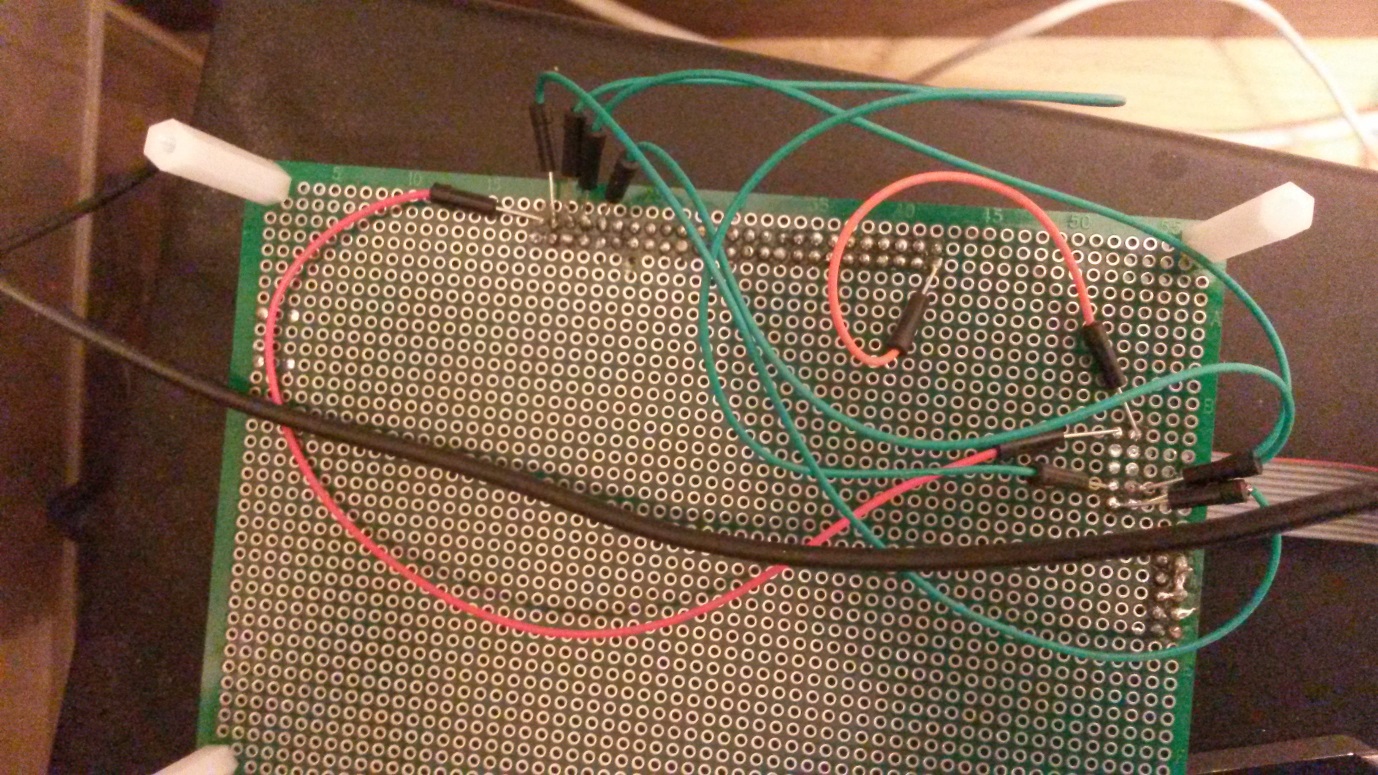
마이크로프로세서 응용 텀 프로젝트 보고서

2009034723 김진희

2013048956 민유림

**1. 회로구성**

A. Easy Processor Kit의 EXT I/O의 Vdd, EX0\_PORTA의 하위 4비트, GND를 각각 스탭모터 컨트롤의 Vcc, A, /A, B, /B에 해당하는 1, 3, 2, 4 핀과 GND에 연결을 쉽게 하기 위해 다음과 같이 회로를 구성함



