**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Arbab Latif  **Dated:** 14/02/2022

**Semester:** 4th **Section:** BEE 12C

**EE-222: Microprocessor Systems**

Lab 2: Memory, Arithmetic and Logical Operations

Group Members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Reg. No** | **Report**  **Marks / 10** | **Viva**  **Marks / 5** | **Total**  **Marks / 15** |
| Danial Ahmad | 331388 |  |  |  |
| Muhammad Umer | 345834 |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Experiment

# Memory, Arithmetic and Logical Operations

## Objectives

1. Appropriately use assembler directives in assembly code
2. Implement basic arithmetic and logical operations in assembly language
3. Differentiate, interpret and implement signed and unsigned arithmetic operations
4. Critically analyze status register for insights in arithmetic manipulations
5. Understand data memory of ATmega16
6. Use data memory to store and load data

## Equipment

Software

* *Atmel Studio*

## Introduction

 Atmel Studio IDE is a free development environment for programming Atmel MCUs, sourced by Microchip Technology Inc. It provides us with the means to simulate assembly language codes on specific Microcontrollers and provides an easy and intuitive way of producing .HEX files, which are what makes burning the code on the hardware possible.

In this specific lab, we familiarize ourselves with different operations available to the ATmega16A microcontroller. To polish our assembly language programming skills, we construct and build five distinct codes accomplishing a certain task. As a byproduct of assembling such programs, we also are able to understand registers better and the passage of data to and from these memory elements.

We also looked at the effect of decrementing beyond 0x00 on the registers as well as functions that store their outputs in the lower half (0 – 15) of General Purpose Registers. Double Dabble algorithm was also implemented through assembly language which essentially converts a binary number into its BCD equivalent. Since we are limited to 8 – bits, we restricted our domain of such numbers to be within 0 to 99..

# Lab Tasks

## Task A

Consult the Datasheet for ATmega16A and jot down the initial and final addresses of each address space in your report.

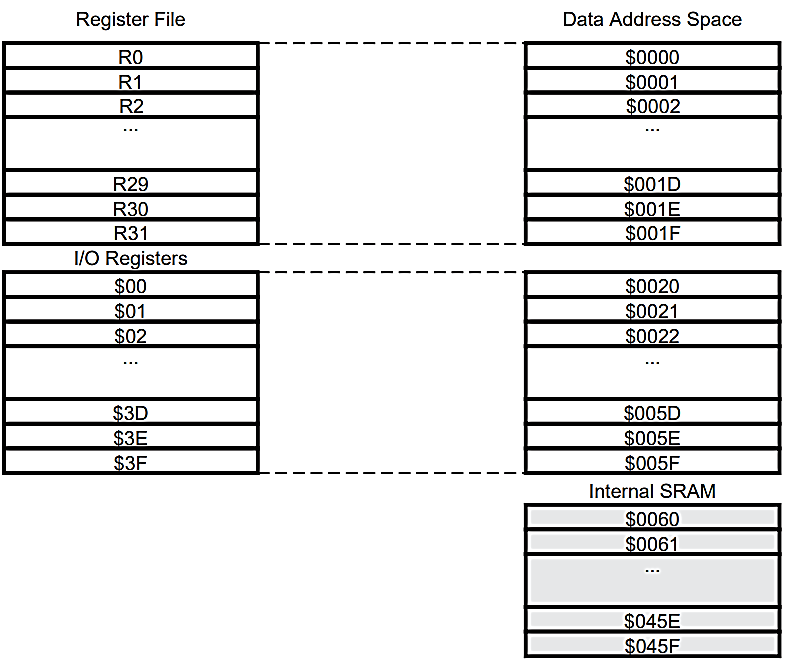


Figure 1 Data Memory Map

|  |  |  |
| --- | --- | --- |
| **Memory** | **Initial Address** | **Final Address** |
| **Program Memory** | $0000 | $1FFF |
| **Data Memory: Register File** | $0000 | $001F |
| **I/O Registers** | $0020 | $005F |
| **Internal SRAM** | $0060 | $045F |

Table 1 Initial and Final Addresses

## Task B

Complete the code given below such that it subtract 1, eight times from R20. Observe Status register on each decrement and record your readings on each instruction in the table. In the first row of table write the value (1/0) of CSR.

Write Yes/No in the second (alteration) row if the value of flag changes from previous one and jot down the reason for change in the third row.



## Code

**;**

**; Lab2.asm**

**;**

**; Created: 16/02/2022 8:59:56 pm**

**; Author : Umer and Danial**

**;**

**.ORG 0x00 ; Origin**

**LDI R20, 4**

**LDI R16, 8 ; Down Counter Tracker**

**Decrement:**

**DEC R20**

**DEC R16**

**BRNE Decrement ; Jumps to 12 if Z flag is set to zero**

**End:**

**RJMP End**

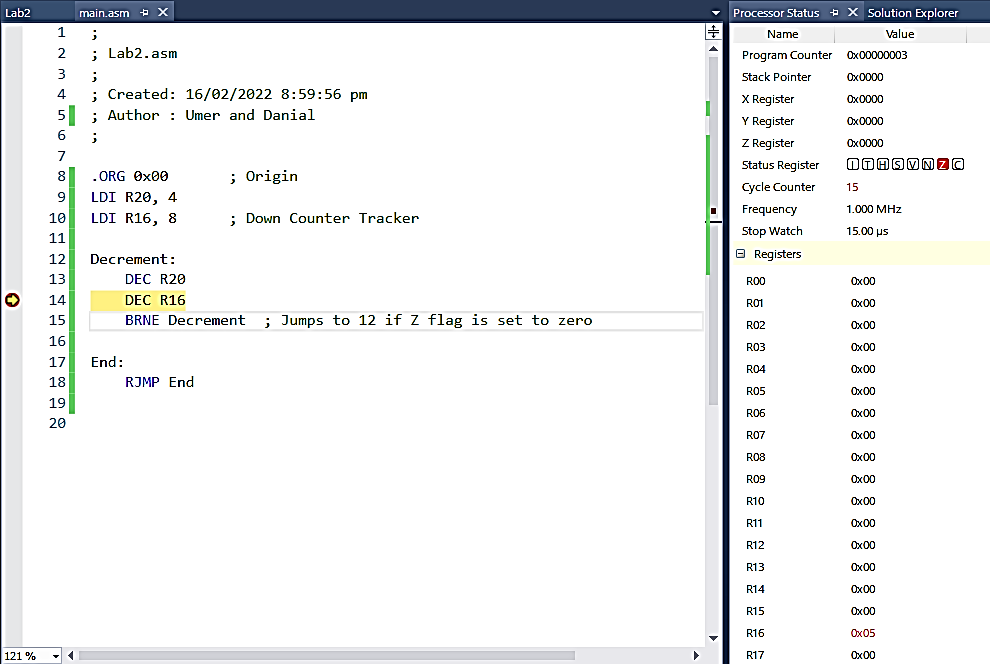


Figure 2 Exemplar Output

* Decimal value 8 is stored in GPR R16 which functions as a counter for the loop ‘DecrementLoop’. The loop executes 8 times, each time subtracting 1 from the value stored in R20. BRNE uses the Z flag and branches every time R16 is not equal to 0.

# Decrement Beyond 0x00

**Iteration 1**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

**Iteration 2**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

**Iteration 3**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

**Iteration 4**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Alterations | No | No | No | No | No | No | Yes | No |
| Reason for alterations |  |  |  |  |  |  | Result of R20 is 0x00 |  |

**Iteration 5**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Alterations | No | No | No | Yes | No | Yes | Yes | No |
| Reason for alterations |  |  |  | Negative value has a sign bit ‘1’ |  | Value in register R20 is negative | R20 has a non-zero value. |  |

**Iteration 6**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

**Iteration 7**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

**Iteration 8**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

## Task C

Multiply 0x011 and 0x0AB.

1. Considering both of them unsigned: **mul**.
2. Considering both of them signed: **muls**.
3. Considering 0x011 signed and 0x0AB unsigned: **mulsu**.

Record and analyze the output in the table.

**;**

**; Lab2.asm**

**;**

**; Created: 16/02/2022 8:59:56 pm**

**; Author : Umer and Danial**

**;**

**.ORG 0x00**

**LDI R16, 0x11**

**LDI R17, 0xAB**

**MULSU R16, R17 ; Swap MUL with MULS, MULSU for all O/P**

**OUT PORTA, R0 ; To record binary output**

**OUT PORTB, R1**

**NOP ; No Operation**

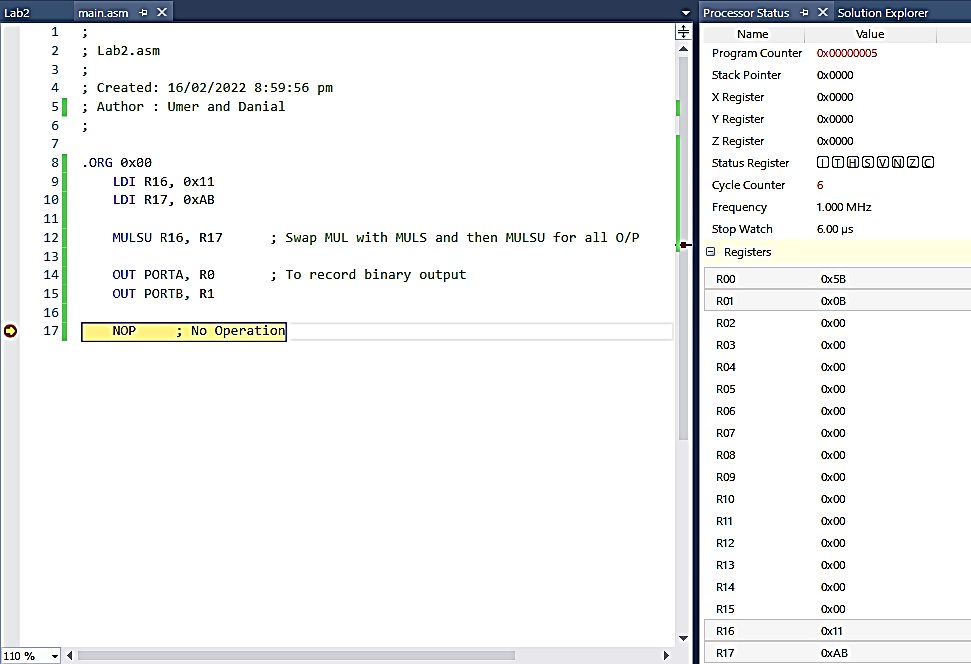


Figure 3 Exemplar Output

**MUL** is swapped with **MULS** (for both signed) and then **MULSU** (for one signed and one unsigned)

# Multiply 0x011 & 0x0AB

## mul 0x011, 0x0AB

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First Operand | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| Hexa-Decimal Value | **0x011** | | | **Decimal Value** | **17** | | | |
| Second Operand | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| Hexa-Decimal Value | **0x0AB** | | | **Decimal Value** | **171** | | | |
| R1 | Bit **7** | Bit **6** | Bit **5** | Bit **4** | Bit **3** | Bit **2** | Bit **1** | Bit **0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| R0 | Bit **7** | Bit **6** | Bit **5** | Bit **4** | Bit **3** | Bit **2** | Bit **1** | Bit **0** |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| Hexa-Decimal Value | **0X0B5B** | | | **Decimal Value** | **2907** | | | |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

## muls 0x011, 0x0AB

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First Operand | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| Hexa-Decimal Value | **0x011** | | | **Decimal Value** | **17** | | | |
| Second Operand | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| Hexa-Decimal Value | **0X0AB** | | | **Decimal Value** | **-85** | | | |
| R1 | Bit **7** | Bit **6** | Bit **5** | Bit **4** | Bit **3** | Bit **2** | Bit **1** | Bit **0** |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| R0 | Bit **7** | Bit **6** | Bit **5** | Bit **4** | Bit **3** | Bit **2** | Bit **1** | Bit **0** |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| Hexa-Decimal Value | **0XFA5B** | | | **Decimal Value** | **-1445** | | | |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Alterations | No | No | No | No | No | No | No | **Yes** |
| Reason for alterations |  |  |  |  |  |  |  | **Bit 15 of result is High** |

## mulsu 0x011, 0x0AB

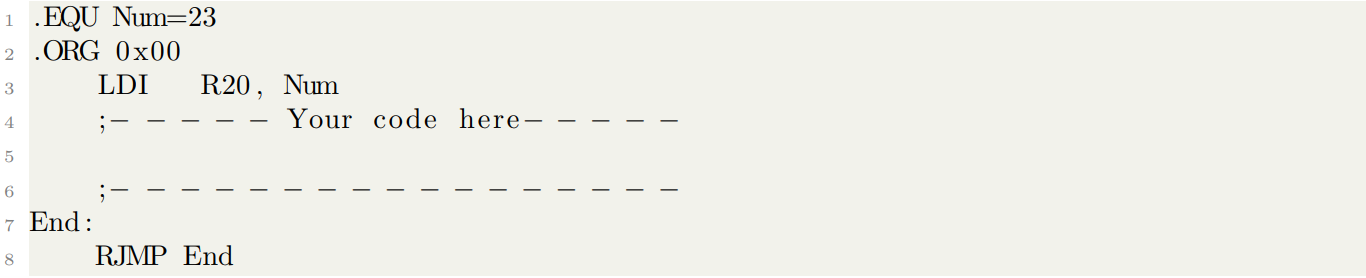
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First Operand | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| Hexa-Decimal Value | **0x011** | | | **Decimal Value** | **17** | | | |
| Second Operand | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| Hexa-Decimal Value | **0x0AB** | | | **Decimal Value** | **171** | | | |
| R1 | Bit **7** | Bit **6** | Bit **5** | Bit **4** | Bit **3** | Bit **2** | Bit **1** | Bit **0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| R0 | Bit **7** | Bit **6** | Bit **5** | Bit **4** | Bit **3** | Bit **2** | Bit **1** | Bit **0** |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| Hexa-Decimal Value | **0X0B5B** | | | **Decimal Value** | **2907** | | | |
| Control Status Register | **I** | **T** | **H** | **S** | **V** | **N** | **Z** | **C** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alterations | No | No | No | No | No | No | No | No |
| Reason for alterations |  |  |  |  |  |  |  |  |

## Task D

In digital systems, packed binary coded decimal (BCD) is a type of binary code in which each decimal digit is represented by four bits. A small algorithm called “Double Dable”6 is used to convert binary numbers to BCD representation. Binary numbers from 0 − 99 will have a BCD representation in 8-bits. Follow the algorithm explained below and write an assembly program that converts binary numbers (0 − 99) to 8-bit BCD. Keep the result in R20.

The lower 4-bits of 8-bit BCD represents the “Unit digit” and upper 4-bits represents “Tens digit”.

1. Clear 8-bit BCD to zero.
2. Check (separately) if “Unit digit” and “Tens digit” is less than or equal to four (1002).
3. If not, add three (112) to them (separately).
4. Shift MSB of binary number to LSB of BCD representation and repeat from step 2 until all the bits of binary number are shifted to BCD.
5. Stop after last shift.



**;**

**; Lab2.asm**

**;**

**; Created: 18/02/2022 11:13:35 pm**

**; Author : Umer and Danial**

**;**

**.EQU Num = 23 ; Number to convert to BCD**

**.ORG 0x00**

**LDI R20, Num ; Binary Equivalent of Base 10**

**MOV R30, R20 ; Copy**

**LDI R25, 0x03 ; Value 3 for Unit Place**

**LDI R26, 0x30 ; Value 3 for Tens Place**

**CLR R16 ; FINAL BCD RESULT IN R16**

**CLR R17 ; Initializing registers to 0x00**

**CLR R18**

**CLR R21**

**CLR R22**

**CLR R23 ; Unit Place**

**CLR R24 ; Tens Place**

**BST R20, 0 ; To cater for even discrepancy**

**BRTC Even**

**L1:**

**ANDI R23, 0x0F ; Dividing No. Into Tens and Unit**

**ANDI R24, 0xF0 ; 0x0F : 0000 1111, 0xF0 : 1111 0000**

**CPI R20, 0X00**

**BREQ BCDexclusiveOR ; Last Shift Stop**

**CPI R23, 0x05 ; Goes to UNIT ADD loop if Unit > 4 -> Unit >= 5**

**BRSH L2**

**L3:**

**CPI R24, 0x50 ; Goes to TENS ADD loop if Unit >= 5**

**BRSH L4**

**L5:**

**LSL R20**

**; Logic: Shift left R20; If R20 MSB was 1 then C Flag is 1**

**; Rotate R16 left by the value of Carry which is 1.**

**; C Flag is 0 otherwise, R16 rotates for 0.**

**ROL R16**

**MOV R23, R16**

**MOV R24, R16**

**RJMP L1**

**L2:**

**ADD R23, R25 ; Adds 3 To Units Place**

**ADD R16, R25**

**RJMP L3**

**L4:**

**ADD R24, R26 ; Adds 3 To Tens Place**

**ADD R16, R26**

**RJMP L5**

**BCDexclusiveOR: ; Using XOR to Store The BCD Value in R23.**

**EOR R23, R24**

**BST R30, 0 ; Counters Even INC by DEC 1/2**

**BRTC Decrement**

**RJMP End**

**Even: ; Increments Even Inputs to Make 0th bit 1**

**INC R20**

**RJMP L1**

**Decrement: ; Counters Even Cases INC by DEC 2/2**

**DEC R16**

**End:**

**RJMP End**

## Output Screenshots

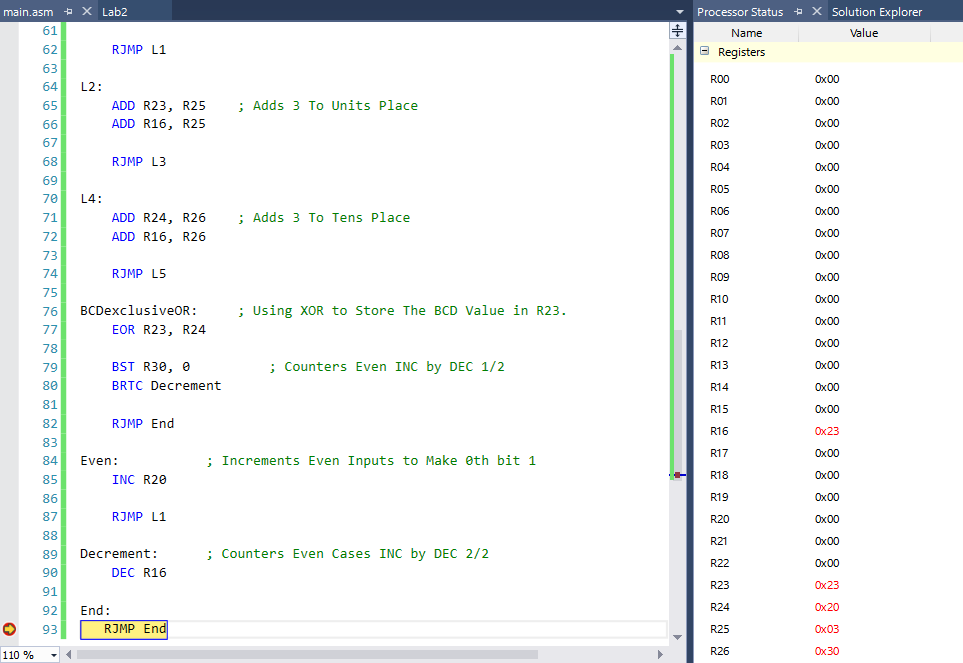


Figure 4 Exemplar Output: For Num = 23

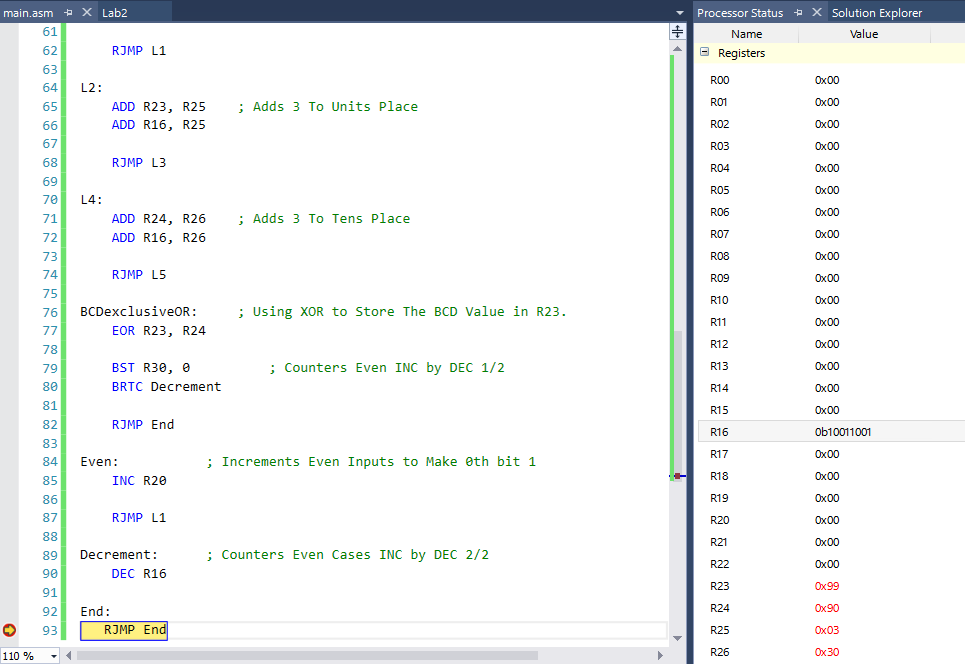


Figure 5 Exemplar Output: For Num = 99

## Task E

When we program AVR microcontrollers in C, the “int” data type is of 2-bytes. But as you have seen till now, in assembly, all the instructions handle data in form of a single byte and the data chunks greater than a byte are first broken down into bytes and then processed separately. Suppose we have two integers 128410 and 77510. Write an assembly program that;

* 1. Step 1: Saves them in memory location 0x200 and 0x300.
  2. Step 2: Calculates sum (1284 + 775) in R20 and R21.
  3. Step 3: Subtracts them (1284 - 775) and keep result in R22 and R23.

**;**

**; Lab2.asm**

**;**

**; Created: 16/02/2022 8:59:56 pm**

**; Author : Umer and Danial**

**;**

**.ORG 0x00**

**LDI R16, 0x05 ; Initialization**

**LDI R17, 0x04**

**LDI R18, 0x03**

**LDI R19, 0x07**

**STS 0x200, R17 ; For Storage (LITTLE ENDIAN)**

**STS 0x201, R16**

**STS 0x300, R19**

**STS 0x301, R18**

**MOV R20, R16 ; For Addition**

**MOV R21, R17**

**ADD R21, R19**

**ADC R20, R18 ; RES in R20, R21**

**MOV R22, R16 ; For Subtraction**

**MOV R23, R17**

**SUB R23, R19**

**SBC R22, R18 ; RES in R22, R23**

**NOP ; No Operation**

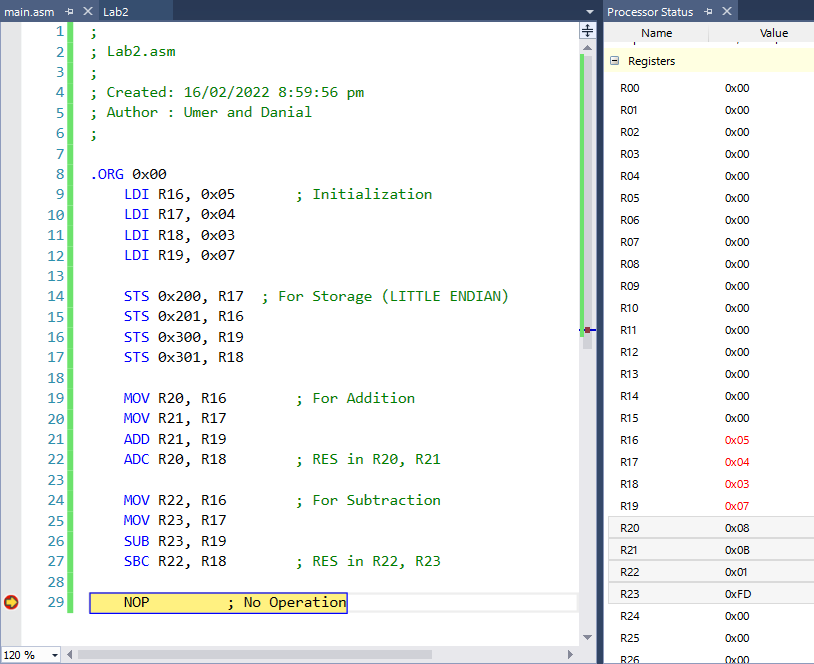


Figure 6 Exemplar Output

## Conclusion

After the conduction of this lab, we have learnt how to utilize common AVR instructions to accomplish a multitude of tasks. These tasks can range from code conversion, simple arithmetic, counters, to large scale projects, etc. We also learned how to utilize branch instructions to get a grasp of loops and alter the execution order, until a certain condition remains true/false. Lastly, we employed storage of binary numbers exceeding 8 – bits (the extent of a single register) by realizing that adjacent memory addresses can be used to store a much larger number.