LCD HD44780 - Giao Tiếp Và Lập Trình Điều Khiển

Trietnguyen, SPKT, 30/6/2007

*** Giới thiệu :**

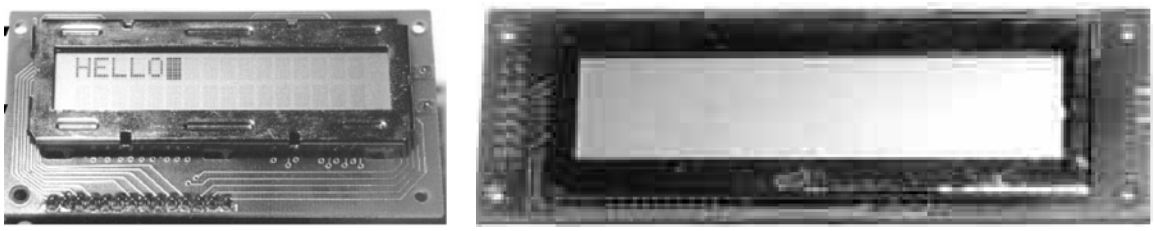
Ngày nay, thiết bị hiển thị LCD (Liquid Crystal Display) được sử dụng trong rất nhiều các ứng dụng của VĐK. LCD có rất nhiều ưu điểm so với các dạng hiển thị khác: Nó có khả năng hiển thị kí tự đa dạng, trực quan (chữ, số và kí tự đồ họa), dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau, tốn rất ít tài nguyên hệ thống và giá thành rẽ …

Bài viết này chủ yếu dựa vào datasheet HD44780 của Hitachi, một loại chíp điều khiển LCD rất thông dụng ở nước ta. Phần đầu là phần giới thiệu và đặc tính của HD44780 có tính chất tham khảo, phần sau là một KIT ứng dụng cụ thể để các bạn có thể tự làm thí nghiệm.

Phần 1 : Tổng Quát Về LCD HD44780

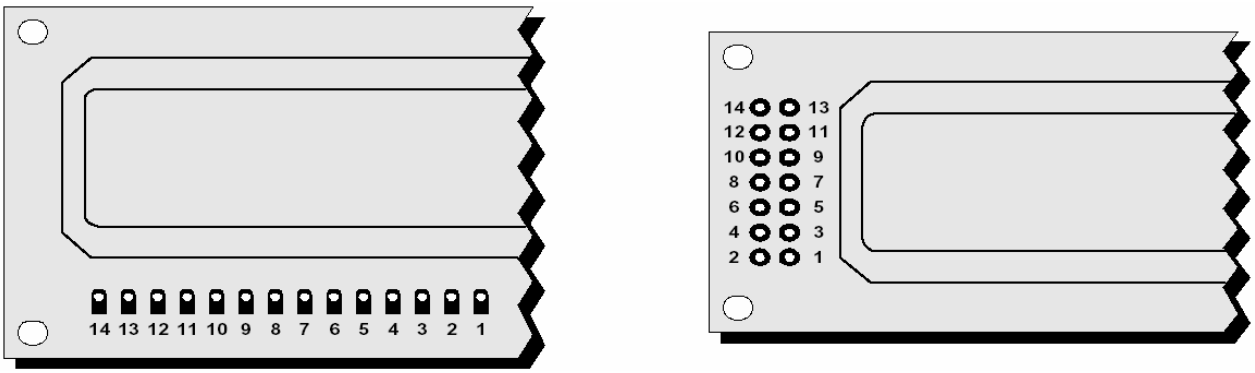
1> Hình dáng và kích thước:

Có rất nhiều loại LCD với nhiều hình dáng và kích thước khác nhau, trên hình 1 là hai loại LCD thông dụng.



Hình 1 : Hình dáng của hai loại LCD thông dụng

Khi sản xuất LCD, nhà sản xuất đã tích hợp chíp điều khiển (HD44780) bên trong lớp vỏ và chỉ đưa các chân giao tiếp cần thiết. Các chân này được đánh số thứ tự và đặt tên như hình 2 :



Hình 2 : Sơ đồ chân của LCD

2> Chức năng các chân :

Chân số	Tên	Chức năng
1	V_{SS}	Chân nối đất cho LCD, khi thiết kế mạch ta nối chân này với GND của mạch điều khiển
2	V_{DD}	Chân cấp nguồn cho LCD, khi thiết kế mạch ta nối chân này với V_{CC}=5V của mạch điều khiển
3	V_{ee}	Chân này dùng để điều chỉnh độ tương phản của LCD.
4	RS	Chân chọn thanh ghi (Register select). Nối chân RS với logic "0" (GND) hoặc logic "1" (V_{CC}) để chọn thanh ghi. + Logic "0": Bus DB0-DB7 sẽ nối với thanh ghi lệnh IR của LCD (ở chế độ "ghi" - write) hoặc nối với bộ đếm địa chỉ của LCD (ở chế độ "đọc" - read) + Logic "1": Bus DB0-DB7 sẽ nối với thanh ghi dữ liệu DR bên trong LCD.
5	R/W	Chân chọn chế độ đọc/ghi (Read/Write). Nối chân R/W với logic "0" để LCD hoạt động ở chế độ ghi, hoặc nối với logic "1" để LCD ở chế độ đọc.
6	E	Chân cho phép (Enable). Sau khi các tín hiệu được đặt lên bus DB0-DB7, các lệnh chỉ được chấp nhận khi có 1 xung cho phép của chân E. + Ở chế độ ghi: Dữ liệu ở bus sẽ được LCD chuyển vào(chấp nhận) thanh ghi bên trong nó khi phát hiện một xung (high-to-low transition) của tín hiệu chân E. + Ở chế độ đọc: Dữ liệu sẽ được LCD xuất ra DB0-DB7 khi phát hiện cạnh lên (low-to-high transition) ở chân E và được LCD giữ ở bus đến khi nào chân E xuống mức thấp.
7-14	DB0-DB7	Tám đường của bus dữ liệu dùng để trao đổi thông tin với MPU. Có 2 chế độ sử dụng 8 đường bus này : + Chế độ 8 bit : Dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7. + Chế độ 4 bit : Dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7 Chi tiết sử dụng 2 giao thức này được đề cập ở phần sau.

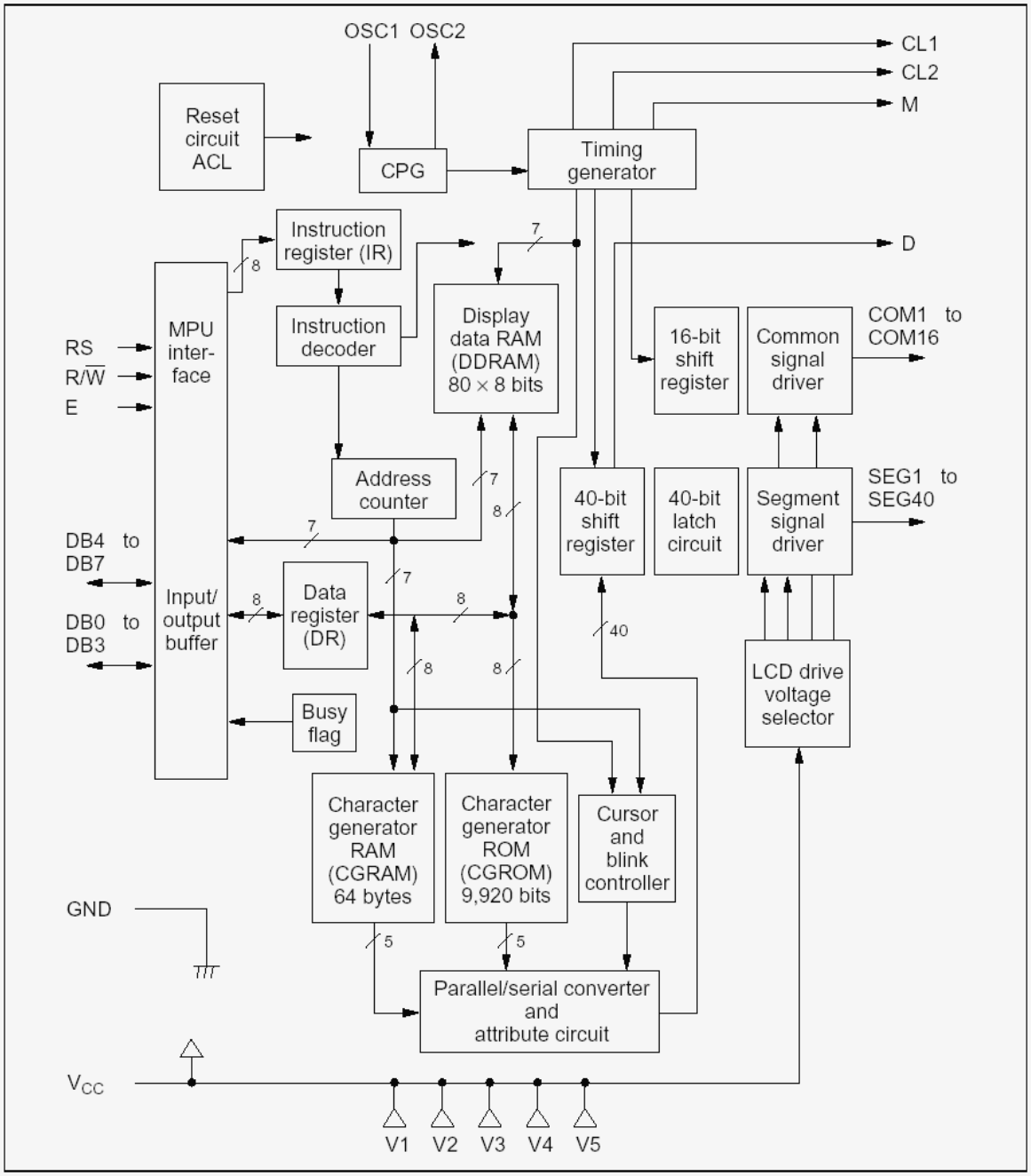
Bảng 1 : Chức năng các chân của LCD

* Ghi chú : Ở chế độ "đọc", nghĩa là MPU sẽ đọc thông tin từ LCD thông qua các chân DBx.

Còn khi ở chế độ "ghi", nghĩa là MPU xuất thông tin điều khiển cho LCD thông qua các chân DBx.

3> Sơ đồ khối của HD44780:

Để hiểu rõ hơn chức năng các chân và hoạt động của chúng, ta tìm hiểu sơ qua chíp HD44780 thông qua các khối cơ bản của nó.



Hình 3 : Sơ đồ khối của HD44780

a> Các thanh ghi :

Chíp HD44780 có 2 thanh ghi 8 bit quan trọng : Thanh ghi lệnh IR (Instructor Register) và thanh ghi dữ liệu DR (Data Register)

- Thanh ghi IR : Để điều khiển LCD, người dùng phải "ra lệnh" thông qua tám đường bus DB0-DB7. Mỗi lệnh được nhà sản xuất LCD đánh địa chỉ rõ ràng. Người dùng chỉ việc cung cấp địa chỉ lệnh bằng cách nạp vào thanh ghi IR. Nghĩa là, khi ta nạp vào thanh ghi IR một chuỗi 8 bit, chíp HD44780 sẽ tra bảng mã lệnh tại địa chỉ mà IR cung cấp và thực hiện lệnh đó.

VD : Lệnh "hiển thị màn hình" có địa chỉ lệnh là 00001100 (DB7…DB0)

Lệnh “hiển thị màn hình và con trỏ” có mã lệnh là 00001110

- Thanh ghi DR : Thanh ghi DR dùng để chứa dữ liệu 8 bit để ghi vào vùng RAM DDRAM hoặc CGRAM (ở chế độ ghi) hoặc dùng để chứa dữ liệu từ 2 vùng RAM này gởi ra cho MPU (ở chế độ đọc). Nghĩa là, khi MPU ghi thông tin vào DR, mạch nội bên trong chíp sẽ tự động ghi thông tin này vào DDRAM hoặc CGRAM. Hoặc khi thông tin về địa chỉ được ghi vào IR, dữ liệu ở địa chỉ này trong vùng RAM nội của HD44780 sẽ được chuyển ra DR để truyền cho MPU.

→ Bằng cách điều khiển chân RS và R/W chúng ta có thể chuyển qua lại giữ 2 thanh ghi này khi giao tiếp với MPU. Bảng sau đây tóm tắt lại các thiết lập đối với hai chân RS và R/W theo mục đích giao tiếp.

