**Sri Venkateswara College**

(University Of Delhi)

# Practical File

Name – Yash Nigam

Course – BSC (Hons.) Electronics

College Roll No – 1622011

Examination Roll No – 22079558006

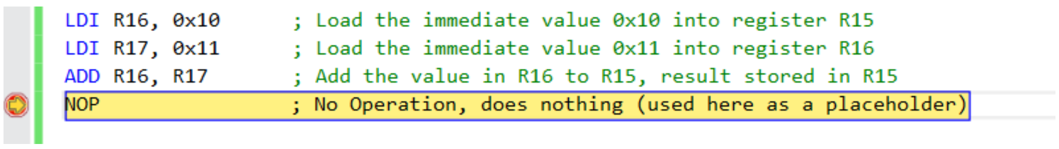
DSC – Electromagnetics

Submitted to – Dr. Sunita Jain (Ma'am)

# Index

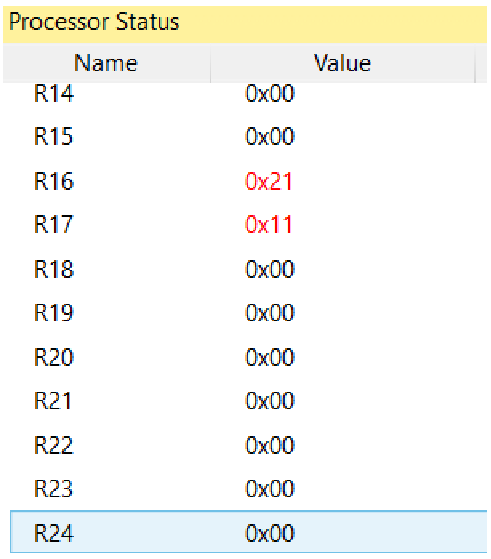
## Q1. Addition of Two 8-bit Numbers

**Code –**



```
LDI R16, 0x10       ; Load the immediate value 0x10 into register R15
LDI R17, 0x11       ; Load the immediate value 0x11 into register R16
ADD R16, R17        ; Add the value in R16 to R15, result stored in R15
NOP                 ; No Operation, does nothing (used here as a placeholder)
```

**Output –**

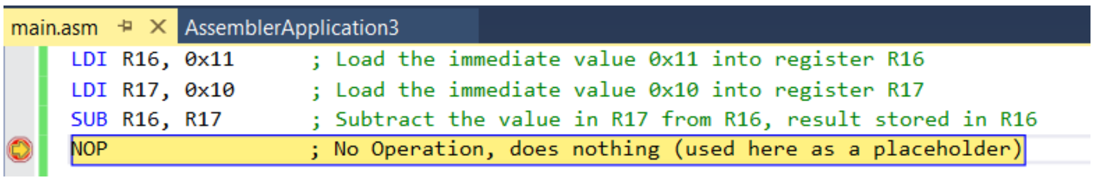


Processor Status

| Name | Value |
|---|---|
| R14 | 0x00 |
| R15 | 0x00 |
| R16 | 0x21 |
| R17 | 0x11 |
| R18 | 0x00 |
| R19 | 0x00 |
| R20 | 0x00 |
| R21 | 0x00 |
| R22 | 0x00 |
| R23 | 0x00 |
| R24 | 0x00 |

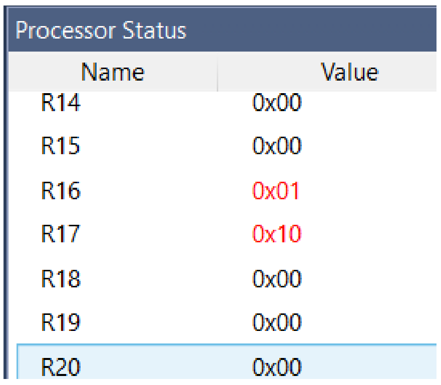
## Q2. Subtraction of Two 8-bit Numbers

**Code –**



```
LDI R16, 0x11       ; Load the immediate value 0x11 into register R16
LDI R17, 0x10       ; Load the immediate value 0x10 into register R17
SUB R16, R17        ; Subtract the value in R17 from R16, result stored in R16
NOP                 ; No Operation, does nothing (used here as a placeholder)
```

