실험 6. LCD 디스플레이

전자공학과 21611646 유준상

Ⅰ 실험 목적

➔ [1] 여러 개의 파일로 구성된 프로젝트의 컴파일과 링크에 의한 다운로드 파일 생성

[2] 변수와 함수는 참조와 호출에 앞서 선언 또는 정의 확인

[3] LCD 디스플레이에 sprint( ) 함수를 이용하여 변수값 출력

[4] double형 출력을 위해 링커 옵션을 변경하는 방법 확인

Ⅱ 실험 도구 및 소자

➔ AVR Studio 4, 브레드 보드, ATmega128 보드, DC 어댑터, PWR B/D, 와이어 스트리퍼, 피복 단선 0.6mm, JTAG 다운로더, LCD, 가변 저항

Ⅲ 소스 코드

[main_3_8.c]

```
#include <stdio.h>
#include <avr/io.h>

#define  F_CPU   16000000UL
#include <util/delay.h>
#include "lcd.h"

// 전역변수 선언
long      a = 0, b = 0;
double   x = 0., y = 0.;

//////////////////////////////////////////////////////////////
```

```
int	main(void)
{	// 지역 변수 선언
	char lcd_string[2][MAX_LCD_STRING];

	LCD_init(); // LCD를 초기화하는 루틴(어떻게 사용할지를 사용 전에 미리 정하기)

	while(1){
		a = a + 1;
		b = b + 2;

		x += 0.1;
		y += 0.2;
// sprintf : 디스플레이할 글자를 메모리 공간 상에 변수를 잡아서 저장
// lcd_string[0], lcd_string[1] 각 행에 쓸 글자를 저장
		sprintf(lcd_string[0], "A=%-5ld B=%-5ld", a, b); // %-5ld : 5칸 decimal로
		LCD_str_write(0, 0, lcd_string[0]);
// %-5.1f : 5칸 float, 1자리 소수점
		sprintf(lcd_string[1], "X=%-5.1f Y=%-5.1f     ", x, y);
		LCD_str_write(1, 0, lcd_string[1]);
		_delay_ms(1000); // 1초마다 디스플레이 하기 위해 딜레이
	}
	return 0;
}
```

[lcd.h]

```
#ifndef  __LCD_H__ // if not defined, #define 문장으로 정의하지 않았다면 아래에 등장하는
		// #endif 문장 이전의 내용들을 include 영역에 포함시키라는 뜻
#define  __LCD_H_
// 디스플레이할 칸 수를 지정 (0x40 : 64칸)
#define  MAX_LCD_STRING		0x40
// 다른 소스 파일에 선언된 함수 사용
extern void	gen_E_strobe(void);
extern void	wait_BusyFlag(void);
extern void	LCD_command(unsigned char data);
extern void	LCD_data_write(unsigned char data);
extern void	LCD_init(void);
extern void	set_cursor(unsigned int row, unsigned int col);
extern void	LCD_str_write(unsigned int row, unsigned int col, char *str);
```

```
#endif
```

[lcd.c]

```
#include <avr/io.h>
#include "lcd.h"

#define  RS        PD5      // LCD 문자디스플레이에 연결된 포트D 의 핀번호
#define  RW        PD6      // RW 제어 신호를 PD6로 받기
#define  E         PD7      // Enable 신호를 PD7로 받기
// E 신호를 일정기간 유지시키는 함수
void      gen_E_strobe(void)
{         volatile  int       i;

          PORTD |= 1<<E;            // E 신호를 High로
          for(i=0; i<10; i++);                  // E 스트로브 신호를 일정기간 High로 유지
          PORTD &= ~(1<<E);                 // E 신호를 Low로
}
// 다른 명령이 동작 중인지 확인하고 기다리는 함수
void      wait_BusyFlag(void)
{         volatile int                  i;
          unsigned char               bf;
// 0번지 읽고 Busy Flag 읽기 위해 입력으로 지정
          DDRC = 0x0;                           // 포트C를 입력핀으로 설정
          PORTD = (PORTD & ~(1<<RS)) | 1<<RW; // RS <- Low, RW <- High
          do{
                    PORTD |= 1<<E;            // E 신호를 High로
                    for(i=0; i<10; i++);      // E 스트로브 신호를 일정기간 High로 유지
                    bf = PINC & 1<<PC7;       // busy flag 읽어 냄
                    PORTD &= ~(1<<E);         // E 신호를 Low로
          }while( bf );                         // bf 값이 0이 아니면 busy, 0 일 때까지 반복
}
// 명령어를 수행하는 함수
void      LCD_command(unsigned char data)
{         wait_BusyFlag();            // busy flag가 0될 때까지 대기, 1이면 다음 동작으로 넘어가기
          DDRC = 0xFF;                          // command 날리기 위해 출력모드로 지정
          PORTC = data;                         // 데이터 버스 상에 올리기
          PORTD &= ~(1<<RS | 1<<RW);    // RS <- 0, RW <-0 = 0번지 write
          gen_E_strobe();             // E 스트로브 신호 만들기
```

