제 11 장 Liquid Crystal Display(LCD) 장치

11.1 개요

마이크로컨트롤러를 사용하는 많은 기기에서 여러 가지 상태를 표시하기 위한 장치로서 LCD 표시장치를 사용하고 있다. LCD 모듈은 디스플레이 드라이버와 디스플레이를 합한 것을 말한다. 마이크로컨트롤러는 드라이버에 데이터를 전달함으로써 LCD 화면에 정보를 표시할 수 있다.

LCD 표시장치는 크게 문자형과 그래픽형 두 가지로 분류된다. 문자형은 드라이버 내에 글꼴을 가지고 있어 영문자나 숫자를 쉽게 표시할 수 있다. 그래픽 형은 문자형에 폰트가 없는 한글 또는 그래픽을 표시할 수 있으나 문자형보다는 사용하기 어렵다. 실습에서는 제어가 쉬운 문자형 LCD를 사용한다.

현재 시판되는 문자형 LCD 표시장치의 드라이버로서 Hitachi사의 HD44780가 가장 많이 쓰인다. 여기서는 HD44780의 특성을 개략적으로 설명하고 LCD를 제어하는 프로그램을 작성하도록 한다. LCD 사용 프로그램은 C-언어의 작성기법인 모듈화프로그램 방법으로 작성하여 이 후의 실습에서도 LCD 프로그램모듈을 계속적으로 사용할 수 있도록 하였다.

LCD 드라이버에 대한 자세한 사항은 HD44780의 Data Sheet 참고하기 바란다.

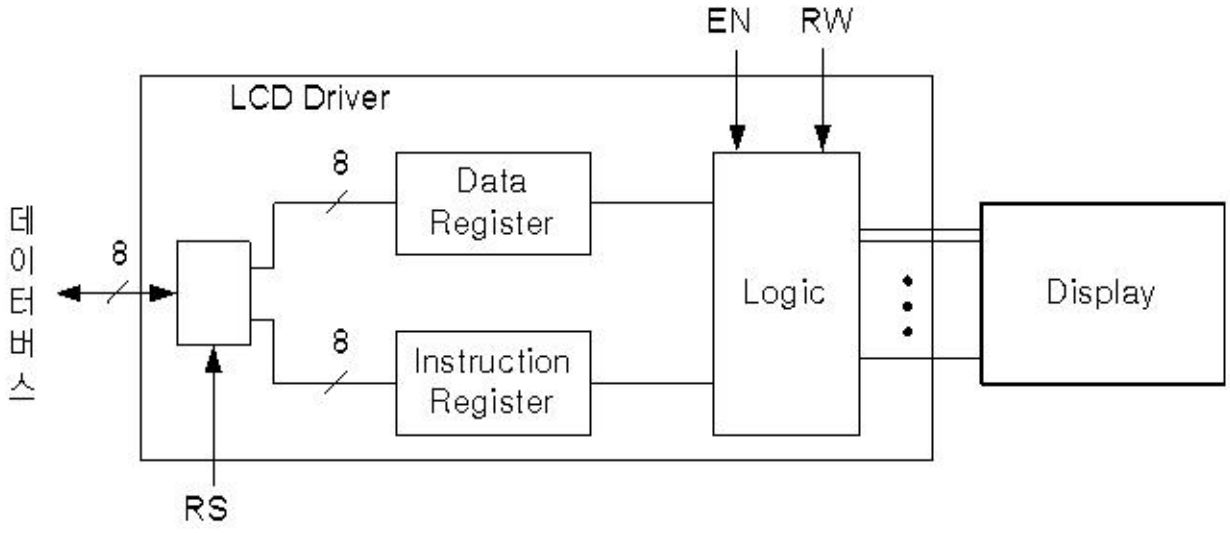


그림 11.1 LCD 드라이버의 개념적 구조

11.2 HD44780 LCD 드라이버

11.2.1 내부구조

그림 11.1은 LCD 드라이버의 개념적 구조를 나타낸다.

(1) 레지스터

LCD 모듈은 2개의 8비트 레지스터, Instruction Register(IR)와 Data Register(DR)를 가지고 있다.

- Instruction Register(IR) : 명령 레지스터
  화면의 지움, 화면쉬프트 등 LCD 동작 명령을 내리거나 드라이버 내 DDRAM(Display Data RAM)과 CGRAM(Character Generator RAM) 주소를 설정하는 데 사용한다. 쓰기만 가능하다.
- Data Register(DR) : 데이터 레지스터
  화면에 표시할 데이터, 새로운 글꼴에 대한 데이터를 임시로 저장하는 레지스터이다. DR에 데이터를 쓰면 IR로 지정된 주소의 DDRAM 또는 CGRAM에 데이터가 전달된다. DR의 데이터를 읽으면 지정된 주소의 DDRAM 또는 CGRAM에 데이터가 CPU에 전달된다.

(2) Busy Flag(BF)

BF는 LCD가 다음 명령을 받을 수 있는지 상태를 나타낸다. BF=1이면 내부 동작중인 상태로 명령을 받을 수 없다.

(3) Address Counter(AC) : 어드레스 카운터

어드레스 카운터는 DDRAM 어드레스 카운터와 CGRAM 어드레스 카운터가 있다. 각각 데이터 쓰기/읽기를 할 DDRAM과 CGRAM의 주소를 나타낸다. IR레지스터에 주소를 써넣음과 동시에 해당 어드레스 카운터에 값이 세트된다. 각 RAM에 데이터를 써넣으면 어드레스 카운터는 자동적으로 1 증가(모드에 따라 1 감소)한다.

(4) 제어신호

그림 7.1과 같이 세 개의 제어신호가 있다.

- EN : 인에이블신호이다.
- RS : DR 또는 IR 레지스터를 설정하는 신호이다.
- R/W : 읽기/쓰기 동작을 나타낸다.

신호 RS와 R/W의 조합에 따라 LCD는 다음 동작을 한다.

표 11.1 핀 설정에 따른 LCD 동작

RS	R/W	동작
0	0	IR을 선택하여 명령쓰기(화면지움 등)
0	1	BF 읽기 / 어드레스카운터 읽기
1	0	DR을 선택하여 DDRAM 또는 CGRAM에 데이터 쓰기
1	1	DR을 선택하여 DDRAM 또는 CGRAM에서 데이터 읽기

(5) Display Data RAM(DDRAM)

DDRAM은 화면에 표시할 8비트 문자를 저장하는 곳으로 80x8 비트의 용량을 갖는다. 화면에 표시되지 않는 영역의 RAM은 일반적인 메모리로 사용될 수 있다. 다음은 2라인 16문자를 표시할 때 DDRAM의 주소와 화면위치의 관계를 보여준다. DDRAM의 01번지에 글자 'A'를 쓰면 화면 0행 1열에 'A'가 나타난다.(화면을 쉬프트하지 않았을 때)

화면 열	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
DDRAM	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	화면 0-행
주소	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	화면 1-행

* 프로그램의 편의성을 위하여 화면의 열과 행을 0부터 시작하였음

표 11.2 DDRAM 주소와 화면위치

(6) Character Generator ROM(CGROM)

CGROM은 표 11.3의 8비트 문자코드에 따라 5x8 도트 문자 또는 5x10도트문자를 발생시킨다. 표에서 (1)~(7)로 표시된 부분은 Character Generator RAM(CGRAM) 부분이며 사용자가 글꼴을 정의할 수 있다.

