

## CHƯƠNG 3

# NGÔN NGỮ LẬP TRÌNH VI ĐIỀU KHIỂN

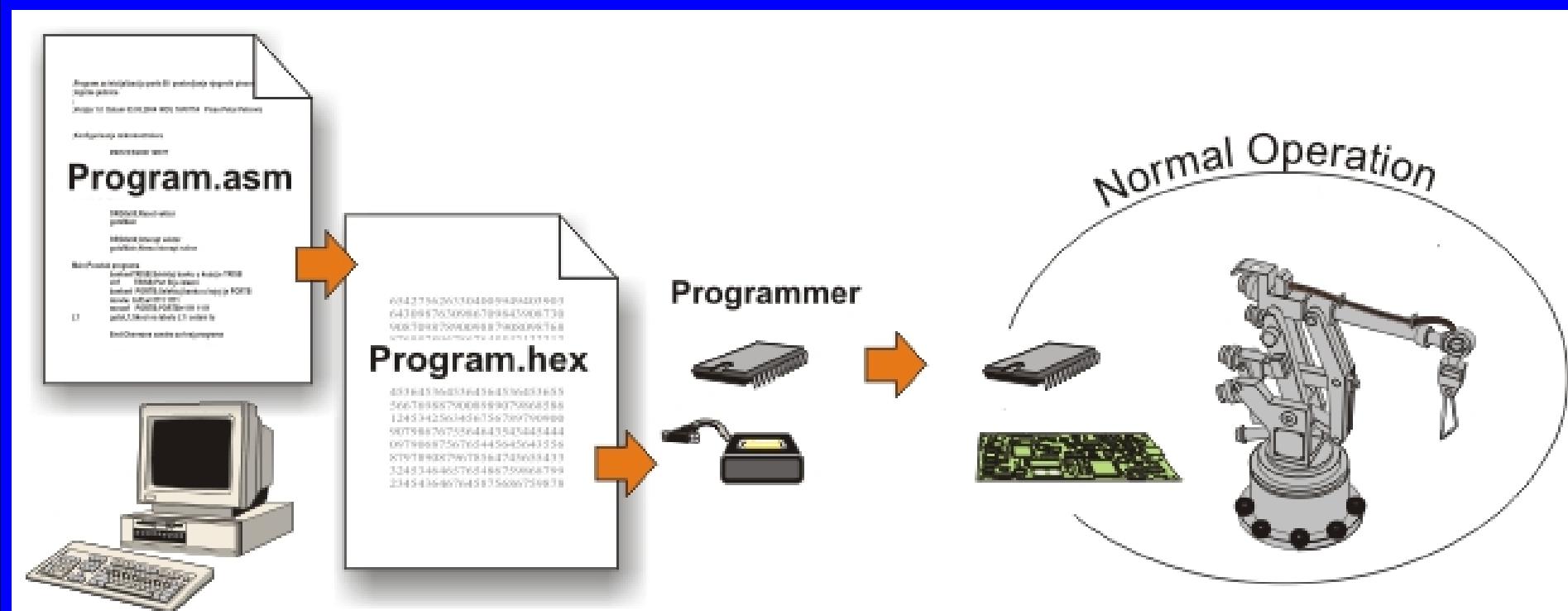
# LẬP TRÌNH CHO VI ĐIỀU KHIỂN

Nội dung gồm có:

- Tổng quan lập trình cho vi điều khiển
- Ngôn ngữ lập trình C cơ bản
- Công cụ phát triển PICmicro.
- Ví dụ làm thế nào để viết một chương trình

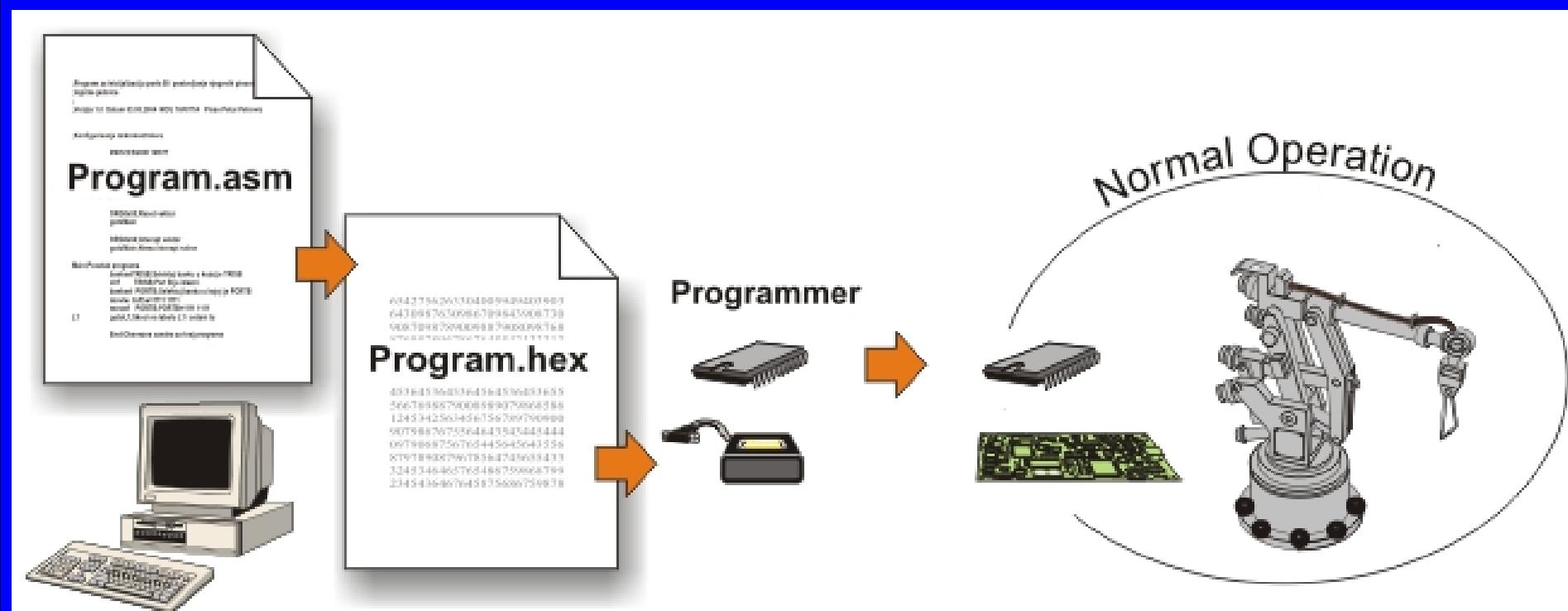
# TỔNG QUAN LẬP TRÌNH CHO VI ĐIỀU KHIỂN

- **Vi điều khiển truyền thông với con người thông qua ngôn ngữ lập trình**
  - **Ngôn ngữ lập trình: cấp thấp hoặc cấp cao**
  - **Ngôn ngữ lập trình được sử dụng trong bài học này là ngôn ngữ C.**



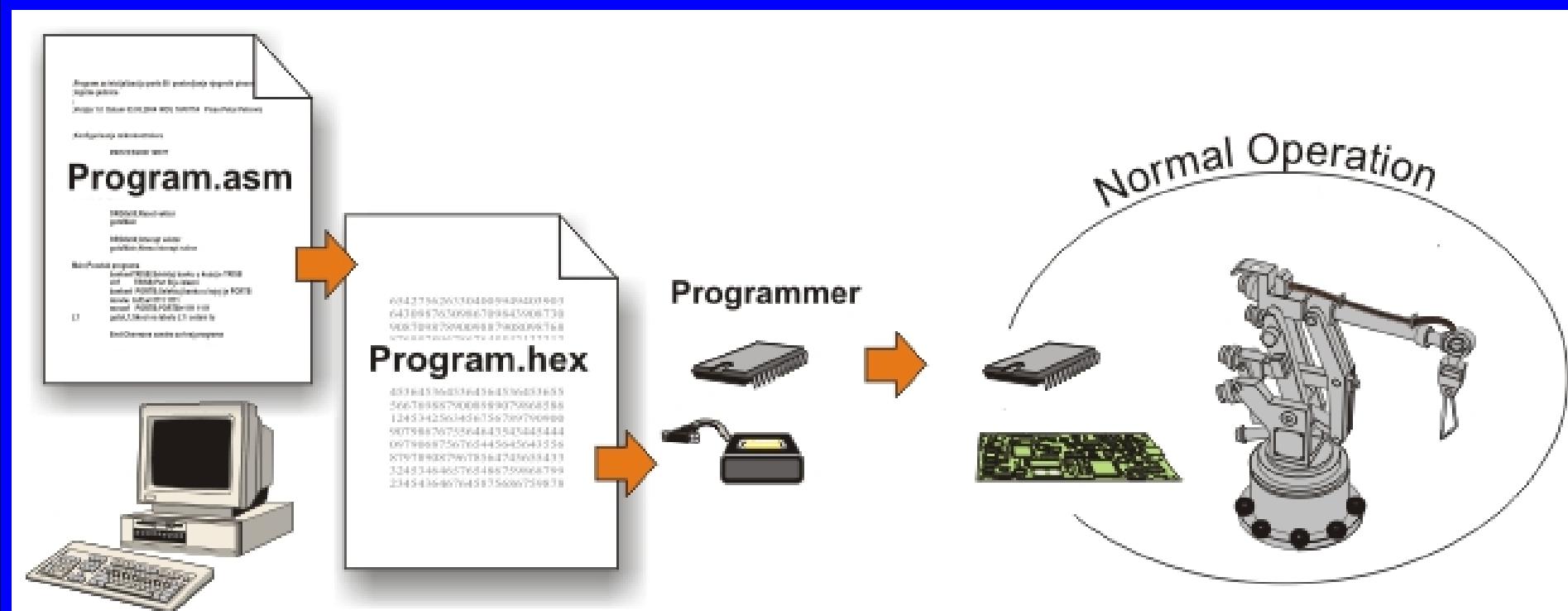
# TỔNG QUAN LẬP TRÌNH CHO VI ĐIỀU KHIỂN

- Hợp ngữ là một tập các qui luật (lệnh) được dùng để viết chương trình cho vi điều khiển
- Để vi điều khiển có thể hiểu một chương trình được viết bằng hợp ngữ thì nó cần phải được biên dịch thành ngôn ngữ máy.



# TỔNG QUAN LẬP TRÌNH CHO VI ĐIỀU KHIỂN

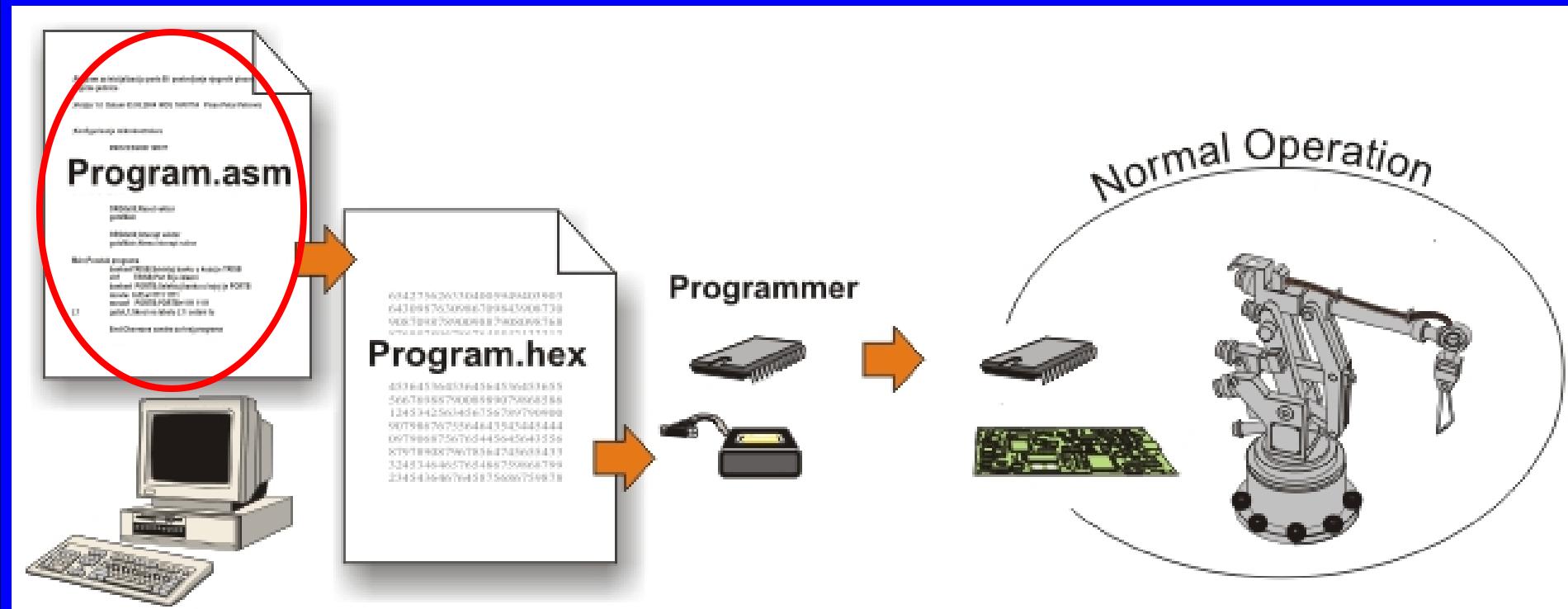
- **Assembly:** là một tập các qui luật (lệnh) được dùng để viết chương trình cho vi điều khiển
- **Assembler:** là một chương trình trên máy tính dùng để dịch các phát biểu hợp ngữ sang ngôn ngữ của các con số 0 hoặc 1.



# TỔNG QUAN LẬP TRÌNH CHO VI ĐIỀU KHIỂN

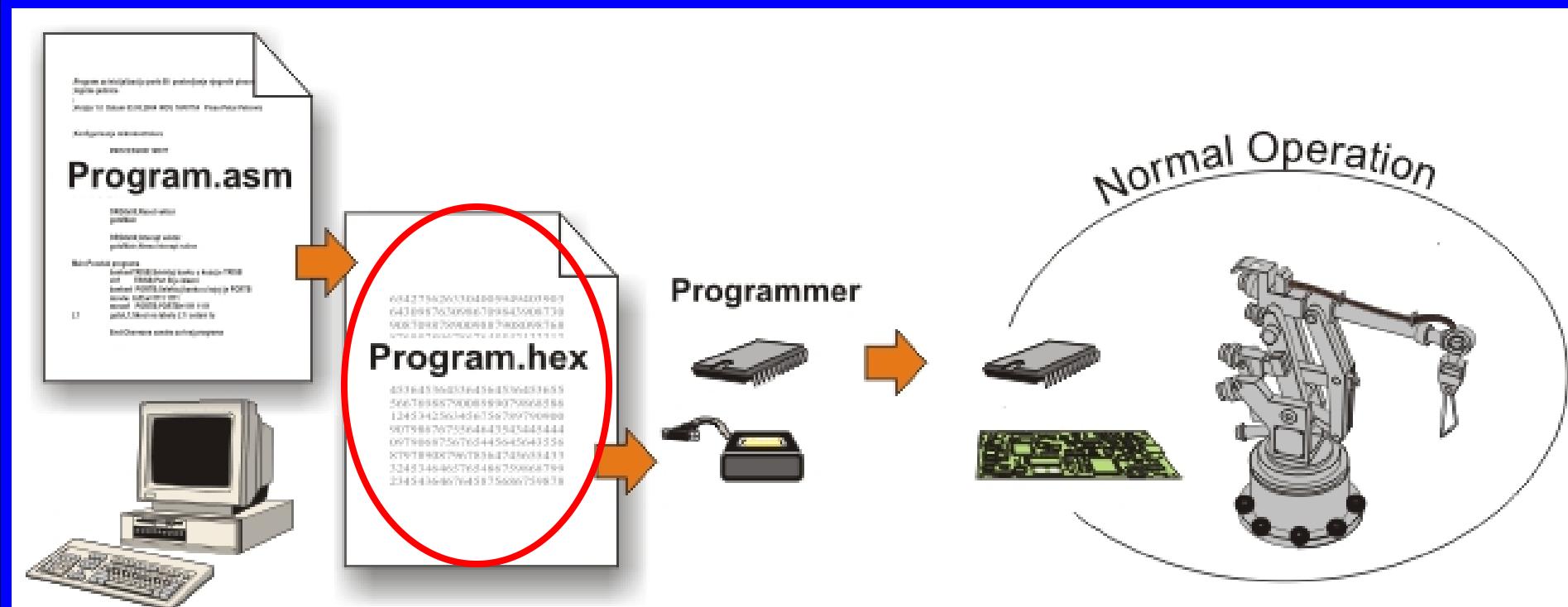
## ➤ Chương trình (Program) / Mã nguồn (Source Code):

- Là tập tin dữ liệu được lưu trên máy tính
- Được thể hiện bằng hợp ngữ (hoặc các ngôn ngữ khác: ngôn ngữ C)
- Con người có thể đọc hiểu (vi điều khiển không hiểu)
- Định dạng tập tin là .ASM và .C (ngôn ngữ C).



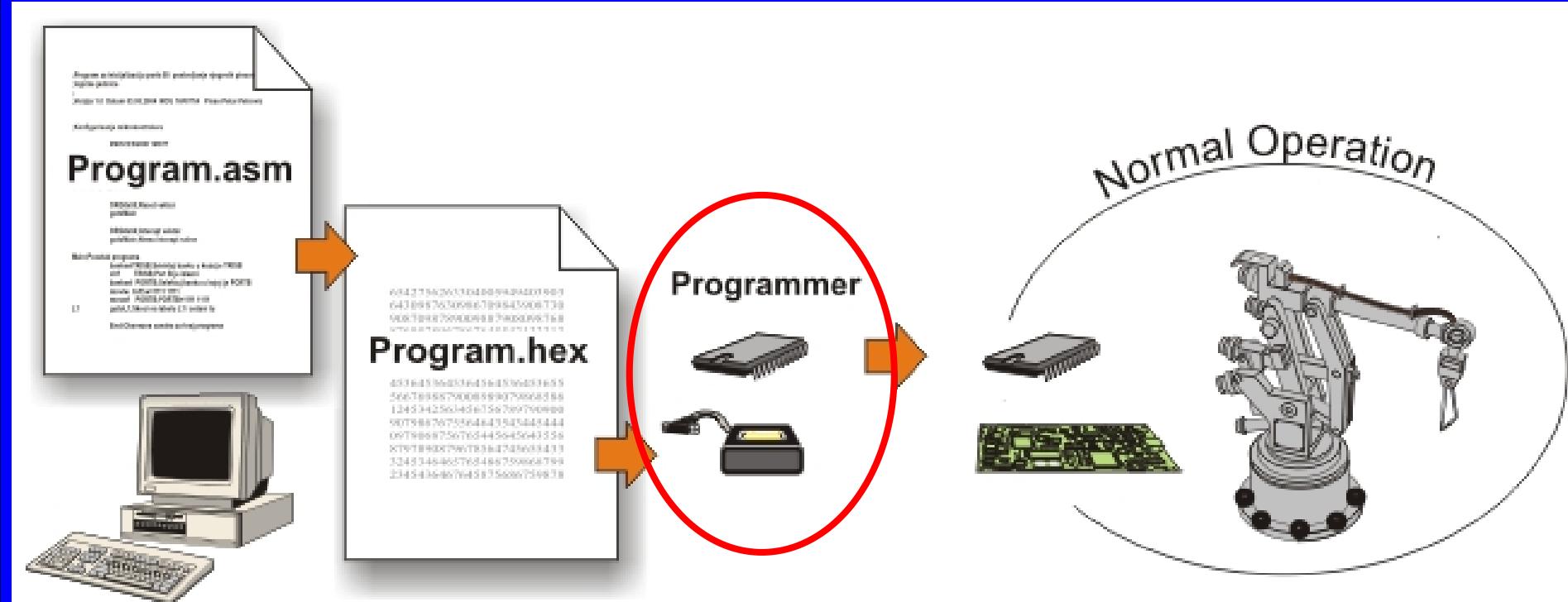
## ➤ Thực thi (Executive) / Mã máy (Machine Code):

- Là tập tin dữ liệu được lưu trên máy tính hoặc nạp PIC
- Được thể hiện bằng các con số 0 hoặc 1
- Vi điều khiển có thể hiểu (con người khó có thể hiểu)
- Định dạng tập tin là .HEX (Ví dụ: Program.hex).



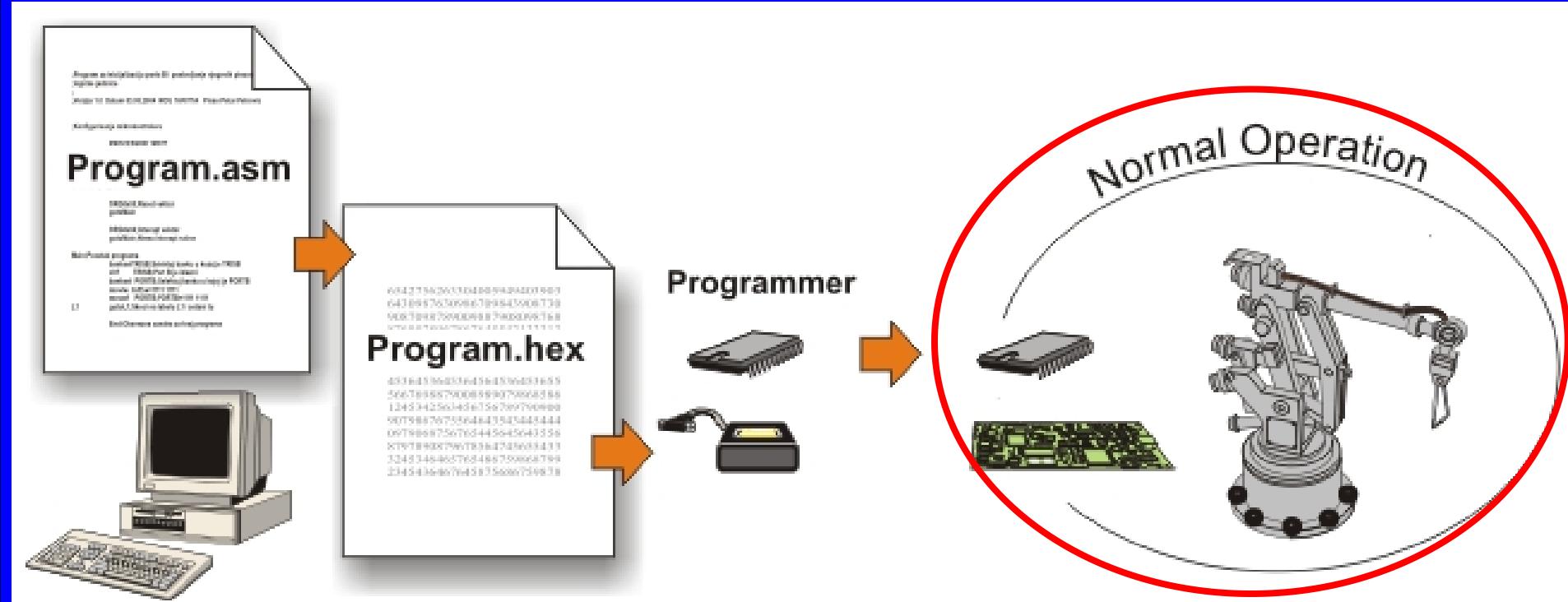
## TỔNG QUAN LẬP TRÌNH CHO VI ĐIỀU KHIỂN

- Chương trình sau khi biên dịch sẽ được nạp vào vi điều khiển bằng các bộ lập trình (Programmer) chuyên dụng dành cho PIC.