**Output –**

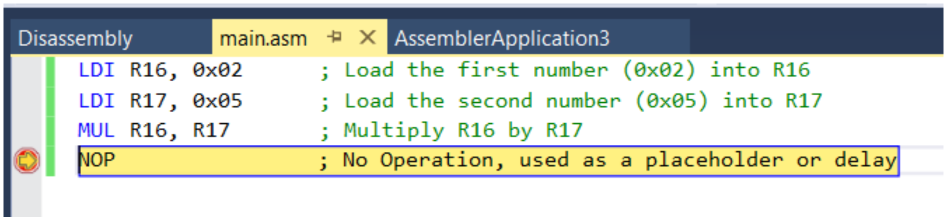


Processor Status

| Name | Value |
|---|---|
| R14 | 0x00 |
| R15 | 0x00 |
| R16 | 0x01 |
| R17 | 0x10 |
| R18 | 0x00 |
| R19 | 0x00 |
| R20 | 0x00 |

**Q3. Multiplication of Two 8-bit Numbers.**

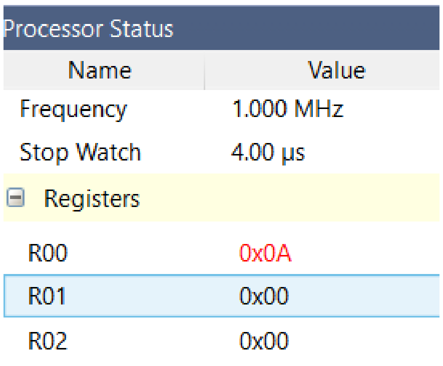
**Code –**



Disassembly | main.asm | AssemblerApplication3

```
LDI R16, 0x02      ; Load the first number (0x02) into R16
LDI R17, 0x05      ; Load the second number (0x05) into R17
MUL R16, R17       ; Multiply R16 by R17
NOP                ; No Operation, used as a placeholder or delay
```

**Output –**

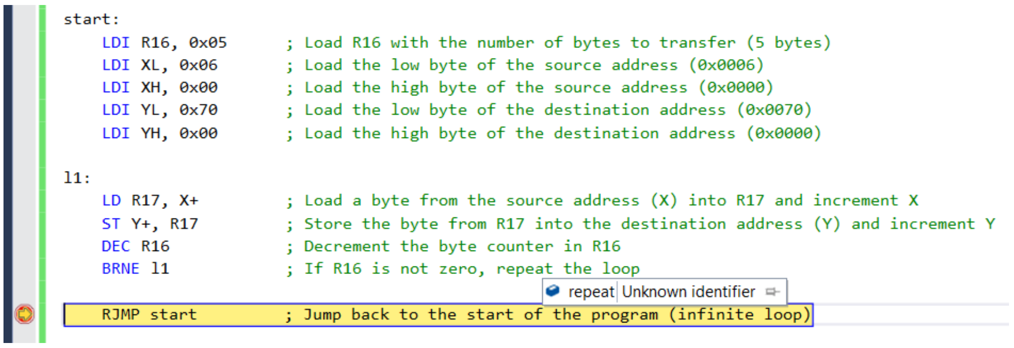


Processor Status

| Name | Value |
|---|---|
| Frequency | 1.000 MHz |
| Stop Watch | 4.00 µs |
| Registers | |
| R00 | 0x0A |
| R01 | 0x00 |
| R02 | 0x00 |