표 11.3 문자코드와 글꼴 대응표

Upper 4 Bits / Lower 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	q	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	ɾ	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	-'	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	※	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	

문자 코드 0b00100001('!')부터 0b01111111('←')까지는 ASCII코드와 일치한다. C-언어에서 문자는 ASCII코드로 표현하므로 문자를 화면에 표시하려면 C-언어에서 문자로 표현된 데이터를 LCD에 출력하면 된다.

(예) C-언어에서 'A'는 표 11.3과 같이 0b01000001(0x41)과 표시되므로 다음 두 문장은 동일하다.

```
x = 'A';   <==> x = 0x41;
```

(6) Character Generator RAM(CGRAM) : LCD에는 몇 개의 글꼴을 사용자가 만들 수 있다. CGRAM에 사용자가 정의한 글꼴을 저장한다. 그림 11.2는 문자코드, CGRAM 주소, 5x7도트사용자 글꼴과의 관계를 보여준다.

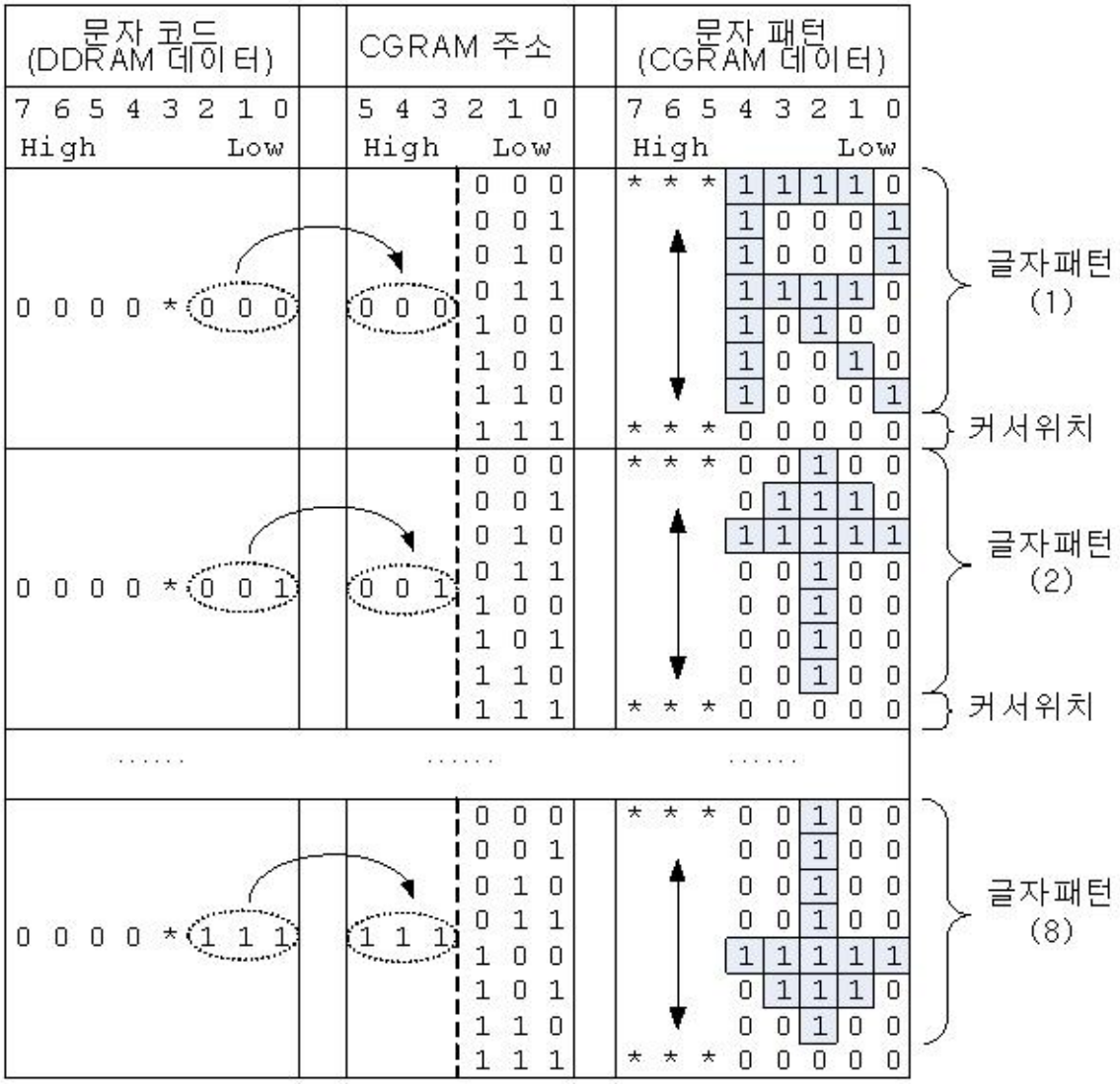


그림 11.2 문자코드, CGRAM 주소 그리고 글꼴과의 관계

11.2.2 LCD 명령

표 11.4 LCD 명령

명령	코드										실행 시간
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
화면 지움	0	0	0	0	0	0	0	0	0	1	1.52ms
커서 홈	0	0	0	0	0	0	0	0	1	*	1.52ms
엔트리모드 세트	0	0	0	0	0	0	0	1	I/D	S	37μs
화면 ON/OFF 제어	0	0	0	0	0	0	1	D	C	B	37μs
커서/화면 이동	0	0	0	0	0	1	S/C	R/L	*	*	37μs
펑션 세트	0	0	0	0	1	DL	N	F	*	*	37μs
CGRAM 주소 설정	0	0	0	1	CGRAM 주소						37μs
DDRAM 주소 설정	0	0	1	DDRAM 주소							37μs
비지 플래그와 어드레스 카운터 읽기	0	1	BF	CGRAM/DDRAM 주소							0μs
CGRAM 또는 DDRAM에 데이터 쓰기	1	0	출력 데이터								37μs
CGRAM 또는 DDRAM의 데이터 읽기	1	1	입력 데이터								37μs

비트설정

I/D: 0 - 커서 위치 감소, 1 - 커서 위치 증가
S : 0 - 화면 이동 없음, 1 - 화면 전체 이동
D : 0 - 화면 끔, 1 - 화면 켬
C : 0 - 커서 끔, 1 - 커서 켬
B : 0 - 커서 블링크 끔, 1 - 커서 블링크 켬
S/C: 0 - 커서 이동, 1 - 화면 이동
R/L: 0 - 왼쪽으로 이동, 1 - 오른쪽으로 이동
DL : 0 - 4비트 인터페이스, 1 - 8비트 인터페이스
N : 0 - 1-라인 디스플레이, 1 - 2-라인 디스플레이
F : 0 - 5x7 도트 문자, 1 - 5x10 도트 문자
BF : 0 - 명령수행 가능, 1 - 명령수행 불가능