RS	R/W	Khi cần
0	**0**	Ghi vào thanh ghi IR để ra lệnh cho LCD (VD: cần display clear,…)
0	**1**	Đọc cờ bận ở DB7 và giá trị của bộ đếm địa chỉ ở DB0-DB6
1	**0**	Ghi vào thanh ghi DR
1	**1**	Đọc dữ liệu từ DR

Bảng 2 : Chức năng chân RS và R/W theo mục đích sử dụng

b> Cờ báo bận BF: (Busy Flag)

Khi thực hiện các hoạt động bên trong chíp, mạch nội bên trong cần một khoảng thời gian để hoàn tất. Khi đang thực thi các hoạt động bên trong chip như thế, LCD bỏ qua mọi giao tiếp với bên ngoài và bật cờ BF (thông qua chân DB7 khi có thiết lập RS=0, R/W=1) lên để báo cho MPU biết nó đang “bận”. Dĩ nhiên, khi xong việc, nó sẽ đặt cờ BF lại mức 0.

c> Bộ đếm địa chỉ AC : (Address Counter)

Như trong sơ đồ khối, thanh ghi IR không trực tiếp kết nối với vùng RAM (DDRAM và CGRAM) mà thông qua bộ đếm địa chỉ AC. Bộ đếm này lại nối với 2 vùng RAM theo kiểu rẽ nhánh. Khi một địa chỉ lệnh được nạp vào thanh ghi IR, thông tin được nối trực tiếp cho 2 vùng RAM nhưng việc chọn lựa vùng RAM tương tác đã được bao hàm trong mã lệnh.

Sau khi ghi vào (đọc từ) RAM, bộ đếm AC tự động tăng lên (giảm đi) 1 đơn vị và nội dung của AC được xuất ra cho MPU thông qua DB0-DB6 khi có thiết lập RS=0 và R/W=1 (xem bảng tóm tắt RS - R/W).

Lưu ý: Thời gian cập nhật AC không được tính vào thời gian thực thi lệnh mà được cập nhật sau khi cờ BF lên mức cao (not busy), cho nên khi lập trình hiển thị, bạn phải delay một khoảng t_{ADD} khoảng 4uS-5uS (ngay sau khi BF=1) trước khi nạp dữ liệu mới. Xem thêm hình bên dưới.

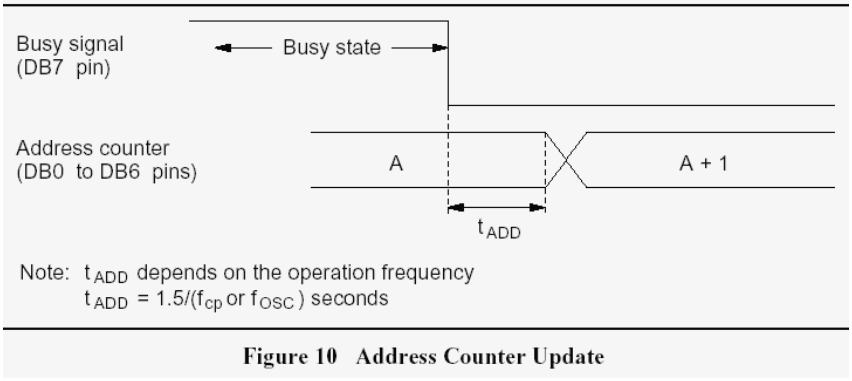


Figure 10 Address Counter Update

Hình 4 : Giản đồ xung cập nhật AC

d> Vùng RAM hiển thị DDRAM : (Display Data RAM)

Đây là vùng RAM dùng để hiển thị, nghĩa là ứng với một địa chỉ của RAM là một ô kí tự trên màn hình và khi bạn ghi vào vùng RAM này một mã 8 bit, LCD sẽ hiển thị tại vị trí tương ứng trên màn hình một kí tự có mã 8 bit mà bạn đã cung cấp. Hình sau đây sẽ trình bày rõ hơn mối liên hệ này :

Display position (digit)	1	2	3	4	5	.....	79	80
DDRAM address (hexadecimal)	00	01	02	03	04	.....	4E	4F

Figure 2 1-Line Display

Display position	1	2	3	4	5	.....	39	40
DDRAM address (hexadecimal)	00	01	02	03	04	.....	26	27
	40	41	42	43	44	.....	66	67

Figure 4 2-Line Display

Hình 4 : Mối liên hệ giữa địa chỉ của DDRAM và vị trí hiển thị của LCD

Vùng RAM này có 80x8 bit nhớ, nghĩa là chứa được 80 kí tự mã 8 bit. Những vùng RAM còn lại không dùng cho hiển thị có thể dùng như vùng RAM đa mục đích.

Lưu ý là để truy cập vào DDRAM, ta phải cung cấp địa chỉ cho AC theo mã HEX

e> Vùng ROM chứa kí tự CGROM: Character Generator ROM

Vùng ROM này dùng để chứa các mẫu kí tự loại 5x8 hoặc 5x10 điểm ảnh/kí tự, và định địa chỉ bằng 8 bit. Tuy nhiên, nó chỉ có 208 mẫu kí tự 5x8 và 32 mẫu kí tự kiểu 5x10 (tổng cộng là 240 thay vì $2^8 = 256$ mẫu kí tự). Người dùng không thể thay đổi vùng ROM này.

Table 2 Example of Correspondence between EPROM Address Data and Character Pattern (5 × 8 Dots)

EPROM Address												Data					
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	O4	O3	O2	O1	LSB O0	
								0	0	0	0	1	0	0	0	0	
								0	0	0	1	1	0	0	0	0	
								0	0	1	0	1	0	1	1	0	
								0	0	1	1	1	1	0	0	1	
								0	1	0	0	1	0	0	0	1	
								0	1	0	1	1	0	0	0	1	
								0	1	1	0	1	1	1	1	0	
0	1	1	0	0	0	1	0	0	1	1	1	0	0	0	0	0	← Cursor position
								1	0	0	0	0	0	0	0	0	
								1	0	0	1	0	0	0	0	0	
								1	0	1	0	0	0	0	0	0	
								1	0	1	1	0	0	0	0	0	
								1	1	0	0	0	0	0	0	0	
								1	1	0	1	0	0	0	0	0	
								1	1	1	0	0	0	0	0	0	
								1	1	1	1	0	0	0	0	0	
Character code								Line position									

Notes:
1. EPROM addresses A11 to A4 correspond to a character code.
2. EPROM addresses A3 to A0 specify a line position of the character pattern.
3. EPROM data O4 to O0 correspond to character pattern data.
4. EPROM data O5 to O7 must be specified as 0.
5. A lit display position (black) corresponds to a 1.
6. Line 9 and the following lines must be blanked with 0s for a 5 × 8 dot character fonts.

Hình 5 : Mối liên hệ giữa địa chỉ của ROM và dữ liệu tạo mẫu kí tự.

Như vậy, để có thể ghi vào vị trí thứ x trên màn hình một kí tự y nào đó, người dùng phải ghi vào vùng DDRAM tại địa chỉ x (xem bảng mối liên hệ giữa DDRAM và vị trí hiển thị) một chuỗi mã kí tự 8 bit trên CGROM. Chú ý là trong bảng mã kí tự trong CGROM ở hình bên dưới có mã ROM A00.

Ví dụ : Ghi vào DDRAM tại địa chỉ "01" một chuỗi 8 bit "01100010" thì trên LCD tại ô thứ 2 từ trái sang (dòng trên) sẽ hiển thị kí tự "b".

Table 4 Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	Ł	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	■

Note: The user can specify any pattern for character-generator RAM.

Bảng 3 : Bảng mã kí tự (ROM code A00)

f> Vùng RAM chứa kí tự đồ họa CGRAM : (Character Generator RAM)

Như trên bảng mã kí tự, nhà sản xuất dành vùng có địa chỉ byte cao là 0000 để người dùng có thể tạo các mẫu kí tự đồ họa riêng. Tuy nhiên dung lượng vùng này rất hạn chế: Ta chỉ có thể tạo 8 kí tự loại 5x8 điểm ảnh, hoặc 4 kí tự loại 5x10 điểm ảnh.

Để ghi vào CGRAM, hãy xem hình 6 bên dưới.

Table 5 Relationship between CGRAM Addresses, Character Codes (DDRAM) and Character Patterns (CGRAM Data)

For 5 × 8 dot character patterns

Character Codes (DDRAM data)	CGRAM Address	Character Patterns (CGRAM data)	
7 6 5 4 3 2 1 0 (High … Low)	5 4 3 2 1 0 (High … Low)	7 6 5 4 3 2 1 0 (High … Low)	
0 0 0 0 * 0 0 0	0 0 0	* * * 1 1 1 1 0	Character pattern (1)
	0 0 1	* 1 0 0 0 1	
	0 1 0	* 1 0 0 0 1	
	0 1 1	* 1 1 1 1 0	
	1 0 0	* 1 0 1 0 0	
	1 0 1	* 1 0 0 1 0	
	1 1 0	* 1 0 0 0 1	
	1 1 1	* * * 0 0 0 0 0	Cursor position
0 0 0 0 * 0 0 1	0 0 0	* * * 1 0 0 0 1	Character pattern (2)
	0 0 1	* 0 1 0 1 0	
	0 1 0	* 1 1 1 1 1	
	0 1 1	* 0 0 1 0 0	
	1 0 0	* 1 1 1 1 1	
	1 0 1	* 0 0 1 0 0	
	1 1 0	* 0 0 1 0 0	
	1 1 1	* * * 0 0 0 0 0	Cursor position
	0 0 0	* * *	
	0 0 1	*	
0 0 0 0 * 1 1 1	1 1 1		
	1 0 0	*	
	1 0 1	*	
	1 1 0	*	
	1 1 1	* * *	

Notes:
1. Character code bits 0 to 2 correspond to CGRAM address bits 3 to 5 (3 bits: 8 types).
2. CGRAM address bits 0 to 2 designate the character pattern line position. The 8th line is the cursor position and its display is formed by a logical OR with the cursor.
   Maintain the 8th line data, corresponding to the cursor display position, at 0 as the cursor display.
   If the 8th line data is 1, 1 bits will light up the 8th line regardless of the cursor presence.
3. Character pattern row positions correspond to CGRAM data bits 0 to 4 (bit 4 being at the left).
4. As shown Table 5, CGRAM character patterns are selected when character code bits 4 to 7 are all 0. However, since character code bit 3 has no effect, the R display example above can be selected by either character code 00H or 08H.
5. 1 for CGRAM data corresponds to display selection and 0 to non-selection.

* Indicates no effect.

Hình 6 : Mối liên hệ giữa địa chỉ của CGRAM, dữ liệu của CGRAM, và mã kí tự.

4> Tập lệnh của LCD :

Trước khi tìm hiểu tập lệnh của LCD, sau đây là một vài chú ý khi giao tiếp với LCD :

* Tuy trong sơ đồ khối của LCD có nhiều khối khác nhau, nhưng khi lập trình điều khiển LCD ta chỉ có thể tác động trực tiếp được vào 2 thanh ghi DR và IR thông qua các chân DBx, và ta phải thiết lập chân RS, R/W phù hợp để chuyển qua lại giữ 2 thanh ghi này. (xem bảng 2)

* Với mỗi lệnh, LCD cần một khoảng thời gian để hoàn tất, thời gian này có thể khá lâu đối với tốc độ của MPU, nên ta cần kiểm tra cờ BF hoặc đợi (delay) cho LCD thực thi xong lệnh hiện hành mới có thể ra lệnh tiếp theo.

* Địa chỉ của RAM (AC) sẽ tự động tăng (giảm) 1 đơn vị, mỗi khi có lệnh ghi vào RAM. (Điều này giúp chương trình gọn hơn)

* Các lệnh của LCD có thể chia thành 4 nhóm như sau :

- Các lệnh về kiểu hiển thị. VD : Kiểu hiển thị (1 hàng / 2 hàng), chiều dài dữ liệu (8 bit / 4 bit), …
- Chỉ định địa chỉ RAM nội.
- Nhóm lệnh truyền dữ liệu trong RAM nội.
- Các lệnh còn lại . (!!!)