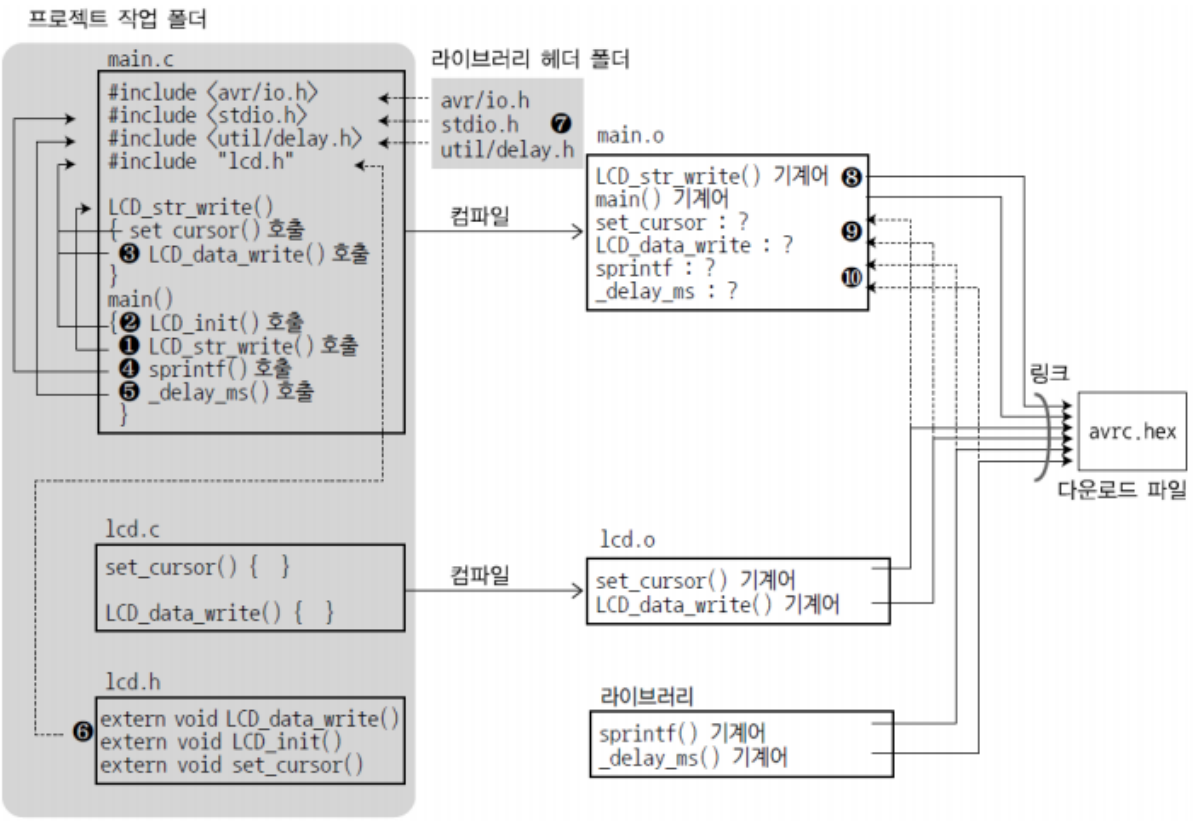
```
}
// 데이터를 쓰기 위한 함수
void	LCD_data_write(unsigned char data)
{	wait_BusyFlag();
	DDRC = 0xFF; // 출력모드로 지정
	PORTC = data; // 데이터 버스 상에 올리기
	PORTD = (PORTD | 1<<RS) & ~(1<<RW); // RS <- 1, RW <-0 = 1번지 write
	gen_E_strobe();
}
// LCD를 쓰기 전 초기화하는 함수
void	LCD_init(void)
{
	DDRD |= 1<<RS | 1<<RW | 1<<E; // RS(5), RW(6), E(7) 핀을 출력핀으로 설정

	PORTD &= ~(1<<RS | 1<<E | 1<<RW);	// 초기에 RS, E, RW <- 0
// 인터페이스/디스플레이 설정 : DL(1-8bit 인터페이스), N(1-두줄표시), F(1-문자5x10도트)
	LCD_command(0x3C);
// 커서 초기 위치 : 커서 위치를 1행 1열로 움직임
	LCD_command(0x02);
// 화면 클리어 : 스페이스에 해당되는 ASCII 문자 0x20인가
//		커서 위치를 1행 1열에 움직임
	LCD_command(0x01);
// 문자 입력 모드 : I/D(1), SH(0) – 커서를 오른쪽을 이동하면서 문자 입력
	LCD_command(0x06);
// 디스플레이 ON/OFF 제어 : D(1-디스플레이ON), C(1-커서ON), B(1-커서깜박임ON)
	LCD_command(0x0F);
}
// (row, col)을 받아 해당 위치로 커서 세팅
void	set_cursor(unsigned int row, unsigned int col)
{
	LCD_command(0x80 + (row % 2) * 0x40 + (col % 0x40));
}
// 함수 정의 : row, col 위치에서 문자열 str 을 LCD에 출력시킨다.
void	LCD_str_write(unsigned int row, unsigned int col, char *str)
{
	int	i;

	set_cursor(row, col);
// 문자열(space 포함)이 끝날 때까지 반복
```

```
        for(i=0; (i+col < MAX_LCD_STRING) && (str[i] != '₩0'); i++)
                LCD_data_write(str[i]);
}
```