B. 스탭모터 컨트롤을 EXT I/O와 연결하기 위한 연결선과 커넥터는   
지인을 통해서 구함

**2. C코드 작성**

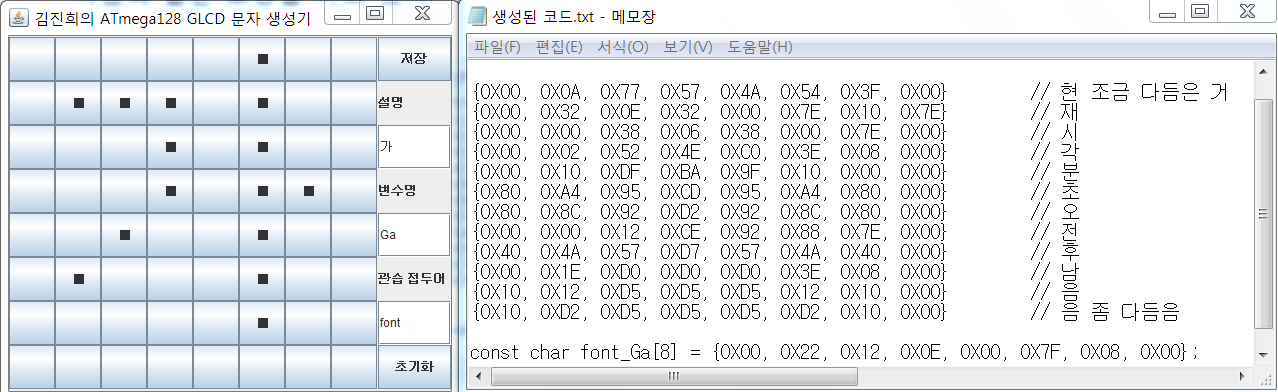
1) 그래픽 LCD와 스탭모터를 사용하기 위해 예제소스에서 파일을 복사하고   
코드 관리하기 수월하기 위해, 헤더파일로 되어있는 코드를 .c파일로 바꾸고, Moter.c파일을 만들어 그래픽 LCD와 관련된 소스파일과 스탭모터와 관련된   
소스파일을 분리함

2) 소스 파일을 읽기 쉽게 하기 위해, 함수가 길어질때마다 일부분을 빼서 함수로 만들고, 여러 파일로 나누어서 작업했는데,   
파일이 너무 많아지다보니까, 불편한 점이 있어서 하위 폴더를 만듬.

3) 제공된 소스 중 그래픽 LCD에 표시할 글자는 일부 제공되지 않아서,   
글자가 정의되는 규칙을 노트에 이진수 형태로 적어 분석해서 그래픽 LCD 상단에 CLOCK이라는 글자의 L자를 직접 만듬.



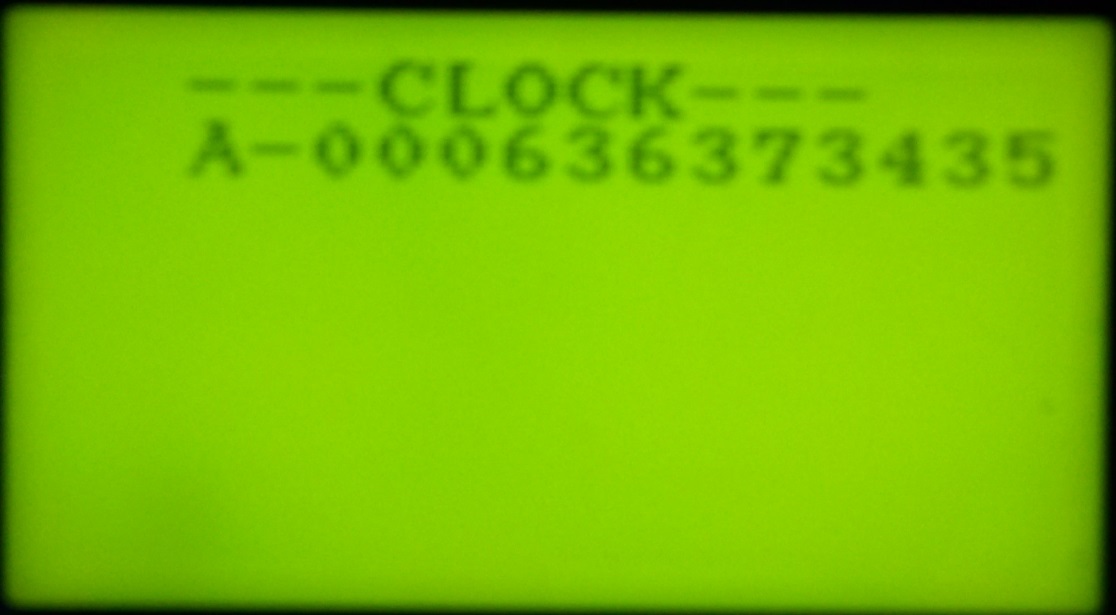
첫 번째 그림은 시행착오.