Bảng 4 : Tập lệnh của LCD

Tên lệnh	Hoạt động	t_{exe} (max)
Clear Display	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 0 0 1 Lệnh Clear Display (xóa hiển thị) sẽ ghi một khoảng trống-blank (mã hiển kí tự 20H) vào tất cả ô nhớ trong DDRAM, sau đó trả bộ đếm địa AC=0, trả lại kiểu hiện thị gốc nếu nó bị thay đổi. Nghĩa là : Tắt hiển thị, con trỏ dời về góc trái (hàng đầu tiên), chế độ tăng AC.	
Return home	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 0 1 * Lệnh Return home trả bộ đếm địa chỉ AC về 0, trả lại kiểu hiển thị gốc nếu nó bị thay đổi. Nội dung của DDRAM không thay đổi.	1.52 ms
Entry mode set	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 1 [I/D] [S] **I/D** : Tăng (I/D=1) hoặc giảm (I/D=0) bộ đếm địa chỉ hiển thị AC 1 đơn vị mỗi khi có hành động ghi hoặc đọc vùng DDRAM. Vị trí con trỏ cũng di chuyển theo sự tăng giảm này. **S** : Khi S=1 toàn bộ nội dung hiển thị bị dịch sang phải (I/D=0) hoặc sang trái (I/D=1) mỗi khi có hành động ghi vùng DDRAM. Khi S=0: không dịch nội dung hiển thị. Nội dung hiển thị không dịch khi đọc DDRAM hoặc đọc/ghi vùng CGRAM.	37 uS

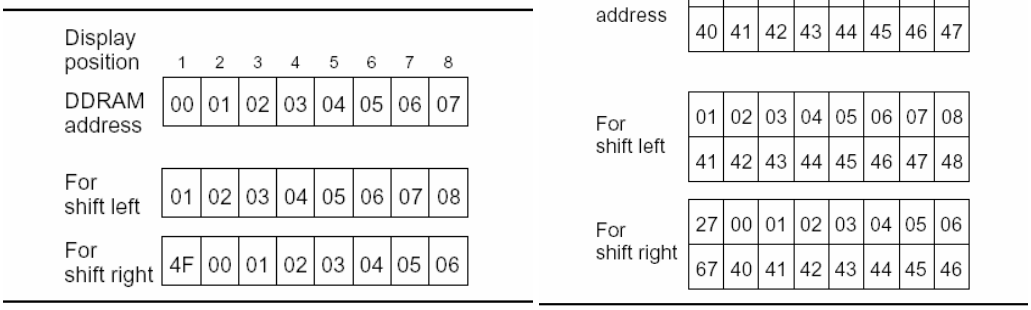
<table>
<tr>
<td></td>
<td>


Figure 3 1-Line by 8-Character Display Example Figure 5 2-Line by 8-Character Display Example

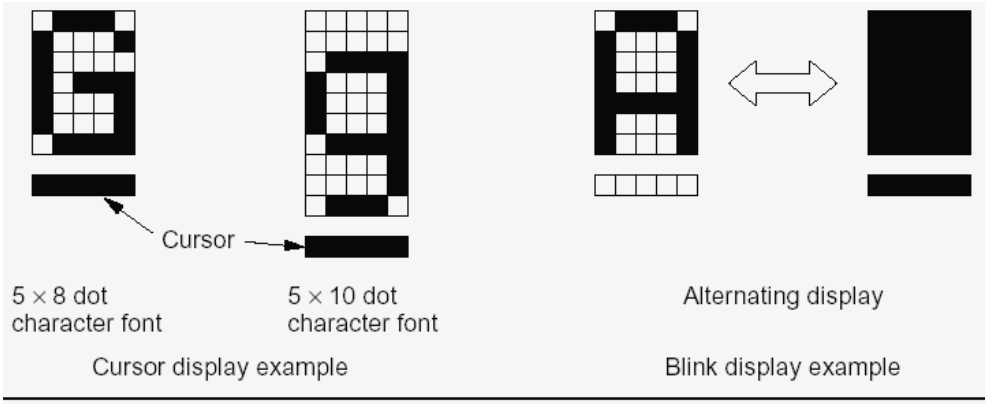
Hình 7 : Hoạt động dịch trái và dịch phải nội dung hiển thị
</td>
<td></td>
</tr>
<tr>
<td>Display on/off control</td>
<td>
Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 0 1 [D] [C] [B]
D: Hiển thị màn hình khi D=1 và ngược lại. Khi tắt hiển thị, nội dung DDRAM không thay đổi.
C: Hiển thị con trỏ khi C=1 và ngược lại. Vị trí và hình dạng con trỏ, xem hình 8
B: Nhấp nháy kí tự tại vị trí con trỏ khi B=1 và ngược lại. Xem thêm hình 8 về kiểu nhấp nháy. Chu kì nhấp nháy khoảng 409,6ms khi mạch dao động nội LCD là 250kHz.


Figure 12 Cursor and Blinking

Hình 8: Kiểu con trỏ, kiểu kí tự và nhấp nháy kí tự
</td>
<td>37uS</td>
</tr>
<tr>
<td>Cursor or display shift</td>
<td>
Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 1 [S/C] [R/L] * *
Lệnh Cursor or display shift dịch chuyển con trỏ hay dữ liệu hiển thị sang trái mà không cần hành động ghi/đọc dữ liệu. Khi hiển thị kiểu 2 dòng, con trỏ sẽ nhảy xuống dòng dưới khi dịch qua vị trí thứ 40 của hàng đầu tiên. Dữ liệu hàng đầu và hàng 2 dịch cùng một lúc. Chi tiết sử dụng xem bảng bên dưới:
<table>
<tr><th>S/C</th><th>R/L</th><th>Hoạt động</th></tr>
<tr><td>0</td><td>0</td><td>Dịch vị trí con trỏ sang trái (Nghĩa là giảm AC một đơn vị).</td></tr>
<tr><td>0</td><td>1</td><td>Dịch vị trí con trỏ sang phải (Tăng AC lên 1 đơn vị).</td></tr>
<tr><td>1</td><td>0</td><td>Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo.</td></tr>
<tr><td>1</td><td>1</td><td>Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.</td></tr>
</table>
</td>
<td>37uS</td>
</tr>
</table>

	Bảng 5: Hoạt động lệnh Cursor or display shift	
Function set	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 1 [DL] [N] [F] * * **DL**: Khi DL=1, LCD giao tiếp với MPU bằng giao thức 8 bit (từ bit DB7 đến DB0). Ngược lại, giao thức giao tiếp là 4 bit (từ bit DB7 đến bit DB0). Khi chọn giao thức 4 bit, dữ liệu được truyền/nhận 2 lần liên tiếp. với 4 bit cao gởi/nhận trước, 4 bit thấp gởi/nhận sau. **N** : Thiết lập số hàng hiển thị. Khi N=0 : hiển thị 1 hàng, N=1: hiển thị 2 hàng. **F** : Thiết lập kiểu kí tự. Khi F=0: kiểu kí tự 5x8 điểm ảnh, F=1: kiểu kí tự 5x10 điểm ảnh. **** Chú ý:*** • ***Chỉ thực hiện thay đổi Function set ở đầu chương trình. Và sau khi được thực thi 1 lần, lệnh thay đổi Function set không được LCD chấp nhận nữa ngoại trừ thiết lập chuyển đổi giao thức giao tiếp.*** • ***Không thể hiển thị kiểu kí tự 5x10 điểm ảnh ở kiểu hiển thị 2 hàng***	37uS
Set CGRAM address	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 1 [ACG][ACG][ACG][ACG][ACG][ACG] Lệnh này ghi vào AC địa chỉ của CGRAM. Kí hiệu [ACG] chỉ 1 bit của chuỗi dữ liệu 6 bit. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ CGRAM tại địa chỉ đã được chỉ định.	37uS
Set DDRAM address	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 1 [AD] [AD] [AD] [AD] [AD] [AD] [AD] Lệnh này ghi vào AC địa chỉ của DDRAM, dùng khi cần thiết lập tọa độ hiển thị mong muốn. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ DDRAM tại địa chỉ đã được chỉ định. Khi ở chế độ hiển thị 1 hàng: địa chỉ có thể từ 00H đến 4FH. Khi ở chế độ hiển thị 2 hàng, địa chỉ từ 00h đến 27H cho hàng thứ nhất, và từ 40h đến 67h cho hàng thứ 2. Xem chi tiết ở hình 4.	37uS
Read BF and address	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [BF] [AC] [AC] [AC] [AC] [AC] [AC] [AC] (RS=0, R/W=1) Như đã đề cập trước đây, khi cờ BF bật, LCD đang làm việc và lệnh tiếp theo (nếu có) sẽ bị bỏ qua nếu cờ BF chưa về mức thấp. Cho nên, khi lập trình điều khiển, bạn phải kiểm tra cờ BF trước khi ghi dữ liệu vào LCD. Khi đọc cờ BF, giá trị của AC cũng được xuất ra các bit [AC]. Nó là địa chỉ của CG hay DDRAM là tùy thuộc vào lệnh trước đó	0uS
Write data to CG or DDRAM	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [Write data] (RS=1, R/W=0) Khi thiết lập RS=1, R/W=0, dữ liệu cần ghi được đưa vào các chân DBx từ mạch ngoài sẽ được LCD chuyển vào trong LCD tại địa chỉ được xác định từ lệnh ghi địa chỉ trước đó (lệnh ghi địa chỉ cũng xác định luôn vùng RAM cần ghi) Sau khi ghi, bộ đếm địa chỉ AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode. Lưu ý là thời gian cập nhật AC không tính vào thời gian thực thi lệnh. Chi tiết về giao thức Ghi dữ liệu, xin xem hình 10.	37uS t_{ADD} 4uS
Read data from CG	**Mã lệnh** : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [Read data] (RS=1, R/W=1)	37uS t_{ADD}

or DDRAM	Khi thiết lập RS=1, R/W=1,dữ liệu từ CG/DDRAM được chuyển ra MPU thông qua các chân DBx (địa chỉ và vùng RAM đã được xác định bằng lệnh ghi địa chỉ trước đó). Sau khi đọc, AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode, tuy nhiên nội dung hiển thị không bị dịch bất chấp chế độ Entry mode. Chi tiết hơn về giao thức đọc dữ liệu, xin xem hình 11.	4uS

5> Giao tiếp giữa LCD và MPU :

a> Đặc tính điện của các chân giao tiếp :

LCD sẽ bị hỏng nghiêm trọng, hoặc hoạt động sai lệch nếu bạn vi phạm khoảng đặc tính điện sau đây:

Chân cấp nguồn (Vcc-GND)	Min:-0.3V , Max+7V
Các chân ngõ vào (DBx,E,…)	Min:-0.3V , Max:(Vcc+0.3V)
Nhiệt độ hoạt động	Min:-30C , Max:+75C
Nhiệt độ bảo quản	Min:-55C , Max:+125C

Bảng 6 : Maximun Rating

Đặc tính điện làm việc điển hình: (Đo trong điều kiện hoạt động Vcc = 4.5V đến 5.5V, T = -30 đến +75C)

Chân cấp nguồn Vcc-GND	2.7V đến 5.5V
Điện áp vào mức cao V_{IH}	2.2V đến Vcc
Điện áp vào mức thấp V_{IL}	-0.3V đến 0.6V
Điện áp ra mức cao (DB0-DB7)	Min 2.4V (khi I_{OH} = -0.205mA)
Điện áp ra mức thấp (DB0-DB7)	Max 0.4V (khi I_{OL} = 1.2mA)
Dòng điện ngõ vào (input leakage current) I_{LI}	-1uA đến 1uA (khi V_{IN} = 0 đến Vcc)
Dòng điện cấp nguồn I_{CC}	350uA(typ.) đến 600uA
Tần số dao động nội f_{OSC}	190kHz đến 350kHz (điển hình là 270kHz)

Bảng 7: Miền làm việc bình thường

b> Sơ đồ nối mạch điển hình:

- Sơ đồ mạch kết nối giữa mô đun LCD và VĐK 89S52 (8 bit).

- Sơ đồ mạch kết nối giữa môđun LCD và VĐK (4 bit).

c> Bus Timing:

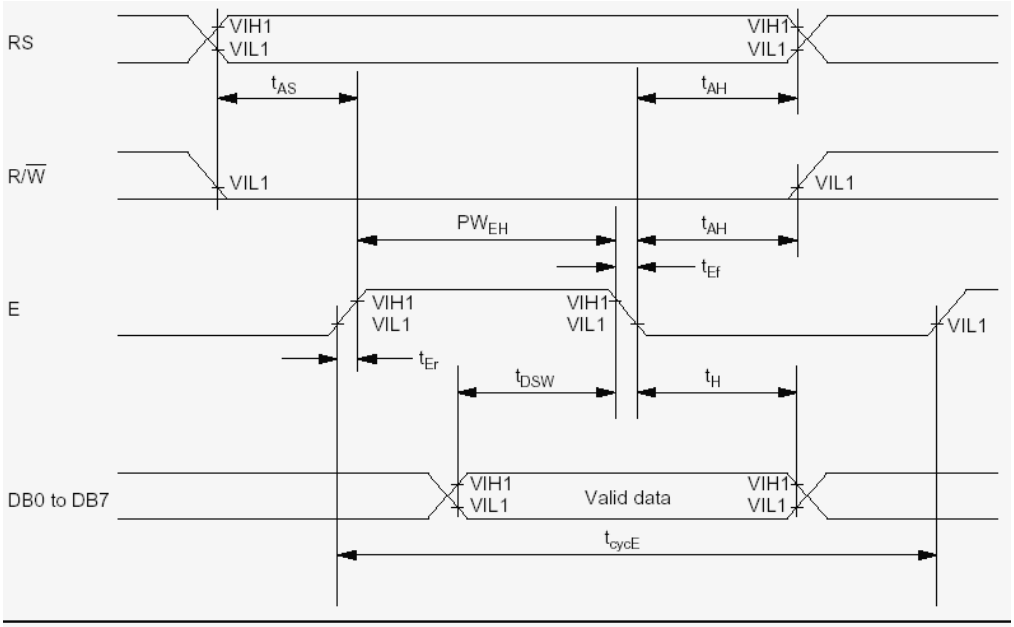
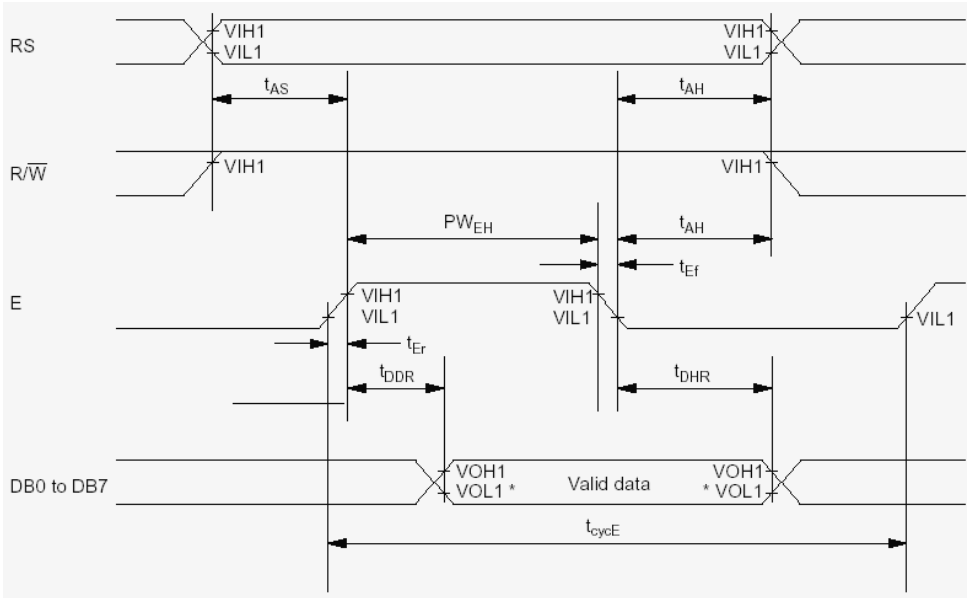