11.2.3 LCD 인터페이스

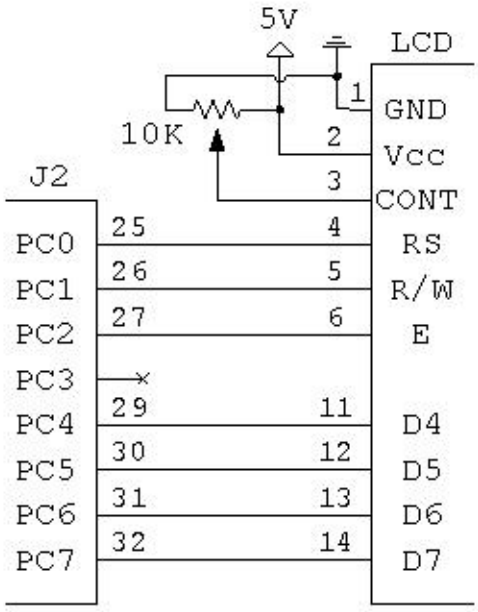


그림 11.3 키트의 4비트 LCD 인터페이스

마이크로컨트롤러와 LCD의 인터페이스는 4비트 인터페이스와 8비트 인터페이스 두 가지가 있다. 이는 사용하는 데이터 버스의 라인 수에 따라 구분이 된다. 8비트 인터페이스는 그림 11.1에 표시된 데이터 버스 8라인을 모두 사용하고, 4비트 인터페이스는 데이터 버스 라인 중 상위 4비트만을 사용한다. 8비트의 데이터를 전달하기 위해 4비트씩 두 번에 걸쳐 상위니블 그리고 하위니블 순으로 전달된다.

여기서는 그림 11.3과 같이 4비트 인터페이스를 사용하여 범용 입출력포트 C에 연결하도록 하자. (※ 그림 4.1에 표시되어 있는 보드상의 LCD인터페이스는 메모리 버스를 사용하는 것으로 여기서는 사용하지 않는다.)

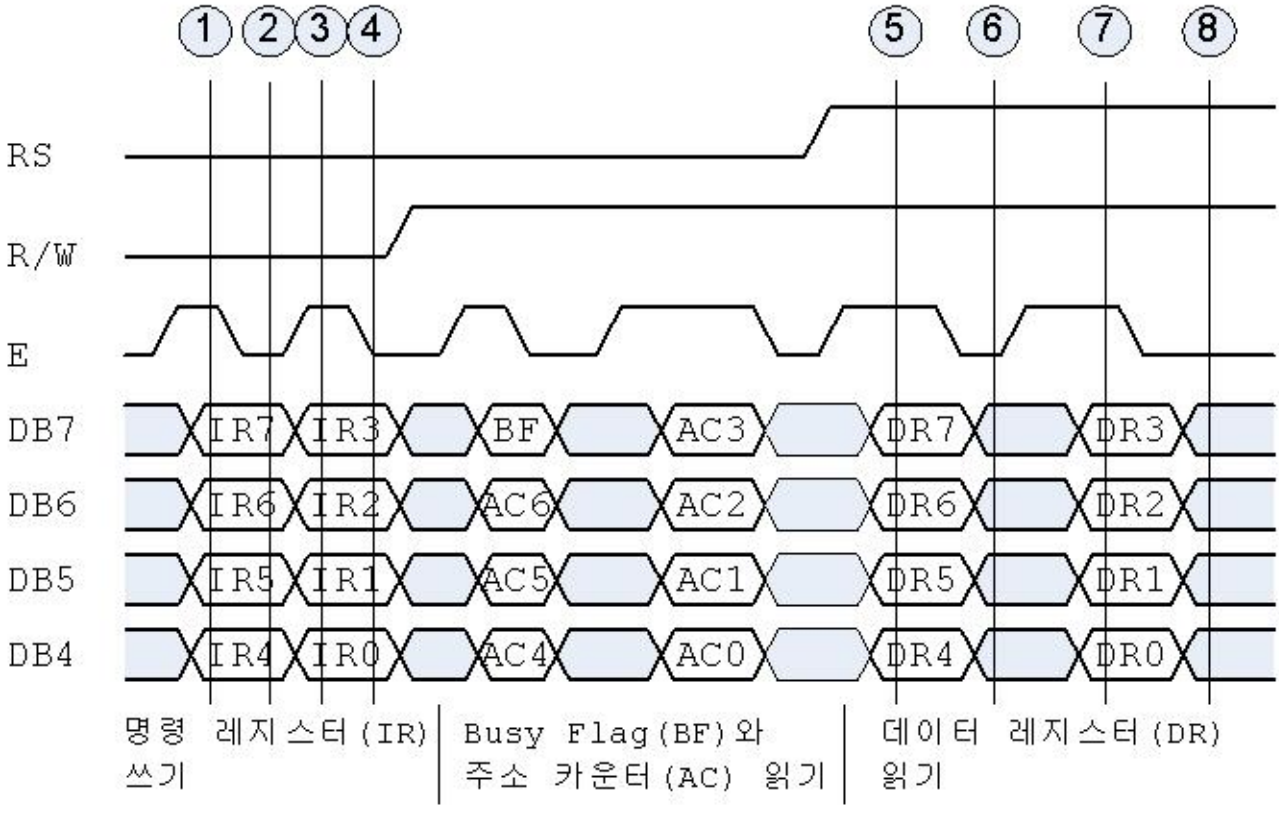


그림 11.4 4비트 인터페이스의 명령/데이터 입출력 타이밍

그림 11.4는 4비트 인터페이스를 사용할 경우 데이터 및 명령어 쓰기/읽기 타이밍을 보여준다. 입출력포트를 사용하여 LCD를 제어할 때 이 타이밍에 따라 포트의 입출력신호를 제어하여야 한다.

11.2.4 초기화

4비트 인터페이스를 사용할 때 LCD 모듈의 초기화 과정은 그림 11.5와 같다.