4) L자를 만드는데 시간이 너무 오래걸려서, 손으로 노트에 그릴 것을 컴퓨터가   
내부적으로 대신 하게 만들기 위해 Eclipse로 글자 제작하는 자바 프로그램을   
제작함. 

그리고 자바 프로그램을 만들어 놓고 보니, 아예 이걸로 한글 문자를 제작해서 넣으면 좀더 참신 할 것 같아서, 그래픽 LCD에 한글을 표시하기로 하게 됨.

5) 그래픽 LCD에 글자 출력할 위치를 정하기 수월하게 하기 위해 RowAndColDefine.h라는 파일에 #define |행 혹은 열 이름| |수| 로 정함

6) 글자 출력할 위치를 정하기 수월하게 하기위해 RowAndColDefine.h   
파일에 행과 열의 이름을 정했는데 불구하고 다음 그림과 같이  
정해놓은 위치에 출력되지 않는 이상한 현상이 있었음.



그래서 난감해 하다가 예제소스에 제공된 함수인  
GLcdClr함수랑 GraphLcdCharactorSet의 차이점을   
비교해서 GraphLcdCharactorSet함수의 일부분을 다음과 같이 수정해서,  
문제를 해결함

GLcdCmd(0xC0,0xB8 + line );  
//GLcdCmd(0x40,gPoz + i); // 요녀석을 내가 아래와 같이 수정함  
 GLcdCmd(0xC0,gPoz + i);  
GLcdSetData(gSet,Charactor[i]);



7) 두 자리 숫자를 표시하기 위해 다음과 같이 함수를 만듬

void ShowNumber(ShortInt number, ShortInt row, ShortInt col)

{

char \*h\_font, \*l\_font;

two\_int\_to\_font(number, &h\_font, &l\_font);

GraphLcdCharactorSet(row,col,h\_font);

GraphLcdCharactorSet(row,col+1,l\_font);

}

void two\_int\_to\_font(ShortInt two\_ShortInt, char \*\*p\_h\_font, char \*\*p\_l\_font)

{

ShortInt h\_ShortInt = two\_ShortInt/10;

ShortInt l\_ShortInt = two\_ShortInt%10;

\*p\_h\_font = one\_int\_to\_font(h\_ShortInt);

\*p\_l\_font = one\_int\_to\_font(l\_ShortInt);

}

char\* one\_int\_to\_font(ShortInt one\_ShortInt)

{

switch(one\_ShortInt) {

case 0 : return font\_0;

case 1 : return font\_1;

case 2 : return font\_2;

case 3 : return font\_3;

case 4 : return font\_4;

case 5 : return font\_5;

case 6 : return font\_6;

case 7 : return font\_7;

case 8 : return font\_8;

