LAB 5

Q1. Explain the purpose/function of each of these files.

.hex: This file format contains the hexadecimal representation of the machine code that will be loaded into the microcontroller's flash memory.

.lss: This file format contains the assembly listing of the program, including the original source code and the corresponding machine code.
.map: This file format contains a detailed map of the program's memory usage, including the memory locations of each variable and function.
.obj: This file format contains the object code generated by the compiler. It contains the machine code instructions and information about the memory locations of the program's variables and functions. This file is used by the linker to generate the final executable code.

Q2 If you look at the .lss file you will see your assembly code at the end of the file. To the left of the assembly code you will see 2 columns of hexadecimal numbers. What information is being expressed in those 2 columns?

The two columns of hexadecimal numbers that you see in the .lss file next to the assembly code are the memory address and the machine code representation of the assembly instruction at that address. The left column shows the memory address where the instruction is located.

This address corresponds to the location in the microcontroller's memory where the instruction will be stored after it is programmed. The right column shows the machine code representation of the assembly instruction at that address. The machine code is a sequence of binary digits that the microcontroller can understand and execute. Each assembly instruction is translated into a specific machine code instruction, and this translation is typically performed by an assembler program. The hexadecimal representation of the machine code is more human-readable than the binary representation.

Q3 Also in the .lss file you will find multiple rows of Assembler Directives. What is the purposes of these directives and how did this information get added to the file?

the Assembler Directives in the .lss file provide information to the assembler about how the assembly code should be translated into machine code and how the resulting code should be organized in memory. They are added to the file by the assembler during the assembly process.

Q4 How many cycles does it take to run the 2 instruction in this program? Is there any way we could determine the number of cycles a program takes without having to simulate it running like this?

The number of cycles required to execute an instruction depends on the specific microcontroller architecture and clock speed being used.

Q5. Notice the Memory panel in the bottom right corner of the IDE display. You can view things such as program memory, ram, eeprom etc. here. From viewing program memory deduce the following

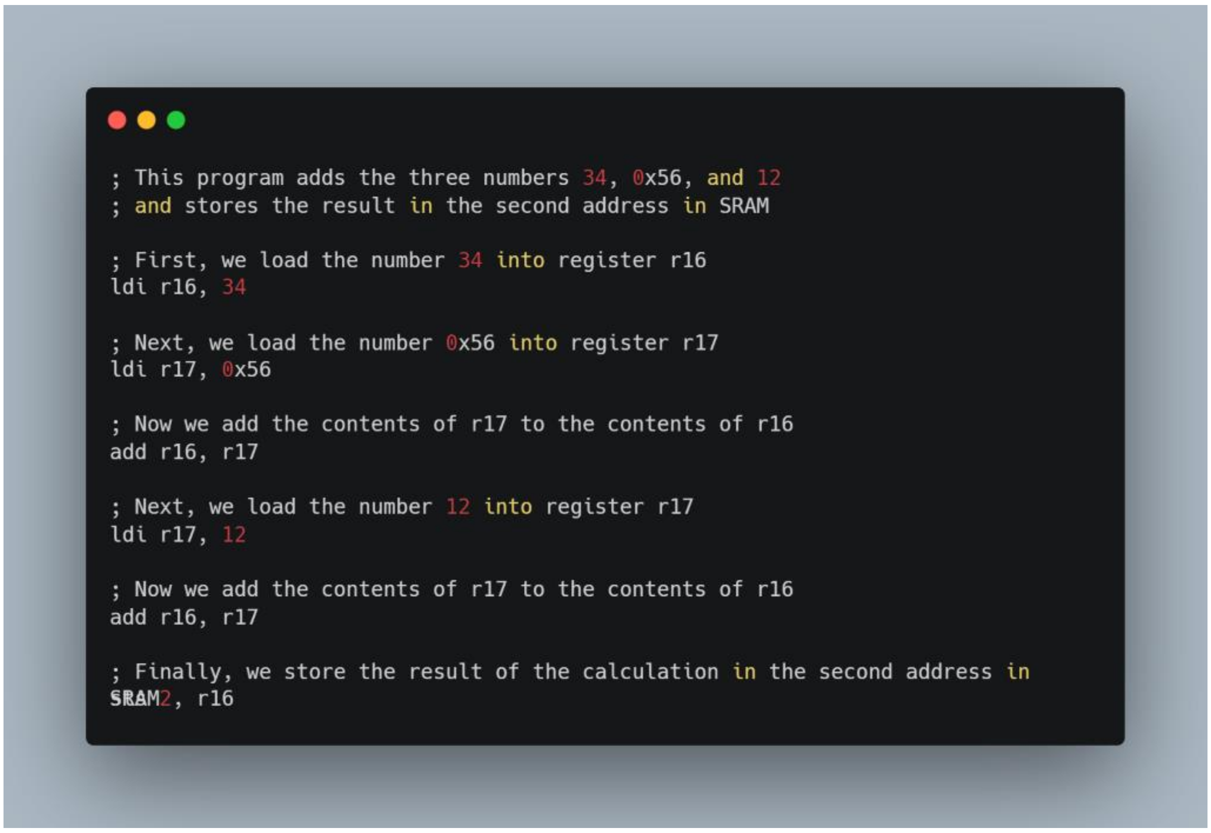
(I) The amount of program ROM (read-only memory) available on the ATmega328p is 32KB (kilobytes) or 32,768 bytes. This means that the microcontroller can store up to 32,768 bytes of machine code instructions in its program memory.