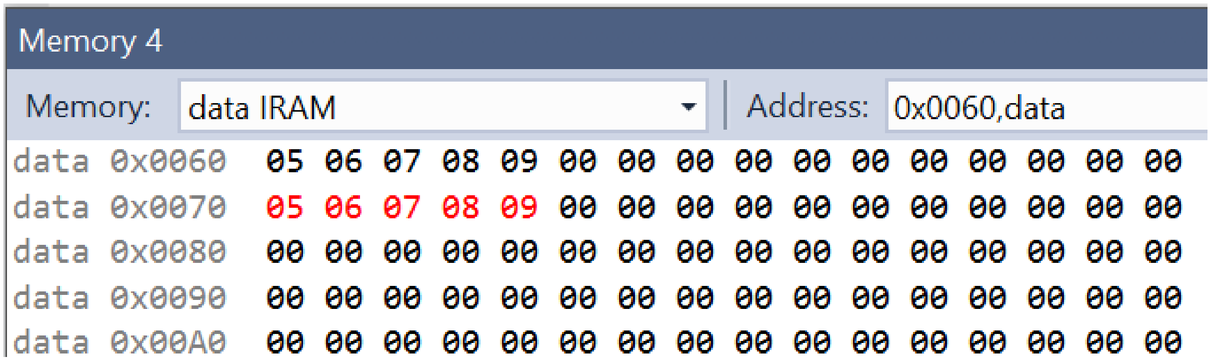
**Q4. Block Transfer of Data**

**Code –**



```
start:
    LDI R16, 0x05       ; Load R16 with the number of bytes to transfer (5 bytes)
    LDI XL, 0x06        ; Load the low byte of the source address (0x0006)
    LDI XH, 0x00        ; Load the high byte of the source address (0x0000)
    LDI YL, 0x70        ; Load the low byte of the destination address (0x0070)
    LDI YH, 0x00        ; Load the high byte of the destination address (0x0000)

l1:
    LD R17, X+          ; Load a byte from the source address (X) into R17 and increment X
    ST Y+, R17          ; Store the byte from R17 into the destination address (Y) and increment Y
    DEC R16             ; Decrement the byte counter in R16
    BRNE l1             ; If R16 is not zero, repeat the loop
                                                   repeat Unknown identifier
    RJMP start          ; Jump back to the start of the program (infinite loop)
```