Ⅳ 사전 지식

➔

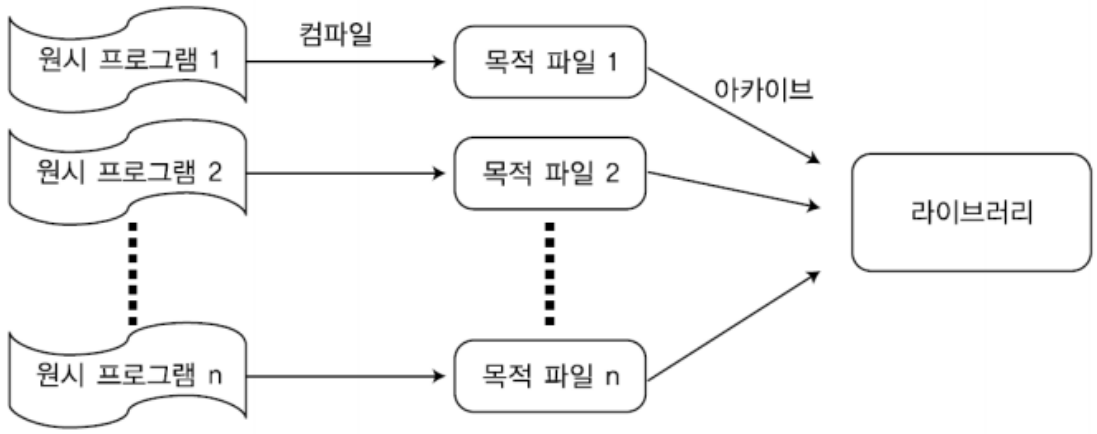
이전의 실험들과는 다르게 이번 실험에서는 총 세 개의 파일(main_3_8.c, lcd.h+lcd.c)을 컴파일을 통해 목적코드(object code)로 변환하고 링커를 통해 합치는 과정이 추가적으로 필요하다. 이 과정이 필요한 이유는 main.o 파일에서 다른 목적코드의 함수 사용이 필요하기 때문이다. 전체적인 과정은 아래와 같다.



위의 코드 그대로 sprint( )를 사용하게 되면, a, b int type은 제대로 출력되지만 부동 소수점이 필요한 double type인 x, y는 제대로 lcd에 디스플레이되지 않는 현상이 나타난다. 부동 소수점 기능을 사용하기 위해 라이브러리를 링킹하여 사용했다. 라이브러리 설명은 아래의 이미지로 대체한다.

□ 라이브러리

- 라이브러리는 미리 컴파일된 목적 파일의 집합
- 라이브러리 파일명은 보통 lib로 시작하고 확장자는 .a
- 일반 프로그래머도 아카이버 ar 명령으로 라이브러리를 만들어 사용

