Lab 6 Memories and Index Registers
Subroutines with Result shown on LCD Display

Submit numDisplay.asm at the end of your lab, not your project.

I. AVR ATmega2560 Memories

Program storage: Flash memory (.cseg directive)
Data storage: SRAM and EEPRROM (.dseg and .eseg directives)

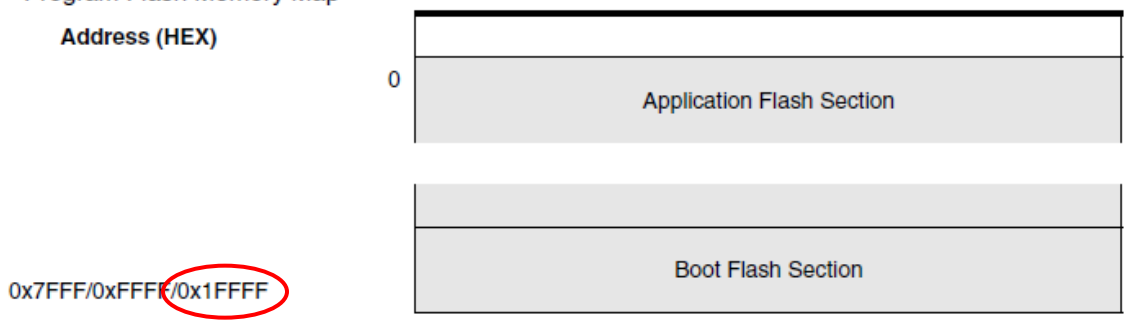
The Configuration Summary of ATmega2560:[1]

Table 2-1. Configuration Summary

Device	Flash	EEPROM	RAM	General Purpose I/O pins	16 bits resolution PWM channels	Serial USARTs	ADC Channels
ATmega2560	256KB	4KB	8KB	86	12	4	16

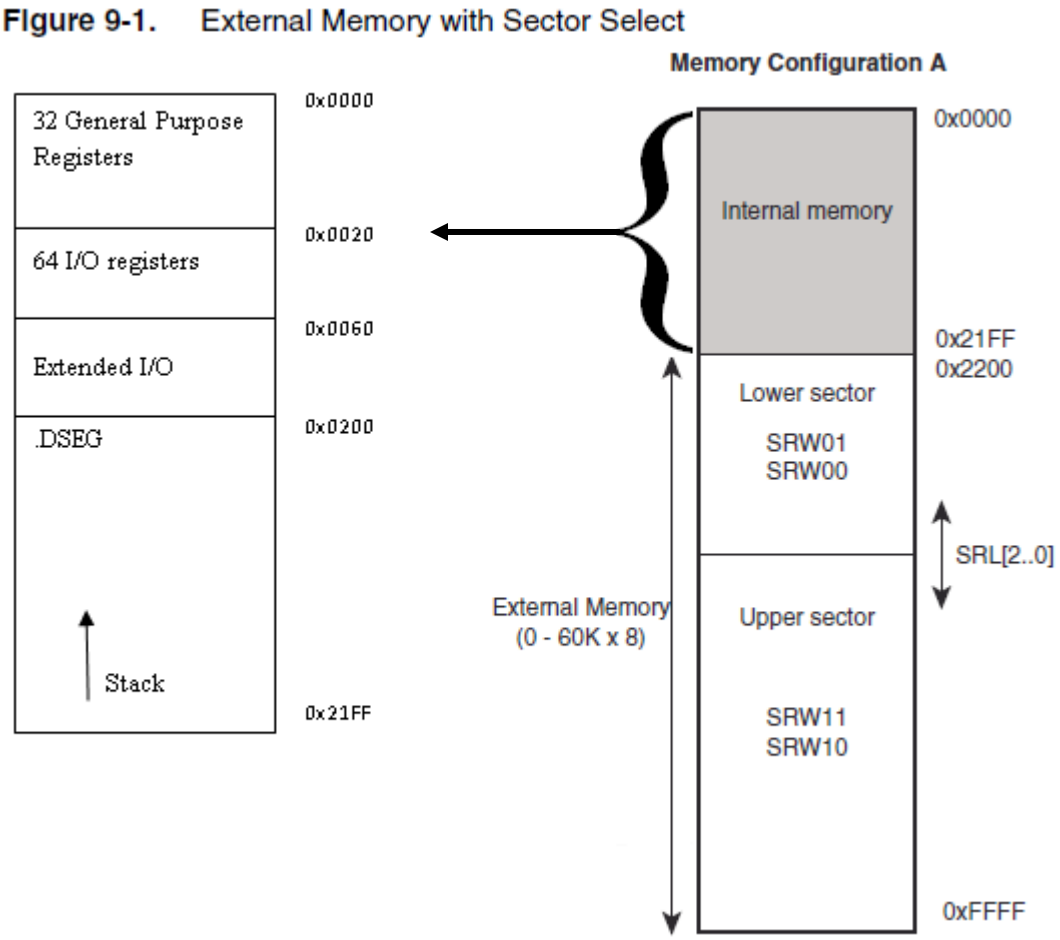
The diagram of the program flash memory[1]:

Figure 8-1. Program Flash Memory Map



What is the largest flash memory address of ATmega2560?
Based on the above table, the memory size is 256KB -> 2^{18}Bytes ->2^{17}words, the largest word address is 0x1FFFF(17bits).

The diagram of the SRAM[1]:



II. Index Registers

Data Direct Addressing:

We practiced the following instruction in the previous labs:

lds R0, msg
sts msg, R0

In general, the instruction takes the form of "lds Rd, k", where the value of k is a 16-bit unsigned integer representing memory address of the SRAM (data memory). The content (1 byte) stored in memory address k is loaded to register Rd.

Data Indirect Addressing:

The AVR processor has three register pairs that can be used for data indirect addressing. The three register pairs (also called index registers) are:

X -> R27:R26 or XH :XL
Y -> R29:R28 or YH :YL
Z -> R31:R30 or ZH :ZL

The address to be accessed must be preloaded into either X, Y, or Z register. What do the following statements do?

LD Rd, X
ST X, Rr
LPM R23, Z+

Indirect addressing is especially suited for accessing arrays, tables, and Stack Pointer.

III. Download lab6_exercise.asm and finish the program as an Exercise. You don't need to submit this.

A C-style string (C string) is stored in the program memory (flash memory). Write a short program to calculate the length of the string and copy the string from the program memory (flash) to the data memory (SRAM) in **reverse order**.

This lab is derived from Chapters 5 and 12 of your textbook (Some Assembly Required by Timothy S. Margush) and the datasheet of ATMEGA 2560 at http://www.atmel.com/images/doc2549.pdf.

1. The diagrams are copied and modified from the datasheet mentioned above.

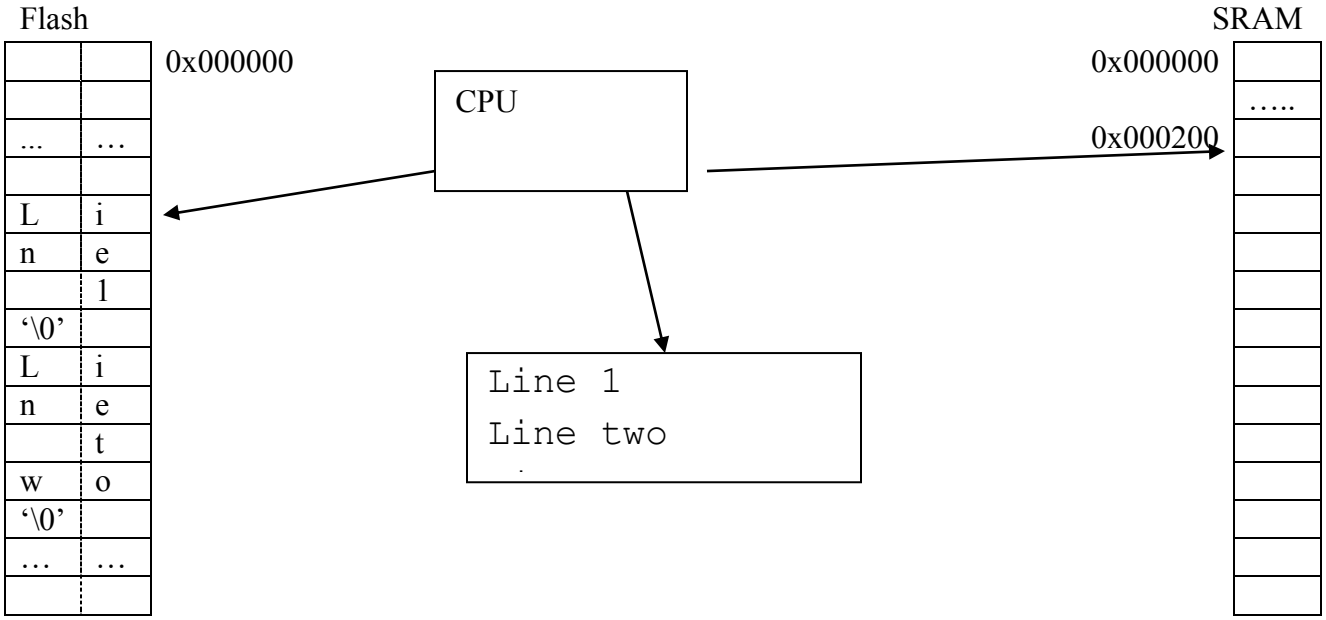
Submit numDisplay.asm at the end of your lab.

