

Z80 kit Calculator Program

Wichit Sirichote, kswichit@kmitl.ac.th

Introduction

This applications note describes the example program using c language to make a simple calculator running on the Z80 kit. The c compiler is sdcc c compiler. The display uses a 20x2 lines text LCD module. Numeric number and keys for mathematics calculations are Z80 kit keypad. The program demonstrates how to interface between c main code and low level monitor subroutines.

LCD module

Before we see the calculator program, let us learn how to use the LCD module with sdcc. The Z80 kit has provided a 16-pin socket for the text LCD module already. The LCD is interfaced to the Z80 bus directly. The LCD's registers are decoded into I/O space i.e.,

LCD register	I/O address
LCD_command_write	0x80
LCD_data_write	0x81
LCD_command_read	0x82
LCD_data_read	0x83

LCD hardware driver

To use the LCD module as a displaying device, we will need the hardware driver. The driver will initialize the operating mode, interfacing to the application program for writing the characters to the LCD memory, clear screen and set the cursor position. Full listing of the LCD driver is shown in listing of lcddriver.c.

Function	description
InitLcd(void)	Set mode LCD
clr_screen(void)	Clear screen
goto_xy(char x,char y)	Set cursor position

putc_lcd(char ch)	Write ASCII letter
LCDWriteText(char *txt)	Write string

Here is the example of printing text to LCD.

First we must set the LCD mode using function InitLcd(). Then set cursor to the 2nd line. The LCDWriteText("Z80 Kit Calculator") function will send the ASCII string to LCD display.

```
InitLcd();
goto_xy(0,1);
LCDWriteText("Z80 Kit Calculator");
```

The example is shown below.



Calling monitor subroutine

We will use Z80 kit keypad for entering numeric number and key add, subtract, multiply and divide.

The monitor subroutine that scans display and keypad is located at 0x05FE.

This subroutine will scan keypad and return internal key code for a given key that pressed.

Sdcc uses word __asm and __endasm for inserting the assembly code.

```
char key;

void scan()
{
    __asm
        push ix
        ld ix,#0x1FB6
        call 0x05FE
        ld (_key),a
        pop ix
```

```

    __endasm;
}

```

We have the global variable **key** for receiving the key code. To save the IX registers, we then push it to stack and pop it before returning.

The IX is loaded with 0x1FB6 which is the buffer memory for the onboard 7-segment display.

Calculator program

The source code of the calculator program is shown in Listing lcdcal.c.

```

main()
{
// initialize LCD module

while(1)
{
    scan();
    GPIO1 = key;
    if(key>=0 && key <10) number_execute();
    else function_exe();
}
}

```

Main function is repeating scanning keypad. When key pressed, the internal key code will be sent to GPIO1 LED for checking.

There are two functions that makes the calculator program.

The first function is for numeric keys 0-9 and the 2nd function is for function keys.

```

void number_execute()
{
    switch(state)
    {
        case 0: put_num(); break;
        case 1: put_num(); break;
    }
}

```

We will use state variable to test what services will be entered. Initially state =0, suppose, we press number 123456. The main function will call number_execute() function. The switch statement with state=0 will call put_num() function.

```

void put_num()
{
    buffer[n]=key+0x30;
    putch_lcd(buffer[n]);
    n++;
}

```

Function put_num() will read key pressed then convert it to ASCII code and write it to buffer array.

Suppose we want to add number, so we press key + (internal code = 0x10).

```

void function_exe()
{
    switch(key)
    {
        case 0x10: state=1;
            putch_lcd('+');
            compute=1;
            buffer[n]=0;
            num1= atol(buffer);
            n=0;
            break;
    }
}

```

We change state to 1, put symbol '+' on LCD, set compute mode=1 (for addition), put terminator 0 to string being received, then convert string '123456' to long number using atol(buffer).

Compute mode is set to 1 for addition.

The index n is then reset to 0 for the next number entering.

Get back to number_execute() function again when we press the 2nd number, says 789012.

Since state is now set to 1, so function put_num() will be entered again.

Let us see again what we press.

123456+789012

To get result, we will press key GO (internal code=0x12).

```

void function_exe()
{
    switch(key)
    {
        case 0x12: state=0;
        buffer[n]=0;
        num2= atol(buffer);
        n=0;
        print_answer();
        break;
    }
}

```

Similarly to key + press, the 2nd number will be converted by num2=atol(buffer) function.

With key GO, we will print the answer with function print_answer().

The compute variable =1 will call number to be added with num1=num1+num2 statement.

Then the result will print to buffer and finally on the 2nd line of the LCD using LCDWriteText(buffer).