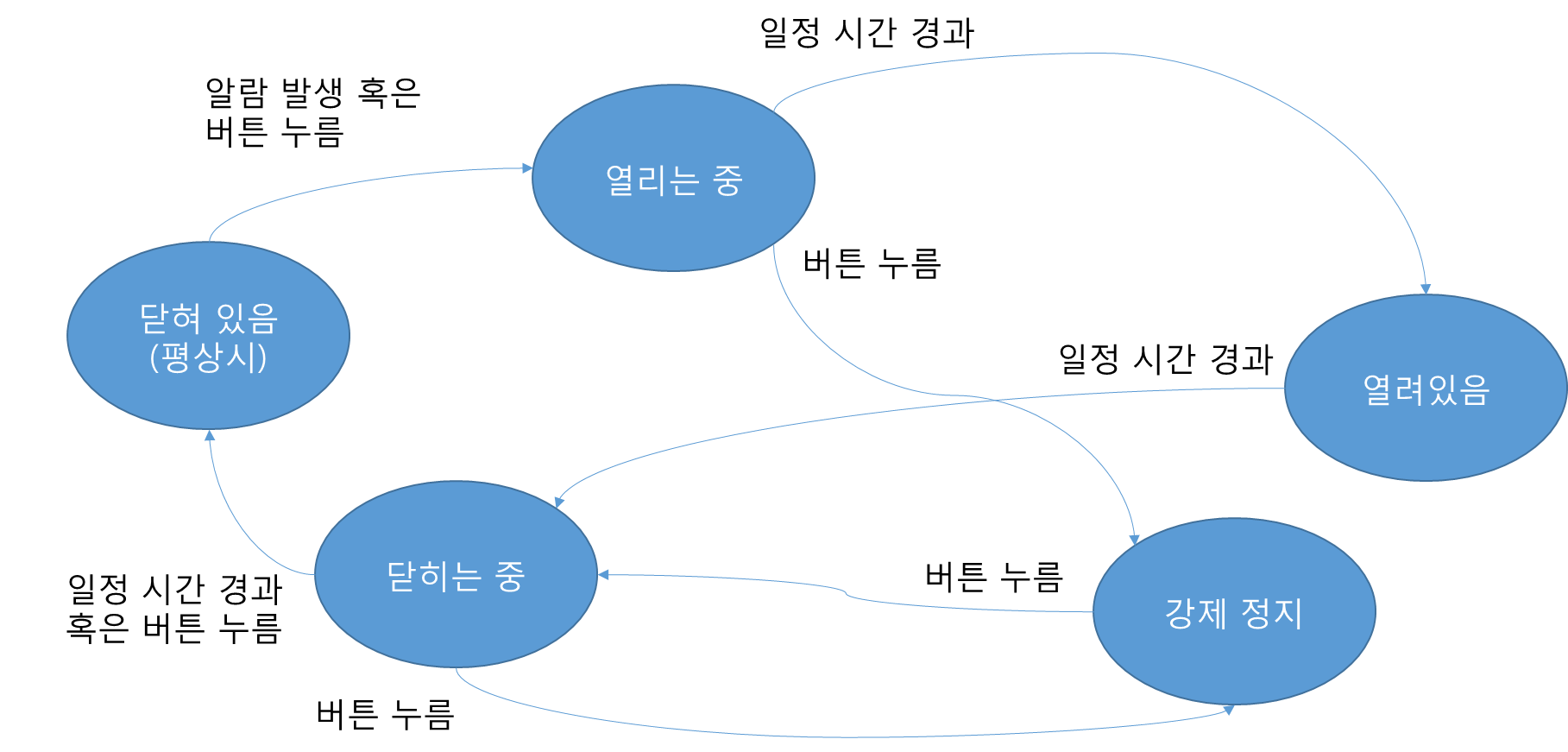
case 9 : return font\_9;

default: return font\_NULL;

}

}

8) 통이 열리고, 일정시간 지나면 닫히는 것을 구현하기 위해 다음과 같은 그림을 그리고 다음과 같이 상태의 수를 정의함.



#define DOOR\_CLOSED 0

#define DOOR\_OPENING 1

#define DOOR\_OPENED 2

#define DOOR\_CLOSING 3

// 버튼을 누를 경우 발생

#define DOOR\_STOPPED 4

// 시간재서 5초 이상이 되면 상태를 바꿈

#define OPENING\_SECOND 5

#define OPENED\_SECOND 5

#define CLOSING\_SECOND 5

9) 작성하다가 막히는 부분 있으면 다음과 같이 7)의 과정에서 만든 ShowNumber함수를 이용해서 문제를 해결함

ShowNumber(secDiff, DEBUG\_ROW, COL\_HOUR);

**3. Assembly 코드 분석**

**모든 코드를 다 분석하기에는 너무 많으므로 C언어에서의 함수 호출, 입출력 등이 어셈블리어로 어떻게 변환이 되는지를 보는 식으로 분석하였다.**

**1) 함수호출, 특정 주소의 위치로의 출력**

@0000031F: main

35: Platform\_Init(); // AVR 초기화

+0000031F: 940E05B3 CALL 0x000005B3 Call subroutine

------------

43: void Platform\_Init(void){

MCUCR = 0x80 ;

