Lab 5 Memories and Index Registers

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]

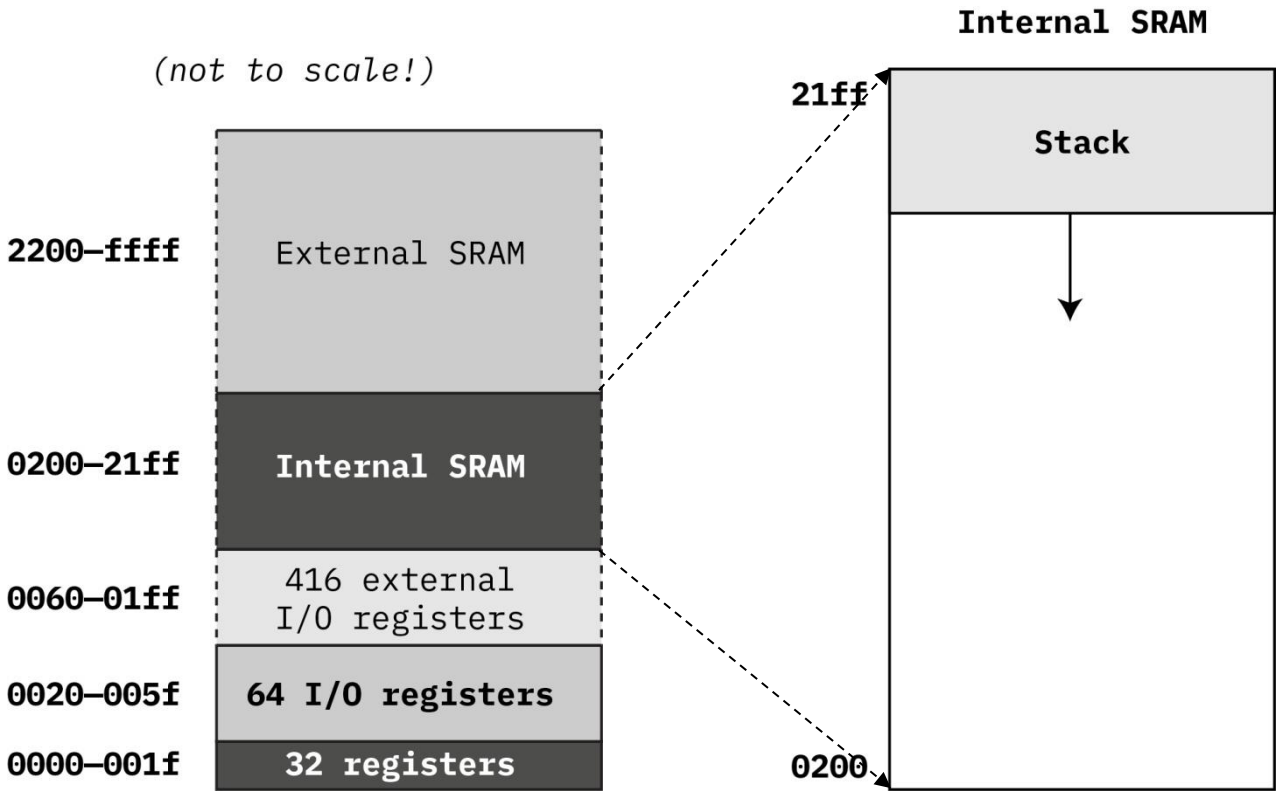
Table 2-1. Configuration Summary

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

What is the largest flash memory address of ATmega2560?

Based on Table 2-1, the memory size is 256KB -> 2^{18}Bytes ->2^{17}words, the largest word address is 0x1FFFF(17bits).

The diagram of the data memory (SRAM): [2]



II. Index Registers

Data Direct Addressing:

We practiced the following instructions in assignment 1:

lds R16, IN1 ;(reverse.asm, load the content stored at memory address 0x200, labeled as IN1, to R16)

sts DISTANCE, r1;(in hamming_distance.asm, save the value in r1 to memory address 0x202, labeled as DISTANCE)

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 the memory address k is loaded to register Rd. Turn to pages 95 and 150 of the AVR Instruction Set Manual (The manual is located at the course website: from the "Content" menu, go to "Lab material-> "AVR_Instruction_Set.pdf"), understand the instructions.