# TỔNG QUAN LẬP TRÌNH CHO VI ĐIỀU KHIỂN

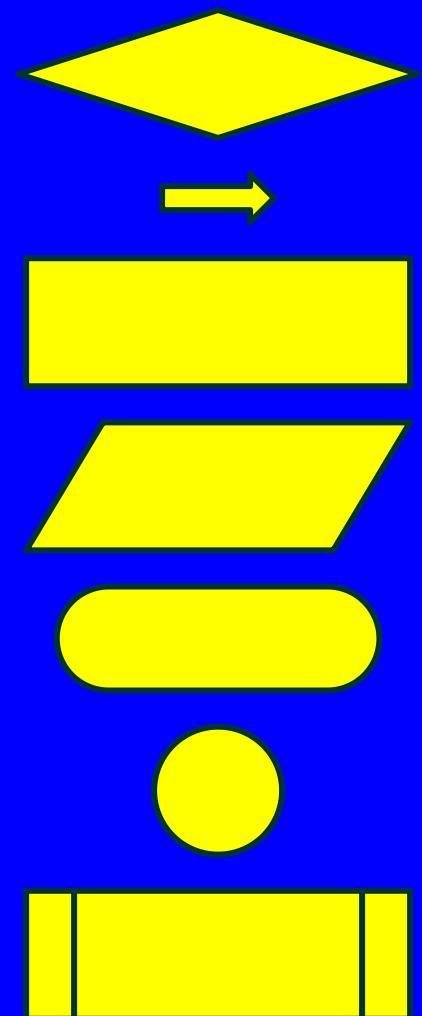
- **Vi điều khiển sau khi được lập trình sẽ được gắn vào bo mạch chủ để điều khiển toàn hệ thống hoạt động theo chương trình mà không cần sự can thiệp của con người.**



# CÁC THÀNH PHẦN CỦA HỢP NGỮ

## ➤ Các ký hiệu thường dùng cho việc lập lưu đồ giải thuật

- Khối quyết định (so sánh, kiểm tra)
- Đường đi của chương trình
- Khối xử lý
- Khối xuất / nhập
- Bắt đầu / Kết thúc chương trình
- Kết nối qua trang chương trình
- Xử lý chương trình con.



# VÍ DỤ MỘT CHƯƠNG TRÌNH ASSEMBLY

Header

```
;Program to initialize port B and set its pins to logic one (1)
```

Directive

```
;Version: 1.0 Date: 03.05.2007 MCU: 16F887 Programmer: John Smith
```

Comment

```
;Configuring microcontroller
```

```
PROCESSOR 16f887
```

```
include "pic16f887inc"
```

Operand

```
;Start of program
```

```
ORG 0x00 ;Reset vector
```

```
goto Main
```

Label

```
Main
```

```
;Start of program
```

```
banksel
```

```
TRISB
```

```
;Select bank containing TRISB
```

```
clrf
```

```
TRISB
```

```
;Port B is configured as output
```

```
banksel
```

```
PORTB
```

```
;Select bank containing PORTB
```

```
movlw
```

```
0xff
```

```
;w=1111 1111
```

```
movwf
```

```
PORTB
```

```
;PORTB=1111 1111
```

```
goto L1
```

```
;Go to label L1 or remain here
```

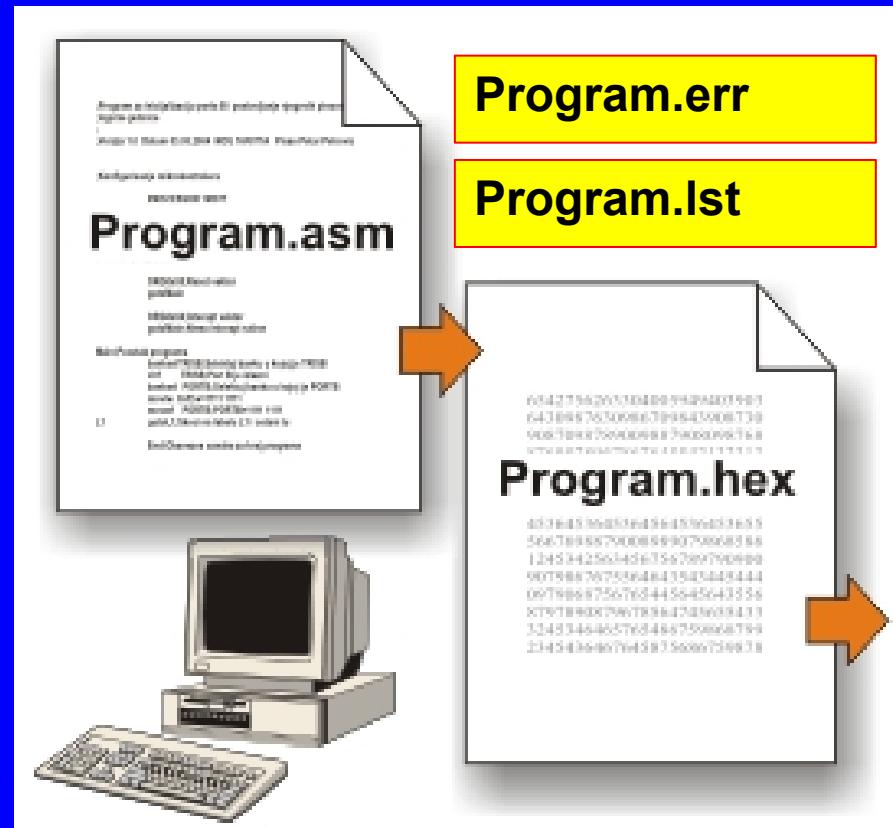
Instruction

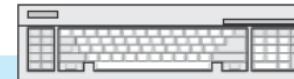
```
End
```

```
;End of program
```

# VÍ DỤ MỘT CHƯƠNG TRÌNH ASSEMBLY

- **Viết code → Tập tin chương trình (.ASM)**
- **Biên dịch →**
  - **Tập tin thực thi (.HEX)**
    - Chứa chương trình sau biên dịch để nạp vào vi điều khiển
  - **Tập tin báo lỗi (.ERR)**
    - Chứa thông báo các lỗi (cú pháp) xảy ra trong quá trình viết code
  - **Tập tin liệt kê (.LST)**
    - Chứa các thông tin về vị trí các lệnh, biến trong bộ nhớ. Cuối mỗi tập tin này có bảng liệt kê các ký hiệu được sử dụng trong chương trình





```
void main() {  
  
    TRISB = 0;           // All port B pins are configured as  
                       // outputs  
    PORTB = 0b01010101; // Logic state on port B pins  
}
```

Program written in C

; ADDRESS	OPCODE	ASM
\$0000	\$2804	GOTO _main
\$0004	\$	_main:
;Test.c,1 ::	void main() {	
;Test.c,3 ::	TRISB = 0;	// All port B pins
\$0004	\$1303	BCF STATUS, RP1
\$0005	\$1683	BSF STATUS, RP0
\$0006	\$0186	CLRF TRISB, 1
;Test.c,4 ::	PORTB = 0b01010101;	// Logic state
\$0007	\$3055	MOVLW 85
\$0008	\$1283	BCF STATUS, RP0
\$0009	\$0086	MOVWF PORTB
;Test.c,5 ::	}	
\$000A	\$280A	GOTO \$

Compiled Program

```
:100000000428FF3FFF3FFF3F03138316860155304F  
:10001000831286000A28FF3FFF3FFF3FFF3FFF3F5D  
:04400E00F22FFFFF8F  
:00000001FF
```

Executable Code of the program (HEX code)



Ví dụ minh họa một chương trình đơn giản viết bằng C và biên dịch thành mã máy

# Ngôn ngữ lập trình C cơ bản

- Xem lại các kiến thức, kỹ năng về lập trình C cơ bản

# \* Các kiểu dữ liệu trong ngôn ngữ C

Có vài kiểu dữ liệu có thể sử dụng trong ngôn ngữ lập trình C. Bảng sau trình bày phạm vi dữ liệu của các kiểu dữ liệu này.

Data type	Description	Size (# of bits)	Range of values
char	Character	8	0 to 255
int	Integer	16	-32768 to 32767
float	Floating point	32	$\pm 1.17549435082 \cdot 10^{-38}$ to $\pm 6.80564774407 \cdot 10^{38}$
double	Double precision floating point	32	from $\pm 1.17549435082 \cdot 10^{-38}$ to $\pm 6.80564774407 \cdot 10^{38}$

Bằng cách thêm tiếp đầu ngữ (prefix) vào bất kỳ kiểu dữ liệu nào, phạm vi giá trị có thể của nó sẽ thay đổi dựa trên số lượng byte nhớ cần thiết.

Data type	Data Type With Prefix	Size (# of bits)	Range
char	signed char	8	-128 to 128
int	unsigned int	16	0 to 65535
	short int	8	0 to 255
	signed short int	8	-128 to 127
	long int	32	0 to 4294967295
	signed long int	32	-2147483648 to 2147483647

## \* Variables (biến)

Bất kỳ số nào thay đổi giá trị của nó trong suốt chương trình hoạt động được gọi là biến.

Ví dụ trong phép cộng 2 số (*number 1* và *number 2*), và kết quả là tổng (*sum*) của 2 số. Như vậy, *number 1*, *number 2* và *sum* là các biến.

### Gán biến

- + Tên biến có thể bao gồm các ký tự từ A-Z (a-z), các số từ 0-9 và ký tự gạch dưới ‘\_’. Trình biên dịch phân biệt rõ ràng giữa chữ hoa và chữ thường. Tên hàm và biến thường sử dụng các ký tự chữ thường, còn tên hằng số thường sử dụng ký tự chữ hoa.
- + Tên biến không được bắt đầu bằng chữ số
- + Một số từ khóa không được sử dụng làm tên biến vì nó đã được sử dụng bởi trình biên dịch. Trình biên dịch cho C có tổng cộng 33 từ khóa.

absolute	data	if	return	typedef
asm	default	inline	rx	typeid
at	delete	int	sfr	typename
auto	do	io	short	union
bit	double	long	signed	unsigned
bool	else	mutable	sizeof	using
break	enum	namespace	static	virtual
case	explicit	operator	struct	void
catch	extern	org	switch	volatile
char	false	pascal	template	while
class	float	private	this	
code	for	protected	throw	
const	friend	public	true	
continue	goto	register	try	

*Bảng từ khóa*

## \* Constant (Hằng số)

Hằng số là một số hay một ký hiệu có giá trị cố định và không thể thay đổi trong suốt chương trình. Khác với biến, hằng số được lưu trong bộ nhớ chương trình của vi điều khiển để tiết kiệm bộ nhớ của RAM. Trình biên dịch nhận diện chúng dựa vào tên và prefix *const*

# **Integer constant (Hằng số nguyên)**

Hằng số nguyên có thể là thập phân, thập lục phân, bát phân hoặc nhị phân. Trình biên dịch nhận diện định dạng của chúng dựa trên prefix kèm theo. Nếu không có prefix, mặc định đó là số thập phân. Kiểu của một hằng số được nhận diện tự động dựa trên giá trị của chúng.

Format	Prefix	Example
Decimal		const MAX = 100
Hexadecimal	0x or 0X	const MAX = 0xFF
Octal	0	const MAX = 016
Binary	0b or 0B	const MAX = 0b11011101

```
const MINIMUM = -100; // Declare constant MINIMUM
```

## *Floating point constant (Hằng số thực)*

```
const T_MAX = 32.60; // Declare temperature T_MAX  
const T_MAX = 3.260E1; // Declare the same constant T_MAX
```

## *Character constant (Hằng số kiểu ký tự)*

```
const I_CLASS = 'A'; // Declare constant I_CLASS  
const II_CLASS = 'B'; // Declare constant II_CLASS
```

## *String constant (Hằng số kiểu chuỗi)*

```
const Message_1 = "Press the START button"; // Message 1 for LCD  
const Message_2 = "Press the RIGHT button"; // Message 2 for LCD  
const Message_3 = "Press the LEFT button"; // Message 3 for LCD
```

## *Enumerated constant (Hằng số kiểu liệt kê)*

```
enum MOTORS {UP, DOWN, LEFT, RIGHT}; // Declare constant MOTORS
```

# \* Operator and operation (Toán tử và các phép toán)

Có hơn 40 phép toán có sẵn trong C, nhưng hầu như chỉ có 10-15 phép toán được sử dụng phổ biến. Các phép toán thực hiện trên 1 hoặc nhiều ô nhớ giữa các biến và hằng số.

## Các phép toán số học

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Reminder

## Các phép gán

Operator	Example	
	Expression	Equivalent
+ =	$a += 8$	$a = a + 8$
- =	$a -= 8$	$a = a - 8$
* =	$a *= 8$	$a = a * 8$
/ =	$a /= 8$	$a = a / 8$
% =	$a %= 8$	$a = a \% 8$

# Tăng và giảm

Tăng và giảm 1 đơn vị sử dụng toán tử ‘++’ và ‘--’. Toán tử này có thể đặt trước hoặc sau biến cần tăng giảm. Trong trường hợp đặt trước biến (++x hoặc --x), biến x sẽ được tăng hoặc giảm 1 đơn vị trước và kết quả đó mới được sử dụng trong biểu thức. Ngược lại, nếu đặt sau biến (x++ hoặc x--), giá trị của biến sẽ được sử dụng trong biểu thức trước, sau đó mới tăng hoặc giảm 1 đơn vị.

Operator	Example	Description
++	++a	Variable "a" is incremented by 1
	a++	
--	--b	Variable "b" is incremented by 1
	b--	

# Toán tử quan hệ

Các toán tử quan hệ được sử dụng trong việc so sánh giữa 2 biến, có thể là số nguyên hoặc số thực. Nếu kết quả của biểu thức là đúng, trả về 1. Nếu kết quả của biểu thức là sai, trả về 0.

Operator	Meaning	Example	Truth condition
>	is greater than	$b > a$	if <b>b</b> is greater than <b>a</b>
$\geq$	is greater than or equal to	$a \geq 5$	If <b>a</b> is greater than or equal to <b>5</b>
<	is less than	$a < b$	if <b>a</b> is less than <b>b</b>
$\leq$	is less than or equal to	$a \leq b$	if <b>a</b> is less than or equal to <b>b</b>
$=$	is equal to	$a == 6$	if <b>a</b> is equal to <b>6</b>
$!=$	is not equal to	$a != b$	if <b>a</b> is not equal to <b>b</b>

# Phép toán với bit

Khác với các phép toán luận lý thực hiện với các biến, phép toán bit thao tác trên các bit đơn của biến.

Operand	Meaning	Example	Result	
<code>~</code>	Bitwise complement	$a = \sim b$	$b = 5$	$a = -5$
<code>&lt;&lt;</code>	Shift left	$a = b << 2$	$b = 11110011$	$a = 11001100$
<code>&gt;&gt;</code>	Shift right	$a = b >> 3$	$b = 11110011$	$a = 00011110$
<code>&amp;</code>	Bitwise AND	$c = a \& b$	$a = 11100011$ $b = 11001100$	$c = 11000000$
<code> </code>	Bitwise OR	$c = a   b$	$a = 11100011$ $b = 11001100$	$c = 11101111$
<code>^</code>	Bitwise EXOR	$c = a ^ b$	$a = 11100011$ $b = 11001100$	$c = 00101111$

# Phép toán luận lý

Có 3 kiểu phép toán luận lý trong ngôn ngữ C, đó là AND, OR và NOT. Phép toán logic trả về kết quả true (1) nếu kết quả của biểu thức khác 0, trả về kết quả false (0) nếu kết quả của biểu thức bằng 0.

Operator	Logical AND		
	Operand1	Operand2	Result
&&	0	0	0
	0	1	0
	1	0	0
	1	1	1

Operator	Logical OR		
	Operand1	Operand2	Result
	0	0	0
	0	1	1
	1	0	1
	1	1	1

Operator	Logical NOT	
	Operand1	Result
!	0	1
	1	0

# Cách sử dụng các phép toán

Ngoại trừ các phép gán, hai toán tử không được viết liền kề nhau.

```
x*%12; // such expression will generate an error
```

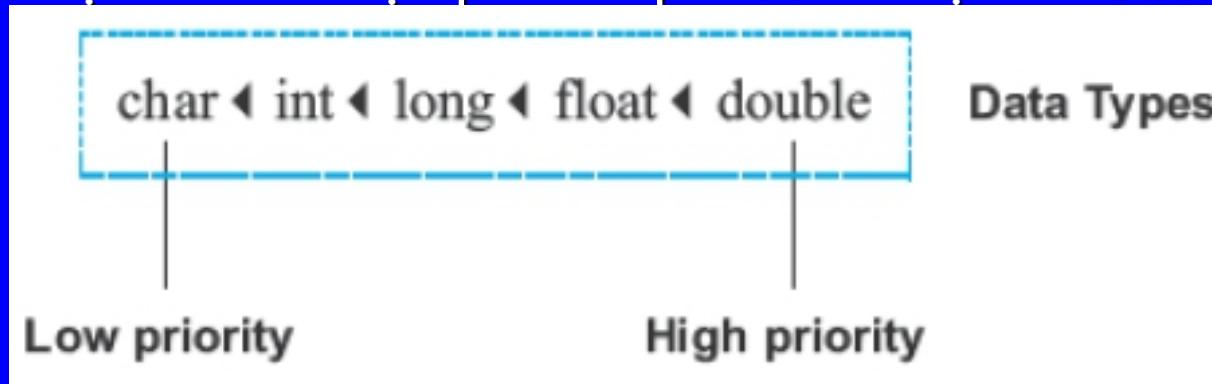
Các phép toán được nhóm lại với nhau sử dụng dấu ngoặc đơn (). Biểu thức trong ngoặc được tính toán trước. Có thể lồng nhiều ngoặc đơn với nhau nếu cần thiết. Mỗi phép toán đều có tính ưu tiên theo thứ tự

Priority	Operators	Associativity
High	() [] -> .	from left to right
	! ~ ++ -- +(unary) -(unary) *Pointer &Pointer	from right to left
	* / %	from left to right

	<code>+ -</code>	from left to right
	<code>&lt; &gt;</code>	from left to right
	<code>&lt; &lt;= &gt; &gt;=</code>	from left to right
	<code>== !=</code>	from left to right
	<code>&amp;</code>	from left to right
	<code>^</code>	from left to right
	<code> </code>	from left to right
	<code>&amp;&amp;</code>	from left to right
	<code>  </code>	from right to left
	<code>?:</code>	from right to left
Low	<code>= += -= *= /= /= &amp;= ^=  = &lt;= &gt;=</code>	from left to right

## \* Biến đổi kiểu dữ liệu

Các kiểu dữ liệu chính được phân cấp theo thứ tự như sau:



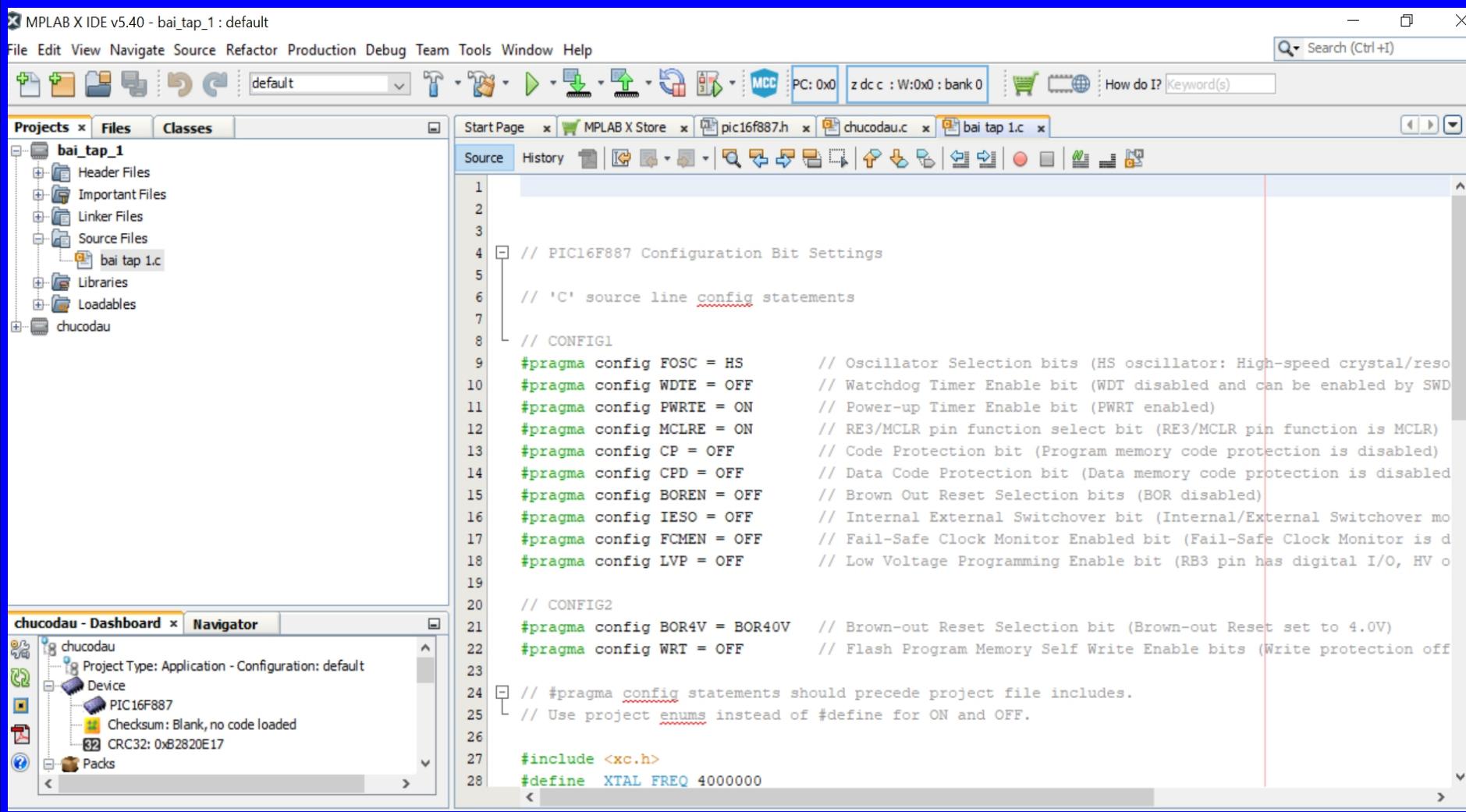
Nếu 2 đại lượng có kiểu khác nhau được sử dụng trong cùng một biểu thức, đại lượng có độ ưu tiên thấp hơn sẽ tự động biến đổi thành kiểu của đại lượng có độ ưu tiên cao hơn.

- Nếu đại lượng có độ ưu tiên cao nhất là kiểu *double*, thì tất cả các đại lượng khác trong biểu thức và kết quả sẽ tự động được biến đổi sang kiểu *double*.
- Nếu đại lượng có độ ưu tiên cao nhất là kiểu *long*, thì tất cả các đại lượng khác trong biểu thức và kết quả sẽ tự động được biến đổi sang kiểu *long*.
- Nếu các đại lượng có kiểu *long* hay *char*, thì tất cả các đại lượng khác trong biểu thức và kết quả sẽ tự động được biến đổi sang kiểu *int*.

Trong phép gán, kiểu của biến bên vế phải sẽ được biến đổi sang kiểu của biến bên vế trái.