+000005B3: E890 LDI R25,0x80 Load immediate

+000005B4: BF95 OUT 0x35,R25 Out to I/O location

**Platform\_Init()함수를 호출하기 위해선 CALL명령을 쓰고, 특정 위치로 출력을 할 때는 LDI명령으로 출력할 값을 범용 레지스터에 넣고, OUT명령으로 그 레지스터의 값을 특정 위치로 출력한다.**

**2) 파라미터가 있는 함수 호출**

37: ShowTime(&CurrentTime, WATCH\_ROW);

+000004D6: EC82 LDI R24,0xC2 Load immediate

+000004D7: E092 LDI R25,0x02 Load immediate

+000004D8: E061 LDI R22,0x01 Load immediate

+000004D9: E070 LDI R23,0x00 Load immediate

+000004DA: 940E049D CALL 0x0000049D Call subroutine

**파라미터가 있는 함수를 호출할 때는 LDI명령으로 범용 레지스터에 값을 넣고 CALL명령으로 함수를 호출한다.**

**여기서는 파라미터 하나당 두 개의 LDI명령을 사용하였다.**

**이는 ATmega128이라는 프로세서가 한 번에 한 바이트만 이동시킬 수 있으므로 주소 값이나 정수와 같이 두 바이트 값을 저장하기 위해서는 두 번의 LDI명령어가 필요하다는 것을 알 수 있다. 따라서 char와 같이 한 바이트 값을 파라미터로 쓰면 LDI명령 사용을 하나씩 줄 일 수 있음을 알 수 있다.**

**3) 함수호출 후 변수선언**

@00000946: GLcdClr

30: { unsigned char i, j, x;

+00000946: 930F PUSH R16 Push register on stack

+00000947: 931F PUSH R17 Push register on stack

+00000969: 911F POP R17 Pop register from stack

+0000096A: 910F POP R16 Pop register from stack

+0000096B: 9508 RET Subroutine return

**함수 안에서 변수를 선언하면, 그 변수의 값을 저장할 레지스터를 사용해야 하므로,**

**그 레지스터의 기존 값을 스택에 저장해서 이 함수를 호출한 함수에서 사용 중이던**

**변수의 값을 보존, 그 함수로 복귀하기 전에 POP해서 함수 호출하기 이전에 사용하던**

**범용 레지스터의 값을 복원한다.**

**4) 무한루프(while(1){...})**

39: mainInLoop();

+00000324: 940E0308 CALL 0x00000308 Call subroutine

+00000326: CFFD RJMP PC-0x0002 Relative jump

**while(1){...}은 RJMP로 변환되었다. 이 코드는 while문안에 넣을 내용을 하나의 함수로 빼 놓아 간단하게 만든 것이다.**

**5) for문**

40: for(j = 0; j <= 63; j++) GLcdSetData(0xC0,0x00); // clear CS1 and CS2 screen

+0000095C: EC80 LDI R24,0xC0 Load immediate

+0000095D: E090 LDI R25,0x00 Load immediate

+0000095E: E060 LDI R22,0x00 Load immediate

+0000095F: E070 LDI R23,0x00 Load immediate

+00000960: 940E06CF CALL 0x000006CF Call subroutine

**그래픽 LCD 제어는 GLcdCmd와 GLcdSetData함수의 주소로 CALL함으로써 이루어 진다.**

+00000962: 5F1F SUBI R17,0xFF Subtract immediate

+00000963: 3410 CPI R17,0x40 Compare with immediate

+00000964: F7B9 BRNE PC-0x08 Branch if not equal

37: for(i = 0; i <= 7; i++){

+00000965: 3B0F CPI R16,0xBF Compare with immediate

+00000966: F011 BREQ PC+0x03 Branch if equal

41: x++;

+00000967: 5F0F SUBI R16,0xFF Subtract immediate

+00000968: CFEA RJMP PC-0x0015 Relative jump

**확인해본 결과 이중 for문은 조건문 2개를 사용하여 점프 명령(RJMP)를**

**한 번만 사용하여 구현되어 있다.**

**6) &연산 (and)**

7: LCD\_CONTROL = signal & 0xC0;