Look at the following example:

Open the reverse.asm of your assignment 1, at the beginning of the file, you see this:

```
ldi R16, 0xA1
sts IN1, R16
ldi R16, 0x74
sts IN2, R16
```

At the end of the file, you see this:

```
    .dseg
    .org 0x200
IN1:    .byte 1
IN2:    .byte 1
OUT1:   .byte 1
OUT2:   .byte 1
```

In the instruction "`sts IN1, R16`", "IN1" is a label for memory address 0x0200 (two bytes), the value 0xA1 (1 byte) is stored at memory address 0x0200. After execution of the whole program, the data memory (SRAM) should look like this:

Label of the memory address	Memory address	Value stored at that address
IN1	0x0200	0xA1
IN2	0x0201	0x74
OUT1	0x0202	0x2E
OUT2	0x0203	0x85
	…….	

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. Refer to lab5.asm for examples on how to load memory addresses from flash/data memory to registers. The following examples assume data memory address is loaded to index register X and flash memory address is loaded to index register Z already:

LD Rd, X ; load one byte pointed to by the X-register into Rd
ST X, Rr ; store one byte in Rr to memory pointed to by the X-register
LPM R23, Z+ ; load one byte pointed to by the Z-register into R23, then increment the address in Z by one.

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

Note: need to load the memory address into index registers X, Y, or Z before using them. For example,

```
lds R16, IN1 ;(reverse.asm, load the content stored at memory
address 0x200, labeled as IN1, to R16)
```
<−Data Direct Addressing

Can be written using Data Indirect Addressing as:

```
ldi XH, high(IN1) ;(load high byte of memory IN1, which is 0x02,
to XH)
ldi XL, low(IN1) ;(load low byte of memory IN1, which is 0x00, to
XL)
ld R16, X ;(load one byte pointed to by register X into R16)
```

III. Download lab5.asm and finish the program.

A C-style string (C string - the last byte of the string is 0x00) is stored in the program memory (flash memory). Write a short program to calculate the length of a string and store the length to the data memory (SRAM). To be specific, the subroutine/function will be labelled "str_length", that the main assembly code will call this subroutine to compute the string length, and finally it is the main routine that saves the length in data memory. The suggested structure of your program is:

---------- main program ----------
Get the memory address of msg1 to srcH:srcL – the register pair to store string address
Call subroutine str_length and store the returned value in register named n
Save n to data memory location named LENGTH1

Get the memory address of msg2 to srcH:srcL – the register pair to store string address
Call subroutine str_length and store the returned value in register named n
Save n to data memory location named LENGTH2

Get the memory address of msg3 to srcH:srcL – the register pair to store string address
Call subroutine str_length and store the returned value in register named n
Save n to data memory location named LENGTH3

Initialize msg1 in the program memory
Initialize msg2 in the program memory
Initialize msg3 in the program memory

Allocate one byte in data memory to store string length of msg1
Allocate one byte in data memory to store string length of msg2
Allocate one byte in data memory to store string length of msg3

---------- end of the main program ----------

---------- subroutine str_length ----------
Copy srcH to ZH, the high byte of the index register Z
Copy srcL to ZL, the low byte of the index register Z
Initialize a register (counter) to store the number of characters
Write a loop, load one character at a time from the program memory
Increment counter
If all 8 bits of the character are 0, end of the loop
Else, back to the loop
---------- end of subroutine str_length ----------

Hint for implementing the subroutine:

1. The memory address of the source string is passed to register pair srcH:srcL, in order to load each character from the program memory to register temp, you need to use lpm (page 97 of the Manual) command, therefore, you need to transfer the memory address from srcH:srcL to ZH:ZL, for example, mov ZL, srcL.
2. Use "LPM Rd, Z+" so that Z will point to the next character after execution of the statement.
3. Check register temp, if it is 0, then that is the end of the string. Need to use a register to count the number of characters in the string.

1. The diagram is copied and modified from page 7 of the datasheet of ATMEGA 2560: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf.
2. The diagrams are provided by Dr. Mike Zastre.