**Output –**



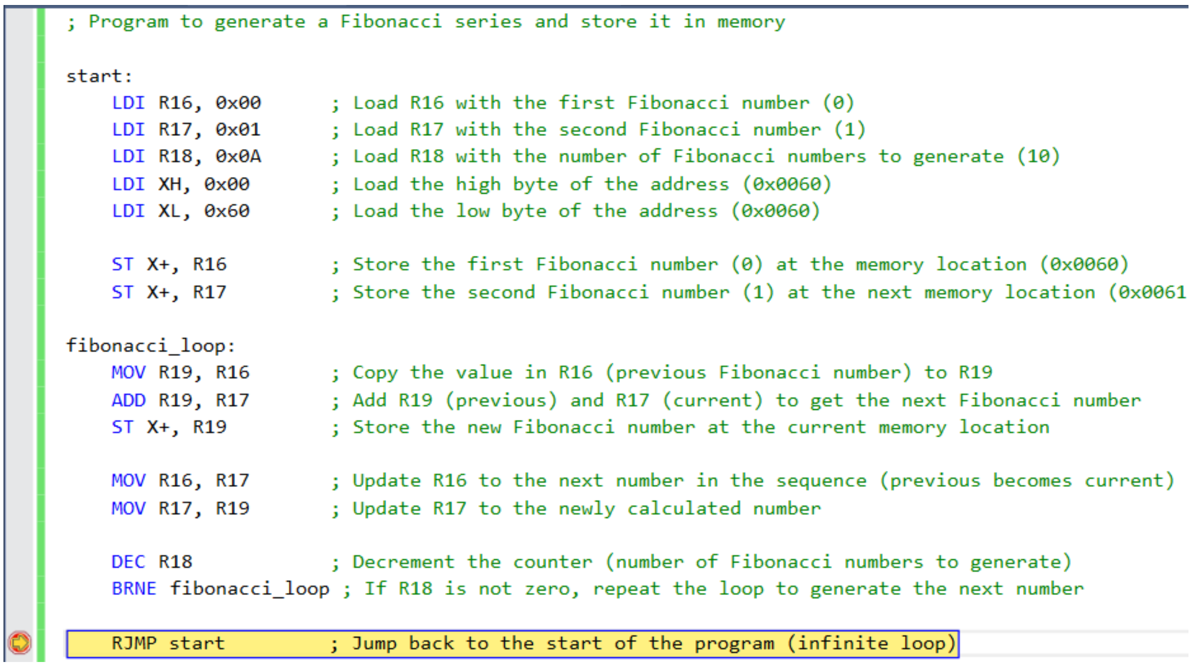
Memory 4

Memory: data IRAM    Address: 0x0060,data

```
data 0x0060  05 06 07 08 09 00 00 00 00 00 00 00 00 00 00 00
data 0x0070  05 06 07 08 09 00 00 00 00 00 00 00 00 00 00 00
data 0x0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data 0x00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

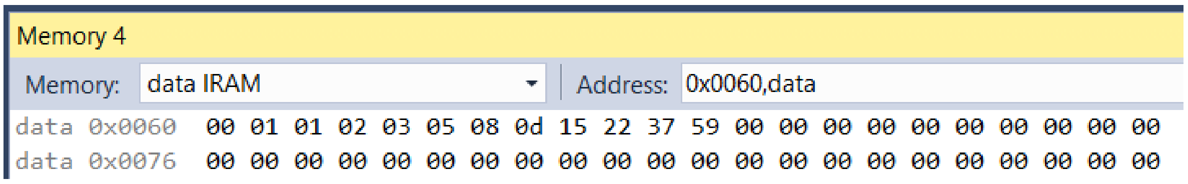
## Q5. Generation of Fibonacci Series

**Code –**



```
; Program to generate a Fibonacci series and store it in memory

start:
    LDI R16, 0x00       ; Load R16 with the first Fibonacci number (0)
    LDI R17, 0x01       ; Load R17 with the second Fibonacci number (1)
    LDI R18, 0x0A       ; Load R18 with the number of Fibonacci numbers to generate (10)
    LDI XH, 0x00        ; Load the high byte of the address (0x0060)
    LDI XL, 0x60        ; Load the low byte of the address (0x0060)

    ST X+, R16          ; Store the first Fibonacci number (0) at the memory location (0x0060)
    ST X+, R17          ; Store the second Fibonacci number (1) at the next memory location (0x0061

fibonacci_loop:
    MOV R19, R16        ; Copy the value in R16 (previous Fibonacci number) to R19
    ADD R19, R17        ; Add R19 (previous) and R17 (current) to get the next Fibonacci number
    ST X+, R19          ; Store the new Fibonacci number at the current memory location

    MOV R16, R17        ; Update R16 to the next number in the sequence (previous becomes current)
    MOV R17, R19        ; Update R17 to the newly calculated number

    DEC R18             ; Decrement the counter (number of Fibonacci numbers to generate)
    BRNE fibonacci_loop ; If R18 is not zero, repeat the loop to generate the next number