IV. Subroutine continues

We wrote a subroutine (strlength) in the previous lab. To be more accurate, we should modify the diagram in the previous lab (see the red characters) as the following:

Address	content	details	notes
0x0000 ~ 0x01FF		General Purpose Registers and I/O Registers	
0x0200			.DSEG
	……..		
		<- Z and SP	
	r1	saved register	Programmer pushes those registers onto the stack in the subroutine
	r0	saved register	
	r31	saved register	
	r30	saved register	
	ret	return address	Assembler pushes the return address onto the stack automatically when "call do_something" is executed.
	ret	return address	
	ret	return address	
	0xCC	parameter (Z + 8)	Programmer pushes the values onto the stack in the **location where a subroutine is called, it could be in the main, or another subroutine.**
0x21FF	0xEE	parameter (Z + 9)	

Download **LCDdefs.inc, lcd.asm,** and **lab6.asm**. HD44780 LCD Driver for ATmega2560 is written in **LDCdefs.inc** and **lcd.asm**. Jason Corless wrote **lab6.asm** to demonstrate how do display strings on LCD.



Note that **str_init** is defined in **lcd.asm**. It copies a string from the program memory (flash) to the data memory (SRAM), like what we did in lab 6. "SP_OFFSET" is defined in **LCDdefs.inc** (line 42 and 43), it is 3 in this case.

Open **lab6.asm**. Understand the code.
In the "display_strings" subroutine, the algorithm is:

Clear the LCD display (**lcd_clr**)
Move the cursor to the desired location (row 0, column 0) (2X16 display) (**lcd_gotoxy**)
Display "Line 1" stored in SDRAM (**lcd_puts**)
Move the cursor to the desired location (row 1, column 0) (2X16 display) (**lcd_gotoxy**)
Display "Line two" stored in SDRAM (**lcd_puts**)

II. Exercises: download numDisplay.asm, Finish implementing display_num subroutine.
The C equivalent code is:

```c
/*division, using repeated subtractions*/
int main()
{
    int dividend=123;
    int quotient=0;
    int divisor=10;
    char num[4];
    num[3]='\0';
    int i=2;
    do{
        while(dividend>=divisor)
        {
            quotient++;
            dividend -= divisor;
        }
            num[i--]=dividend+'0';
            dividend=quotient;
            quotient=0;
    }while(dividend>=divisor);
    num[i]=dividend+'0';
    printf ("%s\n",num);
    return 0;
}
```

To accomplish similar task in assembly language, we need to convert each digit to a character, e.g. int 1 to character '1' and store integer 123 as a string '1' '2' '3' in SRAM. It is required to manipulate memory addresses, and be responsible for parameter passing using stack frame.

Submit numDisplay.asm at the end of your lab.