

TRƯỜNG ĐẠI HỌC VINH  
VIỆN KỸ THUẬT VÀ CÔNG NGHỆ

---



**TÀI LIỆU HƯỚNG DẪN THỰC HÀNH**  
**HỌC PHẦN: ĐIỆN TỬ SỐ VÀ VI XỬ LÝ**  
MÃ HỌC PHẦN: ELE20006

**NGHỆ AN - 2022**

## MỤC LỤC

	Trang
<b>NỘI QUY PHÒNG THÍ NGHIỆM .....</b>	<b>02</b>
<b>BÀI 1:</b>	
<b>MẠCH LOGIC TỔ HỢP .....</b>	<b>03</b>
<b>BÀI 2:</b>	
<b>MẠCH LOGIC TUẦN TỤ .....</b>	<b>17</b>
<b>BÀI 3:</b>	
<b>LẬP TRÌNH CHO VI XỬ LÝ 8086 .....</b>	<b>23</b>
<b>BÀI 4:</b>	
<b>LẬP TRÌNH HỢP NGỮ CHO VI ĐIỀU KHIỂN 8051 .....</b>	<b>37</b>
<b>BÀI 5:</b>	
<b>LẬP TRÌNH CHO BỘ ĐÉM/BỘ ĐỊNH THỜI VI ĐIỀU KHIỂN 8051 .....</b>	<b>55</b>
<b>BÀI 6:</b>	
<b>LẬP TRÌNH SỬ DỤNG NGẮT TRONG VI ĐIỀU KHIỂN 8051 .....</b>	<b>67</b>

BỘ GIÁO DỤC VÀ ĐÀO TẠO  
TRƯỜNG ĐẠI HỌC VINH

**NỘI QUY**  
**PHÒNG THÍ NGHIỆM, PHÒNG HỌC CÓ THIẾT BỊ ĐIỆN TỬ**

Phòng thí nghiệm, Phòng học tiếng, Phòng học máy tính, Phòng Multimedia thuộc trường Đại học Vinh là nơi học tập, nghiên cứu khoa học của cán bộ và học sinh, sinh viên nhà trường. Khi đến làm việc, học tập tại đây, mọi người phải thực hiện đầy đủ nội quy sau đây:

1. Mặc trang phục đúng quy định. Không mang vật dụng dễ gây cháy, nổ, ẩm ướt, quà bánh vào phòng.
2. Nghiêm chỉnh thực hiện hướng dẫn của giáo viên phụ trách. Tuyệt đối không được tuỳ tiện điều chỉnh máy, phương tiện thí nghiệm hoặc thao tác không theo quy trình của bài học. Nếu thấy máy móc, thiết bị thí nghiệm có sự cố thì phải báo cáo ngay cho cán bộ phụ trách để xử lý.
3. Giữ gìn bảo quản chu đáo, không làm hư hỏng máy móc, thiết bị. Ai làm hư hỏng, mất mát tài sản không có lý do chính đáng thì phải bồi thường.
4. Không mang tài sản ra khỏi phòng nếu chưa được phép của Nhà trường.
5. Giữ gìn trật tự, không hút thuốc lá nơi làm việc, học tập. Không vẽ viết bậy lên bàn họ, tường nhà...
6. Phụ tá thí nghiệm, cán bộ kỹ thuật phòng máy chuẩn bị đầy đủ vật tư thiết bị, máy móc học tập, thí nghiệm, ghi chép nhật ký đúng quy định. Kiểm tra, thu nhận, cất đặt thiết bị chu đáo. Vát bỏ mẫu thí nghiệm đúng nơi quy định.
7. Khi học xong, giáo viên cần nhắc nhở học sinh, sinh viên và cán bộ phụ trách đóng máy, lau chùi, thu dọn và tắt điện đúng quy trình.
8. Sau mỗi buổi học, cán bộ phụ tá hoặc kỹ thuật viên phải kiểm tra lại toàn bộ phòng óc, đóng cài cửa cẩn thận trước lúc ra về.
9. Nội quy này áp dụng kể từ ngày ký, các quy định trước đây trái với nội quy này đều bãi bỏ. Ai thực hiện tốt nội quy sẽ được khen thưởng, ai vi phạm sẽ bị xử lý theo pháp luật hiện hành.

**HIỆU TRƯỞNG**

(Đã ký)

## BÀI 1:

# MẠCH LOGIC TỔ HỢP

## 1. MỤC ĐÍCH

Bài thực hành nhằm cung cấp cho sinh viên các kiến thức, kỹ năng về khảo sát, phân tích các phần tử logic cơ bản và các mạch logic tổ hợp sử dụng phần mềm mô phỏng Protues.

## 2. TÓM TẮT LÝ THUYẾT

### 2.1. Các phần tử logic cơ bản

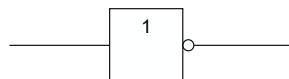
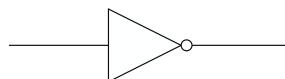
#### 2.1.1. Phép toán NOT và công NOT:

- Phép toán NOT hay còn được gọi là phép đảo hay phép phủ định.
- Biểu diễn:  $y = \bar{x}$
- Bảng trạng thái:

$x$	$y = \bar{x}$
0	1
1	0

- Công NOT là mạch duy nhất có một đầu vào và mức logic ở đầu ra ngược với mức logic ở đầu vào.

- Ký hiệu logic:



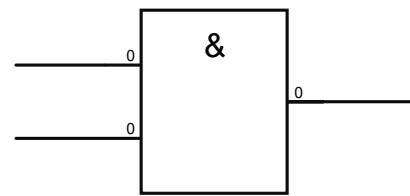
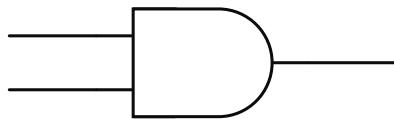
#### 2.1.2. Phép toán AND và công AND:

- Phép toán AND hay còn được gọi là phép nhân logic.
- Biểu diễn:  $y = x_1 x_2$
- Bảng trạng thái:

$x_1$	$x_2$	$y = x_1 x_2$
0	0	0
0	1	0
1	0	0
1	1	1

- Mở rộng cho trường hợp tổng quát n biến:  $y = x_1 x_2 \dots x_n$
- Công AND: là có từ hai đầu vào trở lên và một đầu ra bằng tổ hợp AND các biến đầu vào.

- Ký hiệu logic:



### 2.1.3. Phép toán OR và công OR:

- Phép toán OR hay còn được gọi là phép cộng logic.

- Biểu diễn:  $y = x_1 + x_2$

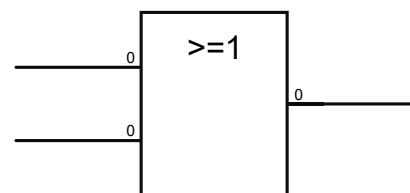
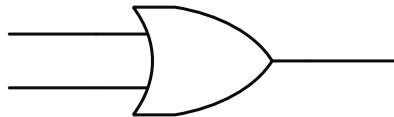
- Bảng trạng thái:

$x_1$	$x_2$	$y = x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	1

- Mở rộng cho trường hợp tổng quát n biến:  $y = x_1 + x_2 + \dots + x_n$

- Công OR: là có từ hai đầu vào trở lên và một đầu ra bằng tổ hợp OR các biến đầu vào.

- Ký hiệu logic:



### 2.1.4. Phép toán NAND và công NAND:

- Phép toán NAND hay NOT AND.

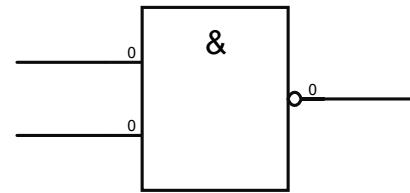
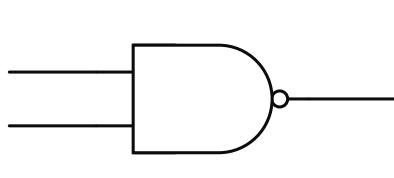
- Biểu diễn:  $y = \overline{x_1 x_2}$

- Bảng trạng thái:

$x_1$	$x_2$	$y = \overline{x_1 x_2}$
0	0	1
0	1	1
1	0	1
1	1	0

- Mở rộng cho trường hợp tổng quát n biến:  $y = \overline{x_1 x_2 \dots x_n}$

- Ký hiệu logic:

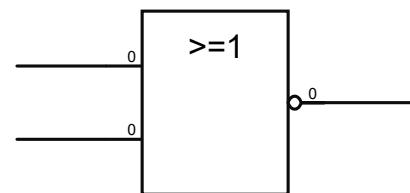
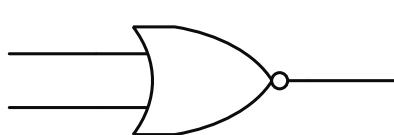


### 2.1.5. Phép toán NOR và công NOR:

- Phép toán NOR hay NOT OR.
- Biểu diễn:  $y = \overline{x_1 + x_2}$
- Bảng trạng thái:

$x_1$	$x_2$	$y = \overline{x_1 + x_2}$
0	0	0
0	1	1
1	0	1
1	1	1

- Mở rộng cho trường hợp tổng quát n biến:  $y = \overline{x_1 + x_2 + \dots + x_n}$
- Ký hiệu logic:



### 2.1.6. Phép toán XOR và công XOR:

- Được gọi là hàm cộng loại trừ, hàm cộng modul 2, hàm cộng tuyệt đối, hàm không tương đương, ....

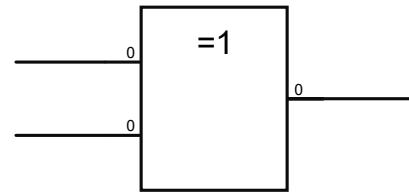
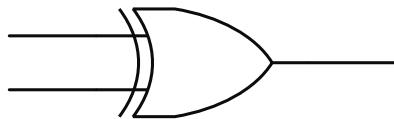
- Biểu diễn:  $y = x_1 \overline{x_2} + \overline{x_1} x_2$  hay  $y = x_1 \oplus x_2$

- Bảng trạng thái:

$x_1$	$x_2$	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- Mở rộng cho trường hợp tổng quát n biến:  $y = x_1 \oplus x_2 \oplus \dots \oplus x_n$
- + Nếu số mức logic 1 ở đầu vào là lẻ  $\rightarrow$  lối ra là mức logic 1.
- + Nếu số mức logic 1 ở đầu vào là chẵn  $\rightarrow$  lối ra là mức logic 0.

- Ký hiệu logic cỗng XOR:



### 2.1.7. Phép toán XNOR và cỗng XNOR:

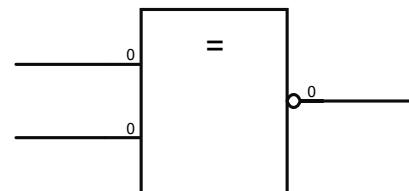
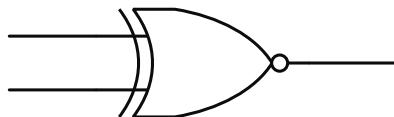
- Được gọi là hàm không hoặc loại trừ, hàm không hoặc tuyệt đối, hàm tương đương, ....

- Biểu diễn:  $y = \overline{x_1 x_2} + x_1 x_2$  hay  $y = \overline{x_1 \oplus x_2} = x_1 \sim x_2$

- Bảng trạng thái:

$x_1$	$x_2$	$y = x_1 \sim x_2$
0	0	1
0	1	0
1	0	0
1	1	1

- Ký hiệu logic cỗng XNOR:



## 2.2. Mạch tổ hợp

Mạch tổ hợp (combinational circuits): là các mạch có giá trị ổn định của tín hiệu lối ra ở một thời điểm bất kỳ chỉ phụ thuộc vào tổ hợp các giá trị đầu vào tại thời điểm đó, không phụ thuộc vào các đầu vào ở trạng thái trước đó.

Ví dụ: các cỗng logic cơ bản; các bộ số học; các bộ hợp kênh và phân kênh; các bộ mã hóa và giải mã ...

### 2.2.1. Bộ so sánh

Trong nhiều trường hợp cần phải so sánh hai số nhị phân  $A$  và  $B$  để chỉ ra được mối quan hệ giữa chúng:  $A > B$ ;  $A < B$ ;  $A = B$ .

a. Bộ so sánh hai số nhị phân 1 bit

Bộ so sánh hai số nhị phân 1 bit là mạch thực hiện chức năng so sánh hai số nhị phân 1 bit.

Bộ so sánh 1 bit có hai ngõ vào và 3 ngõ ra.

Giả sử  $a, b$  là ngõ vào các bit cần so sánh; các ngõ ra thể hiện kết quả so sánh:  $y_1(a < b)$ ,  $y_2(a = b)$ ,  $y_3(a > b)$ .

Bảng chân lý mạch so sánh như sau:

$a$	$b$	$y_1$	$y_2$	$y_3$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

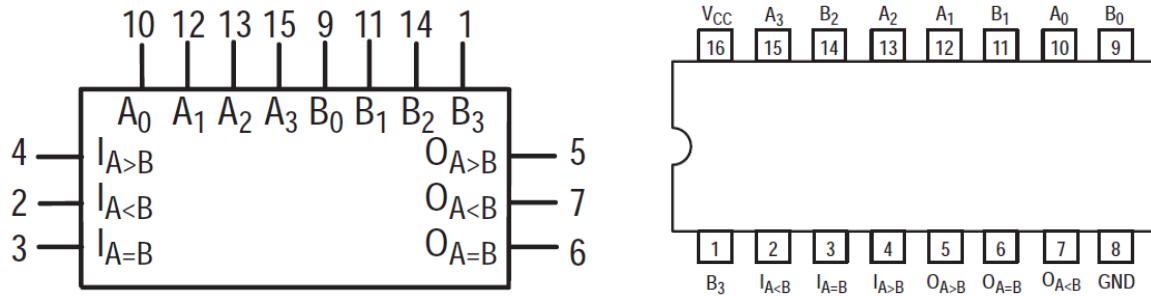
Phương trình logic:  $y_1 = \overline{ab}$ ;  $y_2 = \overline{\overline{ab}} + ab = \overline{a \oplus b}$ ;  $y_3 = ab$ .

Muốn so sánh hai số nhiều bit ta phải tuân theo trình tự so sánh từ bit cao nhất trước (bit có nhiều ý nghĩa nhất):

- Nếu số nào có bit cao lớn hơn thì số đó sẽ lớn hơn và kết thúc việc so sánh.
- Nếu hai bit có trong số cao nhất bằng nhau thì sẽ so sánh hai số có trọng số thấp hơn, cứ như vậy cho đến bit thấp nhất;
- Hai số bằng nhau nếu tất cả các bit tung ứng của hai số đều bằng nhau.

b. Vi mạch (IC) so sánh hai số 4 bit 74LS85

- Ký hiệu logic và sơ đồ các chân vi mạch 74LS85 như sau:



trong đó:  $A_3A_2A_1A_0$  là số hạng thứ nhất;  $B_3B_2B_1B_0$  là số hạng thứ hai;  $I_{A>B}$ ,  $I_{A<B}$ ,  $I_{A=B}$  là các ngõ vào nối tầng;  $O_{A>B}$ ,  $O_{A<B}$ ;  $O_{A=B}$  là các ngõ ra.

Bảng chân lý của vi mạch như sau:

COMPARING INPUTS				CASCAADING INPUTS			OUTPUTS		
$A_3, B_3$	$A_2, B_2$	$A_1, B_1$	$A_0, B_0$	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	X	X	H	L	L	H
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	H	L	L	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	L	H	H	L

Dựa vào bảng chân lý, ta thấy:

- Khi dùng IC 74LS85 để so sánh 2 số 4 bit ta phải giữ ngõ vào nối tầng  $I_{A=B}$  ở

mức cao, hai ngõ vào nối tầng còn lại ở mức thấp, như vậy vi mạch mới thể hiện được kết quả so sánh.

- Khi so sánh 2 số nhiều bit hơn ta phải dùng nhiều IC 74LS85 và nối ngõ ra của vi mạch so sánh bit thấp vào ngõ vào nối tầng tương ứng của các vi mạch so sánh các bit cao hơn và vi mạch so sánh các bit thấp nhất có ngõ vào nối tầng được mắc như khi dùng riêng lẻ.

### 2.2.2. Bộ cộng

#### a. Mạch cộng bán phần (Half adder - HA)

Mạch cộng bán phần là mạch cộng hai số 1 bit.

$a$	$b$	$S$	$C$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Phương trình logic:

$$S = a \oplus b;$$

$$C = ab.$$

#### b. Mạch cộng toàn phần (Full adder - FA)

Sự khác biệt chính giữa bộ cộng bán phần và bộ cộng toàn phần là bộ cộng toàn phần có ba ngõ vào và hai ngõ ra.

$C_{n-1}$	$A_n$	$B_n$	$S_n$	$C_n$
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

Phương trình logic:

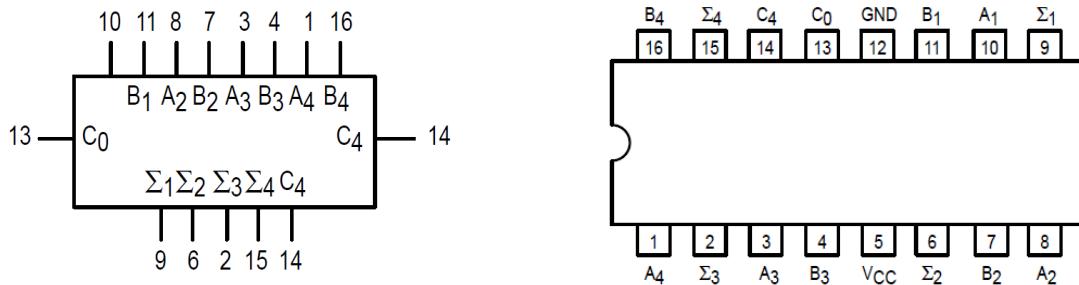
$$S = C_{n-1} + (A_n \oplus B_n);$$

$$C = A_n B_n + C_{n-1} (A_n \oplus B_n).$$

Để 2 số cộng có nhiều bit hơn thì cách cộng cũng sẽ tương tự : trước hết cộng 2 bit LSB để cho bit tổng (LSB). Số nhớ được đưa tới để cộng chung với 2 bit kế tiếp bit LSB để cho bit tổng ở hàng kế tiếp cho đến phép cộng cuối cùng giữa 2 bit MSB để được bit tổng ở hàng đó, số nhớ khi này trở thành bit LSB của tổng.

### b. Vi mạch cộng hai số 4 bit 74LS83

- Ký hiệu logic và sơ đồ các chân vi mạch 74LS83 như sau:



trong đó:  $A_4A_3A_2A_1$  là số hạng thứ nhất;  $B_4B_3B_2B_1$  là số hạng thứ hai;  $C_0$  là các ngõ vào nhớ;  $\Sigma_4 \Sigma_3 \Sigma_2 \Sigma_1$  là các ngõ ra tổng;  $C_4$  là ngõ ra nhớ.

Bảng chân lý của vi mạch như sau:

$C_{(n-1)}$	$A_n$	$B_n$	$\Sigma_n$	$C_n$
L	L	L	L	L
L	L	H	H	L
L	H	L	H	L
L	H	H	L	H
H	L	L	H	L
H	L	H	L	H
H	H	L	L	H
H	H	H	H	H

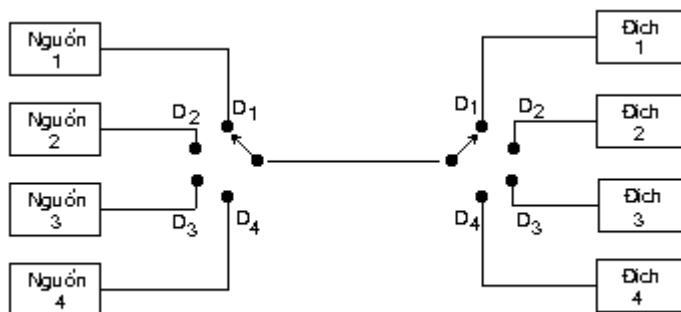
$C_1 - C_3$  are generated internally

$C_0$  — is an external input

$C_4$  — is an output generated internally

#### 2.2.3. Bộ hợp kênh và phân kênh

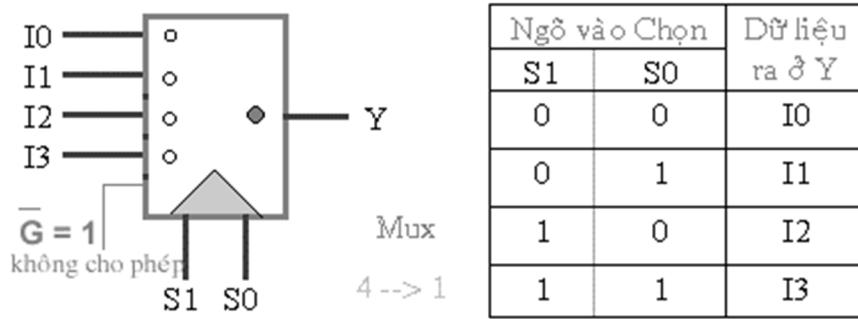
Trong truyền dữ liệu, để tiết kiệm đường truyền, người ta dùng một đường dây để truyền nhiều kênh dữ liệu, như vậy phải thực hiện việc chọn nguồn dữ liệu nào trong các nguồn khác nhau để truyền. Mạch hợp kênh hay còn gọi là mạch chọn dữ liệu sẽ làm công việc này. Ở bên thu, dữ liệu nhận được phải được chuyển tới các đích khác nhau, ta cần mạch phân kênh.



### a. Bộ hợp kênh (Multiplexer - MUX)

Bộ hợp kênh là một dạng mạch tổ hợp cho phép chọn một trong nhiều đường ngõ vào song song (các kênh vào) để đưa tới một ngõ ra (gọi là kênh truyền nối tiếp). Việc chọn đường nào trong các đường ngõ vào do các ngõ vào lựa chọn quyết định.

Ví dụ: mạch dồn kênh 4 sang 1 như sau:



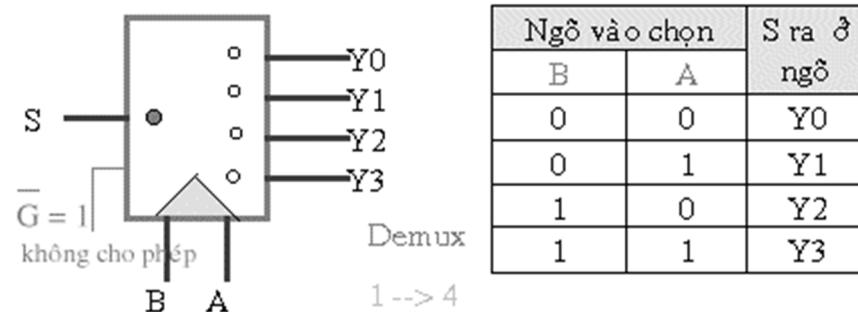
Ta thấy MUX hoạt động như một công tắc nhiều vị trí được điều khiển bởi mã số. Mã số này là dạng số nhị phân, tùy tổ hợp số nhị phân này mà ở bất kì thời điểm nào chỉ có 1 ngõ vào được chọn và cho phép đưa tới ngõ ra.