(ii) The ATmega328p, as well as most microcontrollers, stores bytes in a Little Endian manner. In a Little Endian system, the least significant byte (LSB) of the data is stored at the lowest memory address. The next byte is stored at the next memory address, and so on, up to the most significant byte (MSB) which is stored at the highest memory address.

Programming Tasks

Task 1

Write an AVR assembly program to add the three numbers 34, 0x56, 12. Store the result of the calculation in the second address in SRAM. All lines of your code should be verbosely commented. Assemble and Run the program to validate its operation. Record your assembly code in the notebook and give appropriate screenshots to show that your program ran correctly.

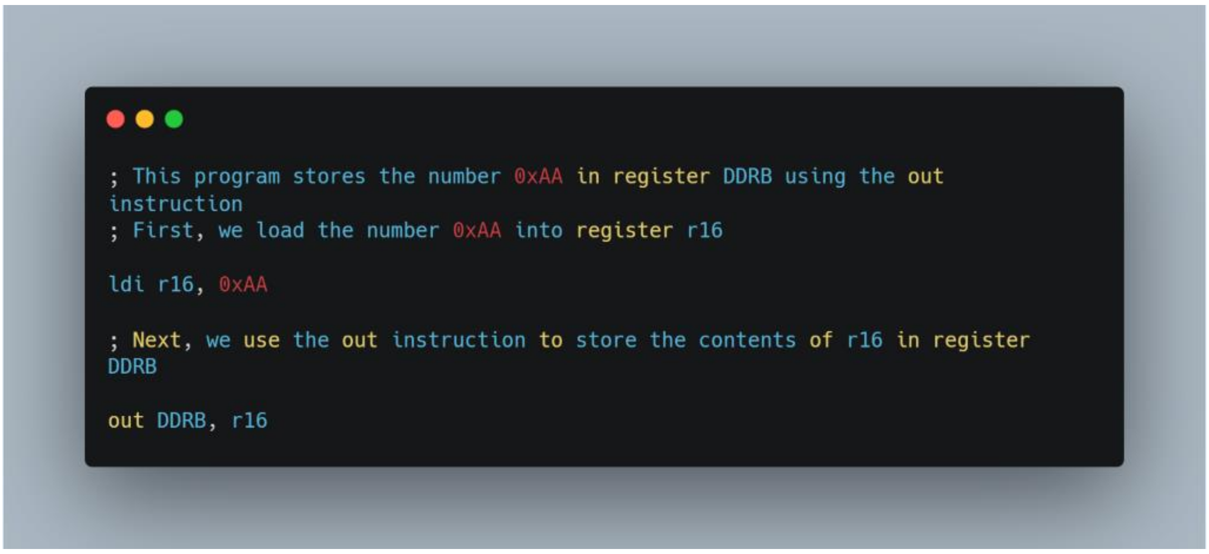


```
; This program adds the three numbers 34, 0x56, and 12
; and stores the result in the second address in SRAM

; First, we load the number 34 into register r16
ldi r16, 34

; Next, we load the number 0x56 into register r17
ldi r17, 0x56

; Now we add the contents of r17 to the contents of r16
add r16, r17

; Next, we load the number 12 into register r17
ldi r17, 12

; Now we add the contents of r17 to the contents of r16
add r16, r17

; Finally, we store the result of the calculation in the second address in
SRAM2, r16
```

