quả phép cộng có nhớ hay có mượn
  + Cờ AF: = 1 khi kết quả phép cộng có nhớ hay có mượn đối với 4 bit thấp
  + Cờ PF: = 1 khi kết quả phép cộng có tổng 8 bit thấp là một số chẵn.
  + Cờ ZF: = 1 khi kết quả phép cộng là 0.
  + Cờ SF: = 1 nếu kết quả phép cộng là một số âm
  + Cờ OF: = 1 nếu kết quả phép cộng bị sai dấu, nghĩa là vượt ra ngoài phạm vi lớn nhất hay nhỏ nhất mà số có dấu có thể chứa trong toán hạng dst.

❖ Lệnh **ADC dst, src**: (Add with Carry)

dst ← dst + src + CF

Lệnh ADC thường dùng để cộng các số lớn hơn 16 bit.

❖ Lệnh **INC dst**: (Increment)

dst ← dst + 1

Dst có thể là reg hay mem.

❖ Lệnh **AAA**: (ASCII Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte).

```
VD:   MOV  AX,9
      MOV  BX,3
      ADD  AL,BL          ; Kết quả là AX = 0Ch
      AAA                 ; AX = 0102h (AH = 1, AL = 2)
```

Lệnh AAA chỉ ảnh hưởng đến các cờ AF và CF, không ảnh hưởng đến các cờ còn lại.

❖ Lệnh **DAA**: (Decimal Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit, nghĩa là 1 byte biểu diễn được các số nguyên từ 0 đến 99).

```
VD:   MOV  AX,4338h
      ADD  AL,AH        ; AX ← 437Bh
      DAA               ; AX ← 4381h (43 + 38 = 81)
```

Lệnh DAA chỉ ảnh hưởng đến các cờ AF, CF, PF, SF, ZF và không ảnh hưởng đến thanh ghi AH.

7.3.2. Xử lý phép trừ

❖ Lệnh **SUB dst,src**:

dst ← dst - src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst chỉ có thể là reg hay mem.

- Không thể trừ trực tiếp thanh ghi đoạn
- Ảnh hưởng đến các cờ AF, CF, OF, PF, SF và ZF.

❖ Lệnh **SBB dst,src**:

dst ← dst – src – CF

Lệnh ADC thường dùng để trừ các số lớn hơn 16 bit.

❖ Lệnh **DEC dst**: (decrement)

dst ← dst – 1

dst là reg hay mem. Lệnh DEC ảnh hưởng đến các cờ AF, OF, PF, SF, ZF.

❖ Lệnh **NEG dst**:

dst ← - *dst*

dst là reg hay mem.

Lệnh NEG ảnh hưởng đến các cờ:

CF = 1 nếu nội dung kết quả là số khác 0.

SF = 1 nếu nội dung kết quả là số âm khác 0.

PF = 1 nếu tổng 8 bit thấp là một số chẵn.

ZF = 1 nếu nội dung kết quả là 0.

OF = 1 nếu nội dung toán hạng dst là 80h (dạng byte) hay 8000h (dạng word).

VD: Nếu muốn thực hiện phép toán 100 – AH, ta không thể cùng lệnh:

```
SUB   100,AH
```

mà phải dùng lệnh:

```
SUB   AH,100
NEG   AH
```

❖ Lệnh **AAS**: (Ascii Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

❖ Lệnh **DAS**: (Decimal Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

7.3.3. Xử lý phép nhân

❖ Lệnh **MUL src**:

Nếu src là reg hay mem 8 bit: AX ← AL*src

Nếu src là reg hay mem 16 bit: DX:AX ← AX*src

Lệnh MUL chỉ ảnh hưởng đến cờ CF và OF.

❖ Lệnh **IMUL src**:

Giống như lệnh MUL nhưng kết quả là số có dấu.

❖ Lệnh **AAM**: (Ascii Adjust for Multiple)

Hiệu chỉnh kết quả phép nhân 2 số BCD dạng không nén, lệnh AAM thực hiện chia AL cho 10, lưu phần thương vào AL và phần dư vào AH. Lệnh AAM ảnh hưởng đến các cờ PF, SF và ZF.

7.3.4. Xử lý phép chia

❖ Lệnh **DIV src**:

Nếu src là reg/mem 8 bit: AL ← AX DIV src và AH ← AX MOD src

Nếu src là reg/mem 16 bit: AX ← DX:AX DIV src và DX ← DX:AX MOD src

Lệnh DIV không ảnh hưởng đến các cờ nhưng xảy ra tràn trong các trường hợp sau:

- Chia cho 0

- Thương lớn hơn 256 đối với dạng 8 bit.
- Thương lớn hơn 65536 đối với dạng 16 bit.

❖ Lệnh **IDIV src**:

Giống như lệnh DIV nhưng kết quả là số có dấu. Các trường hợp tràn:

- Chia cho 0
- Thương nằm ngoài khoảng (-128,127) đối với dạng 8 bit.
- Thương nằm ngoài khoảng (-32767,32768) đối với dạng 16 bit.

❖ Lệnh **AAD**: (Ascii Adjust for Division)

Hiệu chỉnh kết quả phép chia 2 số BCD dạng không nén. Lưu ý rằng lệnh AAD phải được thực hiện trước lệnh chia. Sau khi thực hiện chia thì phải hiệu chỉnh lại dạng BCD bằng cách dùng lệnh AAM.

❖ Lệnh **CBW**: (Convert Byte to Word)

Nếu AL < 80h thì AH = 0, ngược lại AH = 0FFh

Lệnh CBW dùng để chuyển số nhị phân có dấu 8 bit thành số nhị phân có dấu 16 bit.

❖ Lệnh **CWD**: (Convert Word to Double word)

Nếu AX < 8000h thì DX = 0, ngược lại DX = 0FFFFh

Lệnh CWD dùng để chuyển số nhị phân có dấu 16 bit thành số nhị phân có dấu 32 bit chứa trong DX:AX.

7.3.5. Dịch chuyển và quay

❖ Lệnh **SHL**: (Shift Logical Left)

SHL dst,1
SHL dst,CL

Dịch trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← 0

❖ Lệnh **SHR**: (Shift Logical Right)

SHR dst,1
SHR dst,CL

Dịch phải 1 bit hay CL bit.

0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **SAL**: giống SHL

❖ Lệnh **SAR**:

Giống như lệnh SHR nhưng giá trị bit dst7 không thay đổi, nghĩa là

dst7 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **ROL**: (Rotate Left)

ROL dst,1
ROL dst,CL

Quay trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← dst7

❖ Lệnh **ROR**: (Rotate Right)
ROR dst,1
ROR dst,CL
Quay phải 1 bit hay CL bit.
dst0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **RCL**: (Rotate though Carry Left)
RCL dst,1
RCL dst,CL
Quay trái 1 bit hay CL bit.
CF ← dst7 ← dst6 … ← dst0 ← CF

❖ Lệnh **RCR**: (Rotate though Carry Right)
RCR dst,1
RCR dst,CL
Quay phải 1 bit hay CL bit.
CF → dst7 → dst6 … → dst0 → CF

7.3.6. Các lệnh logic

❖ Lệnh **AND**:
AND dst,src
dst ← dst AND src
CF ← 0, OF ← 0
Src là reg, mem hay immed còn dst là reg, mem.

❖ Lệnh **OR**:
OR dst,src
dst ← dst OR src
CF ← 0, OF ← 0

❖ Lệnh **XOR**:
XOR dst,src
dst ← dst XOR src
CF ← 0, OF ← 0

❖ Lệnh **NOT**:
NOT dst
dst ← NOT dst
Lệnh NOT không ảnh hưởng đến các cờ.

❖ Lệnh **TEST**:
TEST dst,src
Lệnh TEST thực hiện phép toán AND 2 toán hạng nhưng chỉ ảnh hưởng đến các cờ và không ảnh hưởng đến toán tử.

7.4. Nhóm lệnh xử lý chuỗi

Bao gồm các lệnh sau:

- Lệnh **MOVS**: chuyển dữ liệu từ vùng nhớ này sang vùng nhớ khác.

  + MOVSB: chuyển 1 byte từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF = 1 thì SI ← SI - 1, DI ← DI - 1.

  + MOVSW: chuyển 1 word từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF ← 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh **CMPS**: so sánh nội dung 2 vùng nhớ

  + CMPSB: so sánh 1 byte tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF ← 1 thì SI ← SI - 1, DI ← DI - 1.

  + CMPSW: so sánh 1 word tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF = 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh **SCAS**: tìm một phần tử trong vùng nhớ, địa chỉ vùng nhớ xác định bằng cặp thanh ghi ES:DI, giá trị cần tìm đặt trong thanh ghi AL, nếu tìm thấy thì ZF = 1. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh **LODS**: đưa một byte hay word có địa chỉ xác định bởi cặp thanh ghi DS:SI vào thanh ghi AL hay AX. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh **STOS**: chuyển nội dung của AL hay AX vào vùng nhớ xác định bởi cặp thanh ghi ES:DI. Giá trị của DI và SI thay đổi giống như trên.

8. Các cấu trúc cơ bản trong lập trình hợp ngữ

8.1. Cấu trúc tuần tự

Cấu trúc tuần tự là cấu trúc đơn giản nhất. Trong cấu trúc tuần tự, các lệnh được sắp xếp tuần tự, lệnh này tiếp theo lệnh kia.

Lệnh 1
Lệnh 2
…
Lệnh n

VD: Cộng 2 giá trị của thanh ghi BX và CX, rồi nhân đôi kết quả, kết quả cuối cùng chứa trong AX

```
MOV  AX,BX
ADD  AX,CX      ; Cộng BX với CX
SHL  AX,1       ; Nhân đôi
```


8.2. Cấu trúc IF – THEN, IF – THEN – ELSE

IF Điều kiện THEN Công việc
IF Điều kiện THEN Công việc1 ELSE Công việc2
VD: Gán BX = |AX|

```
            CMP  AX,0        ; AX > 0?
            JNL  DUONG       ; AX dương
            NEG  AX          ; Nếu AX < 0 thì đảo dấu
DUONG:      MOV  BX,AX
NEXT:
```

VD: Gán CL giá trị bit dấu của AX

```
            CMP  AX,0        ; AX > 0?
            JNS  AM          ; AX âm
            MOV  CL,1        ; CL = 1 (AX dương)
            JMP  NEXT
        AM: MOV  CL,0        ; CL = 0 (AX âm)
        NEXT:
```

8.3. Cấu trúc CASE

CASE Biểu thức
Giá trị 1: Công việc 1
Giá trị 2: Công việc 2
…
Giá trị n: Công việc n
END

VD: Nếu AX > 0 thì BH = 0, nếu AX < 0 thì BH = 1. Ngược lại BH = 2

```
        CMP  AX,0
             JL   AM
        JE   KHONG
             JG   DUONG
DUONG:  MOV  BH,0
        JMP  NEXT
AM:     MOV  BH,1
        JMP  NEXT
KHONG:  MOV  BH,2
NEXT:
```

8.4. Cấu trúc FOR

FOR Số lần lặp DO Công việc

VD: Cho vùng nhớ M dài 200 bytes trong đoạn dữ liệu, chương trình đếm số chữ A trong vùng nhớ M như sau:

```
            MOV  CX,200          ; Đếm 200 bytes
            MOV  BX,OFFSET M     ; Lấy địa chỉ vùng nhớ
            XOR  AX,AX           ; AX = 0
```


```
NEXT:   CMP  BYTE PTR [BX],'A'; So sánh với chữ A
        JNZ  ChuA              ; Nếu không phải là chữ A thì tiếp
        INC  AX                ; tục, ngược lại thì tăng AX
ChuA:   INC  BX
        LOOP       NEXT
```

8.5. Cấu trúc lặp WHILE

WHILE Điều kiện DO Công việc

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
        MOV  BX,1000h
CONT:   CMP  AH,'$'
        JZ   NEXT
        MOV  AH,DS:[BX]
        JMP  CONT
NEXT:
```

8.6. Cấu trúc lặp REPEAT

REPEAT Công việc UNTIL Điều kiện

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
        MOV  BX,1000h
CONT:   MOV  AH,DS:[BX]
        CMP  AH,'$'
        JZ   NEXT
        JMP  CONT
NEXT:
```

9. Các ngắt của 8086

Bảng 3.4:

Vector ngắt	Công dụng
00h	CPU: tác động khi chia cho 0
01h	CPU: chương trình thực thi từng bước
02h	CPU: ngắt không che được
03h	CPU: tạo điểm dừng chochương trình
04h	CPU: tác động khi kết quả số học tràn
05h	Tác động khi nhấn Print Screen
06h - 07h	Dành riêng
08h	Tác động bởi nhịp đồng hồ (18.2 lần/s)
09h	Tác động khi có phím nhấn
0Ah	Dành riêng
0Bh - 0Ch	Tác động phần cứng liên lạc nối tiếp

0Dh	Đĩa cứng
0Eh	Đĩa mềm
0Fh	Máy in
10h	BIOS: màn hình
11h	BIOS: xác định cấu hình máy tính
12h	BIOS: thông báo kích thước RAM
13h	BIOS: gọi các phục vụ đĩa cứng/mềm
14h	BIOS: giao tiếp nối tiếp
15h	BIOS: truy xuất cassette hay mở rộng ngắt
16h	BIOS: xuất / nhập bàn phím
17h	BIOS: máy in
18h	Xâm nhập ROM basic
19h	BIOS: khởi động máy tính
1Ah	BIOS: ngày / giờ hệ thống
1Bh	Lấy điều khiển từ ngắt bàn phím
1Ch	Lấy điều khiển từ ngắt đồng hồ (sau int 08h)
1Dh	Địa chỉ bảng tham số màn hình
1Eh	Địa chỉ bảng tham số đĩa
1Fh	Địa chỉ bộ mã ký tự
20h	DOS: kết thúc chương trình
21h	DOS: các chức năng DOS
22h	Địa chỉ cần chuyển khi kết thúc chương trình
23h	Địa chỉ cần chuyển khi gặp Ctrl – Break
24h	Địa chỉ cần chuyển khi gặp lỗi
25h	DOS: đọc đĩa cứng / mềm
26h	DOS: ghi đĩa cứng / mềm
27h	DOS: chấm dứt chương trình và thường trú
28h – 3Fh	Dành riêng cho DOS
40h	BIOS: các chức năng đĩa mềm
41h	Bảng thông số đĩa cứng thứ nhất
42h – 45h	Dành riêng
46h	Bảng thông số đĩa cứng thứ hai
47h – 49h	Định nghĩa do người sử dụng
4Ah	Giờ báo hiệu (chỉ trong AT)
4Bh – 67h	Định nghĩa do người sử dụng
68h – 6Fh	Không sử dụng
70h	Đồng hồ thời gian thực (chỉ trong AT)
71h – 7Fh	Dành riêng
80h – 85h	Dành riêng
86h – F0h	Sử dụng bởi chương trình thông dịch BASIC
F1h – FFh	Không sử dụng

9.1. Ngắt 21h

❖ **Hàm 01h**: nhập một ký tự từ bàn phím và hiện ký tự nhập ra màn hình. Nếu không có ký tự nhập, hàm 01h sẽ đợi cho đến khi nhập.

- Gọi: AH = 01h
- Trả về: AL chứa mã ASCII của ký tự nhập

```
MOV  AH,01h
INT  21h          ; AL chứa mã ASCII của ký tự nhập
```

❖ **Hàm 02h**: xuất một ký tự trong thanh ghi DL ra màn hình tại vị trí con trỏ hiện hành

- Gọi AH = 02h, DL = mã ASCII của ký tự
- Trả về: không có

```
MOV  AH,02h
MOV  DL,'A'
INT  21h
```

❖ **Hàm 08h**: giống hàm 01h nhưng không hiển thị ký tự ra màn hình

❖ **Hàm 09h**: xuất một chuỗi ký tự ra màn hình tại vị trí con trỏ hiện hành, địa chỉ chuỗi được chứa trong DS:DX và phải được kết thúc bằng ký tự $

- Gọi AH = 09h, DS:DX = địa chỉ chuỗi
- Trả về: không có

```
.DATA
Msg  DB   'Hello$'
…
MOV  AH,09h
LEA  DX,Msg
INT  21h
```

❖ **Hàm 0Ah**: nhập một chuỗi ký tự từ bàn phím (tối đa 255 ký tự), dùng phím ENTER kết thúc chuỗi

- Gọi AH = 0Ah, DS:DX = địa chỉ lưu chuỗi
- Trả về: không có

Chuỗi phải có dạng sau:

- Byte 0: Số byte tối đa cần đọc (kể cả ký tự Enter)
- Byte 1: số byte đã đọc
- Byte 2: lưu các ký tự đọc

```
.DATA
Msg  DB   101          ; Đọc tối đa 100 ký tự
     DB   ?
     DB   101  DUP(?)
     …
     MOV  AH,0Ah
     LEA  DX,Msg
     INT  21h
```

❖ **Hàm 4Ch**: kết thúc chương trình

```
MOV  AH,4Ch
INT  21h
```


9.2. Ngắt 10h

❖ **Xoá màn hình:**
- Gọi AX = 02h
- Trả về: không có

```
MOV  AX,02h
INT  10h
```

❖ **Chuyển toạ độ con trỏ:**
- Gọi AH = 02h, DH = dòng, DL = cột

```
MOV  AH,02h
MOV  DX,0F15h
INT  10h
```

10. Truyền tham số giữa các chương trình

Trong lập trình, một vấn đề ta cần quan tâm là truyền tham số giữa chương trình chính và chương trình con. Để thực hiện truyền tham số, ta có thể dùng các cách sau đây:
- Truyền tham số qua thanh ghi
- Truyền tham số qua ô nhớ (biến)
- Truyền tham số qua ô nhớ do thanh ghi chỉ đến
- Truyền tham số qua stack

10.1. Truyền tham số qua thanh ghi

Ta thực hiện truyền tham số qua thanh ghi bằng cách: một chương trình con sẽ đưa giá trị vào thanh ghi và chương trình con khác sẽ xử lý giá trị trên thanh ghi đó.