Figure 25 Write Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data set-up time	t_{DSW}	80	—	—	
Data hold time	t_H	10	—	—	



Note: * VOL1 is assumed to be 0.8 V at 2 MHz operation.

Figure 26 Read Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data delay time	t_{DDR}	—	—	160	
Data hold time	t_{DHR}	5	—	—	

6> Khởi tạo LCD:

Khởi tạo là việc thiết lập các thông số làm việc ban đầu. Đối với LCD, khởi tạo giúp ta thiết lập các giao thức làm việc giữa LCD và MPU. Việc khởi tạo chỉ được thực hiện 1 lần duy nhất ở đầu chương trình điều khiển LCD và bao gồm các thiết lập sau :

- Display clear : Xóa/không xóa toàn bộ nội dung hiển thị trước đó.
- Function set : Kiểu giao tiếp 8bit/4bit, số hàng hiển thị 1hàng/2hàng, kiểu kí tự 5x8/5x10.
- Display on/off control: Hiển thị/tắt màn hình, hiển thị/tắt con trỏ, nhấp nháy/không nhấp nháy.
- Entry mode set : các thiết lập kiểu nhập kí tự như: Dịch/không dịch, tự tăng/giảm (Increment).

a> Mạch khởi tạo bên trong chíp HD44780:

Mỗi khi được cấp nguồn, mạch khởi tạo bên trong LCD sẽ tự động khởi tạo cho nó. Và trong thời gian khởi tạo này cờ BF bật lên 1, đến khi việc khởi tạo hoàn tất cờ BF còn giữ trong khoảng 10ms sau khi Vcc đạt đến 4.5V (vì 2.7V thì LCD đã hoạt động). Mạch khởi tạo nội sẽ thiết lập các thông số làm việc của LCD như sau:

- Display clear : Xóa toàn bộ nội dung hiển thị trước đó.
- Function set: DL=1 : 8bit; N=0 : 1 hàng; F=0 : 5x8
- Display on/off control: D=0 : Display off; C=0 : Cursor off; B=0 : Blinking off.
- Entry mode set: I/D =1 : Tăng; S=0 : Không dịch.

Như vậy sau khi mở nguồn, bạn sẽ thấy màn hình LCD giống như chưa mở nguồn do toàn bộ hiển thị tắt. Do đó, ta phải khởi tạo LCD bằng lệnh.

b> Khởi tạo bằng lệnh: (chuỗi lệnh)

Việc khởi tạo bằng lệnh phải tuân theo lưu đồ sau của nhà sản xuất :

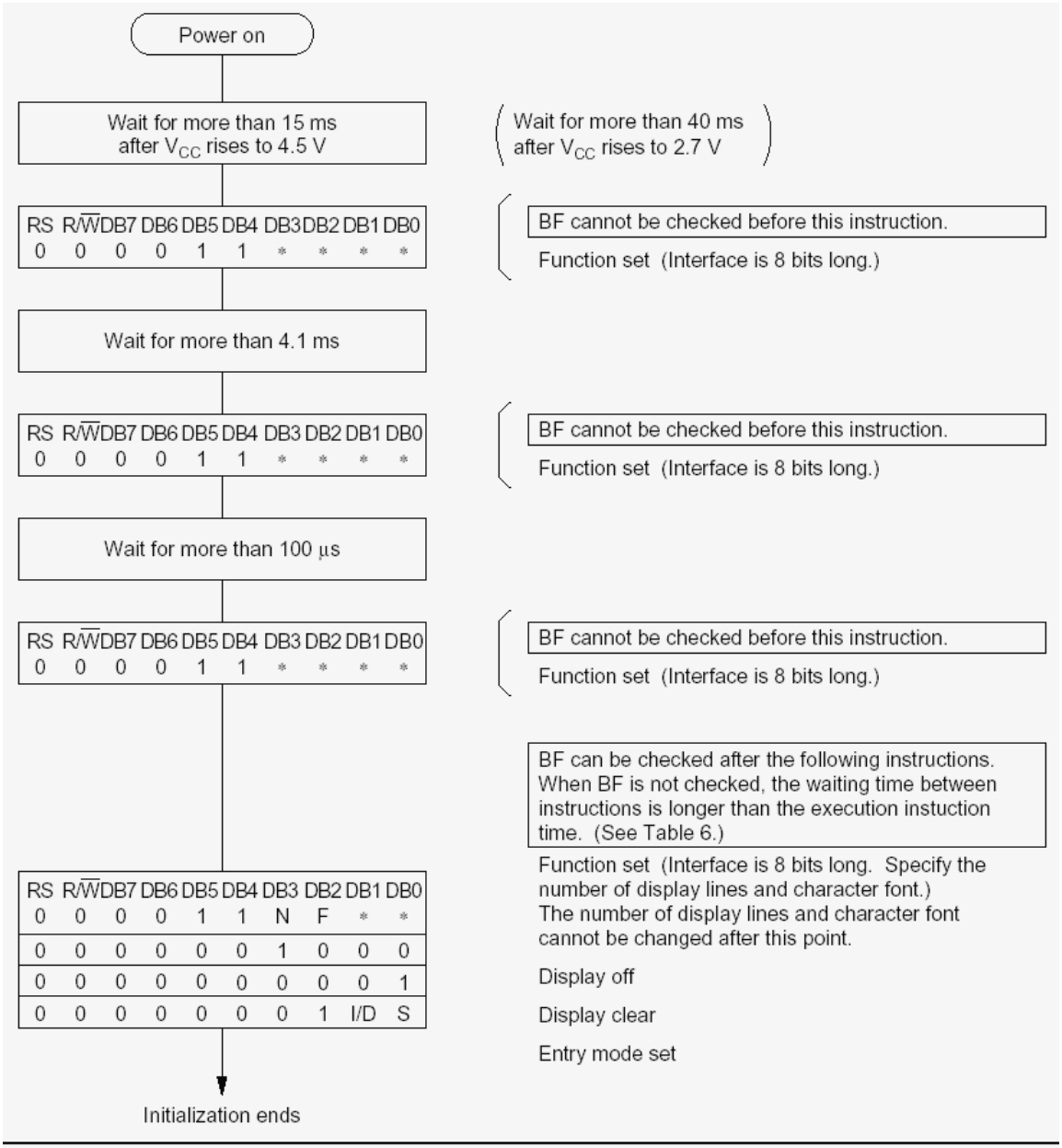


Figure 23 8-Bit Interface

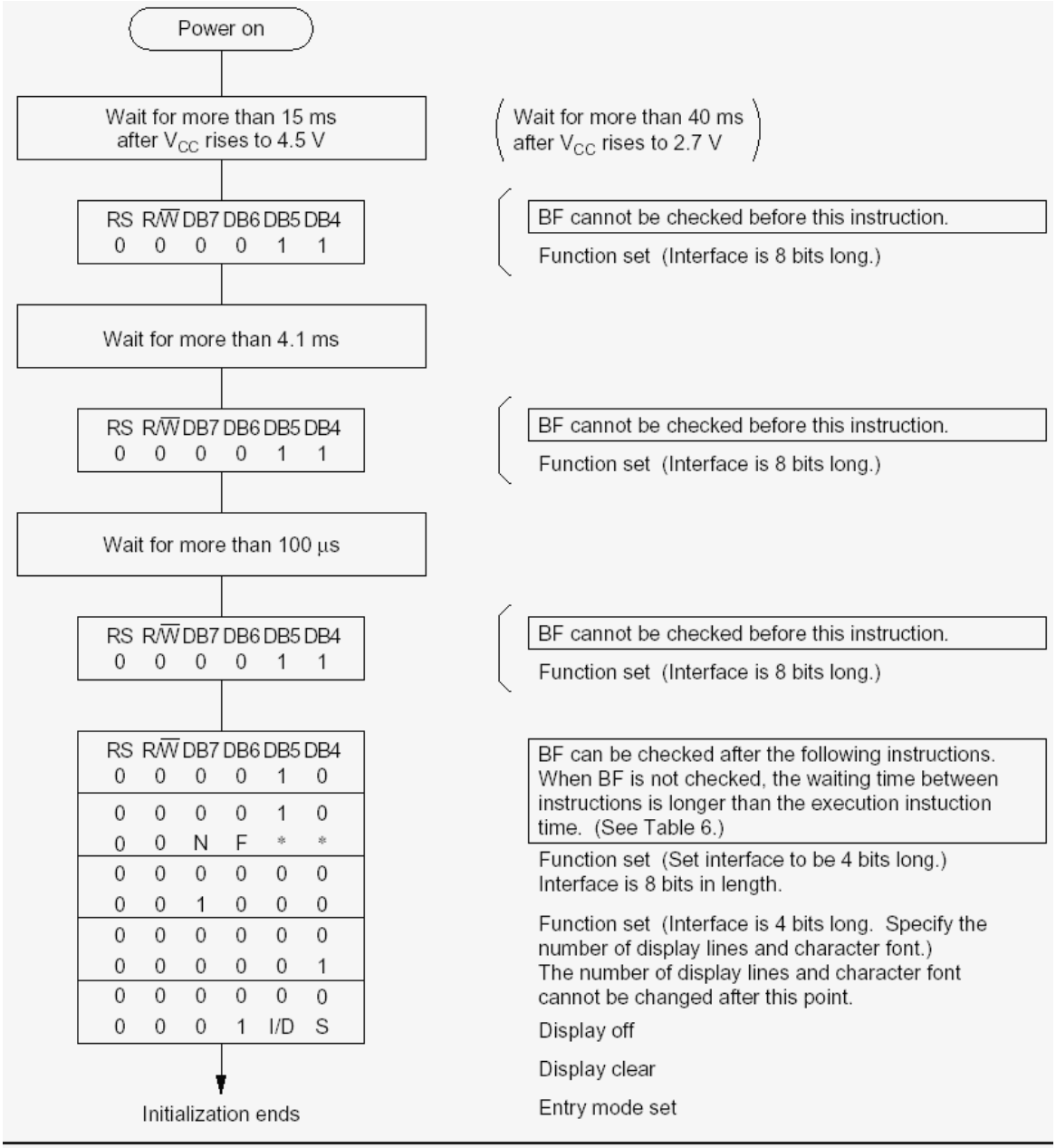


Figure 24 4-Bit Interface

Như đã đề cập ở trên, chế độ giao tiếp mặc định của LCD là 8bit (tự khởi tạo lúc mới bật điện lên). Và khi kết nối mạch theo giao thức 4bit, 4 bit thấp từ DB0-DB3 không được kết nối đến LCD, nên lệnh khởi tạo ban đầu (lệnh chọn giao thức giao tiếp – function set 0010****) phải giao tiếp theo chế độ 8 bit (chỉ gởi 4 bit cao một lần, bỏ qua 4 bit thấp). Từ lệnh sau trở đi, phải gởi/nhận lệnh theo 2 nibble.