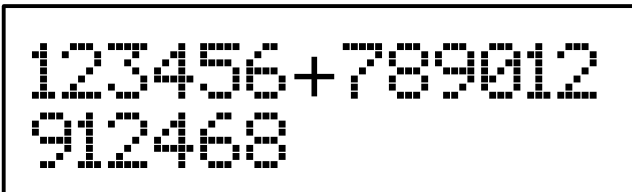
```

void print_answer()
{
    switch(compute)
    {
        case 1:
        num1=num1+num2;
        sprintf(buffer,"%ld",num1);
        goto_xy(0,1);
        LCDWriteText(buffer);
        break;
    }
}

```

Here is the example display on the LCD when we type 123456+789012 then press key GO to compute.

We get the result, 912468 on the 2nd line.



The image shows a rectangular LCD display with a black border. The text is in a pixelated font. The first line displays "123456+789012" and the second line displays "912468".

For new calculation, press key CBR (internal code=0x1A).

The LCD display will be cleared.

More calculations

We can try with subtraction with key - (internal code= 0x11).

We see the switch-case statement in the function_exe() that, the symbol '-' will put to the LCD and set compute mode to 2.

```

case 0x11:
state=1;
putch_lcd('-');
compute=2;
buffer[n]=0;
num1= atol(buffer);
n=0;
break;

```

Similarly to the addition, now with the compute mode=2 in the print_answer() function.

Result will put to num1 with num1-num2. Then print to the LCD.

```

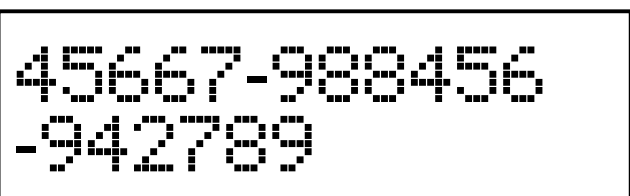
case 2:
num1=num1-num2;
sprintf(buffer,"%ld",num1);
goto_xy(0,1);
LCDWriteText(buffer);
break;

```

Suppose we type,

45667-988456 key GO

We get -942789



The image shows a rectangular LCD display with a black border. The text is in a pixelated font. The first line displays "45667-988456" and the second line displays "-942789".

For multiply, the compute mode =3.

```

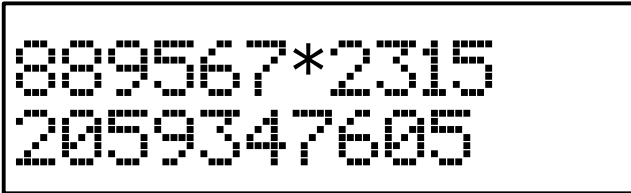
case 3:
num1=num1*num2;
sprintf(buffer,"%ld",num1);
goto_xy(0,1);
LCDWriteText(buffer);
break;

```

Let us try with 889567x2315 the press key GO.

Multiply uses key DATA (internal code = 0x14) for multiply.

Here is the result.



889567*2315
2059347605

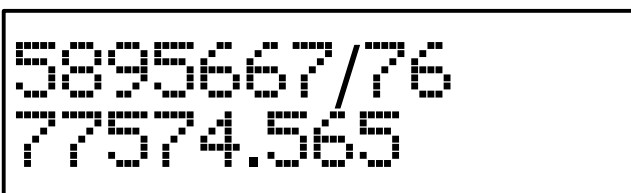
Very nice with long number variable for both num1 and num2.

And the last calculation is division.

We will find the result of division with operator / and % for the remainder.

```
void divide()
{
    num3=num1/num2;
    sprintf(buffer,"%ld",num3);
    goto_xy(0,1);LCDWriteText(buffer);

    num3=num1%num2; num3= (num3*1000)/num2;
    sprintf(buffer,".%ld",num3);
    LCDWriteText(buffer);
}
```



5895667/76
77574.565

Compiling the source code

Refer to the Application Note 001 for runtime startup code modification. We will use it for compiling our program as well. For this program, we will place the code to address 0x2000 and have the data location at 0x8000. Here is the example of MSDOS bat file for easy compilation. We can set its name to make.bat.

```
sdcc -mz80 --code-loc 0x2000 --data-loc 0x8000
--no-std-crt0 mycrt0.rel %1
```

Conclusion

The Z80 kit has CPU bus interface for the text LCD. We can use text LCD module for making a simple calculator. The c source code has two parts, LCD driver and main calculator functions. The Z80 kit provides subroutine that scan keypad, we can access the key code from c function using statement `__asm` and `__endasm` directly. The source code can be compiled with `sdcc` compiler for Z80. The hex file can be downloaded to the kit and test run easily.