VD: Cộng giá trị tại 2 ô nhớ 1000h và 1001h, kết quả chứa trong 1002h (bye cao) và 1003h (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.CODE
main  PROC
      MOV       AX,@DATA
      MOV       DS,AX
      MOV       BYTE PTR DS:[1000h],10h      ; Đưa giá trị vào
      MOV       BYTE PTR DS:[1001h],0FFh     ; các ô nhớ
      CALL      Read
      CALL      Sum
      Mov       AH,4Ch
      INT       21h
main  ENDP
Read  PROC                    ; Đọc dữ liệu vào thanh ghi AX
      MOV       AH,DS:[1000h]
      MOV       AL,DS:[1001h]
      RET
Read  ENDP                    ; Xử lý dữ liệu tại thanh ghi AX
```


```
Sum   PROC
      ADD       AH,AL
      JZ        next
      MOV       DS:[1003h],1
next: MOV       DS:[1002h],AH
RET
Sum   ENDP
END   main
```

10.2. Truyền tham số qua ô nhớ (biến)

Quá trình truyền tham số cũng giống như trên nhưng thay vì thực hiện thông qua thanh ghi, ta sẽ thực hiện thông qua các ô nhớ.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (bye cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
      m1    db    ?
      m2    db    ?
      m3    db    ?
      m4    db    ?
.CODE
main  PROC
      MOV       AX,@data
      MOV       DS,AX
      MOV       m1,10h      ; Đưa giá trị vào
      MOV       m2,0FFh     ; các ô nhớ
      CALL      Sum
      MOV       AH,4Ch
      INT       21h
main  ENDP
Sum   PROC
      MOV       m4,0
      MOV       AH,m1
      ADD       AH,m2
      JNC       next
      MOV       m4,1
next: MOV       m3,AH
RET
Sum   ENDP
END   main
```

10.3. Truyền tham số qua ô nhớ do thanh ghi chỉ đến

Trong cách truyền tham số này, ta dùng các thanh ghi SI, DI, BX để chỉ địa chỉ offset của các tham số còn thanh ghi đoạn mặc định là DS.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (bye cao) và m4 (byte thấp).

```
.MODEL      SMALL
.STACK      100h
.DATA
        m1      db      ?
        m2      db      ?
        m3      db      ?
        m4      db      ?
.CODE
main    PROC
        MOV             AX,@data
        MOV             DS,AX
        LEA             SI,m1
        LEA             DI,m2
        LEA             BX,m3
        MOV             [SI],10h        ; Đưa giá trị vào
        MOV             [DI],0FFh       ; các ô nhớ
        CALL            Sum
        MOV             AH,4Ch
        INT             21h
main    ENDP
Sum     PROC
        MOV             AL,[SI]
        ADD             AL,[DI]
        JZ              next
        MOV             [BX+1],1
next:   MOV             [BX],AL
RET
Sum     ENDP
END     main
```

10.4. Truyền tham số qua stack

Trong phương pháp truyền tham số này, ta dùng stack làm nơi chứa các tham số cần truyền thông qua các tác vụ PUSH và POP.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL      SMALL
.STACK      100h
.DATA
        m1      dw      ?
        m2      dw      ?
        m3      dw      ?
        m4      dw      ?
.CODE
main    PROC
        MOV             AX,@data
```


```
                MOV     DS,AX
                LEA     SI,m1
                LEA     DI,m2
                MOV     [SI],1234h          ; Đưa giá trị vào
                MOV     [DI],0FEDCh         ; các ô nhớ
                PUSH    m1                  ; Đưa vào stack
                PUSH    m2
                CALL    Sum
                POP     m3                  ; Lấy kết quả đưa vào stack
                POP     m4
                MOV     AH,4Ch
                INT     21h
        main    ENDP
        Sum     PROC
                POP     DX    ; Lưu lại địa chỉ trả về của lệnh CALL
                POP     AX    ; Lấy dữ liệu từ stack
                POP     BX
                ADD     AX,BX
                JNC     next
                PUSH    1
        next:   PUSH    AX
                PUSH    DX    ; Trả lại địa chỉ trở về của lệnh CALL
        RET
        Sum     ENDP
        END     main
```

11. Các ví dụ minh hoạ

11.1. In chuỗi ký tự ra màn hình

```
.MODEL      SMALL
.STACK      100h
.DATA
        msg     DB      'Hello$'
.CODE
main    PROC
        MOV     AX,@DATA        ; Khởi động thanh ghi DS
        MOV     DS,AX
        MOV     AX,02h          ; Xoá màn hình
        INT     10h
        MOV     AH,02h          ; Chuyển toạ độ con trỏ
        MOV     DX,0C15h        ; đến dòng 12 (0Ch) và cột 21 (15h)
        INT     10h
        LEA     DX,msg          ; Địa chỉ thông điệp
        MOV     AH,09h          ; In thông điệp ra màn hình
        INT     21h
        MOV     AH,4Ch          ; Kết thúc chương trình
        INT     21h
main    ENDP
```


```
END   main
```

11.2. In chuỗi ký tự ra màn hình tại toạ độ nhập vào

```
.MODEL      SMALL
.STACK      100h
.DATA
        msg     DB      'Hello$'
        msg1    DB      'Nhap vao toa do:$'
        Crlf    DB      0Dh,0Ah,'$'
        Td      DB      3
                DB      ?
                DB      3       DUP(?)
.CODE
main    PROC
        MOV     AX,@DATA
        MOV     DS,AX               ; Khởi động thanh ghi DS
        MOV     AX,02h
        INT     10h                 ; Xóa màn hình
        LEA     DX,msg1
        MOV     AH,09h              ; In thông điệp
        INT     21h
        CALL    Nhap                ; Nhập dòng
        MOV     CL,AL
        LEA     DX,Crlf             ; Xuống dòng
        MOV     AH,09h
        INT     21h
        CALL    Nhap                ; Nhập cột
        MOV     CH,AL
        MOV     AH,02h              ; Chuyển tọa độ con trỏ
        MOV     DX,CX
        INT     10h
        LEA     DX,msg
        MOV     AH,09h              ; In ra màn hình
        INT     21h
        MOV     AH,4Ch              ; Kết thúc chương trình
        INT     21h
main    ENDP
Nhap    PROC
        MOV     AH,0Ah              ; Nhập vào
        LEA     DX,Td
        INT     21h
        LEA     BX,Td               ; Lấy chữ số hàng chục
        MOV     AL,DS:[BX+2]
        SUB     AL,'0'              ; Chuyển từ dạng ký tự sang dạng số
        MOV     BL,10
        MUL     BL                  ; Nhân số hàng chục với 10
        PUSH    AX
        LEA     BX,Td               ; Lấy chữ số hàng dơn vị
```


```
        MOV  AL,DS:[BX+3]
        SUB  AL,'0'
        POP  BX
        ADD  AL,BL
        RET
Nhap    ENDP
END     main
```

11.3. Cộng 2 số nhị phân dài 5 byte

```
.MODEL      SMALL
.STACK      100h
.DATA
        m1      DB      00h,08h,10h,13h,24h,00h
        m2      DB      0FFh,0FCh,0FAh,0F0h,0F1h,00h;
        m3      DB      6       DUP(0)
.CODE
main    PROC
        MOV  AX,@DATA
        MOV  DS,AX              ; Khởi động thanh ghi DS
        LEA  SI,m1
        LEA  DI,m2
        LEA  BX,m3
        MOV  CX,6
        XOR  AL,AL
next:   MOV  AL,[SI]
        ADC  AL,[DI]
        MOV  [BX],AL
        INC  BX
        INC  SI
        INC  DI
        LOOP next
        MOV  AH,4Ch
        INT  21h
main    ENDP
END     main
```

11.4. Nhập một chuỗi ký tự và chuyển chữ thường thành chữ hoa

```
.MODEL      SMALL
.STACK      100h
.DATA
        m1      DB      81
                DB      ?
                DB      81      DUP(?)
        m2      DB      'Chuoi da doi:$'
.CODE
main    PROC
        MOV  AX,@DATA
```

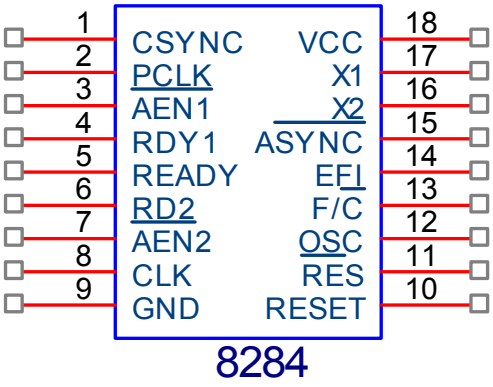