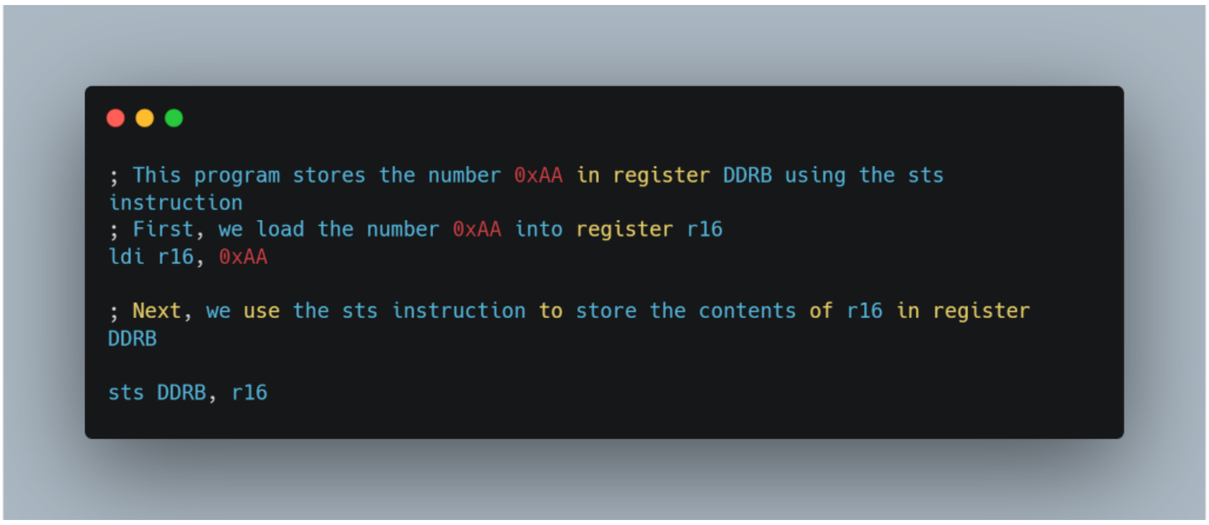
TASK 2

Write an AVR assembly program to store the number 0xAA in register DDRB.

1. Using the **out** assembly instruction.
2. Using the **sts** assembly instruction.



```
; This program stores the number 0xAA in register DDRB using the out
instruction
; First, we load the number 0xAA into register r16

ldi r16, 0xAA

; Next, we use the out instruction to store the contents of r16 in register
DDRB

out DDRB, r16
```

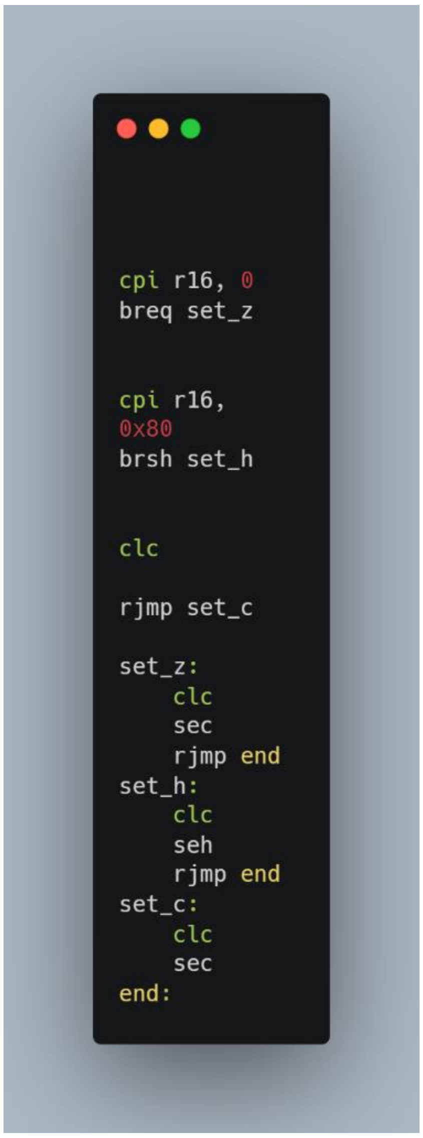


```
; This program stores the number 0xAA in register DDRB using the sts
instruction
; First, we load the number 0xAA into register r16
ldi r16, 0xAA

; Next, we use the sts instruction to store the contents of r16 in register
DDRB

sts DDRB, r16
```

Task 3:

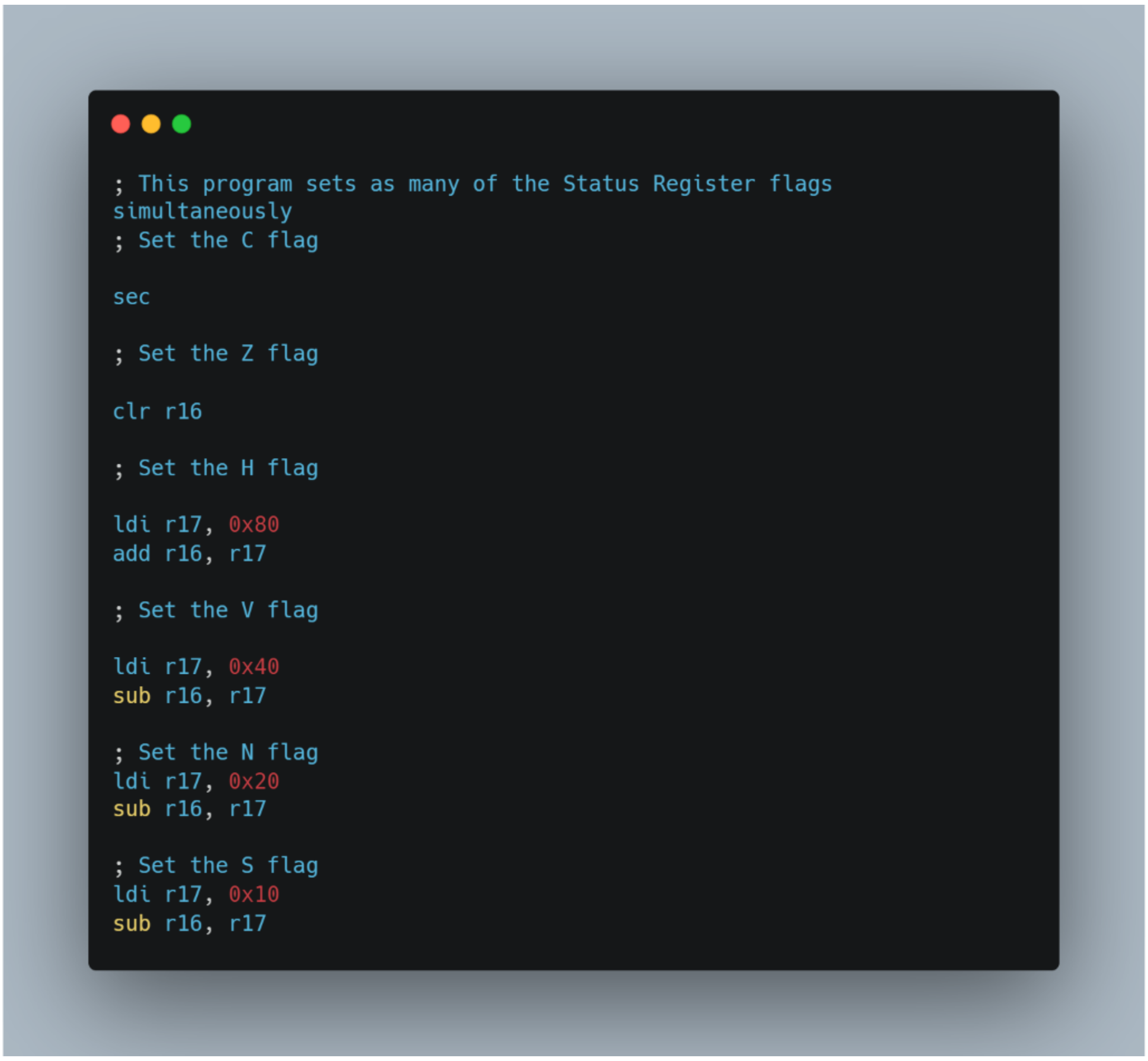
Write an AVR assembly program to set the following flags of the Status Register, in each case only 1 Flag can be set at at time. In each case explain why a particular flag was set.

1. The Z flag
2. The H flag
3. The C flag



```
cpi r16, 0
breq set_z


cpi r16,
0x80
brsh set_h


clc

rjmp set_c

set_z:
    clc
    sec
    rjmp end
set_h:
    clc
    seh
    rjmp end
set_c:
    clc
    sec
end:
```

TASK 4:
Write an AVR assembly program to set as many of the Status Register flags, C,Z,H,V,N,S as possible simultaneously. Explain why each of the flags is set. All lines of your code should be verbosely commented. Record your assembly code in the notebook and give appropriate screenshots to show that your program ran correctly.



```
; This program sets as many of the Status Register flags
simultaneously
; Set the C flag

sec

; Set the Z flag

clr r16

; Set the H flag

ldi r17, 0x80
add r16, r17

; Set the V flag

ldi r17, 0x40
sub r16, r17

; Set the N flag
ldi r17, 0x20
sub r16, r17

; Set the S flag
ldi r17, 0x10
sub r16, r17
```