MUP-LAB_EXPERIMENT-02_AVR-EMULATION--2016

MIRUDHULA | EE23B046

OBJECTIVES:

To implement basic arithmetic and logical manipulation programs using Atmel Atmega8 microcontroller in assembly program emulation, including addition, multiplication and comparison

Problem 1 (Common 8-bit mathematical operation): Given two 8 bit binary words (byte), compute the sum and store it in a register.
Problem 2 (16-bit addition using a 8-bit processor): Given two 16 bit binary words (byte), compute the sum and store it in a register.
Problem 3 (Multiplication of two 8-bit numbers): Given two 8 bit binary words (byte), compute the product and store it in a register.
Problem 4 (Largest of number given): Given a nite set of binary words, identify the largest in the given set

CODE USED:

PROBLEM_1:

```
.CSEG                         ; Code Segment - Start of the code section
LDI ZL, LOW (NUM<<1)          ; Load the lower byte of the address of the NUM label into
register ZL
LDI ZH, HIGH (NUM<<1)         ; Load the higher byte of the address of the NUM label into
register ZH
LDI XL, 0x60                  ; Load the lower byte of the address (0x0060) into register
                              XL
(This is the destination address in SRAM)
LDI XH, 00                    ; Load the higher byte of the address (0x0060) into register XH

                              ; Initialize register R16 with 0 (This will be used for carry
LDI R16, 00
checking)
LPM R1, Z+               ; Load the byte at the address pointed by Z into R1, then increment Z
LPM R2, Z               ; Load the next byte at the address pointed by Z into R2 (Z remains
unchanged)
MOV R15, R1               ; Move the content of R1 (first byte) into R15 for addition

ADD R15, R2               ; Add the contents of R15 (R1) and R2, and store the result in R15
BRCC forward               ; If there is no carry, jump to the label 'forward'
LDI R16, 0x01             ; If carry occurred, load 0x01 into R16 to indicate the carry

forward: ST X+, R1          ; Store the value in R1 (first byte from NUM) into the address
pointed by X, then increment X
ST X, R16                 ; Store the value in R16 (carry flag) into the next address pointed by X
```

```
NOP                       ; No Operation (This is usually used for timing adjustments)
NUM: .db 0x30, 0x88           ; Define the data bytes 0x30 and 0x88 under the label NUM
```

PROBLEM_2:

```
.CSEG                      ; Code Segment - Start of the code section
LDI ZL, LOW (NUM<<1)        ; Load the lower byte of the address of the NUM label into register
ZL
LDI ZH, HIGH (NUM<<1)       ; Load the higher byte of the address of the NUM label into
register ZH
LDI XL, 0x60                ; Load the lower byte of the destination address (0x0060) into
XL (for SRAM)               register
LDI XH, 00                  ; Load the higher byte of the destination address (0x0060) into register
XH

                            ; Load the LSB (Least Significant Byte) of the first number from flash
LPM R1, Z+
into R1
LPM R2, Z+                  ; Load the MSB (Most Significant Byte) of the first number from flash
into R2

LPM R20, Z+                 ; Load the LSB of the second number from flash into R20
LPM R21, Z                  ; Load the MSB of the second number from flash into R21

LDI R16, 0x00               ; Initialize R16 to store the result of the LSB addition
LDI R17, 0x00               ; Initialize R17 to store the result of the MSB addition
LDI R18, 0x00               ; Initialize R18 to store the final carry (if any)

MOV R16, R1                 ; Move the LSB of the first number (R1) into R16
MOV R17, R2                 ; Move the MSB of the first number (R2) into R17

                            ; Add the LSB of the first number (R16) to the LSB of the second
ADD R16, R20
number (R20)
ADC R17, R21                ; Add the MSB of the first number (R17) to the MSB of the second
number (R21), including carry from the previous addition

BRCC forward          ; If the carry flag is clear (no carry), branch to the label 'forward' LDI
R18, 0x01       ; If a carry occurred, load 0x01 into R18 to indicate the carry

forward: ST X+, R16 ; Store the LSB result (R16) in SRAM at the address pointed by X, then
increment X
ST X+, R17                  ; Store the MSB result (R17) in SRAM at the next address pointed by
X, then increment X
ST X, R18                   ; Store the carry flag (R18) in SRAM at the next address pointed by X

NOP                         ; No Operation (used for timing or synchronization purposes)
```

NUM: .db 0xf1, 0x91, 0xb4, 0x37 ; Define the two 16-bit numbers to be added: 0x91F1 and 0x37B4

PROBLEM_3:

.CSEG ; Code Segment - Start of the code section
LDI ZL, LOW (NUM<<1) ; Load the lower byte of the address of the NUM label into register ZL
LDI ZH, HIGH (NUM<<1) ; Load the higher byte of the address of the NUM label into register ZH
LDI XL, 0x60 ; Load the lower byte of the destination address (0x0060) into register XL (for SRAM)
LDI XH, 00 ; Load the higher byte of the destination address (0x0060) into register XH

LPM R18, Z+ ; Load the first 8-bit number from flash into R18, then increment Z to point to the next byte
LPM R19, Z ; Load the second 8-bit number from flash into R19 (Z remains unchanged)

MUL R18, R19 ; Multiply the values in R18 and R19, storing the 16-bit result in R0 (low byte) and R1 (high byte)

ST X+, R0 ; Store the lower byte of the result (R0) in SRAM at the address pointed by X, then increment X
ST X, R1 ; Store the higher byte of the result (R1) in SRAM at the next address pointed by X

NOP ; No Operation (used for timing or synchronization purposes) **NUM: .db 0x5E, 0x1F**

; Define the two 8-bit numbers to be multiplied: 0x5E and 0x1F

PROBLEM_4:

.CSEG ; Code Segment - Start of the code section
LDI ZL, LOW (NUM<<1) ; Load the lower byte of the address of the NUM label into register ZL
LDI ZH, HIGH (NUM<<1) ; Load the higher byte of the address of the NUM label into register ZH
LDI XL, 0x60 ; Load the lower byte of the destination address (0x0060) into register XL (for SRAM)
LDI XH, 00 ; Load the higher byte of the destination address (0x0060) into register XH

LPM R16, Z+ ; Load the first 8-bit number from flash into R16, then increment Z to point to the next byte
LPM R17, Z ; Load the second 8-bit number from flash into R17 (Z remains unchanged)

CP R16, R17 ; Compare R16 with R17 (R16 - R17), setting the condition flags (C, Z, N, V, S, H) based on the result
BRCC forward ; Branch to 'forward' if R16 is greater than or equal to R17 (carry flag is clear)
MOV R16, R17 ; If R16 < R17, move the value in R17 to R16 (R16 now contains the greater number)
forward: ST X, R16 ; Store the greater number (R16) in SRAM at the address pointed by X

NOP ; No Operation (used for timing or synchronization purposes) **NUM : .db 0x1E, 0xBD**

; Define the two 8-bit numbers to be compared: 0x1E and 0xBD

PROCEDURE FOLLOWED:

Step 1: Draw the Flowchart

START -> INITIALIZE VARIABLES -> LOAD VALUES -> PERFORM OPERATIONS -> STORE RESULT -> CHECK FOR END CONDITION -> END.

Step 2: Convert the Flowchart into Assembly Language

Step 3: Compile the Program

Compilling the code and checking for errors.

Step 4: Run and Debug the Program

Use the simulator to monitor register values after each instruction cycle.

For debugging:

- Comparing the register values with expected values
- Utilizing the debugging features in the AVR simulator to step through the program and identify where things go wrong.

Step 5: Verify and Demonstrate

- **Manual Calculation**: Comparing the results obtained from the program with manually calculated results to ensure accuracy.
- Once the program worked as expected, I demonstrated it to the TA.

OBSERVATION:

Problem 1:

- **Observed Output:** The sum of the two 8-bit numbers was correctly stored in the designated register.

- **Carry Flag:** In cases where the sum exceeds 255, the carry flag was set, indicating an overflow.

Problem 2:

- **Carry Handling:** The carry from the lower 8-bit addition was successfully propagated to the higher 8-bit addition.
- **Final Sum:** The 16-bit sum was correctly stored across two registers, with no overflow observed beyond the 16-bit result.

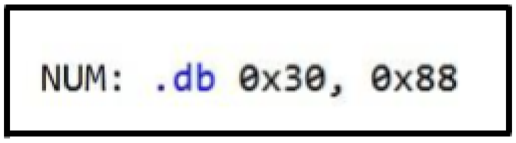
Problem 3:

- **Correct Product Calculation:** The 16-bit product was accurately stored in two registers.
- **Overflow Handling:** No overflow occurred beyond the expected 16-bit result, confirming that the multiplication was handled correctly.

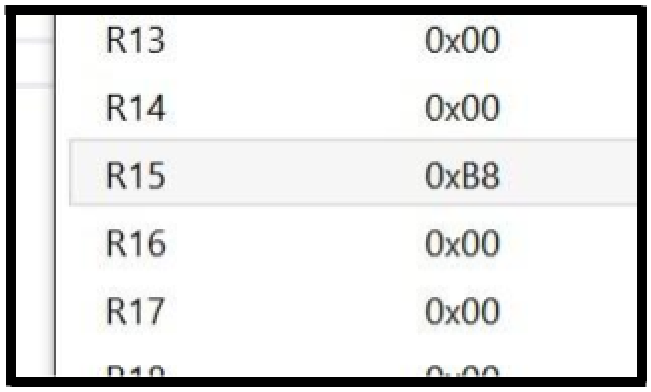
Problem 4:

- **Correct Identification:** The algorithm correctly identified the largest number in the set and stored it in the specified register.
- **Comparison Process:** The comparison logic functioned as expected, with no incorrect comparisons observed during the process.

Problem 1:



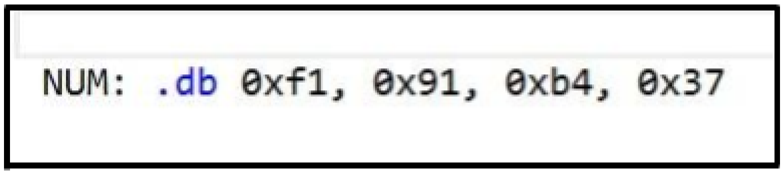
```
NUM: .db 0x30, 0x88
```



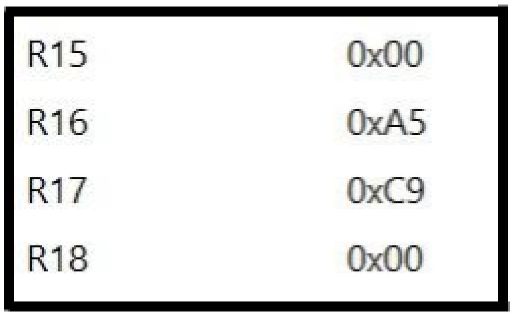
```
R13    0x00
R14    0x00
R15    0xB8
R16    0x00
R17    0x00
```

Here the sum is stored in the R15 register and carry in the R16 register for the given inputs.

Problem 2:



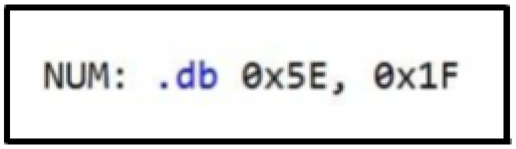
```
NUM: .db 0xf1, 0x91, 0xb4, 0x37
```



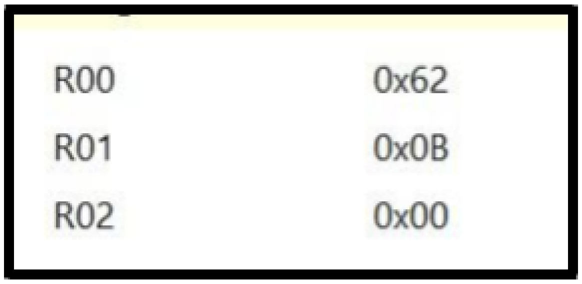
R15	0x00
R16	0xA5
R17	0xC9
R18	0x00

Here the sum of lower bits are stored in the register R16
The other higher bit sums are stored in the register R17
The carry of the 16 bits addition is stored in the register R18, which is 0 in this case.

Problem 3:



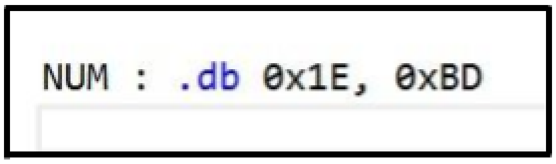
```
NUM: .db 0x5E, 0x1F
```



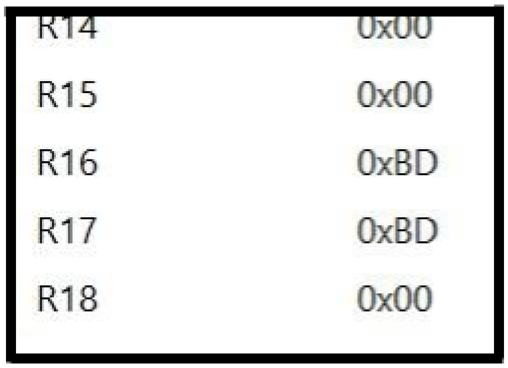
R00	0x62
R01	0x0B
R02	0x00

Here the input values are multiplied and the result is stored in the registers R1:R0

Problem 4:



```
NUM : .db 0x1E, 0xBD
```



R14	0x00
R15	0x00
R16	0xBD
R17	0xBD
R18	0x00

Here the numbers are stored in the registers R16 and R17, after comparison the greatest number will be stored in the register R16.in this case since the R17 value is greater, the R16 register value is replaced by this.

MY CONTRIBUTION:

- I was responsible for the code written for problem_3(8-multiplier) and problem_4(largest of given numbers)
- Tested all the codes with above different inputs and observed the outputs, once in laptop and then in lab.