```
        MOV  DS,AX          ; Khởi động thanh ghi DS
        MOV  ES,AX
        LEA  DX,m1
        MOV  AH,0Ah         ; Nhập chuỗi
        INT  21h
        LEA  SI,m1          ; Lấy địa chỉ chuỗi
        ADD  SI,2
        MOV  DI,SI          ; Chuỗi nguồn và đích trùng nhau
Next:   LODSB               ; Lấy ký tự
        CMP  AL,0Dh         ; Nếu là ký tự Enter thì kết thúc
        JE   quit
        CMP  AL,'a'         ; Nếu ký tự nhập không phải là ký tự
        JB   cont           ; thường từ 'a' đến 'z' thì bỏ qua
        CMP  AL,'z'
        JA   cont
        SUB  AL,20h         ; Chuyển ký tự thường thành ký tự hoa
        STOSB               ; Lưu ký tự vừa chuyển
        DEC  DI             ; Nếu là ký tự thường thì dùng lệnh STOSB
                            ; nên DI tăng lên 1, ta phải giảm DI
cont:   INC  DI             ; Tăng lên ký tự kế
        JMP  next
quit:   MOV  AL,'$'
        STOSB
        MOV  AX,02h         ; Xóa màn hình
        INT  10h
        LEA  DX,m2
        MOV  AH,09h
        INT  21h
        LEA  DX,m1+2
        MOV  AH,09h
        INT  21h
        MOV  AH,4Ch
        INT  21h
main    ENDP
END     main
```


CHƯƠNG 4: TỔ CHỨC NHẬP / XUẤT

1. Các mạch phụ trợ 8284 và 8288

1.1. Mạch tạo xung nhịp 8284

Mạch tạo xung nhịp dùng để cung cấp xung nhịp cho µP.

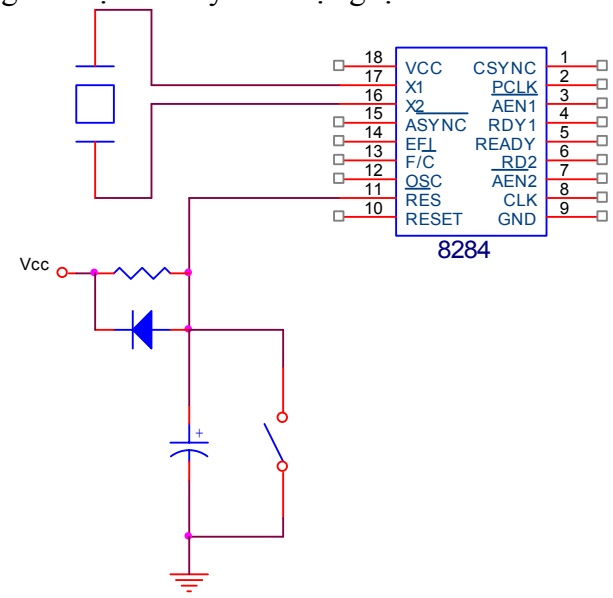


Hình 4.1 – Mạch tạo xung nhịp 8284

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 đất.

PCLK (Peripheral Clock): xung nhịp f = f_X/6 (f_X là tần số thạch anh)

$\overline{AEN1}$, $\overline{AEN2}$ (Address Enable): cho phép chọn các chân RDY1, RDY2 báo hiệu trạng thái sẵn sàng của bộ nhớ hay thiết bị ngoại vi



Hình 4.2 – Mạch khởi động cho 8284

RDY1, RDY2 (Bus ready): tạo các chu kỳ đợi ở CPU

READY: nối đến chân READY của µP.

CLK (Clock): xung nhịp f = f_X/3, nối với chân CLK của µP.

RESET: nối với chân RESET của µP, là tín hiệu khởi động lại toàn hệ thống

$\overline{RES}$ (Reset Input): chân khởi động cho 8284

OSC: ngõ ra xung nhịp 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 nhịp bên ngoài, ngược lại thì dùng xung nhịp từ thạch anh

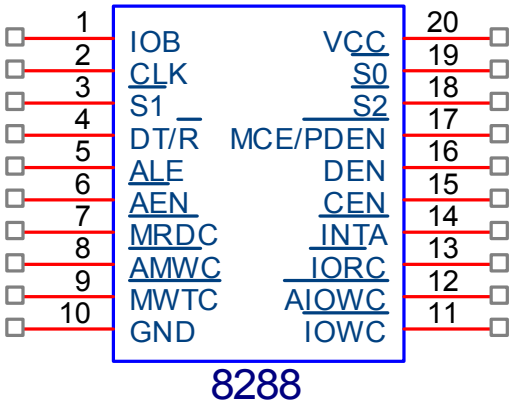
EFI (External Frequency Input): xung nhịp từ bộ dao động ngoài

$\overline{ASYNC}$: chọn chế độ làm việc cho tín hiệu RDY.

X1,X2: ngõ vào của thạch anh

1.2. Mạch điều khiển bus 8288

Mạch điều khiển bus 8288 lấy một số tín hiệu điều khiển của µP và cung cấp các tín hiệu điều khiển cần thiết cho hệ vi xử lý.



Hình 4.3 – 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 nhịp hệ thống.

$\overline{S2}$, $\overline{Sl}$, $\overline{S0}$: các tín hiệu trạng thái lấy trực tiếp từ µP. Tuỳ theo các giá trị nhận được mà 8288 sẽ đưa các tín hiệu theo bảng 4.1.

Bảng 4.1:

$\overline{S2}$	$\overline{Sl}$	$\overline{S0}$	**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

DT/$\overline{R}$ (Data Transmit/Receive): µP truyền (1) hay nhận (0) dữ liệu.

ALE (Address Latch Enable): tín hiệu cho phép 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.

$\overline{MRDC}$ (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 µP.

$\overline{IOWC}$ (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 µP.

$\overline{IORC}$ (I/O Read Command): điều khiển đọc ngoại vi

$\overline{INTA}$ (Interrupt Acknowledge): ngõ ra thông báo µP chấp nhận yêu cầu ngắt của thiết bị ngoại vi

CEN (Command Enable): cho phép đưa ra các tín hiệu của 8288.

DEN (Data Enable): tín hiệu 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.

2. Giao tiếp với thiết bị ngoại vi

2.1. Các kiểu giao tiếp vào / ra

2.1.1. Thiết bị ngoại vi có địa chỉ tách rời với bộ nhớ

Trong cách giao tiếp này, bộ nhớ dùng toàn bộ không gian 1 MB. Các thiết bị ngoại vi sẽ có một không gian 64 KB cho mỗi loại cổng. Trong kiểu giao tiếp này, ta phải dùng tín hiệu IO/$\overline{M}$ và các lệnh trao đổi dữ liệu thích hợp.

Bộ nhớ: IO/$\overline{M}$ = 0, dùng lệnh MOV

Ngoại vi: IO/$\overline{M}$ = 1, dùng lệnh IN (nhập) hay OUT (xuất)

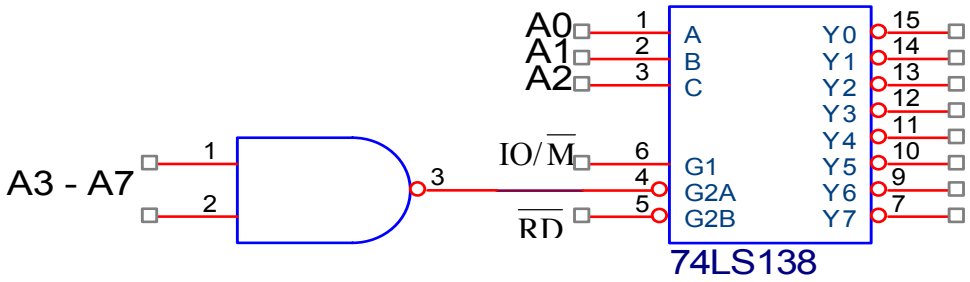
2.1.2. Thiết bị ngoại vi và bộ nhớ có chung không gian địa chỉ

Trong kiểu giao tiếp này, thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian địa chỉ 1 MB và ta chỉ dùng lệnh MOV để thực hiện trao đổi dữ liệu.

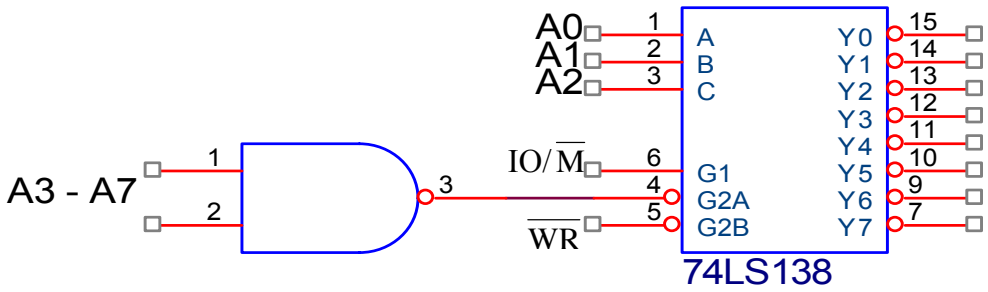
2.2. Giải mã địa chỉ cho thiết bị vào / ra

Việc giải mã địa chỉ cho thiết bị ngoại vi cũng tương tự với việc giải mã địa chỉ cho bộ nhớ. Thông thường, các cổng có địa chỉ 8 bit A0 – A7. Tuy nhiên, trong một số hệ vi xử lý, các cổng sẽ có địa chỉ 16 bit.

Ta có thể dùng mạch NAND để tạo tín hiệu chọn cổng nhưng mạch này chỉ có thể giải mã cho 1 cổng. Trong trường hợp cần nhiều tín hiệu chọn cổng, ta có thể dùng bộ giải mã 74LS138 để giải mã cho 8 cổng khác nhau.



(a) Giải mã cho cổng vào



(b) Giải mã cho cổng ra

Hình 4.4 – Giải mã cho các cổng

2.3. Các mạch cổng đơn giản

Các mạch cổng có thể được xây dựng từ các mạch chốt 8 bit (74LS373: kích theo mức, 74LS374: kích theo cạnh), các mạch đệm 8 bit (74LS245). Chúng được dùng trong các giao tiếp đơn giản để µP và ngoại vi hoạt động tương thích với nhau.

2.4. Giao tiếp vào/ra song song lập trình được 8255A PPI (Programmable Peripheral Interface)

2.4.1. Giới thiệu

8255A là thiết bị xuất nhập song song lập trình được. Nó là một thiết bị I/O đa dụng có thể sử dụng với bất cứ µP nào, có thể lập trình để truyền dữ liệu, từ I/O thông thường đến I/O interrupt.

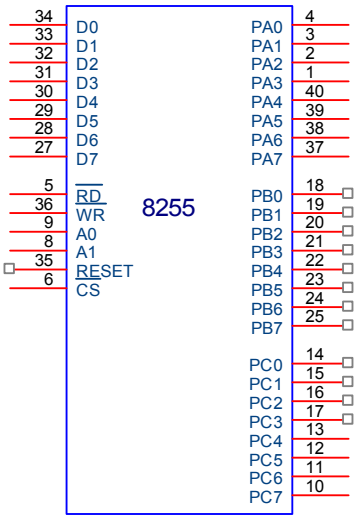
8255A có thể chia thành 3 Port: A, B và C; mỗi port 8 bit trong đó Port C có thể sử dụng như 8 bit riêng hay chia thành 2 nhóm, mỗi nhóm 4 bit: PCH (PC7 ÷ PC4) và PCL (PC3 ÷ PC0).

8255A có thể hoạt động ở 2 chế độ (mode): BSR (Bit Set/Reset) và I/O.

❖ **Chế độ BSR**: dùng để đặt hay xóa các bit của Port C.

❖ **Chế độ I/O**: gồm có 3 chế độ:

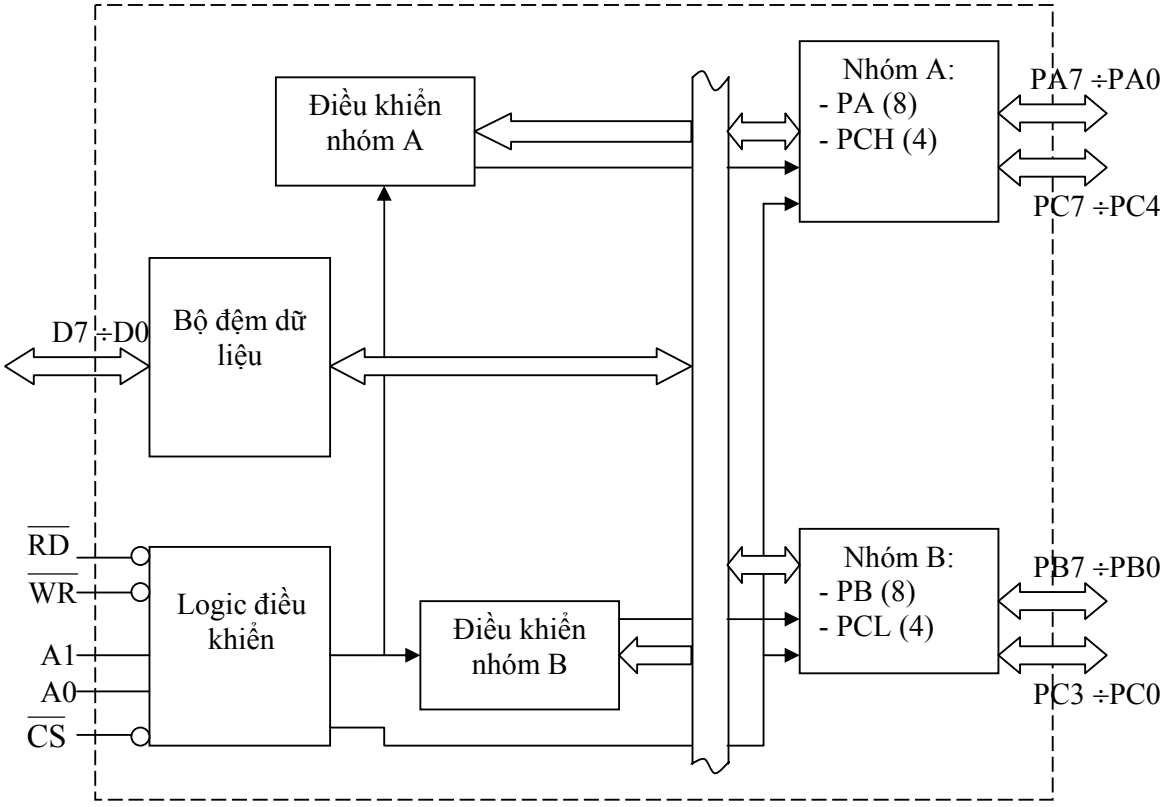
- Chế độ 0: tất cả các Port làm việc như các Port I/O đơn giản.
- Chế độ 1 (chế độ bắt tay: handshake): các Port A và B dùng các bit của Port C làm tín hiệu bắt tay. Trong chế độ này, các kiểu truyền dữ liệu I/O có thể được cài đặt, kiểm tra trạng thái và ngắt.
- Chế độ 2: Port A có thể dùng để truyền dữ liệu song hướng dùng các tín hiệu bắt tay từ Port C còn Port B được thiết lập ở chế độ 0 hay 1.



D7 – D0: bus dữ liệu
PA7 – PA0: Port A
PB7 – PB0: Port B
PC7 – PC0: Port C
A1, A0: giải mã
RESET: ngõ vào Reset
$\overline{CS}$: Chip Select
$\overline{RD}$: Read
$\overline{WR}$: Write
VCC: +5V
GND: 0V

Hình 4.5 – Sơ đồ chân của 8255A

2.4.2. Sơ đồ khối



Hình 4.6 – Sơ đồ khối của 8255A

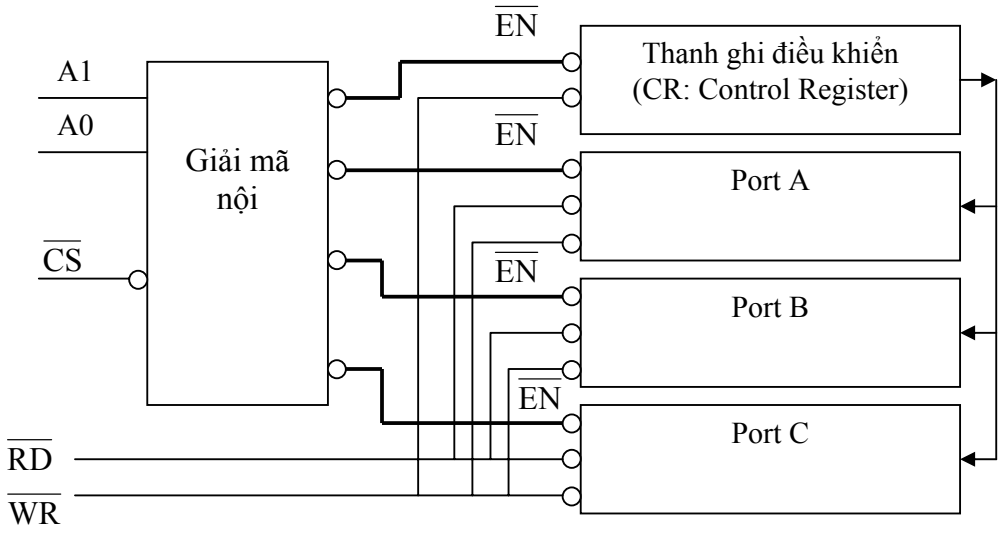
Logic điều khiển của 8255A gồm có 6 đường:

- $\overline{RD}$ (Read): cho phép ĐỌC. Khi chân này ở mức THẤP thì cho phép đọc dữ liệu từ Port I/O đã chọn.

- $\overline{WR}$ (Write): cho phép GHI. Khi chân này ở mức THẤP thì cho phép ghi dữ liệu ra Port I/O đã chọn.
- RESET: khi chân này ở mức cao thì sẽ xoá thanh ghi điều khiển và đặt các Port ở chế độ nhập.
- $\overline{CS}$ (Chip Select): chân chọn chip, thông thường $\overline{CS}$ được nối vào địa chỉ giải mã.
- A1, A0: giải mã xác định Port

Bảng 4.2:

$\overline{CS}$	A1	A0	Chọn
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Thanh ghi điều khiển
1	x	x	8255A không hoạt động

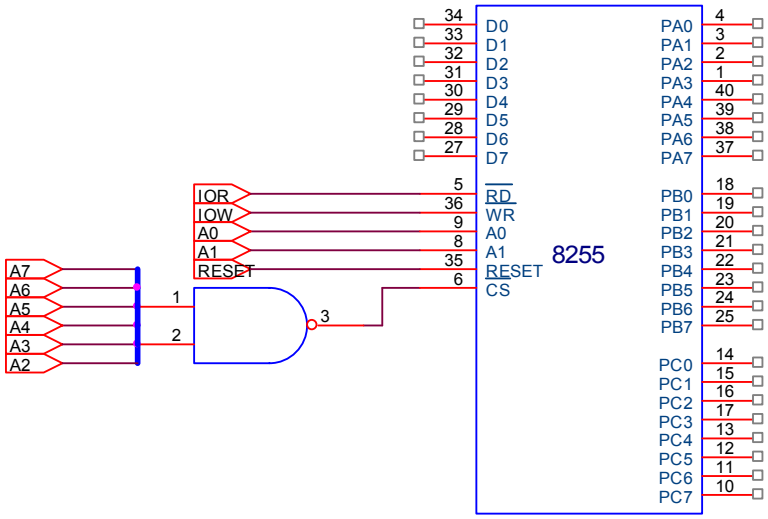


Hình 4.7 – Giải mã chọn các Port

Ví dụ: Xét sơ đồ kết nối 8255A như hình vẽ trang bên:

Theo bảng 4.2, để chọn Port A, ta phải có:

$$\begin{cases} \overline{CS} = 0 \\ A1 = 0 \\ A0 = 0 \end{cases}$$



Hình 4.8 – Logic chọn chip 8255A

Mà $\overline{CS}$ = 0 khi A7 = A6 = A5 = A4 = A3 = A2 = 1. Từ đó ta được địa chỉ Port I/O như sau:

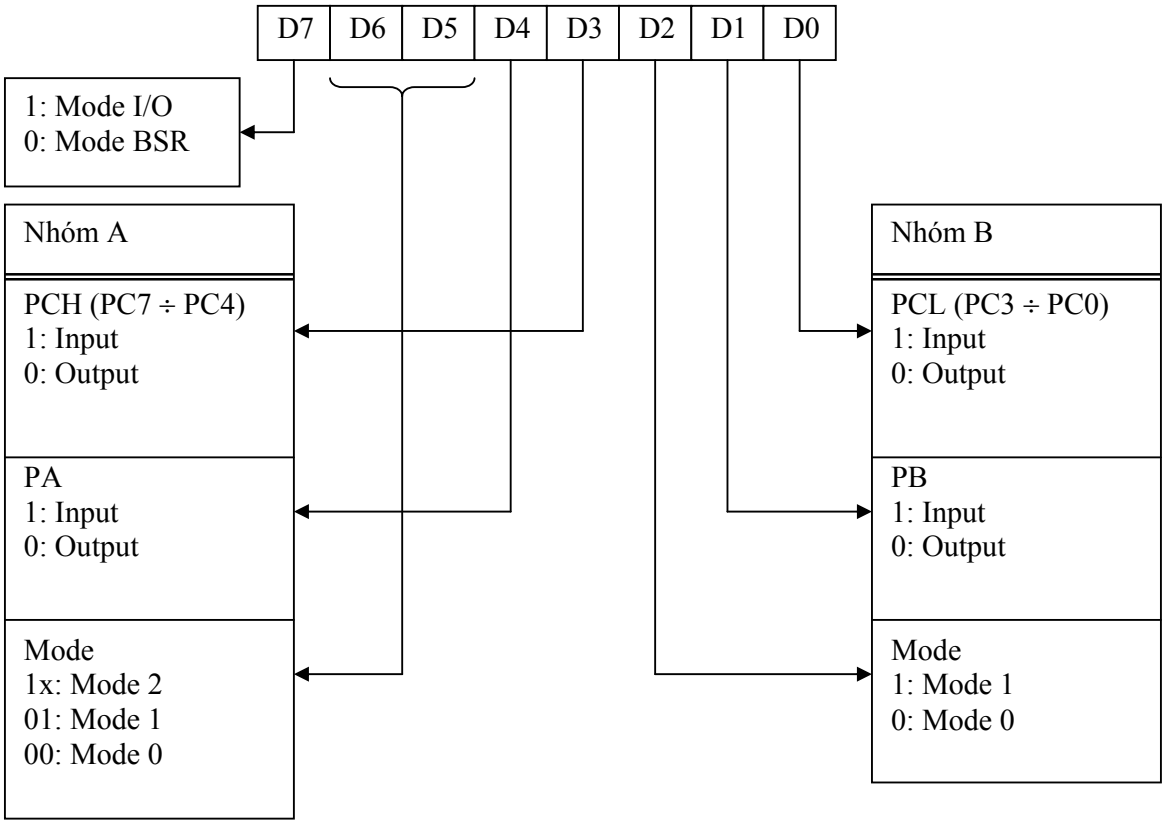
Bảng 4.3:

$\overline{CS}$						A1	A0	Port	Địa chỉ hex
A7	A6	A5	A4	A3	A2	A1	A0		
1	1	1	1	1	1	0	0	A	FCh
						0	1	B	FDh
						1	0	C	FEh
						1	1	CR	FFh

❖ **Thanh ghi điều khiển:**

Như đã biết, 8255A có 2 chế độ hoạt động và các Port của nó có thể có các chức năng I/O khác nhau. Để xác định chức năng của các Port, 8255A có một thanh ghi điều khiển (CR: Control Register). Nội dung của thanh ghi này gọi là từ điều khiển (CW: Control Word). Thanh ghi điều khiển sẽ được truy xuất khi A1 = A0 = 1. Chú ý rằng ta không thể thực hiện tác vụ Đọc đối với thanh ghi này.

Nếu bit D7 = 0, Port C làm việc ở chế độ BSR nhưng từ điều khiển BSR không ảnh hưởng đến chức năng các Port A, B.



Hình 4.9 – Dạng từ điều khiển cho 8255A ở chế độ I/O

2.4.3. Mode 0: Xuất/nhập đơn giản

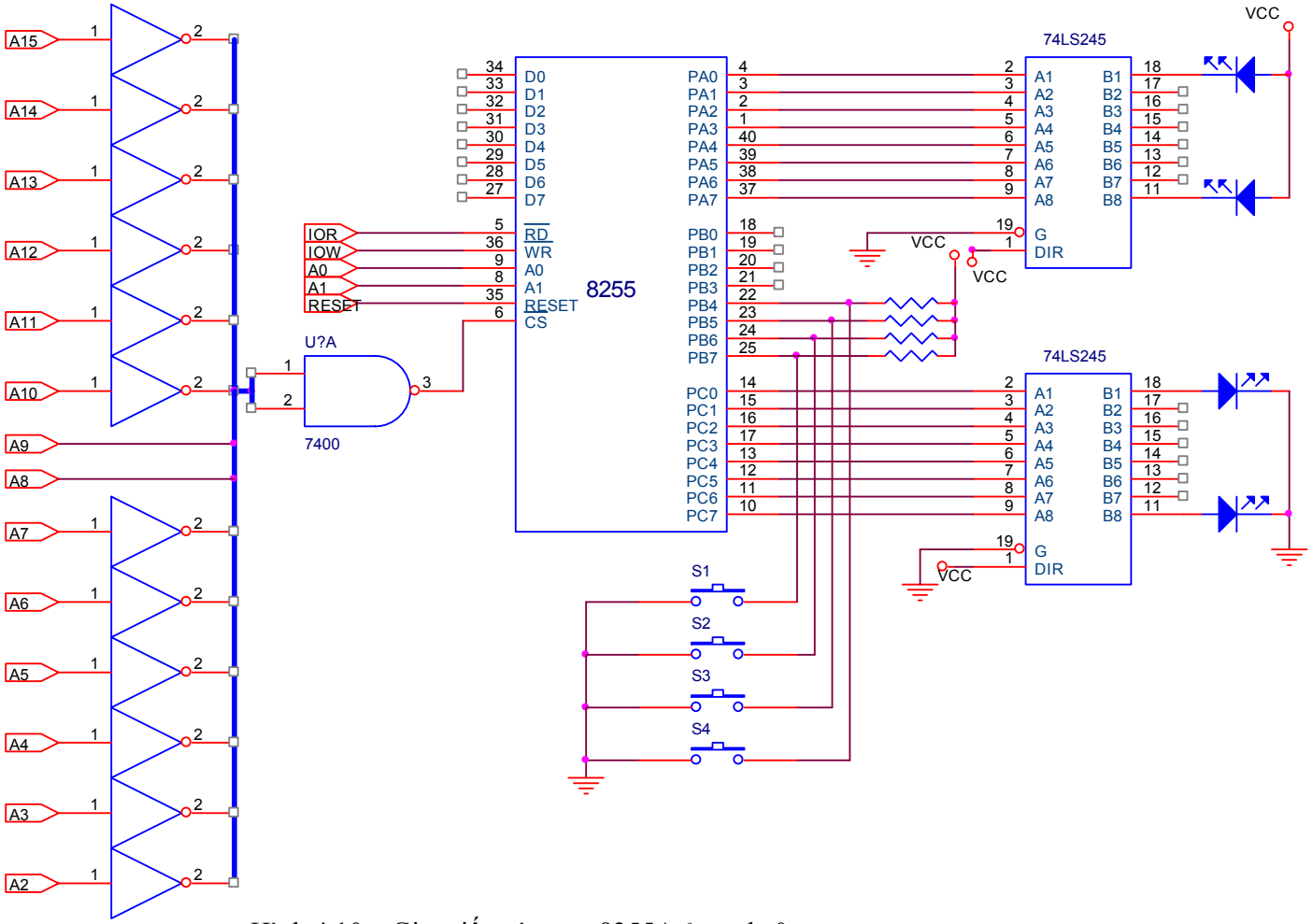
Trong chế độ này, mỗi port (hay nửa port của Port C) làm việc như các port nhập hay xuất với các tính chất sau:

- Các ngõ ra được chốt.
- Các ngõ vào không được chốt.
- Các port không có khả năng bắt tay và ngắt.

Để giao tiếp với ngoại vi thông qua 8255A cần phải:

- ***Xác định địa chỉ của các port A, B, C và CR thông qua các chân chọn chip $\overline{CS}$ và giải mã A1, A0.***
- ***Ghi từ điều khiển vào thanh ghi điều khiển.***
- ***Ghi các lệnh I/O để giao tiếp với ngoại vi qua các port A, B, C.***

Ví dụ: Xét sơ đồ kết nối 8255A như sau:



Hình 4.10 – Giao tiếp các port 8255A ở mode 0

- Xác định địa chỉ port:

Bảng 4.4:

$\overline{CS}$														A1	A0	Port	Địa chỉ hex
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	A	300h
														0	1	B	301h
														1	0	C	302h
														1	1	CR	303h

- Từ điều khiển:

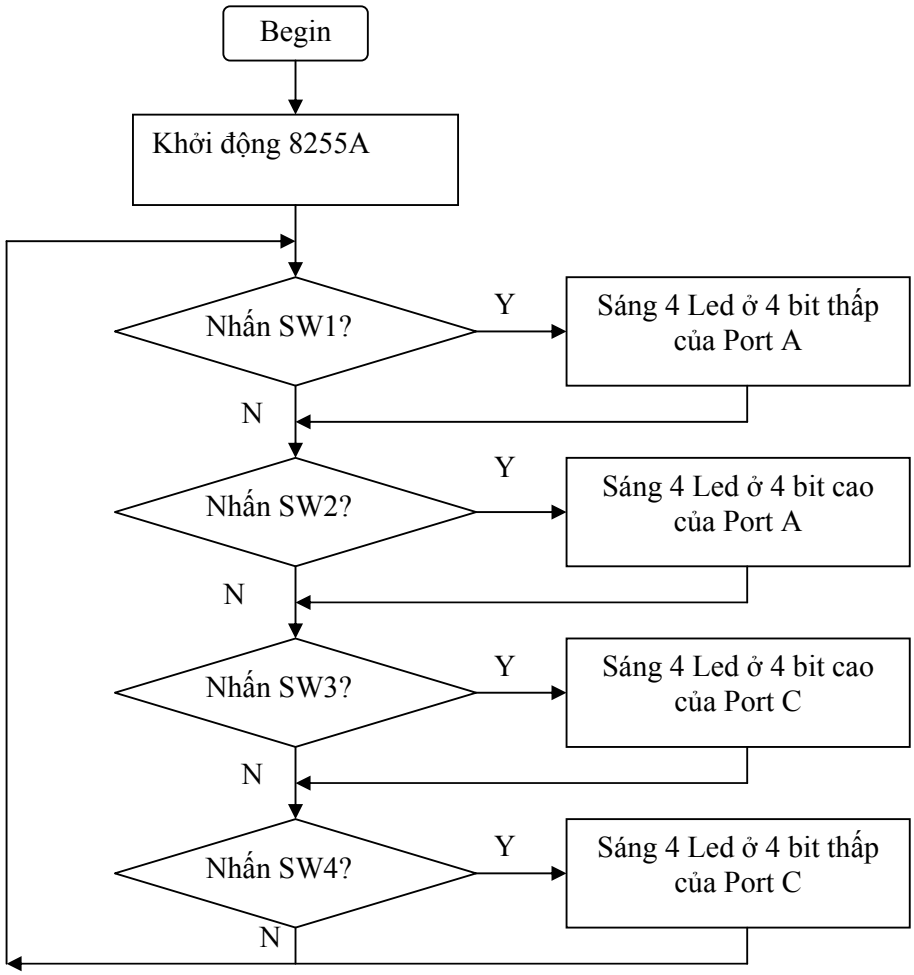
Bảng 4.5:

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	0	0	0	1	0	= 82h
I/O mode	Nhóm A ở mode 0		PA: Output	PCH: Output	Nhóm B ở mode 0	PB: Input	PCL: Output	

Các Port của 8255A được khởi động bằng cách đặt từ điều khiển 82h vào thanh ghi điều khiển.

Trong sơ đồ kết nối này, 4 bit cao của Port B dùng làm Port nhập còn Port A và Port C làm Port xuất. Các tác vụ Đọc và Ghi được phân biệt bằng các tín hiệu điều khiển $\overline{IOR}$ và $\overline{IOW}$.

- Lưu đồ giải thuật:



- Chương trình:

```
        .MODEL      SMALL
        .STACK      100h
        .CODE
main    PROC
        ; Định cấu hình cho 8255
        MOV   AL,82h          ; Từ điều khiển (CW) là 82h
        MOV   DX,303h         ; Địa chỉ thanh ghi điều khiển (CR)
        OUT   DX,AL           ; Ghi CW vào CR
cont:   MOV   DX,301h         ; Địa chỉ Port B
        IN    AL,DX           ; Đọc dữ liệu từ Port B (công tắc)
        AND   AL,0F0h         ; Che 4 bit thấp
        MOV   AH,AL
        CMP   AH,01110000b    ; Kiểm tra công tắc 1
        JNE   notSW1          ; Nếu không nhấn
        MOV   AL,0Fh          ; Nếu nhấn công tắc 1 thì
        MOV   DX,300h         ; xuất ra Port A
        OUT   DX,AL           ; để sáng 4 Led ở 4 bit thấp (Port A)

notSW1: CMP   AH,10110000b    ; Kiểm tra công tắc 2
        JNE   notSW2          ; Nếu không nhấn
```


```
                MOV   AL,0F0h          ; Nếu nhấn công tắc 2 thì
                MOV   DX,300h          ; xuất ra Port A
                OUT   DX,AL            ; để sáng 4 Led ở 4 bit cao (Port A)
notSW2:         CMP   AH,11010000b     ; Kiểm tra công tắc 3
                JNE   notSW3           ; Nếu không nhấn
                MOV   AL,0Fh           ; Nếu nhấn công tắc 3 thì
                MOV   DX,302h          ; xuất ra Port C
                OUT   DX,AL            ; để sáng 4 Led ở 4 bit cao (Port C)
notSW3:         CMP   AH,11100000b     ; Kiểm tra công tắc 4
                JNE   notSW4           ; Nếu không nhấn
                MOV   AL,F0h           ; Nếu nhấn công tắc 4 thì
                MOV   DX,302h          ; xuất ra Port C
                OUT   DX,AL            ; để sáng 4 Led ở 4 bit thấp (Port C)
notSW4:         JMP   cont
        main    ENDP
        END     main
```

2.4.4. Mode BSR

Mode BSR chỉ liên quan đến 8 bit của Port C, có thể đặt hay xoá các bit bằng cách ghi một từ điều khiển thích hợp vào thanh ghi điều khiển. Một từ điều khiển với D7 = 0 gọi là từ điều khiển BSR, từ điều khiển này không làm thay đổi bất cứ từ điều khiển nào được truyền trước đó với D7 = 1, nghĩa là các hoạt động I/O của Port A và B không bị ảnh hưởng bởi từ điều khiển BSR.

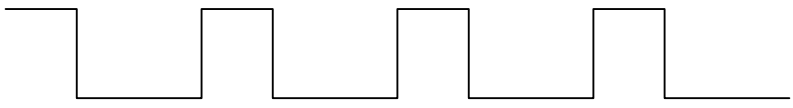
❖ **Từ điều khiển BSR:**

Từ điều khiển BSR khi được ghi vào thanh ghi điều khiển sẽ đặt hay xoá mỗi lần 1 bit.

D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	X				S/R
Mode BSR	Không sử dụng			Chọn bit 000: PC0 001: PC1 010: PC2 011: PC3 100: PC4 101: PC5 110: PC6 111: PC7			0: Xoá (Reset) 1: Đặt (Set)

Ví dụ: Xét sơ đồ kết nối 8255A như hình 4.10. Giả sử ta cần tạo một sóng chữ nhật tại bit PC0.

Để tạo một sóng chữ nhật tại PC0, ta cần 2 mức logic là 0 và 1 tại PC0.



	D7	D6	D5	D4	D3	D2	D1	D0	
Đặt bit PC0 = 1	0	0	0	0	0	0	0	1	= 01h
Xoá bit PC0 = 0	0	0	0	0	0	0	0	0	= 00h

- Địa chỉ thanh ghi điều khiển (bảng 4.4): 303h

- Chương trình con:

```
bsr:    MOV     AL,01h      ; Từ điều khiển BSR
        MOV     DX,303h     ; Địa chỉ thanh ghi điều khiển (CR)
        OUT     DX,AL       ; Đặt PC0 = 1
        CALL    DELAY1      ; Chờ
        MOV     AL,00h      ; Từ điều khiển BSR
        OUT     DX,AL       ; Xóa PC0 = 0
        CALL    DELAY2      ; Chờ
        JMP     bsr
```

Khi sử dụng ở mode BSR, cần chú ý các điều sau:

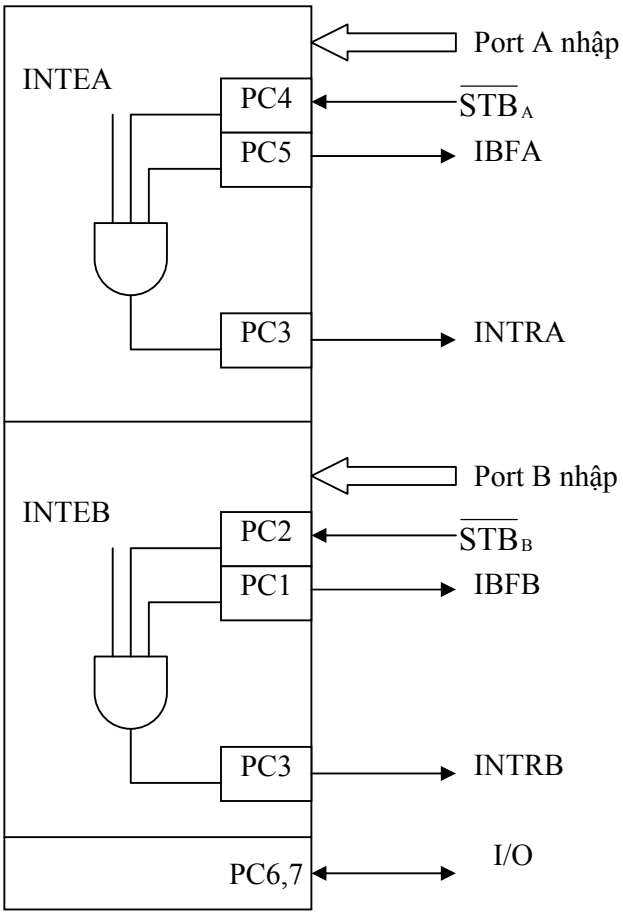
- ***Để đặt hay xoá các bit ở Port C, từ điều khiển được ghi vào thanh ghi điều khiển chứ không ghi vào Port C.***
- ***Một từ điều khiển BSR chỉ ảnh hưởng đến một bit của Port C.***
- ***Từ điều khiển BSR không ảnh hưởng đến I/O mode.***

2.4.5. Mode 1: Nhập / xuất với bắt tay (handshake)

Trong mode 1, các tín hiệu bắt tay được trao đổi giữa µP và thiết bị ngoại vi trước khi truyền dữ liệu. Các đặc tính ở chế độ này là:

- Hai Port A, B làm việc như các Port I/O 8 bit.
- Mỗi Port sử dụng 3 đường từ Port C làm các tín hiệu bắt tay. Hai đường còn lại có thể dùng cho các chức năng I/O đơn giản.
- Dữ liệu nhập / xuất được chốt.
- Hỗ trợ ngắt.

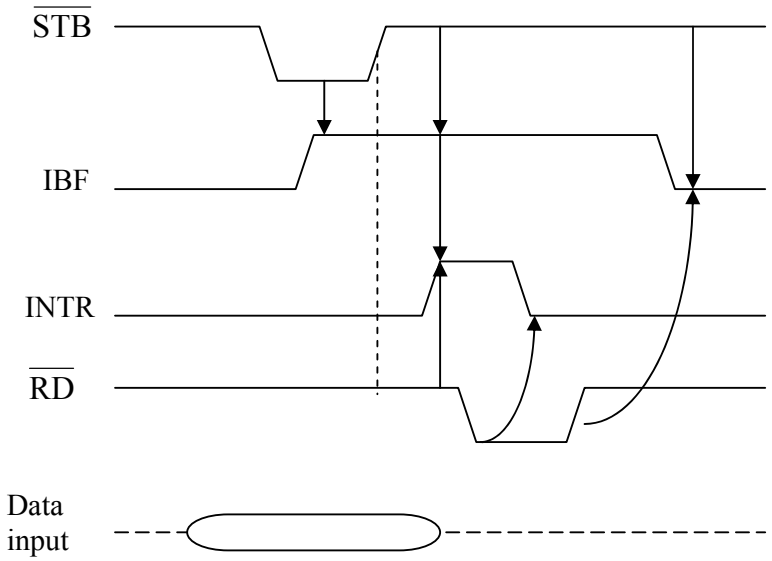
2.4.5.1. Các tín hiệu điều khiển nhập



Hình 4.11 – Cấu hình nhập của 8255A ở mode 1

Theo hình vẽ, ta thấy Port A dùng 3 đường tín hiệu trên PC3, PC4 và PC5; Port B dùng 3 đường tín hiệu trên PC0, PC1 và PC2 làm các tín hiệu bắt tay. Các tín hiệu này có các chức năng sau khi các port A và B được đặt cấu hình là nhập:

- $\overline{STB}$ (Strobe Input): tích cực mức thấp, tín hiệu này được tạo bởi thiết bị ngoại vi để xác định rằng ngoại vi đã truyền 1 byte dữ liệu. Khi 8255A đáp ứng $\overline{STB}$, nó sẽ tạo ra IBF và INTR (hình 4.12).
- IBF (Input Buffer Full): tín hiệu này dùng để xác nhận 8255A đã nhận byte dữ liệu. Nó sẽ bị xoá khi µP đọc dữ liệu.
- INTR (Interrupt Request): Đây là tín hiệu xuất dùng để ngắt µP. Nó được tạo ra nếu $\overline{STB}$, IBF và INTE (flipflop bên trong) đều ở mức logic 1 và bị xoá bởi cạnh xuống của tín hiệu $\overline{RD}$ (Hình 4.12).
- INTE (Interrupt Enable): là một flipflop dùng để cho phép hay cấm quá trình tạo ra tín hiệu INTR. Hai flipflop INTEA và INTEB được đặt / xoá dùng BSR mode thông qua PC4 và PC2.



Hình 4.12 – Dạng sóng định thì cho ngõ vào có strobe

❖ **Các từ điều khiển và trạng thái:**

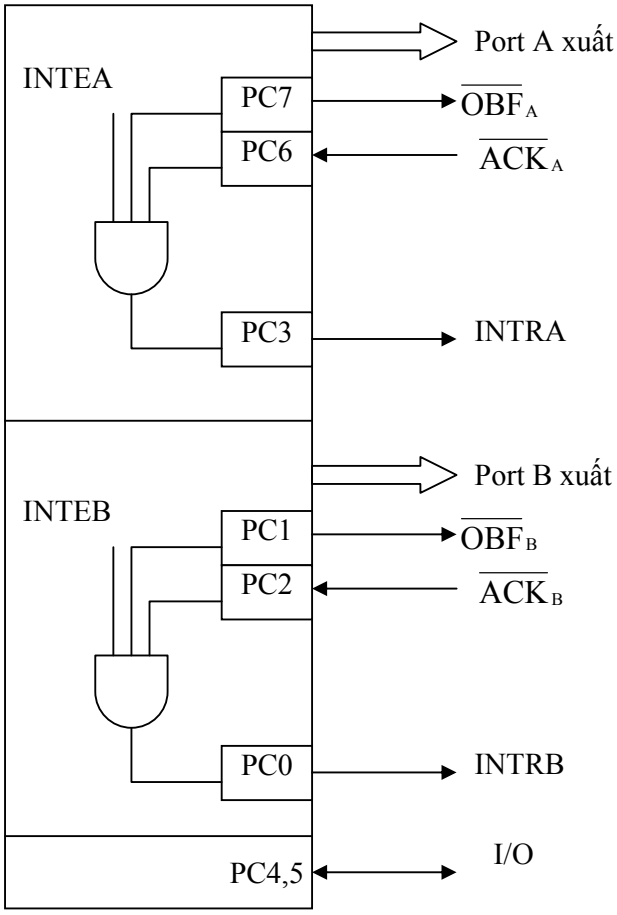
- Từ điều khiển: để xác định từ điều khiển, ta sử dụng hình 3.4.5

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1/0	1	1	X
I/O mode	PA: Mode 1		PA: nhập	PC6,7 1: nhập 0: xuất	PB: Mode 1	PB: nhập	

- Từ trạng thái: sẽ được đặt trong thanh ghi tích luỹ nếu đọc Port C.

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB

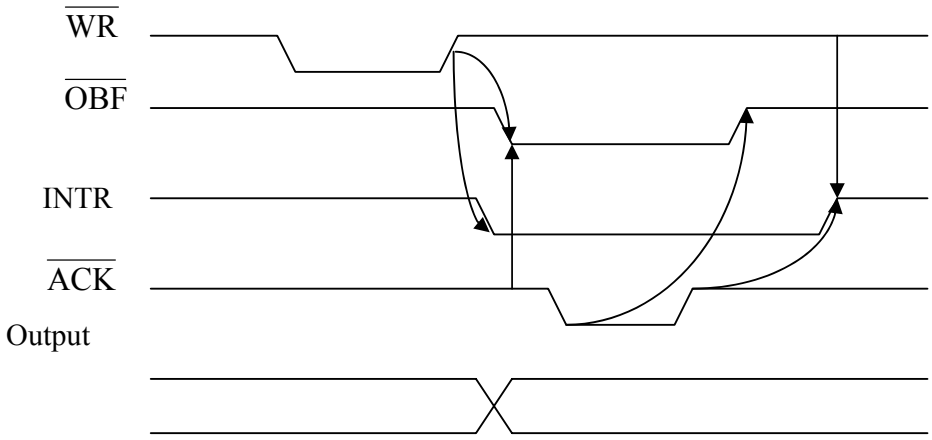
2.4.5.2. Các tín hiệu điều khiển xuất



Hình 4.13 – Cấu hình xuất của 8255A ở mode 1

Chức năng các đường tín hiệu :

- $\overline{OBF}$ (Output Buffer Full): tín hiệu này sẽ xuống mức thấp khi µP ghi dữ liệu vào Port xuất của 8225A. Tín hiệu này đưa đến thiết bị ngoại vi để xác định dữ liệu sẵn sàng đưa vào ngoại vi (Hình 4.14). Nó sẽ lên mức cao khi 8255A nhận $\overline{ACK}$ từ ngoại vi.
- $\overline{ACK}$ (Acknowledge): đây là tín hiệu nhập từ ngoại vi (tích cực mức thấp) xác nhận dữ liệu đã nhập vào ngoại vi.
- INTR (Interrupt Request): đây là tín hiệu xuất, đặt bằng cạnh lên của tín hiệu $\overline{ACK}$. Tín hiệu này có thể dùng để ngắt µP yêu cầu byte dữ liệu kế tiếp để xuất. INTR được đặt khi $\overline{OBF}$, $\overline{ACK}$ và INTE ở mức logic 1 (Hình 4.14) và được xoá bởi cạnh xuống của tín hiệu $\overline{WR}$
- INTE (Interrupt Enable): đây là flipflop nội dùng để tạo tín hiệu INTR. Hai flipflop INTEA và INTEB điều khiển bằng các bit PC6 và PC2 thông qua BSR mode.



Hình 4.14 – Dạng sóng cho xuất strobe (có lấy mẫu) (với bắt tay)

❖ **Từ điều khiển và trạng thái:**

- Từ điều khiển:

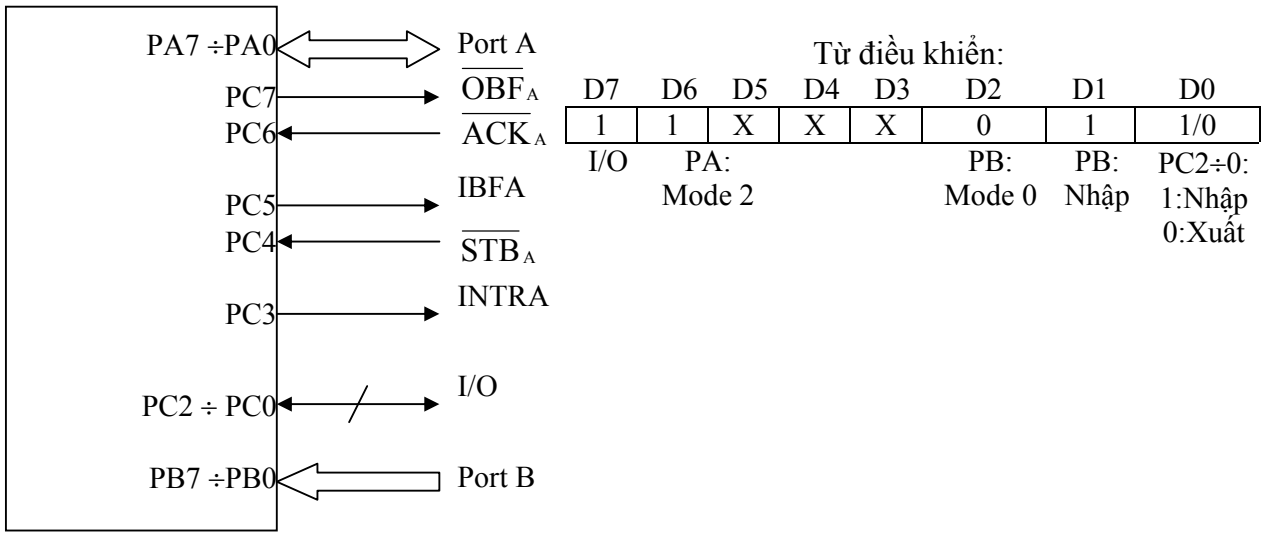
D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	1/0	1	0	X
I/O mode	PA: Mode 1		PA: xuất	PC4,5 1: nhập 0: xuất	PB: mode 1	PB: xuất	

- Từ trạng thái:

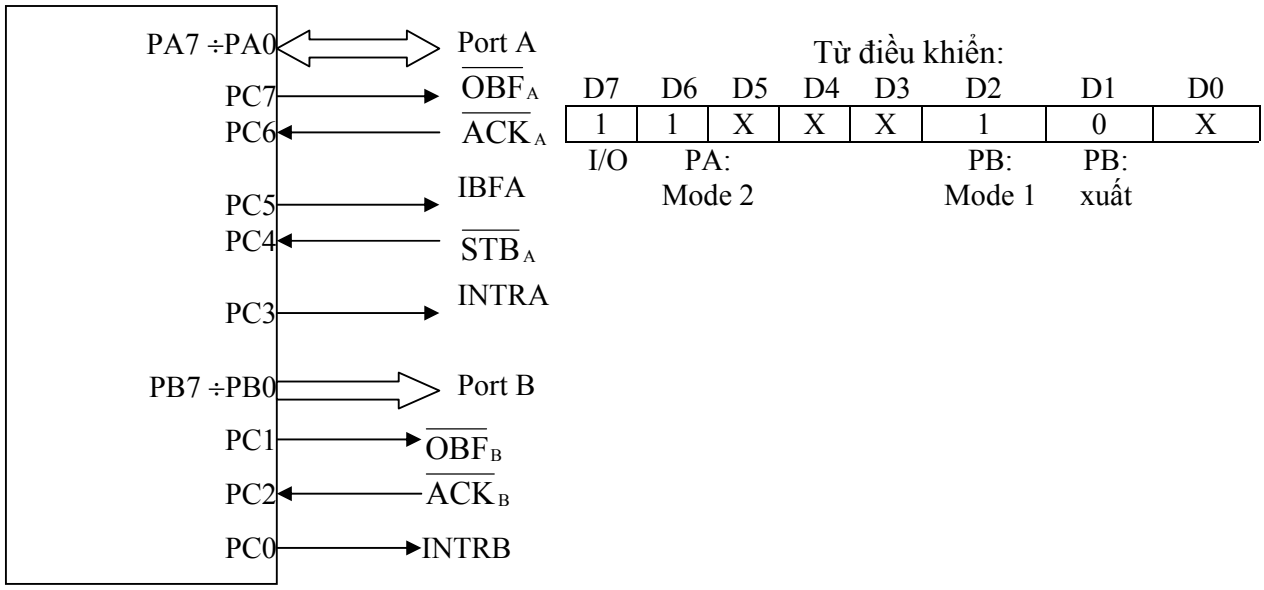
D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBF}_A$	INTEA	I/O	I/O	INTRA	INTEB	$\overline{OBF}_B$	INTRB

2.4.6. Mode 2: Truyền dữ liệu song hướng

Mode nay dùng chủ yếu trong các ứng dụng như truyền dữ liệu giữa hai máy tính hay giao tiếp bộ điều khiển đĩa mềm. Trong mode này, Port A dùng làm Port song hướng và Port B làm việc ở Mode 0 hay 1. Port A sử dụng 5 tín hiệu tại Port C làm các tín hiệu điều khiển để truyền dữ liệu. Ba tín hiệu còn lại của Port C được dùng làm I/O đơn giản hay bắt tay cho Port B.



(a) 8255A ở mode 2 và mode 0 (nhập)



(a) 8255A ở mode 2 và mode 1 (xuất)

Hình 4.15 – 8255A dùng ở Mode 2

2.4.7. Các ví dụ minh họa

2.4.7.1. Giao tiếp với bộ chuyển đổi A/D ADC0804 dùng 8255A ở Mode 0 và Mode BSR

Ta thiết lập 8255A hoạt động như sau:

- Dùng Port A để đọc dữ liệu.
- Dùng PC0, PC3 điều khiển các chân $\overline{RD}$, $\overline{WR}$ của ADC0804.

Xét sơ đồ mạch có logic chọn chip giống như hình 4.10. Tầm địa chỉ Port từ 300h ÷ 303h.

- **Từ điều khiển mode 0:**

Port A: nhập

Pot B: không sử dụng

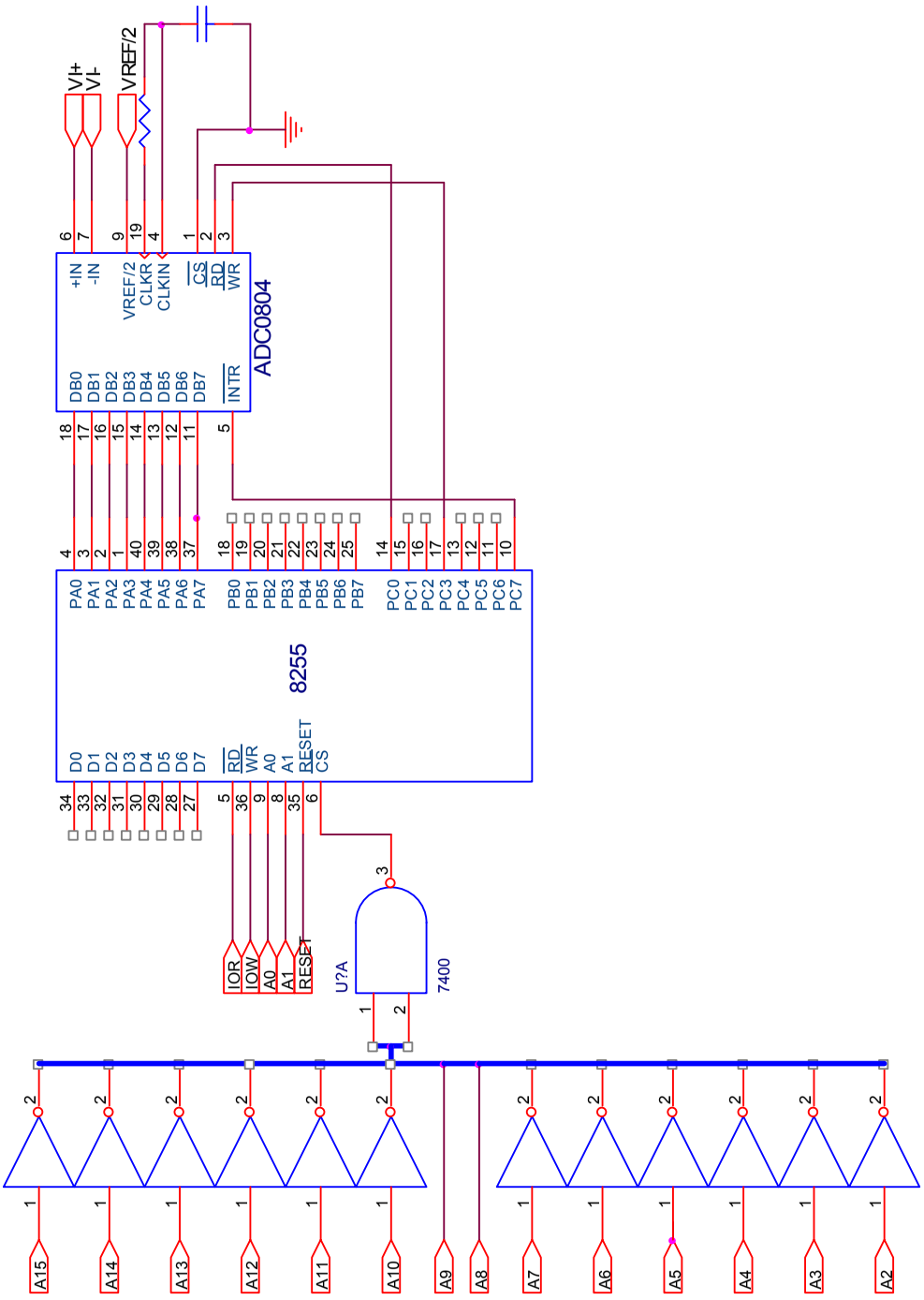
Port Clow: port xuất dùng để điều khiển 2 ngõ $\overline{RD}$, $\overline{WR}$ của ADC0804

Port Chigh: port nhập dùng để đọc trạng thái ở chân $\overline{INTR}$ của ADC0804

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	1	0	0	0	0	= 90h
I/O	PA: mode 0		PA: nhập	PCH: xuất	PB: không sử dụng		PCL: xuất	

- **Từ điều khiển BSR:**

	D7	D6	D5	D4	D3	D2	D1	D0	
Đặt PC0	0	0	0	0	0	0	0	1	= 01h
Xoá PC0	0	0	0	0	0	0	0	0	= 00h
Đặt PC3	0	0	0	0	0	1	1	1	= 07h
Xoá PC3	0	0	0	0	0	1	1	0	= 06h



Hình 4.16 – Giao tiếp bộ chuyển đổi A/D ADC0804 dùng 8255A

- Mô tả chương trình:

  - Khởi động 8255A bằng cách đặt từ điều khiển mode 0 vào thanh ghi điều khiển.
  - Cấp một xung vào chân $\overline{RD}$ của 8255A.
  - Đọc trạng thái của ADC0804 từ chân $\overline{INTR}$.
  - Nếu $\overline{INTR}$ = 0 thì cấp một xung vào chân $\overline{WR}$ của ADC0804 để xuất dữ liệu.
  - Đọc dữ liệu từ ADC0804 vào thông qua Port A.

- Đoạn chương trình thực hiện:

```
adc:  MOV   DX,303h   ; Địa chỉ thanh ghi điều khiển (CR)
      MOV   AL,90h    ; Từ điều khiển (CW)
      OUT   DX,AL     ;Ghi CW vào CR
      MOV   AL,01h    ; Từ điều khiển BSR để PC0 = 1 (RD = 1)
      OUT   DX,AL     ; Xuất ra CR
      MOV   AL,07h    ; Từ điều khiển BSR để PC3 = 1
      OUT   DX,AL     ; Xuất ra CR
      MOV   AL,06h    ; Từ điều khiển BSR để PC3 = 0, tạo xung WR
      OUT   DX,AL     ; Xuất ra CR
      CALL  DELAY     ; Chờ quá trình chuyển đổi thực hiện xong
      MOV   AL,07h    ; Từ điều khiển BSR để PC3 = 1
      OUT   DX,AL     ; Xuất ra CR
      MOV   DX,300h   ; Địa chỉ Port A
      IN    AL,DX     ; Đọc dữ liệu đã chuyển đổi từ ADC0804
      MOV   AL,01h    ; Từ điều khiển BSR để PC0 = 1 (RD = 1)
      OUT   DX,AL     ; Xuất ra CR
      RET             ; vào từ Port A của 8255A
```

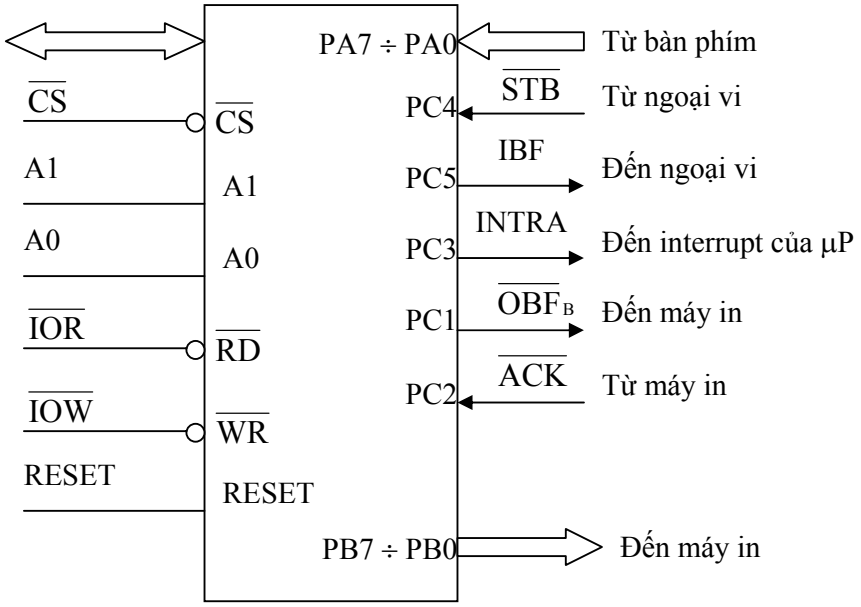
2.4.7.2. Giao tiếp với máy in trong chế độ bắt tay (Mode 1)

Xét mạch giao tiếp 8255A ở mode 1 với Port A được dùng làm Port nhập từ bàn phím với I/O interrupt và Port B được thiết kế làm Port xuất tới máy in với I/O kiểm tra trạng thái. Ta cần thực hiện các công việc sau:

- Xác định địa chỉ Port.
- Xác định từ điều khiển để Port A nhập và Port B xuất ở Mode 1.
- Xác định từ điều khiển BSR cho phép ngắt (INTEA).
- Xác định các byte mặt nạ để kiểm tra các đường $\overline{OBF}_B$ trong I/O kiểm tra trạng thái.
- Viết các lệnh khởi động và chương trình con in các ký tự chứa trong bộ nhớ.

Giả sử logic chọn chip như hình 4.10, địa chỉ Port cho trong bảng 4.4:

PA: FCh
PB: FDh
PC: FEh
CR: FFh



Hình 4.17 – Giao tiếp 8255A ở Mode 1

- **Từ điều khiển:** Port A nhập, Port B xuất ở Mode 1

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	1	1	0	1	0	0	= B4h
I/O	PA: Mode 1		PA: nhập	Không sử dụng	PB: Mode 1	PB: xuất	Không sử dụng	

- **Từ điều khiển BSR**: dùng để đặt flipflop cho phép ngắt của Port A (INTEA), bit PC4 = 1

D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	1	0	0	1	= 09h
BSR mode	Không sử dụng			Bit PC4		Đặt bit (Set)		

- **Từ trạng thái kiểm tra $\overline{OBF}_B$:**

D7	D6	D5	D4	D3	D2	D1	D0
X	x	x	x	x	x	$\overline{OBF}_B$	X

Byte mặt nạ: 0000 0010b

❖ **Khởi động:**