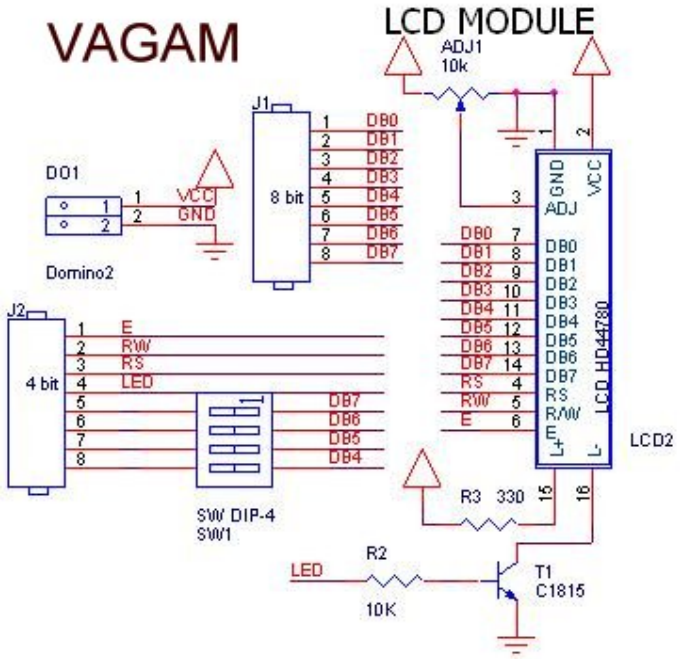
Lưu ý là sau khi thiết lập function set, bạn không thể thay đổi function set ngoại trừ thay đổi giao thức giao tiếp (4bit/8bit).

Phần 2 : KIT THÍ NGHIỆM LCD HD44780

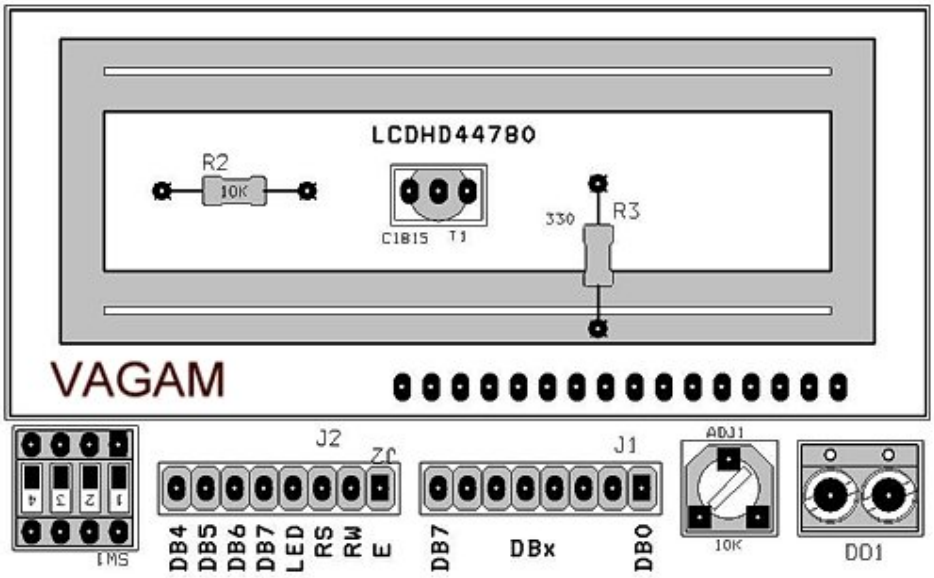
Giới thiệu: Trong phần trước, các bạn đã tìm hiểu các đặc tính của LCD, bây giờ chúng ta cùng thi công một KIT thí nghiệm LCD HD44780 để vận dụng kiến thức trên vào thực tế.

1> Sơ đồ mạch :

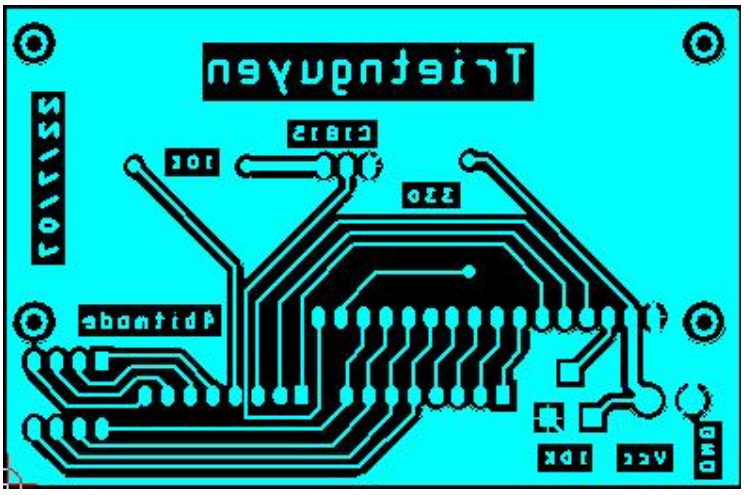
Mạch nguyên lý (schema):



Sơ đồ bố trí linh kiện:



Mạch in : (ko lấy cái này đem in nhá)



Hình ảnh hoàn tất :

2> Chương trình điều khiển:

Vì code hơi dài, xin download code ở diễn đàn VAGAM

Chúng ta sẽ thảo luận chi tiết cách lập trình cho LCD ở diễn đàn VAGAM

3> Lời kết:

Cheer!!!

Phần 3: LCD tutorial

Phần này là quá trình triển khai ứng dụng với LCD, chủ yếu là copy bài viết của tôi ở diễn đàn VAGAM

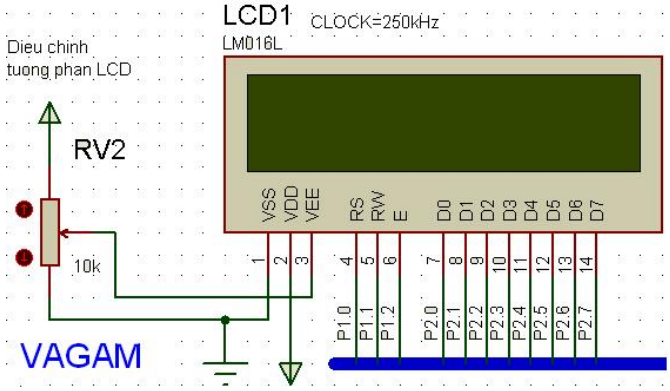
Bài 1: Khởi tạo LCD ở chế độ 8 bit

Khởi tạo LCD là hành động thiết lập các thông số làm việc ban đầu cho nó. Khởi tạo được thực hiện một lần vào đầu chương trình và bao gồm:
+ Thiết lập chế độ giao tiếp (function set): Bài này sẽ giúp bạn khởi tạo LCD ở chế độ 8 bit.
+ Thiết lập chế độ nhập (Entry mode set): Các thiết lập về kí tự
+ Thiết lập chế độ hiển thị (Display control): Kiểu con trỏ, kiểu kí tự ...
+ Xóa màn hình.

Tất cả công cụ cần cho bài học này là:
+ Phần mềm soạn thảo và dịch file ASM nào đó (khuyên dùng MIDE51)
+ Phần mềm mô phỏng: Proteus (Có khó khăn về phần mềm thì liên hệ các thành viên VAGAM)
Kiến thức:
Bạn không nhất thiết phải biết sử dụng proteus, nhưng bạn phải biết cơ bản về vi điều khiển cũng như lập trình cho vi điều khiển.
OK, here we go ...

Bài 1: (tt) Ra 1 lệnh cho LCD

Để khởi tạo, bạn phải "ra lệnh" cho LCD. Để LCD "hiểu" được "lệnh của bạn", bạn phải tuân theo "qui ước nói chuyện" của nó (giao thức), cả "âm lượng" (phần cứng) và "ngôn ngữ" (phần mềm). - Nhập gia tùy tục mà, bạn nhỉ!?
a. Giao thức phần cứng: *(xem chi tiết trong bài viết hoặc datasheet)*
+ LCD giao tiếp theo chuẩn logic TTL thông thường (5V cho logic 1 và 0V cho logic 0) cho nên có thể kết nối trực tiếp với 8051.
+ Chân dữ liệu (những chân dùng để "ra lệnh") của LCD từ chân 7 đến chân 14 (được nhà sản xuất đặt tên là DB0-DB7)
+ Chân tín hiệu là các chân 4(RS), 5(RW), 6(E)
Như vậy là bạn có thể thiết kế mạch ứng dụng cho LCD rồi đấy dễ quá phải ko nào! Dưới đây là một hình kết nối LCD trong file mô phỏng của mình:



- Hình 1: Kết nối phần cứng cho LCD -
b. Giao thức phần mềm:
- Để LCD hiểu được lệnh, chúng ta cần theo đúng giao thức của nhà sản xuất. Hình bên dưới là giao thức ghi một lệnh vào LCD:

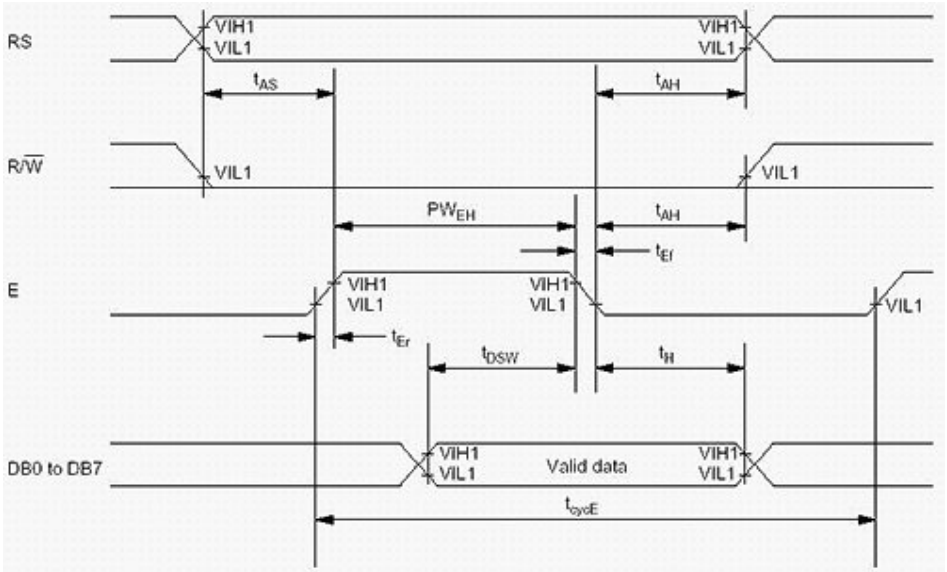
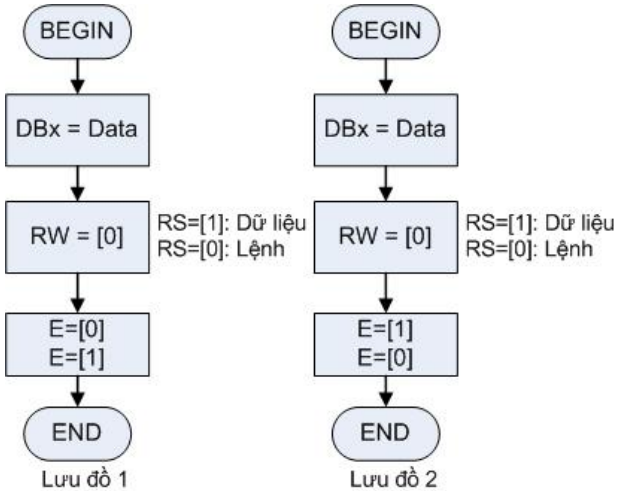


Figure 25 Write Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data set-up time	t_{DSW}	80	—	—	
Data hold time	t_H	10	—	—	

Tớ sẽ giải thích lưu đồ này một chút:
+ Đầu tiên là đường RS, ta thấy nó vẽ 2 đường trên (mức logic 1) và đưới (mức logic 0), tức là chúng ta có thể cấp cho chân này mức 0 hay mức 1 đều được. Mức 0 thì LCD hiểu dữ liệu ở chân DBx là dữ liệu hiển thị, mức 1 thì LCD hiểu dữ liệu ở chân DBx là là lệnh.
+ Chân RW phải được cấp mức logic 0
+ Chân E: Dữ liệu sẽ được nạp vào LCD ở thời điểm chân E từ cao xuống thấp (high-to-low). Nhưng dữ liệu cần đặt ở DBx trước thời điểm đó một khoảng tDSW=80ns và giữ ở đó tH=10ns (trong tài liệu tôi cố tình viết sai thành là low-to-high transition, mong rằng bạn nào quan tâm đọc sẽ phát hiện ra, nhưng hình như ai cũng "chấp nhận" hoặc là "ko ai đọc", nên chẳng ai có ý kiến, hic hic)



Với hai lưu đồ này, theo bạn cái nào tốt hơn? (câu hỏi thảo luận nhé ;)

Từ hai lưu đồ, ta có đoạn chương trình ra một lệnh cho LCD như sau:

```
MOV lcd_port,#00111000B ;  DL=1,  N=1,  F=1,(8bit  mode,  2  line,  5x8)
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E
```

và:

```
MOV lcd_port,#00111000B ;  N=1,  F=1
CLR RS ;RS  =  0
CLR RW ;RW  =  0
SETB E
CLR E
```

với `#00111000B`
là mã lệnh Function Set cho N=1, F=1
Đoạn video clip sau mô tả quá trình tôi thí nghiệm với LCD
Với đoạn **code 1**, ta thấy thời gian nhận lệnh ở LCD là 0.100007758s, còn đoạn **code 2** là 0.100008258s => Đoạn 1 ngắn hơn đoạn 2 : 0.5uS đây là 1 chu kì máy 0.5us (thạch anh 24MHz), chính là thời điểm LCD nhận được tín hiệu "CLR E".

