

EE2016: Microprocessors Lab
Experiment # 2 AVR Atmel8

Amizhthni PRK, EE21B015
Nachiket Dighe, EE21B093

October 3, 2022

Contents

1	Addition of 8-bits	1
1.1	Approach and Flow-chart	1
1.2	Code explanation	2
1.3	Code	2
1.4	Code Inputs and Outputs	3
2	Addition of 16-bits	4
2.1	Approach and Flow-chart	4
2.2	Code	5
2.3	Outputs and code inputs	6
3	Multiplication of 8-bits	7
3.1	Approach and Flow-chart	7
3.2	Code	9
3.3	Outputs and code inputs	9
4	Largest number in a given set	10
4.1	Approach and Flow-chart	10
4.2	Code	10
4.3	Outputs and code inputs	11

List of Figures

1	Flowchart for 8 bit addition	1
2	Sample input (FA + 12)	3
3	Flowchart for 16 bit addition	4
4	Sample input (0005+0104)	6
5	Calculation - multiplication	7
6	Flowchart for multiplication	8
7	Sample input (AB*CD)	9
8	Flowchart for Comparison	10
9	Sample input (5,8,4,9)	11

1 Addition of 8-bits

To implement Addition of two 8-bit numbers using Atmel Atmega-8 microcontroller.

1.1 Approach and Flow-chart

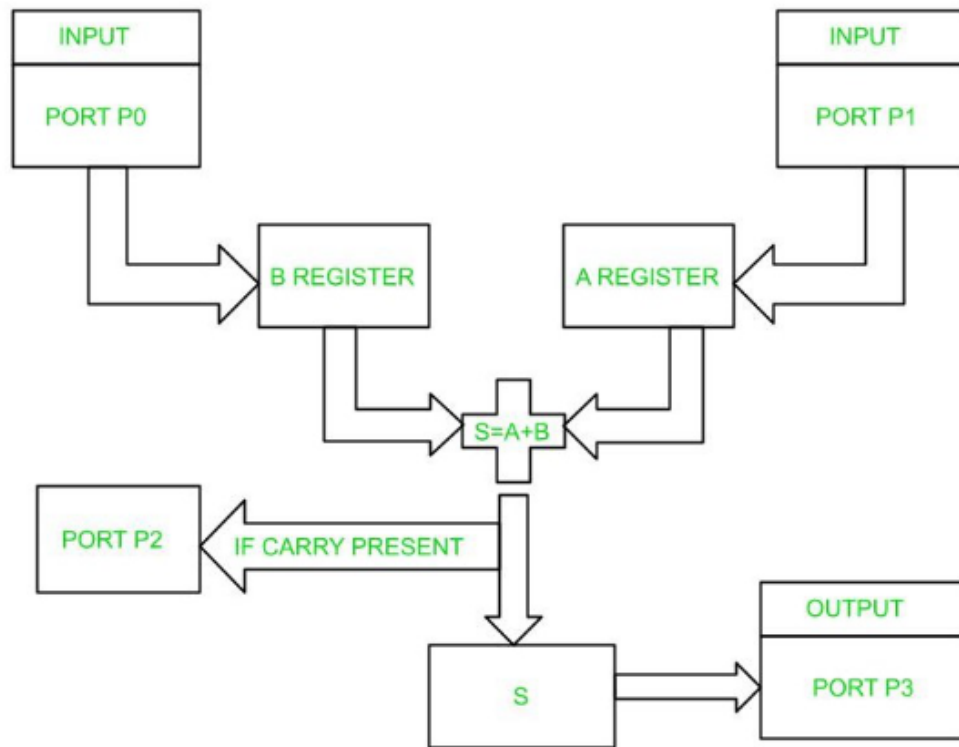


Figure 1: Flowchart for 8 bit addition

- Flowchart has been included under [1](#).
- Initialize Ports P0 and P1 as input ports.
- Initialize Ports P2 and P3 as output ports.
- Initialize the R1 register.
- Move the contents from Port 0 to B register.
- Move the contents from Port 1 to A register.
- Add contents in A and B.
- If carry is present increment R1.
- Move contents in R1 to Port 2.
- Move the sum in step 6 to Port 3.

1.2 Code explanation

- We try to map the code and the flowchart.
- The input 0XFA is given to register 20(A) and input 0X12 is loaded into register 21(B).
- We want to leave the input registers unchanged, i.e. we want to store the sum in a different register. Thus we move the inputs to a different register using the MOV command.
- Since LDI(load immediate) only works on registers 16 through 31, the mov instruction is a useful way to load a constant into one of the lower 16 registers.
- Even though the mov instruction looks like move, it actually copies the contents of one register into another. After the instruction, the source register is left unchanged.
- We initiate register 16 as the carry register, and it has an initial value of 0.
- We add the given two values and store them in a register
- We then use a BRCC command(Branch if Carry Cleared). It creates another branch if the given condition is failed(Port 2). It creates another branch if the carry is not 0, and sets Register 16 at 1. It ignores the command if the carry is 0 (Port 3).
- FA+12 in hexadecimal correspond to 10C. 0C is stored in the output port and the carry which is always 1, is stored in the other register.
- Since the port or memory locations were not initialized by us, it is taken as 0X00 by default.

1.3 Code

The code used for finding the sum of two 8-bit numbers is given below in listing 1.

```
1  /*
2  * BATCH6_1.asm
3  *
4  * Created: 06-09-2022 08:49:47
5  * Author : students
6  */
7
8  // Addition of two 8-bit numbers
9  .CSEG
10 ; inputs
11 LDI R20, 0XFA //loading 0xFA in register R20
12 LDI R21, 0X12 //loading 0x12 in register R21
13
14 MOV R17, R20
15 LDI R16, 0X00
16 ADD R17, R21 // Sum is stored in register R17
17 BRCC NOCARRYLABEL
18 LDI R16,0X01 // carry is stored in register R16
19 NOCARRYLABEL:
20 NOP
```

Listing 1: Addition of 8-bit numbers

1.4 Code Inputs and Outputs

The Final result is stored in register R16 and R17 where R16 is the carry and R17 is the sum. Thus, for inputs 0xFA and 0x12 , **R17 = 0x0C** and **R16 = 0x01**.

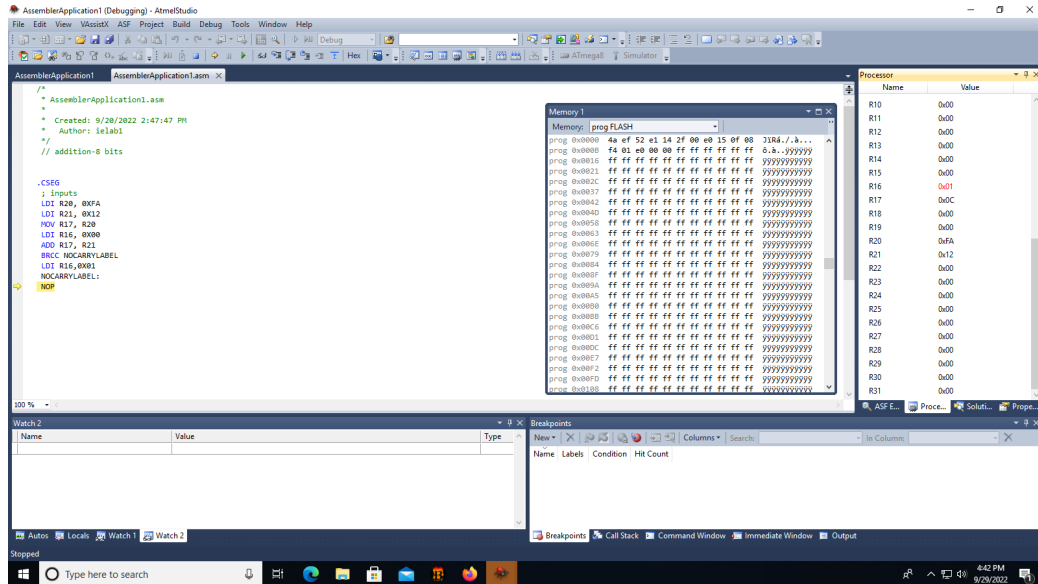


Figure 2: Sample input (FA + 12)

2 Addition of 16-bits

To implement Addition of two 16-bit numbers using Atmel Atmega-8 microcontroller.

2.1 Approach and Flow-chart

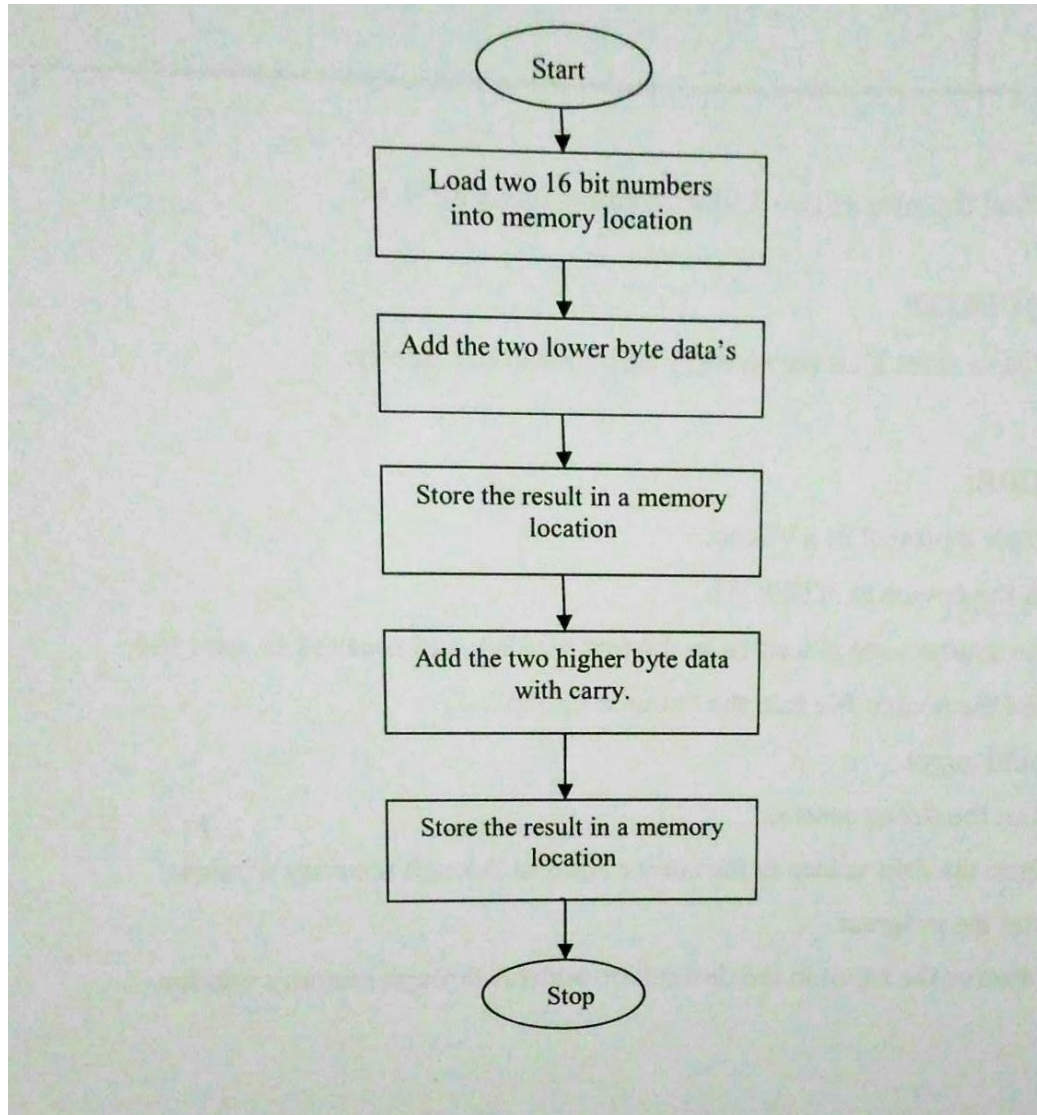


Figure 3: Flowchart for 16 bit addition

- Flowchart has been included under [3](#).
- Load the operands in 4 registers(two registers for storing lower 8-bits and higher 8-bits of a number).
- Initialize a register to 0 for carry.
- Add the lower byte of the two numbers and store the result in a register.
- To check if carry is generated or not we use the BRCC instruction.

- Add the higher byte of the two numbers with the carry that is generated because of the lower bytes and store the result in another register.
- Store the second carry generated by the higher bytes in a third register.
- Store the output of the three registers in some consecutive memory locations.

2.2 Code

The code used for finding the sum of two 16-bit numbers is shown below in listing 4.

```

1  /*
2  * BATCH6_2.asm
3  *
4  * Created: 06-09-2022 08:49:47
5  * Author : students
6  */
7
8  // Addition of two 16-bit numbers
9
10 .CSEG; //Start program
11
12 LDI ZL, LOW(NUM<<1);
13 LDI ZH, HIGH(NUM<<1);
14 LPM R0, Z+;
15 LPM R1, Z+;
16 LPM R20, Z+;
17 LPM R21, Z;
18
19 LDI R16, 0x00; //clearing sumL register
20 LDI R17, 0x00; //clearing sumH register
21 LDI R18, 0x00; //clearing carry register
22
23
24 MOV R16, R0; //R16 <-- R0
25 MOV R17, R1; //R17 <-- R1
26
27 ADD R16, R20; //R16 <-- R16 + R20
28 ADC R17, R21; //R17 <-- R17 + R21 + C (from previous step)
29
30 BRCC noCarry; //Skip to storing values to SRAM
31 LDI R18, 0x01; //making carry 1 if needed
32
33 noCarry: STS 0x60, R16; //Storing value in R16 to SRAM location 0x60 (sumL)
34 STS 0x61, R17; //Storing value in R17 to SRAM location 0x61 (sumH)
35 STS 0x62, R18; //Storing value in R18 to SRAM location 0x62 (carry)
36
37 NOP; End of program
38
39 NUM: .db 0xD3, 0x5F, 0xAB, 0xCD;

```

Listing 2: Addition of 16-bit numbers

2.3 Outputs and code inputs

For input 0005 and 0104 the result is stored in registers R16 = 0x01(higher byte) , R17 = 0x09(lower byte) , R18 = 0(carry)

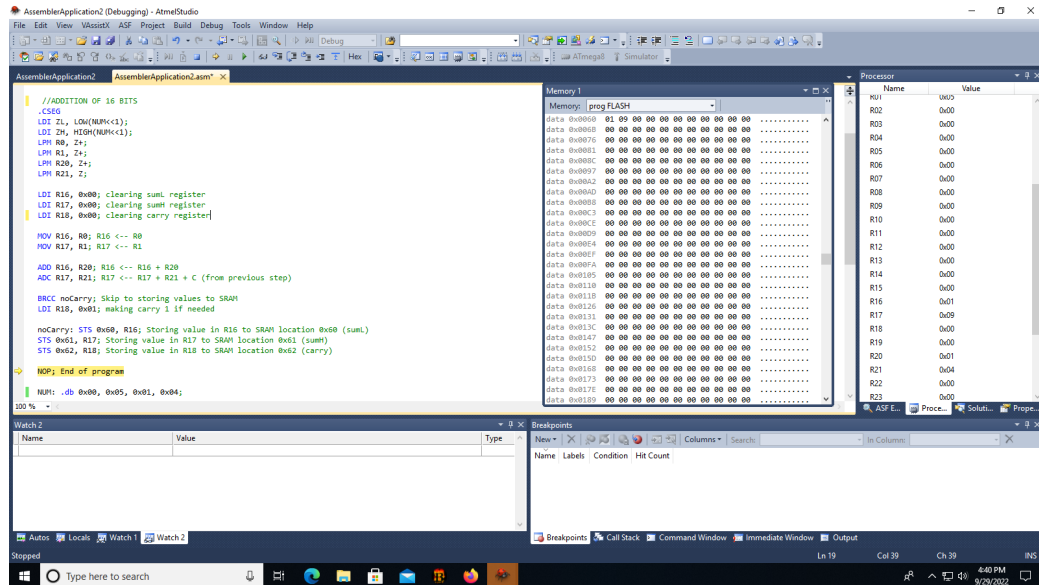


Figure 4: Sample input (0005+0104)

3 Multiplication of 8-bits

To implement Addition of two 16-bit numbers using Atmel Atmega-8 microcontroller.

Hex value:

AB × CD = **88EF**

Decimal value:

171 × 205 = **35055**




Figure 5: Calculation - multiplication

3.1 Approach and Flow-chart

- MUL, multiplication of unsigned integers
- MULS, multiplication of signed integers
- MULSU, multiplication of a signed integer with an unsigned integer
- FMUL, multiplication of unsigned fractional numbers
- FMULS, multiplication of signed fractional numbers
- FMULSU, multiplication of a signed fractional number and with an unsigned fractional number

We use the MUL method in our code.

- Refer [5](#) for calculation.
- Just load the operands into two registers (or only one for square multiply) and execute one of the multiply instructions. The result will be placed in register pair R0:R1. However, note that only the MUL instruction does not have register usage restrictions.
- The answer automatically gets stored in R0:R1 pair.
- We need to use MOVW method to move the entire pair of register values.
- `movw r17:r16,r1:r0` can also be used for the above purpose.
- Flowchart is rather straightforward, given under [6](#)
- ZL is R0 in this case, and ZH is R1. After movw operation, ZL is R16 and ZH is R17. EF are the lower bits and are thus shown in the lower register and 88 are shown in the higher register.

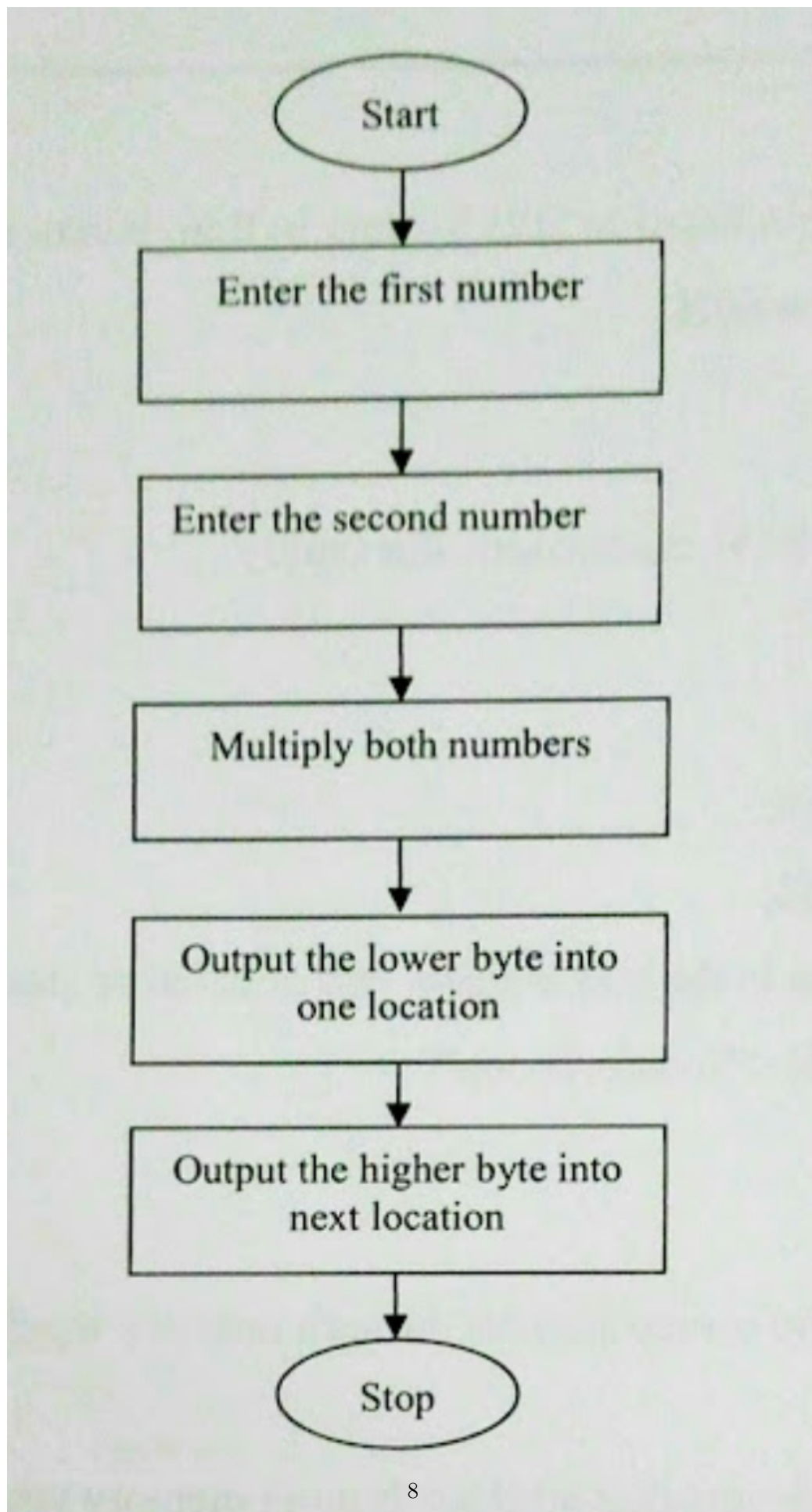


Figure 6: Flowchart for multiplication

3.2 Code

The code used for Calculating the product of two 8-bit numbers is shown below in listing 4.

```

1  /*
2  *  BATCH6_3.asm
3  *
4  *  Created: 06-09-2022 09:35:20
5  *  Author: students
6  */
7
8  // Multiplication of two 8-bit numbers
9
10 .CSEG //Start Program, The CSEG directive defines the start of a
    ↳ CodeSegment.
11 ; inputs
12 LDI R20, 0X23 //taking inputs
13 LDI R21, 0XBA //taking inputs
14
15 MUL R20, R21 //Product is stored in Register:0
16 MOVW R16, R0 // result is stored in R16 and carry in R17, this transfers
    ↳ the pair from r0:r1 to r16:r17
17 NOP //End program

```

Listing 3: Multiplication of 8-bit numbers

3.3 Outputs and code inputs

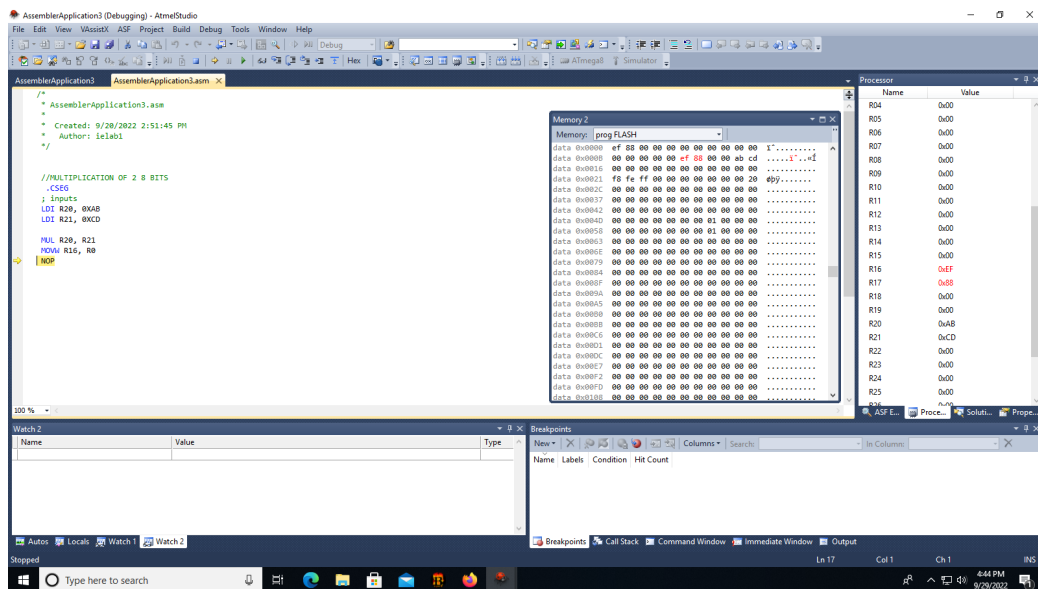


Figure 7: Sample input (AB*CD)

4 Largest number in a given set

To implement Largest number in a given set using Atmel Atmega-8 microcontroller.

4.1 Approach and Flow-chart

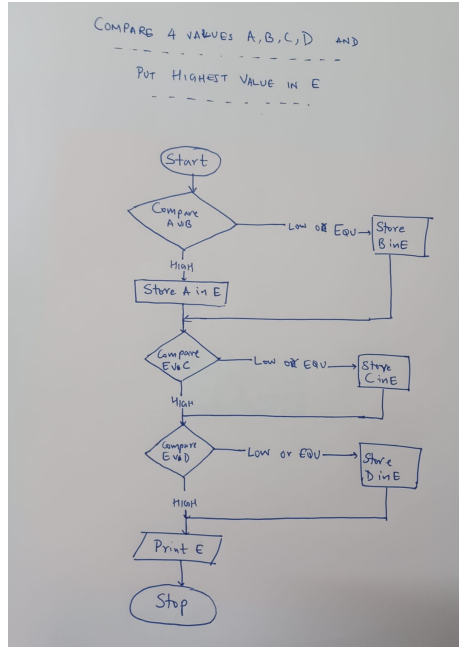


Figure 8: Flowchart for Comparison

- Start the program.
- load the operands in four registers(R20,R21,R22,R23).
- Move the contents of register R20 to R16
- compare R16 and second register(R21) and store the maximum result in register(R16).
- compare third register(R22) with R16 and store the maximum result in register(R16).
- compare third register(R23) with R16 and store the maximum result in register(R16).
- End the program.

4.2 Code

The code used for Finding Largest number of a given set is given below in listing 4.

```
1  /*
2  *  BATCH6_4.asm
3  *
4  *  Created: 06-09-2022 09:52:06
5  *  Author: students
6  */
7
```

```

8 // To find Largest of 4 numbers
9 .CSEG // Start Program, The CSEG directive defines the start of a Code
  ↳ Segment.
10 LDI R20, 12 //GIVING INPUTS
11 LDI R21, 13 //GIVING INPUTS
12 LDI R22, 19 //GIVING INPUTS
13 LDI R23, 29 //GIVING INPUTS
14
15 MOV R16, R20 //R16 HERE IS THE REFERENCE REGISTER, THIS IS DONE TO ENSURE
  ↳ THE INPUT REGISTERS STAY INTACT.
16 CP R21, R16 // Comparing registers R16 and R21, and storing the larger
  ↳ value in R21
17 BRLT LT1// BranchES if r21< r16
18 MOV R16, R21//NOW LARGER ONE OF R20 AND R21 IS IN R16
19 LT1:
20 CP R22, R16 // Compare R16 to R22
21 BRLT LT2//BranchES if r22< r16
22 MOV R16, R22 //NOW LARGER ONE OF R20 AND R21 AND R22 IS IN R16
23 LT2:
24 CP R23, R16 // Compare R16 to R23
25 BRLT LT3// BranchES if r23< r16
26 MOV R16, R23 //NOW LARGER ONE OF R20 AND R21 AND R22 AND R23 IS IN R16
27 LT3: // Final result is stored in register R16
28 NOP // End program

```

Listing 4: Largest Number of a given set

4.3 Outputs and code inputs

For inputs 5,8,4,9 the final result is stored in register R16 = 9.

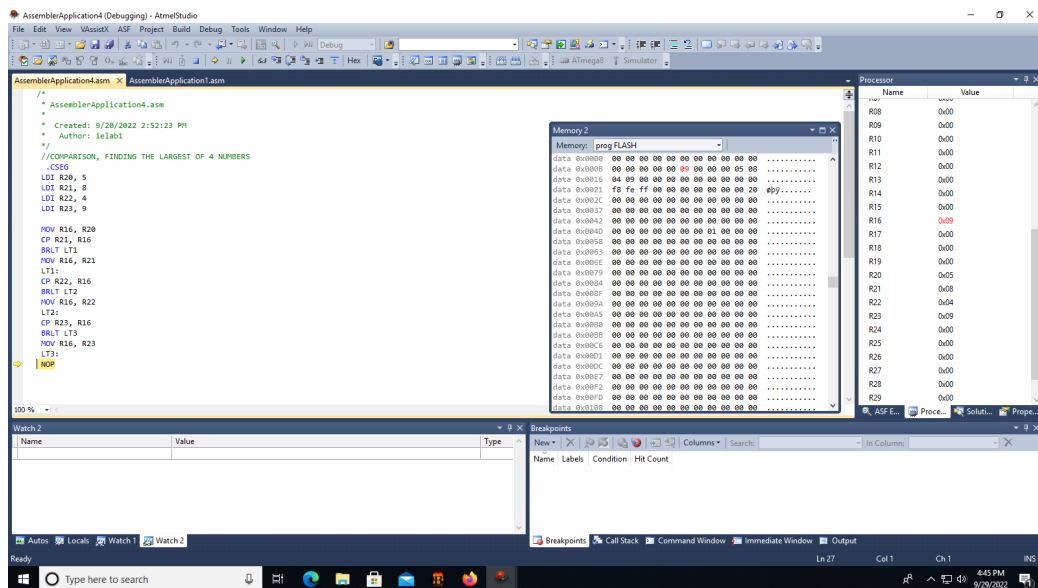


Figure 9: Sample input (5,8,4,9)