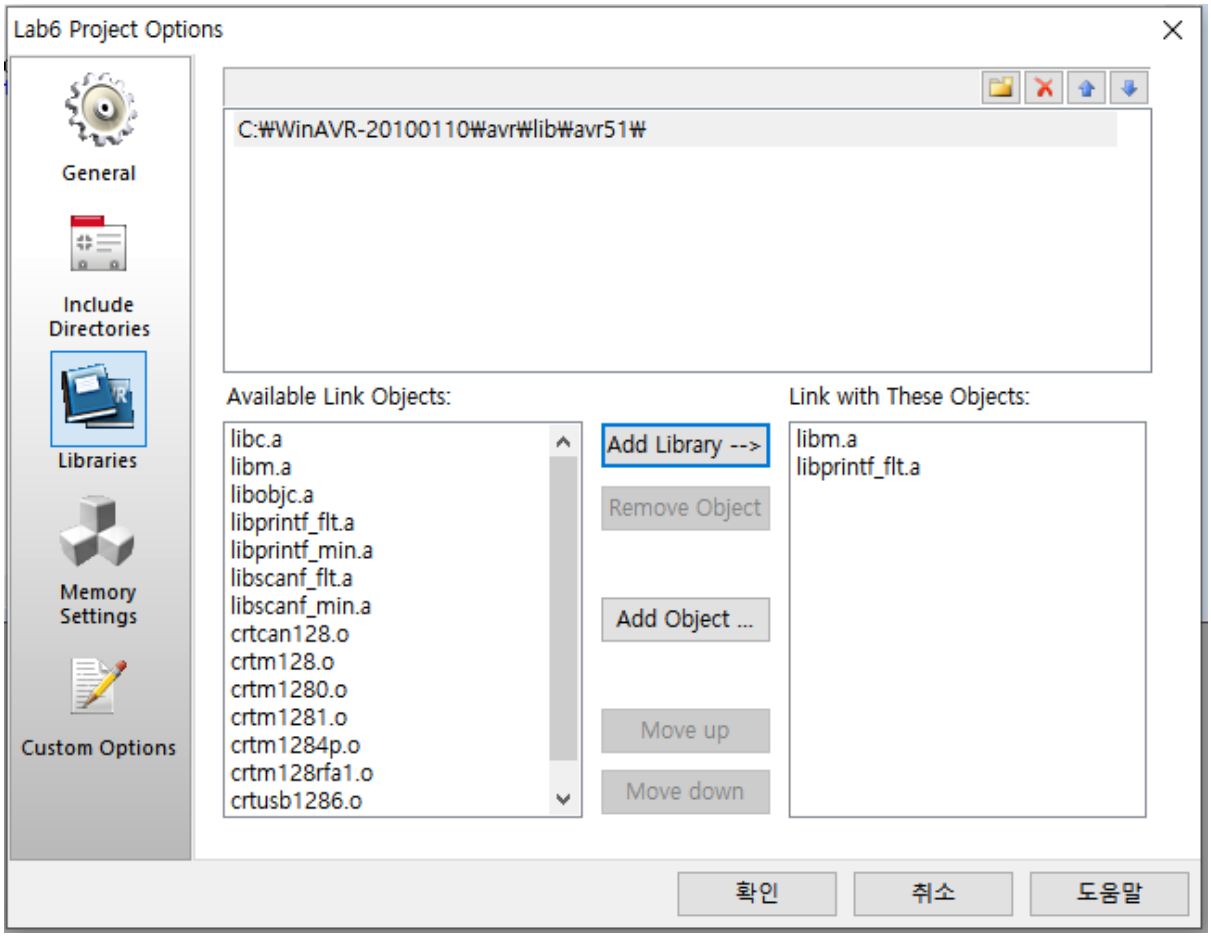


Atmega128은 avr51 architecture용으로 빌드된 라이브러리를 사용하므로 아래와 같이 location을 지정해 주었다. 그리고 avr51에서 포함된 두 라이브러리를 포함시켰다.

- **printf( ) 함수**
  - vfprintf( ) 함수를 기본으로 변형된 함수
  - vfprintf( ) 함수는 % 문자 뒤에 d, c, u, f, o, x 등 매우 다양한 출력 기능
  - 타깃 코드의 크기를 위한 3가지 링커 옵션

구분	기능	라이브러리	링커 옵션	파일 크기
기본형	부동소수점에 대한 출력을 제외하고, 모든 % 형식의 출력 지원	libc.a	없음	보통
최소형	매우 기본적인 정수형과 문자형에 대한 % 형식의 출력만을 지원	libpritnf_min.a	-Wl,-u,vfprintf -lprintf_min	작아짐
충족형	부동소수점에 대한 출력까지 포함한 모든 % 형식의 출력 지원	libprintf_flt.a	-Wl,-u,vfprintf -lprintf_flt	커짐

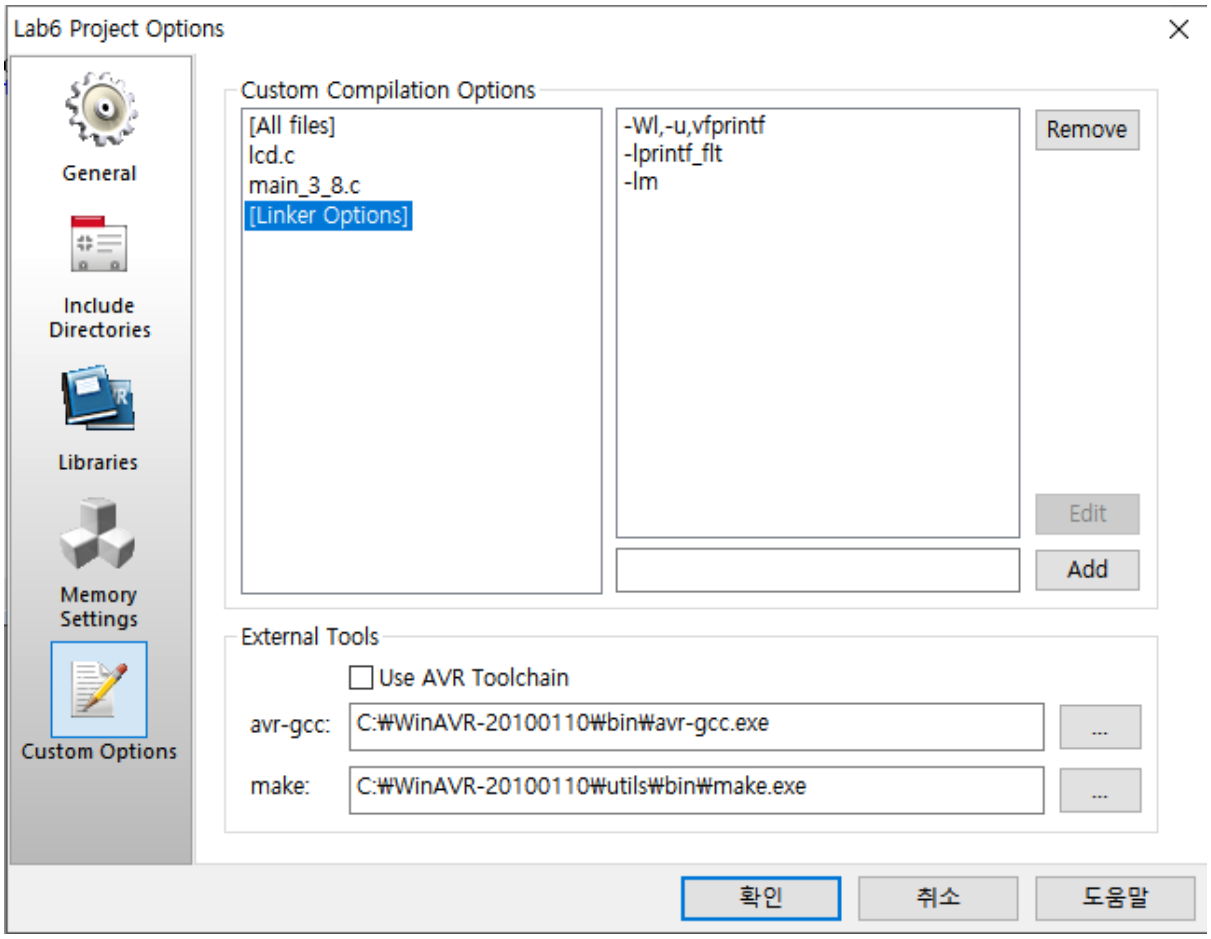
avr studio4 내의 설정은 아래와 같이 하면 된다.