Attachment(s)

LCD-protues.rar (17.8 KB, 1 lần tải)

LCD-codehoc1.rar (0.5 KB, 1 lần tải)

LCDdebugavi1.rar (716.8 KB, 1 lần tải)

Bài 1: (tt) Đọc dữ liệu từ LCD

Như đã đề cập trong bài viết, LCD và 8051 hoạt động không đồng bộ với nhau. 8051 "chạy" nhanh hơn LCD, do đó sau khi "ra một lệnh" cho LCD, nó phải đợi LCD "làm xong" lệnh đó mới được ra lệnh tiếp theo.
Xuất phát từ đây, ta có 2 cách:
+ **Đợi "mù"**: tức là sau khi ra một lệnh, 8051 đợi một khoảng thời gian cố định. Thời gian này phải dài hơn thời gian làm việc của LCD, và do nhà sản xuất qui định(từ 37uS đến 1,52ms - xem thêm trong bài viết)
+ **Đợi cờ báo bận BF từ LCD**: LCD có chân DB7 làm 2 nhiệm vụ, khi "ra lệnh" thì nó đóng vai trò chân dữ liệu, còn khi "đọc dữ liệu (*)" thì nó đóng vai trò cờ báo bận BF. Hoạt động của cờ BF xin xem thêm trong bài viết.
(*) : khi nói "ra lệnh" hoặc "đọc dữ liệu" thì ta ngầm hiểu đó là hành động của 8051. (LCD chỉ là đối tượng điều khiển)
Với cách 1, đoạn mã khởi tạo của chúng ta thế này:

```
MOV ms_num,#100 ;  delay  40ms  after  Vcc  rise  to  2.7V
call delayms

MOV lcd_port,#00111000B ;N=1,  F=1
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E

MOV ms_num,#1 ;  delay  1ms
call delayms
MOV lcd_port,#00001111B ;  display  on,  display  cursor  +  blink
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E
jmp $
```

Lưu ý, để cho gọn, tôi đã lượt bớt phần hàm delayms và các thứ linh tinh khác --> nếu bạn chỉ copy, paste y chang thì ko chạy nhé!!! Muốn thí nghiệm thì download code ở cuối bài viết này.
Sau đoạn chương trình trên, con trỏ trên LCD sẽ chớp, LCD khởi tạo thành công rồi đấy.
Ưu điểm theo cách này là đơn giản và dễ hiểu, tuy nhiên ko tối ưu cho lắm.(tớ thấy có rất nhiều bạn chọn cách này cho gọn :-D)
Với cách 2, ta phải đọc dữ liệu (đọc cờ BF thông qua chân DB7) từ LCD. Đoạn chương trình đọc dữ liệu cũng phải tuân theo giao thức của nhà sản xuất:

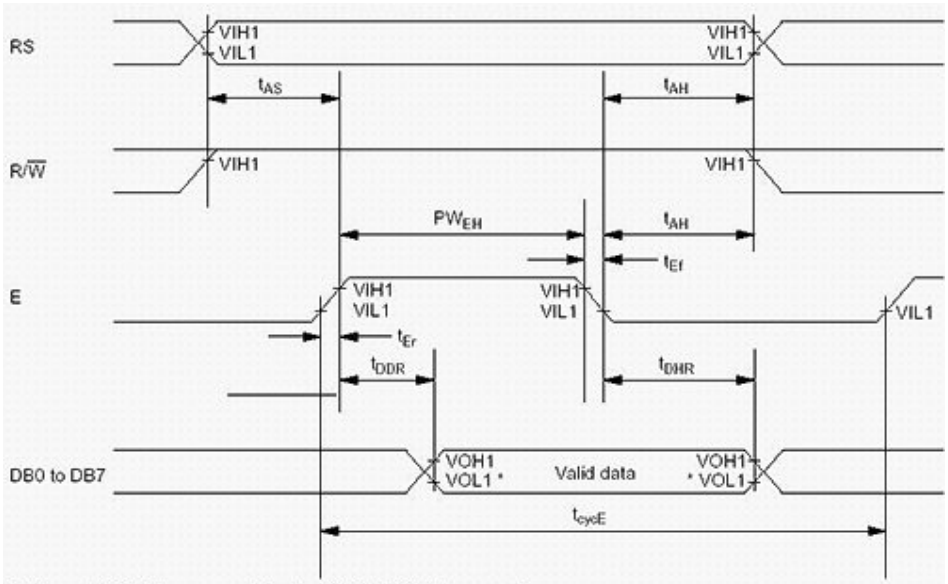
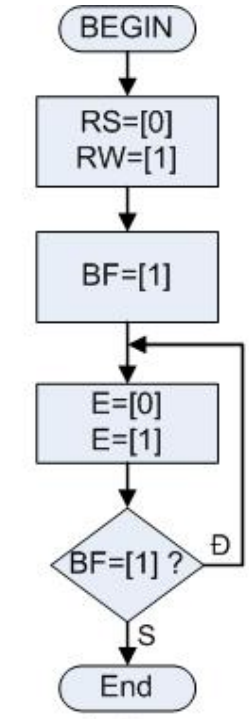


Figure 26 Read Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data delay time	t_{DDR}	—	—	160	
Data hold time	t_{DHR}	5	—	—	

Tôi cũng xin giải thích sơ qua giản đồ xung này, nhìn giống giản đồ write nhưng cách lý giải ko giống đâu bạn ạ:
+ Thứ nhất, chân RW phải treo lên mức cao.
+ Hai, chân E tích cực bằng cạnh lên chứ ko phải cạnh xuống! Sau tDDR=160ns khi có tín hiệu từ mức thấp lên mức cao ở chân E, dữ liệu được LCD xuất ra qua chân DBx, dữ liệu này được giữ tới khi nào chân E xuống thấp trở lại (kéo dài thêm tDHR=5ns)
+ Thời gian giữa hai lần có cạnh lên ở chân E tối thiểu là 0.5uS
+ Thời gian giữ mức cao ở chân E tối thiểu là 230ns
Từ giản đồ xung, ta có lưu đồ hàm kiểm tra cờ BF như sau:



Lưu đồ kiểm tra cờ BF

và đoạn chương trình bằng ASM như sau:

```
check_BF:
        CLR RS              ;RS=0
        SETB RW               ;RW=1
        SETB BF              ;latch  for  read
recheck:    CLR E
        SETB E
        JB BF,recheck
        ret
```

Kết hợp các thứ lại, ta có đoạn chương trình khởi tạo cho LCD theo cách 2 như sau:

```
MOV lcd_port,#00111000B ;N=1,  F=1
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E

call   check_BF
MOV lcd_port,#00001111B ;  display  on,  display  cursor  +  blink
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E
jmp $
```

Điểm khác biệt duy nhất giữa 2 chương trình là dòng

```
call    check_BF       ;  trong  chương  trình  2
```

và

```
MOV ms_num,#1         ;  trong  chương  trình  1
call delayms
```

Ta hãy xem sự tối ưu của chương trình 2 qua video mô phỏng sau:

Ở chương trình 2, thời gian giữa 2 lần nhận lệnh là 0.100052259s-0.100007758s=44.501us (trong datasheet cho thời gian thực thi lệnh này là 37us). => Tối ưu hơn nhiều so với 1ms của chương trình 1 bạn nhỉ!!!

Attachment(s)

LCD-chuongtrinh1.rar (0.5 KB, 0 lần tải)

LCD-chuongtrinh2.rar (0.6 KB, 0 lần tải)

LCDdebugavi2.rar (179.5 KB, 0 lần tải)

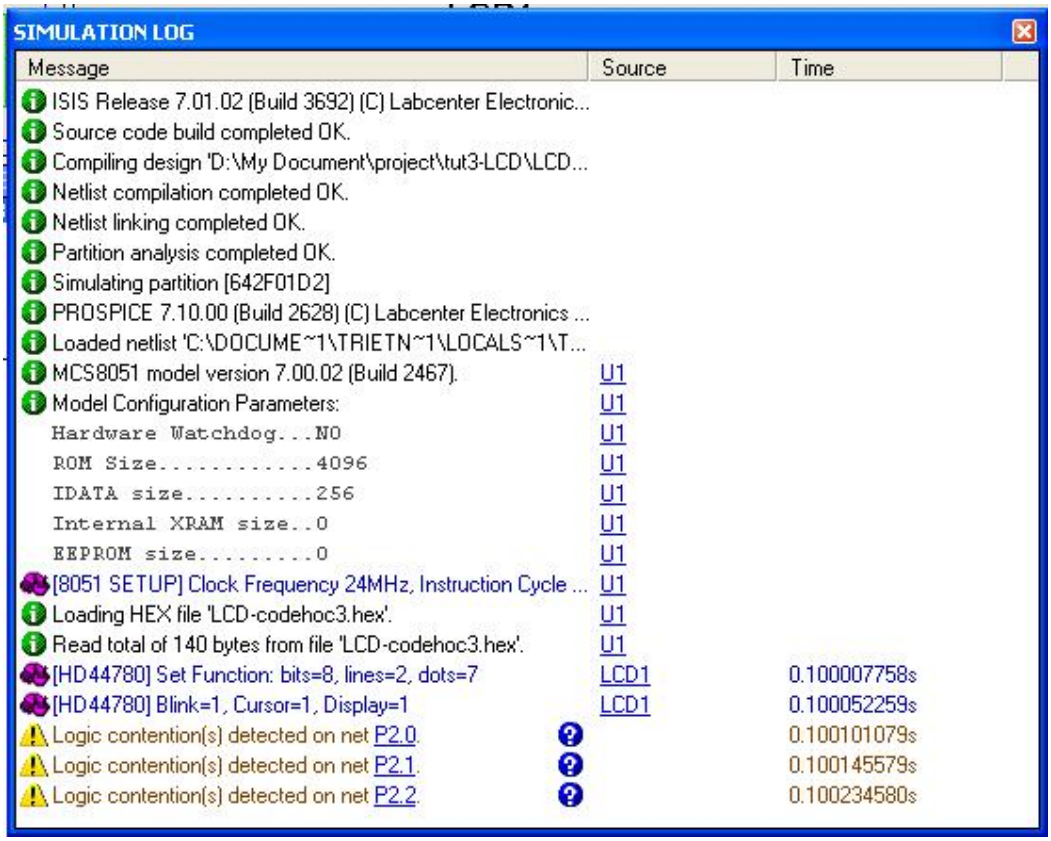
Bài 1: (tt) Xuất dòng chữ

Như vậy bạn đã "ra lệnh" và "đọc dữ liệu" từ LCD. Bây giờ hãy thử xuất một dòng chữ trên LCD xem sao?
Việc xuất chữ lên LCD thực chất cũng là "ra lệnh", khác ở chổ ta thiết lập chân RS=[1] để DBx nối với vùng ram dữ liệu DDR

```
RS bit P1.0
RW bit P1.1
E bit P1.2
BF bit P2.7
ms_num equ 2fh
lcd_port equ P2
lcd_mask equ 0FFh
ORG 000h
jmp main
ORG 030h ;  main  program  begin  here
main:
MOV ms_num,#100 ;  delay  40ms  after  Vcc  rise  to  2.7V
call delayms
MOV lcd_port,#00111000B ;N=1,  F=1
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E
call   check_BF
MOV lcd_port,#00001111B ;  display  on,  display  cursor  +  blink
CLR RS ;RS  =  0
CLR RW ;RW  =  0
CLR E
SETB E
call   check_BF
MOV lcd_port,#'V'
SETB RS
CLR RW
CLR E
SETB E
call   check_BF
MOV lcd_port,#'A'
SETB RS
CLR RW
CLR E
SETB E
call   check_BF
MOV lcd_port,#'G'
SETB RS
CLR RW
CLR E
SETB E
call   check_BF
MOV lcd_port,#'A'
SETB RS
CLR RW
CLR E
SETB E
call   check_BF
MOV lcd_port,#'M'
SETB RS
CLR RW
CLR E
SETB E
jmp $
check_BF:
CLR RS ;RS=0
SETB RW ;RW=1
SETB BF ;  latch  for  read
recheck:CLR E
SETB E
JB BF,recheck
ret
```

```
delayms: ; Xtal = 24MHZ, Tm=0.5us
PUSH 7
PUSH 6
MOV R7,#51
de_rep: MOV R6,#18
DJNZ R6,$
DJNZ R7,de_rep
POP 6
POP 7
DJNZ ms_num,delayms
ret
END
```