```
MOV     DX, 0FFh    ; Khởi động 8255A
MOV     AL, 0B4h    ; ở Mode 1, Port A nhập
OUT     DX, AL      ; Port B xuất
MOV     AL, 09h     ; Đặt INTEA
OUT     DX, AL      ; cho phép INTRA
CALL    print
```

❖ **Chương trình con PRINT:**

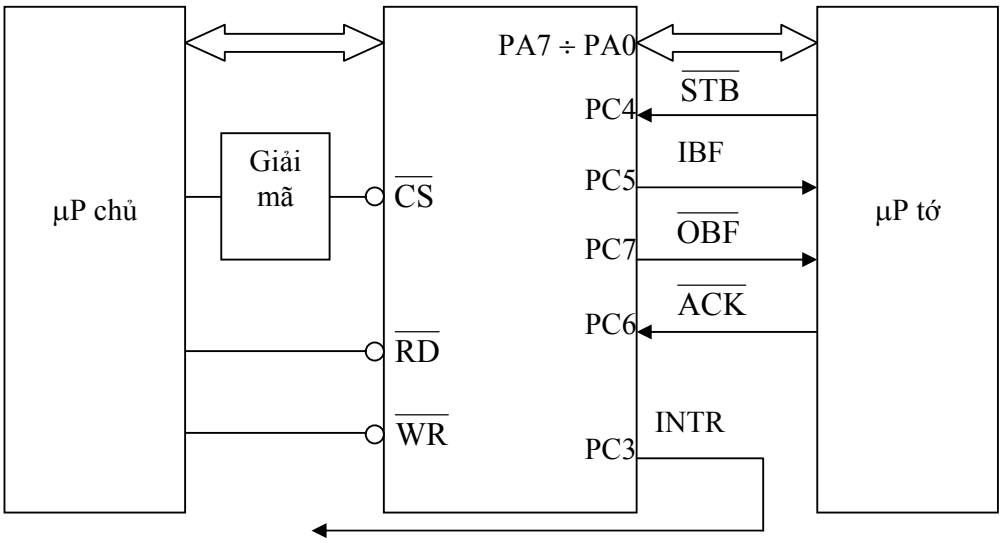
```
print:    LEA     DX,msg      ; Chỉ đến vị trí chứa các ký tự
          MOV     SI, DX
          ADD     SI,2
next:     LODSB                       ; Lấy ký tự từ bộ nhớ
          CMP     AL,0                ; Nếu không còn ký tự nào
          JNE     cont                ; thì kết thúc
          JMP     exit
cont:     MOV     AH,AL               ; Lưu ký tự vừa đọc
          MOV     DX,0FEh
status:   IN      AL,DX               ; Đọc vào từ Port C
          AND     AL,02h              ; Chỉ nhận PC1
          JE      status              ; Nếu máy in không sẵn sàng thì chờ
          MOV     AL,AH
          MOV     DX,0FDh             ; Xuất ký tự đã nhận ra
          OUT     DX,AL               ; máy in (Port B)
          JMP     next        ; Xử lý ký tự kế tiếp
exit:     RET
```

❖ **Mô tả chương trình:**

- Ta sử dụng 8255A trong phần thiết kế này cho phép 2 hoạt động: xuất ra máy in và lấy dữ liệu vào từ bàn phím. Giao tiếp với máy in dùng kiểm tra trạng thái và giao tiếp bàn phím dùng ngắt.
- Trong chương trình con PRINT, ký tự được đặt trong thanh ghi tích luỹ A và trạng thái đọc từ Port C. Ban đầu Port B trống, bit PC1 ($\overline{OBF}_B$) ở mức cao. Ta thực hiện lệnh OUT gởi dữ liệu ra Port B. Tín hiệu $\overline{OBF}_B$ sẽ xuống mức thấp do tác động cạnh lên của tín hiệu $\overline{WR}$, xác định rằng dữ liệu đã gởi ra máy in. Sau khi nhận byte dữ liệu, máy in gởi trở lại tín hiệu $\overline{ACK}$ xác định đã nhận. Tín hiệu $\overline{ACK}$ làm cho $\overline{OBF}_B$ ở mức cao xác định máy in sẵn sàng nhận ký tự kế tiếp và chương trình con PRINT tiếp tục thực hiện cho đến khi không còn ký tự nào trong vùng nhớ.
- Nếu một phím được nhấn khi chương trình con PRINT đang thực thi, byte dữ liệu truyền tới Port A và $\overline{STB}_A$ xuống mức thấp, đặt IBFA lên mức cao. Khi $\overline{STB}_A$ trở lại mức cao thì sẽ tạo ra INTRA. Tín hiệu này tạo ngắt đến µP và điều khiển được chuyển đến chương trình phục vụ ngắt. Chương trình này sẽ đọc nội dung Port A, cho phép ngắt và quay về chương trình con PRINT.

2.4.7.3. Truyền dữ liệu giữa hai microprocessor trong xử lý phân bố dùng 8255A ở Mode 2

Ta thiết kế mạch giao tiếp để truyền dữ liệu hai chiều dạng chủ – tớ (master – slave) giữa hai µP.



Hình 4.18 – Thông tin 2 chiều giữa 2 µP dùng 8255A

Hình 4.18 chỉ sơ đồ khối thiết lập thông tin hay chiều giữa chủ và tớ . Sơ đồ khối chỉ hai data bas hai chiều – chủ và tớ – được nối với nhau thông qua 8225A, trong đó 8225A làm việc như thiết bị giao tiếp của µP chủ. Port A của 8225A được dùng để truyền dữ liệu hai chiều và 4 tín hiệu từ port C được dùng để bắt tay. Quá trình truyền dữ liệu tương tự như Mode 1 của 8225A. Khi µP chủ ghi 1 byte dữ liệu vào 8225A tín hiệu $\overline{OBF}$ xuống mức thấp để báo cho µP tớ biết là đã gởi dữ liệu vào, µP tớ sẽ báo nhận được khi nó đọc byte dữ liệu này. Tương tự, hai tín hiệu bắt tay khác được dùng khi µP tớ truyền 1 byte dữ liệu đến µP chủ.

µP chủ đòi hỏi các port I/O dùng để đọc và ghi dữ liệu và kiểm tra trạng thái của các tín hiệu bắt tay. Tương tự, µP tớ cần các port I/O để thực hiện Đọc và Ghi. Truyền dữ liệu có thể được thực hiện bằng cách kiểm tra trạng thái hay dùng ngắt. Tốc độ xử lý dữ liệu đối với µP chủ quan trọng hơn nên thường dùng µP chủ ở chế độ ngắt và µP tớ ở chế độ kiểm tra trạng thái. Ở ví dụ này, ta sẽ dùng cả 2 µP ở chế độ kiểm trạng thái.

Các hoạt động truyền dữ liệu giữa 2 I/O kiểm tra trạng thái có thể liệt kê như sau:

❖ **Truyền dữ liệu từ µP chủ đến µP tớ:**

1. µP chủ đọc trạng thái của $\overline{OBF}$ để kiểm tra xem µP tớ đã đọc dữ liệu chưa. Đây là chức năng nhập cho µP chủ.
2. µp chủ ghi dữ liệu vào Port A và 8225A báo cho µP tớ biết bằng cách đưa tín hiệu $\overline{OBF}$ xuống mức thấp. Đây là chức năng xuất của µP chủ.

3. µP tớ kiểm tra tín hiệu $\overline{OBF}$ (từ µP chủ) để xác định tính sẵn sàng của dữ liệu. Đây là chức năng nhập đối với µP tớ.
4. µP tớ đọc dữ liệu từ Port A và báo cho biết đã nhận được bằng cách đưa tín hiệu $\overline{ACK}$ xuống mức thấp . Đây là chức năng nhập đối với µP tớ.

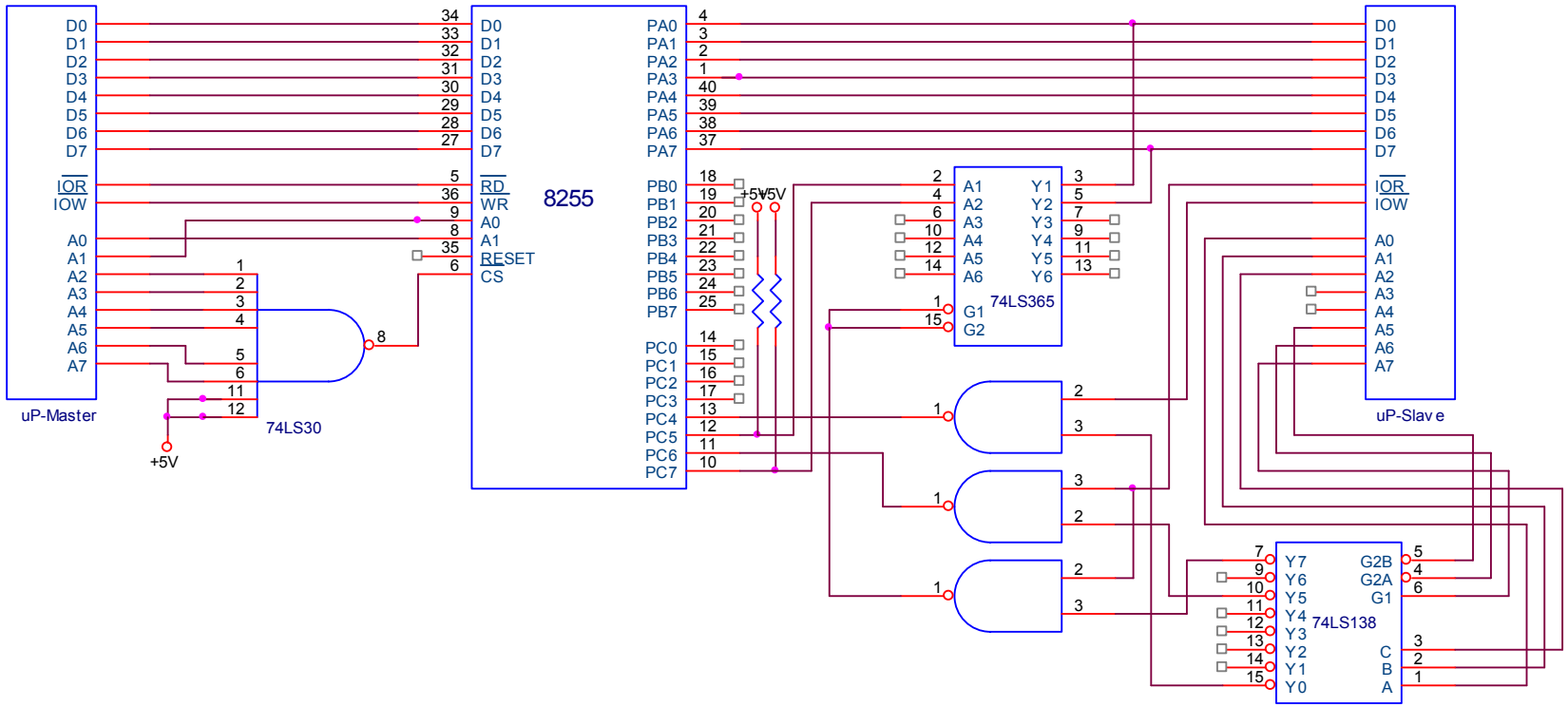
❖ **Truyền dữ liệu từ µP tớ đến µp chủ:**

1. µP tớ kiểm tra tín hiệu bắt tay IBF để xem port A có sẵn sàng truyền dữ liệu hay không để truyền 1 byte . Đây là chức năng nhập đối với µP tớ .
2. µP đặt byte dữ liệu lên data bus và báo cho 8225A biết rằng sẵn sàng gởi dữ liệu bằng cách dùng tín hiệu $\overline{STB}$. Đây là chức năng xuất đối vói µP tớ.
3. 8225A đưa IBF lên mức cao, µP chủ đọc tín hiệu này để xác định dữ liệu sẵn sàng chưa . Đây là chức năng nhập đối với µP chủ .
4. µP chủ đọc byte dữ liệu . Đây là chức năng nhập đối với µP chủ.

❖ **Kết nối phần cứng:**

Hình 4.19 cho thấy sơ đồ kết nối các port cần thiết và logic chọn chip cho 8255A. µP chủ thực hiện giải mã chọn 8255A dùng cổng NAND 8 ngõ vào nên 8255A được chọn khi tất cả các ngõ vào của cổng NAND đều ở mức 1. Từ đó, ta có các địa chỉ Port của 8255A đối với µP chủ là:

PA: FCh
PB: FDh
PC: FEh
CR: FFh



Hình 4.19 – Thông tin hai chiều giữa µP chủ và µP tớ

Port A được sử dụng ở Mode 2 dùng 4 tín hiệu từ Port C. µP chủ kiểm tra các tín hiệu $\overline{ACK}$ và $\overline{STB}$ bằng cách đọc các bit trạng thái $\overline{OBF}$ và IBF ở Port C.

Hai tín hiệu bắt tay khác - $\overline{OBF}$ và IBF – được nối tương ứng với các bit D7 và D0 của data bus của µP tớ thông qua bộ đệm 3 trạng thái 74LS365. Logic giải mã cho các đường tín hiệu tại Port C chính là bộ giải mã 3 sang 8 74LS138. Giả sử các đường logic không sử dụng (A3 và A4) ở mức 0, 8 đường ra của bộ giải mã sẽ cho phép vùng địa chỉ 80h ÷ 87h (Bảng 4.6). Hai đường ra của bộ giải mã được kết hợp với tín hiệu điều khiển $\overline{IOR}$ để tạo ra 2 xung chọn thiết bị nhận (85h và 87h). Xung chọn thiết bị nhập 87h được dùng để đọc trạng thái ở các đường dữ liệu D7 và D0. Đường giải mã có địa chỉ 80h được kết hợp với $\overline{IOW}$ để tạo tín hiệu $\overline{STB}$.

Bảng 4.6:

A7	A6	A5	A4	A3	A2	A1	A0	Chân giải mã	Địa chỉ hex
1	0	0	0	0	0	0	0	Y0	80h
					0	0	1	Y1	81h
					0	1	0	Y2	82h
					0	1	1	Y3	83h
					1	0	0	Y4	84h
					1	0	1	Y5	85h
					1	1	0	Y6	86h
					1	1	1	Y7	87h

❖ **Từ điều khiển mode 2:**

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	0	0	0	0	0	0	= C0h
I/O	Mode 2			Không sử dụng				

❖ **Từ trạng thái mode 2:**

Trạng thái của hoạt động I/O ở Mode 2 có thể kiểm rta bằng cách đọc nội dung Port C.

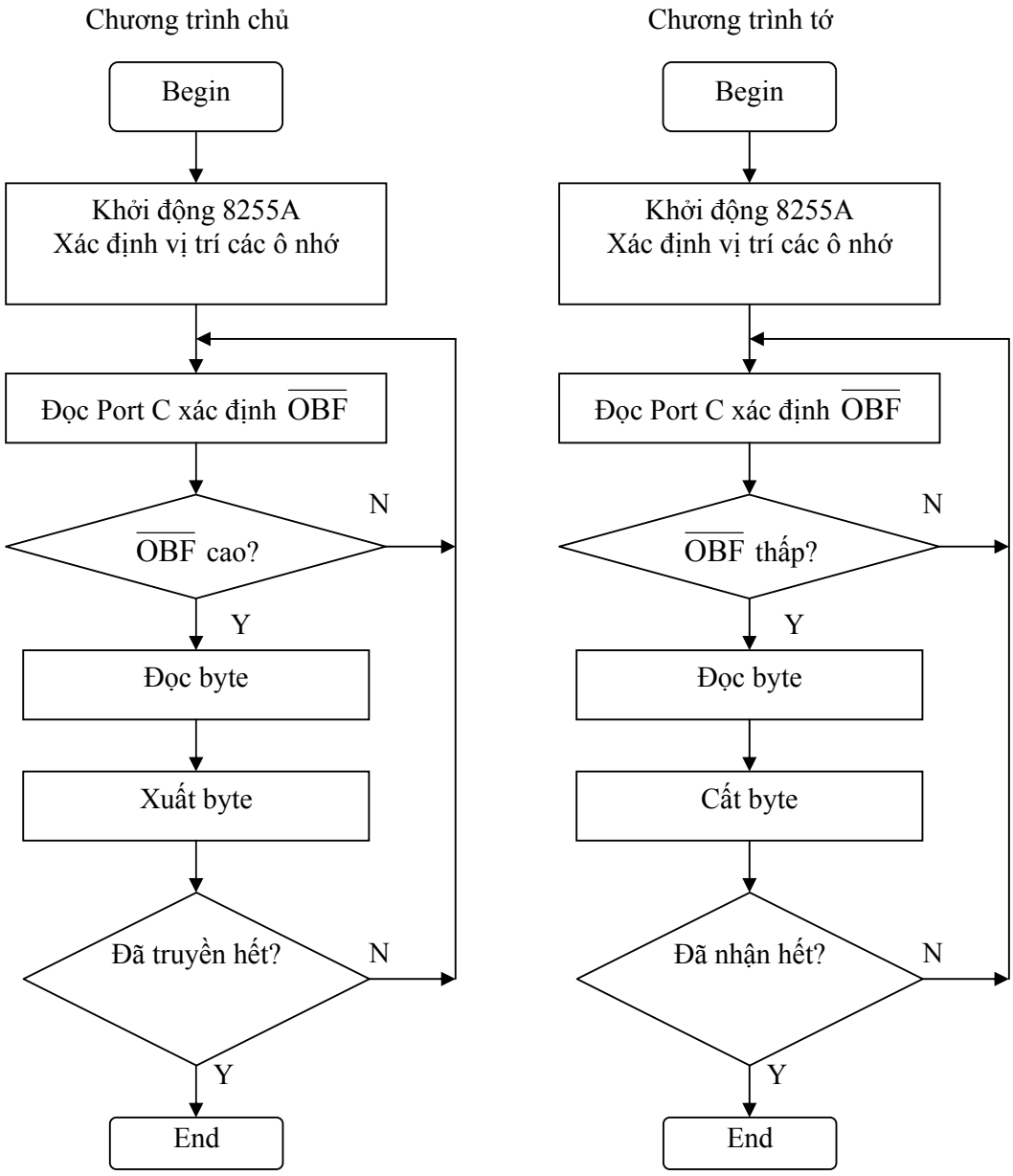
D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBF}_A$	$INTE_1$	IBFA	$INTE_2$	INTRA	X	X	X

Trạng thái của tín hiệu $\overline{OBF}$ được kiểm tra bằng cách đọc bit D7 và trang thái của IBF kiểm tra bằng bit D0.

❖ **Các tác vụ Đọc và Ghi của µP tớ:**

Một byte dữ liệu có thể được đọc bởi µP tớ từ Port A bằng cách gởi một xung chọn thiết bị tác động mức thấp đến tín hiệu $\overline{ACK}$, không cần xây dựng Port nhập. Tương tự, một byte dữ liệu có thể được ghi vào µP bằng cách đưa tín hiệu $\overline{STB}$ xuống thấp.

❖ **Lưu đồ giải thuật:**



❖ **Chương trình:**

➢ **Đoạn chương trình chủ: (Master program)**

```
MOV     SP,stack1
MOV     SI,master       ; Địa chỉ các byte cần xuất
MOV     CX,byte_no      ; Số byte cần xuất
MOV     AL,0C0h         ; Từ điều khiển
MOV     DX,0FFh         ; Địa chỉ thanh ghi điều khiển
OUT     DX,AL
```


```
next:   MOV     DX,0FEh     ; Địa chỉ Port C
wait:   IN      AL,DX       ; Đọc vào từ Port C
        AND     AL,80h      ; Kiểm tra OBF
        JNE     wait        ; Chờ đến khi OBF = 0
        LODSB               ; Đọc byte
        MOV     DX,0FCh     ; Xuất byte vừa đọc
        OUT     DX,AL       ; ra Port A
        LOOP    next        ; Nếu còn byte truyền thì tiếp tục
        END
```

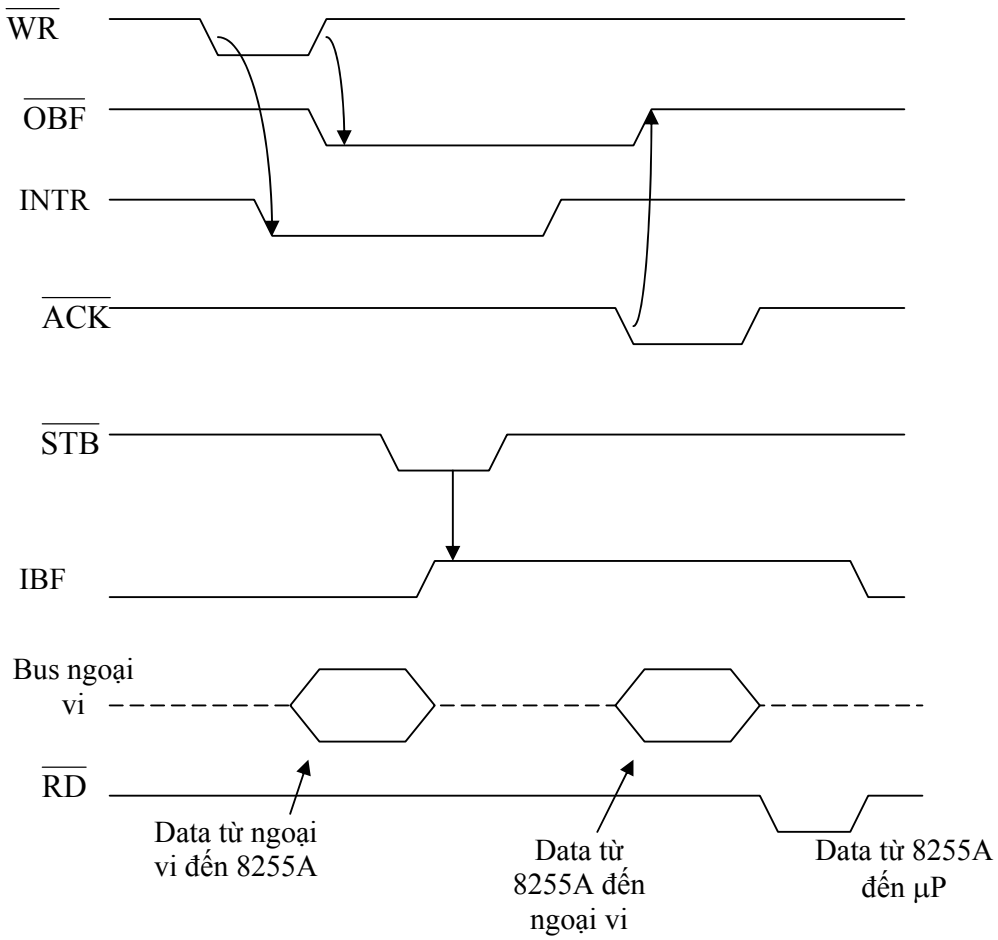
➢ **Đoạn chương trình tớ: (Slave program)**