컴파일할 때 라이브러리를 링킹해주기 위해 아래와 같은 링커 옵션을 사용하였다.

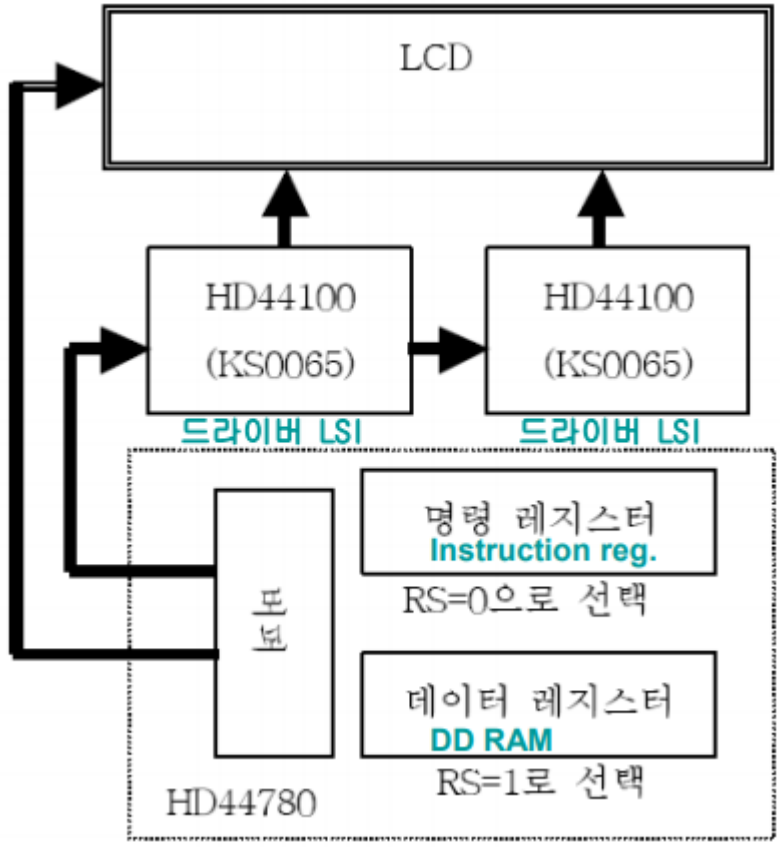
- **printf( ) 함수에서 float와 double 데이터를 출력하는 설정**
  - Project → Configuration Options → Custom Options → Linker Options
  - 아래 옵션 내용 추가

```
-Wl,-u,vfprintf -lprintf_flt
```



여기서 –lm을 추가한 이유는 링크 과정에서 libm.a 라이브러리 탐색을 위해서 추가하였다.

LCD는 Liquid Crystal diode의 약자로 7-segment LED에 비하여 LCD 모듈은 모든 ASCII문자들을 표현할 수 있다. 밝게 보기 위해 LED 등을 사용하여 밝게 조명하는 기능을 부착한 백라이트 형이 있다. 본 실험에서는 백라이트가 없는 LCD를 사용하였다. 둘을 구분하는 방법은 실험에 사용할 LCD가 14핀(백라이트x)인지 16핀(백라이트o)인지로 알 수 있다. LCD의 구조는 아래와 같다.



명령 레지스터로는 CG ROM이 있는데, ASCII code가 가리키는 번지를 가지고 있다. 데이터 레지스터는 CG RAM과 DD RAM이 있다. LCD 모듈의 DD RAM은 표시할 문자들의 ASCII 코드 데이터가 저장되는 내부 메모리이다. 화면의 각 행과 열의 문자 위치에는 고유한 어드레스 값이 부여되어 있다.

16 문자 x 2 라인 LCD 의 DD RAM 번지:

표시 위치	1	2	…	15	16
1 줄[Hex]	00	01	…	0E	0F
2 줄[Hex]	40	41		4E	4F

1번째 행은 00H~0FH이고, 2번째 행은 40H부터 시작된다.

RS(Register Select) bit로 명령 레지스터, 데이터 레지스터 중 사용할 레지스터를 선택할 수 있다. R/W는 Read/Write(bar)로 1일 때 읽기 동작을 0일 때 쓰기 동작을 제어하는 bit이다.