    RJMP start          ; Jump back to the start of the program (infinite loop)
```

**Output –**



Memory 4

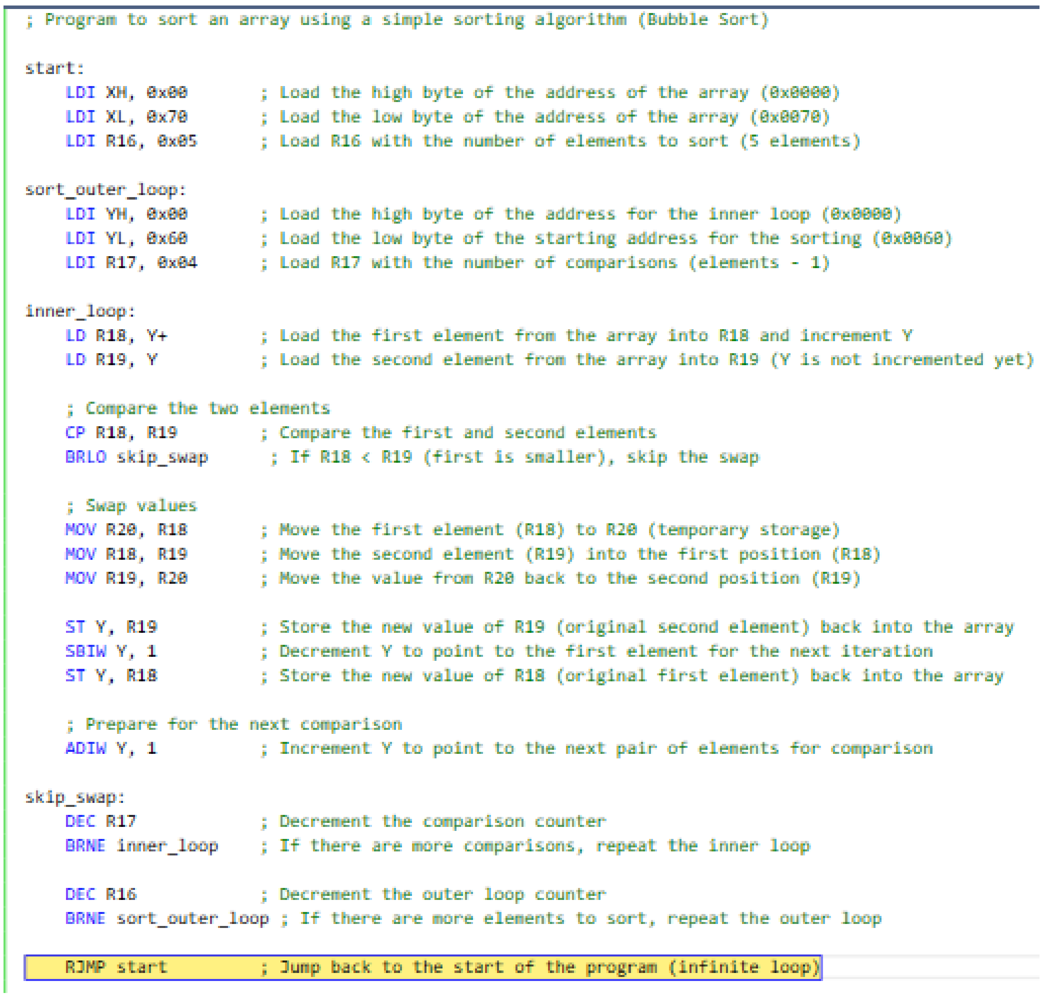
Memory: data IRAM

Address: 0x0060,data

```
data 0x0060  00 01 01 02 03 05 08 0d 15 22 37 59 00 00 00 00 00 00 00 00 00 00
data 0x0076  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