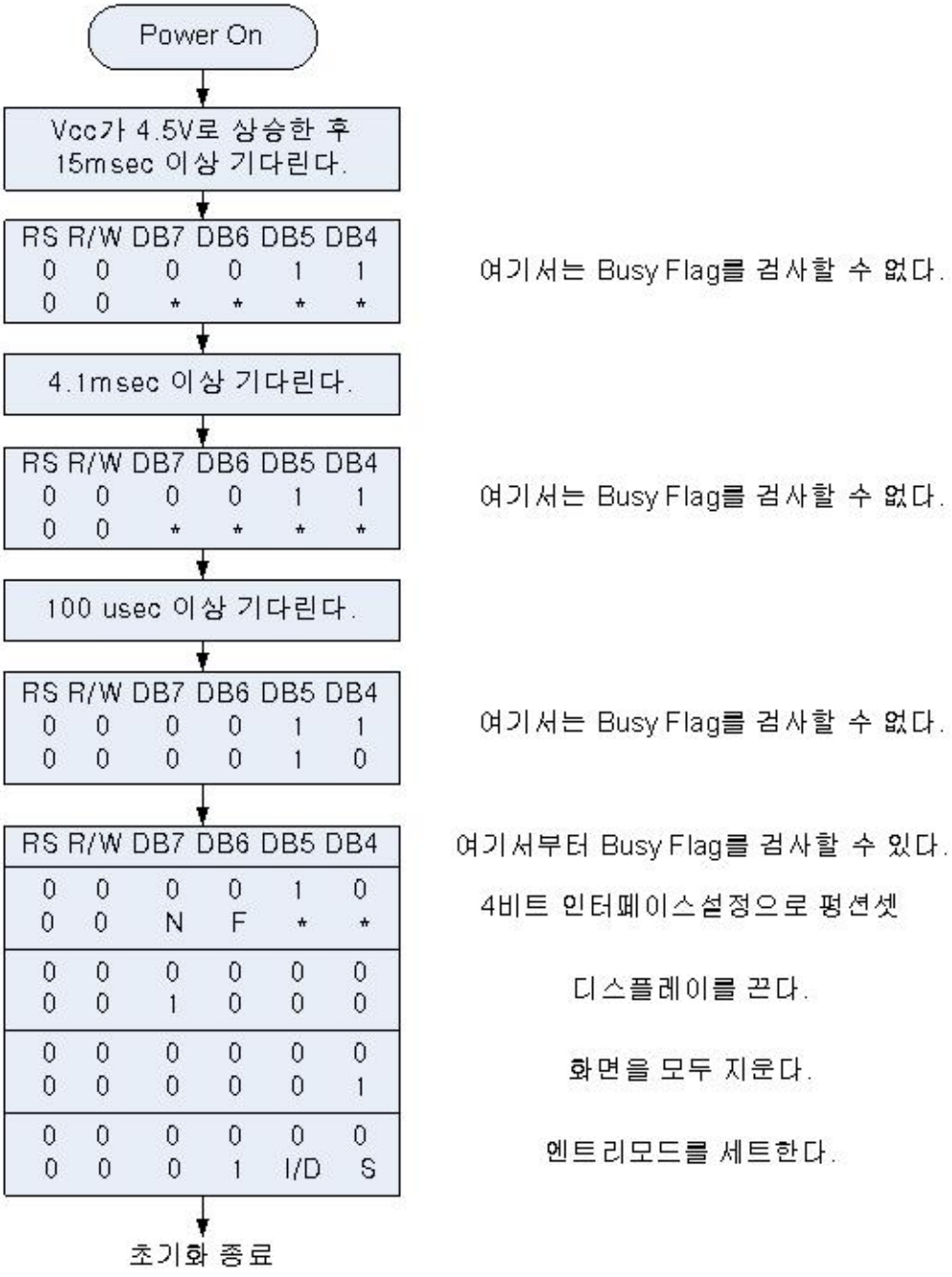


그림 11.5 4비트 인터페이스 LCD 초기화 과정

11.3 LCD 제어프로그램

11.3.1 모듈화 프로그래밍

LCD는 데이터를 화면에 표시하여 사용자에게 필요한 정보를 제공할 목적으로 사용되므로 대부분의 마이크로컨트롤러의 응용에 사용된다고 볼 수 있다. 따라서 한 번 LCD 제어프로그램을 작성하면 여러 응용에서 계속적으로 사용할 수 있다.

프로그램의 재사용에 있어서 중요한 것은 (1) 프로그램을 모듈화하여 LCD제어 모듈만 있으면 LCD를 사용할 수 있고, (2) 11.2절의 내용과 같은 LCD의 제어에 관한 자세한 사항을 몰라도 프로그램을 사용할 수 있어야 한다.

☞ 모듈화란 각 기능별로 프로그램을 작은 부분으로 나누는 것을 말한다. 모듈이 가져야할 기본적 조건은

- 기능적으로 다른 모듈들과 독립적으로 존재
- 독립성을 유지를 위해 데이터를 다른 모듈과 공유하지 않음.

로 표현된다. 한마디로 독립성이다. 독립성이 유지되지 않으면 LCD를 사용할 때 LCD제어 모듈 외에 다른 모듈을 부가적으로 사용하여야 한다. 독립성의 유지로 재사용 항목 (1)을 구현할 수 있다.

☞ 모듈은 하향식설계(Top-Down)방식에 의해 세부 모듈로 다시 구분한다. 각 세부 모듈 역시 모듈 간 독립성을 유지도록 하는 것이 바람직하나, 각 세부 모듈은 엄밀한 독립성을 유지 못할 수도 있다.

☞ 각 세부 모듈은 C-언어의 함수로 구현한다. LCD 제어에 대한 자세한 사항을 함수 내에서 처리되도록 하면 어떤 기능이 필요한 때 기능을 수행하는 세부모듈명(즉, 함수명)만 알면 해당모듈(함수)을 호출하기만 하면 된다. LCD를 초기화할 때 초기화 함수만 호출하면 되지 그림 11.5의 초기화과정을 이해할 필요가 없다. 재사용 항목 (2)는 자세한 기능적 사항을 함수가 처리하도록 함으로써 충족시킬 수 있다.

모듈의 독립성으로 얻을 수 있는 장점은 다음과 같이 기술할 수 있다.

- 프로그램 작성의 용이
- 프로그램 이해가 쉽다.
- 오류정정이 간단하다.
- 동시 개발이 가능

프로그램 모듈을 C-언어로 구현할 때 다음 세 부분으로 구성된다.

.c 프로그램파일	: 세부 모듈을 구현한 함수의 모음
.h 헤더파일	: 프로그램 모듈을 사용할 때 필요한 사항 (함수의 정의, 명령의 정의 등)을 포함
설명서	: 모듈의 사용방법, 기능 등을 기술. 간단한 경우는 헤더파일의 주석문으로 대체 가능

프로그램의 독립성을 위하여 각 부분별로 지켜야 할 사항은 다음과 같다.

- 내부데이터는 외부로부터 숨김
  - 전역변수의 사용 금지
- 내부적으로만 사용하는 세부 모듈을 외부에 숨김
  - static을 사용하여 정적함수로 작성
- 헤더파일에는 외부에서 꼭 알아야만 할 사항만 첨부

11.3.2 프로그램 설계

LCD는 화면에 필요한 데이터를 표시하기 위해서 사용한다. LCD의 기능은 다른 장치(예: 직렬통신장치)와는 독립적인 기능들이므로 LCD만 전용으로 제어하는 프로그램 모듈을 만들 수 있다. 이를 LCD제어 모듈이라 하자.

LCD제어 모듈이 외부 프로그램에 제공하여야 할 기능을 분류하여 보자.

LCD 제어모듈

(1) 초기화 기능
(2) 화면동작제어 기능
(3) 문자 표시 기능
    (3.1) 1 문자를 표시 기능
    (3.2) 문자열을 표시 기능
    (3.3) 표시위치 이동 기능
(4) 사용자 글꼴 등록 기능

위와 같이 외부에 공개될 기능을 수행하기 위해 모듈내부에서 수행되어야 할 기능은 표 11.4와 그림 11.4를 고려하면 5가지로 분류할 수 있다.

(1) 명령 레지스터(IR)에 명령 쓰기
(2) DDRAM 또는 CGRAM에 데이터 쓰기
(3) 비지 플래그/어드레스 카운터 읽기
(4) DDRAM 또는 CGRAM 읽기
(5) 시간지연

위에서 (4)의 읽기는 LCD드라이버 내부 레지스터를 데이터를 저장하는 메모리로 사용하는 것으로서 여기서는 사용하지 않으므로 모듈함수로서 작성하지 않을 것이다.

우선 내부기능을 함수로 작성하자. 함수의 역할은 함수에 필요한 데이터를 호출자로부터 받아 정해진 기능을 수행하고 수행결과를 호출자에게 전달하는 것이다. 따라서 함수를 작성할 때는

- 함수가 수행하여야 할 기능
- 이를 수행 하기위해 호출자로부터 받아야 할 입력
- 수행결과로 호출자에게 전달하여야 할 함수의 출력

을 정의하여야 한다. 이 정의로부터 입력을 위한 함수의 인자, 출력을 위한 함수의 인자 또는 반환 값을 선택한다.