# CÔNG CỤ PHÁT TRIỂN PICmicro

## MPLABX IDE



The screenshot shows the MPLAB X IDE interface with the following details:

- Title Bar:** MPLAB X IDE v5.40 - bai\_tap\_1 : default
- Menu Bar:** File Edit View Navigate Source Refactor Production Debug Team Tools Window Help
- Toolbar:** Includes icons for New, Open, Save, Build, Run, and MCC.
- Status Bar:** PC: 0x0 z dc c : W:0x0 : bank 0
- Search Bar:** Search (Ctrl+I)
- Project Explorer:** Projects x Files Classes. The project "bai\_tap\_1" contains:
  - Header Files
  - Important Files
  - Linker Files
  - Source Files
    - bai\_tap\_1.c
  - Libraries
  - Loadables
- Code Editor:** Shows the source code for "bai\_tap\_1.c" which includes configuration pragmas for PIC16F887.
- Navigator:** chucodau - Dashboard x Navigator. It shows the project structure and device settings for PIC16F887.

```
1
2
3
4 // PIC16F887 Configuration Bit Settings
5
6 // 'C' source line config statements
7
8 // CONFIG1
9 #pragma config FOSC = HS           // Oscillator Selection bits (HS oscillator: High-speed crystal/reso
10 #pragma config WDTE = OFF          // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWD)
11 #pragma config PWRT = ON           // Power-up Timer Enable bit (PWRT enabled)
12 #pragma config MCLRE = ON          // RE3/MCLR pin function select bit (RE3/MCLR pin function is MCLR)
13 #pragma config CP = OFF            // Code Protection bit (Program memory code protection is disabled)
14 #pragma config CPD = OFF           // Data Code Protection bit (Data memory code protection is disabled)
15 #pragma config BOREN = OFF          // Brown Out Reset Selection bits (BOR disabled)
16 #pragma config IESO = OFF           // Internal External Switchover bit (Internal/External Switchover mo
17 #pragma config FCMEN = OFF          // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is d
18 #pragma config LVP = OFF            // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV o
19
20 // CONFIG2
21 #pragma config BOR4V = BOR40V     // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
22 #pragma config WRT = OFF           // Flash Program Memory Self Write Enable bits (Write protection off
23
24 // #pragma config statements should precede project file includes.
25 // Use project enums instead of #define for ON and OFF.
26
27 #include <xc.h>
28 #define XTAL_FREQ 4000000
```

# CÔNG CỤ PHÁT TRIỂN PICmicro

## PICkit™ 2

➤ PICkit2 là một bộ lập trình (programmer) và gỡ rối (debugger) thời gian thực. Nó có các tính năng sau:

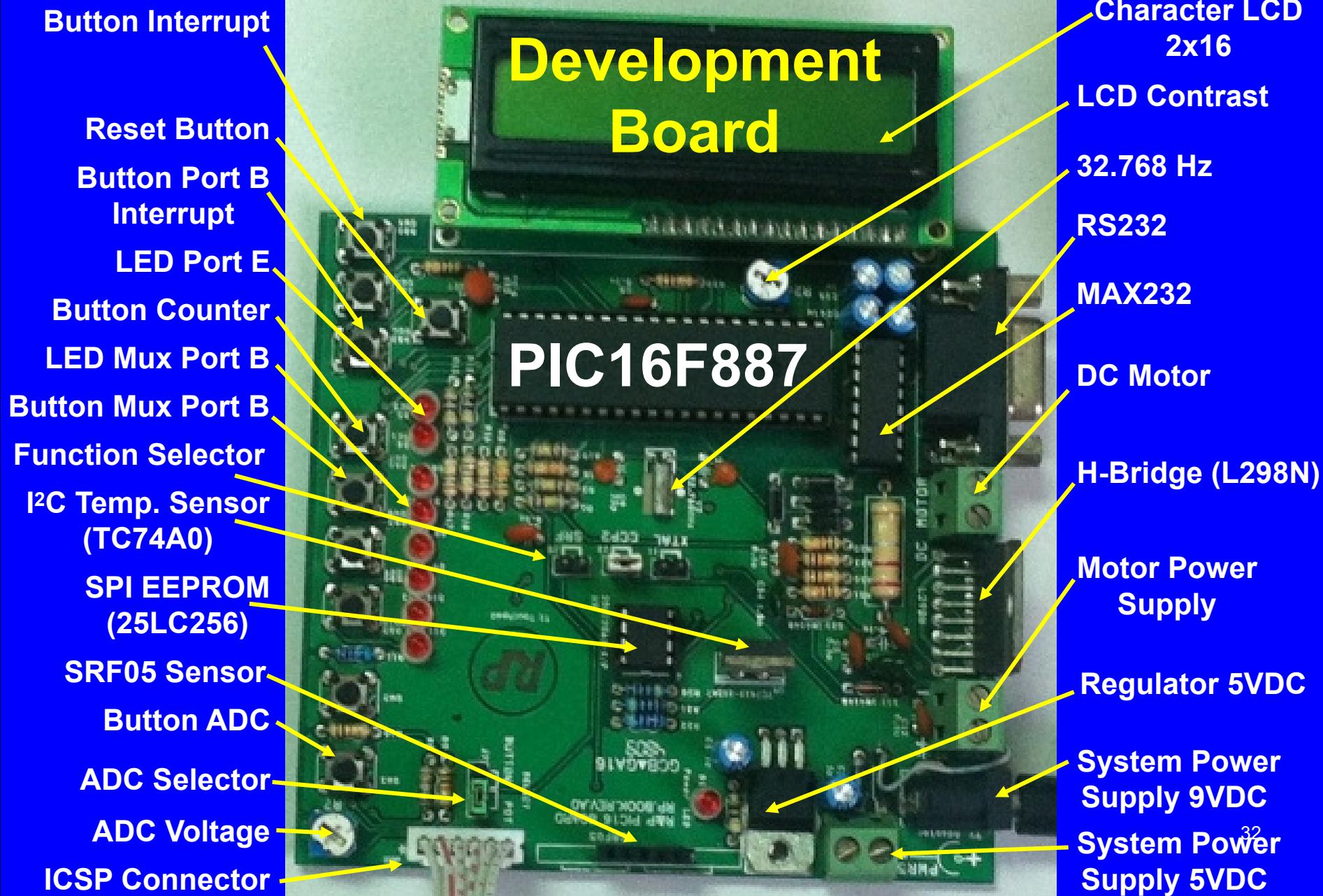
- Chạy gỡ rối trong thời gian thực
- Đọc/Ghi các bộ nhớ

**Flash / RAM / EEPROM**

- Lập trình các bit cấu hình vi điều khiển
- Xóa không gian bộ nhớ chương trình có kiểm tra xác nhận



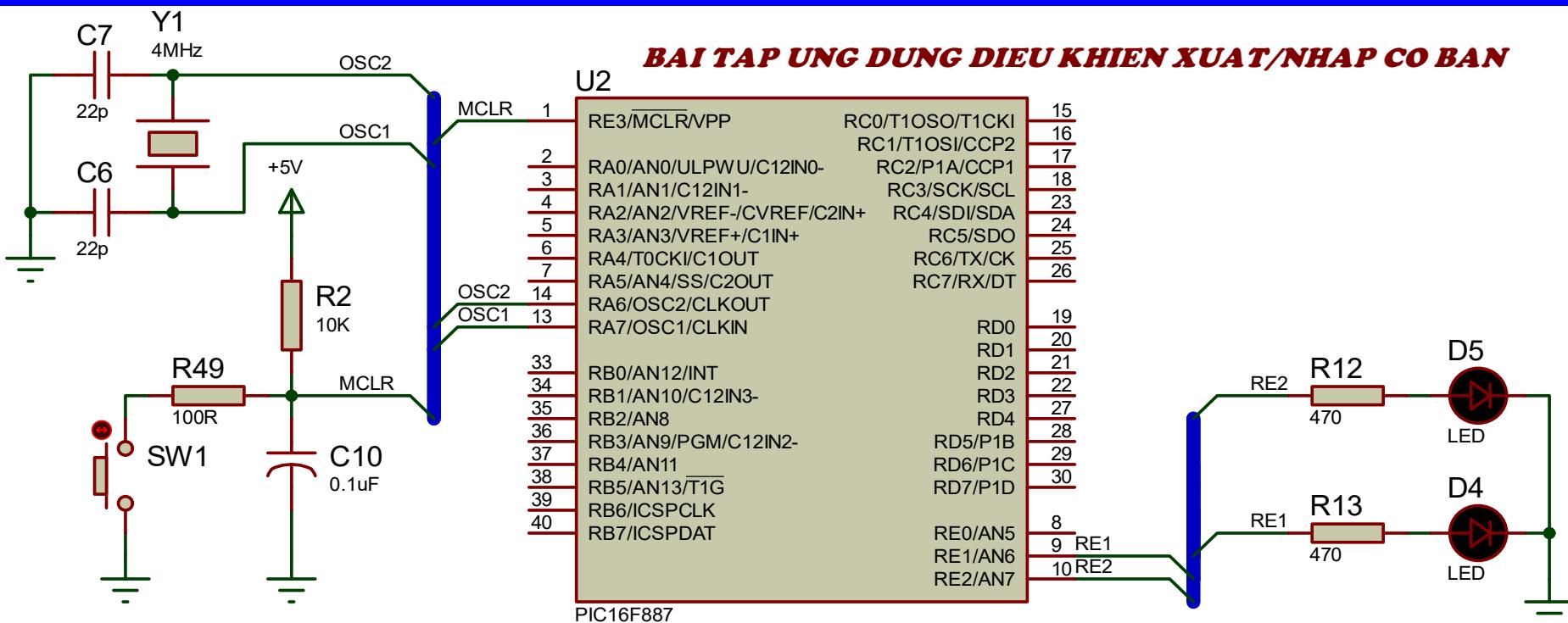
# CÔNG CỤ PHÁT TRIỂN PICmicro



# VÍ DỤ MINH HỌA

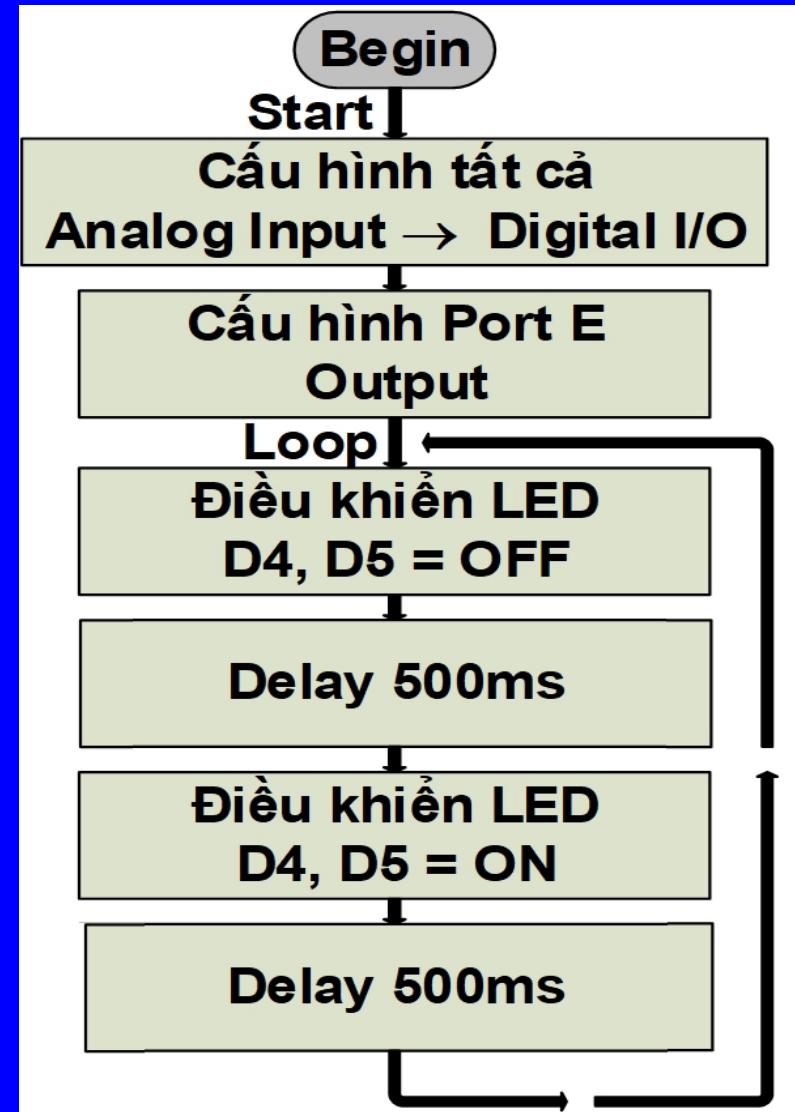
➤ **Ví dụ 1:** Dựa vào sơ đồ, viết chương trình điều khiển LED D4 và D5 chớp tắt

- **Sơ đồ nguyên lý:**



# VÍ DỤ MINH HỌA

- Giải thuật:



# VÍ DỤ MINH HỌA

- Cấu hình (Hi-Tech C):

```
_CONFIG(FOSC_HS & WDTE_OFF & PWRTE_ON &  
MCLRE_ON & CP_OFF & CPD_OFF & BOREN_OFF &  
IESO_OFF & FCMEN_OFF & LVP_OFF & DEBUG_ON);
```

```
#define _XTAL_FREQ 4000000
```

# VÍ DỤ MINH HỌA

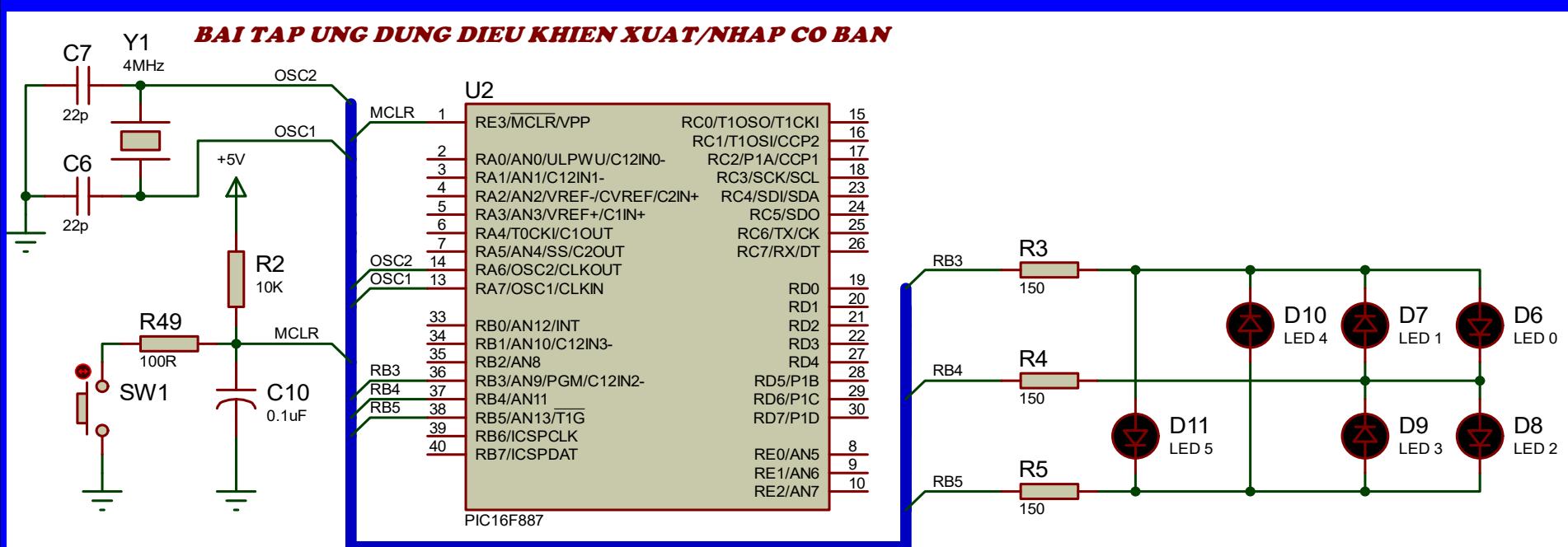
- **Chương trình (Hi-Tech C):**

```
void main (void)
{
    ANSEL = 0;
    ANSELH = 0;
    TRISE = 0x00;
    while(1)
    {
        PORTE = 0x00;
        __delay_ms(500);
        PORTE = 0x06;
        __delay_ms(500);
    }
}
```

# VÍ DỤ MINH HỌA

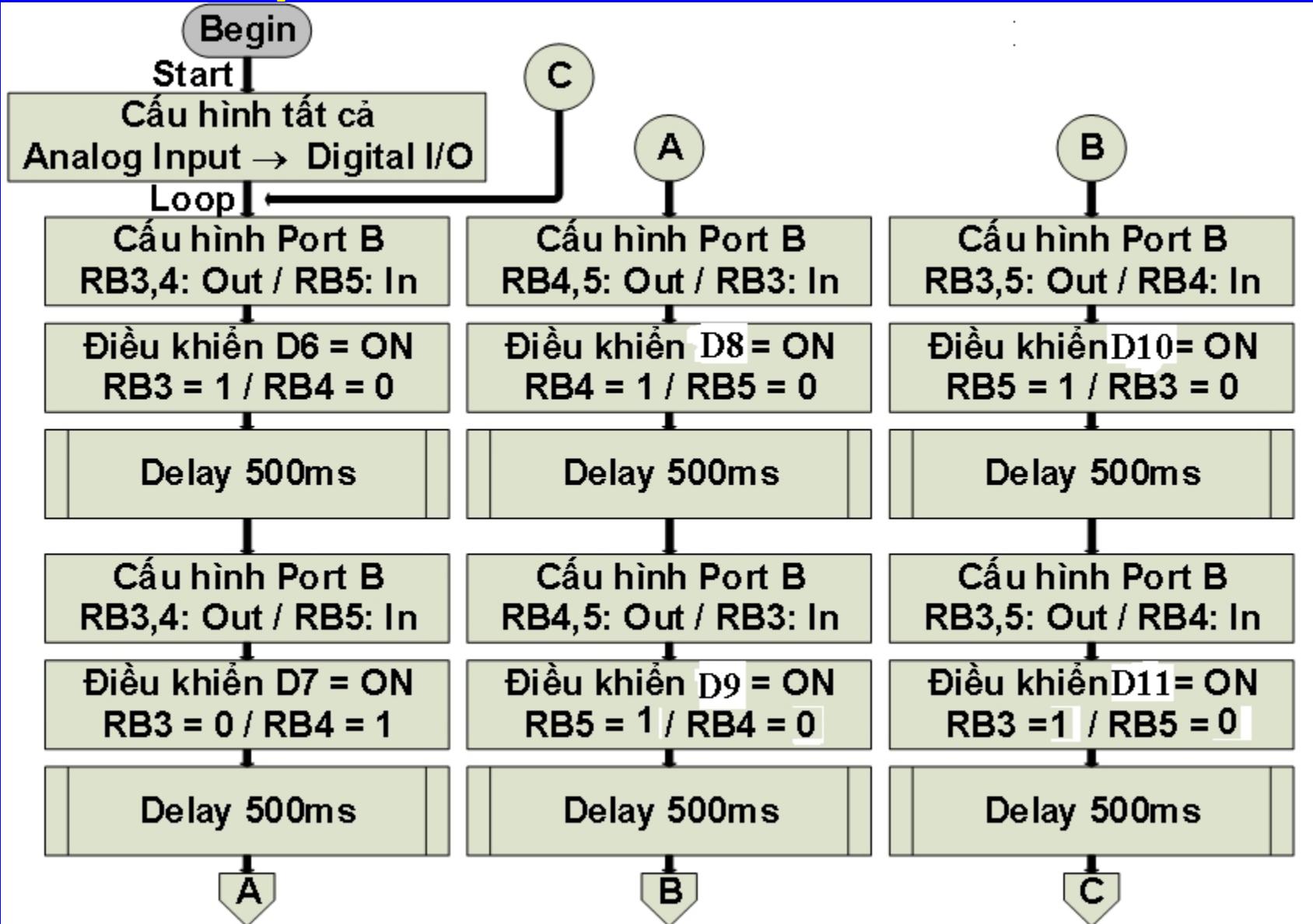
➤ **Ví dụ 2:** Dựa vào sơ đồ, viết chương trình điều khiển LED sáng đuôi D6 → D11

- Sơ đồ nguyên lý:



# VÍ DỤ MINH HỌA

- Giải thuật:



# VÍ DỤ MINH HỌA

- Cấu hình (Hi-Tech C):

```
_CONFIG(FOSC_HS & WDTE_OFF & PWRTE_ON &  
MCLRE_ON & CP_OFF & CPD_OFF & BOREN_OFF &  
IESO_OFF & FCMEN_OFF & LVP_OFF & DEBUG_ON);
```

```
#define _XTAL_FREQ 4000000
```

# VÍ DỤ MINH HỌA

Source: C3\_BT7\_CodeC.c

SIM: C3\_BT7\_CodeC.pdsprj

- Chương trình (Hi-Tech C):

```
void main (void)
{
```

```
ANSEL = 0;
```

```
ANSELH = 0;
```

```
while(1)
```

```
{
```

```
    TRISB = 0b00100000;
```

```
    PORTB = 0b00001000;
```

```
    __delay_ms(500);
```

```
    TRISB = 0b00100000;
```

```
    PORTB = 0b00010000;
```

```
    __delay_ms(500);
```

```
    TRISB = 0b00001000;
```

```
    PORTB = 0b00010000;
```

```
}
```

```
    __delay_ms(500);
```

```
    TRISB = 0b00001000;
```

```
    PORTB = 0b00100000;
```

```
    __delay_ms(500);
```

```
    TRISB = 0b00010000;
```

```
    PORTB = 0b00100000;
```

```
    __delay_ms(500);
```

```
    TRISB = 0b00010000;
```

```
    PORTB = 0b00001000;
```

```
    __delay_ms(500);
```

```
}
```