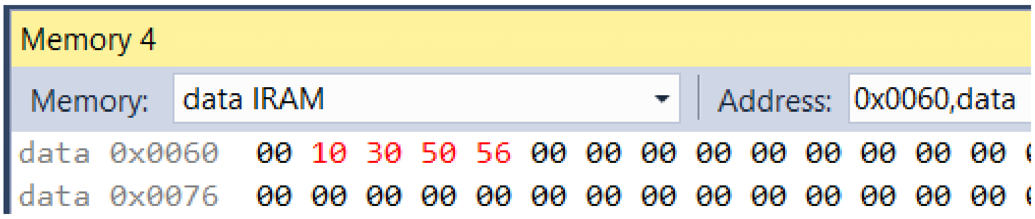
**Q6. Ascending/Descending Order.**

**Code –**



```
; Program to sort an array using a simple sorting algorithm (Bubble Sort)

start:
    LDI XH, 0x00        ; Load the high byte of the address of the array (0x0000)
    LDI XL, 0x70        ; Load the low byte of the address of the array (0x0070)
    LDI R16, 0x05       ; Load R16 with the number of elements to sort (5 elements)

sort_outer_loop:
    LDI YH, 0x00        ; Load the high byte of the address for the inner loop (0x0000)
    LDI YL, 0x60        ; Load the low byte of the starting address for the sorting (0x0060)
    LDI R17, 0x04       ; Load R17 with the number of comparisons (elements - 1)

inner_loop:
    LD R18, Y+          ; Load the first element from the array into R18 and increment Y
    LD R19, Y           ; Load the second element from the array into R19 (Y is not incremented yet)

    ; Compare the two elements
    CP R18, R19         ; Compare the first and second elements
    BRLO skip_swap       ; If R18 < R19 (first is smaller), skip the swap

    ; Swap values
    MOV R20, R18        ; Move the first element (R18) to R20 (temporary storage)
    MOV R18, R19        ; Move the second element (R19) into the first position (R18)
    MOV R19, R20        ; Move the value from R20 back to the second position (R19)

    ST Y, R19           ; Store the new value of R19 (original second element) back into the array
    SBIW Y, 1           ; Decrement Y to point to the first element for the next iteration
    ST Y, R18           ; Store the new value of R18 (original first element) back into the array

    ; Prepare for the next comparison
    ADIW Y, 1           ; Increment Y to point to the next pair of elements for comparison

skip_swap:
    DEC R17             ; Decrement the comparison counter
    BRNE inner_loop     ; If there are more comparisons, repeat the inner loop

    DEC R16             ; Decrement the outer loop counter
    BRNE sort_outer_loop ; If there are more elements to sort, repeat the outer loop

    RJMP start          ; Jump back to the start of the program (infinite loop)
```