11.3.3 내부 함수 작성

(1) 명령레지스터에 명령 쓰기 함수

표 11.3에서 명령레지스터에 전달할 명령은 여러 가지가 있다. 이 기능을 수행할 함수는 전달할 명령을 입력으로 받기로 한다. 함수의 수행결과를 함수 호출자에게 알릴 필요가 없다. 이로부터 함수의 입출력을 다음과 같이 선정한다.

인자 입력 : LCD에 내릴 명령어로 1바이트 형 ==> char command
인자 출력 : 없음
반환 값 : 없음 ==> void형

이로부터 함수 형태를 다음과 같이 선정한다.

```
static void write_command(char command)
{
    . . .
}
```

내부함수이므로 정적함수를 사용하여 외부에 함수를 숨긴다.

입력 인자 command가 다음과 같이 비트별로 표현된다 하면

비트 위치	D7	D6	D5	D4	D3	D2	D1	D0
command	b7	b6	b5	b4	b3	b2	b1	b0

표 11.4와 그림 11.4의 타이밍도를 참고하면 표 11.5와 같은 순서로 포트-C에 데이터를 출력하여야 한다.

출력 포트C 핀	D7	D6	D5	D4	D3	D2	D1	D0	
LCD모듈 핀	D7	D6	D5	D4	x	E	R/W	RS	
출력 포트C에 상위니블 출력	b7	b6	b5	b4	0	1	0	0	<= 그림 11.4 ①
	b7	b6	b5	b4	0	0	0	0	<= 그림 11.4 ②
출력 포트C에 하위니블 출력	b3	b2	b1	b0	0	1	0	0	<= 그림 11.4 ③
	b3	b2	b1	b0	0	0	0	0	<= 그림 11.4 ④

표 11.5 LCD 명령쓰기를 위한 데이터 출력 순서

☞ LCD 출력 포트-C를 매크로를 사용하여 LCD_PORT라고 정의하자.

```
#define LCD_PORT    PORTC
```

☞ 입력변수 command의 상위니블과 E,R/W,RS신호로 데이터를 구성한다.

```
temp = (command & 0xF0)  | 0x04 ;
       ================   =====
            ⇓               ⇓
                           0  0   0  0  0  1  0  0
       command의 상위니블   b7 b6 b5 b4 0  0  0  0
                 비트별 OR  ----------------------
       표 11.5의 데이터 ①   b7 b6 b5 b4 0  1  0  0
```

☞ 구성된 데이터를 LCD_PORT에 출력한다.(그림 11.4 ①)

```
LCD_PORT = temp;          // (그림 11.4 ①)
```

☞ 표 11.5의 데이터 ②를 LCD_PORT에 출력한다. (그림 11.4 ②)

```
LCD_PORT = temp   & ~0x04;    // (그림 11.4 ②)

          temp          b7 b6 b5 b4 0  1  0  0
          ~0x04         1  1  1  1  1  0  1  1
          비트별 AND    -----------------------
       표 11.5의 데이터 ②  b7 b6 b5 b4 0  0  0  0
```

☞ 입력변수 command의 하위니블과 E,R/W,RS신호로 데이터를 구성한다.

```
temp = (command << 4)     |  0x04 ;
       ================      =====
            ⇓                ⇓
                            0  0  0  0  0  1  0  0
       command의 하위니블    b3 b2 b1 b0 0  0  0  0
                 비트별 OR  ----------------------
       표 11.5의 데이터 ③    b3 b2 b1 b0 0  1  0  0
```

☞ 구성된 데이터를 LCD_PORT에 출력한다.(그림 11.4 ③)

```
LCD_PORT = temp;                    // (그림 11.4 ③)
```

☞ 표 11.5의 데이터 ④를 LCD_PORT에 출력한다.(그림 11.4 ④)

```
LCD_PORT = temp   & ~0x04;    // (그림 11.4 ④)
```

temp	b3 b2 b1 b0 0 1 0 0
~0x04	1 1 1 1 1 0 1 1
비트별 AND	----------
표 11.5의 데이터 ④	b3 b2 b1 b0 0 0 0 0

이로부터 write_command()함수를 완성하면 다음과 같다.

```
#define LCD_PORT   PORTC
static void write_command(char command)
{
     char temp;          // 임시변수

     // 상위니블 출력
     temp = (command & 0xF0) | 0x04;
     LCD_PORT = temp;              // RS=0, RW=0, E=1
     LCD_PORT = temp & ~0x04;      // RS=0, RW=0, E=0

     // 하위니블 출력
     temp = (command << 4) | 0x04;
     LCD_PORT = temp;              // RS=0, RW=0, E=1
     LCD_PORT = temp & ~0x04;      // RS=0, RW=0, E=0
}
```

(2) DDRAM 또는 CGRAM에 데이터 쓰기 함수

표 11.4에서 1바이트 단위로 DDRAM 또는 CGRAM에 데이터를 쓴다. 데이터 쓰기 함수는 DDRAM 또는 CGRAM에 쓸 1바이트 데이터를 함수의 입력으로 받는다. 함수의 수행결과를 함수 호출자에게 알릴 필요가 없다. 이로부터 함수의 입출력을 다음과 같이 선정한다.

인자 입력 : 1바이트 형 데이터 ==> char ch
인자 출력 : 없음
반환 값 : 없음 ==> void형

이로부터 함수 형태를 다음과 같이 선정한다.

```
static void write_data(char ch)
{
     . . .
}
```

입력 인자 ch가 다음과 같이 비트별로 표현된다 하면

비트 위치	D7	D6	D5	D4	D3	D2	D1	D0
ch	c7	c6	c5	c4	c3	c2	c1	c0

DDRAM과 CGRAM에 데이터를 쓰려면 데이터 레지스터(DR)을 선택(RS=1)하여 LCD에 데이터를 쓰면 된다. 표 11.4와 그림 11.4의 타이밍도를 참고하면 표11.6의 순서로 출력 포트C에 데이터를 출력하여야 한다.

표 11.6 DDRAM 또는 CGRAM에 데이터 출력 순서

출력 포트 C핀	D7	D6	D5	D4	D3	D2	D1	D0	
LCD모듈 핀	D7	D6	D5	D4	x	E	R/W	RS	
출력 포트C에 상위니블 출력	c7	c6	c5	c4	0	1	0	1	<= 그림 11.4 ⑤
	c7	c6	c5	c4	0	0	0	1	<= 그림 11.4 ⑥
출력 포트C에 하위니블 출력	c3	c2	c1	c0	0	1	0	1	<= 그림 11.4 ⑦
	c3	c2	c1	c0	0	0	0	1	<= 그림 11.4 ⑧

※그림 11.4에서는 데이터 읽기를 나타내므로 데이터 쓰기를 하려면 R/W=0으로 하여야 한다.

write_instruction()함수 작성과 같은 방법으로 표 11.6에 따라 write_data() 함수를 완성하면 다음과 같다.

```
static void write_data(char ch)
{
     char temp;          // 임시변수

     // 상위니블 출력
     temp = (ch & 0xF0) | 0x05;
     LCD_PORT = temp;              // RS=1, RW=0, E=1
     LCD_PORT = temp & ~0x04;      // RS=1, RW=0, E=0

     // 하위니블 출력
     temp = (ch << 4) | 0x05;
     LCD_PORT = temp;              // RS=1, RW=0, E=1
     LCD_PORT = temp & ~0x04;      // RS=1, RW=0, E=0
}
```

(3) 비지 플래그/어드레스 카운터 읽기 함수

비지 플래그 검사는 LCD가 현재 하고 있는 작업이 완료되었는지 판단을 하는 것이므로 작업수행에 필요한 시간만 지연하여도 비슷한 효과를 얻을 수 있다. 여기서는 시간지연으로 대체한다.

```
     #include <util/delay.h>
     static void check_busy(void)
     {
          _delay_ms(1);
     }
```

여기서 _delay_ms()는 WinAVR에서 라이브러리로 제공하는 함수이므로 헤더파일 "util/delay.h"를 첨부하여야 한다.