+000006EC: 7C80 ANDI R24,0xC0 Logical AND with immediate

+000006ED: E000 LDI R16,0x00 Load immediate

+000006EE: E614 LDI R17,0x64 Load immediate

+000006EF: 01F8 MOVW R30,R16 Copy register pair

+000006F0: 8380 STD Z+0,R24 Store indirect with displacement

**&연산은 ANDI명령으로 표현되어 있다.**

**7) 덧셈, 뻴셈, 곱셈, 인덱싱**

86: pd->hour += AlarmTimes[Index].hour – CurrentTime.hour

+ (AlarmTimes[Index].ampm - CurrentTime.ampm)\*12;

+000003C0: 01FB MOVW R30,R22 Copy register pair

+000003C1: E073 LDI R23,0x03 Load immediate

+000003C2: 0FEE LSL R30 Logical Shift Left

+000003C3: 1FFF ROL R31 Rotate Left Through Carry

+000003C4: 957A DEC R23 Decrement

+000003C5: F7E1 BRNE PC-0x03 Branch if not equal

+000003C6: 54EE SUBI R30,0x4E Subtract immediate

+000003C7: 4FFD SBCI R31,0xFD Subtract immediate with carry

+000003C8: 8140 LDD R20,Z+0 Load indirect with displacement

+000003C9: 8151 LDD R21,Z+1 Load indirect with displacement

+000003CA: 918D LD R24,X+ Load indirect and postincrement

+000003CB: 919C LD R25,X Load indirect

+000003CC: 9711 SBIW R26,0x01 Subtract immediate from word

+000003CD: 0F48 ADD R20,R24 Add without carry

+000003CE: 1F59 ADC R21,R25 Add with carry

+000003CF: 918002C2 LDS R24,0x02C2 Load direct from data space

+000003D1: 919002C3 LDS R25,0x02C3 Load direct from data space

+000003D3: 1B48 SUB R20,R24 Subtract without carry

+000003D4: 0B59 SBC R21,R25 Subtract with carry

+000003D5: 8126 LDD R18,Z+6 Load indirect with displacement

+000003D6: 8137 LDD R19,Z+7 Load indirect with displacement

+000003D7: 918002C8 LDS R24,0x02C8 Load direct from data space

+000003D9: 919002C9 LDS R25,0x02C9 Load direct from data space

+000003DB: 1B28 SUB R18,R24 Subtract without carry

+000003DC: 0B39 SBC R19,R25 Subtract with carry

+000003DD: 01C9 MOVW R24,R18 Copy register pair

+000003DE: 0F88 LSL R24 Logical Shift Left

+000003DF: 1F99 ROL R25 Rotate Left Through Carry

+000003E0: 0F82 ADD R24,R18 Add without carry

+000003E1: 1F93 ADC R25,R19 Add with carry

+000003E2: 0F88 LSL R24 Logical Shift Left

+000003E3: 1F99 ROL R25 Rotate Left Through Carry

+000003E4: 0F88 LSL R24 Logical Shift Left

+000003E5: 1F99 ROL R25 Rotate Left Through Carry

+000003E6: 0F48 ADD R20,R24 Add without carry

+000003E7: 1F59 ADC R21,R25 Add with carry

+000003E8: 9611 ADIW R26,0x01 Add immediate to word

+000003E9: 935C ST X,R21 Store indirect

+000003EA: 934E ST -X,R20 Store indirect and predecrement

**위와같이 다항식을 쓰면, 어셈블리어로 표현하기 매우 복잡해진다.**

**그리고 곱셈은 수업시간에 들었던 MUL 명령을 쓰는 것이 아닌,**

**다음과 같은 명령어로 구현되어 있음을 볼 수 있다.**

+000003E2: 0F88 LSL R24 Logical Shift Left

+000003E3: 1F99 ROL R25 Rotate Left Through Carry

**8) 키 입력 받기**

24: key\_data = KEY\_SCAN ;

+0000023F: 91800000 LDS R24,0x0000 Load direct from data space

장치로부터 입력 받을 때는 LDS명령을 쓴다.

**이 프로젝트의 거의 모든 기능은 여기서 분석해서 알아본 방식대로 구현됨을 알 수 있다.**

**4. 신호 측정 및 분석(Skip)**

**5. 필수 구현요소**

A. 타이머 인터럽트: 제공한 예제 소스를 그대로 사용함.

B. 입력: 키패드에서 SW4, 출력: 외부I/O에 연결된 스탭모터