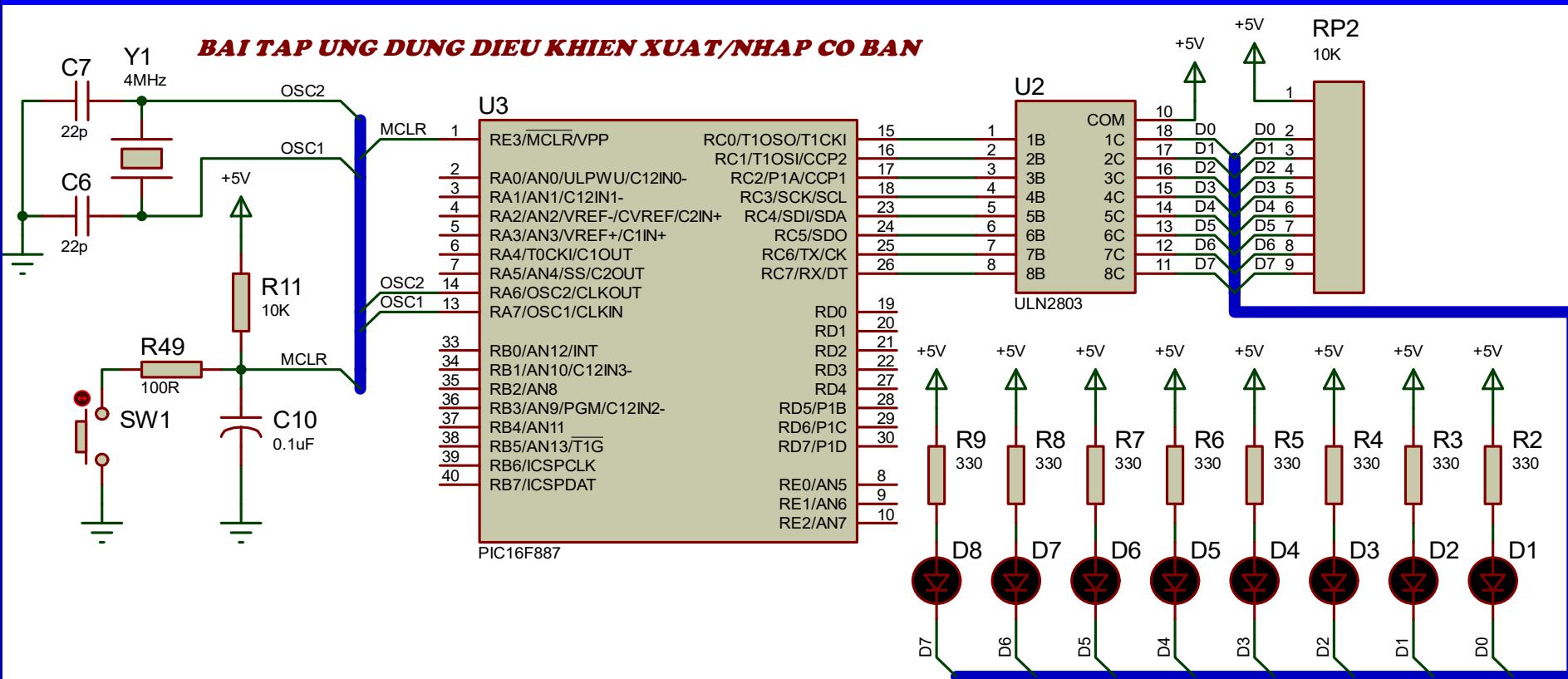
# BÀI TẬP ỨNG DỤNG

- **Bài tập 1:** Dựa vào sơ đồ, viết chương trình điều khiển LED D6 và D8 chớp tắt
- **Bài tập 2:** Dựa vào sơ đồ, viết chương trình điều khiển LED D10 và D11 chớp tắt
- **Bài tập 3:** Dựa vào sơ đồ, viết chương trình điều khiển LED sáng dần D6 → D11

# VÍ DỤ MINH HỌA

➤ **Ví dụ 3:** Dựa vào sơ đồ, viết chương trình điều khiển LED sáng dần D1 → D8

- Sơ đồ nguyên lý:**

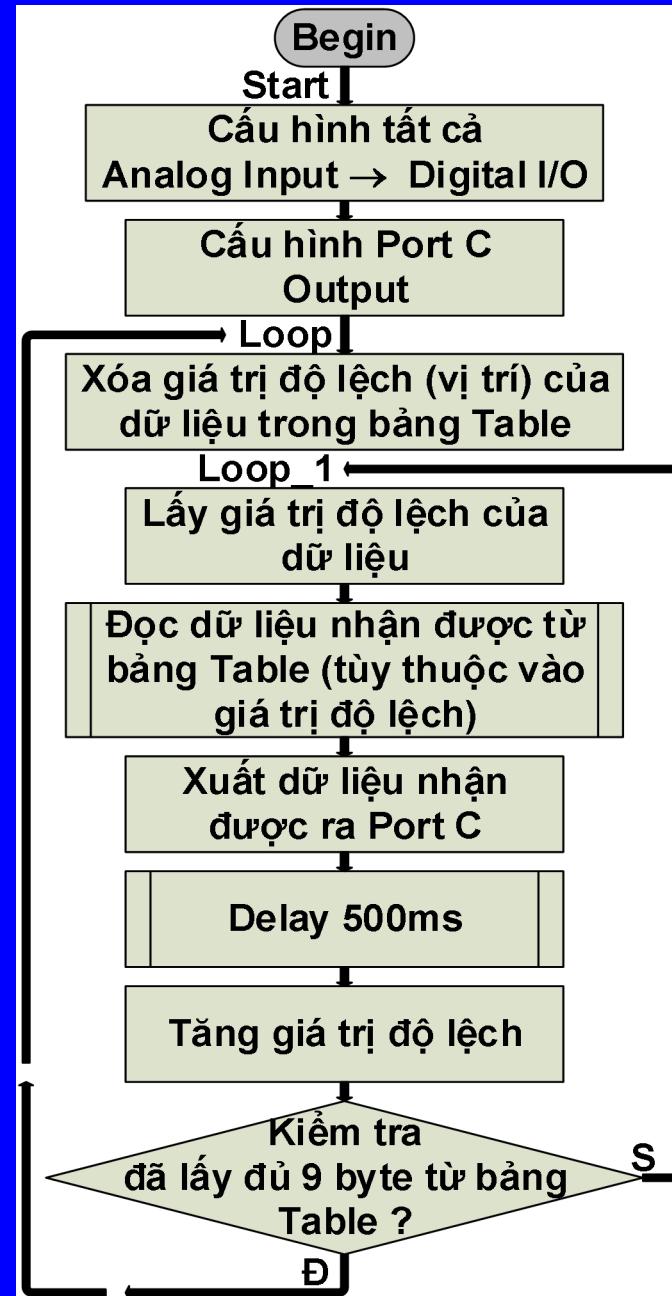


# VÍ DỤ MINH HỌA

- **Giải thuật:**

**Bảng dữ liệu Table**

```
b'00000000
b'00000001
b'00000011
b'00000111
b'00001111
b'00011111
b'00111111
b'01111111
b'11111111
```



# VÍ DỤ MINH HỌA

- Cấu hình (Hi-Tech C):

```
_CONFIG(FOSC_HS & WDTE_OFF & PWRTE_ON &  
MCLRE_ON & CP_OFF & CPD_OFF & BOREN_OFF &  
IESO_OFF & FCMEN_OFF & LVP_OFF & DEBUG_ON);
```

```
#define _XTAL_FREQ 4000000
```

# VÍ DỤ MINH HỌA

- **Chương trình (Hi-Tech C):**

```
void main (void)
{
    unsigned char j,temp;
    ANSEL = 0;
    ANSELH = 0;
    TRISC = 0x00;
    while(1)
    {
        PORTC = 0x00;
        __delay_ms(500);
    }
}
```

```
temp = 0x01;
for(j=1;j<=8;j++)
{
    PORTC = temp;
    __delay_ms(500);
    temp = temp | (temp << 1);
}
}
```

# BÀI TẬP ỨNG DỤNG

- **Bài tập 1:** Dựa vào sơ đồ, viết chương trình điều khiển LED hoạt động tuần tự:
- D1 → D8 tắt hết
  - D1 → D8 sáng hết
  - D1 → D8 tắt hết
  - D1 → D4 sáng dần, LED D5 → D8 tắt
  - D1 → D4 tắt, LED D5 → D8 sáng đuổi
  - Lặp lại các thao tác trên 3 lần rồi các LED tắt hết và dừng hoạt động
- **Bài tập 2:** Dựa vào sơ đồ, viết chương trình điều khiển LED sáng dần D1 → D8

# Bài tập

**Viết chương trình điều khiển 8LED đơn nối PORTD hoạt động tuần tự theo chu trình sau, ban đầu 8 led tắt hết**

- 8 LED chớp tắt 4 lần
- 8 LED sáng đuổi 3 lần
- 8 LED sáng dần 2 lần
- 8 LED sáng đòn 2 lần
- Lặp lại các thao tác trên 3 lần rồi các LED sáng hết và dừng hoạt động

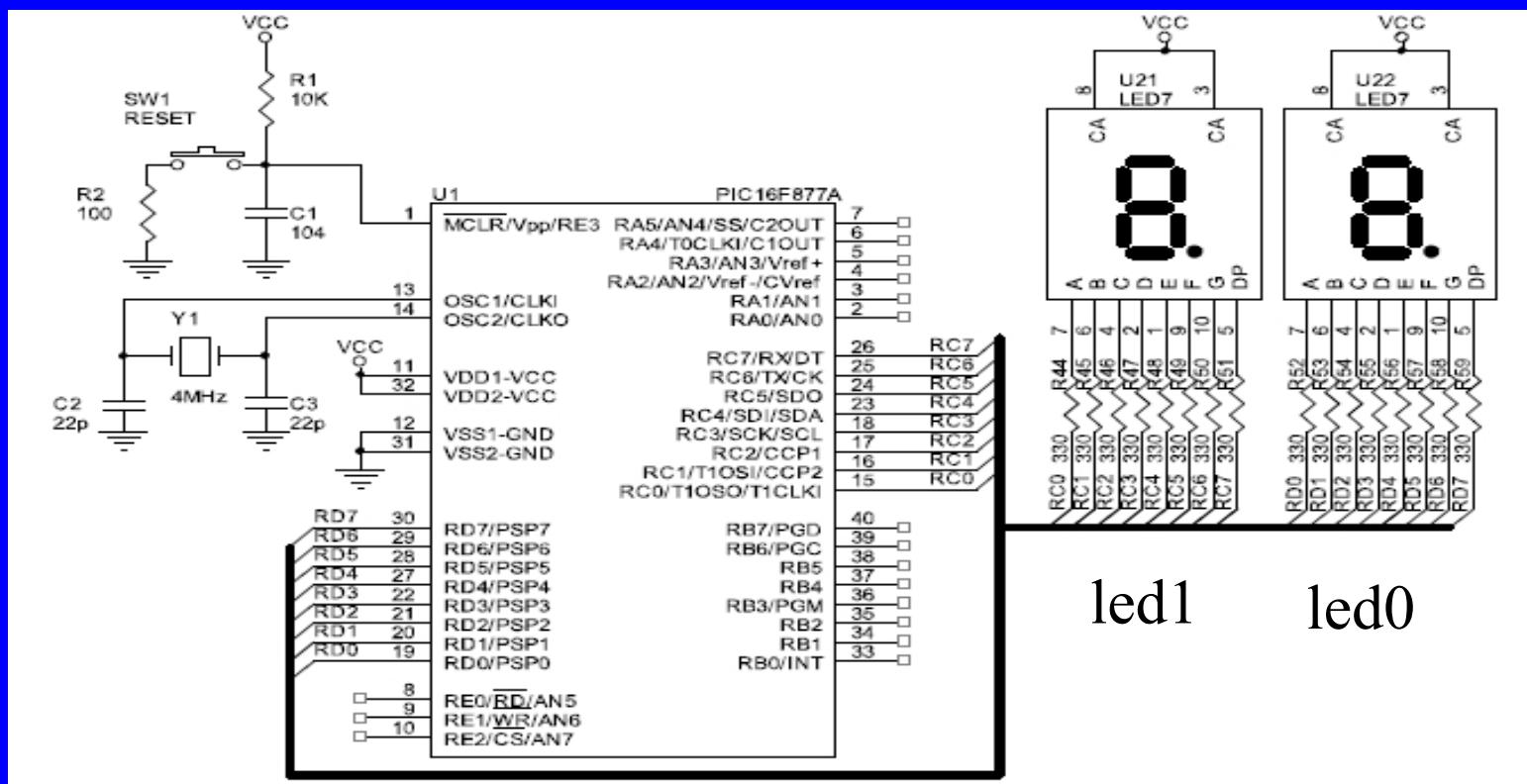
**Thời gian thay đổi trạng thái là 0,4s,  $f_{osc} = 4\text{MHz}$**

# Bài tập

- **Bài tập 1:** Dựa vào sơ đồ (**Ví dụ 2**), viết chương trình điều khiển LED D6->D11 sáng dần
- **Bài tập 2:** Dựa vào sơ đồ (**Ví dụ 2**), viết chương trình điều khiển LED D10 và D11 chớp tắt
- **Bài tập 3:** Dựa vào sơ đồ (**Ví dụ 3**), viết chương trình điều khiển LED hoạt động tuần tự:
  - D1 → D8 tắt hết
  - D1 → D8 sáng hết
  - D1 → D8 tắt hết
  - D1 → D4 sáng dần, LED D5 → D8 tắt
  - D1 → D4 tắt, LED D5 → D8 sáng đuôi
  - Lặp lại các thao tác trên 3 lần rồi các LED tắt hết và dừng hoạt động

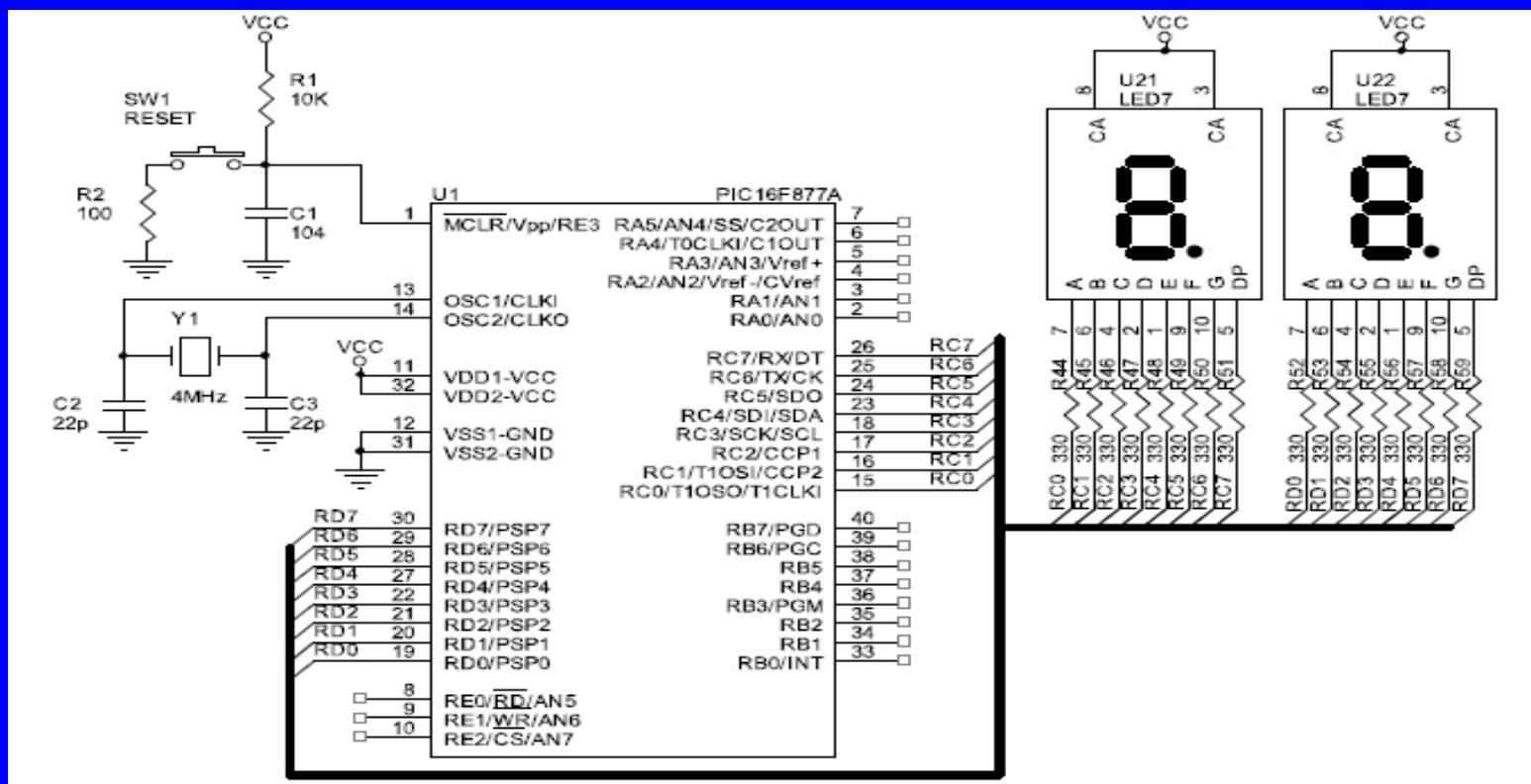
# VÍ DỤ MINH HỌA

- **Ví dụ 4:** Dựa vào sơ đồ, viết chương trình điều khiển hiển thị số 1 chớp tắt trên led 1
- **Sơ đồ nguyên lý: bộ hiển thị không đa hợp, ngoại vào dữ liệu kiểu 7 đoạn**



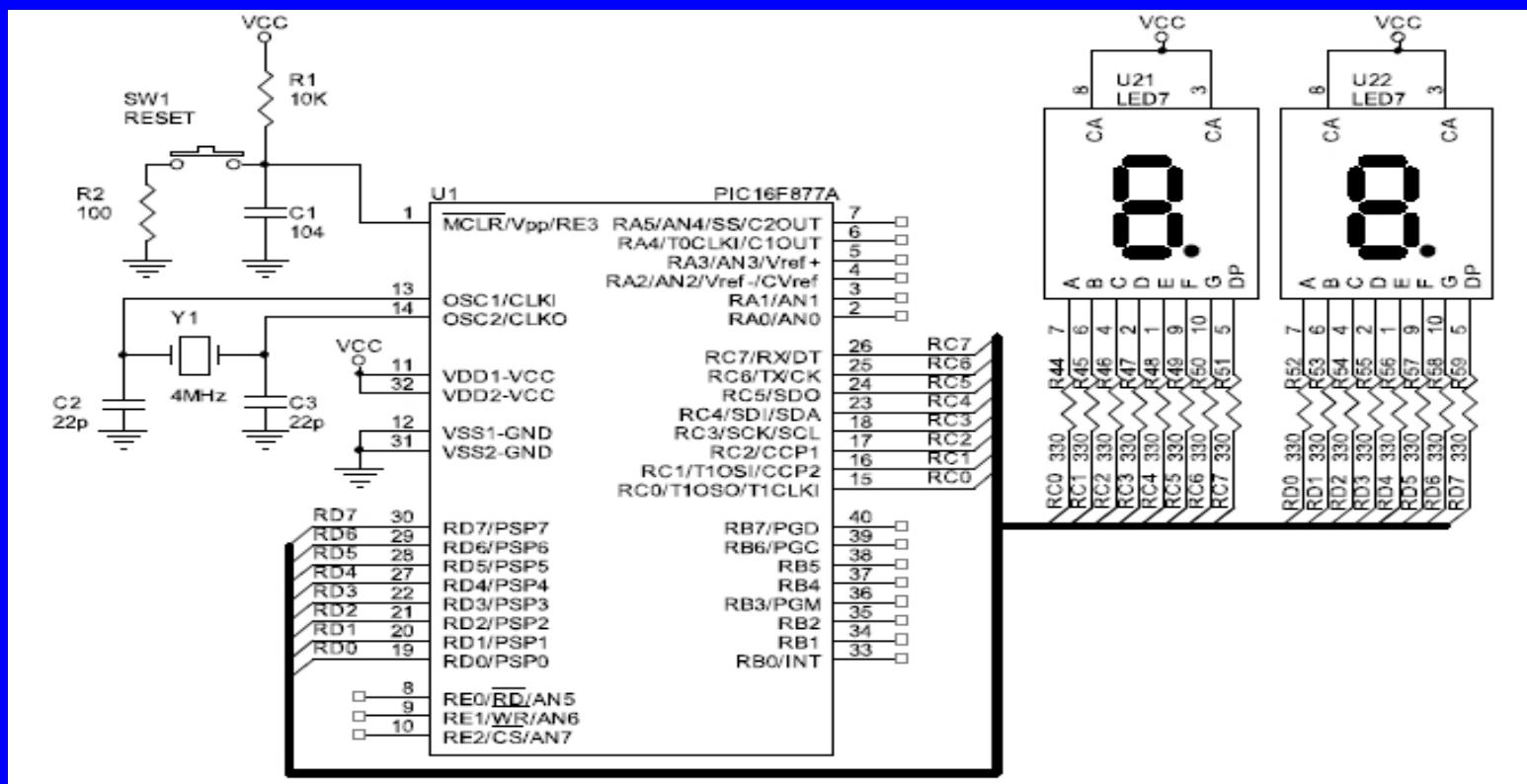
# VÍ DỤ MINH HỌA

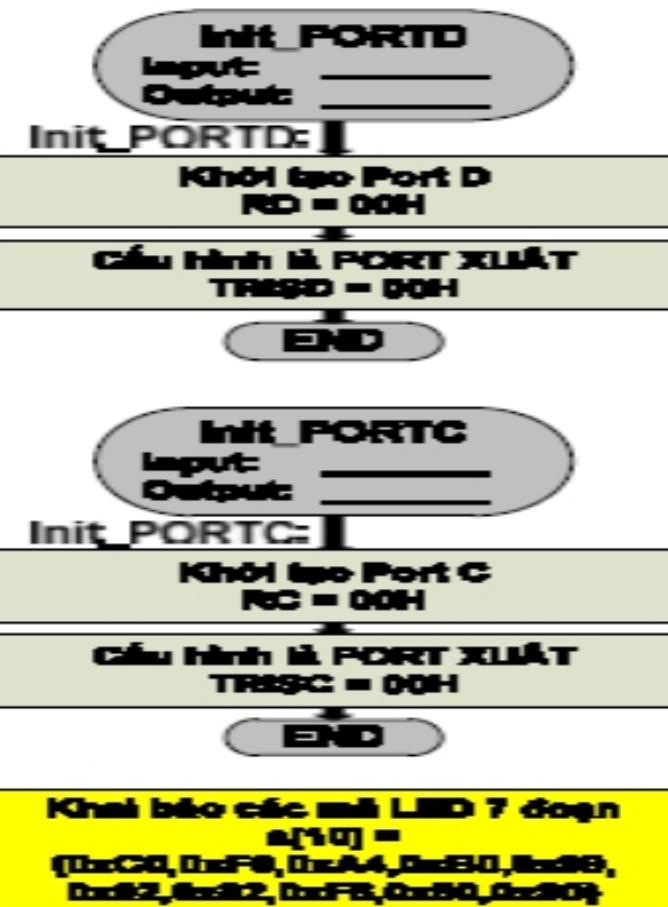
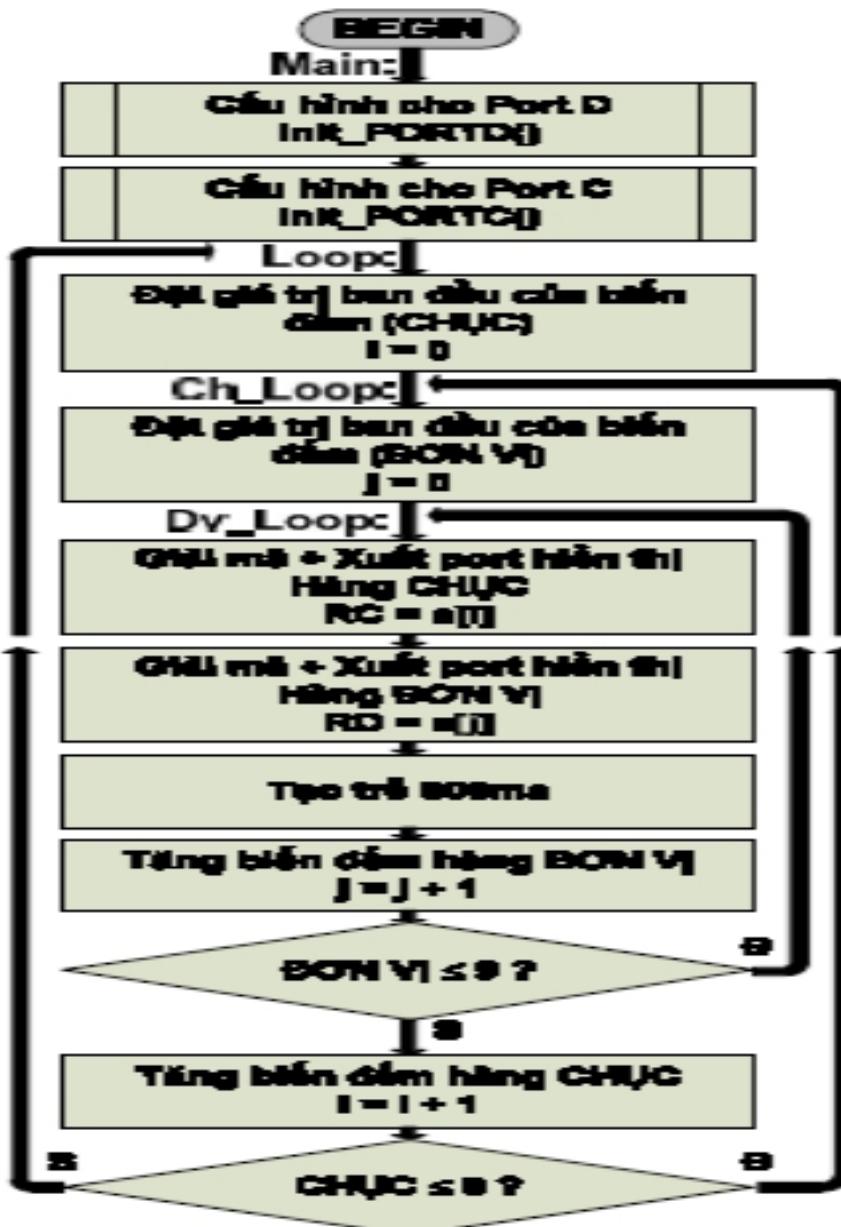
- **Ví dụ 5:** Dựa vào sơ đồ, viết chương trình điều khiển hiển thị chữ Cd chớp tắt trên led
- **Sơ đồ nguyên lý:** bộ hiển thị không đa hợp, ngoại vào dữ liệu kiểu 7 đoạn



# VÍ DỤ MINH HỌA

- **Ví dụ 6:** Dựa vào sơ đồ, viết chương trình điều khiển hiển thị tăng dần các số từ 00-99
- **Sơ đồ nguyên lý:** bộ hiển thị không đa hợp, ngoại vào dữ liệu kiểu 7 đoạn

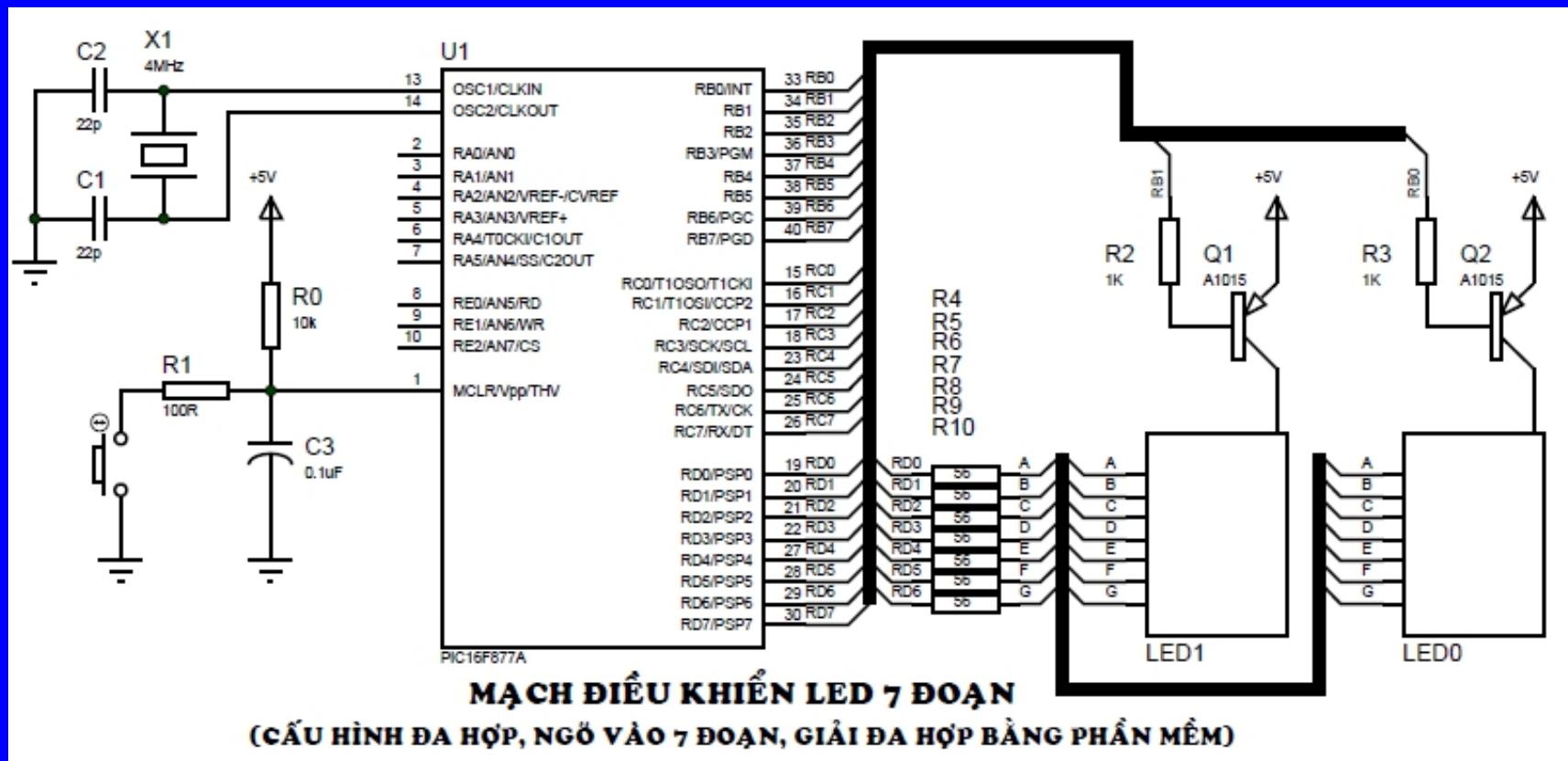


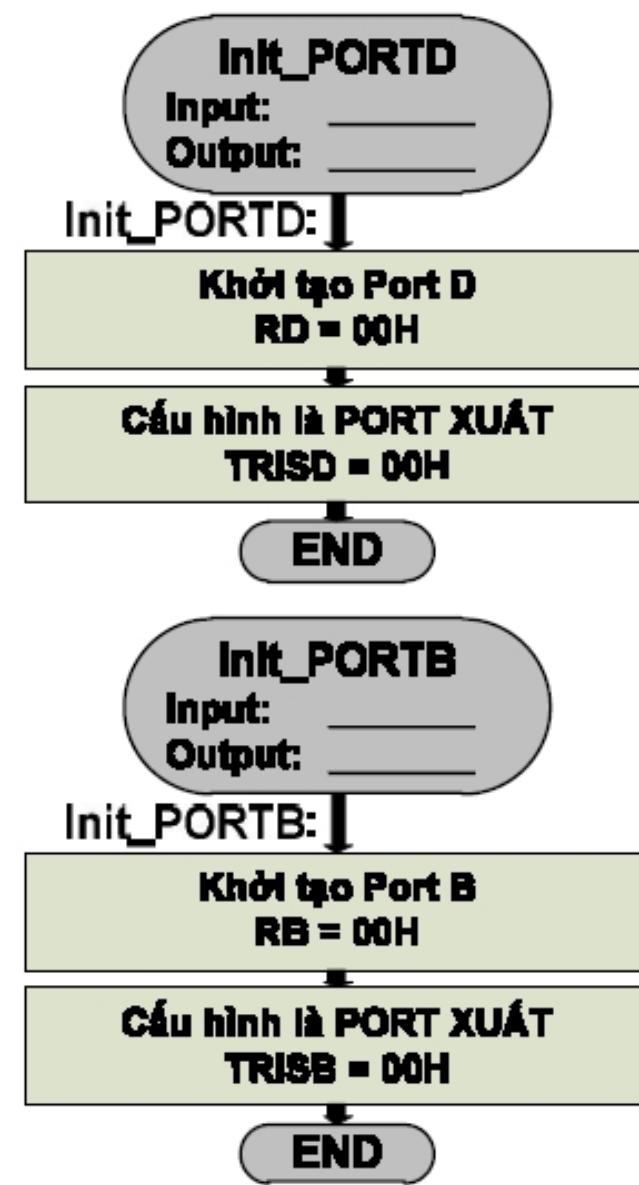
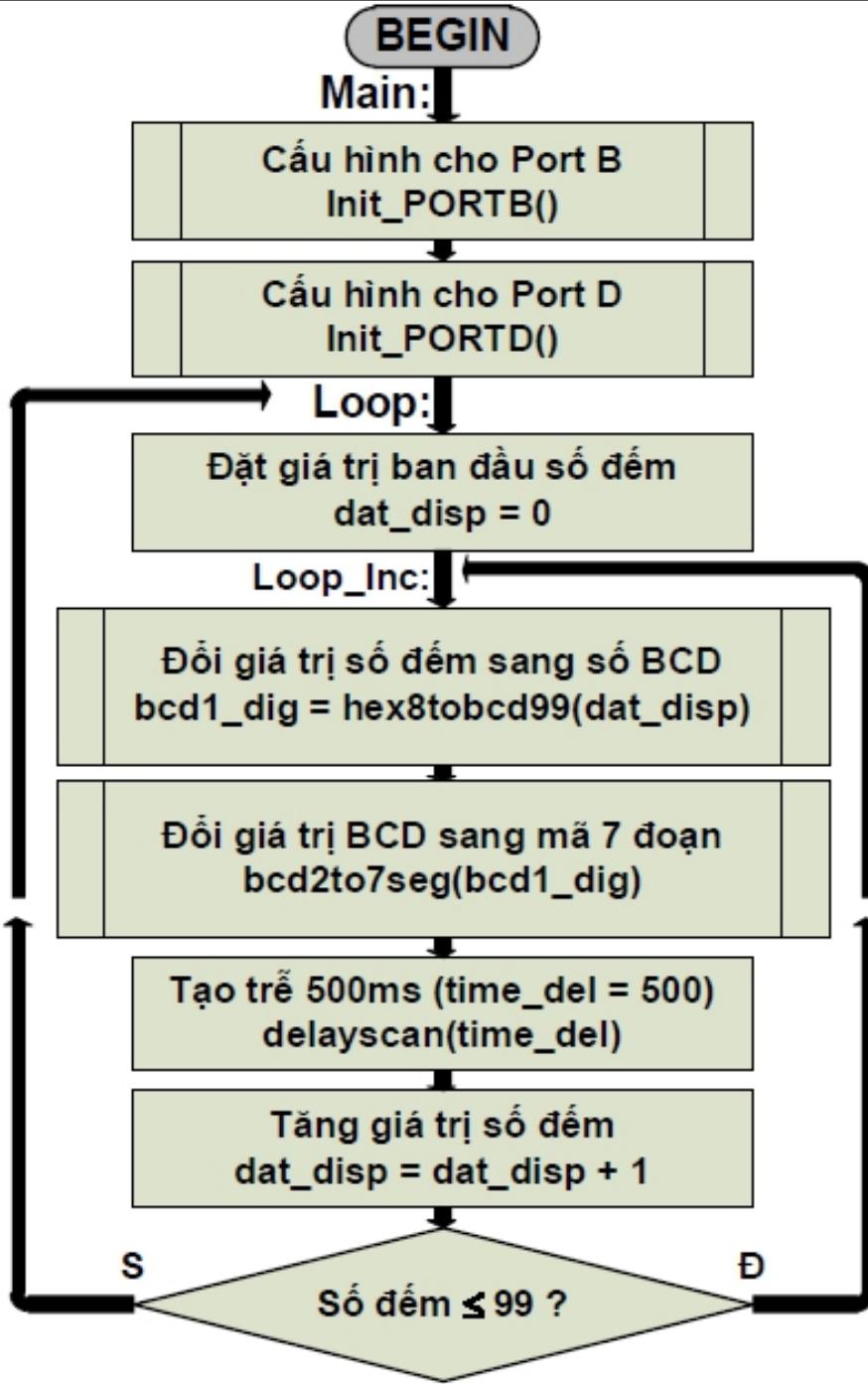


# VÍ DỤ MINH HỌA

➤ **Ví dụ 7:** Dựa vào sơ đồ, viết chương trình điều khiển hiển thị tăng dần các số từ 00-99

- **Sơ đồ nguyên lý:** bộ hiển thị đa hợp, ngõ vào dữ liệu kiểu 7 đoạn

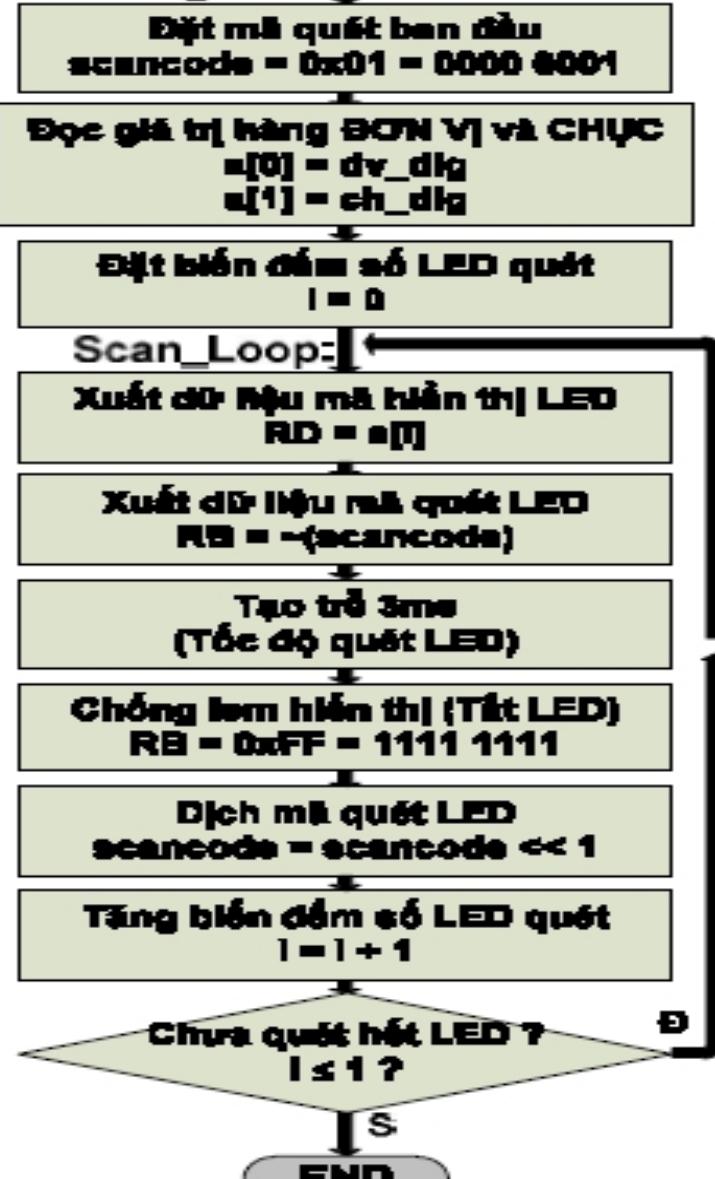




## Disp7segmul2

**Input:** ch\_dig,dv\_dig = Giá trị hiển thị.  
**Output:** PortD = Các thanh của LED.  
 PortB = Chọn LED.

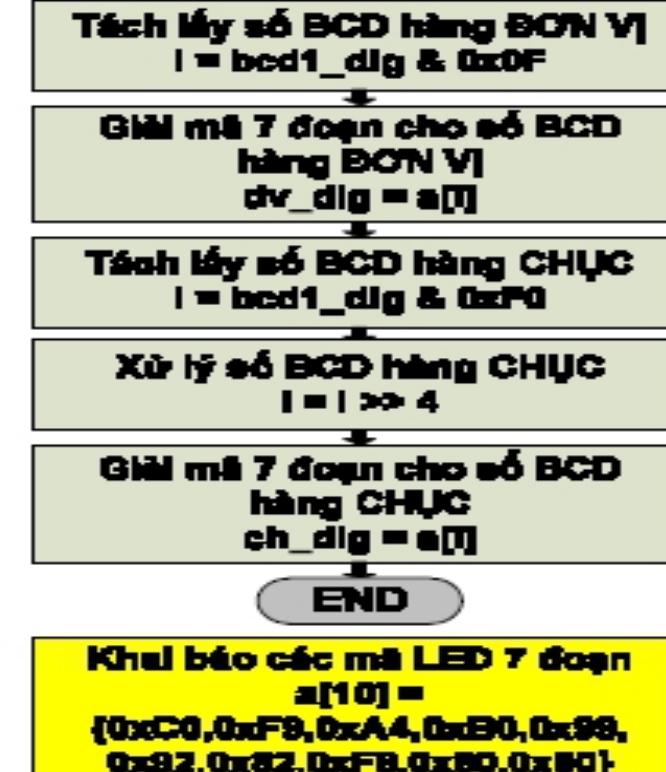
### Disp7segmul2:



## Bcd2to7seg

**Input:** bcd1\_dig = Số BCD nén.  
**Output:** ch\_dig,dv\_dig = Giá trị 7 đoạn.

### Bcd2to7seg:



## DelayScan

**Input:** time\_del: Thời gian chờ (ms).  
ch\_dig,dv\_dig = Giá trị hiển thị.

**Output:** \_\_\_\_\_

### DelayScan:

Đặt giá trị ban đầu cho số lần  
lặp để tạo trễ (đơn vị tính ms)  
 $I = 1$

### DelayScan\_Loop:

Quét LED để hiển thị  
disp7segmul2()

Tăng biến đếm số lần lặp  
 $I = I + 1$

Chưa đủ thời gian chờ?  
 $I \leq (\text{time\_del})$

S

**END**

## Hex8toBcd99

**Input:** X = Giá trị HEX.

**Output:** Y = Giá trị BCD nén.

### Hex8toBcd99:

Xác định giá trị hàng CHỤC của  
số HEX (X)  
CHỤC =  $X / 10$

Xác định giá trị hàng ĐƠN VỊ  
của số HEX (X)  
ĐƠN VỊ =  $X \% 10$

Ghép hàng CHỤC và ĐƠN VỊ  
thành số BCD nén (Y)  
 $Y = (\text{CHỤC} \ll 4) | \text{ĐƠN VỊ}$

**END**

```
#include <htc.h>
#include <math.h>
// Cau hinh su dung dao dong noi 4MHz
__CONFIG(HS & WDTDIS & PWRTEN & UNPROTECT & BORDIS &
LVPDIS);
#define _XTAL_FREQ 4000000
// Khai bao bien.
unsigned char ch_dig,dv_dig; // Bien chua gia tri hien thi tren tung LED
(bien toan cuc).
void init_PORTD(void)
{
    PORTD = 0X00; // Khoi tao Port D.
    TRISD = 0X00; // Cau hinh PORTD la ngo ra.
}
void init_PORTB(void)
{
    PORTB = 0X00; // Khoi tao Port B.
    TRISB = 0X00; // Cau hinh PORTB la ngo ra.
}
```

```
void disp7segmul2()      // Ham quet hien thi LED 7 doan (2 LED 7 doan).
{
// Khai bao bien.
unsigned char scancode,i;
unsigned char a[2];
// Dinh nghia ham.
scancode = 0x01;          // Ma quet LED ban dau - 00000001.
a[0] = dv_dig;            // Nap gia tri hang don vi.
a[1] = ch_dig;            // Nap gia tri hang chuc.
for(i=0;i<=1;i++)
{
PORTD = a[i];              // Xuat ma hien thi.
PORTB = ~scancode;         // Xuat ma quet (Dao ma do LED kich hoat
muc thap).
__delay_ms(3);              // Tan so quet LED - 3ms.
PORTB = 0xFF;                // Chong lem hien thi.
scancode = scancode << 1; // Dich ma quet LED.
}
}
```

```
//Ham giao ma tu so BCD nen sang ma 7 doan (1 so BCD nen).
void bcd2to7seg(unsigned char bcd1_dig)
{
// Khai bao bien.
unsigned char i;

// Khai bao hang.
const char a[10] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};

// Dinh nghia ham.
i = bcd1_dig & 0x0F;      // Tach lay so BCD hang don vi.
dv_dig = a[i];            // Lay ma 7 doan tuong ung cho BCD hang don vi.

i = bcd1_dig & 0xF0;      // Tach lay so BCD hang chuc.
i = i >> 4;
ch_dig = a[i];            // Lay ma 7 doan tuong ung cho BCD hang chuc.
}
```

//// Ham tao thoi gian tre co quet LED 7 doan de hien thi.

```
void delayscan(unsigned int time_del)
```

```
{
```

// Khai bao bien.

```
unsigned int i;
```

// Dinh nghia ham.

```
for (i=1;i<=(time_del/6);i++) // Vong lap tao thoi gian tre time_del(ms).
```

```
{
```

```
    disp7segmul2();
```

// Quet LED de hien thi

```
}
```

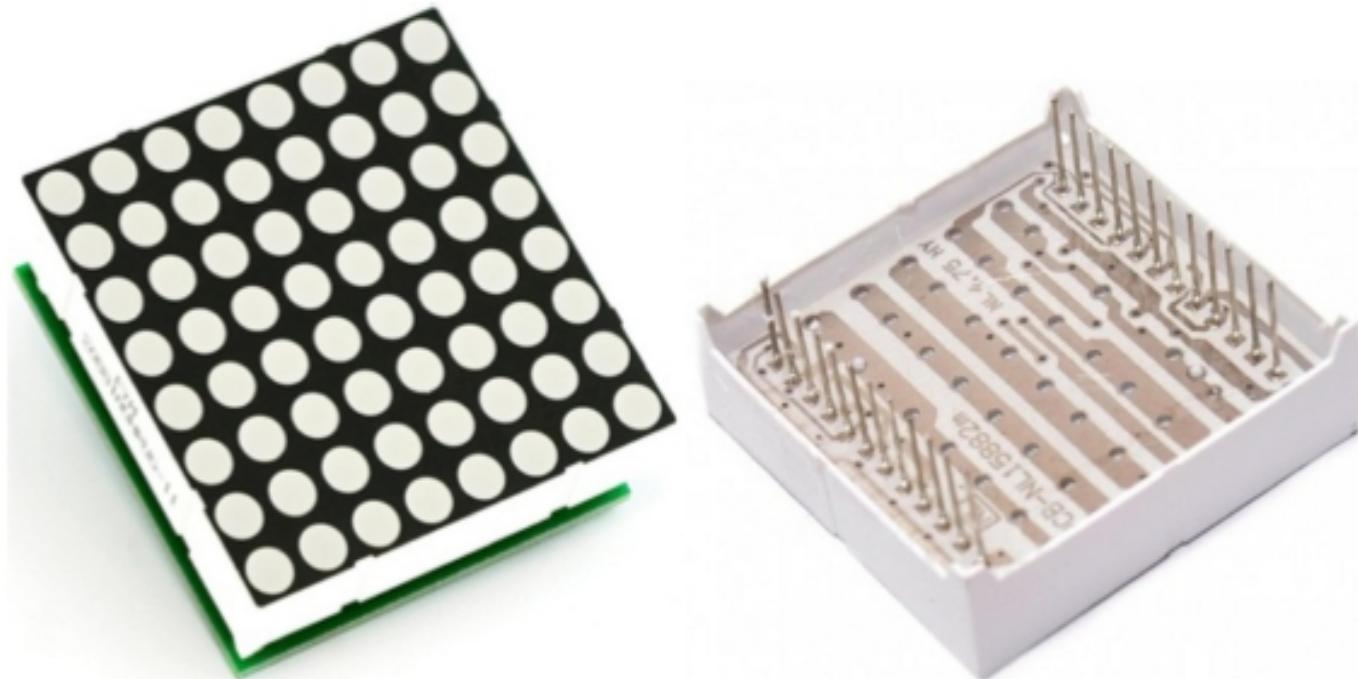
```
}
```

```
// Ham chuyen doi so hex 8 bit (<=99) sang so BCD nen.  
// *****  
// Ten:          hex8tobcd99.  
// Dau vao:      x = Gia tri HEX (00 - 99).  
// Dau ra:       y = Gia tri BCD nen (00 - 99).  
// *****/  
unsigned char hex8tobcd99 (unsigned char x)  
{  
    // Khai bao bien.  
    unsigned char y;  
    // Dinh nghia ham.  
    y = (x / 10) << 4;  
    y = y | (x % 10);  
    return (y);  
}
```

```
// Chuong trinh chinh.  
void main(void)  
{  
// Khai bao bien.  
    unsigned char bcd1_dig,dat_disp;  
    unsigned int time_del;  
// Chuong trinh.  
    init_PORTB();          // Cau hinh Port B.  
    init_PORTD();          // Cau hinh Port D.  
    while(1)                // Vong lap cho ngat.  
    {  
        for(dat_disp=0;dat_disp<=99;dat_disp++)  
        {  
            bcd1_dig = hex8tobcd99(dat_disp); // Doi so HEX sang BCD.  
            bcd2to7seg(bcd1_dig); // Doi so BCD sang ma 7 doan  
            time_del = 500;           // Thoi gian tre 500ms.  
            delayscan(time_del); // Tao tre va quet LED de hien thi.  
        }  
    }  
}
```