(4) 시간지연함수

시간지연함수는 WinAVR의 시간지연 라이브러리 함수를 사용한다. msec단위의 지연을 위해서는 _delay_ms(), μsec단위의 지연을 위해서는 _delay_us()함수를 사용한다.

☞ 라이브러리 함수를 쓰지 못할 때는 6.4절에서 작성하였던 msec_delay()함수를 시간지연함수로 사용하면 된다.

11.3.4 LCD모듈 함수 작성

여기서는 앞에서 작성한 내부함수를 사용하여 LCD 제어프로그램 즉 LCD 모듈을 구성하는 함수들을 하나의 파일에 모두 작성한다. 응용프로그램들은 이 함수들을 호출하여 LCD에 문자를 표시할 수 있다.

☞ 구성함수가 LCD 모듈의 일원이라는 것을 나타내기 위해 접두어 Lcd를 함수명에 사용하기로 한다.

☞ 표 11.4를 보면 명령의 각 비트를 적당히 세트/리세트하여 LCD에 여러 가지 명령을 내릴 수 있다. 실제로 LCD를 사용함에 있어서 모든 명령을 사용하는 것이 아니므로 많이 사용할 명령을 미리 분류하고 명령에 대한 코드 값을 추출한다.

화면지움 명령 : 코드 값 0b00000001 = 0x01
커서 홈 : 코드 값 0b00000010 = 0x02
엔트리모드 : 글자를 쓴 후 커서 위치가 증가함 ==> I/D = 1
글자를 쓸 때 화면의 이동이 없음 ==> S = 0
코드 값 0b00000110 = 0x06
화면 켜기 : 화면 켬, 커서 끔, 블링크 끔 ==> D=1, C=0, B=0
코드 값 0b00001100 = 0x0C
화면 끄기 : 화면 끔, 커서 끔, 블링크 끔 ==> D=0, C=0, B=0
코드 값 0b00001000 = 0x08

커서 켜기 : 화면 켬, 커서 켬, 블링크 끔 ==> D=1, C=1, B=0
코드 값 0b00001110 = 0x0E
커서 끄기 : 화면 켜기와 같음
커서 왼쪽이동 : 커서이동, 왼쪽이동 ==> S/C=0, R/L=0
코드 값 0b00010100 = 0x10
커서 오른쪽이동 :커서이동, 오른쪽이동 ==> S/C=0, R/L=1
코드 값 0b00010000 = 0x14
화면 왼쪽이동 : 화면이동, 왼쪽이동 ==> S/C=1, R/L=0
코드 값 0b00011100 = 0x1C
화면 오른쪽이동 :화면이동, 오른쪽이동 ==> S/C=1, R/L=1
코드 값 0b00011000 = 0x18
펑션 셋 : 4비트 인터페이스 ==> D/L = 0
2-line 화면 ==> N = 1
5x7도트 문자 사용 ==> F = 0
코드 값 0b00101000 = 0x28

☞ 코드 값으로는 명령의 의미를 파악하기 힘들므로 매크로를 사용하여 코드에 의미를 부여한다.

```
#define ALLCLR          0x01 // 화면을 지움
#define HOME            0x02 // 커서 홈
#define ENTMOD          0x06 // 엔트리모드
#define DISP_ON         0x0C // 화면 켜기
#define DISP_OFF        0x08 // 화면 끄기
#define CURSOR_ON       0x0E // 커서 켜기
#define CURSOR_OFF      0x0C // 커서 끄기
#define CURSOR_LSHIFT   0x10 // 커서 왼쪽 이동
#define CURSOR_RSHIFT   0x14 // 커서 오른쪽 이동
#define DISP_LSHIFT     0x18 // 화면 왼쪽 이동
#define DISP_RSHIFT     0x1C // 화면 오른쪽 이동
#define FUNSET          0x28 // 펑션세트
```

여기서부터 11.3.1 절에서 요구한 LCD제어 프로그램의 기능을 작성하도록 하자.

(1) 초기화 모듈 : LcdInit()

초기화를 위해 함수에 필요한 입력은 없으며 함수의 결과를 출력할 필요가 없다.

인자 입력 : 없음
인자 출력 : 없음
반환 값 : 없음 ==> void형

그림 11.5의 초기화 과정을 따라 다음과 같이 작성한다. 여기서 그림 11.5에서 필요한 비트들은 N=1, F=0, I/D=1, S=0으로 앞의 명령어를 구성할 때와 같은 것을 사용한다.

```
#define   LCD_DDR   DDRC        // 방향레지스터 정의

void LcdInit(void)              // LCD 초기화 함수
{
     LCD_DDR   = 0xFF;   // LCD포트를 출력포트로 설정

     _delay_ms(15);                 // 15msec delay
     write_instruction(0x30);       // 0x30=0b00110000
     _delay_ms(5);                  // 5msec delay
     write_instruction(0x30);
     _delay_ms(1);                  // 1msec delay
     write_instruction(0x32);       // 0x32=0b00110010

     LcdCommand(FUNSET);        //FUNSET:0x28=0b00101000
     LcdCommand(DISP_OFF);      //DISP_OFF:0x08=0b00001000
     LcdCommand(ALLCLR);        //ALLCLR:0x01=0b00000001
     LcdCommand(ENTMOD);        //ENTMOD:0x06=0b00000110
     LcdCommand(DISP_ON);       //화면을 켠다.
}
```

☞ LCD를 연결한 포트는 출력포트이므로 포트의 방향을 출력포트로 하여야 한다. LCD 포트 방향레지스터는 매크로를 사용하여 정의한다.

```
#define   LCD_DDR   DDRC      // 방향레지스터 정의
```

☞ 함수에서 FUNSET을 출력하는 부분부터는 비지플래그를 판별하여야 하므로 write_instruction()함수 대신 비지플래그를 판별하는 화면동작제어 함수인 LcdCommand()를 사용하였다. LcdCommand()함수는 뒤에서 작성될 것이다.

☞ 함수의 마지막 문장은 그림 11.5에서는 표시되지 않은 과정이다. 이는 그림 11.5의 초기화 후 화면이 꺼져 있는 상태가 되므로 화면을 켜기 위함이다.

(2) 화면동작제어 모듈(LcdCommand())

화면동작제어는 명령레지스터에 앞에서 매크로로 정의된 명령을 IR에 쓰면 된다. 단 write_instruction()함수로 명령을 쓰기 전에 비지플래그를 판별하여 LCD에 명령을 쓸 수 있을 때를 기다려야 한다. 함수의 입출력은

인자 입력 : LCD에 내릴 명령어로 1바이트 형 ==> char command
인자 출력 : 없음
반환 값 : 없음 ==> void형

```
void LcdCommand(char command)     // LCD 명령함수
{
     checkbusy();          // LCD 명령이 가능할 때까지 기다림
     write_instruction(command);  // 명령전달
}
```

☞ 입력 command로 어떠한 1 바이트 값을 입력해도 관계는 없으나, 의미가 있는 동작 명령을 주기위해서는 앞에서 정의한 매크로를 사용하는 것이 좋다. (예) LcdCommand(ALLCLR);   // 화면을 지운다.