Khi debug trong proteus, mặc dù LCD vẫn hiện chữ "VAGAM", nhưng nó báo lỗi to tướng:




```
Logic contention(s) detected on net P2.0,P2.1,P2.2 !!!
```

Tạm dịch là: "Có nối tắt Vcc với GND ở các chân logic!"
Sở dĩ tôi muốn nói đến cái này vì thấy nhiều bạn viết mà ko chú ý làm giảm tuổi thọ LCD và 8051 mà ko rõ nguyên nhân (cứ đổ thừa "LCD hàng china")
Sai là ở chỗ:
+ Sau câu lệnh "call check_BF" là lệnh "MOV lcd_port,#data": trong khi LCD xuất dữ liệu ra sau lệnh check_BF thì ta lại cấp dữ liệu mới vào DBx => xung khắc
+ Trong hàm check_BF, ta chỉ chốt 1 chân BF để đọc, các chân còn lại không kiểm soát => Nguy cơ xung đột khi LCD xuất dữ liệu ra DBx
Từ nhược điểm trên, chúng ta cải tiến lại chương trình:
+ Chuyển câu lệnh "MOV lcd_port,#data" ra sau hai câu lệnh "SETB RS" và "CLR RW" để LCD ở trạng thái "đọc lệnh từ MPU" và treo DBx lên tổng trở cao.
+ Đổi "SETB BF" thành "MOV lcd_port,#0FFh"
Như vậy, chương trình thành (download code bên dưới)
Chúng ta cũng kết thúc bài 1 ở đây! (Phù, khỏe quá)

Attachment(s)
LCD-codehoc4.rar (0.6 KB, 0 lần tải)

Bài 2: Tối ưu code

Từ chương trình cuối, chúng ta cải tiến một chút để cho code ngắn gọn lại:

```
 dta: call check_BF
SETB RS ;RS  =  1,  DR  selected
CLR RW ;RW  =  0,  write  to  LCD
exec: MOV lcd_port,A
SETB E
CLR E
ret
```

Bây giờ hành động ghi dữ liệu đơn giản là lời gọi hàm dta:

```
MOV A,#dữ_liệu
Call dta
```

Với hành động "xuất lệnh", ta biến đổi hàm dta một tí thành hàm cmd:

```
 cmd: call check_BF
CLR RS ;RS  =  0,  IR  selected
CLR RW ;RW  =  0,  write  to  LCD
exec: MOV lcd_port,A
SETB E
CLR E
ret
```

Lúc này, hành động ra một lệnh cho LCD như sau:

```
 MOV A,#mã_lệnh
Call cmd
```

Do giống nhau phần exec, ta có thể viết lại:

```
 cmd: call check_BF
CLR RS ;RS  =  0,  IR  selected
CLR RW ;RW  =  0,  write  to  LCD
jmp exec
dta: call check_BF
SETB RS ;RS  =  1,  DR  selected
CLR RW ;RW  =  0,  write  to  LCD
exec: MOV lcd_port,A
SETB E
CLR E
Ret
```

Cuối cùng, sau khi cải tiến lần 1, ta có đoạn chương trình hiện chữ VAGAM như sau (*):

```
 MOV A,#mode8bit
call cmd

MOV A,#cursor_on
call cmd

MOV A,#'V'
call dta

MOV A,#'A'
call dta

MOV A,#'G'
call dta

MOV A,#'A'
call dta

MOV A,#'M'
call dta

jmp $

cmd: call check_BF
CLR RS ;RS  =  0,  IR  selected
```

```
CLR RW ;RW = 0, write to LCD
jmp exec
dta: call check_BF
SETB RS ;RS = 1, DR selected
CLR RW ;RW = 0, write to LCD
exec: MOV lcd_port,A
SETB E
CLR E
return: ret
```

* Các chi tiết phụ được bỏ bớt, chương trình đầy đủ có thể tải ở đây

codetoiuu1.asm

Việc xuất dòng chữ VAGAM xem ra còn có tính lặp lại. Ta tối ưu một lần nữa bằng cách sử dụng con trỏ DPTR và kiểu định địa chỉ gián tiếp để tạo vòng lặp

```
 putcmd: MOV A,#00h
MOVC A,@A+dptr
CJNE A,#endcmd,next_pc
jmp return
next_pc:
CALL cmd
inc   dptr
jmp putcmd
```

Ta xuất ra một chuỗi lệnh bằng cách nạp địa chỉ của dữ liệu:

```
 MOV dptr,#initcmd ; Lần lượt ra các lệnh trong dãy initcmd
call putcmd
```

Với nhãn initcmd nằm ở cuối chương trình:

```
 initcmd: DB mode8bit,cursor_on,clr_disp,endcmd
```

Tương tự như vậy, ta làm hàm xuất chuỗi dữ liệu:

```
 putstr: MOV A,#00h
MOVC A,@A+dptr
CJNE A,#endstr,next_ps
jmp return
next_ps:
CALL dta
PUSH ms_num
call delayms
POP ms_num
inc   dptr
jmp putstr
```

Và việc xuất chuỗi dữ liệu ra đơn giản là lệnh nạp nhãn dữ liệu cho dptr và lời gọi hàm putstr:

```
 MOV ms_num,#200 ; Mỗi kí tự xuất ra delay 200ms
MOV dptr,#messa1 ; Lần lượt xuất từng kí tự trong dãy messa1
call putstr
```

Với nhãn messa1 nằm ở cuối chương trình

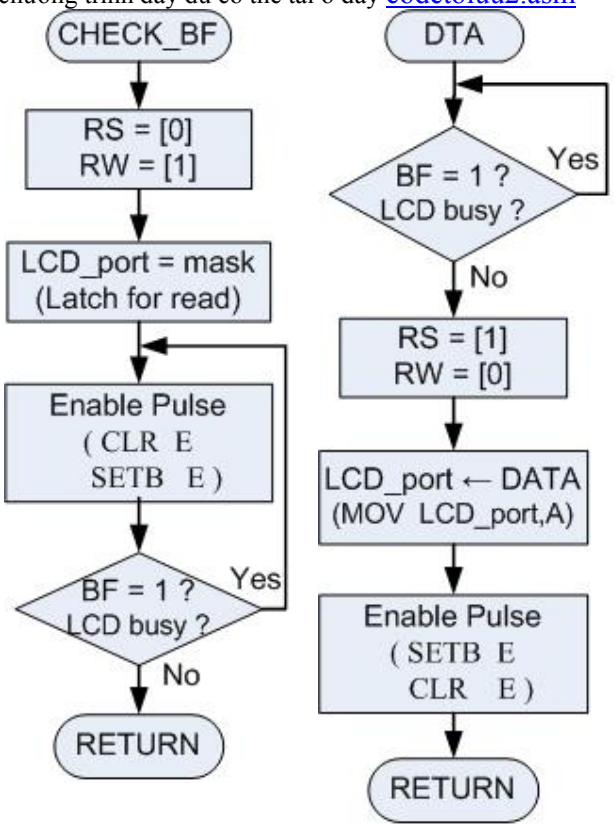
```
 messa1:   DB 'VAGAM',endstr
```

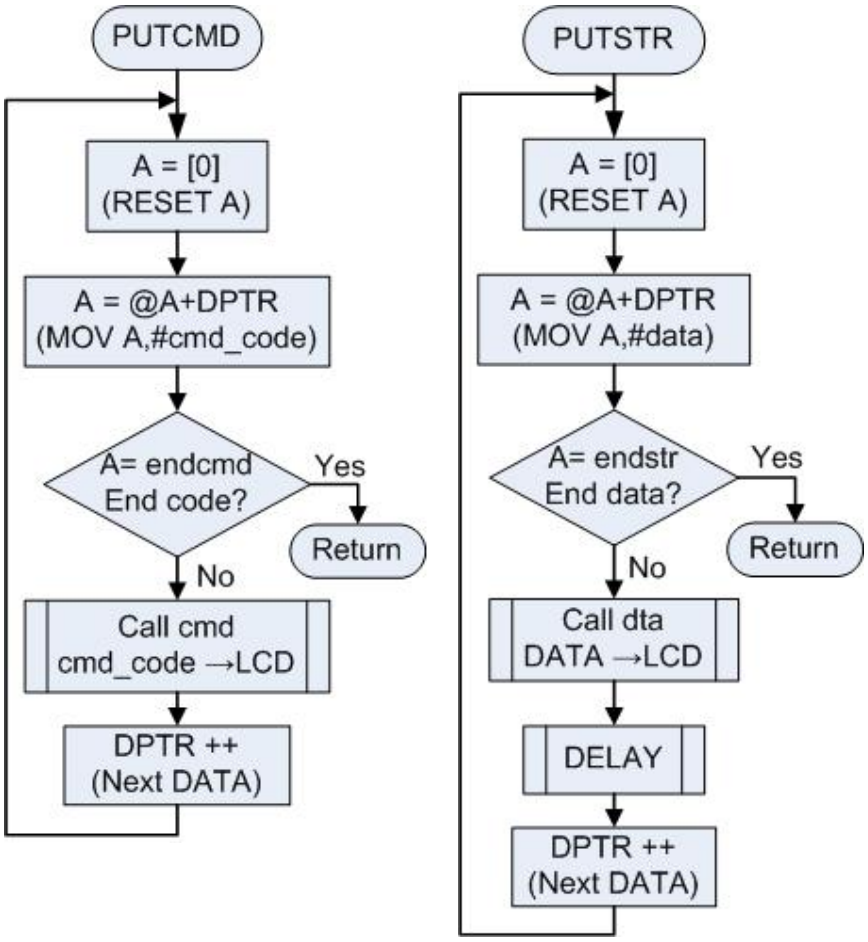
Kết hợp lại ta có đoạn chương trình sau:

```
  ...
MOV dptr,#initcmd ;  initial
call putcmd

MOV ms_num,#200
MOV dptr,#messa1 ;
call putstr

jmp $

putstr: MOV A,#00h
MOVC A,@A+dptr
CJNE A,#endstr,next_ps
jmp return
next_ps:
CALL dta
```

```
PUSH ms_num
call delayms
POP ms_num
inc   dptr
jmp putstr

putcmd:
MOV A,#00h
MOVC A,@A+dptr
CJNE A,#endcmd,next_pc
jmp return
next_pc:
CALL cmd
inc   dptr
jmp putcmd

cmd: call check_BF
CLR RS ;RS  =  0,  IR  selected
CLR RW ;RW  =  0,  write  to  LCD
jmp exec
dta: call check_BF
SETB RS ;RS  =  1,  DR  selected
CLR RW ;RW  =  0,  write  to  LCD
exec: MOV lcd_port,A
SETB E
CLR E
Ret
...
initcmd: DB mode8bit,cursor_on,clr_disp,endcmd
messa1:   DB 'VAGAM',endstr
END
```

* Các chi tiết phụ được bỏ bớt, chương trình đầy đủ có thể tải ở đây [codetoiuu2.asm]

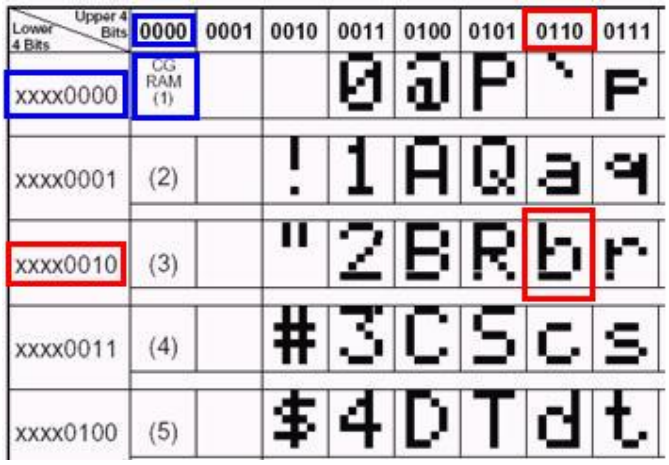




Việc tối ưu code có thể còn ngắn và tối ưu hơn nữa, cũng như thêm các hàm tạo hiệu ứng động cho kí tự. Tuy nhiên do tính chất tutorial là giới thiệu, chúng ta kết thúc bài tối ưu code ở đây.