Các mạch dồn kênh thường gấp là 2 sang 1, 4 sang 1, 8 sang 1, ..., nói chung là từ  $2^n$  sang 1.

#### a. Bộ phân kênh (Demultiplexer - DEMUX)

Bộ phân kênh hay còn gọi là tách kênh, giải đa hợp có chức năng ngược lại với mạch dồn kênh tức là: tách kênh truyền thành một trong các kênh dữ liệu song song tuỳ vào mã chọn ngõ vào.

Ví dụ: mạch phân kênh 1 sang 4 như sau:



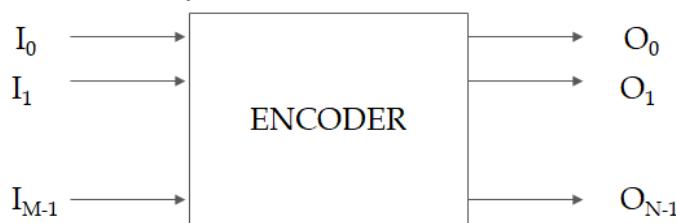
Có thể xem mạch phân kênh giống như một công tắc cơ khí được điều khiển chuyển mạch bởi mã số. Tuỳ theo mã số được áp vào ngõ chọn mà dữ liệu từ 1 đường sẽ được đưa ra đường nào trong số các đường song song.

Các mạch tách kênh thường gấp là 1 sang 2; 1 sang 4; 1 sang 8;... Nói chung từ 1 đường có thể đưa ra  $2^n$  đường, và số đường để chọn sẽ phải là n.

#### 2.2.4. Bộ chuyển đổi mã

##### a. Mạch mã hóa

Mạch mã hóa có M đầu vào và chỉ một trong số đó được kích hoạt tại thời điểm xác định, tạo mã đầu ra N bit, tùy thuộc vào đầu vào nào được kích hoạt.



Ví dụ: bộ mã hóa thập phân sang mã BCD 8421; bộ mã hóa ưu tiên.

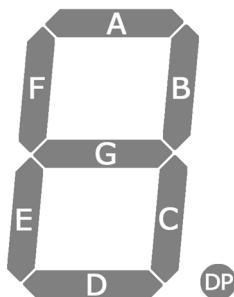
### b. Mạch giải mã

Quá trình ngược lại với mã hóa được gọi là giải mã, nghĩa là từ một tổ hợp giá trị của nhóm mã n chữ số hệ nhị phân ta lại tìm được 1 trong N ký hiệu hoặc số tương ứng với tổ hợp đó.

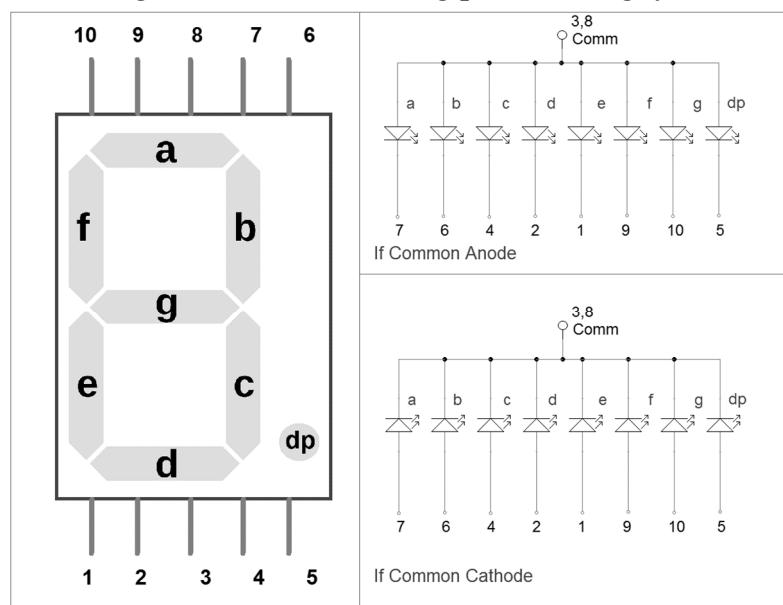
Về thực chất các bộ giải mã cũng là các bộ biến đổi mã, ví dụ: biến đổi các mã nhị phân, BCD sang mã thập phân hay mã 7 đoạn.

### c. Vi mạch giải mã BCD sang led 7 đoạn 74LS47:

LED 7 đoạn là thiết bị hiển thị điện tử để hiển thị số được sử dụng rộng rãi trong đồng hồ số, đồng hồ đo điện tử, máy tính cá nhân.



- Cấu tạo: LED 7 đoạn bao gồm 8 LED được kết nối song song để có thể thấp sáng hiển thị số (0, 1, 2, 3, 4, 5, 7, 8, 9, A, B, C, D, E, F, ...). Mỗi đoạn (LED) được đánh dấu bởi ký tự từ A tới G. Đoạn thứ tám gọi là “chấm thập phân” (Decimal Point) ký hiệu DP được sử dụng khi hiển thị số không phải là số nguyên.



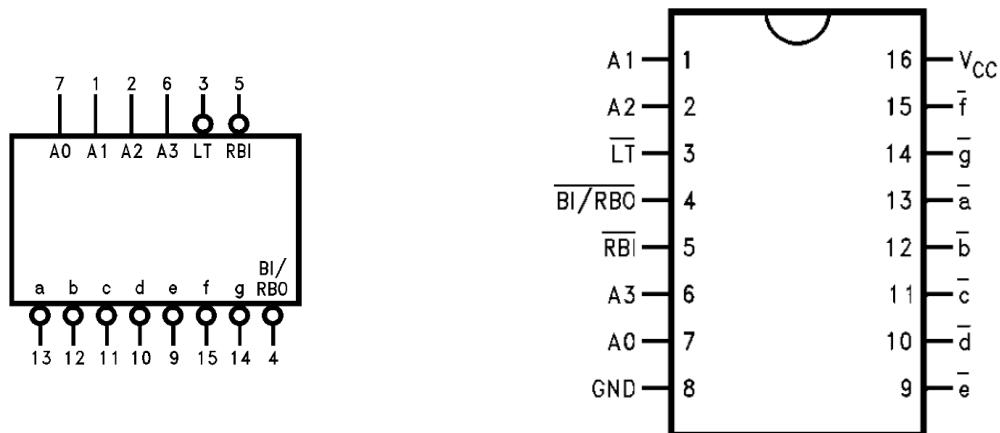
- Phân loại LED 7 đoạn: dựa vào các cực được nối, có thể phân loại LED 7 đoạn như sau:

+ Loại dương chung (Common Anode): nếu cực dương (anode) của tất cả 8 LED được nối với nhau và các cực âm (cathode) đứng riêng lẻ.

+ Loại âm chung (Common Cathode): nếu cực âm (cathode) của tất cả 8 LED được nối với nhau và các cực dương (anode) đứng riêng lẻ.

- Chân ngõ ra: LED 7 đoạn có 10 chân, trong đó 8 chân được nối với LED (A, B, C, D, E, F, G, và DP). Tùy vào loại LED 7 đoạn, hai chân giữa được đánh dấu COM hoặc dương chung hoặc âm chung của các LED.

- Ký hiệu logic và sơ đồ các chân vi mạch 74LS83 như sau:



Pin Names	Description
A0–A3	BCD Inputs
RBI	Ripple Blanking Input (Active LOW)
LT	Lamp Test Input (Active LOW)
BI/RBO	Blanking Input (Active LOW) or Ripple Blanking Output (Active LOW)
a–g	Segment Outputs (Active LOW) (Note 1)

Truth Table

Decimal or Function	Inputs							Outputs							Note
	LT	RBI	A3	A2	A1	A0	BI/RBO	a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	

Ghi chú:

- Chân  $\overline{BI}$  /  $\overline{RBO}$  được nối theo kiểu điểm AND bên trong IC và được dùng như ngã vào xóa (Blanking Input,  $\overline{BI}$ ) và/hoặc ngõ ra xóa dọn sóng (Ripple Blank Output,  $\overline{RBO}$ ). Ngõ vào  $\overline{BI}$  phải được để hở hay giữ ở mức cao khi cần thực hiện giải mã cho số ra. Ngõ vào xóa dọn sóng (Ripple Blank Input,  $\overline{RBI}$ ) phải để hở hay ở mức cao khi muốn đọc số 0.

- Khi đưa ngõ vào  $\overline{BI}$  xuống thấp, ngõ ra lên 1 (không tác động) bất chấp các ngõ vào còn lại. Ta nói IC làm việc dưới điều kiện bị ép buộc và đây là trường hợp duy nhất  $\overline{BI}$  giữ vai trò ngõ vào.

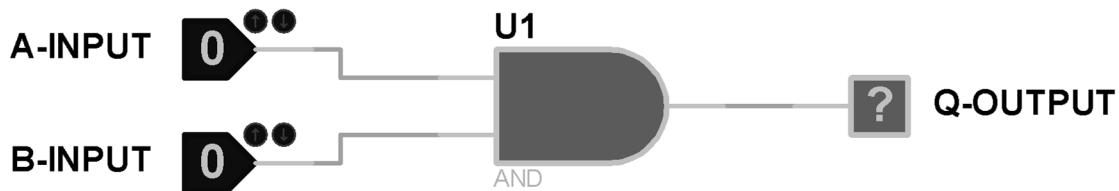
- Khi ngõ vào  $\overline{RBI}$  ở mức 0 và  $A0 = A1 = A2 = A3 = 0$ , tất cả các ngõ ra kể cả  $\overline{RBO}$  đều xuống 0. Ta nói IC làm việc dưới điều kiện đáp ứng.

- Khi  $\overline{BI}$  /  $\overline{RBO}$  để hở hay được giữ ở mức 1 và ngõ vào thử đèn (Lamp test,  $LT$ ) xuống 0, tất cả các led đều cháy (ngõ ra xuống 0).

### 3. NỘI DUNG THỰC HÀNH

#### 3.1. Khảo sát hoạt động của các công logic cơ bản

Sử dụng phần mềm Protues để khảo sát hoạt động của mạch AND theo sơ đồ như sau:



Sử dụng các phần tử: AND; LOGICPROBE (BIG); LOGICSTATE.

Tương tự, khảo sát của các công logic khác: OR, NAND, NOR, XOR, XNOR.

Ghi kết quả khảo sát vào bảng sau:

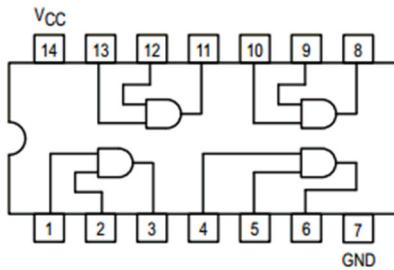
$A$	$B$	$Q = AB$	$Q = A + B$	$Q = \overline{AB}$	$Q = \overline{A} + \overline{B}$	$Q = A \oplus B$	$Q = A \sim B$
0	0						
0	1						
1	0						
1	1						

#### 3.2. Thiết kế mạch logic sử dụng các công logic cơ bản

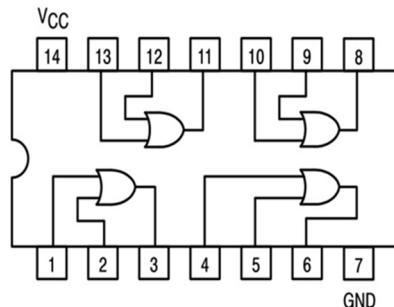
Thiết kế mạch logic tóm hợp bỏ phiếu với 3 đầu vào  $A, B, C$  và 1 đầu ra  $Y$ . Kết quả bỏ phiếu được thông qua khi có đa số phiếu được bỏ.

a. Thiết kế và mô phỏng hoạt động mạch logic mạch sử dụng cỗng logic cơ bản từ các vi mạch sau:

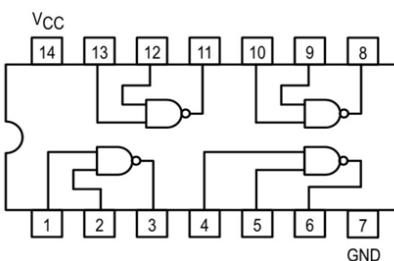
- 74LS08 (vi mạch AND 2 ngõ vào):



- 74LS32 (vi mạch OR 2 ngõ vào):

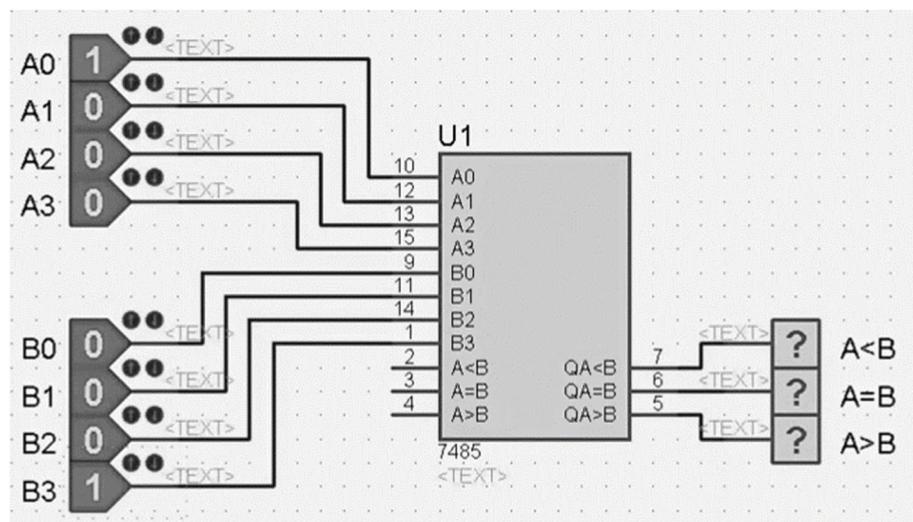


b. Thiết kế và mô phỏng hoạt động mạch logic chỉ sử dụng NAND 2 đầu vào từ vi mạch 74LS00:



### 3.3. Bộ so sánh

a. Thiết kế và mô phỏng hoạt động của mạch so sánh hai số nhị phân 4 bit sử dụng vi mạch 74LS85 với các toán hạng đầu vào so sánh là mã nhị phân của hai chữ số cuối cùng trong mã số sinh viên.



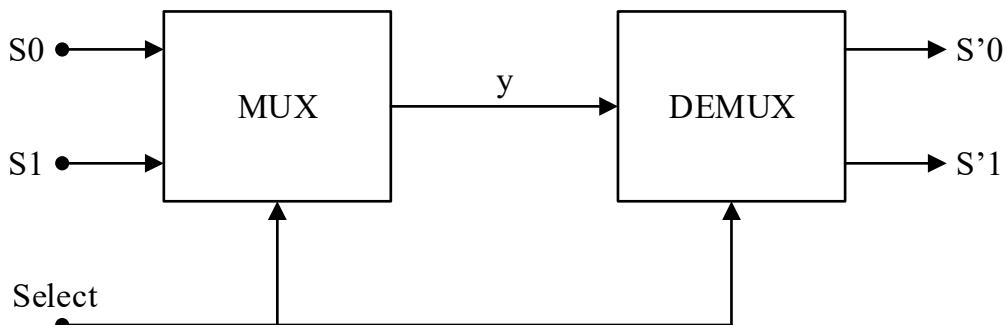
b. Thiết kế và mô phỏng hoạt động của mạch so sánh hai số nhị phân 8 bit trên cơ sở ghép nối các vi mạch 74LS85 với các toán hạng đầu vào so sánh là mã nhị phân của hai số đầu tiên và hai số cuối cùng trong mã số sinh viên.

### 3.4. Bộ cộng

- a. Mô phỏng hoạt động của các bộ cộng bán phần và bộ cộng toàn phần.
- b. Thiết kế và mô phỏng hoạt động của bộ cộng hai số nhị phân 4 bít sử dụng vi mạch 74LS83 với các toán hạng đầu vào là mã nhị phân của hai chữ số cuối cùng trong mã số sinh viên.
- c. Thiết kế và mô phỏng hoạt động của bộ cộng hai số nhị phân 8 bít sử dụng vi mạch 74LS83 với các toán hạng đầu vào là mã nhị phân của hai số đầu tiên và hai số cuối cùng trong mã số sinh viên.

### 3.5. Bộ hợp kênh và phân kênh

Thiết kế mạch kết hợp MUX - DEMUX với sơ đồ khối như sau:



trong đó: MUX và DEMUX là các bộ ghép kênh và phân kênh sao cho:

- + MUX : Khi  $Select = 0$  thì  $y = S0$ ;  
Khi  $Select = 1$  thì  $y = S1$ .
  - + DEMUX : tương tự có tín hiệu ra  $S'0 = S0$  và  $S'1 = S1$ .
- Thành lập bảng chân lý và phương trình logic của mạch.
  - Thực hiện mạch bằng các phần tử logic cơ bản.

### 3.6. Bộ giải mã

Thiết kế mạch giải mã BCD sang LED 7 đoạn sử dụng vi mạch 74LS47 trong đó số BCD đầu vào là hai chữ số cuối cùng trong mã số sinh viên, kết quả hiển thị lên 2 LED 7 đoạn.

## BÀI 2:

# MẠCH LOGIC TUẦN TỤ

### 1. MỤC ĐÍCH

Bài thực hành nhằm cung cấp cho sinh viên các kiến thức, kỹ năng về khảo sát, phân tích các đặc tính của các trigo số (Flip-Flop) và mạch logic tuần tự sử dụng phần mềm mô phỏng Protues.

### 2. TÓM TẮT LÝ THUYẾT

Mạch logic tuần tự (mạch dãy - Sequential Circuits): là các mạch có giá trị tín hiệu lối ra không chỉ phụ thuộc vào các giá trị đầu vào ở thời điểm hiện tại mà còn phụ thuộc vào các giá trị đầu vào ở trạng thái trước đó.

Mạch dãy là phần tử nhớ, điển hình là các trigo (Flip-Flop).

Nếu sự thay đổi trạng thái chỉ xảy ra khi có một tín hiệu tham khảo gọi là xung nhịp (clock) thì hệ được gọi là hệ đồng bộ và những hệ có trạng thái thay đổi không cần xung nhịp được gọi là hệ không đồng bộ.

#### 2.1. Các trigo số (Flip-Flop)

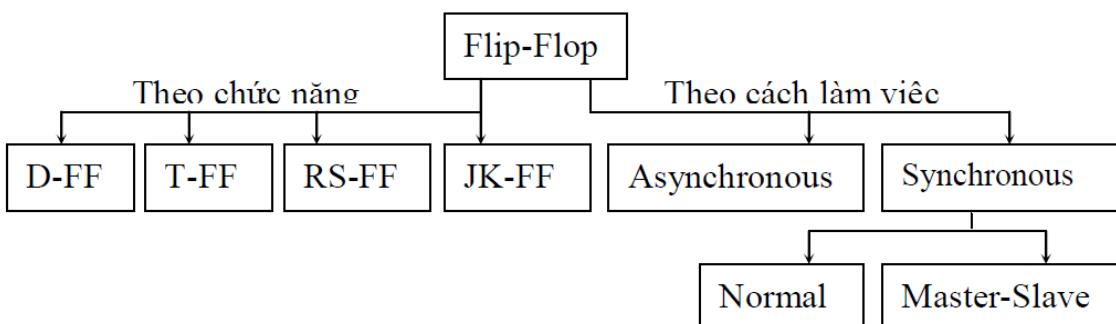
Trigo trong tiếng Anh gọi là Flip - Flop viết tắt là FF. Nó là một phần tử nhớ có hai trạng thái cân bằng ổn định tương ứng với 2 mức logic 0 và 1.

Dưới tác động của các tín hiệu điều khiển ở lối vào, trigo có thể chuyển về một trong hai trạng thái cân bằng, và giữ nguyên trạng thái đó chừng nào chưa có tín hiệu điều khiển làm thay đổi trạng thái của nó.

Trạng thái tiếp theo của trigo phụ thuộc không những vào tín hiệu ở lối vào mà còn phụ thuộc vào cả trạng thái đang hiện hành của nó.

Đang chạy, nếu ngừng các tín hiệu điều khiển ở lối vào nó vẫn có khả năng giữ trạng thái hiện hành của mình trong một thời gian dài, chừng nào mà nguồn điện nuôi mạch trigo không bị ngắt thì thông tin dưới dạng nhị phân lưu giữ trong trigo vẫn được duy trì. Như vậy, nó được sử dụng như một phần tử nhớ.

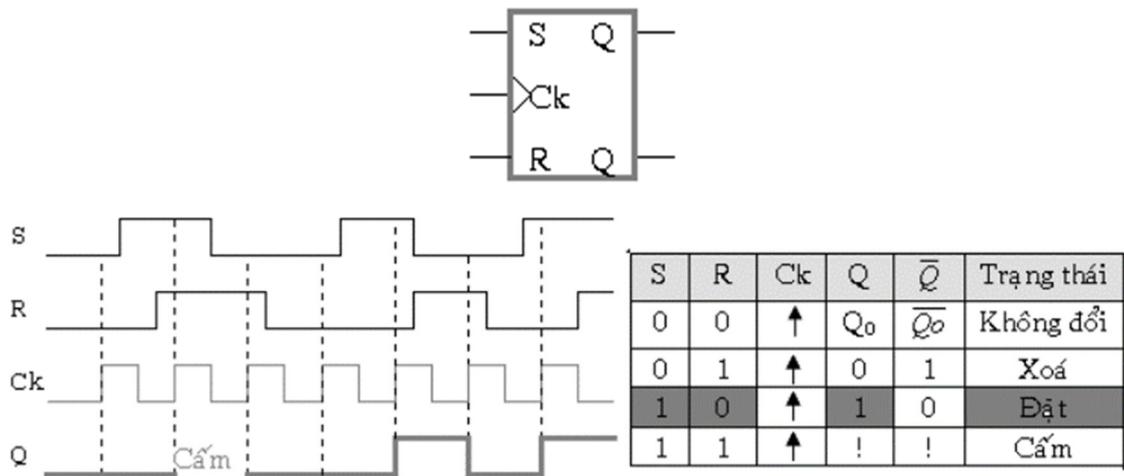
Trigo được cấu thành từ 1 nhóm các công logic, mặc dù công logic tự thân nó không có khả năng lưu trữ, nhưng có thể nối nhiều công với nhau theo cách thức cho phép lưu giữ được thông tin. Mỗi sự sắp xếp công khác nhau sẽ cho ra các trigo khác nhau.



