EXPERIMENT-1 REPORT

KARNIKA SAMYUKTHA C U - EE24B114

AIM

Implement the following basic arithmetic and logical manipulation programs using the Atmel Atmega8 microcontroller in assembly program emulation:

1. (Common 8-bit mathematical operation): Given two 8-bit binary words (byte), compute the sum and store it in a register.

2. (16-bit addition using an 8-bit processor): Given two 16-bit binary words (byte), compute the sum and store it in a register.

3. (Multiplication of two 8-bit numbers): Given two 8-bit binary words (byte), compute the product and store it in a register.

4. (Largest of given numbers): Given f i v e 8-bit binary words, identify the largest number.

APPARATUS REQUIRED

1. Atmel AVR instruction set.

2. A Windows PC with Microchip Studio 7 IDE for developing and debugging AVR microcontroller applications.

3. Basic knowledge of Assembly Programming Language.

GENERAL PROCEDURE

Step 1: Draw the flow chart for the above problem.

Step 2: Convert the flow chart into the assembly language program, identifying the corresponding instructions from the Atmega8 instruction set.

Step 3: Get it compiled by the AVR compiler.

Step 4: Run the program and note down the values of the register for each cycle.

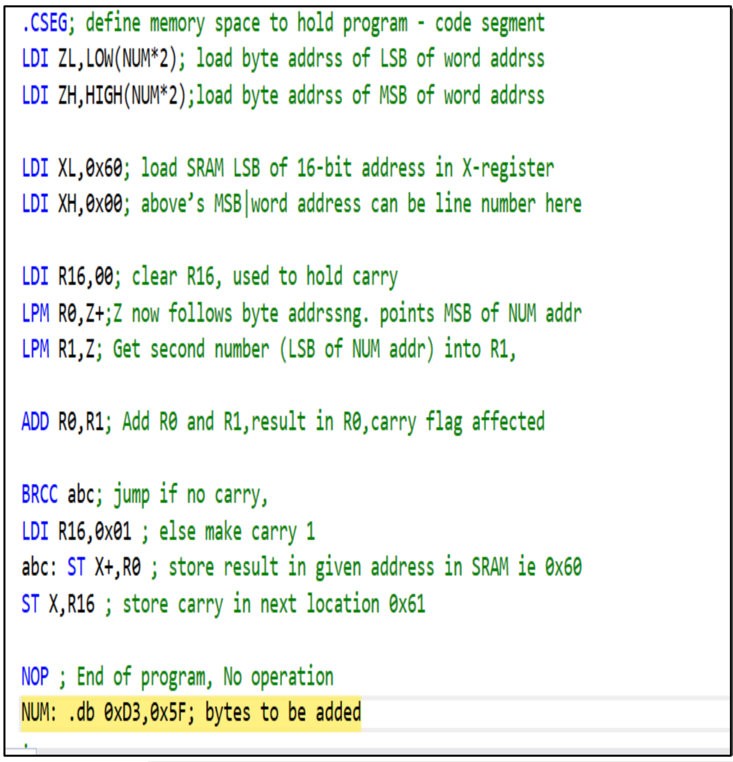
Step 5: Verify the result with the manually calculated answer. If the resultant word does not match, debug the code until the correct result is obtained.

IMPLEMENTATION OF PROGRAMS

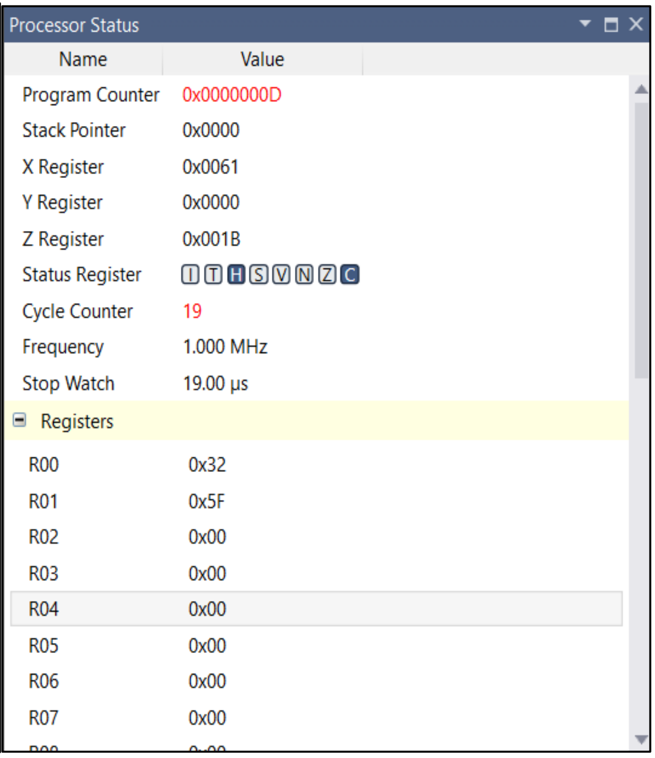
1. Sum of two 8 Bit Numbers

ADD Rd, Rr → Adds two registers, stores result in Rd, and affects flags in SREG (Status Register)
ADC Rd, Rr → Adds two registers with carry (used for multi-byte addition).

The Result of adding numbers in R0,R1 is stored in R0

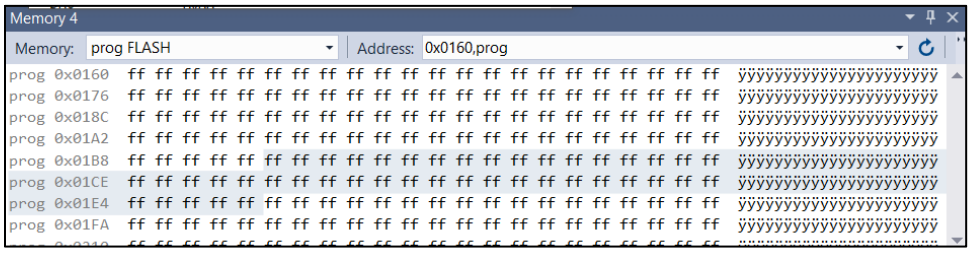


```
.CSEG; define memory space to hold program - code segment
LDI ZL,LOW(NUM*2); load byte addrss of LSB of word addrss
LDI ZH,HIGH(NUM*2);load byte addrss of MSB of word addrss

LDI XL,0x60; load SRAM LSB of 16-bit address in X-register
LDI XH,0x00; above’s MSB|word address can be line number here

LDI R16,00; clear R16, used to hold carry
LPM R0,Z+;Z now follows byte addrssng. points MSB of NUM addr
LPM R1,Z; Get second number (LSB of NUM addr) into R1,

ADD R0,R1; Add R0 and R1,result in R0,carry flag affected

BRCC abc; jump if no carry,
LDI R16,0x01 ; else make carry 1
abc: ST X+,R0 ; store result in given address in SRAM ie 0x60
ST X,R16 ; store carry in next location 0x61

NOP ; End of program, No operation
NUM: .db 0xD3,0x5F; bytes to be added
```



Processor Status

Name	Value
Program Counter	0x0000000D
Stack Pointer	0x0000
X Register	0x0061
Y Register	0x0000
Z Register	0x001B
Status Register	I T H S V N Z C
Cycle Counter	19
Frequency	1.000 MHz
Stop Watch	19.00 µs
Registers	
R00	0x32
R01	0x5F
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00



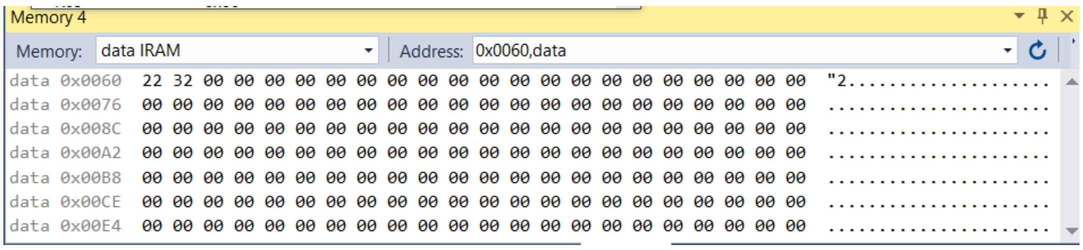
Memory 4

Memory: prog FLASH Address: 0x0160,prog

```
prog 0x0160  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x0176  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x018C  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x01A2  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x01B8  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x01CE  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x01E4  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
prog 0x01FA  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

2. Sum of two 16 Bit Numbers

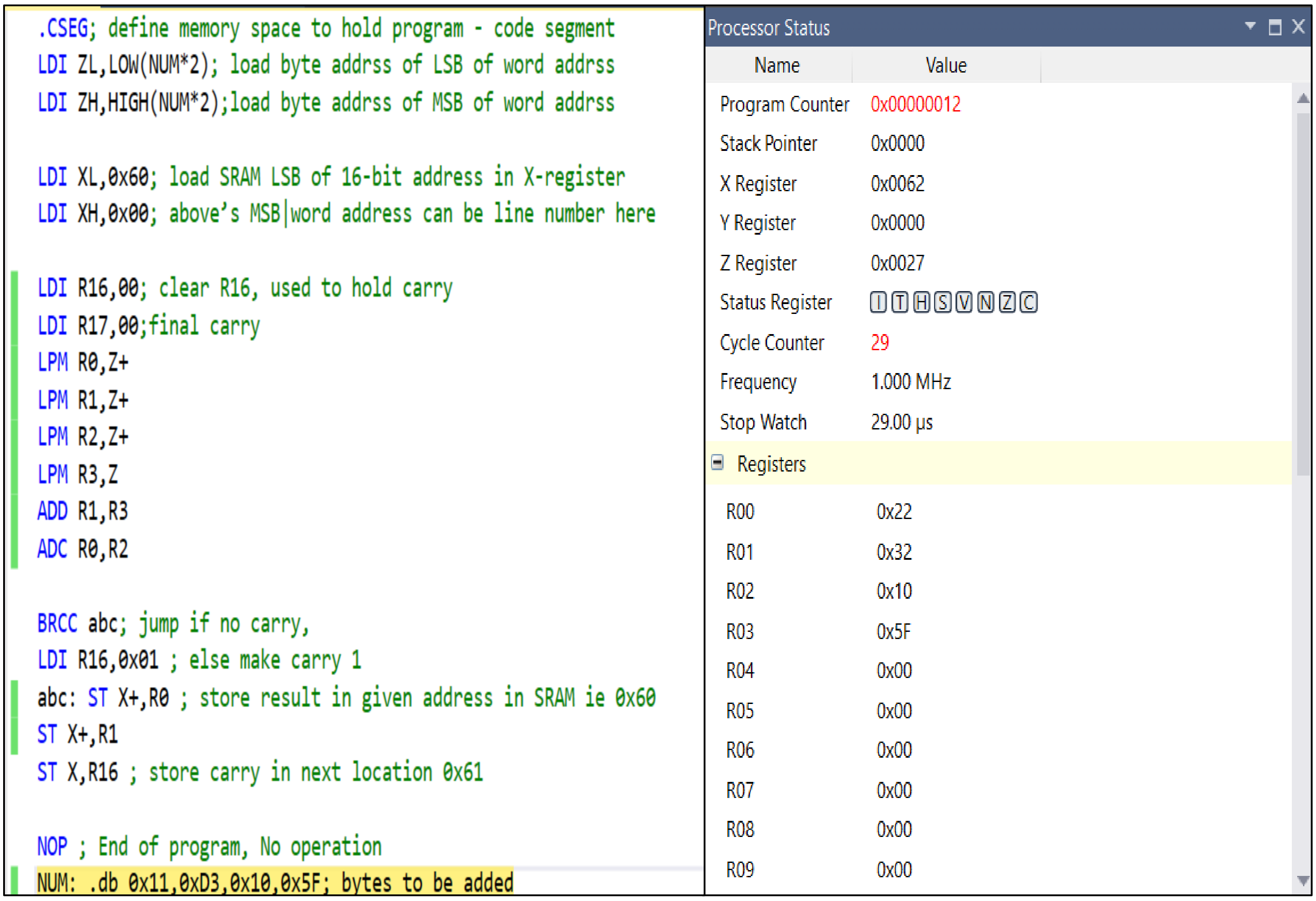
- The 16-bit addition code adds the low bytes first (ADD R0, R2), which updates the carry flag, then adds the high bytes with carry (ADC R1, R3) so the overflow from the low-byte addition is included.
- The final 16-bit sum is stored in R1:R0



Memory 4

Memory: data IRAM Address: 0x0060,data

```
data 0x0060  22 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  "2....................
data 0x0076  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
data 0x008C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
data 0x00A2  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
data 0x00B8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
data 0x00CE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
data 0x00E4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
```

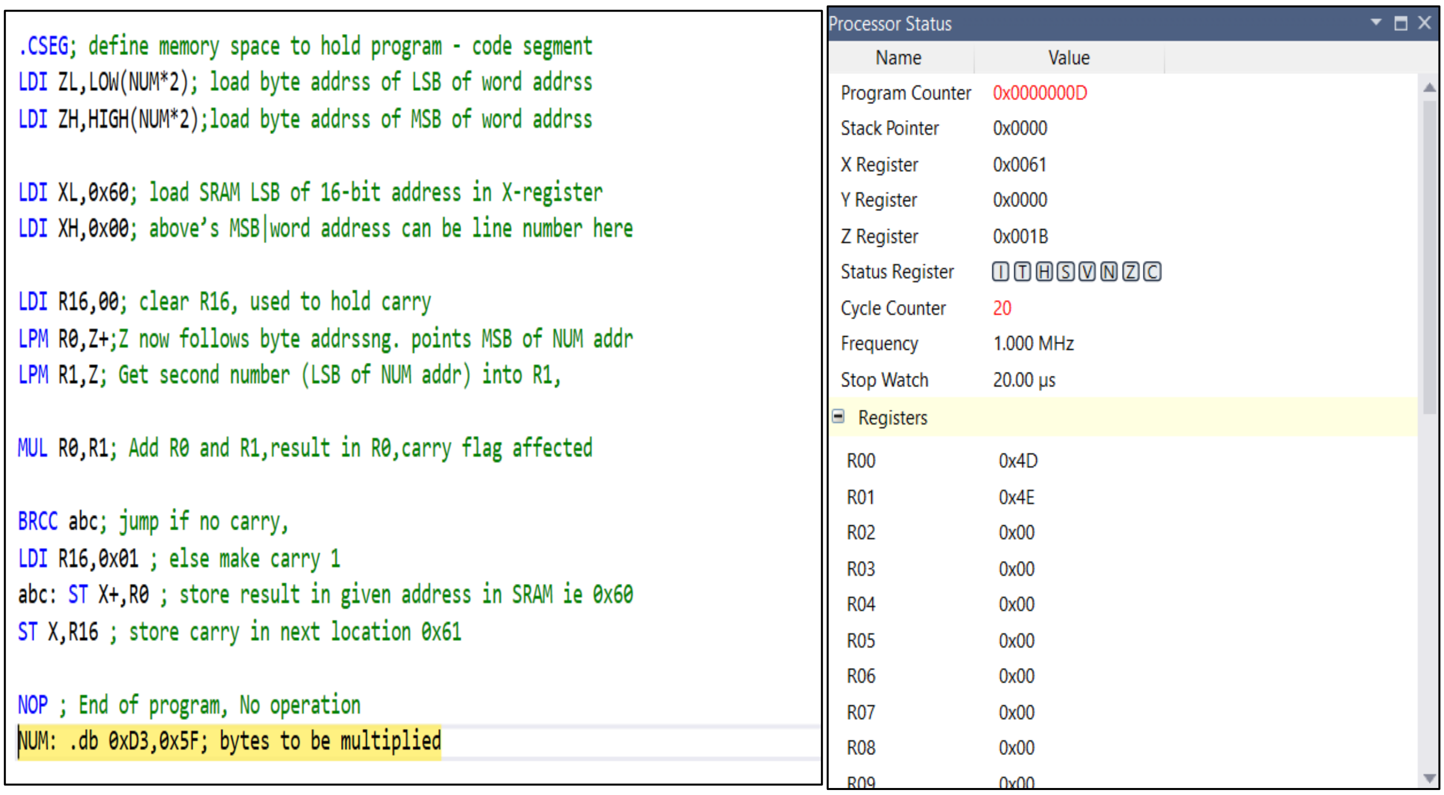


```
.CSEG; define memory space to hold program - code segment
LDI ZL,LOW(NUM*2); load byte addrss of LSB of word addrss
LDI ZH,HIGH(NUM*2);load byte addrss of MSB of word addrss

LDI XL,0x60; load SRAM LSB of 16-bit address in X-register
LDI XH,0x00; above's MSB|word address can be line number here

LDI R16,00; clear R16, used to hold carry
LDI R17,00;final carry
LPM R0,Z+
LPM R1,Z+
LPM R2,Z+
LPM R3,Z
ADD R1,R3
ADC R0,R2

BRCC abc; jump if no carry,
LDI R16,0x01 ; else make carry 1
abc: ST X+,R0 ; store result in given address in SRAM ie 0x60
ST X+,R1
ST X,R16 ; store carry in next location 0x61

NOP ; End of program, No operation
NUM: .db 0x11,0xD3,0x10,0x5F; bytes to be added
```

Processor Status

Name	Value
Program Counter	0x00000012
Stack Pointer	0x0000
X Register	0x0062
Y Register	0x0000
Z Register	0x0027
Status Register	I T H S V N Z C
Cycle Counter	29
Frequency	1.000 MHz
Stop Watch	29.00 µs
Registers	
R00	0x22
R01	0x32
R02	0x10
R03	0x5F
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00

3. Multiplication of two 8 Bit Numbers
   - Two 8 bit numbers, 0XD3, 0X5F are multiplied through MUL function
   - The result is stored in registers R0,R1: 0x4E4D



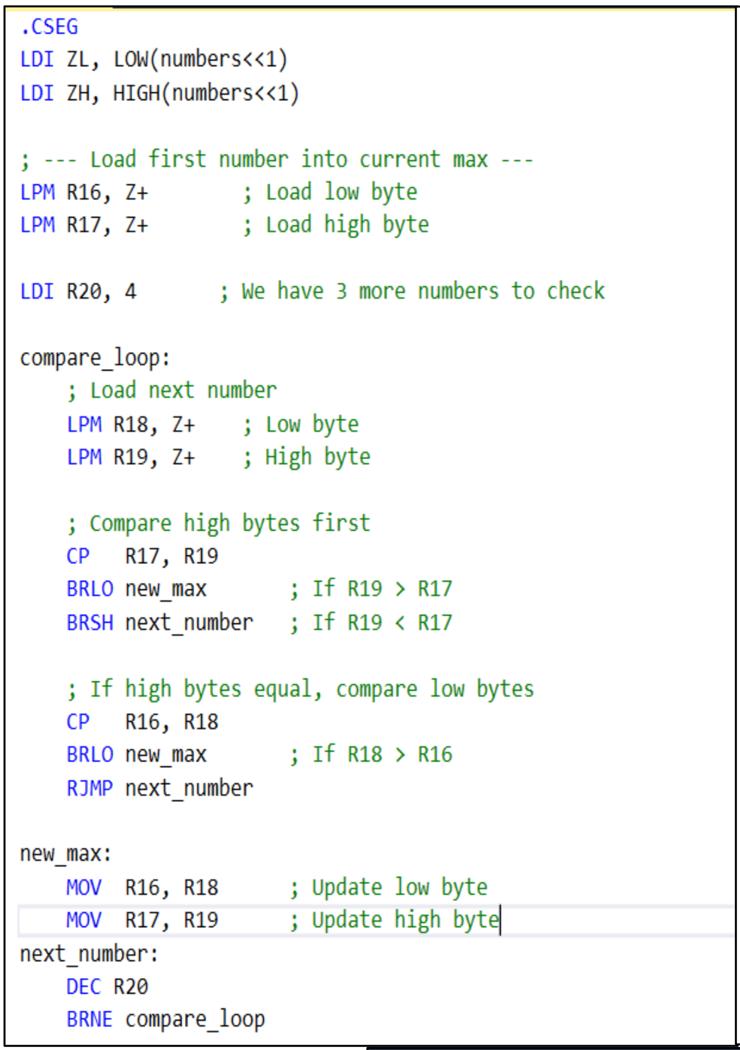
```
.CSEG; define memory space to hold program - code segment
LDI ZL,LOW(NUM*2); load byte addrss of LSB of word addrss
LDI ZH,HIGH(NUM*2);load byte addrss of MSB of word addrss

LDI XL,0x60; load SRAM LSB of 16-bit address in X-register
LDI XH,0x00; above's MSB|word address can be line number here

LDI R16,00; clear R16, used to hold carry
LPM R0,Z+;Z now follows byte addrssng. points MSB of NUM addr
LPM R1,Z; Get second number (LSB of NUM addr) into R1,

MUL R0,R1; Add R0 and R1,result in R0,carry flag affected

BRCC abc; jump if no carry,
LDI R16,0x01 ; else make carry 1
abc: ST X+,R0 ; store result in given address in SRAM ie 0x60
ST X,R16 ; store carry in next location 0x61

NOP ; End of program, No operation
NUM: .db 0xD3,0x5F; bytes to be multiplied
```

Processor Status

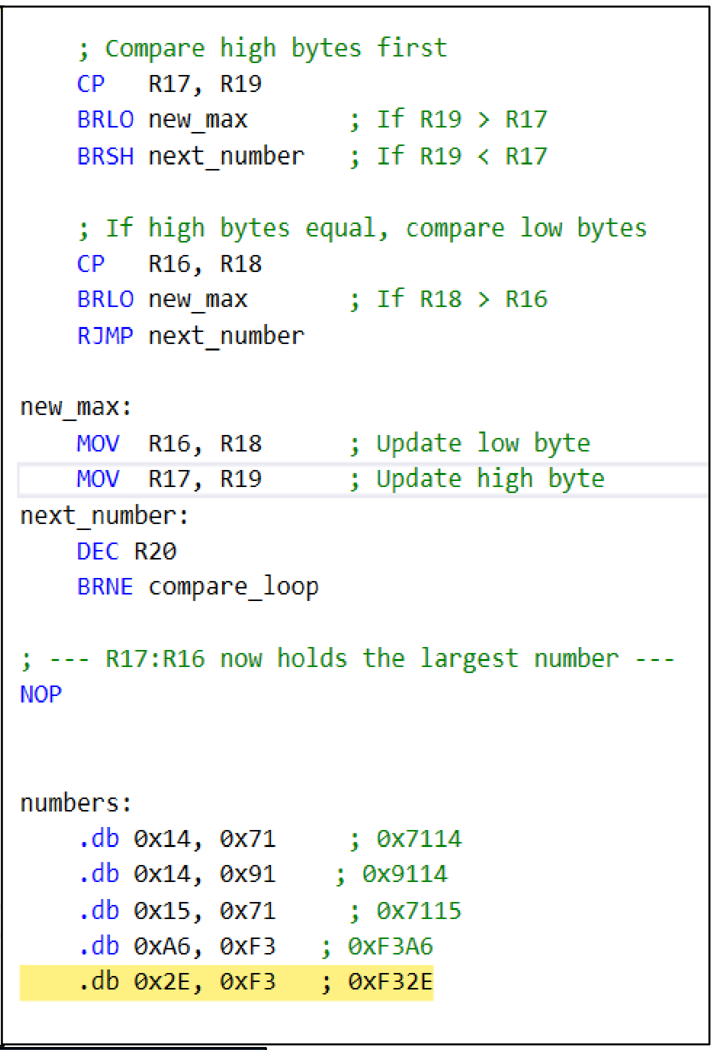
Name	Value
Program Counter	0x0000000D
Stack Pointer	0x0000
X Register	0x0061
Y Register	0x0000
Z Register	0x001B
Status Register	I T H S V N Z C
Cycle Counter	20
Frequency	1.000 MHz
Stop Watch	20.00 µs
Registers	
R00	0x4D
R01	0x4E
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00

4. Comparision of five 16 Bit Numbers

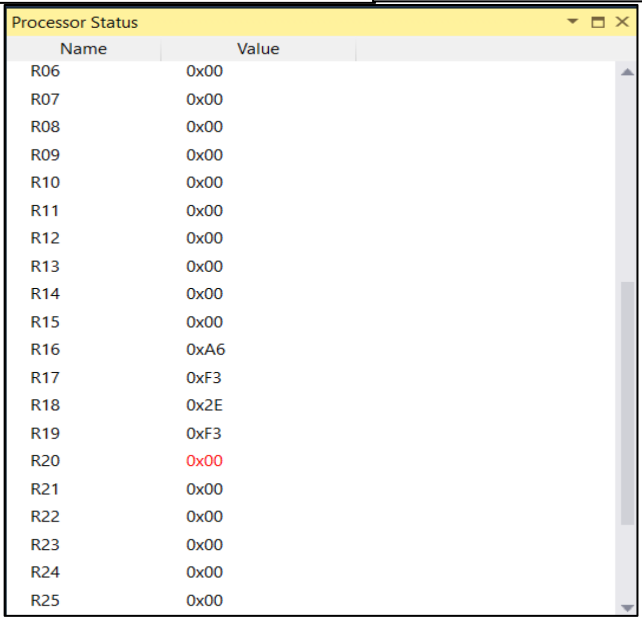
- The 5-number comparison code works by storing the first number as the current maximum, then sequentially comparing each of the remaining four numbers with it using CP (compare) and a conditional branch (BRLO or BRSH) to update the maximum register only if the new number is larger.
- After all comparisons, the register holds(R17:R16) the largest value



```
.CSEG
LDI ZL, LOW(numbers<<1)
LDI ZH, HIGH(numbers<<1)

; --- Load first number into current max ---
LPM R16, Z+        ; Load low byte
LPM R17, Z+        ; Load high byte

LDI R20, 4       ; We have 3 more numbers to check

compare_loop:
    ; Load next number
    LPM R18, Z+    ; Low byte
    LPM R19, Z+    ; High byte

    ; Compare high bytes first
    CP   R17, R19
    BRLO new_max       ; If R19 > R17
    BRSH next_number   ; If R19 < R17

    ; If high bytes equal, compare low bytes
    CP   R16, R18
    BRLO new_max       ; If R18 > R16
    RJMP next_number

new_max:
    MOV  R16, R18      ; Update low byte
    MOV  R17, R19      ; Update high byte
next_number:
    DEC R20
    BRNE compare_loop
```



```
    ; Compare high bytes first
    CP   R17, R19
    BRLO new_max       ; If R19 > R17
    BRSH next_number   ; If R19 < R17

    ; If high bytes equal, compare low bytes
    CP   R16, R18
    BRLO new_max       ; If R18 > R16
    RJMP next_number

new_max:
    MOV  R16, R18      ; Update low byte
    MOV  R17, R19      ; Update high byte
next_number:
    DEC R20
    BRNE compare_loop

; --- R17:R16 now holds the largest number ---
NOP


numbers:
    .db 0x14, 0x71     ; 0x7114
    .db 0x14, 0x91    ; 0x9114
    .db 0x15, 0x71     ; 0x7115
    .db 0xA6, 0xF3   ; 0xF3A6
    .db 0x2E, 0xF3   ; 0xF32E
```



Processor Status

Name	Value
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0xA6
R17	0xF3
R18	0x2E
R19	0xF3
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x00
R25	0x00