RS,R/W 선택 표:

RS	R/W	동 작
0	0	각종 제어 명령 쓰기(Clear Disp, Cursor Home… 등)
0	1	Busy Flag(D7) ㅣ Address_Counter(D6~D0) 읽기 (Busy Flag: 1=Busy, 0=Not Busy)
1	0	CGRAM, DDRAM에 데이터 쓰기 동작
1	1	CGRAM, DDRAM에 데이터 읽기 동작

RS, R/W 선택에 따라 위와 같은 동작을 수행한다. Busy Flag는 앞에 내보낸 명령어나 데이터가 완전히 들어갔는지 확인한다. 이때 D7인 최상위 비트가 1이면 동작 중, 0이면 동작이 완료됨을 뜻한다. RS=1일 때 데이터 레지스터의 어느 번지에 어떤 ASCII code를 쓸 건지를 데이터 버스로 옮기는 동작을 한다.

아래 표는 각 code에 따른 동작 명령과 인터페이스 신호를 나타낸다.

동작 명령	RS	R/W	D7~D0							
			D7	D6	D5	D4	D3	D2	D1	D0
화면 클리어	0	0	0	0	0	0	0	0	0	1
커서 초기 위치	0	0	0	0	0	0	0	0	1	–
문자 입력 모드	0	0	0	0	0	0	0	1	I/D	SH
디스플레이 On/Off 제어	0	0	0	0	0	0	1	D	C	B
커서 디스플레이 천이	0	0	0	0	0	1	S/C	R/L	–	–
인터페이스/디스플레이 설정	0	0	0	0	1	DL	N	F	–	–
CGRAM 주소 설정	0	0	0	1	CGRAM 주소(AC5~AC0)					
DDRAM 주소 설정	0	0	1	DDRAM 주소(AC6~AC0)						
BF과 AC 읽기	0	1	BF	주소 카운터(AC6~AC0)						
RAM 데이터 읽기	1	0	데이터(D7~D0)							
RAM 데이터 쓰기	1	1	데이터(D7~D0)							

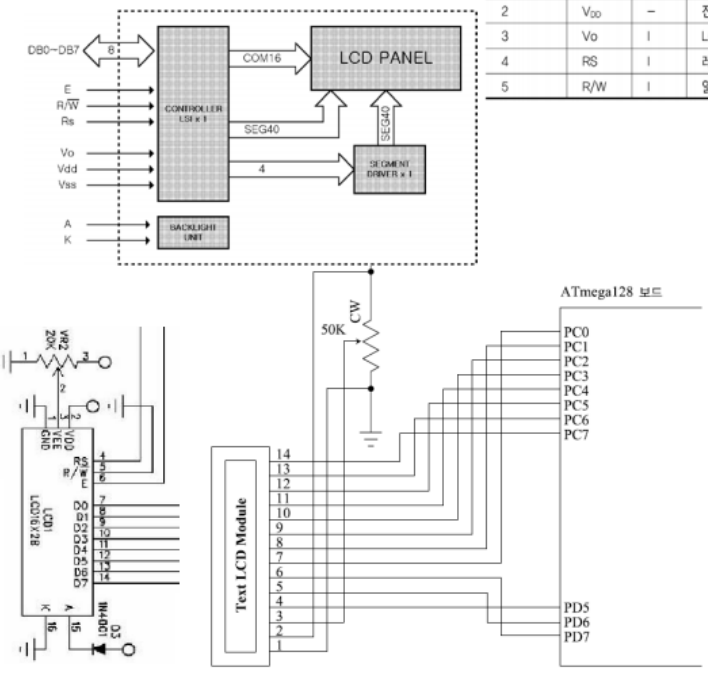
동작 명령	비트	동작
화면 클리어	–	스페이스에 해당되는 ASCII 문자 0x20 인가 커서 위치를 제1행 1열에 움직임
커서 초기 위치	–	커서 위치를 제1행 1열로 움직임
문자 입력 모드	(I/D, SH)	(1, 0) 커서를 오른쪽으로 이동하면서 문자 입력 (0, 0) 커서를 왼쪽으로 이동하면서 문자 입력 (–, 1) 커서 이동 없이 같은 자리에서 문자 입력
디스플레이 On/Off 제어	D	1 : 디스플레이 ON 0 : 디스플레이 OFF
	C	1 : 커서 ON 0 : 커서 OFF
	B	1 : 커서 깜박임 ON 0 : 커서 깜박임 OFF
커서 디스플레이 천이	(S/C, R/L)	(1, 0) 화면 전체 커서와 함께 왼쪽으로 스크롤 (1, 1) 화면 전체 커서와 함께 오른쪽으로 스크롤 (0, 0) 커서만 좌로 이동 (0, 1) 커서만 우로 이동
인터페이스/ 디스플레이 설정	DL	1 : 8비트 인터페이스 0 : 4비트 인터페이스
	N	1 : 두 줄 표시 0 : 한 줄 표시
	F	1 : 문자 5×10 도트 0 : 문자 5×7 도트
CGRAM 주소 설정	AC5~AC0	CGRAM 주소 카운터(AC) AC5~AC0값 설정
DDRAM 주소 설정	AC6~AC0	DDRAM 커서 위치 설정 AC6~AC0값 설정 0x00~0x0F : 제1행의 0열부터 15열까지 지정 0x40~0x4F : 제2행의 0열부터 15열까지 지정
BF과 AC 읽기	BF, AC6~AC0	내부 동작 여부에 따른 Busy Flag와 주소 카운터값을 읽음 BF : 1 내부 동작 진행 중 BF : 0 내부 동작 완료, 다음 동작 명령 수령 가능

이번 실험에서 사용한 회로는 아래와 같다.