**Output –**



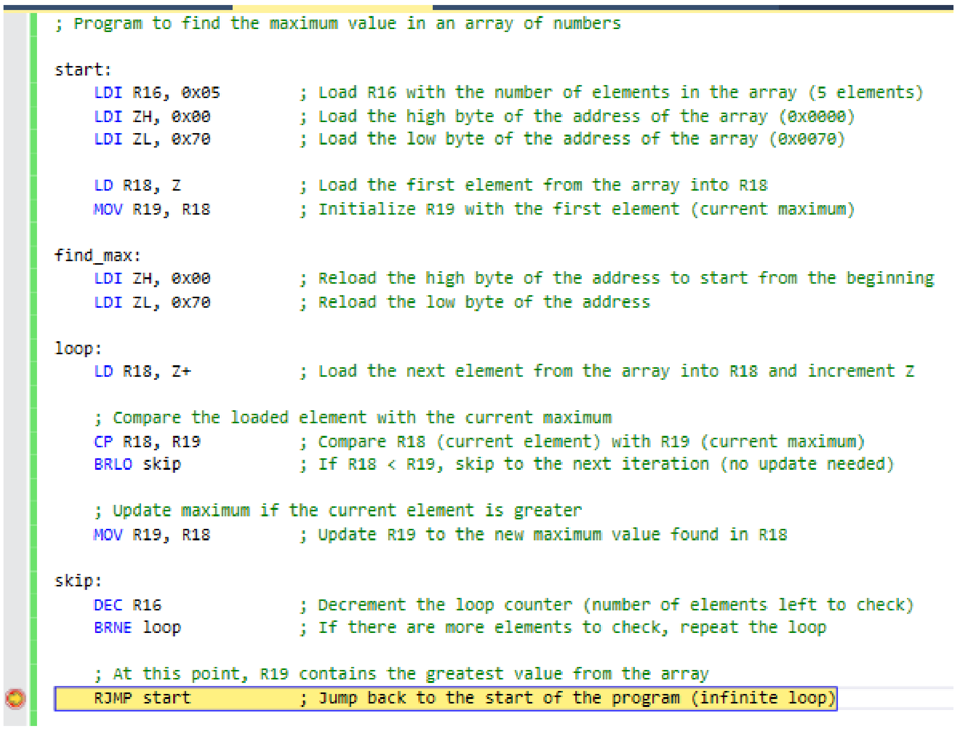
Memory 4

Memory: data IRAM  Address: 0x0060,data

```
data 0x0060  00 10 30 50 56 00 00 00 00 00 00 00 00 00
data 0x0076  00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

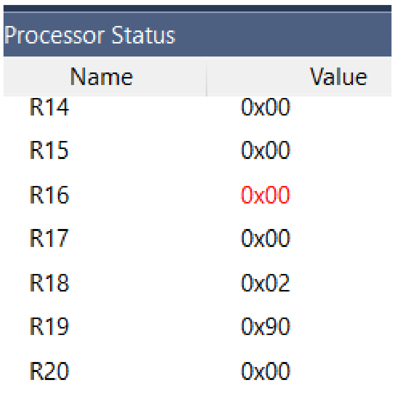
## Q7. Searching Minima/ Maxima from a given string.

### Code –



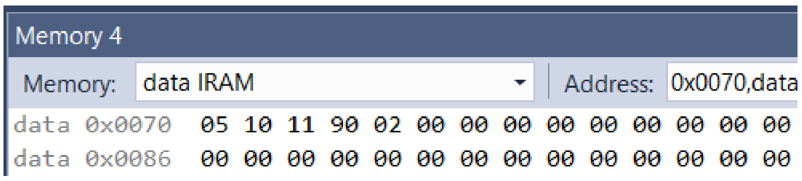
```
; Program to find the maximum value in an array of numbers

start:
    LDI R16, 0x05        ; Load R16 with the number of elements in the array (5 elements)
    LDI ZH, 0x00         ; Load the high byte of the address of the array (0x0000)
    LDI ZL, 0x70         ; Load the low byte of the address of the array (0x0070)

    LD R18, Z            ; Load the first element from the array into R18
    MOV R19, R18         ; Initialize R19 with the first element (current maximum)

find_max:
    LDI ZH, 0x00         ; Reload the high byte of the address to start from the beginning
    LDI ZL, 0x70         ; Reload the low byte of the address

loop:
    LD R18, Z+           ; Load the next element from the array into R18 and increment Z

    ; Compare the loaded element with the current maximum
    CP R18, R19          ; Compare R18 (current element) with R19 (current maximum)
    BRLO skip            ; If R18 < R19, skip to the next iteration (no update needed)

    ; Update maximum if the current element is greater
    MOV R19, R18         ; Update R19 to the new maximum value found in R18

skip:
    DEC R16              ; Decrement the loop counter (number of elements left to check)
    BRNE loop            ; If there are more elements to check, repeat the loop

    ; At this point, R19 contains the greatest value from the array
    RJMP start           ; Jump back to the start of the program (infinite loop)
```

### Output –



Processor Status

| Name | Value |
|---|---|
| R14 | 0x00 |
| R15 | 0x00 |
| R16 | 0x00 |
| R17 | 0x00 |
| R18 | 0x02 |
| R19 | 0x90 |
| R20 | 0x00 |



Memory 4

Memory: data IRAM    Address: 0x0070,data

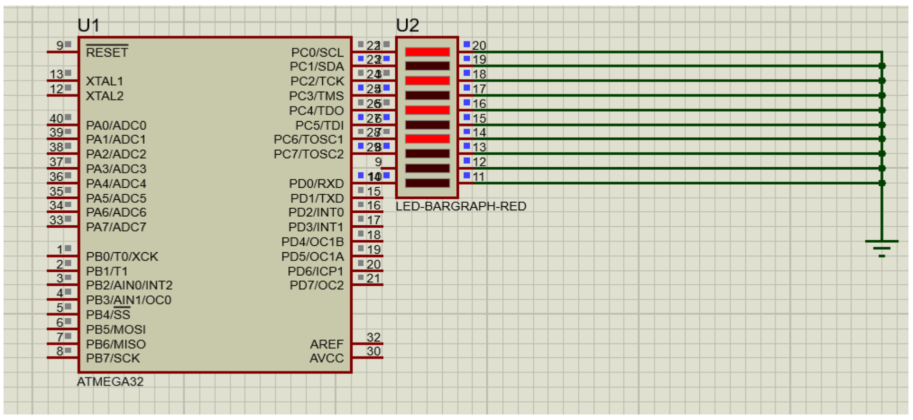
```
data 0x0070  05 10 11 90 02 00 00 00 00 00 00 00 00 00
data 0x0086  00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