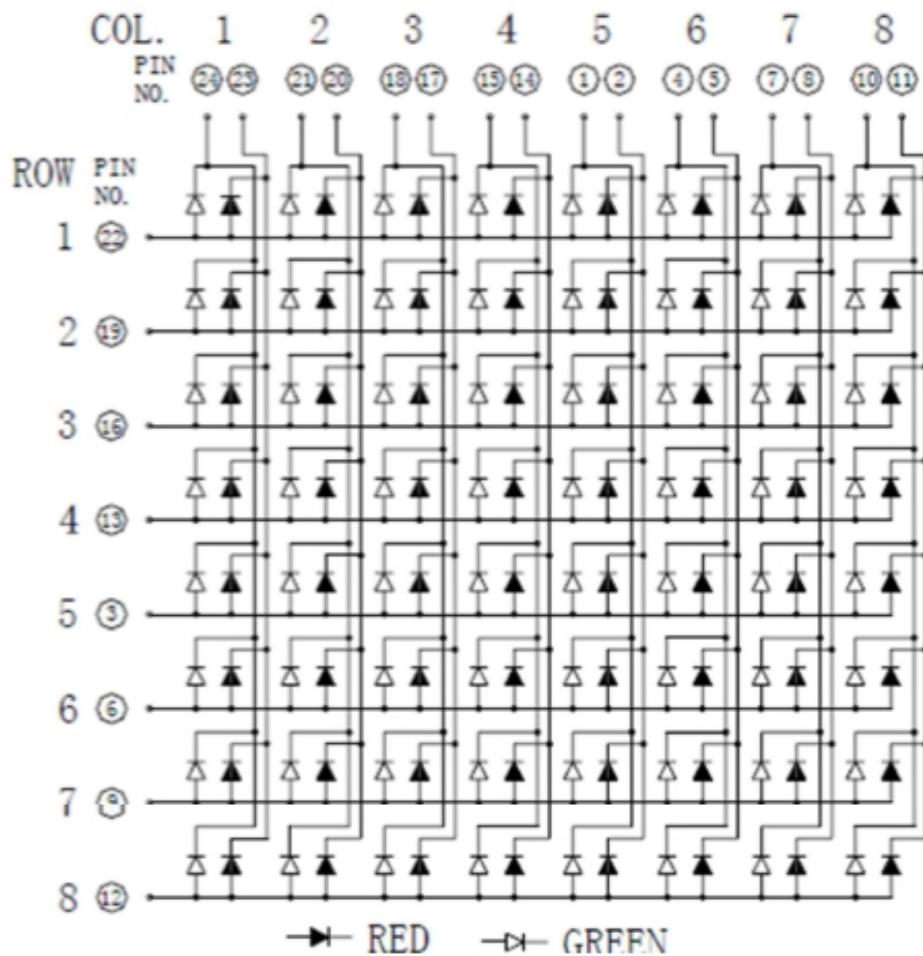
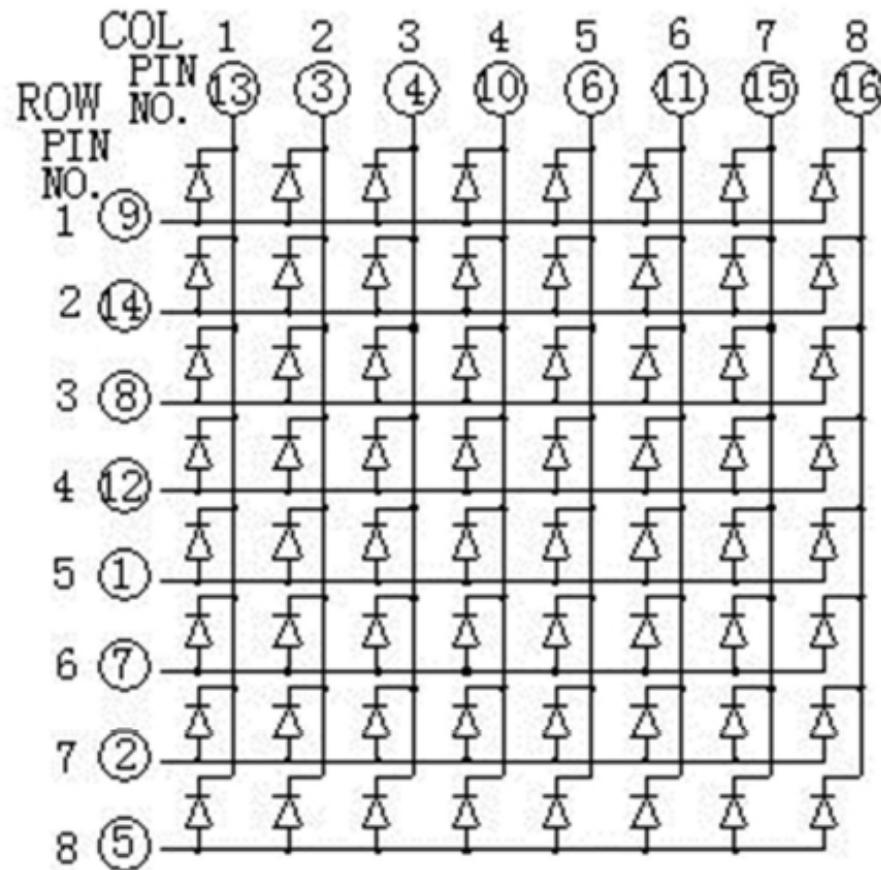
# Led ma trận

Hình dạng của LED ma trận 8x8 ngoài thực tế như sau:



*Hình 3.1: Hình dạng LED ma trận 8x8 có hai màu (24 chân).*

# Cấu tạo Led ma trận



Hình 3.2: Cấu tạo bên trong của LED ma trận 8x8 có một màu (trái) và hai màu (phải).

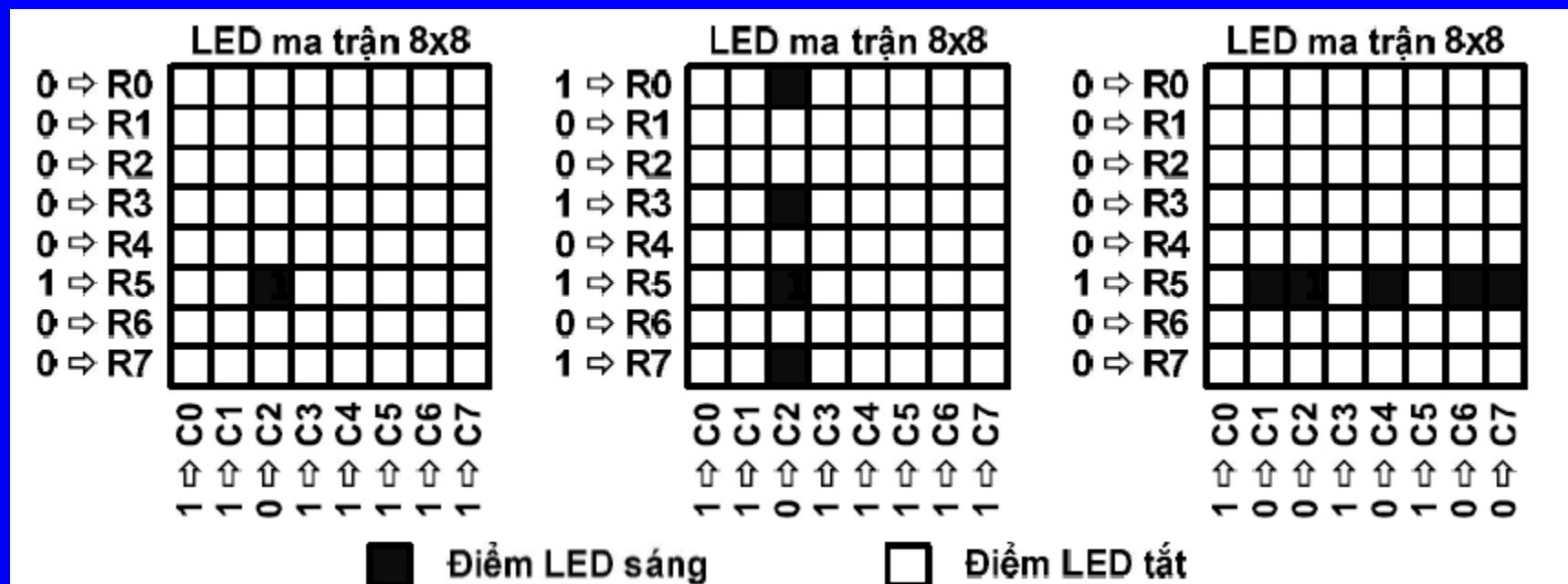
# Nguyên lý điều khiển led ma trận

- Phương pháp chỉ thị tĩnh
- Phương pháp chỉ thị động (pp quét led)

# Nguyên lý điều khiển led ma trận

## ➤ Phương pháp chỉ thị tĩnh:

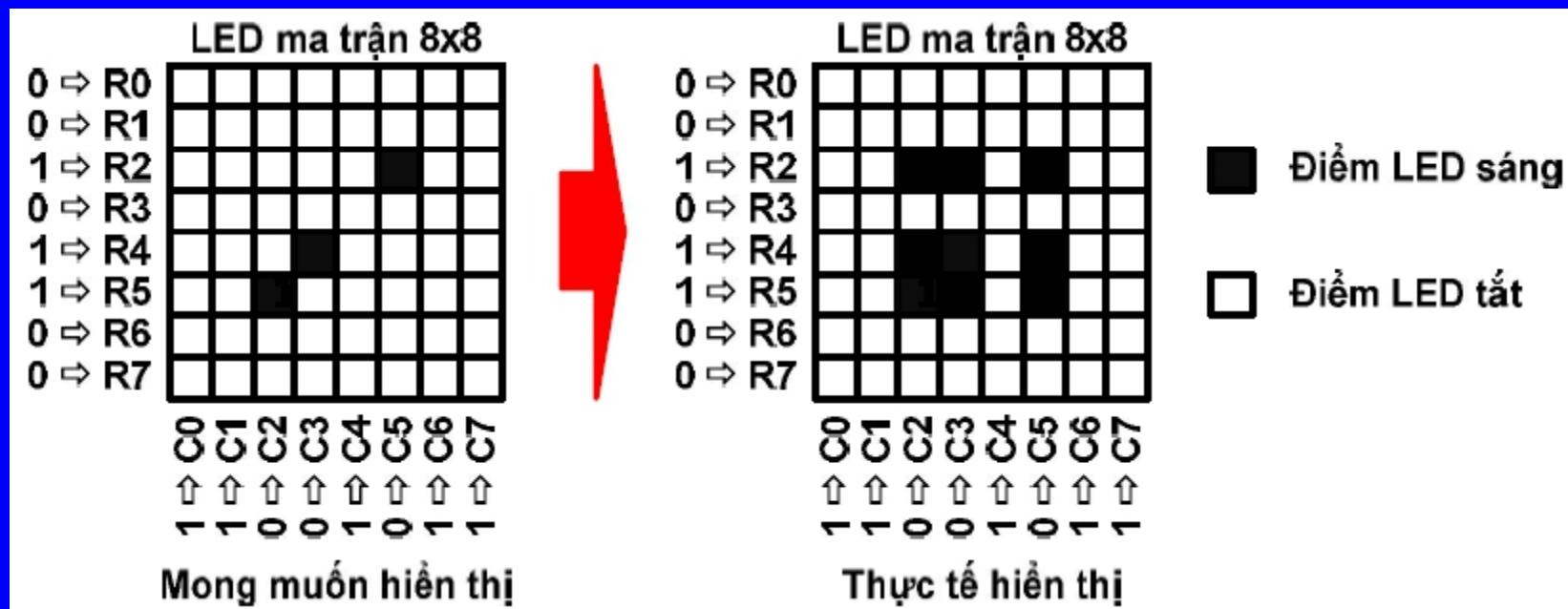
- Hàng là Anode, cột là Cathode của các điểm LED



# Nguyên lý điều khiển led ma trận

## ➤ Phương pháp chỉ thị tĩnh:

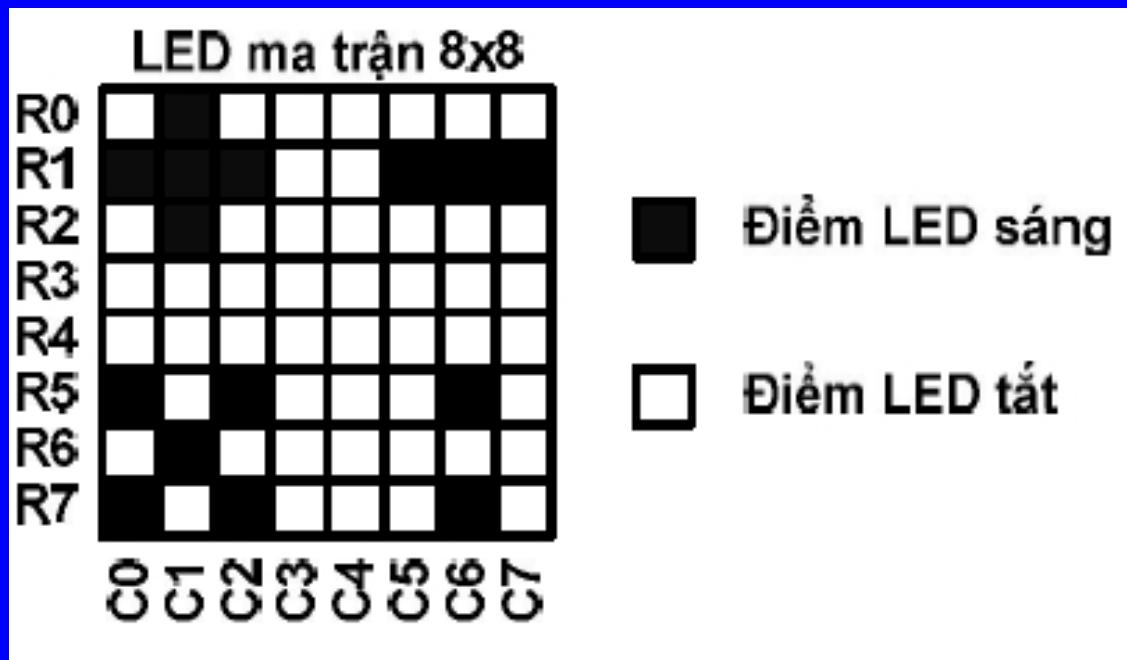
- Trường hợp LED cần sáng nằm trên nhiều hàng, nhiều cột khác nhau



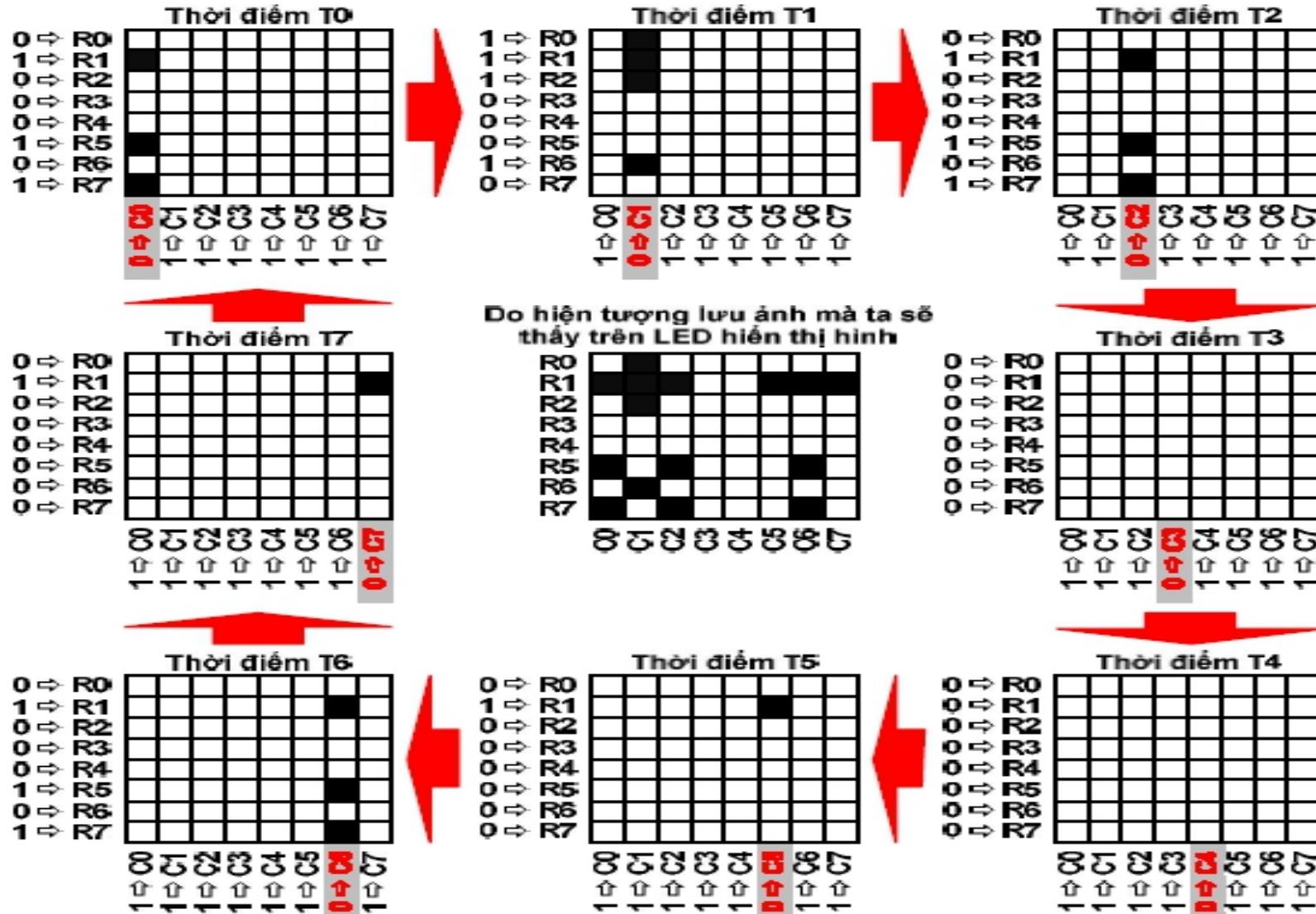
# Nguyên lý điều khiển led ma trận

- Phương pháp chỉ thị động  
(pp quét led, quét cột)
  - Điều khiển các điểm LED trên cùng 1 cột sáng, quét lần lượt các cột cho đến hết.
  - Hiện tượng lưu ảnh trên võng mạc mắt người

# Ví dụ Điều khiển led hiển thị như sau



# PP điều khiển thực hiện liên tục và lần lượt qua 8 thời điểm T0-T7 tương ứng 8 cột C0-C7

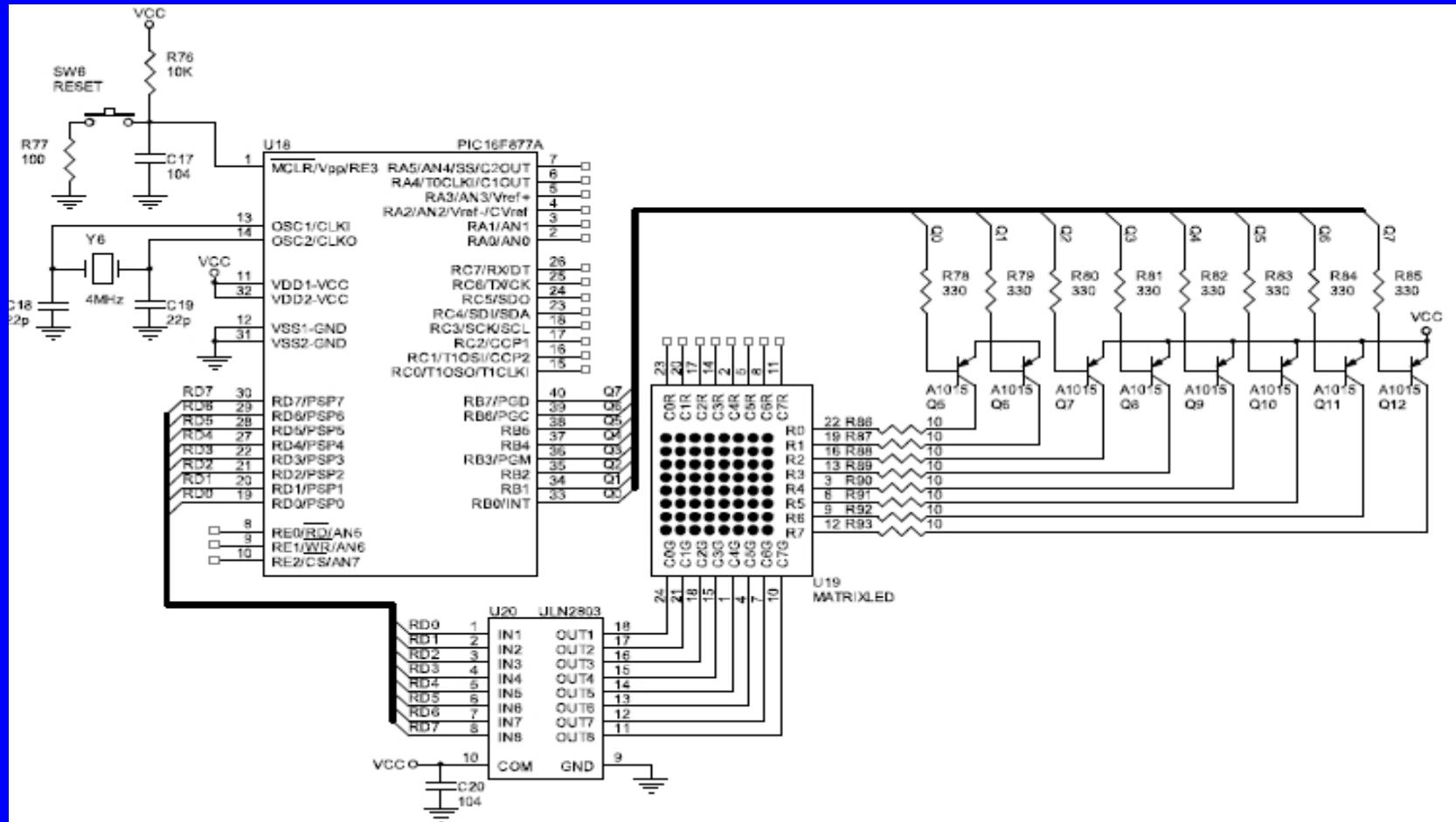


Thời gian duy trì một thời điểm phải đảm bảo thỏa mãn điều kiện lưu ảnh đồng thời cũng phải đảm bảo thời gian để các điểm LED phát sáng. Thông thường ta áp dụng công thức tính thời gian duy trì một thời điểm như sau:

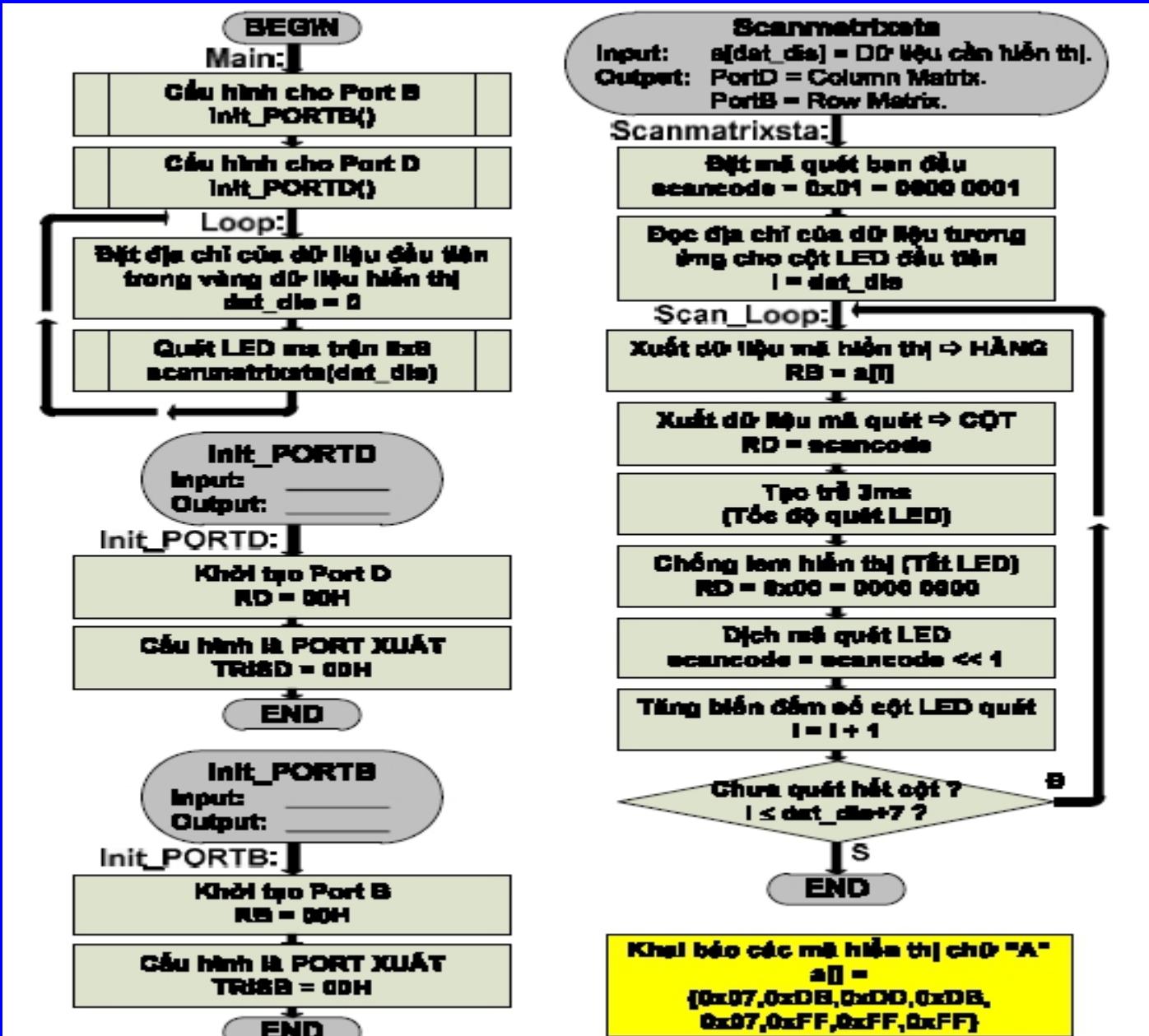
$$t_{DELAY} = \frac{40(ms)}{\text{Tổng số cột của LED ma trận}}$$

- o Nếu ta chọn thời gian này quá lớn thì hình ảnh trên LED ma trận sẽ bị chớp (rung) liên tục.
- o Nếu ta chọn thời gian này quá nhỏ thì hình ảnh trên LED ma trận không bị chớp (rung) nhưng độ sáng của các điểm LED sẽ bị giảm đi rất nhiều.

# Ví dụ 1:đkhiển led ma trận hiển thị chữ A đứng yên không dịch chuyển



# Giải thuật



```

// *****
#include <htc.h>
#include <math.h>
// Cau hinh su dung dao dong noi 4MHz
__CONFIG(HS & WDTDIS & PWRREN & UNPROTECT & BORDIS & LVPDIS);
#define _XTAL_FREQ 4000000
// Khai bao bien.
// Khai bao du lieu hien thi.
const unsigned char a[] = {0x07,0xDB,0xDD,0xDB,0x07,0xFF,0xFF,0xFF};
    // Ma hien thi chu "A"
// *****
// Ten CTC:      init_PORTD.
// Thong so dau vao:
// Thong so dau ra:
// Cong dung:      Khoi tao Port D.
// *****

void init_PORTD(void)
{
    PORTD = 0X00;
    TRISD = 0X00;
}
// *****
// Ten CTC:      init_PORTB.
// Thong so dau vao:
// Thong so dau ra:
// Cong dung:      Khoi tao Port B.
// *****

void init_PORTB(void)
{
    PORTB = 0X00;
    TRISB = 0X00;
}

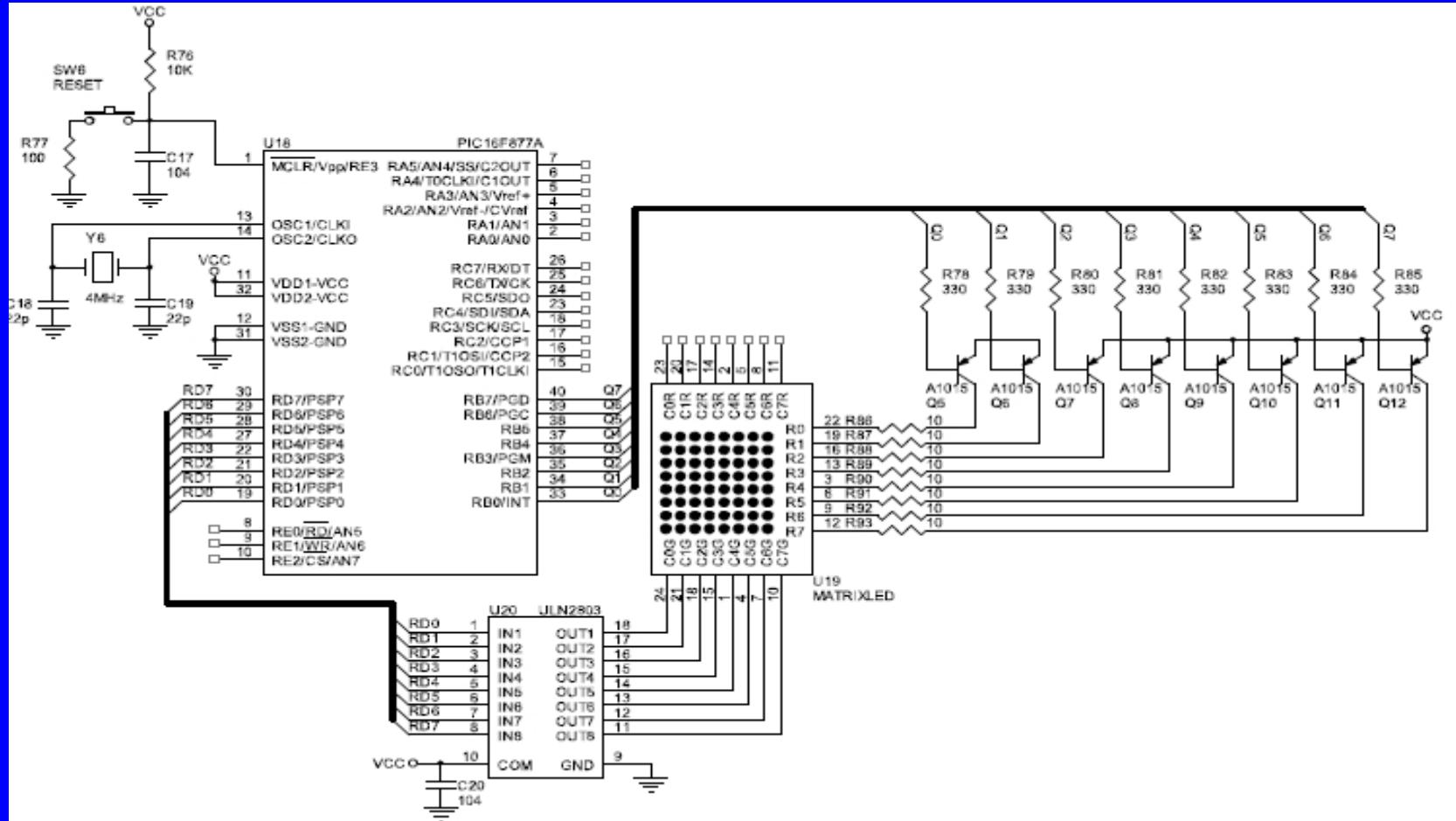
```

```
// *****
// Ham quet ma tran 8 x 8.
// *****
// LED: Hang (Row) -> PNP BJT -> Anode LED -> Kich muc thap (0).
//      Cot (Column) -> NOT GATE -> Cathode LED -> Kich muc cao (1)
// Du lieu hien thi chua trong o nho.
// *****
// Ten:      scanmatrixsta(dat_dis).
// Dau vao:   a[dat_dis] = Du lieu can hien thi.
// Dau ra:    PortD = Column Matrix (NOT Buffer).
//             PortB = Row Matrix (PNP BJT).
// Che do:   STATIC DISP, SCAN COLUMN (ACT = 1) - DISPLAY ROW (ACT = 0).
// *****

void scanmatrixsta(unsigned char dat_dis)
{
// Khai bao bien.
unsigned char scancode,i;
// Dinh nghia ham.
scancode = 0x01;
for(i=dat_dis;i<=dat_dis+7;i++)
{
PORTB = a[i];
PORTD = scancode;
_delay_ms(3);
PORTD = 0x00;
scancode = scancode << 1;
}
}
```

```
// Chuong trinh chinh.  
void main (void)  
{  
// Khai bao bien.  
unsigned char i,dat_dis;  
// Chuong trinh.  
    init_PORTB();  
    init_PORTD();  
    while(1)  
    {  
        dat_dis = 0x00;  
        scanmatrixsta(dat_dis);  
    }  
}
```

# Ví dụ 2:đkhiển led ma trận hiển thị chữ B đứng yên không dịch chuyển



# LCD

**LCD ký tự (Character Liquid Crystal Display):**



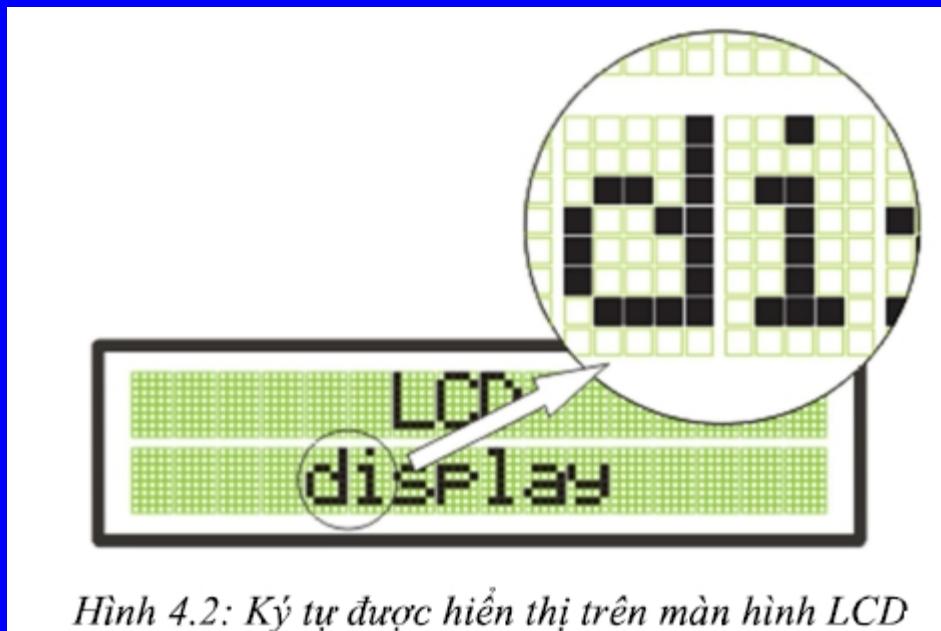
*Hình 4.1: Hình dạng modul LCD ký tự (loại 2 dòng x 16 ký tự).*

2 dòng x 16 ký tự

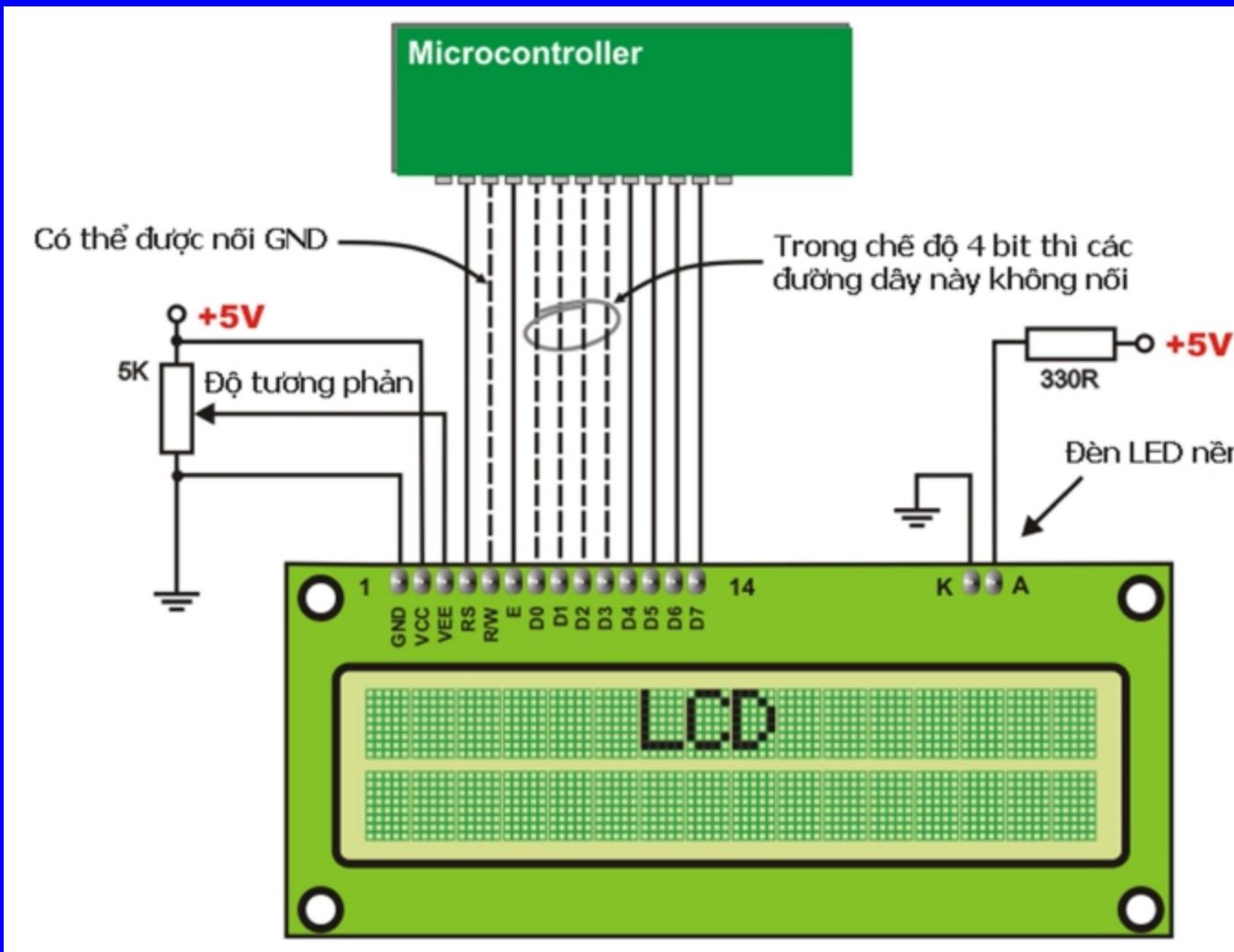
4 dòng x 20 ký tự

# Màn hình LCD

- Có 2 dòng, mỗi dòng có khả năng hiển thị 16 kí tự, mỗi kí tự là một ma trận điểm(dot matrix) 5x8 hoặc 5x11



# Kết nối LCD



# Điều khiển hiển thị trên LCD

- ❑ Bổ sung thư viện **Lcd.c** và **Lcd.h** vào project đang lập trình
  - ❑ **Lcd.h** chứa khai báo lệnh và khai báo kết nối
  - ❑ **Lcd.c** chứa nội dung của các hàm điều khiển LCD

# Điều khiển hiển thị trên LCD

## ❑ Các lệnh cơ bản trong hiển thị trên LCD

### ❑ Khởi động và khai báo cấu hình LCD :

`lcd_init();`

### ❑ Định vị kí tự trên màn hình:

`lcd_gotoxy(độ cột, độ hàng);`

### ❑ Hiển thị 1 kí tự trên màn hình:

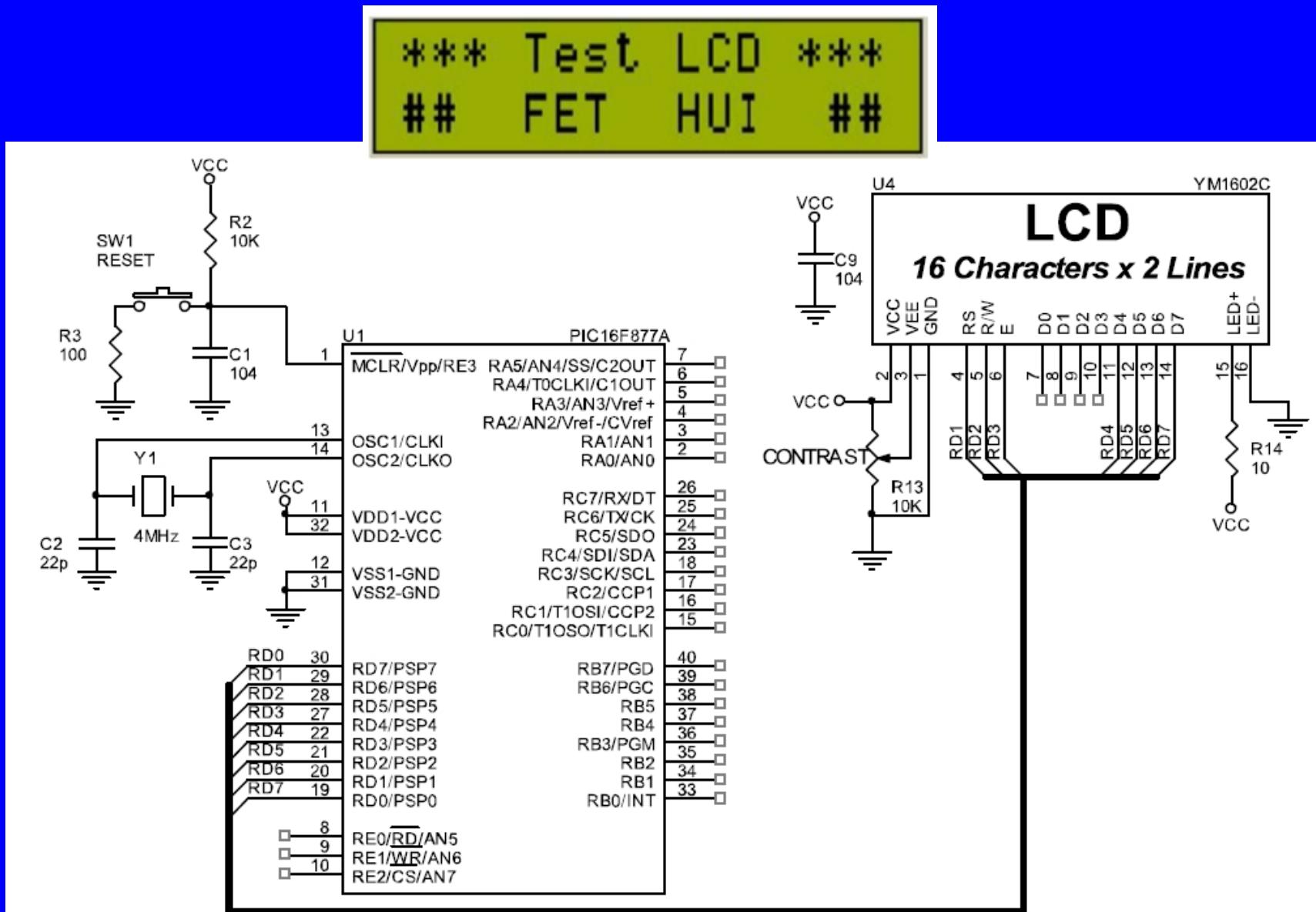
`lcd_putc('kí tự');`

`lcd_putc(mã ASCII);`

### ❑ Hiển thị 1 chuỗi nhiều kí tự trên màn hình

`lcd_puts("chuỗi");`

# Ví dụ 1: Điều khiển LCD hiển thị thông tin sau đúng yên không dịch chuyển

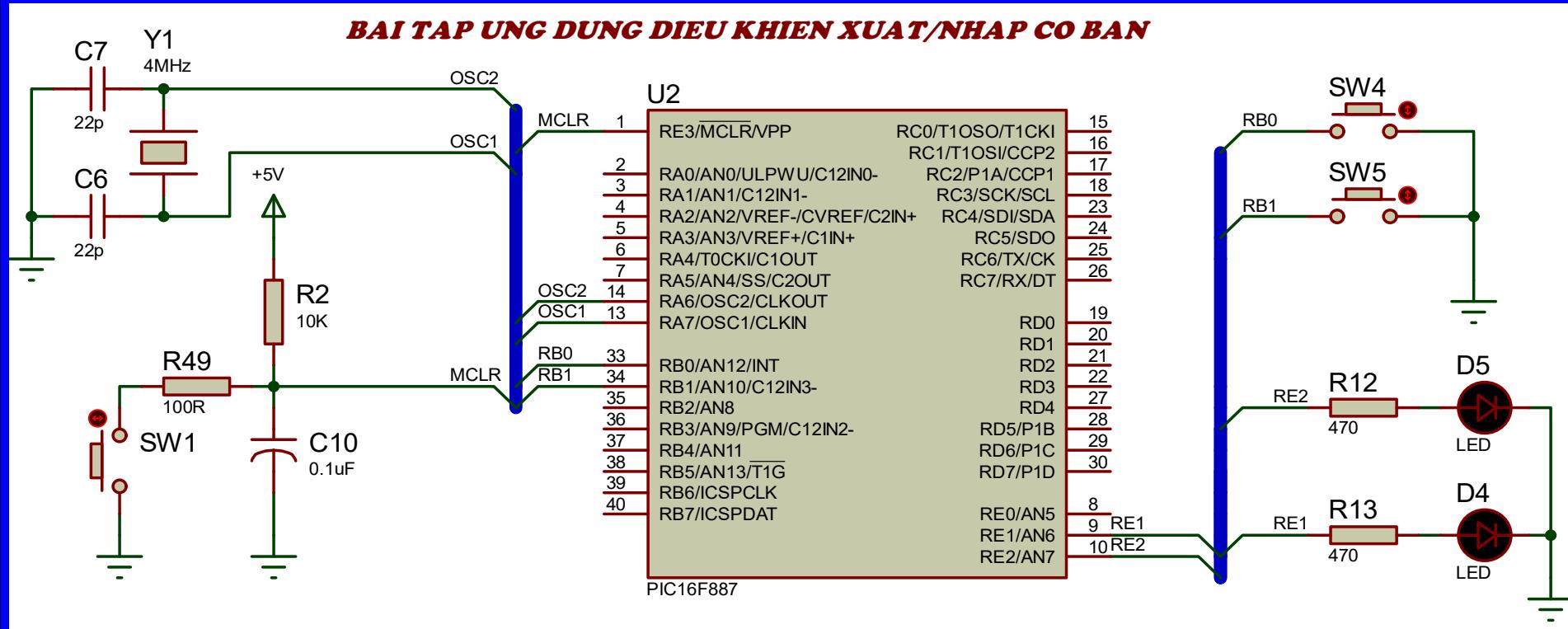


```
#include <htc.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "lcd.h"
__CONFIG(HS & WDTDIS & PWRTEN & UNPROTECT &
BORDIS & LVPDIS);
#define _XTAL_FREQ 4000000
//chuong trinh chinh
void main()
{
lcd_init();
lcd_puts("\f*** Test LCD ***\n## FET HUI ##");
while(1);
}
```

# Bài tập về nút nhấn

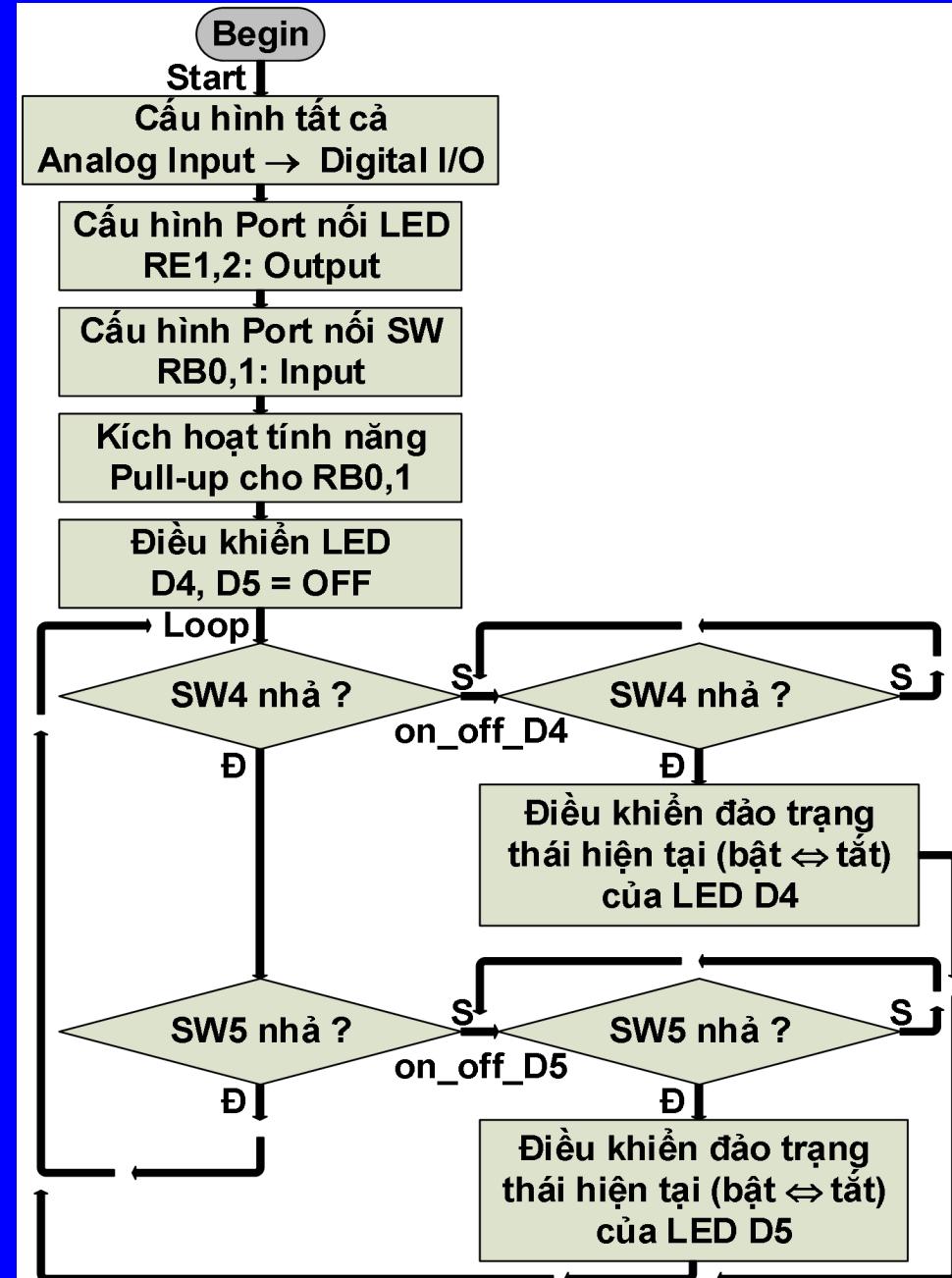
➤ **Ví dụ 1:** Dựa vào sơ đồ, viết chương trình điều khiển LED D4 và D5 bằng SW4 và SW5. Bật/tắt LED sau mỗi lần nhấn SW tương ứng.

- **Sơ đồ nguyên lý:**



# VÍ DỤ MINH HỌA

- **Giải thuật:**



# VÍ DỤ MINH HỌA

- Cấu hình (Hi-Tech C):

```
_CONFIG(FOSC_HS & WDTE_OFF & PWRTE_ON &  
MCLRE_ON & CP_OFF & CPD_OFF & BOREN_OFF &  
IESO_OFF & FCMEN_OFF & LVP_OFF & DEBUG_ON);
```

```
#define _XTAL_FREQ 4000000
```

# VÍ DỤ MINH HỌA

Source: C3\_BT9\_CodeC.c

SIM: C3\_BT9\_CodeC.pdsprj

- Chương trình (Hi-Tech C):

```
void main (void)
{
    ANSEL = 0;
    ANSELH = 0;
    TRISE = 0x00;
    TRISB = 0x03;
    nRBPU = 0;
    WPUB = 0x03;
    PORTE = 0x00;
    while(1)
    {
        if(!RB0)
        {
            __delay_ms(5);
            while(!RB0);
```

```
        }
        __delay_ms(5);
        RE1 = !RE1;
    }
    if(!RB1)
    {
        __delay_ms(5);
        while(!RB1);
        __delay_ms(5);

        RE2 = !RE2;
    }
}
```

- **Ví dụ 2:** Dựa vào sơ đồ trên, Vẽ lưu đồ giải thuật và viết chương trình điều khiển LED D4(nối với chân RD2) và LED D5(RD3) bằng SW4 (nối với chân RB4) và SW5(RB5)
- Khi nhấn-nhả SW4 thì LED D4 sáng, D5 tắt.
- Khi nhấn-nhả SW5 thì LED D4 tắt, D5 sáng
- Ban đầu các led tắt hết

## VÍ DỤ 2

- Chương trình (Hi-Tech C):

```
void main (void)
```

```
{
```

```
ANSEL = 0;
```

```
ANSELH = 0;
```

```
TRISD = 0x00;
```

```
TRISB = 0x30;
```

```
nRBPU =0;
```

```
WPUB = 0b00110000;
```

```
PORTD = 0x00;
```

```
while(1)
```

```
{
```

```
if(!RB4)
```

```
{
```

```
__delay_ms(10);
```

```
while(!RB4);
```

```
RD2 =1;RD3=0;  
}  
if(!RB5)  
{  
__delay_ms(5);  
while(!RB5);  
__delay_ms(5);  
  
RD2 =0;RD3=1;  
}
```

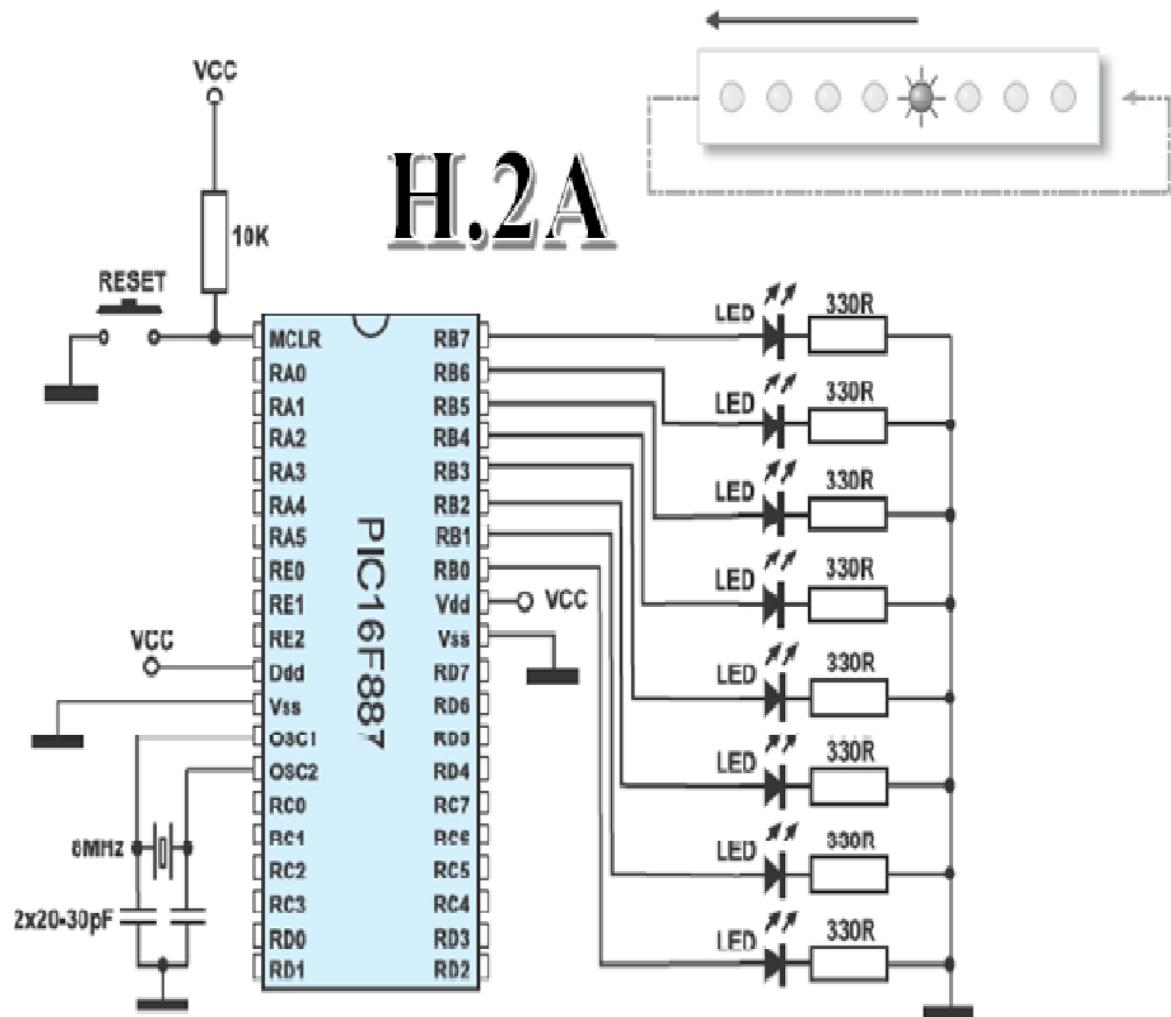
```
}
```

- **Ví dụ 3:** Dựa vào sơ đồ trên, Vẽ lưu đồ giải thuật và viết chương trình điều khiển LED D4(nối với chân RC1) và LED D5(RC2) bằng SW4 (nối với chân RA0) và SW5(RA1)
- Khi nhấn-nhả SW4 thì LED D4 sáng, D5 tắt.
- Khi nhấn-nhả SW5 thì LED D4 tắt, D5 sáng
- Ban đầu các led tắt hết

# Bài tập 1:

Cho mạch điện như hình vẽ H2A. Vẽ khung đồ giải thuật và viết chương trình điều khiển 8 LED đơn thực hiện chu trình sáng đuổi các LED (theo chiều từ dưới lên trên) như trong hình minh họa. Thời gian thay đổi trạng thái là 0,5s.

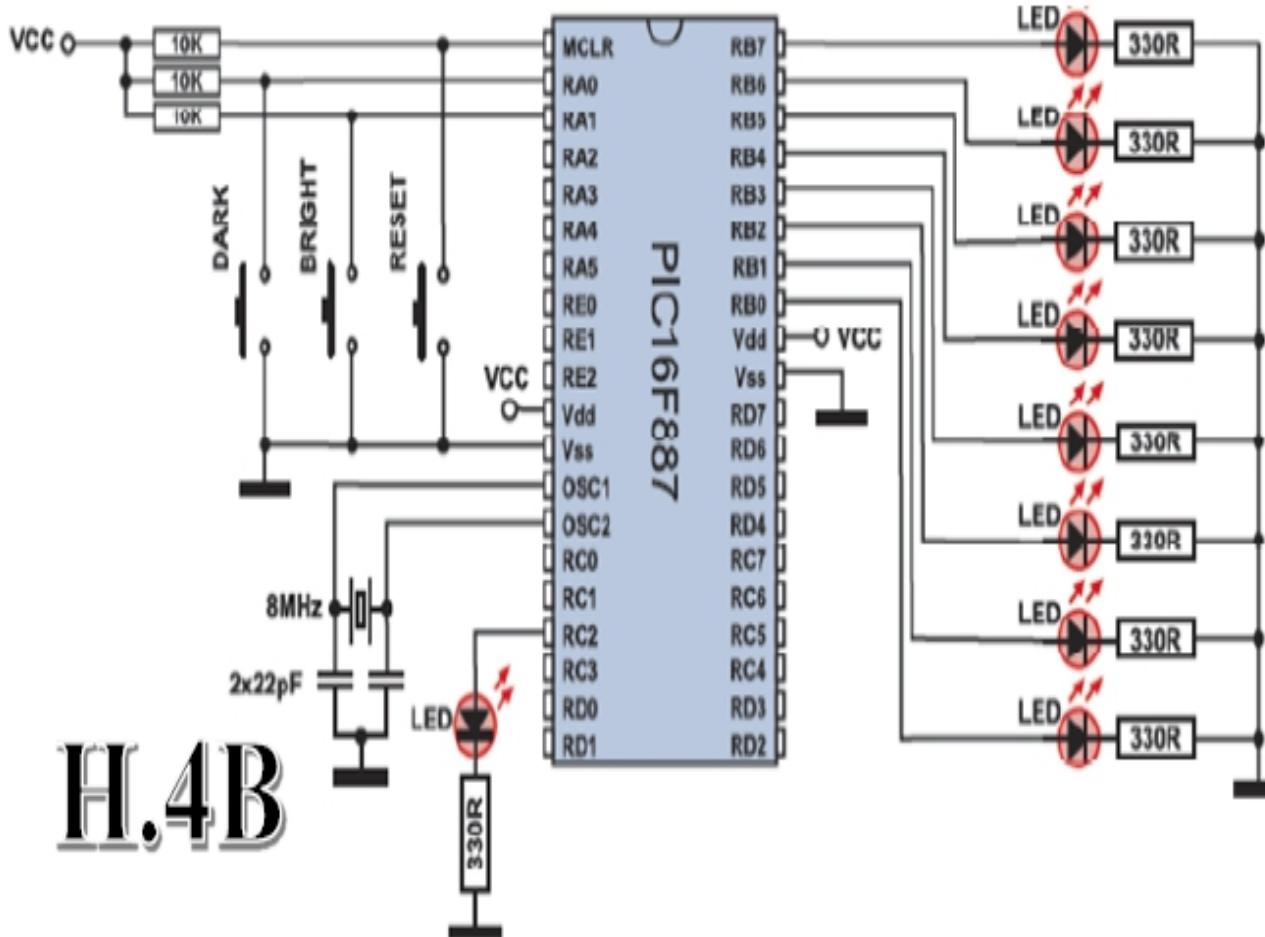
Lưu ý: Ban đầu thì 8 LED tắt hết và chu trình chưa được thực hiện. Sau 5 giây thì chu trình sáng đuổi các LED này mới bắt đầu được thực hiện và diễn ra liên tục trong vòng 12 giây. Cuối cùng thì chu trình sẽ dừng lại và 8 LED sáng hết.



## Bài tập 2:

Cho mạch như Hình H.4B. Vẽ lưu đồ giải thuật và viết chương trình điều khiển 8 LED đơn của Port B sáng hoặc tắt theo sự điều khiển của các nút nhấn. Các LED này sẽ sáng (BRIGHT) hoặc tắt (DARK) sau khi nhấn-nhả nút nhấn tương ứng.

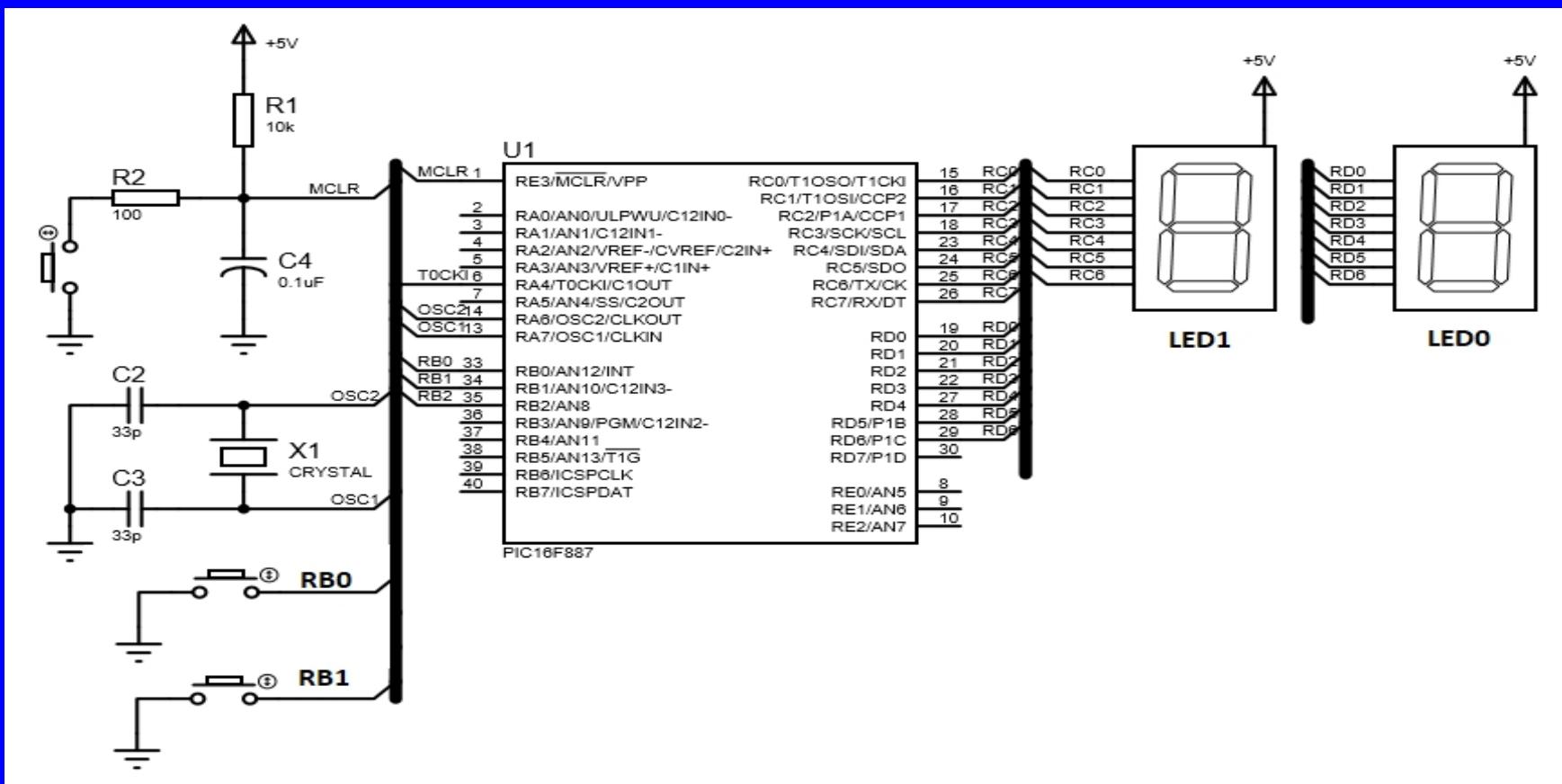
Lưu ý: Ban đầu thì các LED tắt hết. Mỗi lần nhấn-nhả các nút nhấn thì LED đơn tại chân RC2 đều sáng/tắt 1 lần với  $t_{sáng/tắt} = 0.1s$ , sau đó mới bắt đầu thực hiện điều khiển 8 LED đơn của Port B.



# Bài tập 3:

Cho sơ đồ mạch như hình vẽ, viết chương trình cho PIC điều khiển led bảy đoạn theo yêu cầu sau:

- Khi nhấn nút RB0 : led đếm lên 00 đến 35, thời gian chờ tại mỗi trạng thái đếm là 500ms
- Khi nhấn nút RB1: led hiển thị chữ HL chớp tắt 4 lần với  $f = 1\text{Hz}$



# Chương trình C

```
void khai_bao_port (void)
{
    ANSEL = ANSELH = 0;
    TRISD=0;PORTD=0xFF;
    TRISC=0;PORTC=0xFF;
    TRISB = 0X03;
    nRBPU = 0;
    WPUB0 = 1;
    WPUB1 = 1;
}
```

```
void dem00_35 (void)
{
    unsigned char CH,DV,i;
    const unsigned char a[] =
    {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x8
    2,0xF8,0x80,0x90};
    for (i=0; i<36; i++)
    {
        CH=i/10;
        DV=i%10;
        PORTD = a[DV];
        PORTC = a[CH];
        __delay_ms(500);
    }
}
```

```
void chu_HL (unsigned char n)
{
while(n--)
{
PORTC = 0X89;
PORTD = 0XC7;
__delay_ms(500);
PORTC = 0xFF;
PORTD = 0xFF;
__delay_ms(500);
}
}
```

```
void main (void){
    khai_bao_port();
    while (1)
    {
        if(!RB0)
        {
            __delay_ms(10);
            while(!RB0);
            dem00_35();
            PORTC = 0xFF;
            PORTD = 0xFF;
        }
        else if(!RB1)
        {
            __delay_ms(10);
            while(!RB1);
            chu_HL(4);
            PORTC = 0xFF;
            PORTD = 0xFF;
        }
    }
}
```