Resource

1. `sdcc` c compiler, [sdcc.sourceforge.net](https://sourceforge.net/projects/sdcc/)

```
// Example of using sdcc c compiler for Z80 Microprocessor Kit
// Written by Wichit Sirichote, (C) 2015
//
// Simple calculator
// key 0-9 is for numeric number
// key + for Add
// key - for subtract
// key DATA for multiply
// key ADDR for divide
// key CBR for clear display new calculation

//#include <stdio.h>

#include <d:\sdcc\z80\lcddriver.c>
#include <stdio.h>
#include <stdlib.h>

// function prototype declaration

void LcdReady();
void goto_xy(char x,char y);
void InitLcd(void);
char *Puts(char *str);
void putch_lcd(char ch);
void LCDWriteText(char *txt);
void LCDWriteConstText(const char *txt);

int j;
long num1;
long num2;
long num3;

char key;
char state;
char n;
char compute;

long k,l;

char buffer[20];
char __at 0x1FB6 buffer1[6]; // display buffer

char buffer[20];

// test calling monitor's routine scan display and keypad directly
// scan until key press
// register A was key that pressed
// save it to variable key for c program access

void scan()
{
    __asm
        push ix
            ld ix,#0x1FB6

            call 0x05FE

            ld (_key),a

        pop ix
    __endasm;
}

void put_num()
{
    buffer[n]=key+0x30;
    putch_lcd(buffer[n]);
    n++;
}

void divide()
```

```
{
    num3=num1/num2; sprintf(buffer,"%ld",num3); goto_xy(0,1);LCDWriteText(buffer);
    num3=num1%num2; num3= (num3*1000)/num2; sprintf(buffer,"%ld",num3); LCDWriteText(buffer);
}

void print_answer()
{
    switch(compute)
    {
case 1: num1=num1+num2; sprintf(buffer,"%ld",num1); goto_xy(0,1);LCDWriteText(buffer); break;
case 2: num1=num1-num2; sprintf(buffer,"%ld",num1); goto_xy(0,1);LCDWriteText(buffer); break;
case 3: num1=num1*num2; sprintf(buffer,"%ld",num1); goto_xy(0,1);LCDWriteText(buffer); break;
case 4: divide();break;
    }
}

void number_execute()
{
    switch(state)
    {
        case 0: put_num(); break;
        case 1: put_num(); break;
    }
}

void function_exe()
{
    switch(key)
    {
case 0x10: state=1; putch_lcd('+'); compute=1; buffer[n]=0; num1= atol(buffer); n=0; break;
case 0x12: state=0; buffer[n]=0; num2= atol(buffer); n=0; print_answer(); break;
case 0x1a: InitLcd(); break;
case 0x11: state=1; putch_lcd('-'); compute=2; buffer[n]=0; num1= atol(buffer); n=0; break;
case 0x14: state=1; putch_lcd('x'); compute=3; buffer[n]=0; num1= atol(buffer); n=0; break;
case 0x19: state=1; putch_lcd('/'); compute=4; buffer[n]=0; num1= atol(buffer); n=0; break;
    }
}

void main()
{
    state=n=compute=0;
    InitLcd();
    buffer1[0]=buffer1[1]=buffer1[2]=buffer1[3]=buffer1[4]=buffer1[5]=0; // turn off display

    goto_xy(0,1);
    LCDWriteText("Z80 Kit Calculator");
    goto_xy(0,0);

    while(1)
    {
        scan();
        GPIO1 = key; // for key pressed
        if(key>=0 && key <10) number_execute();
        else function_exe();
    }
}
```

```
// LCD driver for Z80 Trainer Kit
//

__sfr __at 0x40 GPIO1;
__sfr __at 0x80 LCD_command_write;
__sfr __at 0x81 LCD_data_write;
__sfr __at 0x82 LCD_command_read;
__sfr __at 0x83 LCD_data_read;

#define BUSY 0x80

void LcdReady()
{
while(LCD_command_read&BUSY)
continue; // wait until busy flag =0
}
void clr_screen(void)
{
LcdReady();
LCD_command_write=0x01;
}

void goto_xy(char x,char y)
{
LcdReady();
switch(y){
case 0 : LCD_command_write=0x80+x; break;
case 1 : LCD_command_write=0xC0+x; break;
case 2 : LCD_command_write=0x94+x; break;
case 3 : LCD_command_write=0xd4+x; break;
}
}
void InitLcd(void)
{
LcdReady();
LCD_command_write=0x38;
LcdReady();
LCD_command_write=0x0c;
clr_screen();
goto_xy(0,0);
}
char *Puts(char *str)
{
unsigned char i;
for (i=0; str[i] != '\0'; i++){
LcdReady();
LCD_data_write=str[i];
}
return str;
}
void putchar_lcd(char ch)
{
LcdReady();
LCD_data_write=ch;
}

void LCDWriteText(char *txt) {
while(*txt)
putchar_lcd(*txt++);
}

void LCDWriteConstText(const char *txt) {
while(*txt)
putchar_lcd(*txt++);
}
```