(3) 문자 표시 모듈

(가) 한 문자 표시 모듈(LcdPutchar())

문자를 화면에 표시하려면 DDRAM에 데이터를 출력하면 된다. 이는 write_data()함수를 사용하면 되지만 데이터를 쓰기 전에 비지플래그를 판별하여 데이터를 쓸 수 있을 때를 기다려야 한다. 함수의 입출력은

인자 입력 : 화면에 표시할 문자(1바이트 형 데이터) ==> char
인자 출력 : 없음
반환 값 : 없음 ==> void형

```
void LcdPutchar(char ch)// LCD에 문자 하나를 쓰는 함수
{
     checkbusy();         // LCD 명령이 가능할 때까지 기다림
     write_data(ch);      // 데이터전달
}
```

☞ 화면에 나타나는 문자는 표11.3에 따라 출력된다.

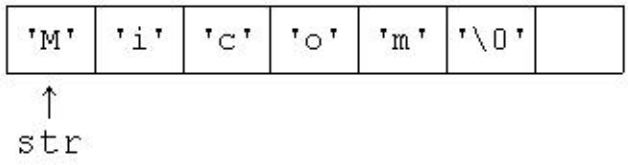
```
(예) LcdPutchar('A')          // 화면에 글자 'A'를 표시
     LcdPutchar(0x41);        // 'A'=0x41=0b01000001
                              // 화면에 글자 'A'를 표시
```

(나) SRAM영역의 문자열 표시 모듈(LcdPuts())

SRAM 영역의 문자열 입력을 받아 LCD 화면에 표시한다. 입출력은

인자 입력 : SRAM 영역의 문자열 ==> char * str
인자 출력 : 없음
반환 값 : 없음 ==> void형

문자열 입력을 str로 받는다면 str이 가리키는 문자를 출력하고 str이 가리키는 위치를 다음으로 이동한다.(str++) 이 동작을 NULL 문자를 만날 때까지 반복한다.



```
void LcdPuts(char *str) // LCD에 문자열을 쓰는 함수
{
     while(*str)    // *str이 NULL 문자가 아니면 루프를 돈다.
     {
          LcdPutchar(*str);  // 문자 *str을 화면에 출력
          str++;             // str이 다음 문자를 가리킴
     }
}
```

(다) 표시위치 이동(LcdMove())

앞의 함수 LcdPutchar()는 현재 DDRAM의 어드레스 카운터가 가리키는 위치에 문자를 출력하고 어드레스 카운터를 자동적으로 하나 증가시킨다. 따라서 LcdPutchar()는 이전에 쓴 문자 다음에 문자를 쓴다. 새로운 위치에 문자를 쓰려면 쓰기 전에 어드레스 카운터를 새로운 위치로 이동시켜야 한다. 이는 표 11.4의 DDRAM의 주소설정 명령으로 수행한다.

이 함수의 입력은 글자를 쓸 위치이다.

인자 입력 : 위치(행, 열) ==> char line, char pos
인자 출력 : 없음
반환 값 : 없음 ==> void형

함수 내에서는 행, 열 데이터로부터 DDRAM의 주소를 계산하고 이를 LCD 명령으로 출력하면 된다. 표11.2를 여기에 다시 표시하여 보면

화면 열	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
DDRAM	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	화면 0-행
주소	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	화면 1-행

0행에 해당하는 DDRAM의 주소는 열과 같고 1행에 해당하는 DDRAM의 주소는 화면 열에 0x40을 더하면 된다.

☞ (line << 6)은 line=0일 때 0이고 line=1일 때 0b01000000 = 0x40이다.

```
void LcdMove(char line, char pos) // line : 행, pos : 열
{
     char addr;

     addr = (line << 6) + pos;     // DDRAM의 주소 계산
     addr |= 0x80;  // 비트7을 세트 => DDRAM 주소설정 명령

     LcdCommand(addr);             // 명령 전달
}
```

11.3.5 파일 작성

LCD 제어모듈의 헤더파일과 소스파일은 다음 사항들을 고려하여 작성하였으며 이름은 각각 lcd.h 와 lcd.c이다.

(1) 헤더파일 작성

헤더파일의 작성에서 고려하여야 할 사항은 다음과 같다.

(1) 외부에서 호출하는 함수의 선언만 포함시키고, 내부함수는 외부에서 접근하는 함수가 아니므로 헤더파일이 아니라 소스파일에서 선언한다.
(2) 매크로 역시 외부에서 사용하는 것만 헤더파일에 포함하고 내부용은 소스파일에서 정의한다.
  - 11.3.3절 (1)에서 사용한 매크로 LCD_PORT는 내부용이고
  - 11.3.4절에서 정의한 매크로들은 외부에서 LCD 명령을 내릴 때 사용하므로 외부용이다.
(3) 헤더파일 중복 첨부를 피하기 위해 선행처리기 #ifndef-#endif쌍을 사용한다. 헤더파일의 중복을 피하기 위해 정의하는 매크로의 이름은 일반적으로 쓰지 않는 것을 택한다. (예) __LCD_H__

(2) 소스파일 작성

소스파일의 이름은 모듈의 내용을 반영하도록 채택하고 앞에서 작성한 내부함수와 외부 함수 모두를 다음 사항을 고려하여 포함시킨다.

(1) 외부함수의 정의는 모두 헤더파일에 포함되어 있으므로 헤더파일을 첨부하고 소스파일 내에는 내부함수의 선언 및 내부용 매크로만 정의한다.
(2) 함수를 작성할 때 함수의 기능 및 입출력관계를 주석문에 상세히 기록하여 함수의 사용방법을 알 수 있도록 한다.

[헤더파일 lcd.h]

```
//================================================
//  파일명 : lcd.h - LCD제어 모듈의 헤더파일
//================================================

#ifndef __LCD_H__
#define __LCD_H__

// LCD 제어 명령

#define ALLCLR          0x01 // 화면을 지운다.
#define HOME            0x02 // 커서를 홈으로 보낸다.
#define LN21            0xc0 // 커서를 2번째 라인의
                             // 첫번째에 위치시킴
#define ENTMOD          0x06 // entry mode
#define FUNSET          0x28 // function set
#define DISP_ON         0x0c // 디스플레이를 켠다.
#define DISP_OFF        0x08 // 디스플레이를 끈다.
#define CURSOR_ON       0x0e // 커서를 켠다.
#define CURSOR_OFF      0x0c // 커서를 끈다.
#define CURSOR_LSHIFT   0x10 // 커서를 왼쪽을 이동시킨다.
#define CURSOR_RSHIFT   0x14 // 커서를 오른쪽으로 이동시킨다.
#define DISP_LSHIFT     0x18 // 디스플레이를 왼쪽으로
                             // 이동시킨다.
#define DISP_RSHIFT     0x1c // 디스플레이를 오른쪽으로
                             // 이동시킨다.

// LCD 제어모듈 함수

void LcdInit(void);
void LcdCommand(char command);
void LcdMove(char line, char pos);
void LcdPutchar(char ch);
void LcdPuts(char* str);

#endif    // __LCD_H__
```