□ 실험 회로

핀 번호	명칭	I/O	기능
1	V_{SS}	–	Ground
2	V_{DD}	–	전원
3	Vo	I	LCD 밝기 조절
4	RS	I	레지스터 선택
5	R/W	I	읽기/쓰기

핀 번호	명칭	I/O	기능
6	E	I	읽기/쓰기 가능 신호
7~14	D0~D7	I	데이터 입출력
15	BLA	I	백라이트 애노드
16	BLK	I	백라이트 캐소드



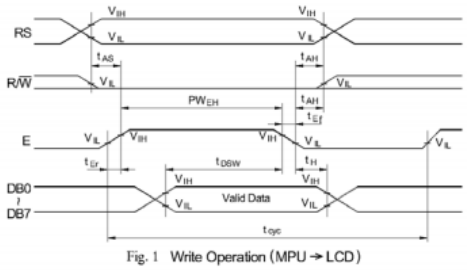


Fig. 1 Write Operation (MPU → LCD)

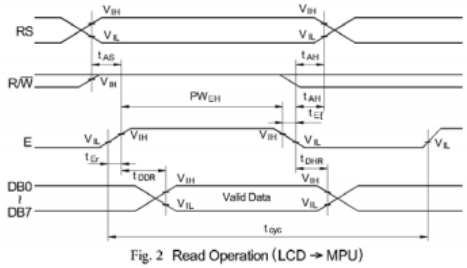


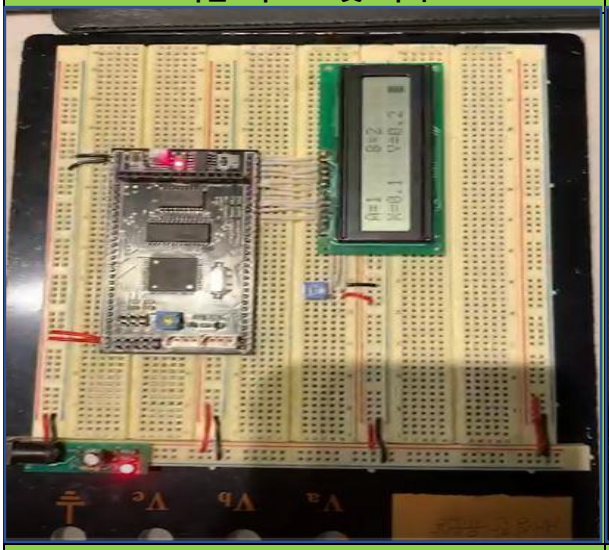
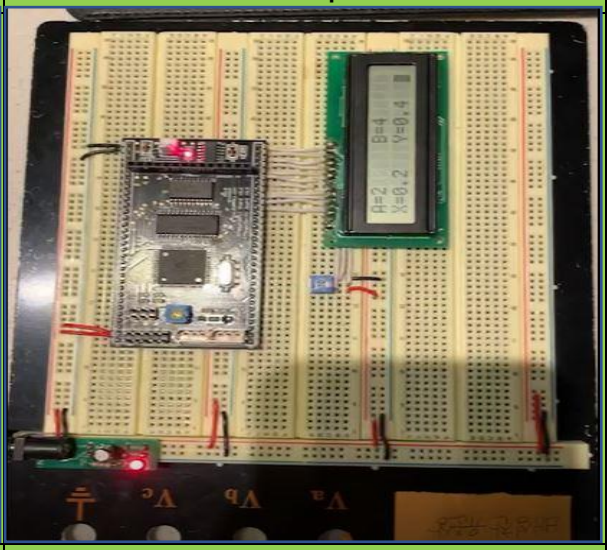
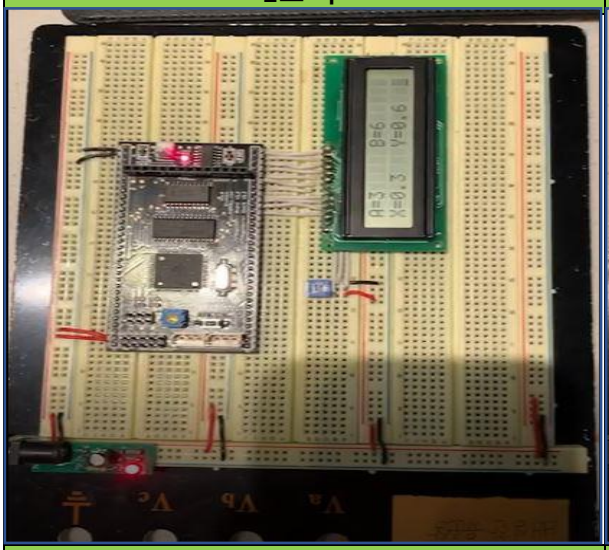
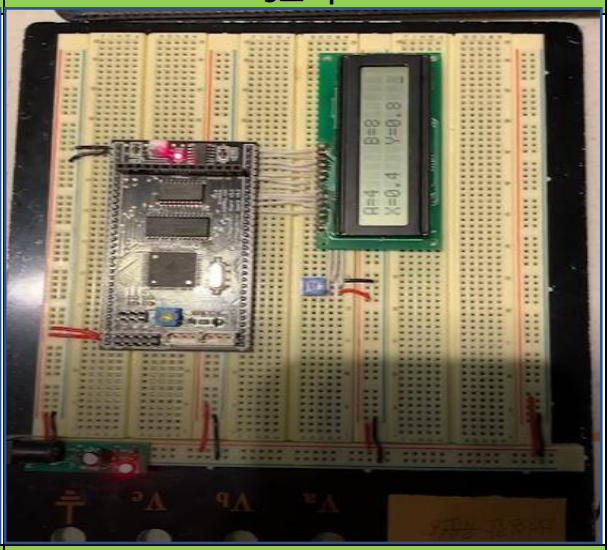
Fig. 2 Read Operation (LCD → MPU)