```
        MOV     ES,stack2
        MOV     DI,slave    ; Địa chỉ các byte sẽ lưu
        MOV     CX,byte_no  ; Số byte cần nhận
next:   MOV     DX,87h
wait:   IN      AL,DX       ; Đọc OBF
        AND     AL,80h      ; Kiểm tra OBF
        JE      wait        ; Chờ đến khi OBF = 1
        MOV     DX,85h
        IN      AL,DX       ; Đọc dữ liệu
        STOSB               ; Cất vào ô nhớ
        LOOP    next        ; Nếu còn byte truyền thì tiếp tục
        END
```

- Ta thấy rằng cả hai chương trình sẽ kiểm tra trạng thái $\overline{OBF}$. Chương trình chủ đợi cho đến khi $\overline{OBF}$ lên mức cao sẽ ghi một byte vào Port A. Ngược lại, chương trình tớ đợi cho đến khi $\overline{OBF}$ xuống mức thấp thì sẽ đọc dữ liệu.

- Khi µP chủ ghi một byte dữ liệu, nó sẽ chốt tại Port A và byte dữ liệu được đặt trên data bus của µP tớ khi $\overline{ACK}$ xuống mức thấp.

- Hai chương trình trên chỉ cho phép truyền một khối dữ liệu từ µP chủ đến µP tớ nhưng không thể truyền ngược lại. Để chuyển một khối dữ liệu từ µP tớ đến µP chủ, cần phải đọc tín hiệu IBF. µP chủ đợi cho đến khi IBF = 1 thì sẽ đọc một byte dữ liệu còn µP tớ đợi cho đến khi IBF = 0 thì ghi một byte dữ liệu.
- Giản đồ thời gian ở hình 3.4.16 cho thấy tín hiệu INTR dùng để truyền dữ liệu bằng ngắt. Trong ví dụ này, ta không sử dụng ngắt.



Hình 4.20 – Giản đồ thời gian ở Mode 2

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.

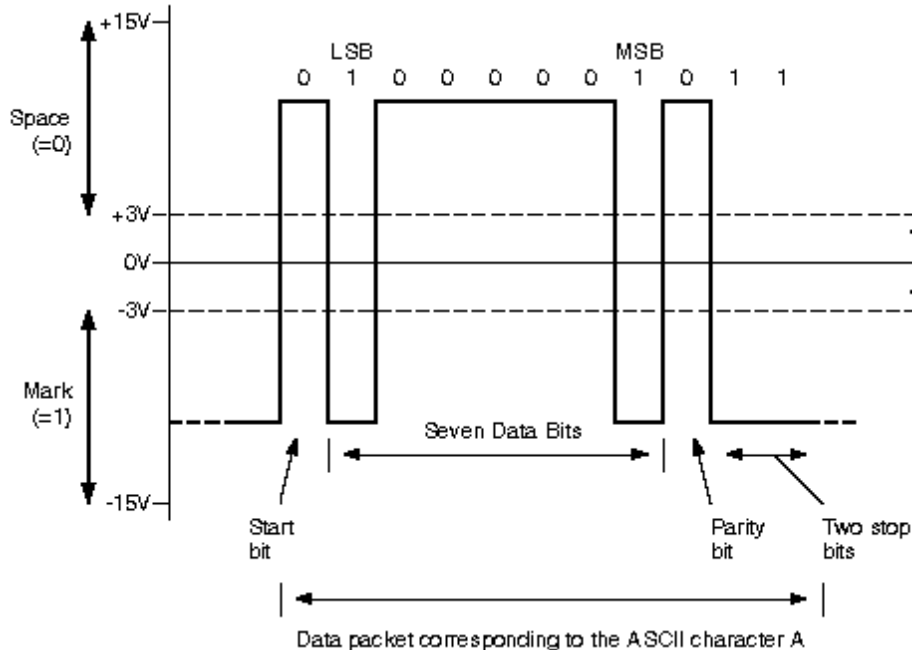
- Song công (full-duplex): số liệu được truyền đồng thời theo 2 hướng.

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):



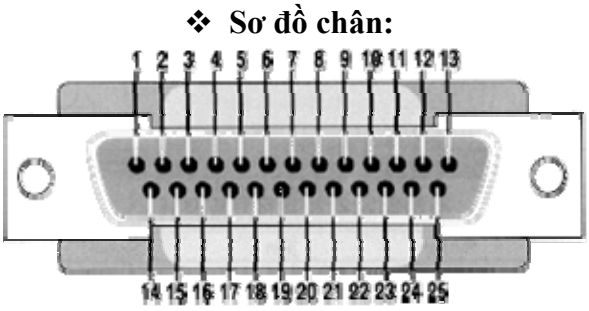
Hình 4.1 – Tín hiệu truyền của ký tự 'A'

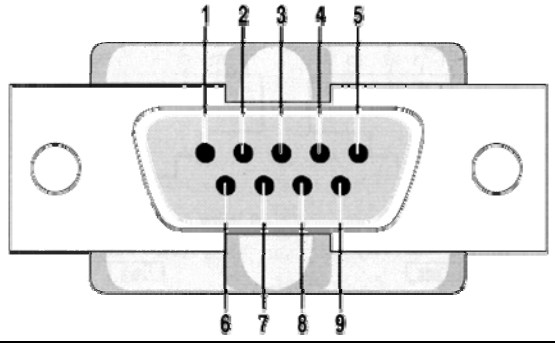
Các đặc tính kỹ thuật của chuẩn RS-232 như sau:

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:**





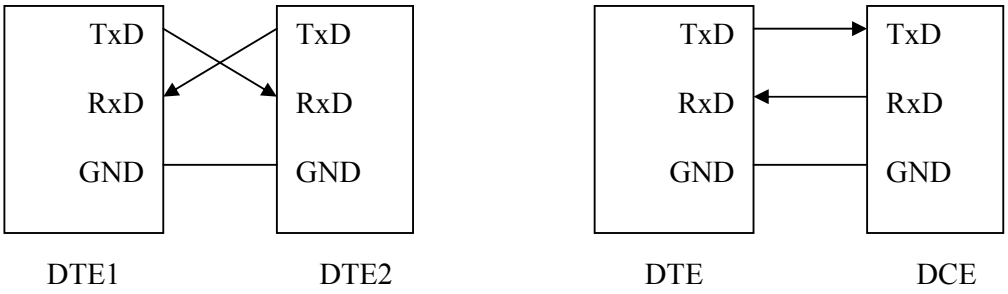
Hình 4.2 – Sơ đồ chân cổng nối tiếp

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

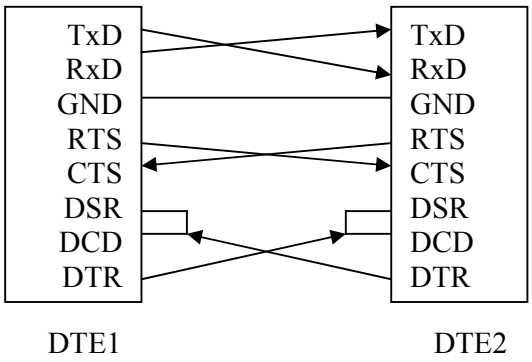
Các sơ đồ khi kết nối dùng cổng nối tiếp:



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:

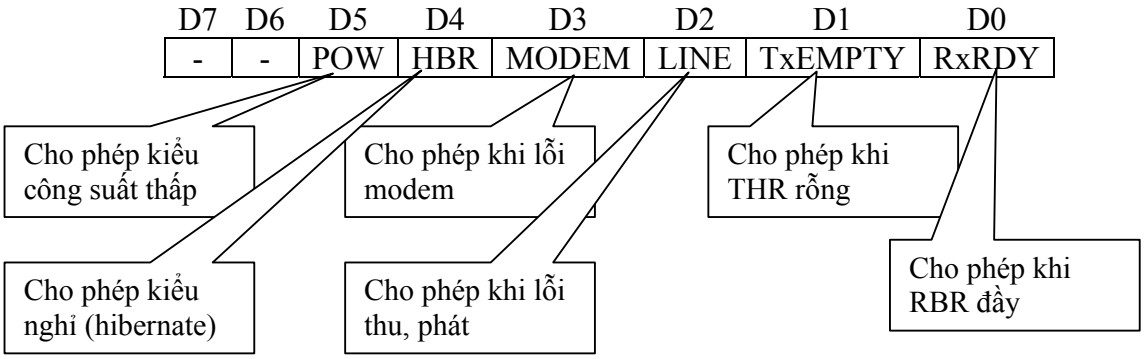
D7 D6	D5	D4	D3	D2 D1	D0
00: không có FIFO 11: cho phép FIFO	Cho phép FIFO 64 byte (trong 16750)	-	1: ngắt time-out (trong 16550)	Xác định nguồn gốc ngắt	0: có ngắt 1: không ngắt

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

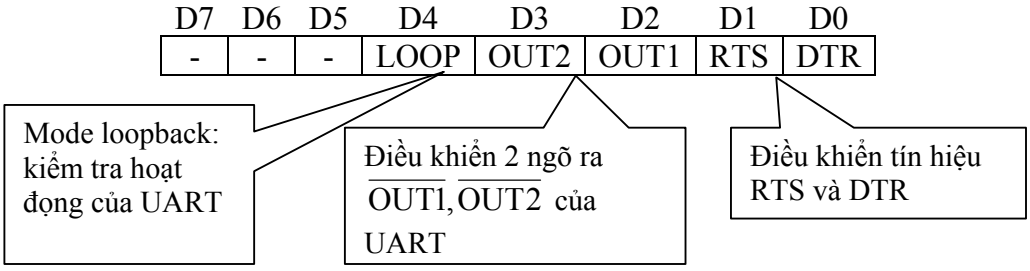
(mức 1 ưu tiên cao nhất)

❖ **IER (Interrupt Enable Register):**

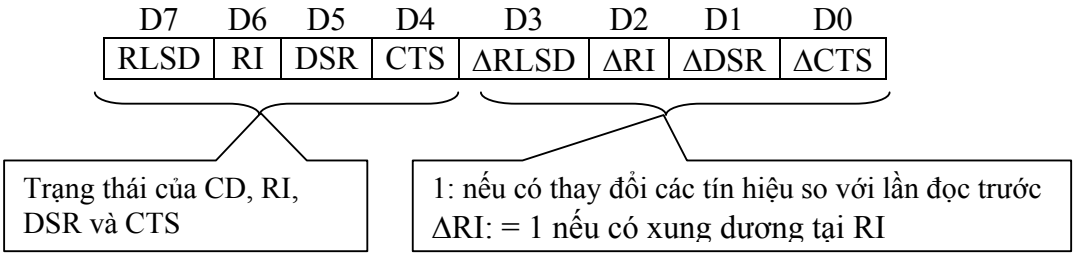
IER cho phép hay cấm các nguyên nhân ngắt khác nhau (1: cho phép, 0: cầm ngắt)



❖ **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:

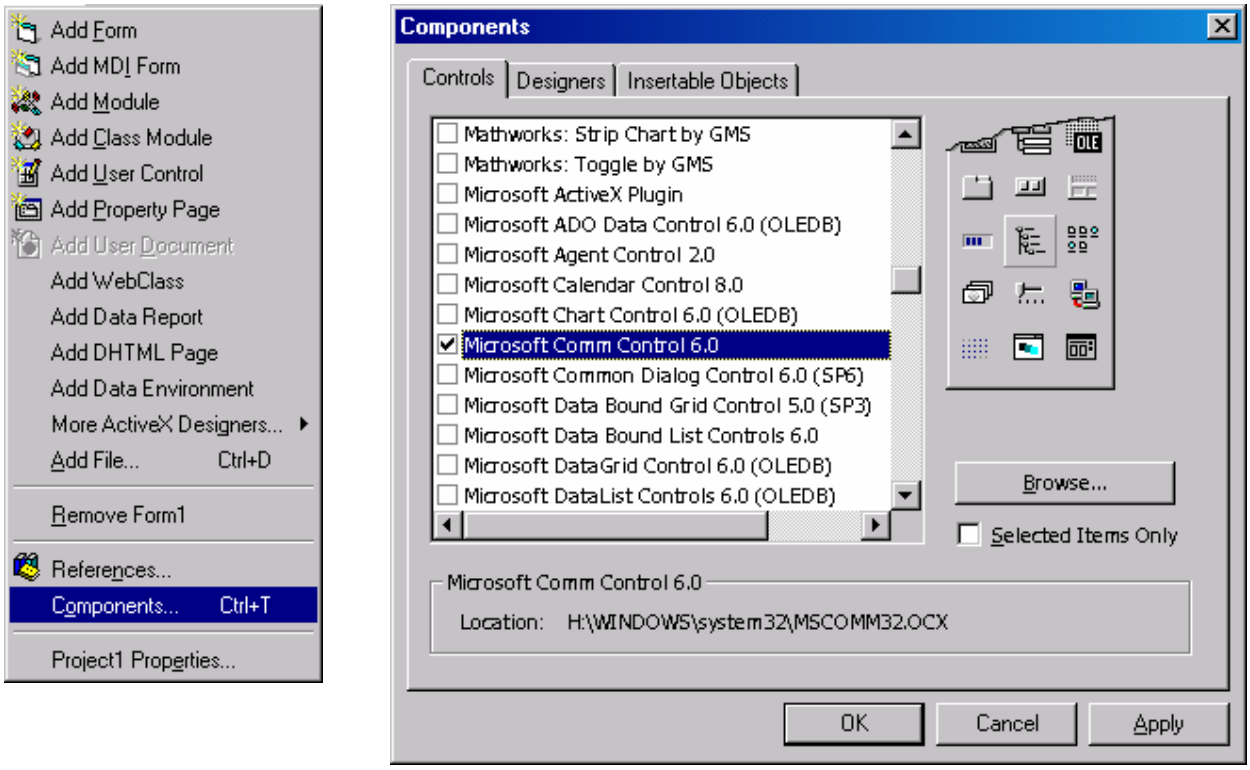
- **Điều khiển sự kiện:**

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.

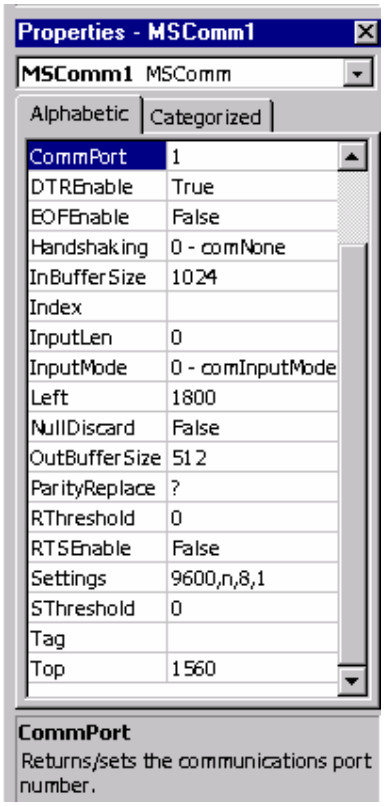
ActiveX MsComm được bổ sung vào một Visual Basic Project thông qua menu **Project > Components**:



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:

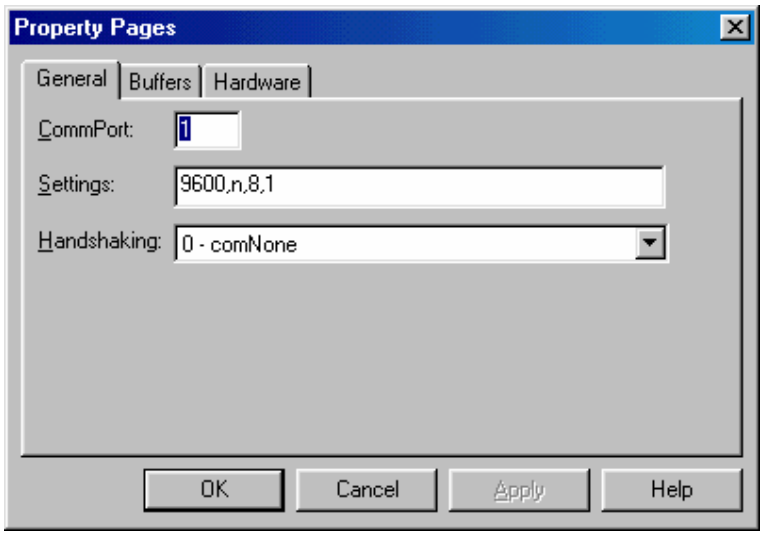
Lỗi	Giá trị	Mô tả
ComBreak	1001	Nhận tín hiệu Break
ComCTSTO	1002	Carrier Detect Timeout
ComFrame	1004	Lỗi khung
ComOver	1006	Phần cứng không đọc ký tự trước khi gởi ký tự kế
ComCDTO	1007	Carrier Detect Timeout
ComRxOver	1008	Tràn bộ đệm nhận
ComRxParity	1009	Lỗi parity
ComTxFull	1010	Tràn bộ đệm truyền

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.

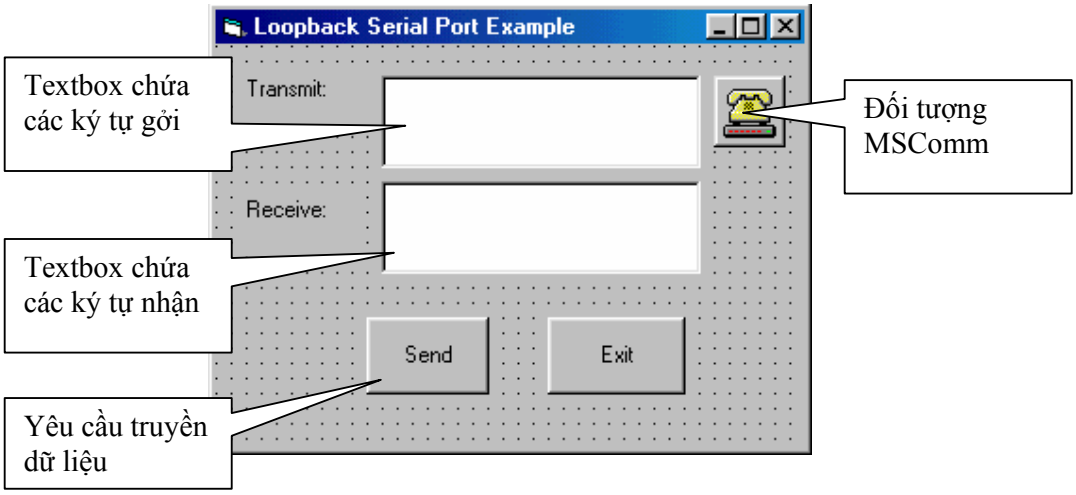
Một chương trình truyền nhận đơn giản thực hiện bằng cách nối chân TxD với RxD của cổng COM1 (loopback). Phương pháp này dùng để kiểm tra cổng nối tiếp.

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:

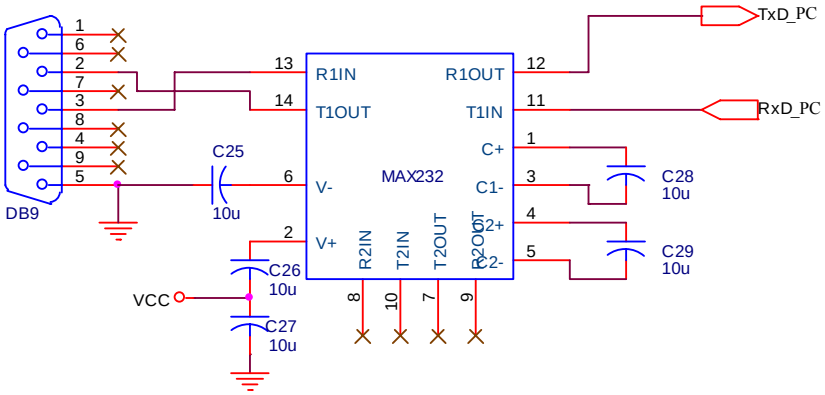
```
VERSION 5.00
Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
```


```
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
```

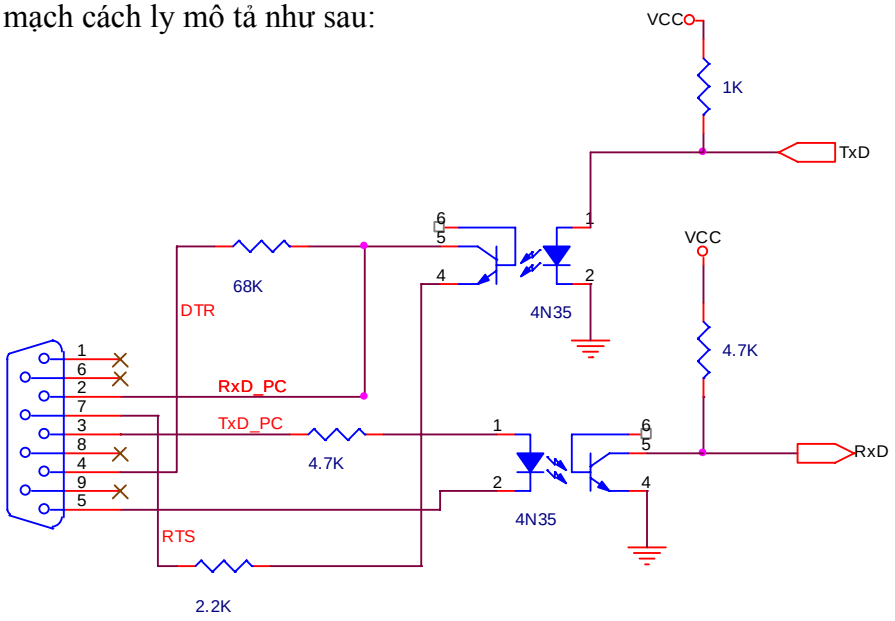

5. Giao tiếp với vi điều khiển

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:



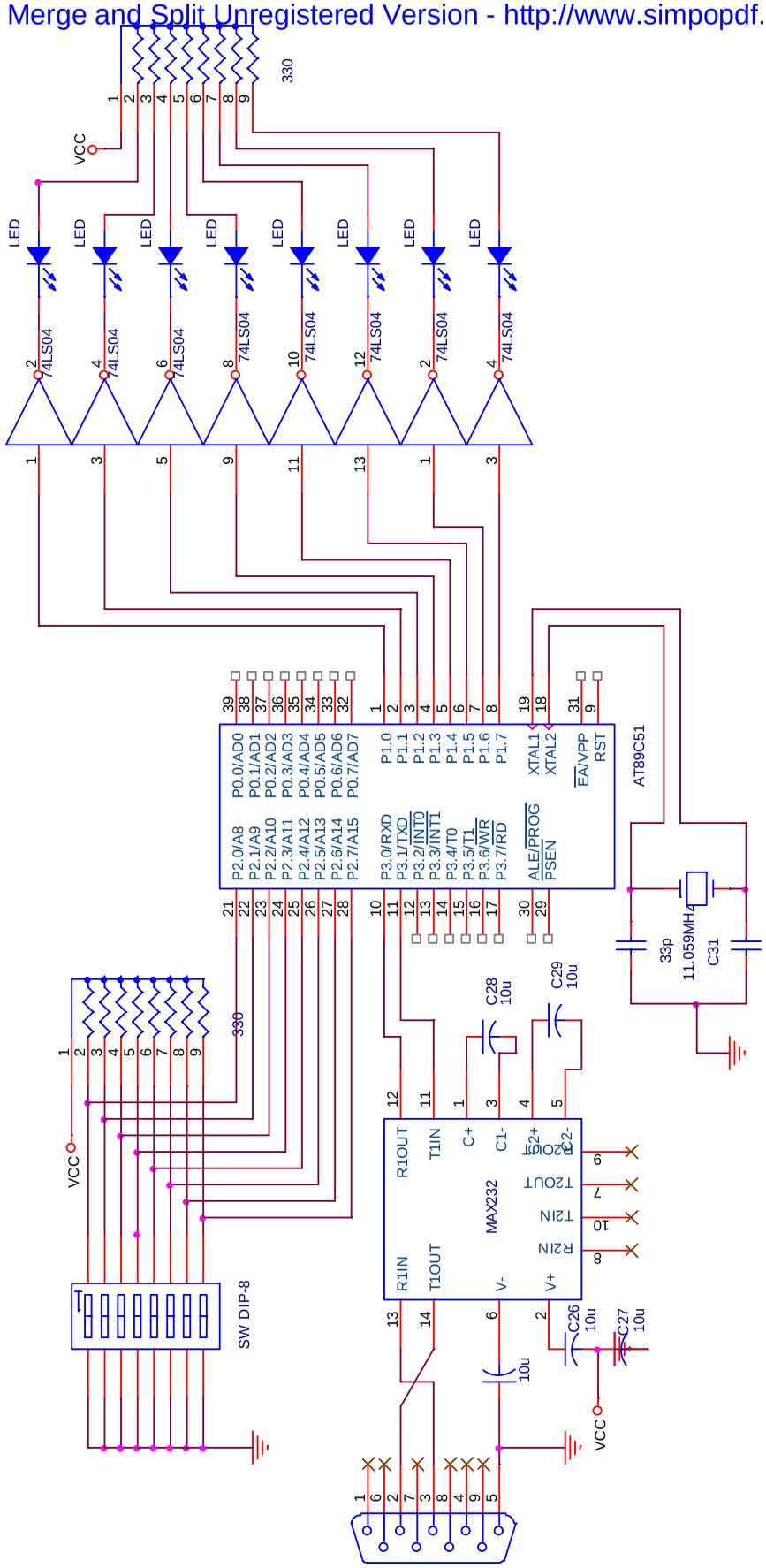
Hình 4.9 – Mạch chuyển mức logic TTL ↔ RS232

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:

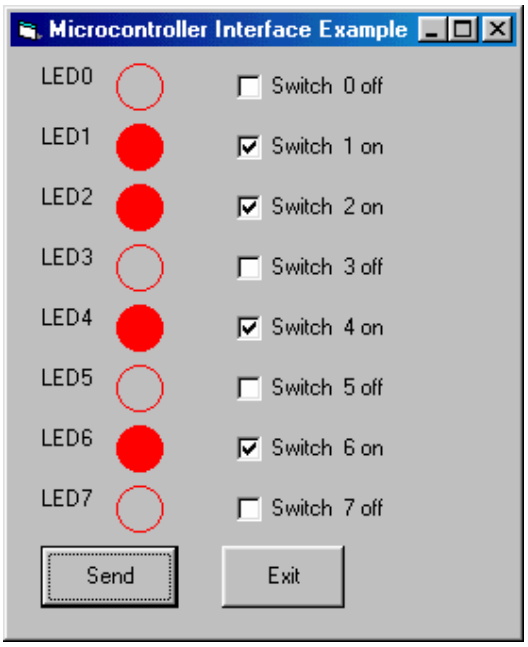


Hình 4.11 – Kết nối với vi điều khiển

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 tăc 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.

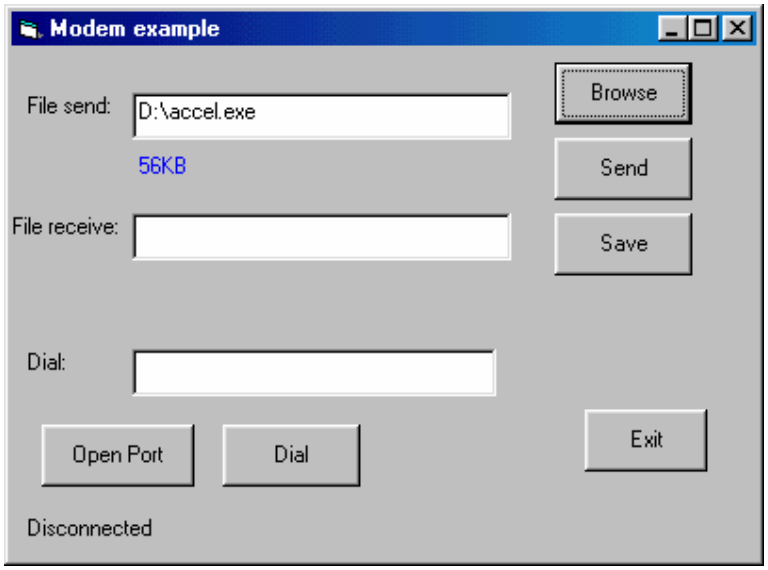
Thanh ghi S11: xác định tốc độ quay số khi sử dụng phương pháp quay số tone, giá trị mặc nhiên tùy vào modem, thường vào khoảng 70ms.

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
```

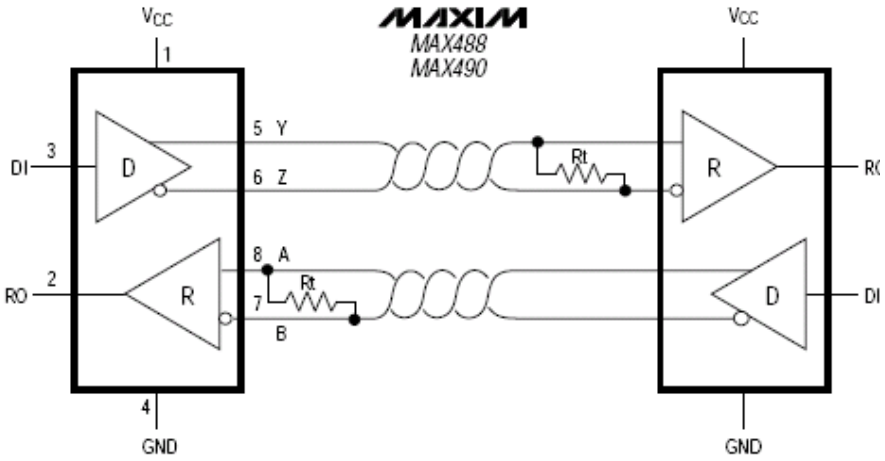

```
Private Sub cmdOpen_Click()
MSComm1.PortOpen = Not MSComm1.PortOpen
If MSComm1.PortOpen Then
    cmdOpen.Caption = "Close Port"
    MSComm1.Output = "ATS0=5" + vbCr
    Call Form_Load
Else
    cmdOpen.Caption = "Open Port"
    lblStatus.Caption = "Disconnected"
End If
End Sub

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
```

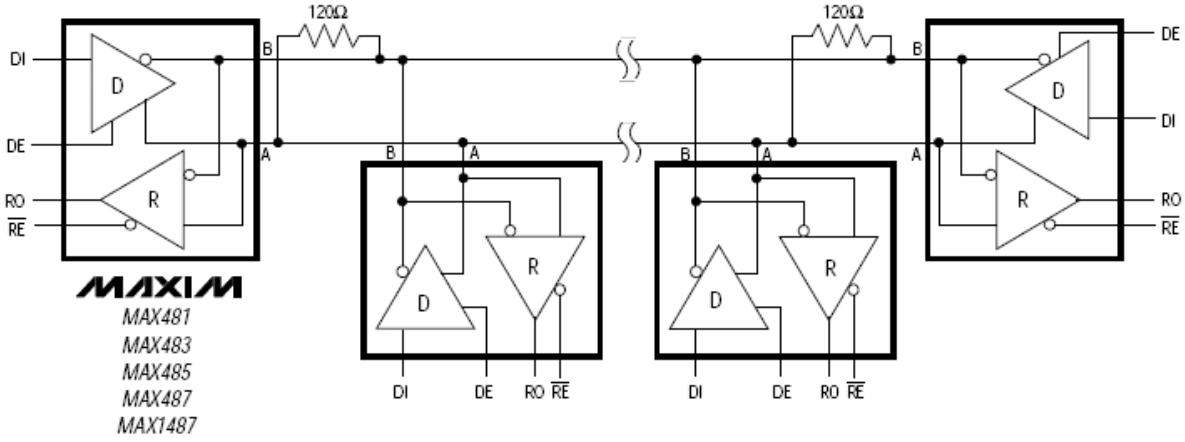

```
                    Buffer = ""
                    Exit Sub
                End If
                If InStr(Buffer, "CONNECT") Then
                    Connected = True
                    lblStatus.Caption = "Connected"
                    Exit Sub
                End If
                If ReceiveFlag 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



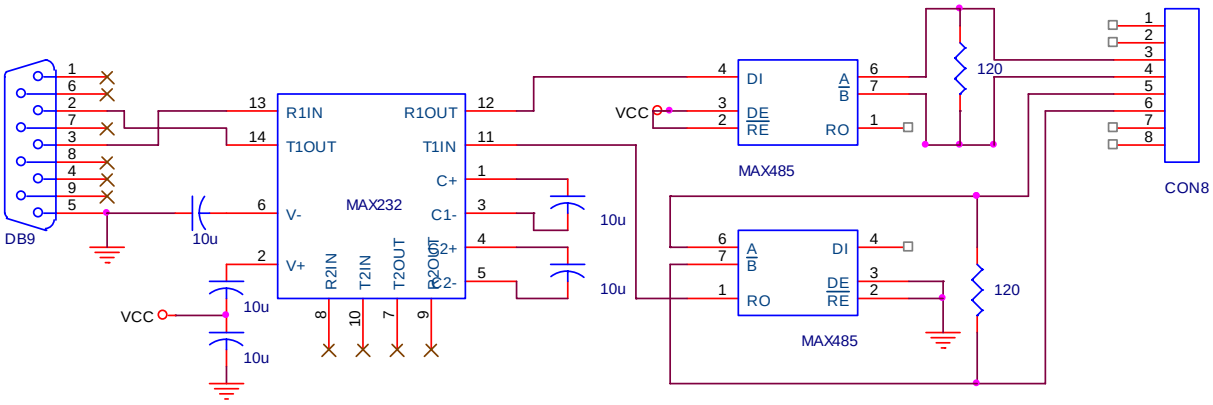
Hình 4.14 – Chuẩn giao tiếp RS485

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

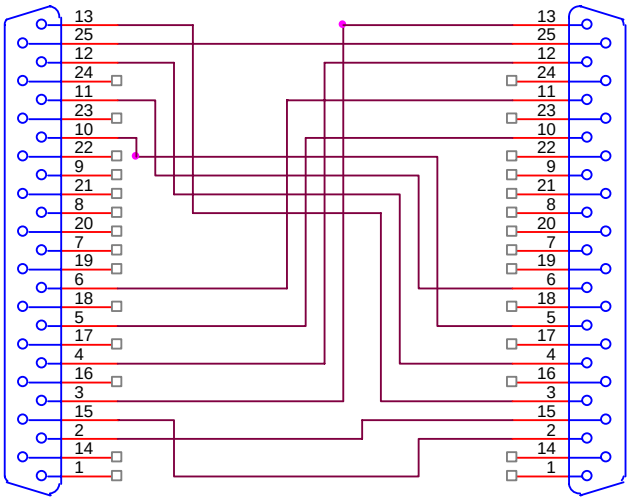
Chú ý rằng chân BUSY được nối với cổng đảo trước khi đưa vào thanh ghi trạng thái, các bit $\overline{\text{SELECTIN}}$, $\overline{\text{AUTOFEED}}$ và $\overline{\text{STROBE}}$ được đưa qua cổng đảo trước khi đưa ra các chân của cổng máy in.

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:

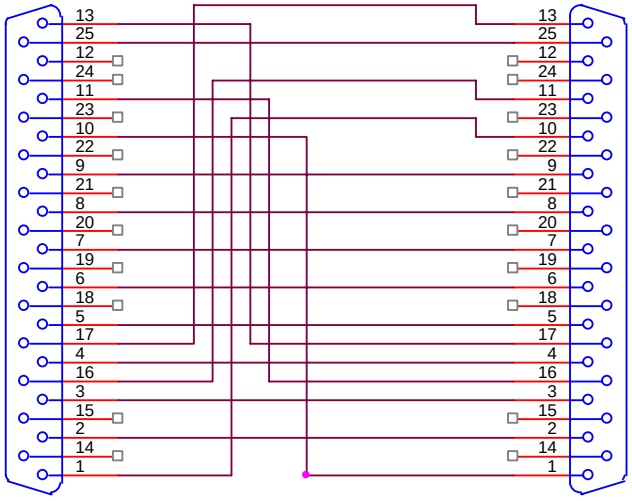


Hình 5.1 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ chuẩn

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.



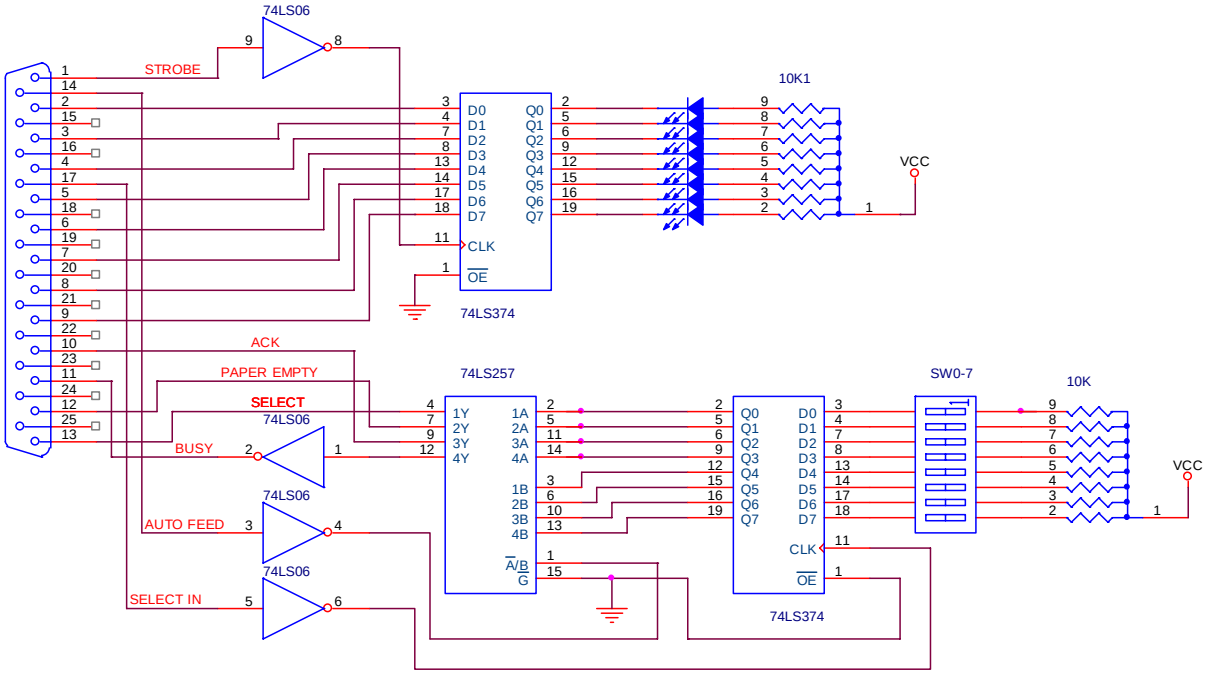
Hình 5.2 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ mở rộng

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	$\overline{\text{ACK}}$

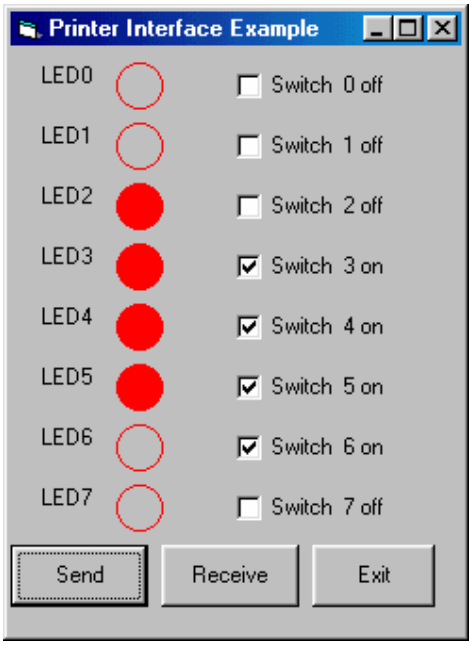
2.2. Giao tiếp thiết bị khác

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
```