[프로그램 lcd.c]

```
//=========================================================
//  LCD.C : LCD 구동함수의 모음
//=========================================================
#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"

// LCD 포트 주소
#define LCD_PORT   PORTC
#define LCD_DDR    DDRC
// 내부 함수
static void checkbusy(void);
static void write_command(char command);
static void write_data(char ch);

//====================================
// 기능 : LCD Display를 초기화한다.
//====================================

void LcdInit(void)
{
     LCD_DDR   = 0xFF;              // LCD포트를 출력으로 설정
     _delay_ms(15);
     write_command(0x30);
     _delay_ms(5);
     write_command(0x30);
     _delay_ms(1);
     write_command(0x32);

     LcdCommand(FUNSET);
     LcdCommand(DISP_OFF);
     LcdCommand(ALLCLR);
     LcdCommand(ENTMOD);

     LcdCommand(DISP_ON);           // 화면을 켠다.
}
```

```
//=================================================
// LCD에 명령을 출력하는 함수
//
//   입력 : command - LCD에 내리는 명령
//          lcd.h에 정의된 명령을 사용할 것
//
//=================================================
void LcdCommand(char command)
{
     checkbusy();
     write_command(command);
}

//=================================================
// 현재위치에 문자 하나를 출력한다.
//
//   입력 : ch - 화면에 쓸 문자 코드
//
//=================================================
void LcdPutchar(char ch)
{
     checkbusy();
     write_data(ch);
}

//=================================================
// 현재위치에 문자열을 출력한다.
//
//   입력 : str - 출력할 문자열
//
//=================================================
void LcdPuts(char* str)
{
     while(*str)    // *str이 NULL 문자가 아니면 루프를 돈다.
     {
          LcdPutchar(*str);  // 문자 *str을 화면에 출력
          str++;             // str이 다음 문자를 가리킴
     }
}
```

```
//=================================================
// 글자를 쓸 위치를 지정된 위치(line, pos)로 이동시킨다.
// 입력 :  line - 화면의 행(0행부터 시작)
//         pos  - 화면의 열(0열부터 시작)
//=================================================
void LcdMove(char line, char pos)
{
     pos = (line << 6) + pos;
     pos |= 0x80;         // 비트 7를 세트한다.

     LcdCommand(pos);
}

//=========================================
// 명령 레지스터에 명령을 쓴다.
// 입력 : command - LCD에 내리는 명령 코드
//=========================================
static void write_command(char command)
{
     char temp;

     //
     // 상위 니블 출력
     //
     temp = (command & 0xF0)|0x04;//0x04 : RS=0(명령)
                                  // RW=0(쓰기),E=1
     LCD_PORT = temp;
     LCD_PORT = temp & ~0x04;     // E = 0

     //
     // 하위 니블 출력
     //
     temp = (command << 4) | 0x04; // 0x04 :RS=0(명령)
                                   // RW=0(쓰기),E=1
     LCD_PORT = temp;
     LCD_PORT = temp & ~0x04;      // E = 0
     _delay_us(1);                 // LCD에 따라서 1 usec
                                   // 지연이 필요한 경우가 있음
}
```

```
//=========================================
// 데이터 레지스터에 명령을 쓴다.
// 입력 : ch - LCD에 쓸 데이터
//=========================================
static void write_data(char ch)
{
     unsigned char temp;

     // 상위 니블 출력
     //
     temp = (ch & 0xF0)|0x05;      // 0x05:RS=1(데이터)
                                   // RW=0(쓰기),E=1
     LCD_PORT = temp;
     LCD_PORT = temp & ~0x04;      // E = 0

     // 하위 니블 출력
     //
     temp = (ch << 4) | 0x05;      // 0x05:RS=1(데이터)
                                   // RW=0(쓰기),E=1
     LCD_PORT = temp;
     LCD_PORT = temp & ~0x04;      // E = 0
}

//=========================================
// 1msec의 시간지연으로 BF 플래그 검사를 대체
//=========================================

static void checkbusy()
{
     _delay_ms(1);
}
```

11.4 실습

프로그램 prac11-4.c는 앞에서 작성한 LCD 제어프로그램의 사용방법을 보여주는 프로그램이다.

☞ LCD를 사용하므로 프로젝트에 LCD제어프로그램의의 소스파일인 lcd.c를 포함시켜야 한다.
☞ 프로그램에서 LCD제어 함수와 매크로를 사용하기 위해서 lcd.h를 포함시켜야 한다.
☞ lcd.c와 lcd.h는 프로젝트가 있는 폴더에 있어야 한다.
☞ 문자열을 나타내는 lcdtitle[]과 string[]은 문자열이므로 LcdPuts()함수를 사용하여 출력한다.

과제

1. 프로그램 prac11-4.c를 다음 동작을 하도록 변경한다.
   커서를 HOME으로 보낸 다음 커서를 켜고, 커서를 왼쪽으로 10번 오른 쪽으로 10번 이동하도록 한다.
2. 다음 동작을 하도록 프로그램을 작성하라.
   처음 윗줄에 "Merry"가 한자씩 출력되고 잠시 후 "Christmas!"가 출력된 후 아랫줄에서 "Happy New Year!"가 역시 한자씩 출력된다. 이 문자 전체가 3번 깜빡거리고 전체가 모두 왼쪽으로 이동하여 없어진다. 그리고 다시 윗줄 중간에 "Merry Christmas!" 아랫줄 중간에 "Happy New Year!"가 계속 깜빡거린다.
3. 소스파일 lcd.c는 LCD를 포트C에 연결하도록 작성되어 있다. LCD를 포트-A에 연결하고 동작시키려면 lcd.c에서 어느 부분을 어떻게 변경하여야 하는 가?

[프로그램 prac11-4.c]

```
//================================================
//  실습 11.4 - LCD제어 모듈 테스트
//================================================
#include <avr/io.h>           // I/O 레지스터정의
#include <util/delay.h>       // 시간지연함수
#include "lcd.h"              // LCD 제어프로그램 헤더파일

void msec_delay(int n);       // 시간지연함수
static char lcdtitle[] = "LCD Display";     // 실습 제목

int main()
{
     char i;
     char string[20];    // 문자열을 구성할 배열

     LcdInit();               // LCD 초기화
     LcdMove(0,0);            // HOME
     LcdPuts(lcdtitle);       // 실습 제목을 쓴다.

     string[0]  = 'M';
     string[1]  = 'E';
     string[2]  = 'C';
     string[3]  = 'H';
     string[4]  = 'A';
     string[5] = '\0';              // 문자열의 끝

     LcdMove(1,0);
     LcdPuts(string);         // SRAM내의 문자열 출력

     while(1)
     {
          for(i=0; i<10; i++)
          {
               msec_delay(1000);            //1초간 시간지연
               LcdCommand(DISP_RSHIFT);     // 오른 쪽 이동
          }
```

```
            for(i=0; i<10; i++)
            {
                msec_delay(1000);
                LcdCommand(DISP_LSHIFT);     // 왼쪽 이동
            }
        }
}

//============================
// 시간지연함수
//============================
void msec_delay(int n)
{
        for(; n >0; n--)          // 1msec 시간지연을 n회 반복
            _delay_ms(1);         // 1msec 시간지연
}
```