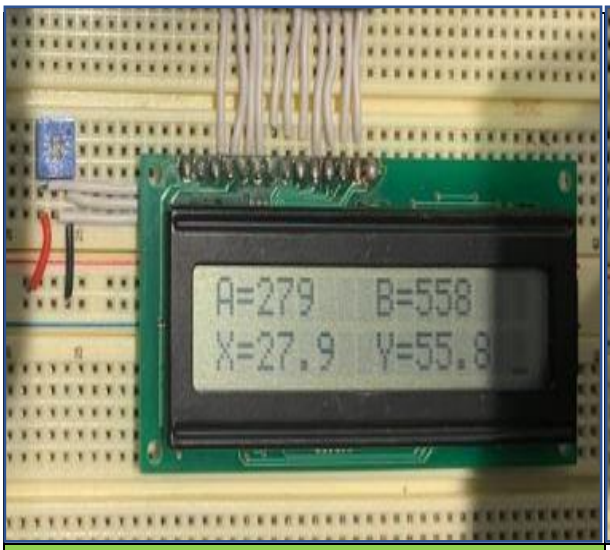
추가적으로 쓰기/읽기 동작도 볼 수 있는데 두 동작 모두 E(Enable)신호가 High일 때 동작을 수

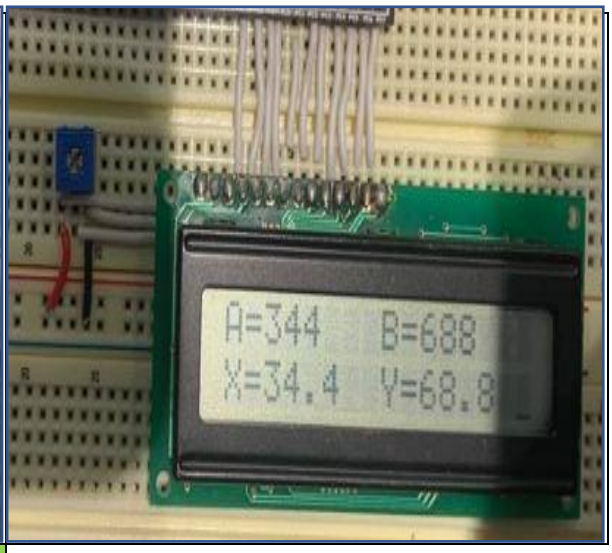
행한다.

V 회로도 및 7-segment 작동

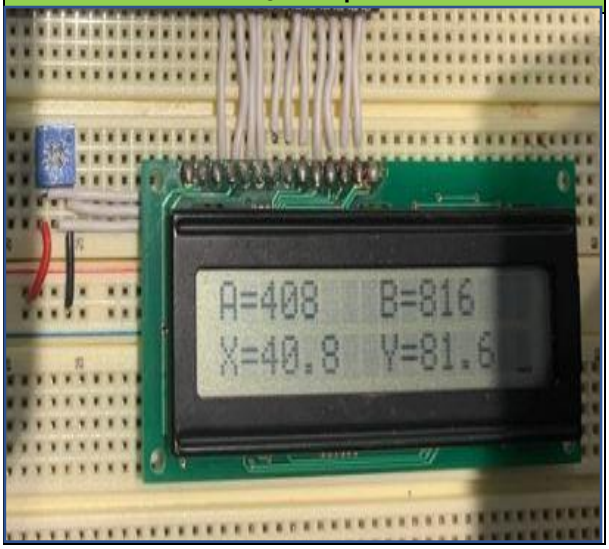
➔ 위의 회로와 동일하게 연결하여 회로를 구성하였고 1초마다 a는 1씩, b는 2씩, x는 0.1씩, y는 0.2씩 증가함을 볼 수 있다.

기본 회로도 및 시작	1초 후
	
2초 후	**3초 후**
	
278초 후	**343초 후**





407초 후



V 결과 및 토의

➔ 이번 실험에서는 Atmega128과 LCD를 연결하여 LCD 디스플레이에 변수값을 출력하는 실험을 수행하였다. 부동 소수점을 출력하기 위해 부동 소수점 출력을 지원하는 라이브러리를 링커 옵션과 함께 추가적으로 링킹하여 수행하였다. 이번 실험을 통해 헤더파일을 #include로 불러오기와 세 파일을 합치기 위해 목적코드로 컴파일 후 링크하는 것도 배웠다.