**Q8. Toggle all 8 leds connected with a port.**

**Code –**

```
#include <avr/io.h> // Include the AVR I/O library for register definitions
int main(void)
{
   // Set PORTB pin 1 (PB1) as input and all other PORTB pins as output
   DDRB = DDRB & 0B11111101; // Clear the second bit (PB1) to configure it as input
   // Set PORTC pin 7 (PC7) as output
   DDRC = DDRC | 0B10000000; // Set the highest bit (PC7) to configure it as output
   /* Main application loop */
   while (1)
   {
      // Check if the second bit (PB1) of PINB is high (button pressed)
      if (PINB & 0b00000010)
      {
         // If PB1 is high, set PC7 high (turn on the output)
         PORTC = PORTC | 0B10000000; // Set PC7 to high (1)
      }
      else
      {
         // If PB1 is low (button not pressed), set PC7 low (turn off the output)
         PORTC = PORTC & 0b01111111; // Clear PC7 to low (0)
      }
   }
   return 0; // This return statement is never reached, as the while loop is infinite
}
```

**Output –**

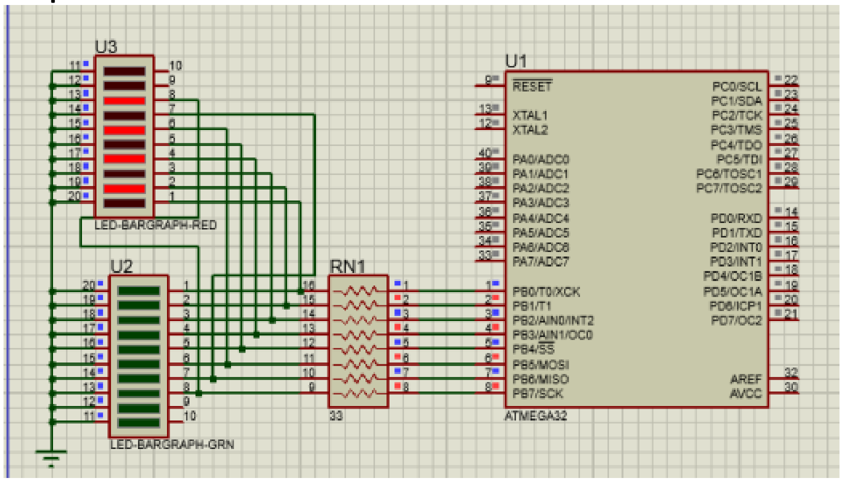


**Q9. Toggle All 8 LEDs.**

**Code –**

```
#include <avr/io.h>      // Include the AVR I/O library for register definitions
#include <util/delay.h> // Include the delay library for the _delay_ms function
int main(void)
{
   // Configure all pins of PORTB as output
   DDRB = 0xFF; // Set Data Direction Register B to all 1s (0xFF), making PORTB pins outputs
   // Main application loop
   while (1)
   {
      PORTB = 0xAA; // Set PORTB to 0xAA (10101010 in binary)
      _delay_ms(1000);   // Wait for 1000 milliseconds (1 second)

      PORTB = 0x55;    // Set PORTB to 0x55 (01010101 in binary)
      _delay_ms(1000);   // Wait for 1000 milliseconds (1 second)
   }
   return 0; // This return statement is never reached due to the infinite loop
}
```

**Output –**



U3
LED-BARGRAPH-RED
U2
LED-BARGRAPH-GRN
RN1
U1
ATMEGA32
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC