Emulating the Atmel AVR Microcontroller using AVR Assembly Language Programming on Microchip Studio

EE2016 Microprocessors

Sanjeev Subrahmaniyan S B

EE23B102

Abstract

The Atmel AVR microcontroller series are a RISC based 8-bit processor suited for beginners and professionals to implement small scale computational and embedded applications. In this experiment, the Atmel ATMega8 microcontroller is emulated on the Microchip studio software. Assembly language programming is used to implement simple programs. The simulators interfaces are used to monitor the progress of the programs and gain experience of assembly programming and memory handling. The implemented programs include addition and multiplication of 8-bit numbers, addition of 16-bit numbers and iterative maximum finding on the 8-bit microcontroller.

1 Introduction

Microcontrollers are indispensable in the electronic industry, being the base for all prototyping and testing. In this experiment, the Atmel ATmega8 microcontroller is emulated to gain experience of assembly language programming. This also helps one familiarise with the instruction set of the microcontroller and gain an under the hood understanding of the internal working of the processor and memory elements.

2 Objectives

The tasks emulated are:

1. Addition of two 8-bit numbers

2. Addition of two 16-bit numbers
3. Multiplication of two 8-bit numbers
4. Determining the maximum element from a given finite set of 8-bit numbers

3 Assembly language programs and interpretation

All of the programs are written with appropriate comments and flowcharts to explain the function performed at each step.

3.1 Addition of two 8-bit numbers

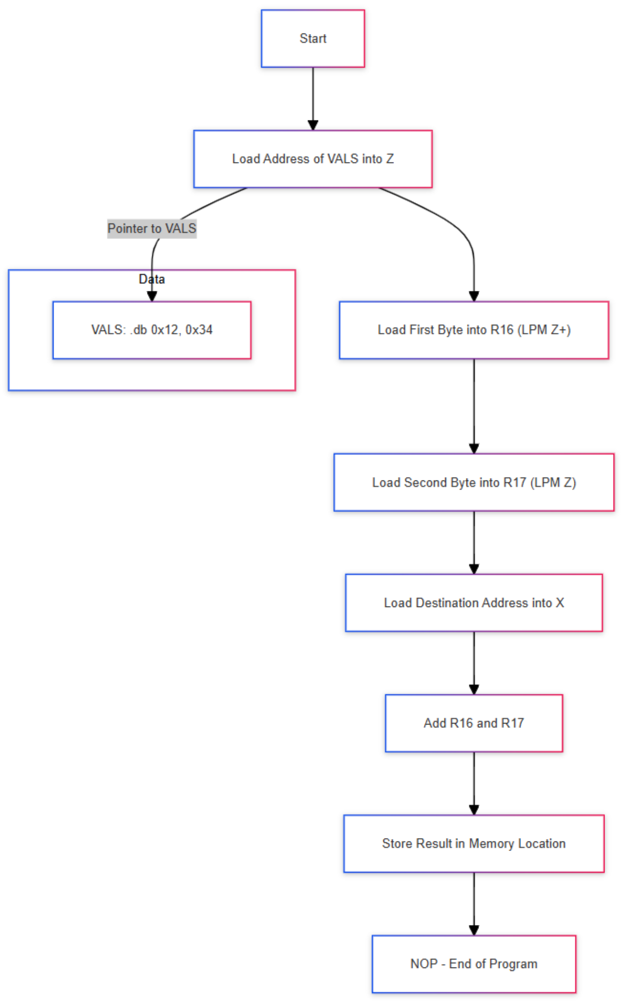
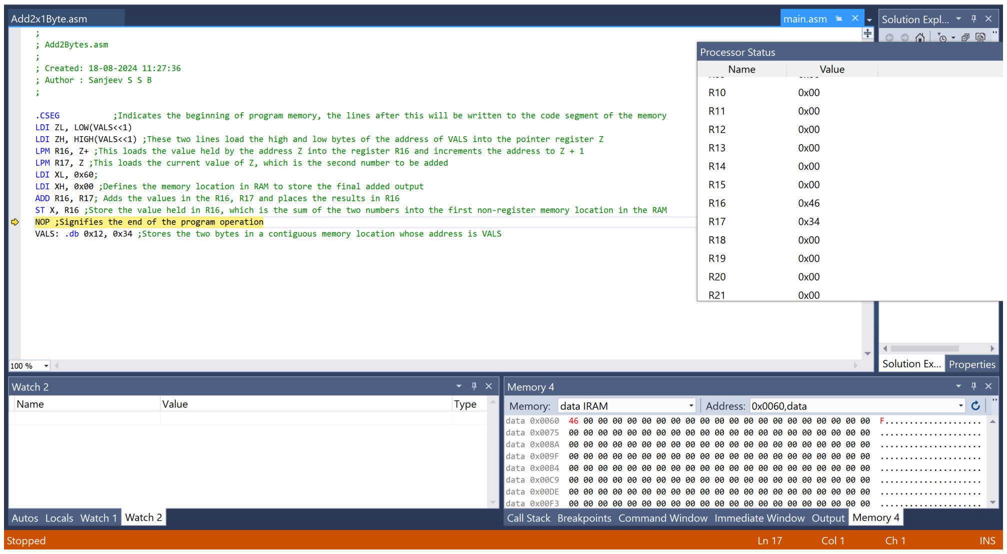

Start
Load Address of VALS into Z
Pointer to VALS
Data
VALS: .db 0x12, 0x34
Load First Byte into R16 (LPM Z+)
Load Second Byte into R17 (LPM Z)
Load Destination Address into X
Add R16 and R17
Store Result in Memory Location
NOP - End of Program

Figure 1: Pseudocode flowchart for addition



```
;
; Add2Bytes.asm
;
; Created: 18-08-2024 11:27:36
; Author : Sanjeev S S B
;

.CSEG           ;Indicates the beginning of program memory, the lines after this will be written to the code segment of the memory
LDI ZL, LOW(VALS<<1)
LDI ZH, HIGH(VALS<<1) ;These two lines load the high and low bytes of the address of VALS into the pointer register Z
LPM R16, Z+ ;This loads the value held by the address Z into the register R16 and increments the address to Z + 1
LPM R17, Z ;This loads the current value of Z, which is the second number to be added
LDI XL, 0x60;
LDI XH, 0x00 ;Defines the memory location in RAM to store the final added output
ADD R16, R17; Adds the values in the R16, R17 and places the results in R16
ST X, R16 ;Store the value held in R16, which is the sum of the two numbers into the first non-register memory location in the RAM
NOP ;Signifies the end of the program operation
VALS: .db 0x12, 0x34 ;Stores the two bytes in a contiguous memory location whose address is VALS
```

Processor Status

Name	Value
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x46
R17	0x34
R18	0x00
R19	0x00
R20	0x00
R21	0x00

Memory: data IRAM   Address: 0x0060,data

```
data 0x0060  46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  F....................
data 0x0075  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x008A  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x009F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00B4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00C9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00DE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00F3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
```

Figure 2: Addition of two 8-bit numbers

The addition of two 8-bit numbers is done using the ADD opcode. This is used because there is no carry initially. The program is explained through the comments written alongside the it. In the end, the ST operation is used to store the final answer into RAM memory, which is highlighted in red in the bottom-right corner of the image. The register values at the end of the program are also shown in the processor status dialog.

3.2 Multiplication of two 8-bit numbers

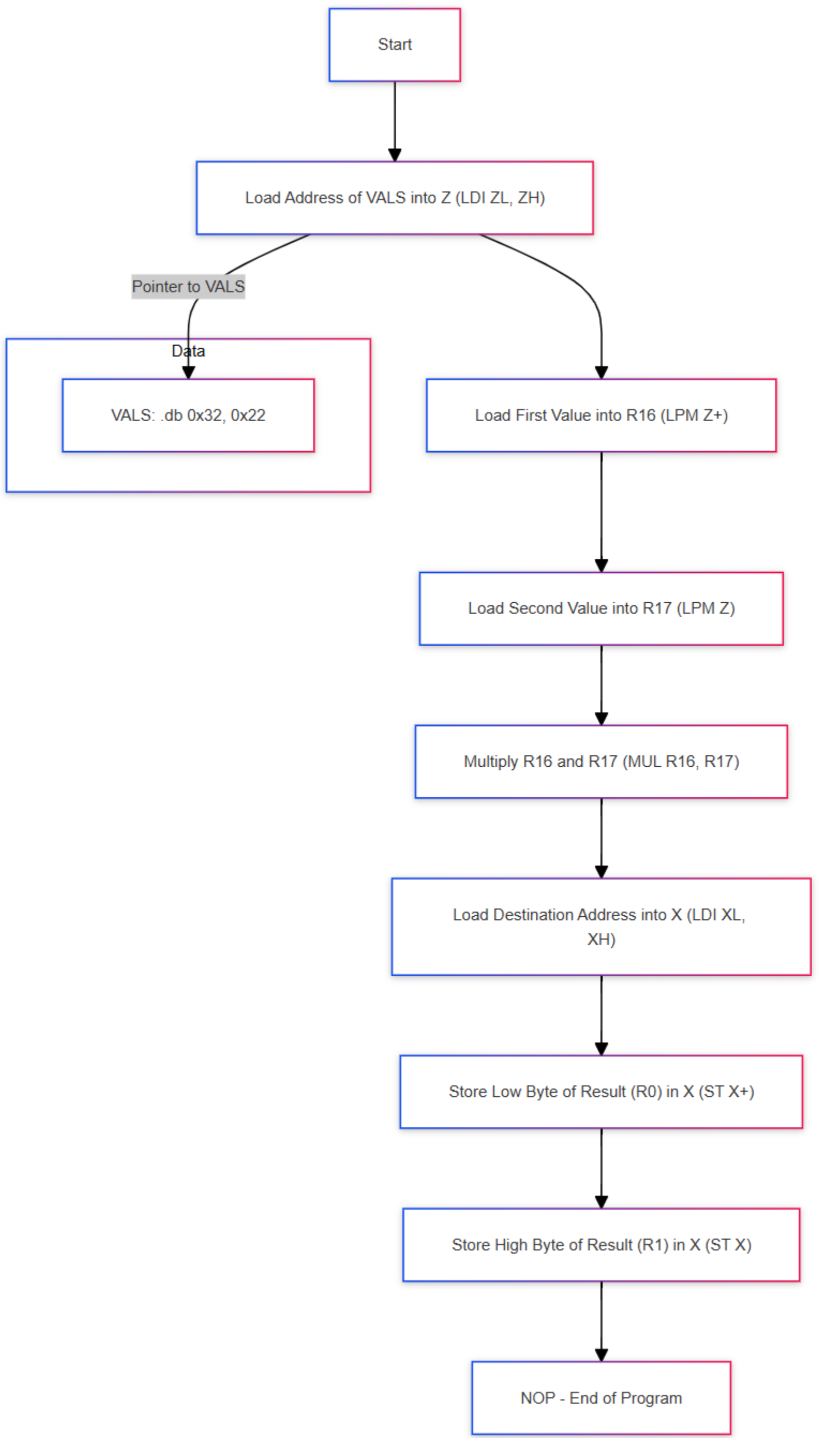
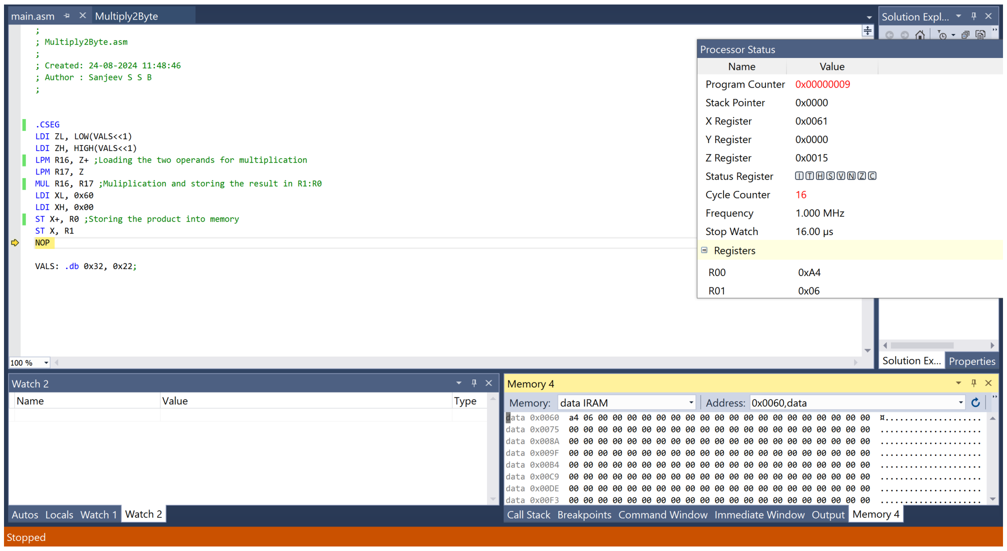

Start
Load Address of VALS into Z (LDI ZL, ZH)
Pointer to VALS
Data
VALS: .db 0x32, 0x22
Load First Value into R16 (LPM Z+)
Load Second Value into R17 (LPM Z)
Multiply R16 and R17 (MUL R16, R17)
Load Destination Address into X (LDI XL, XH)
Store Low Byte of Result (R0) in X (ST X+)
Store High Byte of Result (R1) in X (ST X)
NOP - End of Program

Figure 3: Flowchart for multiplication of two numbers



```
;
; Multiply2Byte.asm
;
; Created: 24-08-2024 11:48:46
; Author : Sanjeev S S B
;


.CSEG
LDI ZL, LOW(VALS<<1)
LDI ZH, HIGH(VALS<<1)
LPM R16, Z+ ;Loading the two operands for multiplication
LPM R17, Z
MUL R16, R17 ;Muliplication and storing the result in R1:R0
LDI XL, 0x60
LDI XH, 0x00
ST X+, R0 ;Storing the product into memory
ST X, R1
NOP

VALS: .db 0x32, 0x22;
```

Processor Status

Name	Value
Program Counter	0x00000009
Stack Pointer	0x0000
X Register	0x0061
Y Register	0x0000
Z Register	0x0015
Status Register	I T H S V N Z C
Cycle Counter	16
Frequency	1.000 MHz
Stop Watch	16.00 µs
Registers	
R00	0xA4
R01	0x06

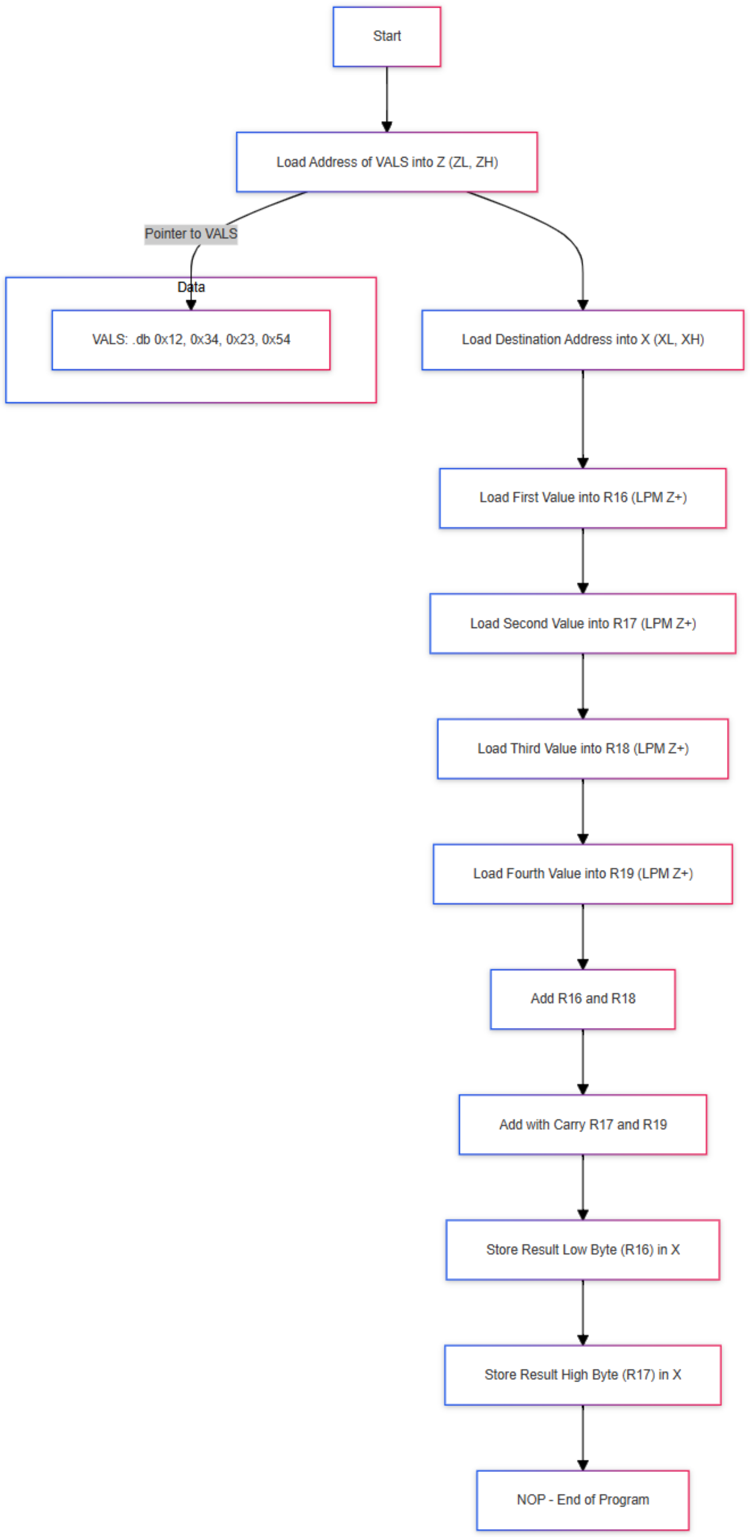
Memory 4 — Memory: data IRAM — Address: 0x0060,data

```
data 0x0060  a4 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ¤....................
data 0x0075  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x008A  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x009F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00B4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00C9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00DE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x00F3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
```

Figure 4: Multiplication of two 8-bit numbers

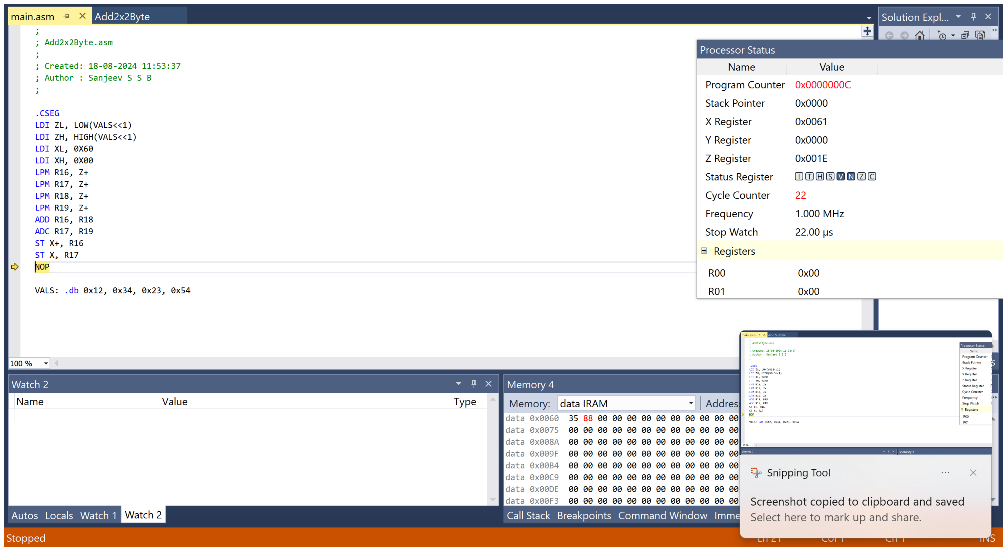
The multiplication of two 8-bit numbers is done similarly using the MUL operation. Similar to the previous program, the final answer, which is a 16-bit result, is stored in two contiguous locations in memory as shown in the bottom right part of the image.

3.3 Addition of two 16-bit numbers



Start
Load Address of VALS into Z (ZL, ZH)
Pointer to VALS
Data
VALS: .db 0x12, 0x34, 0x23, 0x54
Load Destination Address into X (XL, XH)
Load First Value into R16 (LPM Z+)
Load Second Value into R17 (LPM Z+)
Load Third Value into R18 (LPM Z+)
Load Fourth Value into R19 (LPM Z+)
Add R16 and R18
Add with Carry R17 and R19
Store Result Low Byte (R16) in X
Store Result High Byte (R17) in X
NOP - End of Program

Figure 5: Pseudocode flowchart for addition of words(16-bit)



```
;
; Add2x2Byte.asm
;
; Created: 18-08-2024 11:53:37
; Author : Sanjeev S S B
;

.CSEG
LDI ZL, LOW(VALS<<1)
LDI ZH, HIGH(VALS<<1)
LDI XL, 0X60
LDI XH, 0X00
LPM R16, Z+
LPM R17, Z+
LPM R18, Z+
LPM R19, Z+
ADD R16, R18
ADC R17, R19
ST X+, R16
ST X, R17
NOP

VALS: .db 0x12, 0x34, 0x23, 0x54
```

Processor Status

Name	Value
Program Counter	0x0000000C
Stack Pointer	0x0000
X Register	0x0061
Y Register	0x0000
Z Register	0x001E
Status Register	I T H S V N Z C
Cycle Counter	22
Frequency	1.000 MHz
Stop Watch	22.00 µs
Registers	
R00	0x00
R01	0x00

```
data 0x0060  35 88 00 00 00 00 00 00 00 00 00 00
data 0x0075  00 00 00 00 00 00 00 00 00 00 00 00
data 0x008A  00 00 00 00 00 00 00 00 00 00 00 00
data 0x009F  00 00 00 00 00 00 00 00 00 00 00 00
data 0x00B4  00 00 00 00 00 00 00 00 00 00 00 00
data 0x00C9  00 00 00 00 00 00 00 00 00 00 00 00
data 0x00DE  00 00 00 00 00 00 00 00 00 00 00 00
data 0x00F3  00 00 00 00 00 00 00 00 00 00 00 00
```

Snipping Tool
Screenshot copied to clipboard and saved
Select here to mark up and share.

Figure 6: Addition of two 16-bit numbers

The 16-bit addition is handled by first adding the last bytes of both the numbers using the ADD operation and then using the ADC operation to add the first bytes along with the carry from the first addition. Similar to the previous programs, the final answer is stored in a memory location as seen in the bottom-right of the image.

3.4 Finding maximum of a finite set of numbers

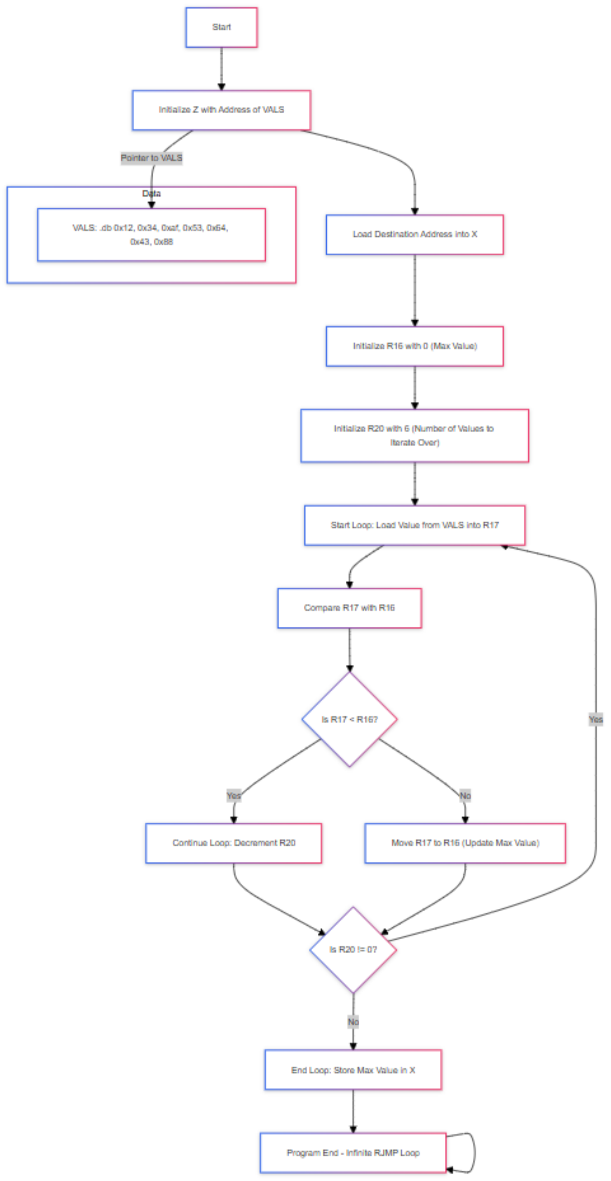
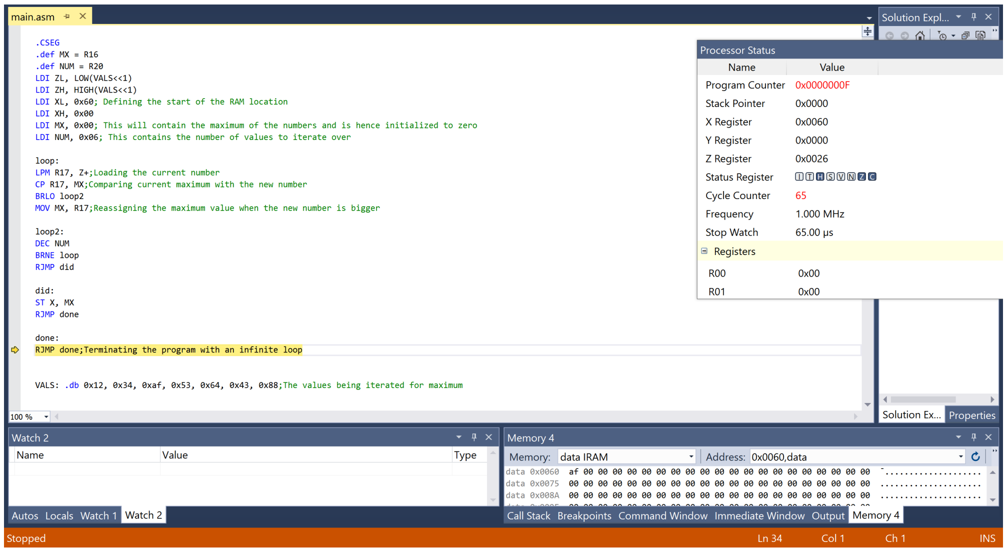

Start
Initialize Z with Address of VALS
Pointer to VALS
Data
VALS: .db 0x12, 0x34, 0xaf, 0x53, 0x64, 0x43, 0x88
Load Destination Address into X
Initialize R16 with 0 (Max Value)
Initialize R20 with 6 (Number of Values to Iterate Over)
Start Loop: Load Value from VALS into R17
Compare R17 with R16
Is R17 < R16?
Yes
No
Continue Loop: Decrement R20
Move R17 to R16 (Update Max Value)
Is R20 != 0?
Yes
No
End Loop: Store Max Value in X
Program End - Infinite RJMP Loop

Figure 7: Pseudocode flowchart for finding the maximum of the set of numbers



```
.CSEG
.def MX = R16
.def NUM = R20
LDI ZL, LOW(VALS<<1)
LDI ZH, HIGH(VALS<<1)
LDI XL, 0x60; Defining the start of the RAM location
LDI XH, 0x00
LDI MX, 0x00; This will contain the maximum of the numbers and is hence initialized to zero
LDI NUM, 0x06; This contains the number of values to iterate over

loop:
LPM R17, Z+;Loading the current number
CP R17, MX;Comparing current maximum with the new number
BRLO loop2
MOV MX, R17;Reassigning the maximum value when the new number is bigger

loop2:
DEC NUM
BRNE loop
RJMP did

did:
ST X, MX
RJMP done

done:
RJMP done;Terminating the program with an infinite loop


VALS: .db 0x12, 0x34, 0xaf, 0x53, 0x64, 0x43, 0x88;The values being iterated for maximum
```

Processor Status

Name	Value
Program Counter	0x0000000F
Stack Pointer	0x0000
X Register	0x0060
Y Register	0x0000
Z Register	0x0026
Status Register	I T H S V N Z C
Cycle Counter	65
Frequency	1.000 MHz
Stop Watch	65.00 µs
Registers	
R00	0x00
R01	0x00

Watch 2

Name	Value	Type

Memory 4

Memory: data IRAM Address: 0x0060,data

```
data 0x0060  af 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ¯....................
data 0x0075  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
data 0x008A  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....................
```

Figure 8: Finding maximum of a set of numbers

The determination of the maximum of numbers is done by iteratively comparing the values using the CP operation, which are held in a temporary register R17. A loop is implemented which checks if the new number is bigger than the current maximum and updates the maximum if that is the case. A counter variable NUM is used to keep track of the number of elements iterated over. The final answer is stored into the RAM as seen in the bottom-right of the image.

4 Procedure

The procedure undertaken for the experiment is as follows:

1. Create an assembler project on the Microchip Studio software.
2. Create flowcharts to ideate on the solution and validate its functionality.
3. Write the assembly language program to perform the given task.
4. Build the solution and debug the solution using breakpoints.
5. Monitor the processor registers and memory values to infer the completion of the program.
6. Report the observations.

5 My contribution to the experiment

1. Write the assembly programs and psedocode to perform the given tasks(because this was a simple assignment, we all wrote our own programs).
2. Debug and execute the programs and compile the results.

6 Conclusion

The experiment provided hands-on experience with assembly language programming on the Atmel ATmega8 microcontroller using Microchip Studio. By implementing fundamental operations such as addition, multiplication, and maximum finding, a deeper understanding of low-level programming concepts, processor architecture, and memory management was gained. The iterative nature of the tasks reinforced the importance of logical flow and the correct use of instructions like ADD, ADC, MUL, and CP in managing data efficiently. The use of simulation tools in Microchip Studio offered valuable insights into how microcontrollers execute instructions and handle registers and memory, contributing to a more comprehensive understanding of embedded systems development.