### 2.1.1. Trigơ RS

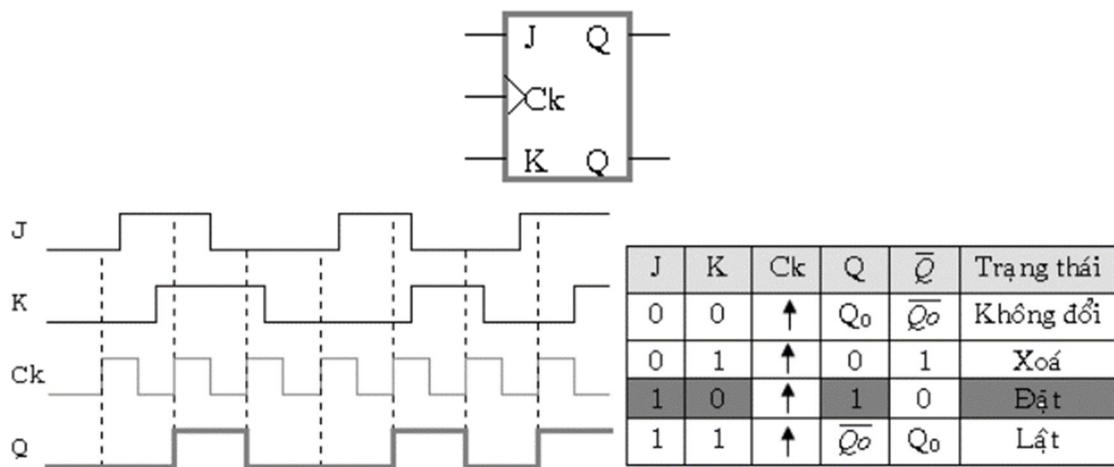
Trigơ RS (mạch lật đặt lại) là 1 trigơ có hai đầu vào điều khiển R, S; trong đó: S là đầu vào thiết lập “1” (Set) còn R là đầu vào xoá “0” (Reset).

$$\text{Phương trình đặc trưng: } Q_{n+1} = S + \bar{R}Q_n$$



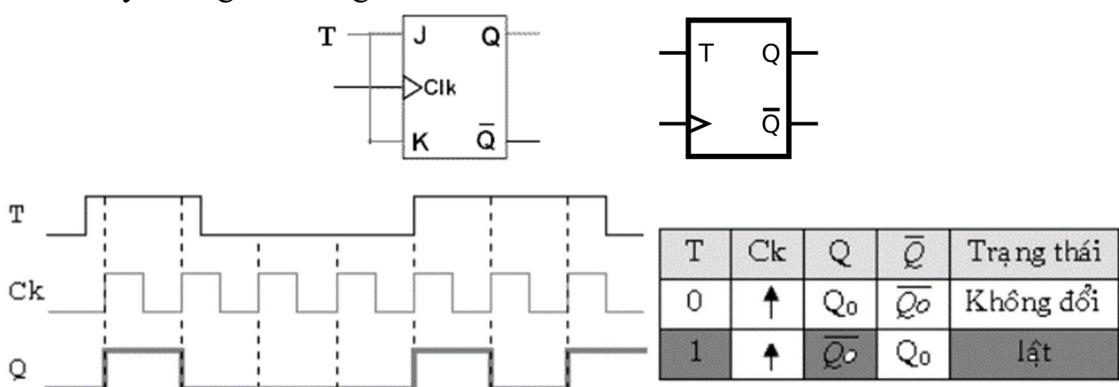
### 2.1.2. Trigơ JK

FF JK bổ sung thêm trạng thái cho FF RS (tránh trạng thái cấm).



### 2.1.3. Trigơ T:

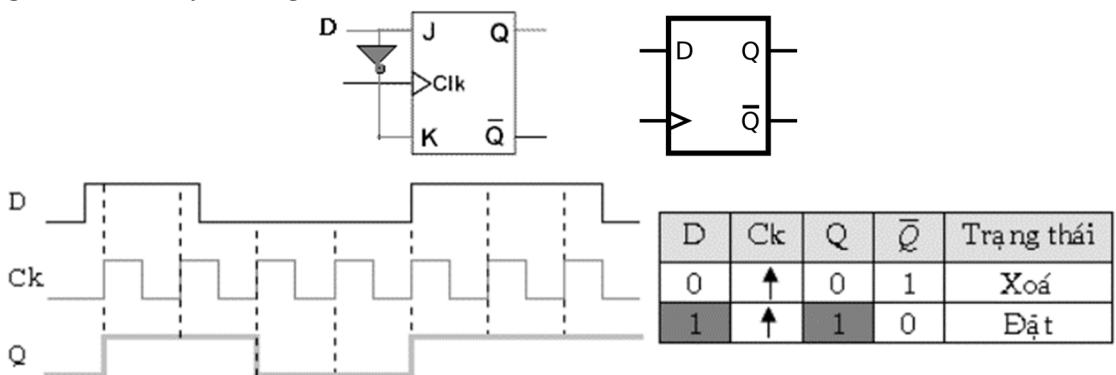
Khi nối chung 2 ngõ vào JK như hình dưới thì sẽ được FF T: chỉ có một ngõ vào T, ngõ ra sẽ bị lật lại trạng thái ban đầu khi ngõ T tác động và mỗi khi có cạnh sườn lên hay xuống của xung Ck.



#### 2.1.4. Trigor D:

Khi nối ngõ vào của FF RS hay JK như hình thì sẽ được FF D: chỉ có 1 ngõ vào gọi là ngõ vào data (dữ liệu) hay delay(trì hoãn).

Hoạt động của FF D rất đơn giản: ngõ ra sẽ theo ngõ vào mỗi khi xung Ck tác động cạnh lên hay xuống.



## 2.2. Bộ đếm

### 2.2.1. Khái niệm

Đếm là khả năng nhớ được số xung đầu vào; mạch điện thực hiện thao tác đếm được gọi là bộ đếm.

Bộ đếm là một thao tác cơ bản cực kỳ quan trọng và được sử dụng vô cùng rộng rãi, từ các thiết bị đo chỉ thị số đến các máy tính điện tử số loại lớn, bất kỳ hệ thống số hiện đại nào cũng đều hiện diện bộ đếm.

### 2.2.2. Phân loại bộ đếm:

- Căn cứ vào sự khác biệt của tình huống chuyển đổi trạng thái các trigor trong bộ đếm: Bộ đếm đồng bộ (bộ đếm song song) và bộ đếm không đồng bộ (bộ đếm nối tiếp).

- Căn cứ vào sự khác biệt về hệ số đếm của bộ đếm: Bộ đếm nhị phân, bộ đếm thập phân, ...

- Căn cứ vào tác động của xung đếm đầu vào mà số đếm của bộ đếm tăng hay giảm: bộ đếm thuận, bộ đếm nghịch và bộ đếm thuận nghịch.

### 2.2.3. Các vi mạch đếm thông dụng

- Các vi mạch đếm nhị phân 4 bit:

- + 74LS93 : mạch đếm 16 không đồng bộ mã nhị phân;

- + 74LS161 : mạch đếm 16 đồng bộ mã nhị phân;

- + 74LS191 : mạch đếm 16 đồng bộ có các mode điều khiển;

- + 74LS193 : mạch đếm nhị phân đồng bộ 4 bit tiến/lùi;

- Các vi mạch đếm 10 mã BCD:

- + 74LS90 : mạch đếm 10 không đồng bộ;

- + 74LS92 : mạch đếm 10 đồng bộ mã BCD có các mode điều khiển;

- + 74LS160 : mạch đếm 10 đồng bộ đặt trước, theo mã BCD;

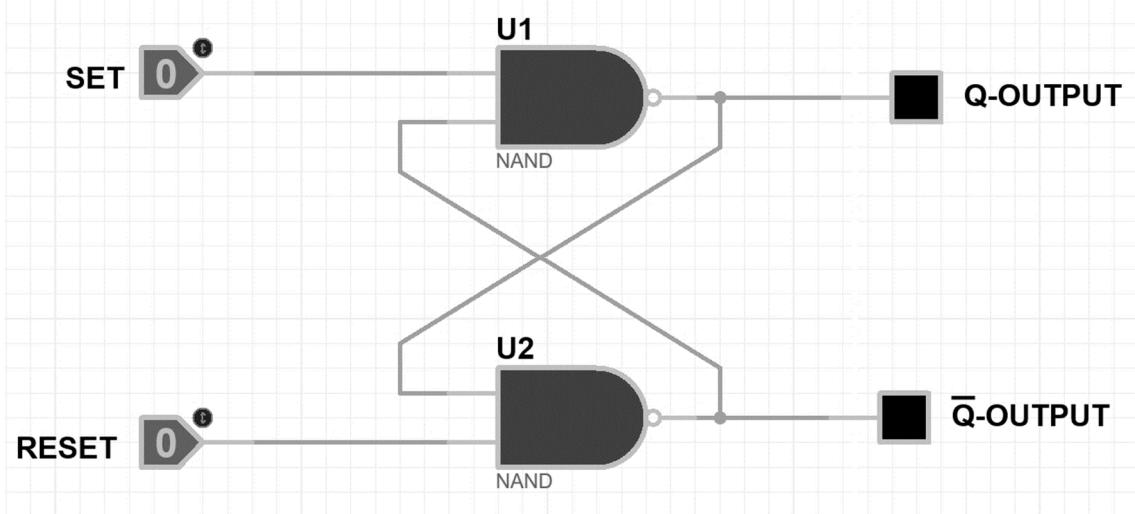
- + 74LS190 : mạch đếm 10 đồng bộ theo mã BCD.

### 3. NỘI DUNG THỰC HÀNH

#### 3.1. Khảo sát hoạt động của các trigger số

##### 3.1.1. Trigger RS

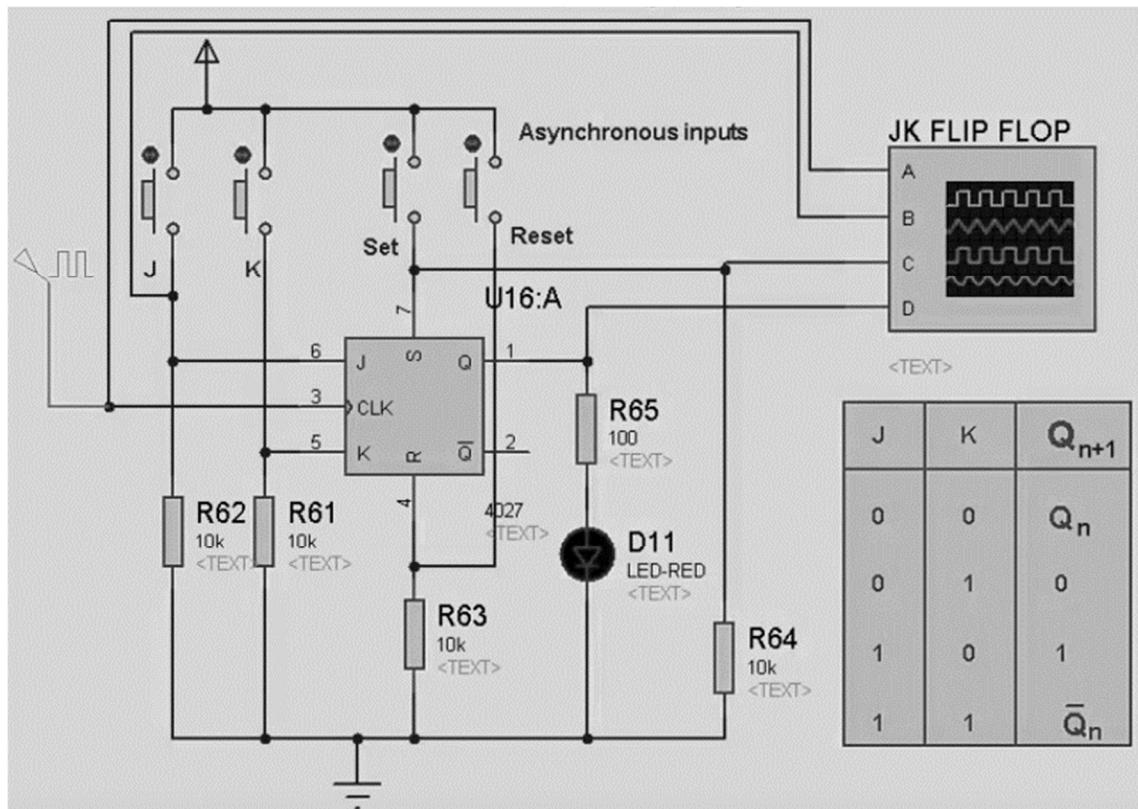
Sử dụng phần mềm Protues để khảo sát hoạt động của trigger RS theo sơ đồ sử dụng cổng NAND và sơ đồ sử dụng cổng NOR.



Lập bảng chân lý của sơ đồ mô phỏng trên.

##### 3.1.2. Trigger JK

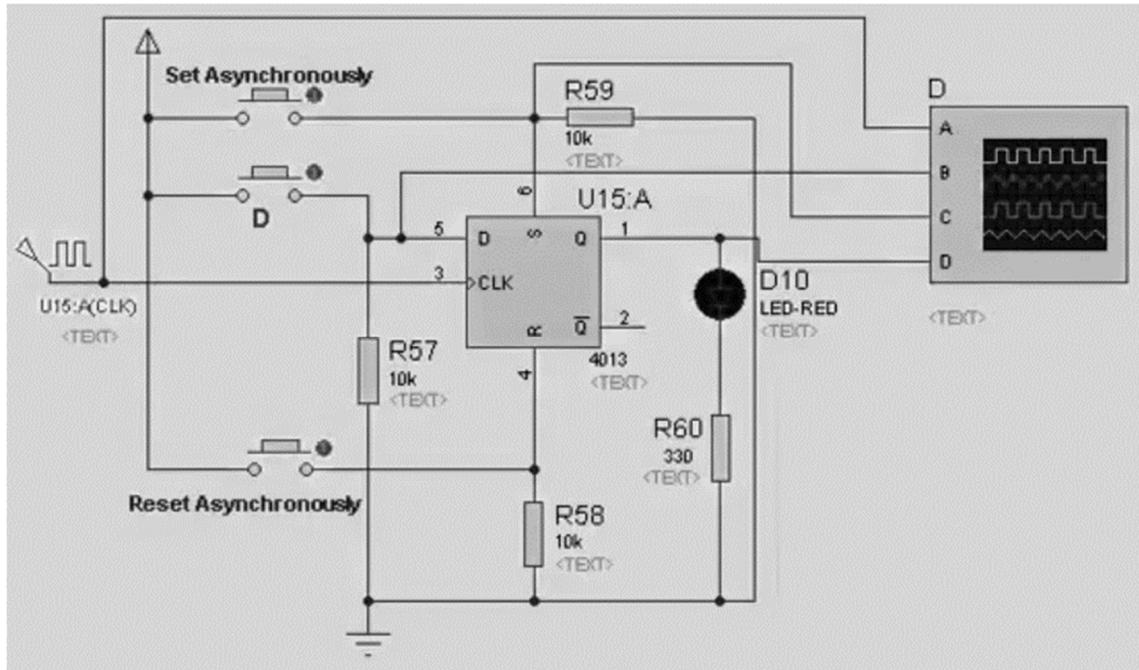
Sử dụng phần mềm Protues để khảo sát hoạt động của trigger JK theo sơ đồ như sau:



Sử dụng xung CLK với tần số 1 Hz. Kiểm tra hoạt động của sơ đồ mô phỏng và vẽ lại giản đồ xung nhịp của sơ đồ mô phỏng.

### 3.1.3. Trigor D

Sử dụng phần mềm Protues để khảo sát hoạt động của trigor D theo sơ đồ như sau:

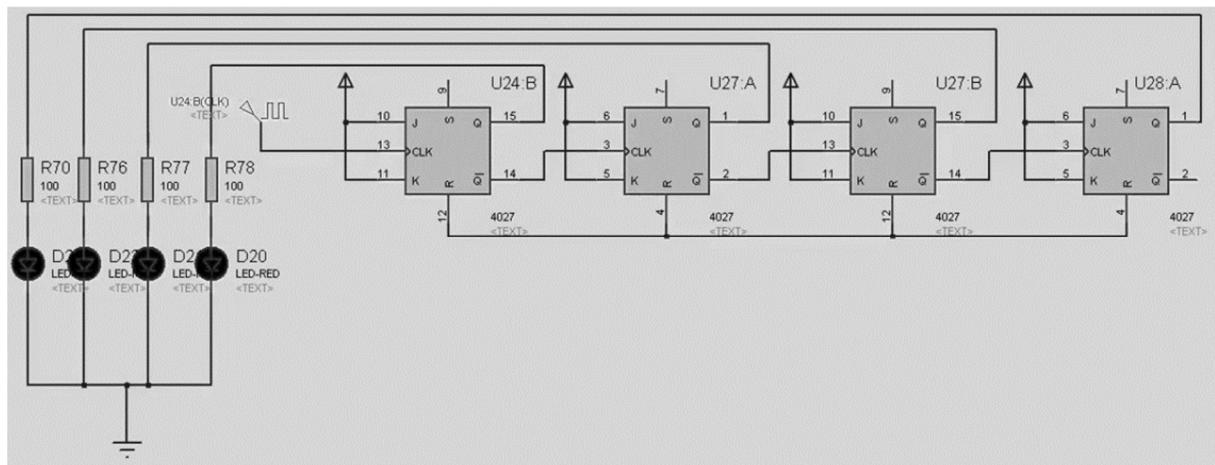


Sử dụng xung CLK với tần số 1 Hz. Kiểm tra hoạt động của sơ đồ mô phỏng và vẽ lại giản đồ xung nhịp của sơ đồ mô phỏng.

## 3.2. Khảo sát hoạt động của các bộ đếm

### 3.2.1. Khảo sát bộ đếm nhị phân không đồng bộ 4 bit

Sử dụng phần mềm Protues để khảo sát hoạt động của bộ đếm tiến nhị phân không đồng bộ 4 bit sử dụng trigor JK theo sơ đồ như sau:



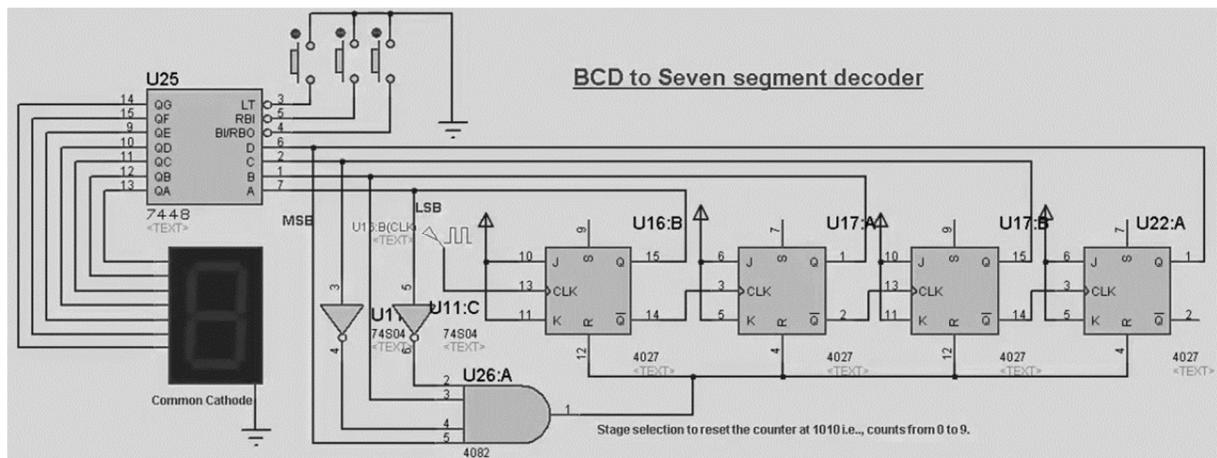
Sử dụng xung CLK với tần số 1 Hz.

Kiểm tra hoạt động của sơ đồ mô phỏng, xây dựng bảng trạng thái và vẽ lại giản đồ xung nhịp của sơ đồ mô phỏng.

*Trên cơ sở sơ đồ mô phỏng trên, hãy xây dựng sơ đồ của bộ đếm lùi nhị phân không đồng bộ 4 bit.*

### 3.2.2. Khảo sát bộ đếm theo mã BCD sử dụng trigger JK hiển thị lên LED 7 đoạn

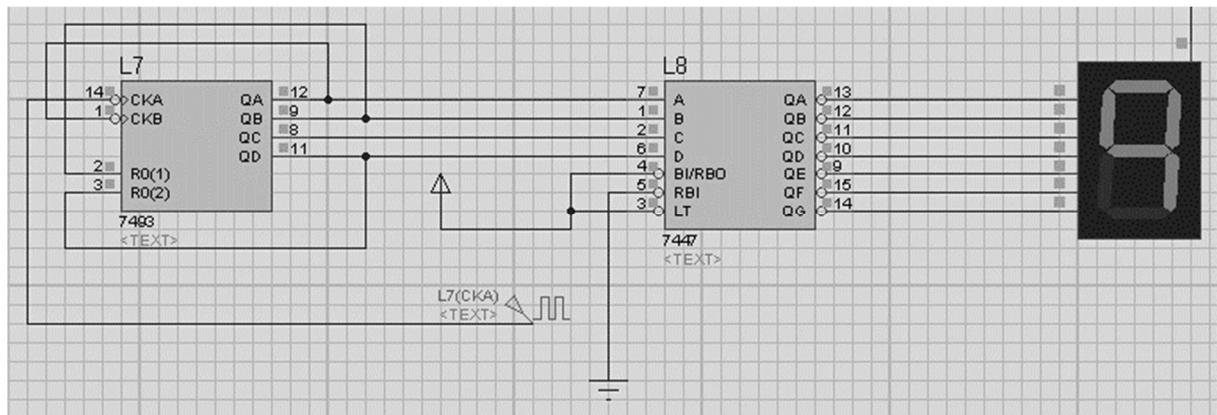
Sử dụng phần mềm Protues để khảo sát hoạt động của bộ đếm BCD sử dụng trigo JK theo sơ đồ như sau:



Sử dụng xung CLK với tần số 1 Hz.

Kiểm tra hoạt động của sơ đồ mô phỏng, xây dựng bảng trạng thái và vẽ lại giản đồ xung nhịp của sơ đồ mô phỏng.

3.2.2. Khảo sát bộ đếm theo mã BCD sử dụng vi mạch 74LS93 và hiển thị lên LED 7 đoạn



Sử dụng xung CLK với tần số 1 Hz. Kiểm tra hoạt động của sơ đồ mô phỏng, xây dựng bảng trạng thái và vẽ lại giản đồ xung nhịp của sơ đồ mô phỏng.

Do mạch chỉ sử dụng một LED 7 thanh nên chỉ có thể hiển thị được các giá trị từ 0 đến 9 mặc dù mạch có thể đếm từ 0 đến 15.

Ngoài việc đếm từ 0 đến 9 ta còn có thể điều khiển mạch đếm từ 0 đến 1 giá trị bất kì (trong khoảng 1 đến 9) bằng cách kết nối 2 chân R0(1) & R0(2) của 74LS93 tới các chân A, B, C, D hay V<sub>CC</sub> (vì khi cả R1 & R2 cùng bằng 1 thì mạch đếm sẽ được Reset). Ví dụ: để cho mạch chỉ đếm từ 0 đến 4 thì có nghĩa là khi đến 5 thì 2 chân R0(1) & R0(2) phải bằng 1. Ta thấy giá trị 5 tương ứng với: A = 1, B = 0, C = 1, D = 0 vì vậy ta chỉ cần cho R1 nối với A & R2 nối với C.

*Trên cơ sở đó hãy xây dựng sơ đồ mô phỏng bộ đếm từ 00 đến hai số cuối cùng trong mã số sinh viên và hiển thị lên 2 đèn LED 7 thanh.*

## BÀI 3:

# LẬP TRÌNH HỢP NGỮ CHO 8086

## 1. MỤC ĐÍCH

Bài thực hành nhằm cung cấp cho sinh viên các kiến thức về lập trình hợp ngữ cho vi xử lí INTEL 8086, kỹ năng lập trình hợp ngữ cho vi xử lí và sử dụng phần mềm EMU8086 để khảo sát vi xử lí INTEL 8086.

## 2. TÓM TẮT LÝ THUYẾT

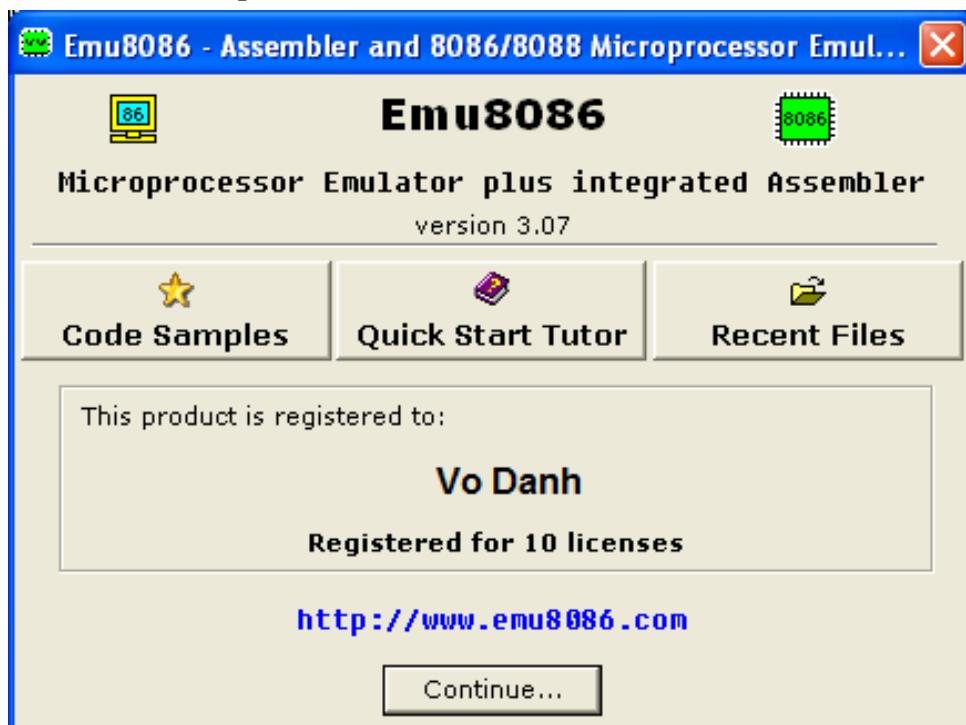
### 2.1. Hướng dẫn sử dụng EMU8086

EMU8086 là công cụ mạnh của người lập trình Hợp ngữ. Bao gồm nhiều chức năng như: thực thi chương trình dạng EXE (EXE Template), thực thi chương trình dạng COM (COM Template, Thực thi đoạn lệnh hợp ngữ (BIN Template), tạo đoạn Boot máy (BOOT Template).

#### 2.1.1. Khởi động Emu8086

Nhấp đúp biểu tượng trên desktop, màn hình khởi động như hình 1 xuất hiện.

- Code Samples : Chọn file chương trình mẫu để thực hiện.
- Quick Start Tutor: Truy cập trang web hướng dẫn (phải có kết nối Internet).
- Recent Files: Chọn file trong danh sách file thường dùng.
- Continue ...: Tiếp tục vào màn hình làm việc.



Hình 1: Màn hình khởi động Emu8086

Khi bấm Continue, màn hình làm việc xuất hiện như hình 2 với file chương trình mẫu “Hello World” mặc nhiên xuất hiện trong vùng soạn thảo.

Hình 3 là công cụ Number Convertor (Bấm vào nút Convertor trên thanh công cụ) rất hữu dụng khi muốn chuyển đổi giá trị giữa các hệ thống số với nhau.

The screenshot shows the Emu8086 interface with the assembly code for a 'Hello World' sample. The code includes directives like #make\_COM#, ORG 100H, and JMP START, followed by data messages in DB. A red box highlights the 'New' button in the toolbar, and another red box highlights the 'Convertor' button.

```

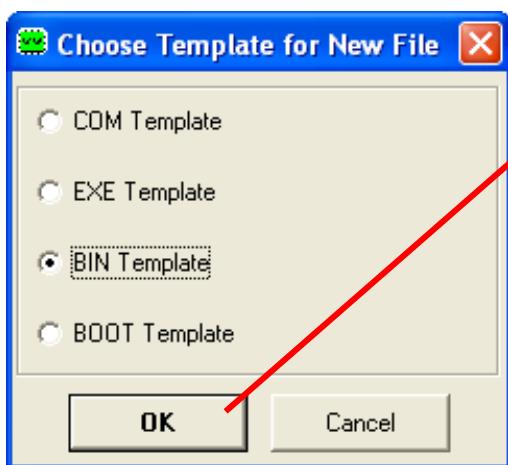
01 ; Hello World Sample!
02
03 ; Standard header:
04     #make_COM#
05     ORG 100H
06
07 ; Jump to start:
08     JMP START
09
10 ; Data:
11 msg DB 13, 10, 'Hello, World!', 13, 10
12     DB 'Thank you for choosing Emu8086.', 13, 10
13     DB 'The best Microprocessor Emulator!', 13, 10

```

Hình 2: Màn hình Emu8086



Hình 3: Chức năng Convertor



Hình 4: Chọn template

The screenshot shows the Emu8086 interface with the BIN Template code. It starts with #make\_BIN# and includes directives for segment and offset, and register values AX=1234#, BX=0000#, CX=0000#, DX=0000#.

```

#make_BIN#
; where to load?
; (usually the same values
; should be set to CS:IP)
#LOAD_SEGMENT=1234#
#LOAD_OFFSET=0000#
; set these values to registers on
#AX=1234#
#BX=0000#
#CX=0000#
#DX=0000#

```

Hình 5: BIN Template

### 2.1.2. Soạn thảo lệnh hợp ngữ để khảo sát

Để mở vùng làm việc mới chọn NEW, xuất hiện hình 4 để chọn Template

Để khảo sát lệnh Intel-8086 thì chọn chức năng thực thi lệnh (BIN Template).

Vùng làm việc BIN Template xuất hiện như hình 5.

Trong BIN Template, quan trọng nhất là dòng đầu tiên #make\_bin# dùng để xác định chế độ dịch lệnh của Emu8086. Tuyệt đối không được thay đổi dòng lệnh giả này.

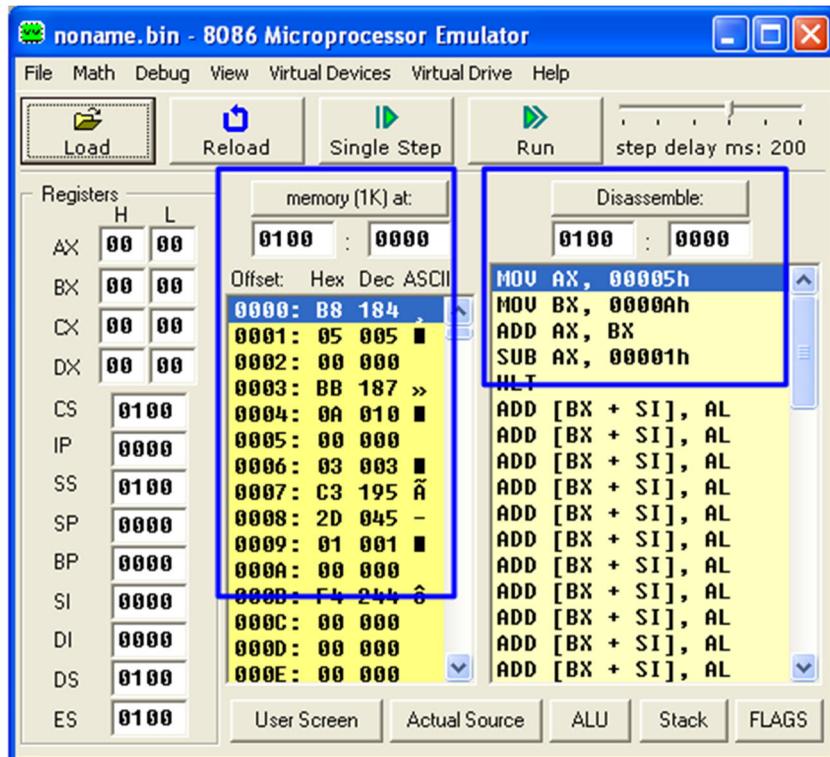
Các dòng còn lại dùng để khởi động các giá trị ban đầu cho các thanh ghi và thông số địa chỉ segment/offset cho chương trình. Các dòng này không quan trọng, có thể xóa bỏ được hoặc thay đổi giá trị khởi động khác. Khi các dòng này bị xóa bỏ thì các thông số và thanh ghi sẽ được khởi động theo giá trị mặc định. Để đơn giản, nên xóa bỏ từ dòng 2 đến hết.

### 2.1.3. Khảo sát lệnh (Giả lập - Emulate)

Để khảo sát lệnh bằng cách giả lập, chọn Emulate, khi đó màn hình giả lập xuất hiện như hình 7. Màn hình giả lập gồm 3 vùng: Thanh ghi (Registers), Bộ nhớ 1KB

(Memory) chứa mã máy nhị phân và vùng hiển thị lệnh hợp ngữ tương ứng với mã máy nhị phân (Disassemble).

Giá trị các thanh ghi được trình bày ở dạng số Hex. Vùng bộ nhớ trình bày Hex – Dec – ASCII đối với từng ô nhớ (địa chỉ offset).



Hình 7. Màn hình giả lập

- Reload: Nạp lại đoạn lệnh
- Run: Chạy cả đoạn lệnh từ đầu cho đến khi gặp lệnh HLT (dừng)
- Single Step: Mỗi khi Single Step được bấm thì CPU chỉ chạy 1 lệnh hiện hành duy nhất (xác định bằng vệt sáng màu xanh) và dừng lại chờ cho đến khi Single Step được bấm tiếp. Như vậy, việc khảo sát lệnh có thể thực hiện thông qua Single Step.

Các thành phần khác còn có thể xem được trạng thái khi CPU thực hiện lệnh trong chế độ giả lập như ALU, Stack và FLAGS (thanh ghi Cờ) bằng cách bấm vào các nút tương ứng

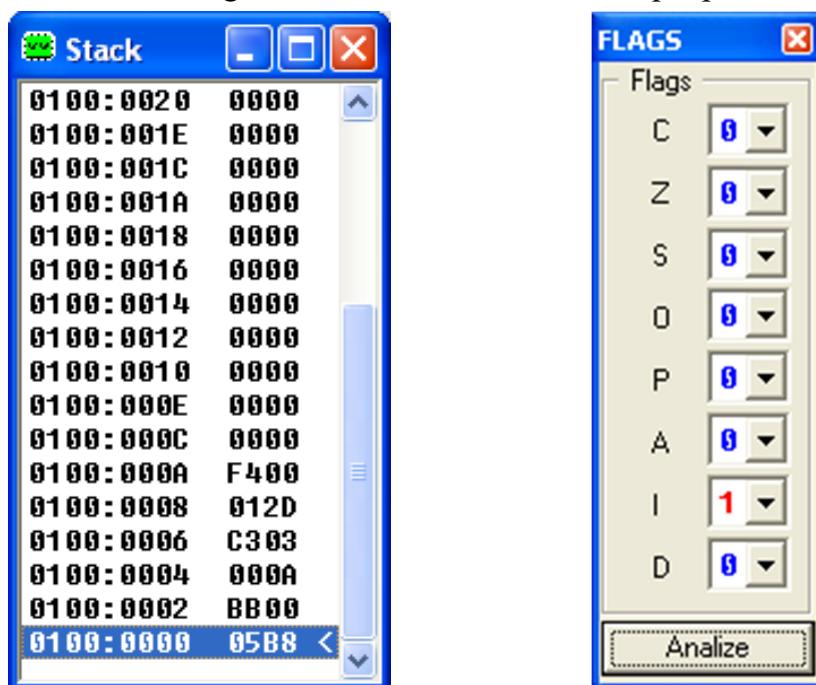
- Hình 8 cho biết trạng thái ALU khi thực hiện các phép toán (giá trị ở dạng nhị phân 16 bit). Dòng đầu tiên là thứ tự bit, dòng thứ 2 là giá trị toán hạng nguồn 1, dòng thứ 3 là giá trị toán hạng nguồn 2 và dòng cuối là giá trị kết quả sau khi thực hiện phép toán.

The screenshot shows the ALU status window. It displays a 16-bit binary state table with columns labeled F through 0. The first row shows the bit positions, and subsequent rows show the state of each bit for different operations.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Hình 8: Trạng thái ALU

- Hình 9 trình bày nội dung ngăn xếp ở dạng Hex 2 byte
- Hình 10 thể hiện trạng thái các cờ sau khi thực hiện phép toán



Hình 9: Stack

Hình 10: Thanh ghi cờ

#### 2.1.4. Thực thi chương trình dạng EXE hay COM

Emu8086 có thể thực thi chương trình Hợp ngữ viết theo cấu trúc dạng EXE hay COM. Khi đó trong vùng soạn thảo, không có dòng lệnh giả #make\_bin# và soạn thảo chương trình theo đúng cấu trúc dạng chương trình tương ứng.

Ví dụ: Chương trình dạng EXE như hình 11

```

C:\Program Files\Emu8086v3.07\Samples\...
File Edit Bookmarks Macro Compile Emulator Math Help
New Open Samples Save Compile Emulate
01 CSEG SEGMENT
02     assume CS: CSEG
03 begin: MOU AX, 5
04     MOU BX, 10
05     ADD AX, BX
06     SUB AX, 1
07     MOU AH, 4Ch
08     INT 21h
09 CSEG ENDS
10     END begin
11

```

The screenshot shows the Emu8086 interface with the assembly code for a program named 'begin'. The code consists of several instructions: MOU AX, 5; MOU BX, 10; ADD AX, BX; SUB AX, 1; MOU AH, 4Ch; and INT 21h. The code is enclosed in a CSEG segment definition. The assembly window has a toolbar with New, Open, Samples, Save, Compile, and Emulate buttons. The status bar at the bottom shows the number 11.

Hình 11: Ví dụ chương trình dạng EXE

## 2.2. Cấu trúc chương trình hợp ngữ dạng EXE

```
.model small
.stack 100h
.data
;Khai bao du lieu
.code
main proc
mov ax,@data
mov ds,ax
;Chuong trinh hop ngu
mov ah,4Ch
int 21h
main endp
;----- Cac chuong trinh con ---
Ctcl proc
; Ma lenh cua chuong trinh con
ret
Ctcl endp
;-----
End
```

## 2.3. 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 0Bh: kiểm tra phím nhấn trên bàn phím*

Gọi: AH = 0Bh

Trả về: AL = 0FFh nếu có nhấn phím, AL = 0 nếu không nhấn phím

- *Hàm 4Ch: kết thúc chương trình*

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

## 2.4. Ngắt 10H

- *Hàm 02h: Gọi AH = 02h, DH = dòng, DL = cột*

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

## 2.5. Lệnh so sánh

Cú pháp: CMP Trái, Phải ; Cờ  $\leftarrow$  Trái - Phải

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

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

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

Trái, Phải: Immed, Reg, Mem

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

Ví dụ: so sánh hai số nguyên dương

```
MOV AH, 1      ; AH ← 1
MOV AL, 2      ; AL ← 2
CMP AH, AL     ; CF ← 1, ZF ← 0 vì AH < AL
```

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

## 2.6. Lệnh nhảy không điều kiện

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

Có các trường hợp sau:

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

Ví dụ:

JMP SHORT Calculate

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

Ví dụ:

JMP Calculate

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

Ví dụ:

JMP FAR PTR Calculate

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

- JMP <con trỏ 4 byte> ; (far indirect jump). Tương tự trường hợp trên, nhưng con trỏ gồm cả segment và offset. Chỉ khác ở khai báo con trỏ

- JMP <thanh ghi 2 byte> ; (indirect jump via regs). Nhảy đến địa chỉ lưu trong thanh ghi AX.

## 2.7. Lệnh nhảy có điều kiện

Cú pháp: J<điều kiện> <Label>

Các lệnh nhảy có điều kiện bắt đầu bằng chữ J sau đó là các chữ cái biểu thị

điều kiện (ví dụ JGE: Jump if Greater than or Equal, nhảy nếu lớn hơn hay bằng), tiếp sau là một tên nhãn.

Điều kiện để lệnh nhảy xem xét khi thi hành là giá trị các cờ được tạo ra từ lệnh CMP hay TEST. Khi sử dụng lệnh nhảy có điều kiện sau khi thực hiện phép so sánh, phải đặc biệt lưu ý toán hạng trong phép so sánh là số có dấu (signed) hay không có dấu (unsigned) để lựa chọn lệnh cho phù hợp.

Một số lệnh nhảy có điều kiện thường dùng:

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

## 2.5. Lệnh lặp

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

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

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

Ví dụ: Nhập mảng A gồm 10 ký tự

```

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

```

### 3. NỘI DUNG THỰC HÀNH

#### 3.1. Các lệnh cơ bản

3.1.1: Thực hiện chương trình sau (cộng 49h với 8Ah):

```
.model small
.stack 100h
.data
msg db 'Hello$'
.code
main proc
mov ax,@data
mov ds,ax
mov al,49h
add al,8Ah
mov ah,4Ch
int 21h
main endp
;-----
End
```

- Nhấn vào nút Emulate để thực hiện mô phỏng.
- Tại cửa sổ mô phỏng, chọn menu View > Flags để hiển thị nội dung các cờ.
- Nhấn nút Run thực thi chương trình và quan sát nội dung các cờ. Giải thích.

3.1.2: Thực hiện chương trình cộng 2 số và kiểm tra nội dung các cờ: CF, ZF, SF, OF, PF, AF. Từ đó rút ra kết luận về mục đích của các cờ này.

- 0FFh + 01h
- 0FFh + 10h
- 40h + 55h
- 22h + 8Fh

#### 3.2. Sử dụng ngắt 21h và ngắt 10h

3.2.1. Dùng hàm 09h xuất chuỗi ra màn hình:

```
.model small
.stack 100h
.data
msg db 'Hello$'
.code
main proc
mov ax,@data
```

```

    mov ds,ax
    mov ah,09h ; Xuất chuỗi ra màn hình

    lea dx,msg
    int 21h
    exit:
    mov ah,4Ch
    int 21h
    main endp
; -----
End

```

3.2.2. Bỏ dấu \$ ở cuối chuỗi Hello, thực hiện lại chương trình và nhận xét kết quả.

3.2.3. Thực hiện giống như bài 1.12 nhưng thực hiện liên tục cho đến khi nhấn một phím bất kỳ trên bàn phím thì dừng.

Gợi ý: dùng hàm 0Bh để kiểm tra phím nhấn, nếu có nhấn phím thì kết thúc chương trình.

```

    mov ah,0Bh
    int 21h
    cmp al,00h
    jne exit
    ; Xuất chuỗi
    ...
exit:

```

3.2.4. Xuất chuỗi 'Hello' ra màn hình tại hàng 10, cột 10. Gợi ý: dùng hàm 02h của ngắt 10h chuyển toạ độ trước khi xuất chuỗi.

3.2.5. Nhập một ký tự từ bàn phím và xuất ký tự vừa nhập ra màn hình tại hàng 11, cột 10.

```

.model small
.stack 100h
.data
msg db 'Hello$'
.code
main proc
    mov ax,@data
    mov ds,ax
    mov ah,08h      ; Nhập ký tự từ bàn phím
    int 21h
    push ax        ; Lưu ký tự vừa nhập
    mov ah,02h      ; Chuyển toạ độ con trỏ

```

```

    mov dx,0B0Ah
    int 10h

    pop ax
    mov dl,al          ; Xuất ký tự
    mov ah,02h
    int 21h
    mov ah,4Ch
    int 21h
    main endp
;-----
End

```

**3.2.6. Nhập một ký tự từ bàn phím và xuất ra màn hình ở dạng chữ hoa. Gợi ý: thêm một đoạn chương trình kiểm tra ký tự nhập, nếu là ký tự chữ thường thì chuyển thành chữ hoa rồi xuất ta màn hình.**

```

    cmp al,'a'          ; Nếu < 'a'
    jb next             ; hay > 'z' thì không phải là
    cmp al,'z'          ; chữ thường
    ja next
    sub al,20h          ; Chuyển từ chữ thường -> hoa
next:

```

**3.2.7. Nhập liên tục các ký tự, xuất ra màn hình ở dạng chữ hoa và kết thúc chương trình khi nhấn phím ESC. Gợi ý: sau khi nhập thì kiểm tra ký tự vừa nhập, nếu là ESC thì thoát (mã ASCII của phím ESC là 27 hay 1Bh).**

```

    cmp al,27
    je exit
    ...
exit:
    mov ah,4Ch
    int 21h

```

**3.2.8. Dùng hàm 02h của ngắt 21h để xuất chuỗi:**

```

.model small
.stack 100h
.data
msg db 'Hello$'
.code
main proc

```

```

mov ax,data
mov ds,ax


---


mov si,0
lap:
mov dl,msg[si]
cmp dl,'$'
je exit
mov ah,02h
int 21h
inc si
jmp lap
exit:
mov ah,4Ch
int 21h
main endp
end main

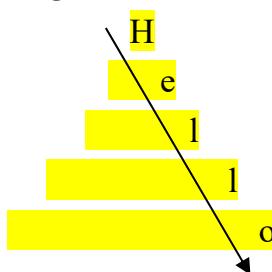
```

*3.2.9. Dùng hàm 02h của ngắt 21h để xuất chuỗi 'Hello' theo chiều thẳng đứng bắt đầu tại hàng 1, cột 10. Gợi ý: dùng hàm 02h của ngắt 10h để di chuyển toạ độ con trỏ sau khi xuất một ký tự.*



H  
e  
l  
l  
o

*3.2.10. Xuất chuỗi 'Hello' theo đường chéo như hình vẽ bắt đầu tại hàng 01 và cột 01.*



H  
e  
l  
l  
o

### 3.3. Cấu trúc rẽ nhánh

Chương trình sau đây nhận 1 ký tự. Nếu là ký tự HOA thì in ra màn hình "Ky tu HOA". Ngược lại in ra câu "Ky tu thuong". (Mã ASCII của ký tự HOA <= 'Z').

```

DSEG SEGMENT
tbaol DB " : ky tu HOA.$"
tbaot DB " : ky tu thuong.$"

```

```

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

```

- Dịch và chạy chương trình ở những trường hợp khác nhau để xem kết quả trên màn hình.
- Vẽ lưu đồ điều khiển của chương trình.
- Tại sao cần phải có lệnh JMP EXIT? Nếu không có lệnh ấy thì chương trình thực hiện như thế nào? Chạy lại chương trình để kiểm chứng.
  - Thay lệnh JA NHAN bằng lệnh JNA NHAN. Sửa chương trình sao cho kết quả không thay đổi.
  - Khi ký tự nhập vào không phải là chữ cái thì kết quả in ra màn hình là gì? Tại sao?

### 3.4. Cấu trúc vòng lặp

- Xem chương trình in ra màn hình lần lượt các ký tự từ A đến Z được viết như sau.
  - Dịch và chạy chương trình để xem kết quả trên màn hình.
  - Vòng lặp trong chương trình bao gồm đoạn lệnh nào? Viết theo kiểu while do hay repeat ... until hay for? Vẽ lưu đồ chương trình.
  - Sửa chương trình để in ra màn hình lần lượt các ký tự từ 'Z' đến 'A'.

- Tiếp tục sửa chương trình sao cho giữa các ký tự có 1 dấu cách trống;
- Dùng lệnh LOOP để viết lại chương trình trên theo cấu trúc vòng lặp for.

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

### 3.5. Bài tập thực hành

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

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

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

## BÀI 4:

# LẬP TRÌNH HỢP NGỮ CHO VI ĐIỀU KHIỂN 8051

## 1. MỤC ĐÍCH

Bài thực hành nhằm cung cấp cho sinh viên các kỹ năng:

- Những hiểu biết cơ bản về vi điều khiển 8051; khảo sát hoạt động các hoạt động cơ bản của vi điều khiển 8051;
- Xây dựng những ứng dụng phần cứng cơ bản sử dụng vi điều khiển 8051;
- Sử dụng phần mềm Reads51 để lập trình ngôn ngữ lập trình Assembly cho vi điều khiển.

## 2. TÓM TẮT LÝ THUYẾT

### 2.1. Tóm tắt về lịch sử của vi điều khiển 8051

Vào năm 1981 hãng Intel giới thiệu một số bộ vi điều khiển được gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4K byte ROM trên chip, hai bộ định thời, một cổng nối tiếp và 4 cổng (đều rộng 8 bit) vào ra tất cả được đặt trên một chip. Lúc ấy nó được coi là một “hệ thống trên chip”. 8051 là một bộ xử lý 8 bit có nghĩa là CPU chỉ có thể làm việc với 8 bit dữ liệu tại một thời điểm. Dữ liệu lớn hơn 8 bit được chia ra thành các dữ liệu 8 bit để cho xử lý. 8051 có tất cả 4 cổng vào - ra I/O mỗi cổng rộng 8 bit. Mặc dù 8051 có thể có một ROM trên chip cực đại là 64 K byte, nhưng các nhà sản xuất lúc đó đã cho xuất xưởng chỉ với 4K byte ROM trên chip.

Vi điều khiển 8051 đã trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác sản xuất và bán bất kỳ dạng biến thể nào của 8051 mà họ thích với điều kiện họ phải để mã lại tương thích với 8051. Điều này dẫn đến sự ra đời nhiều phiên bản của 8051 với các tốc độ khác nhau và dung lượng ROM trên chip khác nhau được bán bởi hơn nửa các nhà sản xuất. Điều này quan trọng là mặc dù có nhiều biến thể khác nhau của 8051 về tốc độ và dung lượng nhớ ROM trên chip, nhưng tất cả chúng đều tương thích với 8051 ban đầu về các lệnh. Điều này có nghĩa là nếu ta viết chương trình của mình cho một phiên bản nào đó thì nó cũng sẽ chạy với mọi phiên bản bất kỳ khác mà không phân biệt nó từ hãng sản xuất nào.

### 2.2. Bộ vi điều khiển 8051

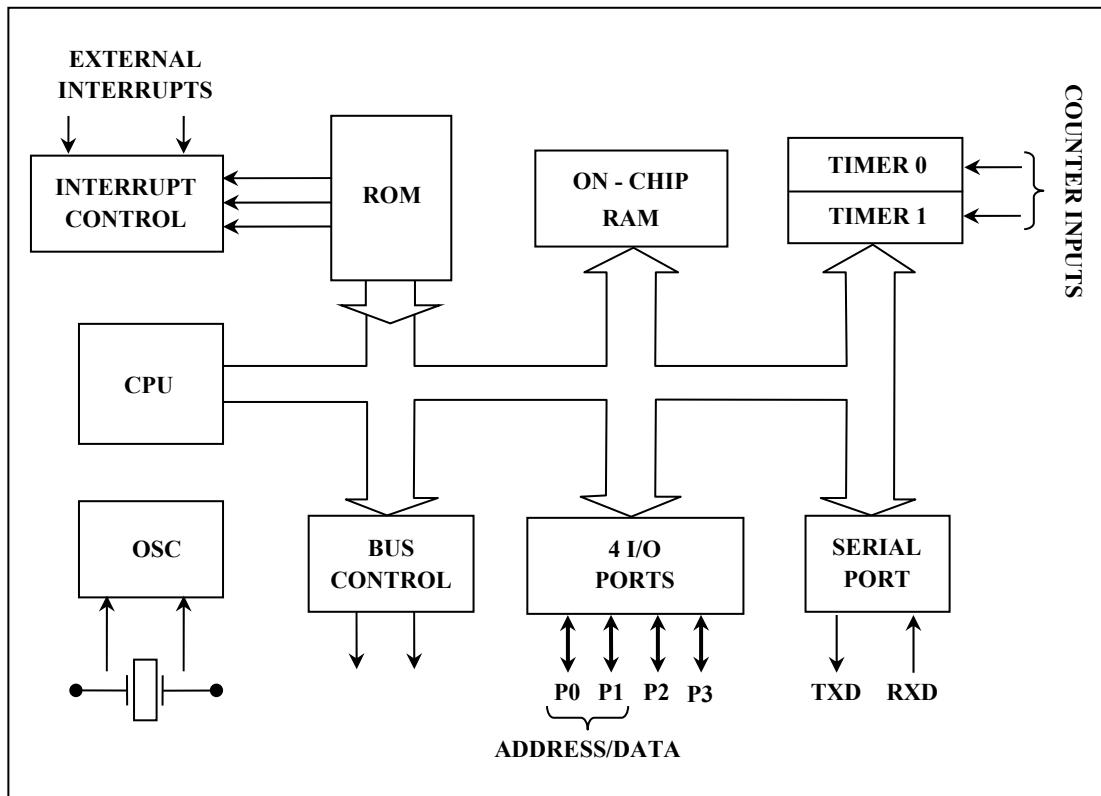
Bộ vi điều khiển 8051 có kiến trúc Harvard là thành viên đầu tiên của họ 8051. Hãng Intel ký hiệu nó là MCS 51.

Cấu trúc bên trong và các đặc tính của 8051 như hình 1.

### 2.3. Bộ vi điều khiển AT89C51

Đặc điểm và chức năng hoạt động của các IC họ MSC-51 hoàn toàn tương tự như nhau. Ở đây giới thiệu AT89C51 là một bộ vi điều khiển do hãng ATMEL sản

xuất với công nghệ CMOS có tốc độ cao vào công suất thấp với bộ nhớ Flash có thể lập trình được. Tương thích với chuẩn công nghiệp của 8051 và 8052 về chân ra và tập lệnh.



Hình 1.1:Sơ đồ khái niệm 8051.

Đặc tính	Số lượng
ROM trên chip	4K byte
RAM	128 byte
Bộ định thời	2
Các chân Vào - Ra	32
Cổng nối tiếp	1
Nguồn ngắn	6

Bảng 1.1 :Các đặc tính của 8051 đầu tiên.

### 2.3.1. Các đặc trưng của AT89C51

- 4 KB EPROM bên trong.
- 128 Byte RAM nội.
- 4 Port xuất /nhập I/O 8 bit.
- Giao tiếp nối tiếp.
- 64 KB vùng nhớ mã ngoài

- 64 KB vùng nhớ dữ liệu ngoại.
- Xử lý Boolean (hoạt động trên bit đơn).
- 210 vị trí nhớ có thể định vị bit.
- 4 $\mu$ s cho hoạt động nhân hoặc chia.

### 2.3.2. Sơ đồ và chức năng các chân của AT8C951



1	P1.0	VCC	40
2	P1.1	P0.0	39
3	P1.2	P0.1	38
4	P1.3	P0.2	37
5	P1.4	P0.3	36
6	P1.5	P0.4	35
7	P1.6	P0.5	34
8	P1.7	P0.6	33
9	RESET	P0.7	32
18	XTAL1	EA/VP	31
19	XTAL2	ALE/P	30
10	RXD	PSEN	29
11	TXD	P2.7	28
12	INT0	P2.6	27
13	INT1	P2.5	26
14	T0	P2.4	25
15	T1	P2.3	24
16	WR	P2.2	23
17	RD	P2.1	22
20	GND	P2.0	21

Hình 1.2: Pin Configurations.

AT89C51 có tất cả 40 chân có chức năng như các đường xuất nhập. Trong đó có 24 chân có tác dụng kép (có nghĩa là 1 chân có 2 chức năng), mỗi đường có thể hoạt động như đường xuất nhập hoặc như đường điều khiển hoặc là thành phần của các bus dữ liệu và bus địa chỉ.

#### a. Các cổng xuất nhập:

- *Port 0*: Port 0 là port có 2 chức năng ở các chân 32 - 39 của 8951. Trong các thiết kế cỡ nhỏ không dùng bộ nhớ mở rộng nó có chức năng như các đường I/O. Đối với các thiết kế cỡ lớn có bộ nhớ mở rộng, nó được kết hợp giữa bus địa chỉ và bus dữ liệu.

- *Port 1*: Port 1 là port I/O trên các chân 1-8. Các chân được ký hiệu P1.0, P1.1, P1.2, ... p1.7 có thể dùng cho giao tiếp với các thiết bị ngoài nếu cần. Port 1 không có chức năng khác, vì vậy chúng chỉ được dùng cho giao tiếp với các thiết bị bên ngoài.

- *Port 2*: Port 2 là 1 port có tác dụng kép trên các chân 21- 28 được dùng như các đường xuất nhập hoặc là byte cao của bus địa chỉ đối với các thiết bị dùng bộ nhớ mở rộng.

- *Port 3*: Port 3 là port có tác dụng kép trên các chân 10-17. Các chân của port này có nhiều chức năng, các công dụng chuyển đổi có liên hệ với các đặc tính đặc biệt của 8951 như ở bảng sau:

Bit	Tên	Chức năng chuyển đổi
P3.0	RXT	Ngõ vào dữ liệu nối tiếp.
P3.1	TXD	Ngõ xuất dữ liệu nối tiếp.
P3.2	INT0\	Ngõ vào ngắt cứng thứ 0
P3.3	INT1\	Ngõ vào ngắt cứng thứ 1
P3.4	T0	Ngõ vào của TIMER/COUNTER thứ 0
P3.5	T1	Ngõ vào của TIMER/COUNTER thứ 1
P3.6	WR\	Tín hiệu ghi dữ liệu lên bộ nhớ ngoài
P3.7	RD\	Tín hiệu đọc bộ nhớ dữ liệu ngoài

Bảng 1.2: Chức năng của các chân của Port 3

b. Các ngõ tín hiệu điều khiển:

- Ngõ tín hiệu PSEN (Program store enable):

PSEN là tín hiệu ngõ ra ở chân 29 có tác dụng cho phép đọc bộ nhớ chương trình mở rộng thường được nối đến chân OE\ (Output Enable) của Eprom cho phép đọc các byte mã lệnh.

PSEN ở mức thấp trong thời gian AT89C51 lấy lệnh. Các mã lệnh của chương trình được đọc từ EPROM qua bus dữ liệu và được chốt vào thanh ghi lệnh bên trong AT89C51 để giải mã lệnh. Khi AT89C51 thi hành chương trình trong EPROM nội PSEN sẽ ở mức logic 1.

- Ngõ tín hiệu điều khiển ALE (Address Latch Enable):

Khi AT89C51 truy xuất bộ nhớ bên ngoài, port 0 có chức năng là bus địa chỉ và bus dữ liệu do đó phải tách các đường dữ liệu và địa chỉ. Tín hiệu ra ALE ở chân thứ 30 dùng làm tín hiệu điều khiển để giải đa hợp các đường địa chỉ và dữ liệu khi kết nối chúng với IC chốt.

Tín hiệu ra ở chân ALE là một xung trong khoảng thời gian port 0 đóng vai trò là địa chỉ thấp nên chốt địa chỉ hoàn toàn tự động.

Các xung tín hiệu ALE có tốc độ bằng 1/6 lần tần số dao động trên chip và có thể được dùng làm tín hiệu clock cho các phần khác của hệ thống. Chân ALE được dùng làm ngõ vào xung lập trình cho EPROM trong AT89C51.

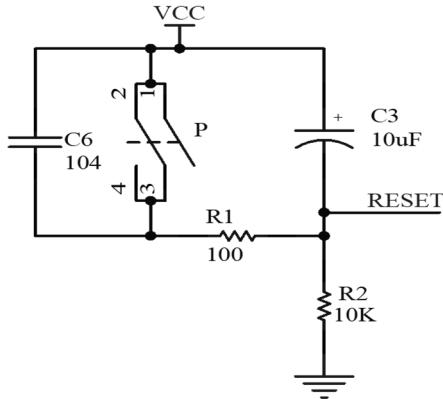
- Ngõ tín hiệu EA\ (External Access):

Tín hiệu vào EA\ ở chân 31 thường được mắc lên mức 1 hoặc mức 0. Nếu ở mức 1, AT89C51 thi hành chương trình từ EPROM nội trong khoảng địa chỉ thấp 4 Kbyte. Nếu ở mức 0, 8951 sẽ thi hành chương trình từ bộ nhớ mở rộng. Chân EA\ được lấy làm chân cấp nguồn 12V khi lập trình cho EPROM trong AT89C51.

- Ngõ tín hiệu RST (Reset):

Ngõ vào RST ở chân 9 là ngõ vào Reset của AT89C51. Khi ngõ vào tín hiệu này đưa lên cao ít nhất là 2 chu kỳ máy, các thanh ghi bên trong được nạp những giá

trị thích hợp để khởi động hệ thống. Khi cấp điện mạch tự động Reset.



Hình 1.3: Mạch Reset hệ thống

Các ngõ vào bộ dao động X1,X2:

Bộ dao động được tích hợp bên trong AT89C51, khi sử dụng AT89C51 người thiết kế chỉ cần kết nối thêm thạch anh và các tụ. Tần số thạch anh thường sử dụng cho AT89C51 là 12Mhz. Chân 40 (Vcc) được nối lên nguồn 5V.

#### 2.3.4. Cấu trúc bên trong của vi điều khiển AT89C51

Các chip vi điều khiển dùng làm thành phần trung tâm trong các thiết kế hướng điều khiển. Bộ nhớ thường có dung lượng bé hơn nhiều so với một hệ vi xử lý. Nó còn không có ổ đĩa và hệ điều hành. Chương trình điều khiển phải thường trú trong ROM. Do vậy, chương trình vẫn được lưu giữ ngay cả khi mất điện. Do lý do trên, chip 8051 có không gian bộ nhớ riêng cho chương trình và dữ liệu.

Cả bộ nhớ chương trình và bộ nhớ dữ liệu đều nằm trong chip. Tuy nhiên ta có thể mở rộng bộ nhớ chương trình và bộ nhớ dữ liệu bằng cách sử dụng các chip nhớ bên ngoài với dung lượng tối đa là 64K cho bộ nhớ chương trình và 64K bộ nhớ dữ liệu.

- Bộ nhớ chương trình (ROM):

Bộ nhớ chương trình lưu giữ chương trình điều khiển chip 8051. Sau khi RESET, CPU bắt đầu thực hiện chương trình từ địa chỉ 0000H. Khi chương trình lớn quá kích thước bộ nhớ chương trình bên trong chip, chương trình này phải được nạp vào bộ nhớ chương trình ngoài. Nếu chương trình nằm trong ROM nội, chân /EA phải được treo lên 5V. Nếu chương trình ở ROM ngoài, chân /EA phải nối đất. Việc truy xuất chương trình ở bộ nhớ ngoài phải kết hợp với chân tín hiệu truy xuất bộ nhớ ngoài /PSEN.

- Bộ nhớ dữ liệu (RAM)

AT89C51 có 128 byte RAM ở bên trong chip. RAM trong AT89C51 bao gồm nhiều thành phần: phần lưu trữ đa dụng, phần lưu trữ địa chỉ hóa từng bit, các bank thanh ghi và các thanh ghi chức năng đặc biệt.

Hệ 8051 có bộ nhớ theo cấu trúc Harvard: có những vùng bộ nhớ riêng biệt cho chương trình và dữ liệu. Chương trình và dữ liệu có thể chứa bên trong 8951 nhưng AT89C51 vẫn có thể kết nối với 64K byte bộ nhớ chương trình và 64K byte dữ liệu.

Bản đồ bộ nhớ data trên chip như sau:

Byte	Địa chỉ Bit	Byte	Địa chỉ Bit	
7F	Vùng RAM đa mục đích	FF		
		F0	F7 F6 F5 F4 F3 F2 F1 F0	B
		E0	E7 E6 E5 E4 E3 E2 E1 E0	ACC
		D0	D7 D6 D5 D4 D3 D2 D1 D0	PSW
30		B8	- - - BC BB BA B9 B8	IP
2F	7F 7E 7D 7C 7B 7A 79 78	B0	B7 B6 B5 B4 B3 B2 B1 B0	P.3
2E	77 76 75 74 73 72 71 70	A8	AF      AC AB AA A9 A8	IE
2D	6F 6E 6D 6C 6B 6A 69 68	A0	A7 A6 A5 A4 A3 A2 A1 A0	P2
2C	67 66 65 64 63 62 61 60			
2B	5F 5E 5D 5C 5B 5A 59 58	99	Không định địa chỉ bit	SBUF
2A	57 56 55 54 53 52 51 50	98	9F 9E 9D 9C 9B 9A 99 98	SCON
29	4F 4E 4D 4C 4B 4A 49 48			
28	47 46 45 44 43 42 41 40	90	97 96 95 94 93 92 91 90	P1
27	3F 3E 3D 3C 3B 3A 39 38			
26	37 36 35 34 33 32 31 30	8D	Không định địa chỉ bit	TH1
25	2F 2E 2D 2C 2B 2A 29 28	8C	Không định địa chỉ bit	TH0
24	27 26 25 24 23 22 21 20	8B	Không định địa chỉ bit	TL1
23	1F 1E 1D 1C 1B 1A 19 18	8A	Không định địa chỉ bit	TL0
22	17 16 15 14 13 12 11 10	89	Không định địa chỉ bit	TMOD
21	0F 0E 0D 0C 0B 0A 09 08	88	8F 8E 8D 8C 8B 8A 89 88	TCON
20	07 06 05 04 03 02 01 00	87	Không định địa chỉ bit	PCON
1F	Bank 3			
18		83	Không định địa chỉ bit	DPH
17	Bank 2	82	Không định địa chỉ bit	DPL
10		81	Không định địa chỉ bit	SP
0F	Bank 1	80	87 86 85 84 83 82 81 80	P0
08				
07	Bank 0			
00	Dãy thanh ghi mặc định R0 - R7			

RAM

CÁC THANH GHI CHỨC NĂNG  
ĐẶC BIỆT

Hình 1.4: Bản đồ bộ nhớ Data trên chip AT89C51.

Hai đặc tính cần chú ý là:

- Các thanh ghi và các port xuất nhập đã được định vị (xác định) trong bộ nhớ và có thể truy xuất trực tiếp giống như các địa chỉ bộ nhớ khác.
- Ngăn xếp bên trong Ram nội nhỏ hơn so với Ram ngoại như trong các bộ Microcontroller khác.

RAM bên trong AT89C51 được phân chia như sau:

- Các bank thanh ghi có địa chỉ từ 00H đến 1FH.
- RAM địa chỉ hóa từng bit có địa chỉ từ 20H đến 2FH.
- RAM đa dụng từ 30H đến 7FH.
- Các thanh ghi chức năng đặc biệt từ 80H đến FFH.

Sau đây ta sẽ xét cụ thể địa chỉ truy xuất và chức năng từng vùng RAM nội của AT89C51.

- *Vùng RAM đa dụng:*

Tùy hình vẽ cho thấy 80 byte đa dụng chiếm các địa chỉ từ 30H đến 7FH, 32 byte dưới từ 00H đến 1FH cũng có thể dùng với mục đích tương tự (mặc dù các địa chỉ này đã có mục đích khác).

Mọi địa chỉ trong vùng RAM đa dụng đều có thể truy xuất tự do dùng kiểu địa chỉ trực tiếp hoặc gián tiếp.

- *RAM có thể truy xuất từng bit:*

AT89C51 chứa 210 bit được địa chỉ hóa, trong đó có 128 bit có chứa các byte chứa các địa chỉ từ 20H đến 2FH và các bit còn lại chứa trong nhóm thanh ghi có chức năng đặc biệt. Ý tưởng truy xuất từng bit bằng phần mềm là các đặc tính mạnh của microcontroller xử lý chung. Các bit có thể được đặt, xóa, AND, OR, ... , với một lệnh đơn. Đa số các microcontroller xử lý đòi hỏi một chuỗi lệnh đọc - sửa - ghi để đạt được mục đích tương tự. Ngoài ra các port cũng có thể truy xuất được từng bit.

128 bit có chứa các byte có địa chỉ từ 00H - 1FH cũng có thể truy xuất như các byte hoặc các bit phụ thuộc vào lệnh được dùng.

- *Các bank thanh ghi :*

32 byte thấp của bộ nhớ nội được dành cho các bank thanh ghi. Bộ lệnh 8951 hỗ trợ 8 thanh ghi có tên là R0 - R7 và theo mặc định sau khi reset hệ thống, các thanh ghi này có các địa chỉ từ 00H - 07H.

Các lệnh dùng các thanh ghi R0 - R7 sẽ ngắn hơn và nhanh hơn so với các lệnh có chức năng tương ứng dùng kiểu địa chỉ trực tiếp. Các dữ liệu được dùng thường xuyên nên dùng một trong các thanh ghi này.

Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi R0 - R7 để chuyển đổi việc truy xuất các bank thanh ghi ta phải thay đổi các bit chọn bank trong thanh ghi trạng thái.

## 2.4. Các hoạt động chức năng chính của 8051

### 2.4.1. Hoạt động định thời

Bộ định thời của Timer là một chuỗi các Flip Flop được chia làm 2, nó nhận tín hiệu vào là một nguồn xung clock, xung clock được đưa vào Flip Flop thứ nhất là xung clock của Flip Flop thứ hai mà nó cũng chia tần số clock này cho 2 và cứ tiếp tục.

Vì mỗi tầng kế tiếp chia cho 2, nên Timer n tầng phải chia tần số clock ngõ vào cho  $2^n$ . Ngõ ra của tầng cuối cùng là clock của Flip Flop tràn Timer hoặc cờ mà nó kiểm tra bởi phần mềm hoặc sinh ra ngắt. Giá trị nhị phân trong các FF của bộ Timer có thể được nghĩ như đếm xung clock hoặc các sự kiện quan trọng bởi vì Timer được khởi động. Ví dụ Timer 16 bit có thể đếm đến từ FFFFH sang 0000H.

Các bộ định thời dùng để tạo ra các khoảng thời gian khác nhau, dùng để đếm sự kiện hoặc dùng để tạo tốc độ baud cho việc truyền thông nối tiếp. Trong 8051 có hai bộ định thời là T0 và T1. Việc lựa chọn chế độ hoạt động cho các Timer này nhờ vào thanh ghi TMOD. Còn việc điều khiển các Timer hoạt động nhờ vào thanh ghi TCON. Có 4 mode hoạt động cho các Timer. Mode 0 là chế độ định thời 13 - bit, Mode 1 là chế độ định thời 16-bit, Mode 2 là chế độ định thời tự nạp lại 8 - bit, Mode 3 là chế độ định thời chia xé và có hoạt động khác nhau cho từng bộ định thời.

Các Timer của 8951 được truy xuất bởi việc dùng 6 thanh ghi chức năng đặc biệt như sau :

Timer SFR	Purpose	Address	Bit-Addressable
TCON	Control	88H	YES
TMOD	Mode	89H	NO
TL0	Timer 0 low-byte	8AH	NO
TL1	Timer 1 low-byte	8BH	NO
TH0	Timer 0 high-byte	8CH	NO
TH1	Timer 1 high-byte	8DH	NO

### 2.4.2. Hoạt động của ngắt trong 8051

Có 5 nguyên nhân để tạo ra ngắt trong 8051. Đó là 2 ngắt ngoài, hai ngắt do bộ định thời và một ngắt do port nối tiếp. Khi ta thiết lập trạng thái ban đầu (sau khi RESET), tất cả các ngắt đều bị vô hiệu hóa và sau đó chúng được cho phép riêng rẽ bằng phần mềm.

Khi một ngắt được chấp nhận, giá trị được nạp cho bộ đếm chương trình được gọi là véc tơ ngắt. Véc tơ ngắt là địa chỉ bắt đầu của trình phục vụ ngắt của các ngắt tương ứng. Các véc tơ ngắt được cho ở bảng sau :

NGUỒN NGẮT	CỜ	ĐỊA CHỈ VECTO NGẮT
Reset hệ thống	RST	0000H
Ngắt ngoài 0	IE0	0003H
Bộ định thời 0	TF0	000BH
Ngắt ngoài 1	IE1	0013H
Bộ định thời 1	TF1	001BH
Port nối tiếp	RI or TI	0023H
Bộ định thời 2	TF2 or EXF2	002BH

Khi chương trình đang thực hiện, nếu có ngắt với ưu tiên cao xuất hiện, trình phục vụ ngắt cho ngắt có mức ưu tiên thấp tạm dừng. Ta không thể tạm dừng một chương trình ngắt có mức ưu tiên cao hơn. Khi có 2 ngắt khác nhau xuất hiện đồng thời, ngắt có mức ưu tiên cao sẽ được phục vụ trước. Khi 2 ngắt có cùng mức ưu tiên xuất hiện đồng thời, chuỗi vòng cờ định sẽ xác định ngắt nào được phục vụ trước. Chuỗi vòng này sẽ là ngắt ngoài 0, ngắt ngoài 1, ngắt do bộ định thời 0, ngắt do bộ định thời 1, ngắt do port nối tiếp, ngắt do bộ định thời 2 (đối với 8052).

#### 2.4.3. *Hoạt động RESET của 8051*

8051 được reset bằng cách giữ chân RST ở mức cao tối thiểu hai chu kỳ máy và sau đó chuyển về mức thấp. Trạng thái của tất cả các thanh ghi sau khi reset hệ thống như sau:

THANH GHI	NỘI DUNG
Bộ đếm chương trình PC	0000H
Thanh chứa A	00H
Thanh ghi B	00H
PSW	00H
SP	07H
DPRT	0000H
Port 0 đến Port 3	FFH
IP	xxx00000B
IE	0xx00000B
Các thanh ghi định thời	00H
SCON	00H

THANH GHI	NỘI DUNG
SBUF	00H
PCON (HMOS)	0xxxxxxxxB
PCON (CMOS)	0xxx0000B

Khi reset hệ thống thanh ghi PC được nạp địa chỉ 0000H, khi đó chương trình sẽ bắt đầu từ địa chỉ đầu tiên trong bộ nhớ chương trình. Nội dung của RAM trên chip không bị ảnh hưởng khi ta reset hệ thống.

## 2.5. Tóm tắt tập lệnh của 8051

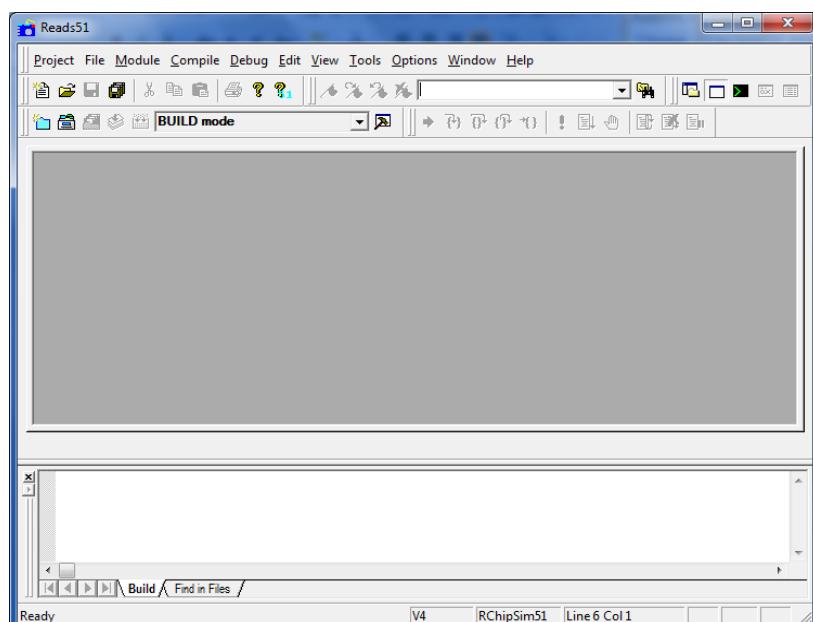
Tập lệnh của MSC-51 được tối ưu hóa cho các dụng điều khiển 8 bit. Nhiều kiểu định địa chỉ cô đọng và nhanh, dùng để truy xuất RAM nội được dùng đến nhằm tạo thuận lợi cho các thao tác trên các cấu trúc dữ liệu nhỏ. Tập lệnh cũng hỗ trợ các biến 1 bit cho phép quản lý bit trực tiếp trong các hệ logic và điều khiển có yêu cầu xử lý bit.

Cũng như trong các vi xử lý 8 bit, các lệnh của 8051 có các opcode 8 bit, do đó số lệnh có thể lên đến 256 lệnh (thực tế có 255 lệnh, một lệnh không được định nghĩa). Ngoài opcode, một số lệnh còn có thêm 1 hay 2 byte nữa cho dữ liệu hoặc địa chỉ. Tập lệnh có 139 lệnh 1 byte, 94 lệnh 2 byte và 24 lệnh 3 byte. Tập lệnh của 8051 được chia thành 5 nhóm:

- Số học
- Logic
- Chuyển dữ liệu
- Chuyển điều khiển
- Nhóm lệnh rẽ nhánh.

## 2.6. Lập trình cho AT89C51 sử dụng trình biên dịch Reads51

Giao diện chính của trình biên dịch Reads51 như sau:



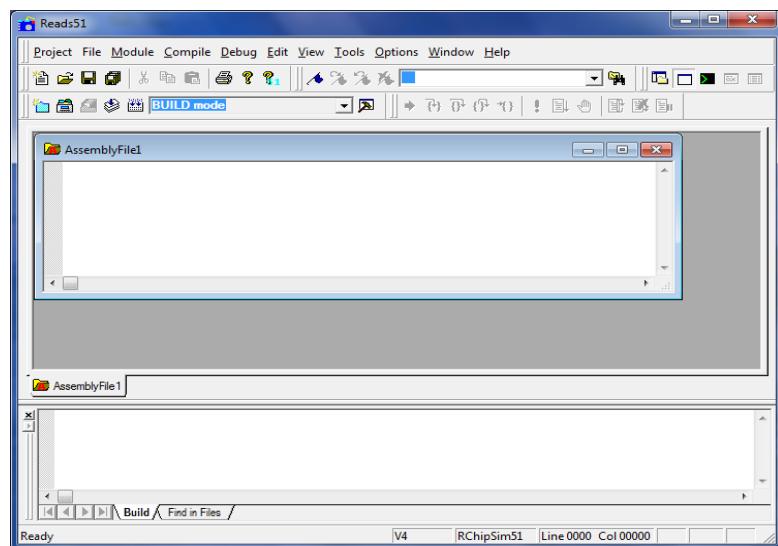
Hình 1.4: Trình biên dịch Reads51

Các bước để viết một chương trình ứng dụng bằng Assembly như sau:

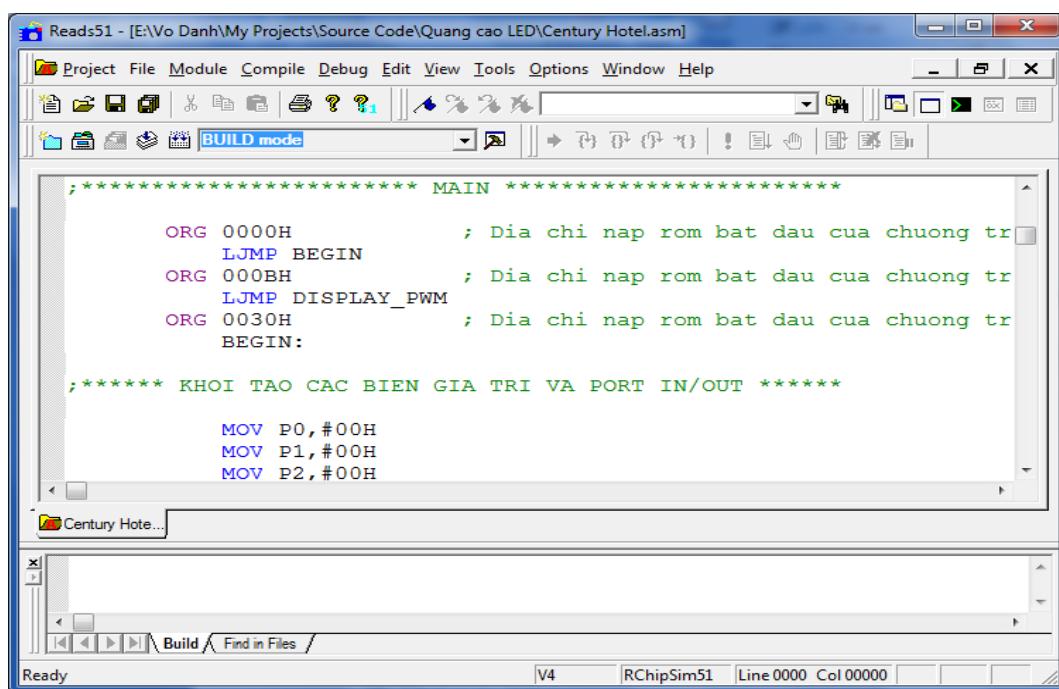
2.6.1. Tạo một file mới, vào menu File - New File (hoặc Ctrl + N), chọn Asembly File và bấm Ok.



Khi đó sẽ xuất hiện một màn hình soạn thảo với tên file mặc định là AssemblyFile1, chúng ta vào menu File - Save (hoặc Ctrl + S) để lưu lại file với tên file mới.



2.6.2. Soạn thảo chương trình ứng dụng.



```
;***** MAIN *****
ORG 0000H ; Dia chi nap rom bat dau cua chuong tr
LJMP BEGIN
ORG 000BH ; Dia chi nap rom bat dau cua chuong tr
LJMP DISPLAY_PWM
ORG 0030H ; Dia chi nap rom bat dau cua chuong tr
BEGIN:

;***** KHOI TAO CAC BIEN GIA TRI VA PORT IN/OUT *****

MOV P0,#00H
MOV P1,#00H
MOV P2,#00H
```

Cáu trúc một chương trình hợp ngữ cho 8051 (sử dụng trình hợp ngữ Reads51)

; \*\*\*\* Đầu chương trình, khai báo file chứa địa chỉ của các thanh ghi SFR

```
#include <sfr51.inc>
; * Định nghĩa tên gọi cho các chân cổng vào/ra (nếu muốn)
  #define led1 P1.0
  #define led2 P1.1
  ...
; * Khai báo các biến dạng byte (nếu có)
  var1 data 0x30
  var2 data 0x31
  ...
; * Khai báo các biến dạng bit (nếu có)
  flag1 bit 0x00
  flag2 bit 0x01
  ...
; * Định nghĩa các hằng số (nếu có)
  constant1 equ 123
  constant2 equ 456
  ...
; * Tạo mã đặt tại địa chỉ RESET
  org 0x0000
  ljmp main
; * Tạo mã đặt tại các vector ngắn (nếu sử dụng ngắn)
  org 0x0003
  ljmp ChuongTrinhXuLyNgatNgoai0
  org 0x000B
  ljmp ChuongTrinhXuLyNgatTimer0
  ...
; Đặt địa chỉ đầu cho chương trình chính
  org 0x0030
  MAIN:
; Bắt đầu viết các lệnh cho chương trình chính từ đây
  mov SP,#0x6F
; Viết các thủ tục khởi tạo hệ thống
  ...
; Viết thân chương trình chính (vòng lặp chính)
  MAIN_LOOP:
  ...
```

```

sjmp MAIN_LOOP
; * Viết các chương trình con và các chương trình xử lý
ngắt (nếu có)
ChuongTrinhCon1:
; Các lệnh xử lý của chương trình con 1
...
RET      ; Kết thúc bằng lệnh RET
ChuongTrinhCon2:
; Các lệnh xử lý của chương trình con 2
...
RET      ; Kết thúc bằng lệnh RET
...
ChuongTrinhXuLyNgatNgoai0:
; Các lệnh xử lý của chương trình xử lý ngắt ngoài 0
...
RETI     ; Kết thúc bằng lệnh RETI
ChuongTrinhXuLyNgatTimer0:
; Các lệnh xử lý của chương trình xử lý ngắt timer 0
...
RETI     ; Kết thúc bằng lệnh RETI
...
;* Định nghĩa các bảng hằng số trong bộ nhớ chương trình
Bang1:
db 0,1,0x02,0x86
Bang2:
db 156,235,8,9
END ; Chỉ dẫn báo hiệu kết thúc toàn bộ đoạn chương trình

```

### **Chú ý:**

- Các chữ viết sát lè, kết thúc bằng dấu “:” là các nhãn, đó có thể là đầu một chương trình con hoặc đơn giản chỉ là một nhãn phụ trong thân một chương trình nào đó.

- Các chữ được đặt sau dấu “;” sẽ được coi là các câu chú thích và trình hợp ngữ sẽ bỏ qua, nội dung tùy ý nhưng phải trên một dòng, nếu kéo dài xuống dòng khác thì phải thêm dấu “;” khác vào trước phần chú thích của dòng đó.

- Giữa các tên (nhãn) phải sự thống nhất khi khai báo và lúc sử dụng trong câu lệnh, không được khai báo một kiểu, dùng trong lệnh lại kiểu khác đi.

#### 2.6.3. Biên dịch (Build) file Assembly vừa soạn thảo thành mã Hex:

Sau khi biên dịch nếu có lỗi sẽ được thông báo tại cửa sổ Output ngay bên dưới màn hình soạn thảo (View - Output Windows) bao gồm tên của lỗi, dòng lệnh xảy ra

lỗi. Nếu chương trình không có lỗi sẽ hiện ra thông báo “Successful build”. Chương trình được dịch thành file “.HEX” và lưu cùng vị trí với file Assembly.

1.2.2.4. Sau khi đã có file “.HEX” ta sử dụng một mạch nạp 8051 để nạp chương trình vào ROM của 8051.

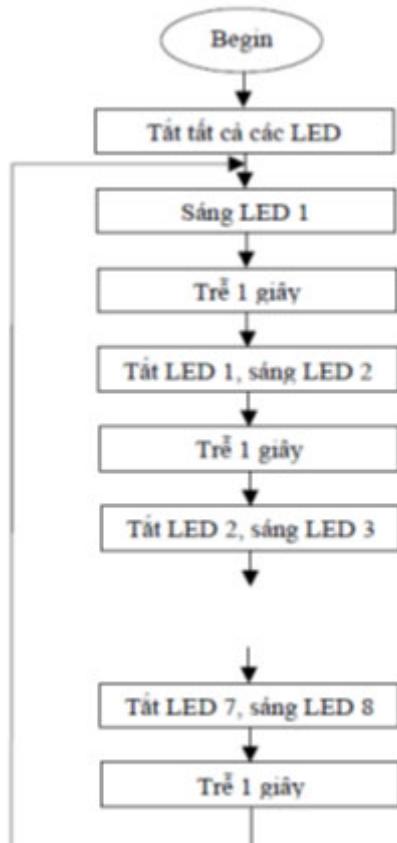
The screenshot shows the Reads51 software interface. The assembly code in the editor window is as follows:

```
;*****  
ORG 0  
ORG 0  
ORG 0 0030H ; Dia chi nap rom bat dau cua chuong tr  
BEGIN:  
  
;***** KHOI TAO CAC BIEN GIA TRI VA PORT IN/OUT *****  
  
MOV P0,#00H  
MOV P1,#00H  
MOV P2,#00H
```

The status bar at the bottom indicates "E:\Vo Danh\My Projects\Source Code\Quang cao LED\Century Hotel.HEX is up to date Successful build."

### 3. NỘI DUNG THỰC HÀNH

3.1. Cho 8 LED đơn nối vào Port 1 của vi điều khiển 8051. Hãy lập trình để cho sáng lần lượt từng LED, mỗi LED sáng 1 giây.



Trên đây là lưu đồ, việc thể hiện bằng lệnh lưu đồ trên có thể bằng cách chân phương (làm lần lượt) hoặc có thể dùng lệnh quay để đưa bit = 0 ra lần lượt các chân cổng làm LED sáng theo lần lượt.

- Cách chân phương:

```
#include    <sfr51.inc>
org 00h
ljmp main
org 40h
main:
mov SP,#5fh ; Việc tắt tất cả các LED được tự
; động làm do khi reset lên, các
; chân cổng đều = 1
mov     p1,#11111110b ; Sáng LED 1
lcall   delay_1s
mov     p1,#11111101b ; Tắt LED 1, sáng LED 2
lcall   delay_1s
mov     p1,#11111011b ; Tắt LED 2, sáng LED 3
        lcall   delay_1s
; ..... ; Làm các bước tương tự
mov p1,#01111111b
lcall   delay_1s
sjmp   main
delay_1s:
mov     r1,#10
loop1:
mov     r2,#100
loop2:
mov     r3,#100
loop3:
nop
;... <Đất cả 8 lệnh Nop>
nop
djnz   r3,loop3
    djnz   r2,loop2
    djnz   r1,loop1
ret
end
```

- Nếu sử dụng lệnh quay thì chương trình sẽ gọn hơn:

```

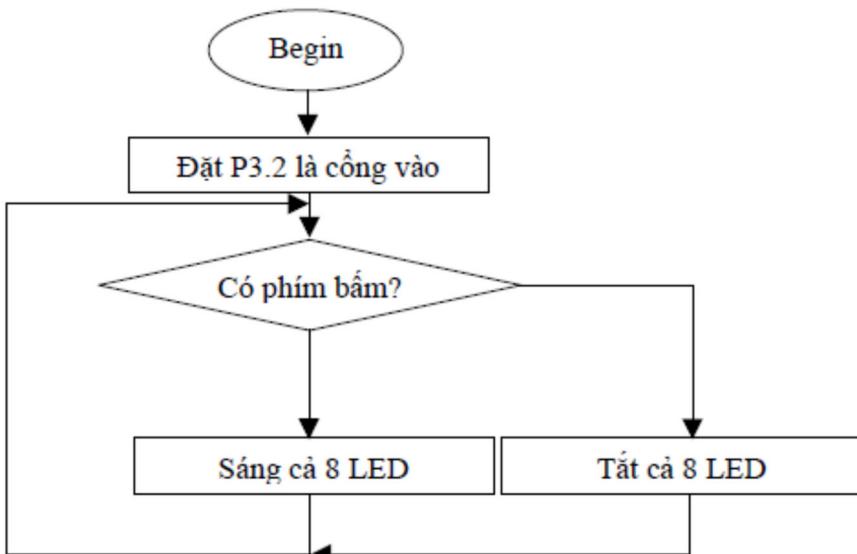
main:
    mov    SP, #5fh
    mov    a, #11111110b
main_loop:
    mov    P1, a
    lcall   delay_1s
    rl    a
    sjmp   main_loop
delay_1s:
;... như ở trên ...
end

```

Lưu ý là trong các đoạn chương trình trên, ngăn xếp được khởi tạo với giá trị ban đầu của thanh ghi con trả ngăn xếp SP là 5fh, tức là các địa chỉ hay dữ liệu cát vào ngăn xếp sẽ bắt đầu từ ô nhớ 60h trở đi. Việc khởi tạo ngăn xếp là một thao tác không thể thiếu khi trong chương trình có sử dụng lệnh gọi chương trình con hoặc các ngắt.

**3.2.** Cho phím bấm nối với P3.2 và 8 LED đơn nối vào Port1 của vi điều khiển 8051. Kiểm tra phím bấm, khi phím được bấm thì sáng cả 8 LED, khi không bấm phím thì tắt cả 8 LED.

Lưu đồ thuật toán như sau:



Với lưu đồ thuật toán như trên ta viết được chương trình sau:

```

org 00h
ljmp main
org 40h
main:
; Việc đặt cổng P3.2 làm cổng vào được tự động

```

```

; thực hiện khi 8051 reset xong.
; Các LED cũng tự động được tắt vì khi 8051 reset
; xong, các cổng đều = 1
jnb P3.2, phim_bam
mov P1,#0ffh ; tắt tất cả các LED nếu
; không có phím bấm
sjmp main
phim_bam:
mov P1,#0
sjmp main ; sáng tất cả các LED nếu có
; phím bấm
end

```

Với cùng mạch phần cứng như trên, ta có thể thực hành về ngắt của vi điều khiển. Bài toán thực hành đặt ra có thể là nếu có phím bấm thì đảo trạng thái của 8 LED (đang sáng thì thành tắt và ngược lại). Như vậy nếu ta bấm phím, trạng thái của LED sẽ được đảo lại, mỗi lần bấm phím đảo một lần.

Nếu không sử dụng ngắt, ta có thể viết chương trình gần giống với chương trình trên, chỉ khác là phải thêm thao tác đợi nhả phím ra trước khi quay trở lại quét kiểm tra điện áp tại chân P3.2. Nếu không có thao tác này, chân P3.2 xuống mức 0 sẽ bị gây ra nhiều lần đảo trạng thái LED bởi vì chân P3.2 sẽ được quét liên tục, thấy còn ở mức 0 là lại đảo trạng thái, cho đến khi nhả phím ra thì các LED sẽ cùng sáng hoặc cùng tắt, tùy thuộc vào việc lần đảo trạng thái nào được thực hiện cuối cùng. Như vậy sẽ không đúng với mong muốn là chỉ đảo một lần khi bấm một lần.

```

...
main:
jb P3.2,$ ; Nhảy tại chỗ chờ cho đến
; khi nào P3.2 = 0
; (tức là chờ cho đến khi có phím bấm)
phim_bam:
mov a,P1 ; đọc giá trị hiện thời của
; cổng P1
cpl a ; đảo trạng thái đọc được đi
mov P1,a ; đưa trở lại cổng P1 làm
; đảo trạng thái LED
jnb P3.2,$ ; nhảy tại chỗ để đợi cho đèn khi
nào P3.2 = 1
sjmp main ; trở lại
end

```

**3.3.** Cho phím bấm nối với P3.2 và 8 LED đơn nối vào Port1 của vi điều khiển 8051. Kiểm tra phím bấm, khi bấm phím lần 1 đèn LED ở P1.0 sáng, bấm phím lần 2 thì đèn LED ở P1.0 sáng, tiếp tục đến lần bấm phím thứ 8 thì đèn LED ở P1.7 sáng, bấm phím lần 9 thì quay về sáng đèn LED ở P1.0 .v.v.

**3.4.** Cho phím bấm nối với P3.2 và 8 LED đơn nối vào Port1 của vi điều khiển 8051. Khi khởi động các đèn LED sáng từ P1.0 đến P1.7, khi phím được bấm thì đảo chiều sáng của các đèn LED từ P1.7 đến P1.0. Tiếp tục đảo chiều ở lần bấm phím tiếp theo.

## BÀI 5:

# LẬP TRÌNH CHO BỘ ĐÉM/BỘ ĐỊNH THỜI VI ĐIỀU KHIỂN 8051

## 1. MỤC ĐÍCH

Bài thực hành nhằm cung cấp cho sinh viên các kỹ năng:

- Lập trình cho bộ đếm và bộ định thời trong vi điều khiển 8051.
- Khảo sát hoạt động I/O vi điều khiển 8051 thông qua việc giao tiếp với Led 7 đoạn và và phím bấm. Ứng dụng một số cấu trúc giải thuật cơ bản trong lập trình điều khiển IO sử dụng vi điều khiển 8051.

## 2. TÓM TẮT LÝ THUYẾT

### 2.1. Cấu trúc bộ đếm/bộ định thời trên 8051

- Vi điều khiển 8051 có hai bộ định thời là Timer 0 và Timer1;
- Cả hai bộ định thời Timer 0 và Timer 1 đều có độ dài 16 bit, được truy cập như hai thanh ghi tách biệt gồm byte thấp và byte cao;
- Các thanh ghi của bộ Timer 0:

TH0								TL0							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- Các thanh ghi của bộ Timer 1:

TH1								TL1							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

- Cả hai bộ định thời dùng chung một thanh ghi để thiết lập các chế độ làm việc khác nhau của bộ định thời (được gọi là TMOD)

- Thanh ghi TMOD là thanh ghi 8 bit gồm có 4 bit thấp được thiết lập dành cho bộ Timer 0 và 4 bit cao dành cho Timer 1:
  - + 02 bit thấp của chúng dùng để thiết lập chế độ của bộ định thời;
  - + 02 bit cao dùng để xác định phép toán.

MSB				LSB			
GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER1				TIMER0			

- Bít cổng GATE:

- + Mỗi bộ định thời thực hiện điểm khởi động và dừng;
- + Một số bộ định thời thực hiện điều này bằng phần mềm, một số khác bằng

phản cứng và một số khác vừa bằng phản cứng vừa bằng phản mềm. Các bộ định thời trên 8051 có cả hai.

+ Khi bít GATE = 0, khởi động và dừng bộ định thời được thực hiện bằng phản mềm;

+ Khi bít GATE = 1, khởi động và ngừng bộ định thời bằng phản cứng từ nguồn ngoài;

+ Để tránh sự lẩn lộn ngay từ bây giờ ta đặt GATE = 0 có nghĩa là không cần khởi động và dừng các bộ định thời bằng phản cứng từ bên ngoài.

- C/T

+ Được dùng để quyết định xem bộ định thời được dùng như một bộ tạo trễ thời gian hay bộ đếm sự kiện.

+ Nếu bít C/T = 0 thì nó được dùng như một bộ tạo trễ thời gian, nguồn xung clock cho chế độ trễ thời gian là tần số thạch anh của 8051.

+ Nếu bít C/T = 1 thì nó được dùng như một đếm sự kiện, nguồn xung clock được đưa đến từ các chân 14 và 15 (T0, T1) của vi điều khiển.

- Các bít M1, M0:

+ Là các bít chế độ của các bộ Timer 0 và Timer 1;

M1	M0	Chế độ	Chế độ hoạt động
0	0	0	Bộ định thời 13 bít gồm 8 bít là bộ định thời / bộ đếm 5 bít đặt trước
0	1	1	Bộ định thời 16 bít (không có đặt trước)
1	0	2	Bộ định thời 8 bít tự nạp lại
1	1	3	Chế độ bộ định thời chia tách

## 2.2. Lập trình cho bộ đếm

Việc lập trình cho bộ đếm/bộ định thời trong 8051 được thực hiện theo các bước như sau:

B1. Nạp giá trị cho thanh ghi TMOD xác định bộ định thời nào và chế độ định thời nào được chọn;

B2. Nạp giá trị đếm ban đầu cho thanh ghi TL và TH;

B3. Khởi động bộ định thời;

B4. Kiểm tra trạng thái bật cờ của bộ định thời TF bằng lệnh “JNB TFx, đích”.

Thoát khỏi vòng lặp khi TF được bật lên mức cao (mức 1);

B5. Dừng bộ định thời;

B6. Xóa cờ TF cho vòng kế tiếp;

B7. Quay trở lại bước 2 để nạp lại TL và TH.

### 3. NỘI DUNG THỰC HÀNH

#### 3.1. Tạo thời gian trễ bằng Timer

Viết chương trình nhấp nháy các đèn LED nối vào cổng P0 với thời gian 50ms tạo ra bởi Timer 1 với thạch anh tần số 12MHz.

```
org 00h
main:
    mov P0, #00h           ; Bật LED
    call delay             ; Đợi 50ms
    mov P0, #0ffh          ; Tắt LED
    call delay             ; Đợi 50ms
    sjmp main             ; Lặp lại
; Chương trình con delay
delay:
    mov TMOD, #10h         ; Timer 1, mode 1 (16 bit )
    mov TH1, #3Ch           ; or mov TH1, #high(-50000)
    mov TL1, #B0Fh          ; or mov TL1, #low(-50000),
    setb TR1               ; Cho phép timer chạy
    jnb TF1, $              ; Nhảy tại chỗ, không làm gì
    clr TR1                ; Nếu timer tràn thì dừng timer
    clr TF1                ; Xoá cờ tràn
    ret                    ; Kết thúc chương trình con delay
end
```

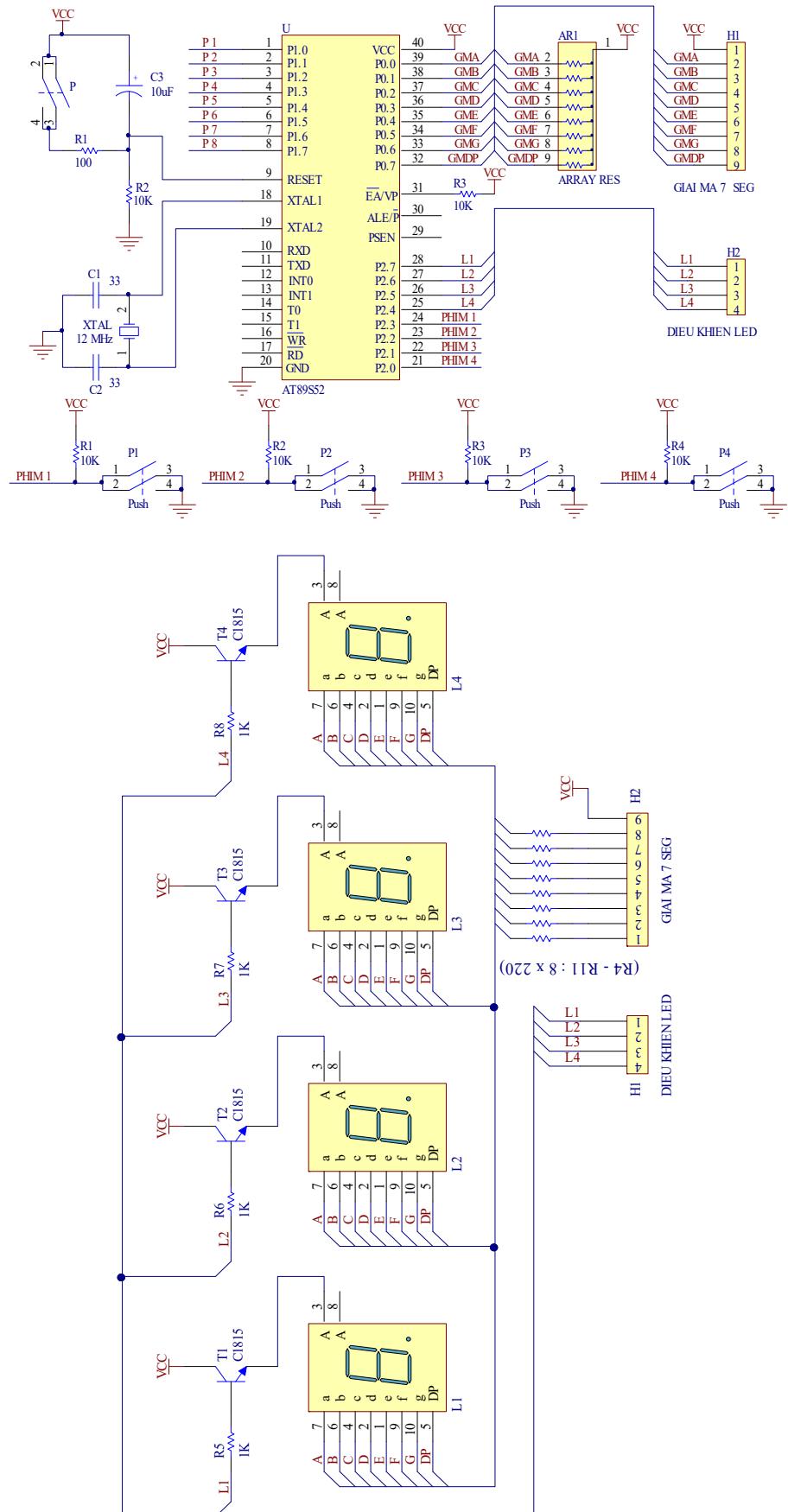
Thời gian tối đa mà Timer có thể định thời được là 65536 chu kỳ máy nên nếu muốn tạo những khoảng thời gian lâu hơn thì cần dùng thêm vòng lặp.

Ví dụ : Nếu muốn tạo ra thời gian trễ 1 giây thì ta chỉ cần cho Timer chạy 50ms, và lặp lại 20 lần.

```
delay:
    mov R4, #20             ; Số lần lặp
loop:
    mov TMOD, #10h          ; Timer 1, mode 1 (16 bit )
    mov TH1, #3Ch            ; or mov TH1, #high(-50000)
    mov TL1, #B0h            ; or mov TL1, #low(-50000),
    setb TR1                ; Cho phép timer chạy
    jnb TF1, $              ; Nhảy tại chỗ, không làm gì
    clr TR1                ; Nếu timer tràn thì dừng timer
    clr TF1                ; Xoá cờ tràn
    djnz R4, loop           ; Tiếp tục lặp
ret                     ; Kết thúc chương trình con delay
```

### 3.2. Quét hiển thị lên LED 7 thanh

Cho sơ đồ nguyên lý gồm 4 LED 7 đoạn và 4 phím bấm giao tiếp với vi điều khiển AT89C51 như sau:



Với sơ đồ nguyên lý như trên chúng ta sẽ viết chương trình hiển thị lên LED 7 đoạn nội dung từ 4 ô nhớ nằm trong bộ nhớ dữ liệu, mỗi LED ứng với một ô nhớ.

Nguyên lý của việc quét hiển thị bằng LED 7 đoạn như sau: chúng ta dựa theo nguyên tắc quét ảnh vì mắt chúng ta có hiện tượng lưu ảnh trên võng mạc nên khi ta quét với tần số nhanh thì mắt chúng ta không có cảm giác là nó sáng lần lượt các LED mà có cảm giác là nó sáng tất. Đầu tiên ta phải gửi dữ liệu muốn hiển thị cho LED1 và chân bật chân điều khiển của LED1 đó lên để cho LED1 đó hiện thị. Sau đó ta lại gửi tiếp dữ liệu muốn hiển thị LED 2 và cũng phải bật chân điều khiển của LED2 đó. Để cho số tiếp theo hiện thị lên LED2. Cứ như thế cho LED 3 và LED4. Quá trình này được diễn ra nhanh với tần số cao. Nên ta không hề có cảm giác là các LED sáng lần lượt. Để kiểm tra nguyên lý của nó chúng ta hãy tăng thời gian delay lên để thấy rõ quét LED của nó!

Ví dụ: Để hiện thị được số 1234 trên 4 con LED 7 vạch. Đầu tiên ta phải cho đèn 1 sáng sau đó đèn 2 sáng sau đó đèn 3 4. Ta cho các LED này hiện thị lần lượt và được lặp đi lặp lại với tần số cao nên ta sẽ được số 1234 hiện thị trên LED.

```
; ----- * DIA CHI CAC THANH GHI VA CAC PORT * -----
#include <sfr51.inc>

; ----- * KHAI BAO CAC HANG VA BIEN * -----
L1 EQU 20H ; Dia chi chua du lieu LED 1
L2 EQU 21H ; Dia chi chua du lieu LED 2
L3 EQU 22H ; Dia chi chua du lieu LED 3
L4 EQU 23H ; Dia chi chua du lieu LED 4

; ----- * KHAI BAO CAC CHAN CHUC NANG * -----
DKL1 BIT P1.3 ; Dieu khien on/off L1
DKL2 BIT P1.2 ; Dieu khien on/off L2
DKL3 BIT P1.1 ; Dieu khien on/off L3
DKL4 BIT P1.0 ; Dieu khien on/off L4
DK_DATA EQU P0 ; Dieu khien dua du lieu LED 7
; thanh ra P0

; ----- * KHAI BAO ORG * -----
ORG 0000H ; Dia chi ROM bat dau chuong
; trinh (Sau khi Reset)
LJMP MAIN ; Khoi dong lai toan bo chuong
; trinh
ORG 0030H ; Nhay ra khoi vung cac vecto ngat
```

```

; ----- * MAIN PROGRAM * -----
MAIN:
; Khoi tao gia tri cac PORT, cac dia chi nho quan trong
MOV L1,#01H           ; Khoi tao L1 chua gia tri 01
MOV L2,#02H           ; Khoi tao L1 chua gia tri 02
MOV L3,#03H           ; Khoi tao L1 chua gia tri 03
MOV L4,#04H           ; Khoi tao L1 chua gia tri 04
MOV DK_DATA,FFH       ; Xoa du lieu trong tai PORT 2
; Bat dau qua trinh hien thi len LED 7 SEG
HIEN THI:
LED1:
MOV A,L1      ; Chuyen gia tri LED 1 vao thanh ghi A
LCALL CHUYEN_MA    ; Gọi chương trình chuyen ma
MOV DK_DATA,A      ; Chuyen du lieu ra PORT 2
SETB DKL1          ; Bat sang LED 1
LCALL DELAY        ; DELAY
CLR DKL1          ; Tat LED 1 de hien thi LED 2
LED2:
MOV A,L2
LCALL CHUYEN_MA
MOV DK_DATA,A
SETB DKL2
LCALL DELAY
CLR DKL2
LED3:
MOV A,L3
LCALL CHUYEN_MA
MOV DK_DATA,A
SETB DKL3
LCALL DELAY
CLR DKL3
LED4:
MOV A,L4
LCALL CHUYEN_MA
MOV DK_DATA,A
SETB DKL4
LCALL DELAY
CLR DKL4
SJMP HIEN THI

```

```

; PROGRAM ----* CHUYEN DOI MA BCD-7 THANH *-----
CHUYEN_MA:           ; Cấu trúc CASE
CJNE A,#00H,SO1      ; So sánh giá trị trong A với 0,
                      ; nếu không bằng thì nhảy sang SO1
MOV  A,#11000000B    ; Nếu bằng thì chuyển 0 sang mã 7 thanh
SJMP END_CM          ; Nhảy về kết thúc quá trình chuyển mã

SO1:
CJNE A,#01H,SO2
MOV  A,#11111001B
SJMP END_CM

SO2:
CJNE A,#02H,SO3
MOV  A,#10100100B
SJMP END_CM

SO3:
CJNE A,#03H,SO4
MOV  A,#10110000B
SJMP END_CM

SO4:
CJNE A,#04H,SO5
MOV  A,#10011001B
SJMP END_CM

SO5:
CJNE A,#05H,SO6
MOV  A,#10010010B
SJMP END_CM

SO6:
CJNE A,#06H,SO7
MOV  A,#10000010B
SJMP END_CM

SO7: CJNE A,#07H,SO8
MOV  A,#11111000B
SJMP END_CM

SO8: CJNE A,#08H,SO9
MOV  A,#10000000B
SJMP END_CM

SO9: MOV  A,#10010000B
END_CM:
RET                  ; Kết thúc chương trình con chuyển mã

```

DELAY:

```
MOV TMOD,#10H          ; Timer 1, mode 1 (16 bit )
MOV TH1,#3CH            ; or mov TH1,#high(-50000)
MOV TL1,#B0FH            ; or mov TL1,#low(-50000),
SETB TR1                ; Cho phép timer chạy
JNB TF1,$                ; Nhảy tại chỗ, không làm gì
CLR TR1                  ; Nếu timer tràn thì dừng timer
CLR TF1                  ; Xoá cờ tràn
RET                      ; Kết thúc chương trình con Delay
END
```

Nạp chương trình và quan sát trạng thái các đèn LED 7 đoạn.

Thay đổi thời gian trễ của chương trình DELAY thành 5 ms và nhận xét kết quả.

### 3.3. Giao tiếp Input với phím bấm

Với sơ đồ nguyên lý như trên chúng ta sẽ viết chương trình hiển thị lên LED 7 đoạn nội dung từ 4 ô nhớ nằm trong bộ nhớ dữ liệu, mỗi LED ứng với một ô nhớ. Sau đó chúng ta sẽ thay đổi nội dung trong từng ô nhớ bằng phím bấm (mỗi lần bấm phím giá trị của mỗi LED sẽ tăng 1 đơn vị) và cập nhật giá trị lên các LED 7 đoạn tương ứng.

```
; ----- * DIA CHI CAC THANH GHI VA CAC PORT * -----
#include <sfr51.inc>
; ----- * KHAI BAO CAC HANG VA BIEN * -----
L1 EQU 20H              ; Dia chi chua du lieu LED 1
L2 EQU 21H              ; Dia chi chua du lieu LED 2
L3 EQU 22H              ; Dia chi chua du lieu LED 3
L4 EQU 23H              ; Dia chi chua du lieu LED 4

; ----- * KHAI BAO CAC CHAN CHUC NANG * -----
DKL1 BIT P1.3            ; Dieu khien on/off L1
DKL2 BIT P1.2            ; Dieu khien on/off L2
DKL3 BIT P1.1            ; Dieu khien on/off L3
DKL4 BIT P1.0            ; Dieu khien on/off L4
DK_DATA EQU P0           ; Dieu khien dua du lieu LED 7
                           ; thanh ra P0
PUSH_1 BIT P2.7          ; Phim bam 1
PUSH_2 BIT P2.6          ; Phim bam 2
PUSH_3 BIT P2.5          ; Phim bam 3
PUSH_4 BIT P2.4          ; Phim bam 4
```

```

; ----- * KHAI BAO ORG * -----
ORG 0000H      ; Dia chi ROM bat dau chuong trinh
LJMP MAIN       ; Khoi dong lai toan bo chuong trinh
ORG 0030H      ; Nhay ra khoi vung cac vecto ngat
; ----- * MAIN PROGRAM * -----
MAIN:
; Khoi tao gia tri cac PORT, cac dia chi nho
MOV L1,#01H      ; Khoi tao L1 chua gia tri 01
    MOV L2,#02H      ; Khoi tao L1 chua gia tri 02
    MOV L3,#03H      ; Khoi tao L1 chua gia tri 03
    MOV L4,#04H      ; Khoi tao L1 chua gia tri 04
    MOV DK_DATA,FFH   ; Xoa du lieu trong tai PORT 2
MAIN_LOOP:
    LCALL HIEN THI
    LCALL PHIM_BAM
    SJMP MAIN_LOOP
; Chuong trinh con hien thi len LED 7 thanh
HIEN THI:
LED1:
    MOV A,L1 ; Chuyen gia tri LED 1 vao thanh ghi A
    LCALL CHUYEN_MA ; Gọi chuong trinh chuyen ma
    MOV DK_DATA,A ; Chuyen du lieu hien thi ra PORT 2
    SETB DKL1      ; Bat sang LED 1
    LCALL DELAY     ; DELAY
    CLR DKL1       ; Tat LED 1 de hien thi LED 2
LED2:
    MOV A,L2
    LCALL CHUYEN_MA
    MOV DK_DATA,A
    SETB DKL2
    LCALL DELAY
    CLR DKL2
LED3:
    MOV A,L3
    LCALL CHUYEN_MA
    MOV DK_DATA,A
    SETB DKL3
    LCALL DELAY
    CLR DKL3

```

```
LED4:
```

```
    MOV A, L4
    LCALL CHUYEN_MA
    MOV DK_DATA, A
    SETB DKL4
    LCALL DELAY
    CLR DKL4
    RET
; PROGRAM ----* CHUYEN DOI MA BCD-7 THANH *-----
CHUYEN_MA:
CJNE A, #00H, SO1
MOV A, #11000000B
SJMP END_CM
SO1: CJNE A, #01H, SO2
    MOV A, #11111001B
    SJMP END_CM
SO2: CJNE A, #02H, SO3
    MOV A, #10100100B
    SJMP END_CM
SO3: CJNE A, #03H, SO4
    MOV A, #10110000B
    SJMP END_CM
SO4: CJNE A, #04H, SO5
    MOV A, #10011001B
    SJMP END_CM
SO5: CJNE A, #05H, SO6
    MOV A, #10010010B
    SJMP END_CM
SO6: CJNE A, #06H, SO7
    MOV A, #10000010B
    SJMP END_CM
SO7: CJNE A, #07H, SO8
    MOV A, #11111000B
    SJMP END_CM
SO8: CJNE A, #08H, SO9
    MOV A, #10000000B
    SJMP END_CM
SO9: MOV A, #10010000B
END_CM:
RET           ; Kết thúc chương trình con chuyển mã
```

```

; PROGRAM ----* XU LY PHIM BAM *-----
PHIM_BAM:
    JNB PUSH_1,XU_LY_PHIM_1
    JNB PUSH_2,XU_LY_PHIM_2
    JNB PUSH_3,XU_LY_PHIM_3
    JNB PUSH_4,XU_LY_PHIM_4
    SJMP END_PHIM_BAM

XU_LY_PHIM_1:
    JNB PUSH_1,XU_LY_PHIM_1
    MOV A,L1
    CJNE A,#09H,TANG_L1
    MOV A,#00H
    SJMP END_XU_LY_PHIM_1

TANG_L1:
    INC A
    END_XU_LY_PHIM_1:
    MOV L1,A
    SJMP END_PHIM_BAM

XU_LY_PHIM_2:
    JNB PUSH_2,XU_LY_PHIM_2
    MOV A,L2
    CJNE A,#09H,TANG_L2
    MOV A,#00H
    SJMP END_XU_LY_PHIM_2

TANG_L2:
    INC A
    END_XU_LY_PHIM_2:
    MOV L2,A
    SJMP END_PHIM_BAM

XU_LY_PHIM_3:
    JNB PUSH_3,XU_LY_PHIM_3
    MOV A,L3
    CJNE A,#09H,TANG_L3
    MOV A,#00H
    SJMP END_XU_LY_PHIM_3

TANG_L3:
    INC A
    END_XU_LY_PHIM_3:
    MOV L3,A
    SJMP END_PHIM_BAM

```

```

XU_LY_PHIM_4:
    JNB PUSH_4,XU_LY_PHIM_4
    MOV A,L4
    CJNE A,#09H,TANG_L4
    MOV A,#00H
    SJMP END_XU_LY_PHIM_4
TANG_L4:
    INC A
END_XU_LY_PHIM_4:
    MOV L4,A
END_PHIM_BAM:
    RET
DELAY:
    MOV TMOD,#10H      ; Timer 1, mode 1 (16 bit )
    MOV TH1,#ECH        ; or mov TH1,#high(-5000)
    MOV TL1,#78FH       ; or mov TL1,#low(-5000),
    SETB TR1            ; Cho phép timer chạy
    JNB TF1,$           ; Nhảy tại chỗ, không làm gì
    CLR TR1             ; Nếu timer tràn thì dừng timer
    CLR TF1             ; Xoá cờ tràn
    RET                 ; Kết thúc chương trình con Delay
END

```

*Nạp chương trình và quan sát trạng thái các đèn LED 7 đoạn, nhấn các phím bấm và cho biết kết quả.*

*Thiết kế bộ đếm từ 0000 đến MMDD hiển thị lên 04 LED 7 thanh; trong đó MM là tháng sinh (01÷12) và DD là ngày sinh (01÷31); khởi động bằng phím bấm nối đến P2.4, dừng bộ đếm bằng phím nối đến P2.5, reset bộ đếm bằng phím nối đến P2.6. Thời gian chuyển trạng thái giữa các giá trị đếm là 01s thực hiện bằng Timer 0.*

## BÀI 5:

# NGẮT TRONG VI ĐIỀU KHIỂN 8051

## 1. MỤC ĐÍCH

Bài thực hành nhằm cung cấp cho sinh viên các kỹ năng:

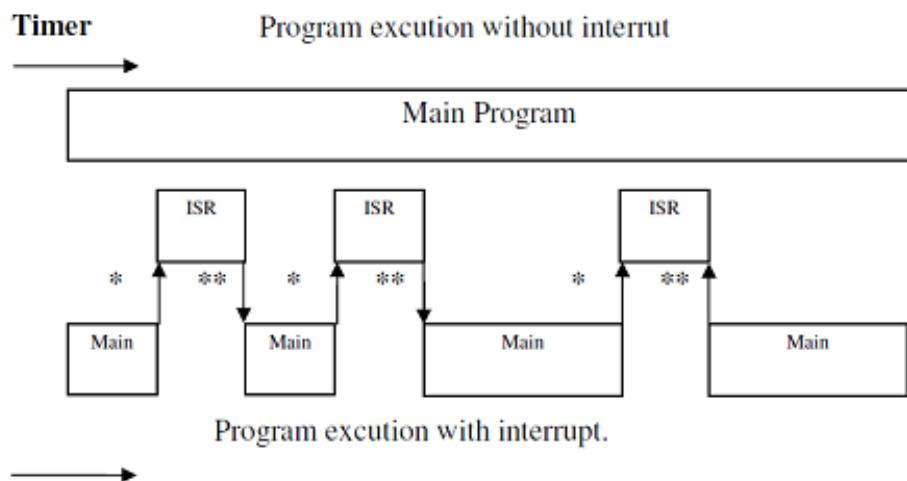
- Khảo sát hoạt động của ngắt trong vi điều khiển 8051;
- Ứng dụng ngắt trong quét hiển thị lên LED 7 thanh và điều chế độ rộng xung PWM.

## 2. TÓM TẮT LÝ THUYẾT

### 2.1. Hoạt động của ngắt trong vi điều khiển 8051

Ngắt (interrupt) là sự xảy ra của một “điều kiện - sự kiện” làm cho chương trình hiện hành bị tạm ngưng trong khi “điều kiện - sự kiện” đó được phục vụ bởi một chương trình khác.

Các ngắt đóng vai trò quan trọng trong việc thiết kế và hiện thực các ứng dụng của bộ vi điều khiển. Các ngắt cho phép hệ thống đáp ứng một sự kiện (nguyên nhân gây ra ngắt) theo một cách không đồng bộ và thực hiện xử lý sự kiện trong khi một chương trình khác đang thực thi. Một hệ thống được điều khiển bởi ngắt cho ta cảm giác đang làm nhiều công việc đồng thời. Tuy vậy CPU vẫn nhiên không thể thực thi nhiều hơn một lệnh ở một thời điểm nhưng CPU có thể ngưng tạm thời việc thực thi một chương trình để thực thi một chương trình khác rồi sau đó quay trở về thực thi tiếp chương trình đang bị tạm ngưng. Trong một hệ thống được điều khiển bởi ngắt, việc ngưng tạm thời nhằm để đáp ứng một sự kiện ngắt xuất hiện không đồng bộ với chương trình chính đang được thực thi (hay nói cách khác là CPU không biết trước là sẽ bị ngắt khi nào).



Hình 5.1: Hoạt động có của vi điều khiển có ngắt và không có ngắt.

Chương trình xử lý một ngắt được gọi là trình phục vụ ngắt ISR (Interrupt Service Routine) hay quản lý ngắt (interrupt handler). ISR được thực thi nhằm đáp ứng

một ngắt và trong trường hợp tổng quát thực hiện việc xuất nhập đối với một thiết bị. Khi một ngắt xuất hiện, việc thực thi chương trình chính tạm thời bị dừng vì CPU thực hiện việc rẽ nhánh đến trình phục vụ ngắt ISR. CPU thực thi ISR để thực hiện một công việc và kết thúc việc thực thi này khi gặp lệnh “quay về từ một trình phục vụ ngắt”; chương trình chính được tiếp tục tại nơi bị tạm dừng. Ta có thể nói chương trình chính được thực thi ở mức nền (base level) còn ISR được thực thi ở mức ngắt (interrupt level).

Một chương trình chính không có ngắt thì chạy liên tục; đối với chương trình có hoạt động ngắt thì cứ khi nào điều kiện ngắt được đảm bảo thì con trỏ lệnh sẽ nhảy tới vị trí của chương trình phục vụ ngắt thực hiện xong chương trình phục vụ ngắt lại quay về đúng chỗ cũ thực hiện tiếp chương trình chính. Ngắt đối với người mới học vi điều khiển là rất khó hiểu, vì đa số các tài liệu đều không giải thích ngắt để làm gì. Có nhiều loại ngắt khác nhau nhưng tất cả đều có chung một đặc điểm, ngắt dùng cho mục đích đa nhiệm (thực hiện nhiều nhiệm vụ).

## 2. 2. Các nguyên nhân ngắt trong 8051

Trong vi điều khiển 8051 có các nguyên nhân gây ra ngắt như sau:

NGUYÊN NHÂN NGẮT	CỜ NGẮT	ĐỊA CHỈ VECTO NGẮT
Reset hệ thống	RST	0000H
Ngắt ngoài 0	IE0	0003H
Bộ định thời 0	TF0	000BH
Ngắt ngoài 1	IE1	0013H
Bộ định thời 1	TF1	001BH
Port nối tiếp	RI hoặc TI	0023H

## 2.3. Các bước lập trình sử dụng ngắt trong 8051

Để sử dụng ngắt chúng ta phải làm các công việc sau:

### B1. Khởi tạo ngắt:

Dùng ngắt nào thì cho phép ngắt đó hoạt động bằng cách gán giá trị tương ứng cho thanh ghi cho phép ngắt IE( Interrupt Enable):

EA	---	ET2	ES	ET1	EX1	EX0	ET0
----	-----	-----	----	-----	-----	-----	-----

Thanh ghi điều khiển các nguồn ngắt IE (0: không cho phép; 1: cho phép)

IE.7 EA Cho phép/ không cho phép toàn cục

IE.6 --- Không sử dụng

IE.5 ET2 Cho phép ngắt do bộ định thời 2

IE.4 ES Cho phép ngắt do port nối tiếp

IE.3 ET1 Cho phép ngắt cho bộ định thời 1

IE.2 EX1 Cho phép ngắt từ bên ngoài (ngắt ngoài 1)

IE.1 EX0 Cho phép ngắt từ bên ngoài (ngắt ngoài 0)

IE.0 ET0 Cho phép ngắt do bộ định thời 0

IE là thanh ghi có thể xử lí từng bit.

Ví dụ : Để cho phép ngắt Timer 1 thì ET1 = 1; không cho phép ngắt Timer 1 thì ET1 = 0.

### B2. Cấu hình cho ngắt

Trong một ngắt nó lại có nhiều chế độ. Ví dụ: với ngắt Timer, chúng ta phải cấu hình cho Timer chạy ở chế độ nào, chế độ Timer hay Counter (tạo trễ thời gian hay đếm sự kiện), chế độ 16 bit, hay 8 bit,... bằng cách gán các giá trị tương ứng cho thanh ghi TMOD (Timer MODE).

TMOD	TÊN BIT	CHỨC NĂNG			
7	GATE	Bit điều khiển Timer 1			
6	C/T	Bit chọn chức năng đếm hoặc định thời			
5	M1	Bit chọn chế độ thứ 1 cho bộ định thời 1			
4	M0	Bit chọn chế độ thứ 2 cho bộ định thời 1			
		M1	M2	Chế độ	Chức năng
		0	0	0	Chế độ định thời 13 bit
		0	1	1	Chế độ định thời 16 bit
		1	0	2	Chế độ tự nạp lại 8 bit
		1	1	3	Chế độ định thời chia sẻ
3	GATE	Bit điều khiển Timer 0			
2	C/T	Bit chọn chức năng đếm hoặc định thời			
1	M1	Bit chọn chế độ thứ 1 cho bộ định thời 0			
0	M0	Bit chọn chế độ thứ 2 cho bộ định thời 0			

Ví dụ: Cấu hình cho bộ định thời 1 chế độ Timer (tạo trễ thời gian), với bộ đếm 8 bit tự động nạp lại (Auto reload): TMOD = 0x20.

### B3. Bắt đầu chương trình có ngắt

- Trước khi bắt đầu cho chạy chương trình ta phải cho phép ngắt toàn cục được xảy ra bằng cách gán EA bằng 1, thì ngắt mới xảy ra.

- Thường thì ngay vào đầu chương trình chúng ta đặt công việc khởi tạo, cấu hình và cho phép kiểm tra ngắt. Ví dụ với bộ định thời Timer ta gán các giá trị phù hợp cho thanh ghi TCON (Timer CONtrol).

TCON		ĐIỀU KHIỂN BỘ ĐỊNH THỜI
TCON.7	TF1	Cờ tràn của bộ định thời 1. Cờ này được set bởi phần cứng khi có tràn, được xóa bởi phần mềm hoặc phần cứng khi bộ xử lý trả đến chương trình ngắn.
TCON.6	TR1	Bit điều khiển hoạt động của bộ định thời 1. Bit này được set hoặc xóa bởi phần mềm để điều khiển bộ định thời hoạt động hay ngưng.
TCON.5	TF0	Cờ tràn của bộ định thời 1.
TCON.4	TR0	Bit điều khiển hoạt động của bộ định thời 1.
TCON.3	IE1	Cờ ngắn ngoài 1 (kích khởi cảnh). Cờ này được set bởi phần cứng khi có cảnh âm (xuống) xuất hiện trên chân INT1, được xóa bởi phần mềm hoặc phần cứng khi CPU trả đến trình phục vụ ngắn.
TCON.2	IT1	Cờ ngắn ngoài 1 (kích khởi cảnh hoặc mức). Cờ này được set bởi phần cứng khi có cảnh âm (xuống) hoặc mức thấp xuất hiện trên chân ngắn ngoài.
TCON.1	IE1	Cờ ngắn ngoài 0 (kích khởi cảnh).
TCON.0	IT0	Cờ ngắn ngoài 1 (kích khởi cảnh hoặc mức).

Ví dụ để chạy bộ định thời Timer 1 ta đặt: TR1 = 0; TR1 (Timer Run 1) hoặc TCON=0xxx;

## 2.2. Pulse Width Modulation (PWM)

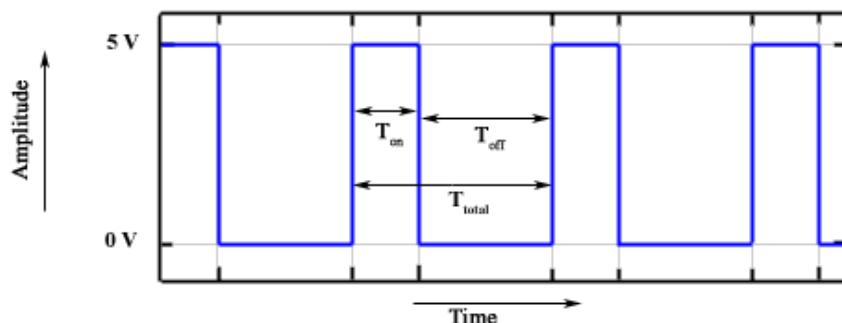
### 2.2.1. Khái niệm về điều chế độ rộng xung PWM

Pulse Width Modulation (PWM) là một trong những kỹ thuật thường được sử dụng trong các hệ thống điều khiển ngày nay. PWM được sử dụng rộng rãi trong các ứng dụng điều khiển như: điều khiển tốc độ, điều khiển nguồn điện, đo lường và truyền thông.

### 2.2.2. Nguyên tắc cơ bản của PWM

Điều chế độ rộng xung PWM là tạo ra một xung vuông có chu kỳ có thể thay đổi được từ đó làm cho điện áp ra thay đổi như là một giá trị trung bình của dạng sóng

Có thể biểu diễn nguyên tắc cơ bản của phương pháp điều chế độ rộng xung PWM thông qua các công thức sau:



Xét một sóng vuông như hình trên:

$T_{on}$  là khoảng thời gian đầu ra ở mức cao,  $T_{off}$  là khoảng thời gian đầu ra ở mức thấp,  $T_{total}$  là khoảng thời gian đạt được dạng sóng như vậy (chu kỳ).

$$T_{total} = T_{on} + T_{off}$$

Nhiệm vụ của một chu kỳ sóng vuông được định nghĩa là:

$$D = T_{on} / (T_{on} + T_{off}) = T_{on} / T_{total}$$

Điện áp đầu ra được tính như sau:

$$V_{out} = D \cdot V_{in} = V_{in} \cdot T_{on} / T_{total}$$

Ý tưởng cơ bản nhằm tạo ra PWM trên vi điều khiển 8051 là việc thiết lập mức cao và mức thấp tại các cổng IO trong những khoảng thời gian xác định. Như đã nói trong phần giới thiệu về PWM rằng bằng cách thay đổi thời gian  $T_{on}$ , chúng ta có thể thay đổi độ rộng của sóng vuông trong một khoảng thời gian của sóng vuông đó.

Chúng ta sẽ sử dụng Timer0 ở chế độ 0, giá trị cao và thấp cấp sẽ được đặt sao cho có thể tạo nên một khoảng thời gian xác định. Nếu ở byte cao chúng ta đặt một giá trị X trong TH0 và các byte thấp TL0 sẽ được tải với 255-X vì vậy mà tổng số còn lại là 255.

### 2.2.3. Thiết kế phần mềm điều chế độ rộng xung

+ Cách 1: Sử dụng các hàm tạo trễ thời gian để tạo ra 01 xung ở 01 chân của vi điều khiển.

Tuy nhiên khi dùng các hàm này trong thời gian có xung lên mức 1 (5V) và thời gian xuống mức 0 (0V) bộ vi điều khiển không làm được các công việc khác.

Nếu như chúng ta có nhu cầu cần 2 bộ phát xung PWM trên hai chân của vi điều khiển thì phương pháp này sẽ gặp nhiều khó khăn, để khắc phục vấn đề này chúng ta sử dụng cách sau.

+ Cách 2: Dùng ngắt Timer của bộ vi điều khiển.

## 3. NỘI DUNG THỰC HÀNH

Viết chương trình điều chế độ rộng xung PWM trên chân P3.0 của vi điều khiển 8051.

```
#include <sfr51.inc>

PWM_PIN BIT      P3.0          ; Pin PWM

ORG 0000H      ; Dia chi ROM bat dau chuong trinh
LJMP MAIN      ; Khoi dong lai toan bo chuong trinh
ORG 000BH      ; Dia chi ngat cua bo dinh thoi 0
LJMP TIMER_0_INTERRUPT ; Dia chi cua
                        ; chuong trinh PWM
ORG 0030H      ; Nhay ra khoi vung cac vecto ngat
```

```

MAIN:
; Khoi tao ngat cua bo dinh thoi 0
MOV TMOD,#00000000B      ; Timer 0 che do 13 bit
MOV TCON,#00H              ; Khoi tao thanh ghi TCON
SETB TCON.4                ; Cho phep Timer 0 hoat dong
MOV IE,#10000010B          ; Cho phep ngat bo dinh thoi 0
MOV IP,#00000010B          ; Dat muc uu tien cho Timer 0
MOV R7,#125

HERE: LJMP HERE

TIMER_0_INTERRUPT:
JB F0, HIGH_DONE           ; If F0 flag is set then we
                             ; just finished
                             ; the high section of the
LOW_DONE:                  ; cycle so Jump to HIGH_DONE
SETB F0                     ; Make F0=1 to indicate
                             ; start of high section
SETB PWM_PIN                ; Make PWM output pin High
MOV TH0,R7                  ; Load high byte of timer
                             ; with R7 (pulse width
                             ; control value)
CLR TF0                     ; Clear the Timer 0
                             ; interrupt flag
RETI                         ; Return from Interrupt to where

HIGH_DONE:
CLR F0                      ; Make F0=0 to indicate
                             ; start of low section
CLR PWM_PIN                 ; Make PWM output pin low
MOV A,#0FFH                  ; Move FFH (255) to A
CLR C                        ; Clear C (the carry bit) so
                             ; it does not affect the
                             ; subtraction
SUBB A, R7                  ; Subtract R7 from A. A =
                             ; 255 - R7.
MOV TH0, A                  ; so the value loaded into
                             ; TH0 + R7 = 255
CLR TF0                     ; Clear the Timer 0
; interrupt flag
RETI
END

```

*Thay đổi thời gian  $T_{on}$  và  $T_{off}$  và theo dõi sự thay đổi của độ rộng xung trên đầu ra của chân P3.0.*

*- Viết chương trình điều chỉnh độ rộng xung PWM trên chân P3.0 của vi điều khiển 8051 sử dụng 02 phím nối vào P2.3 và P2.3 là 2 phím tăng và giảm, thay đổi thời gian  $T_{on}$  và  $T_{off}$  và theo dõi sự thay đổi của độ rộng xung trên đầu ra của chân P3.0.*

*- Viết chương trình cho đồng hồ số hiển thị giờ, phút hiển thị lên 04 LED 7 thanh sử dụng ngắt Timer 1 để tạo trễ thời gian và ngắt Timer 0 để hiển thị các giá trị lên LED 7 thanh.*