Bài 3: Lập trình hiển thị kí tự đồ họa

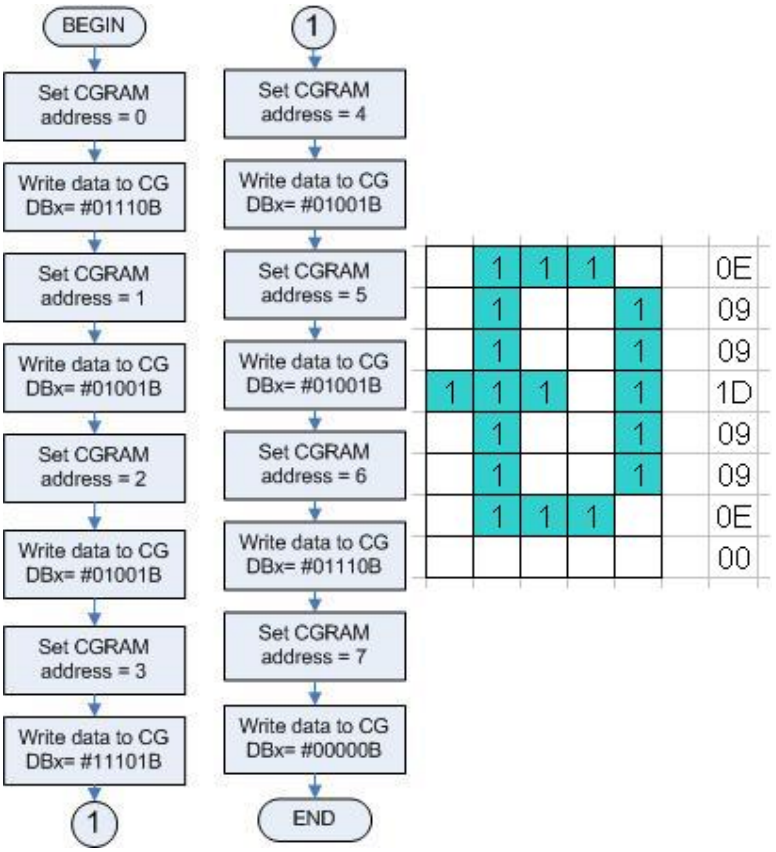
Ở các bài trước chúng ta biết cách xuất ra một kí tự lên địa chỉ hiển thị trong LCD. Ví dụ để hiển thị chữ 'b' lên LCD tại địa chỉ hiện thời ta ra một lệnh xuất dữ liệu (hàm dta) với mã lệnh là "01100010" (đây chính là vị trí chữ 'b' trong bảng mã kí tự LCD ở trang 6 của bài viết). Và do trùng với bảng mã ASCII, nên thay vì viết #01100010B ta có thể viết #'b' trong ASM.



Bây giờ nếu ta xuất một mã dữ liệu "#00000000B" thì LCD hiển thị cái gì? Câu trả lời nằm ở vùng RAM đồ họa CGRAM (xin xem thêm về CGRAM tại trang 7 của tài liệu). Từ bảng 5 bên dưới và lệnh Write data to CG or DDRAM, ta có lưu đồ ghi một kí tự đồ họa "Đ" vào CGRAM tại địa chỉ #00000000B.

For 5 × 8 dot character patterns

Character Codes (DDRAM data)	CGRAM Address	Character Patterns (CGRAM data)	
7 6 5 4 3 2 1 0 High Low	5 4 3 2 1 0 High Low	7 6 5 4 3 2 1 0 High Low	
0 0 0 0 * 0 0 0	0 0 0	0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0
			Character pattern (1) Cursor position
0 0 0 0 * 0 0 1	0 0 1	0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
			Character pattern (2) Cursor position
		0 0 0 0 0 1	
0 0 0 0 * 1 1 1	1 1 1	1 0 0 1 0 1 1 1 0 1 1 1	



Từ lưu đồ, ta cải tiến hàm putstr đã có để được hàm put_cg:

```
put_cg:  MOV      R0,#40h
putcg:  MOV      A,R0
CALL      cmd
INC      R0
MOV      A,#00h
MOVC      A,@A+dptr
CJNE      A,#endcmd,next_cg
JMP      return
next_cg:
CALL      dta
INC      dptr
JMP      putcg
```

Và sau đây là chương trình hiện chữ "Cơ Điện Tử" trên màn hình LCD.

Chúng ta kết thúc bài 3 ở đây, cũng kết thúc luôn phần lập trình LCD theo giao thức 8 bit bằng ASM. Nếu nắm vững kiến thức này, các bạn có thể lập trình cho LCD bằng ngôn ngữ gì cũng được (C chẳng hạn) và bằng vi điều khiển gì cũng được nốt ...
Các bài tiếp theo chúng ta sẽ lập trình LCD theo chế độ 4 bit, đây là chế độ được ưa chuộng trong thực tế ...
Attachment(s)
LCDCGRAM-code.rar (1.2 KB, 11 lần tải)
LCDCGRAM-code.rar (1.2 KB, 11 lần tải)

Bài 4: Giao tiếp LCD HD44780 theo chế độ 4 bit

Ở bài học này chúng ta sẽ sử dụng LCD theo chế độ 4 bit, tức là chỉ dùng 4 chân DB4-DB7 để truyền dữ liệu.

Về giao thức giao tiếp, chế độ 4 bit không được đề cập chi tiết trong bài viết vì nó cơ bản giống như giao thức 8 bit, chỉ khác ở một điểm là ta gởi/nhận mỗi lần 4 bit, với 4 bit cao gởi/nhận trước. Hình bên dưới minh họa cho giao thức gởi một lệnh đến LCD theo chế độ 4 bit. Giữa hai nibble (4-bit) ta cần thêm một xung enable.

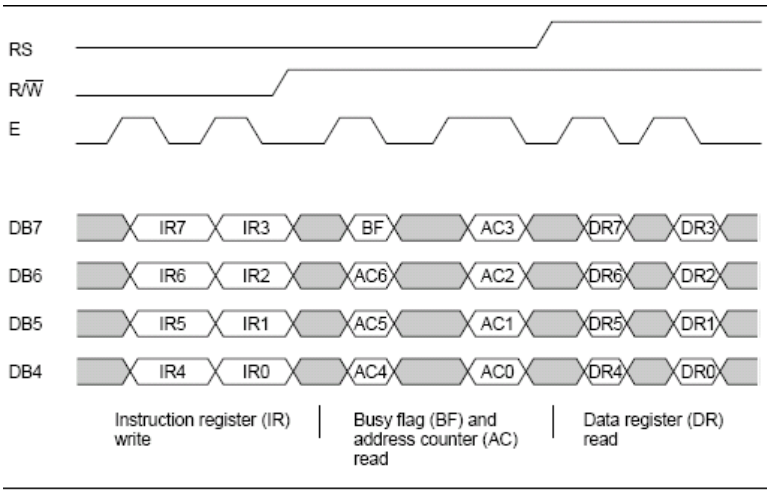
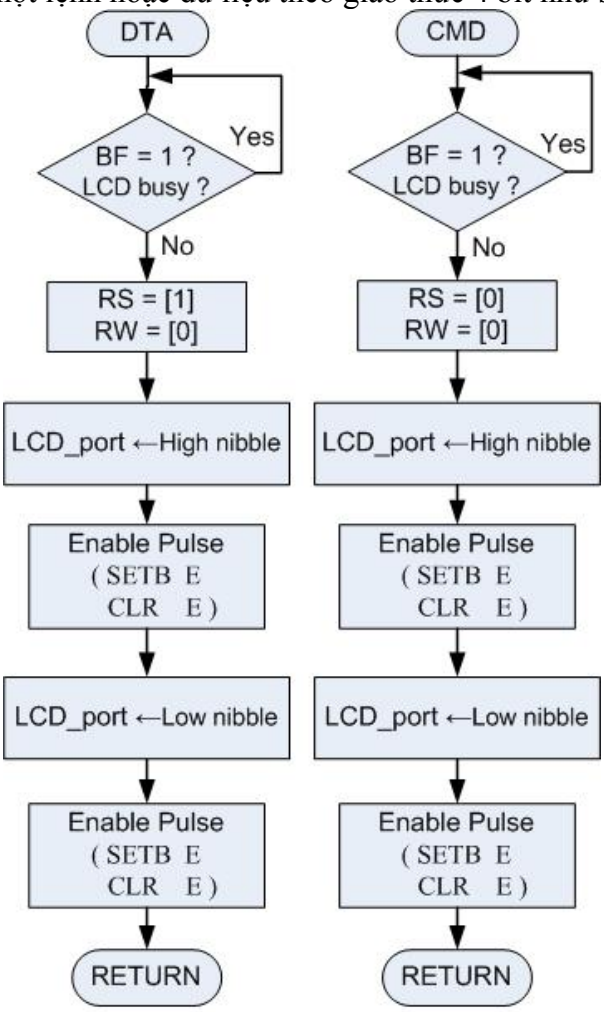


Figure 9 4-Bit Transfer Example

Từ đây là có lưu đồ xuất một lệnh hoặc dữ liệu theo giao thức 4 bit như sau:

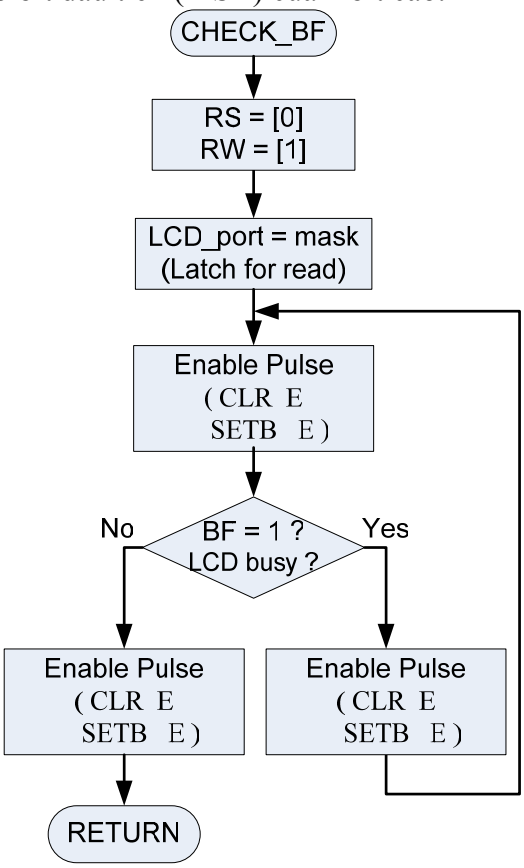


Với một chút kinh nghiệm lập trình ASM, ta có thể viết chương trình cho lưu đồ trên với một chút cải tiến (bằng cách dùng chương trình con). Một chương trình mẫu được cho bên dưới:

```
dta:    call    check_BF        ; Kiểm tra cờ BF
        CLR     RW              ; Thiết lập chế độ nhận dữ liệu
        SETB    RS              ; //
        jmp     exec
cmd:    call    check_BF
```

```
        CLR     RW              ; Thiết lập chế độ nhận lệnh
        CLR     RS              ; //
exec:   PUSH    ACC
        call    nibble          ; 4 bit cao gởi đi trước
        POP     ACC
        SWAP    A               ; Đổi 4 bit thấp lên vị trí 4 bit cao
        call    nibble          ; 4 bit thấp gởi sau
return: ret
nibble:                         ; chương trình con gởi đi 4 bit
        ORL     lcd_port,#lcd_mask ; Che 4 chân không dùng
        ORL     A,#data_mask    ; lcd_mask=0F0h, data_mask=0Fh
        ANL     lcd_port,A      ; 4 chân port không dùng không đổi
        SETB    E               ; Xung enable
        CLR   E
        ret
```

Việc đọc cờ bận BF cũng tương tự như ở chế độ 8 bit, 4 bit cao đọc trước, 4 bit thấp đọc sau (bỏ qua 4 bit sau). Cờ BF nằm ở bit đầu tiên (MSB) của 4 bit cao.



Đoạn chương trình mẫu:

```
check_BF:
        CLR     RS              ; Thiết lập chế độ xuất dữ liệu
        SETB    RW              ;
        ORL     lcd_port,#lcd_mask ; Chốt để đọc vào
        jmp     next_BF
recheckBF:
        clr     E               ; Xung enable
        setb    E
next_BF:
```

```
clr     E
setb    E
JB      BF,recheckBF
clr     E
setb    E
ret
```

Đó là tất cả những gì cơ bản về chế độ 4 bit (chỉ có thế :D). Nhưng xin lưu ý các bạn một điều về khởi tạo LCD: (trích lại trong bài viết trang 15)

*"Như đã đề cập ở trên, chế độ giao tiếp mặc định của LCD là 8bit (tự khởi tạo lúc mới bật điện lên). Và khi kết nối mạch theo giao thức 4bit, 4 bit thấp từ DB0-DB3 không được kết nối đến LCD, nên lệnh khởi tạo ban đầu (lệnh chọn giao thức giao tiếp – function set 0010****) phải giao tiếp theo chế độ 8 bit (chỉ gởi 4 bit cao một lần, bỏ qua 4 bit thấp). Từ lệnh sau trở đi, phải gởi/nhận lệnh theo 2 nibble."*

Quá trình tiến hóa code đã được nói tới ở các bài trước, bài này chỉ nêu ra đoạn code cuối cùng, đoạn code hiện chữ "Cơ Điên Tử" trên LCD (có thể down load ở diễn đàn VAGAM)