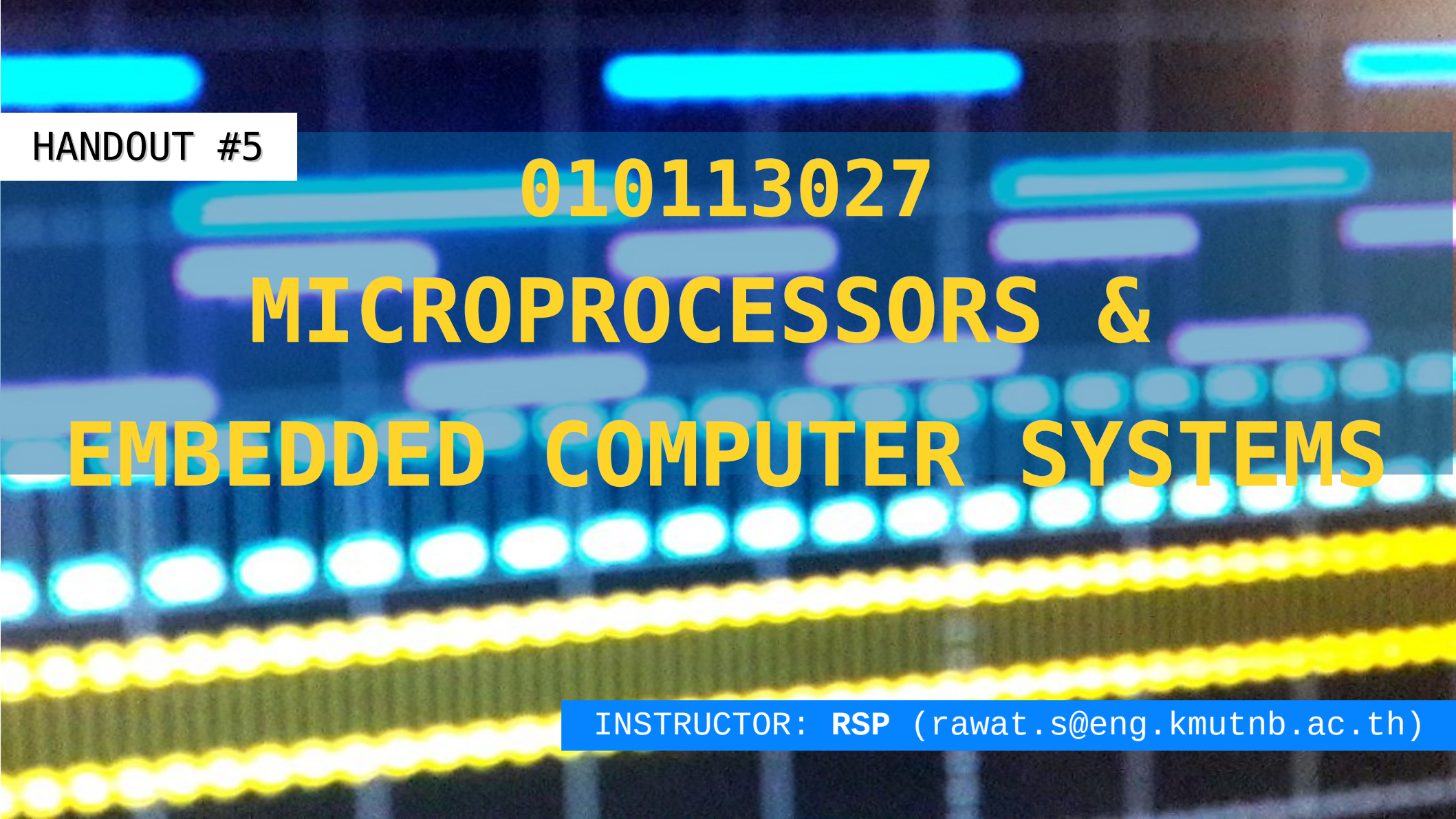


HANDOUT #5

010113027
MICROPROCESSORS & EMBEDDED COMPUTER SYSTEMS

INSTRUCTOR: **RSP** (rawat.s@eng.kmutnb.ac.th)

Key Topics: AVR Assembly Programming

- Arithmetic & Logical Operations on Single or Multiple Data Bytes
- Control Flow: Branches, Skips, Loops, Bit Checks, Comparisons
- Storing and Accessing Data in SRAM and Flash
- Subroutine Calls
- Stack Operations
- C Code vs. Assembly Code and Disassembly
- Inline AVR Assembly in C Code

Example 1: ALU Operations and SREG Flags

```
.CSEG           ; Code segment (Flash memory)
.ORG 0x0000     ; Start at reset vector (word address = 0x0000)
.DEF A = R16    ; define A as R16
.DEF B = R17    ; define B as R17

    LDI A, 255  ; A=255 (unsigned) = -1 (signed)
    LDI B, 1    ; B=1   (unsigned) = +1 (signed)
    ADD A, B    ; A=A+B
                ; Result: A=0x00, Flags: V=0, N=0, Z=1, C=1, S=0

    LDI A, 0x7F ; A=127 (unsigned) = +127 (signed)
    LDI B, 1    ; B=1   (unsigned) = +1   (signed)
    ADD A, B    ; A=A+B
                ; Result: A=0x80, Flags: V=1, N=1, Z=0, C=0, S=0

    LDI A, 0x7F ; A=127 (unsigned) = +127 (signed)
    LDI B, 0x80 ; B=128 (unsigned) = -128 (signed)
    ADD A, B    ; A=A+B
                ; Result: A=0xFF, Flags: V=0, N=1, Z=0, C=0, S=1

    LDI A, 0xFF ; A=255 (unsigned) = -1 (signed)
    LDI B, 0xFF ; B=255 (unsigned) = -1 (signed)
    ADD A, B    ; A=A+B
                ; Result: A=0xFE, Flags: V=0, N=1, Z=0, C=1, S=1
    CLR A       ; Clear register A
    EOR B,B     ; Clear register B (using exclusive OR)
    OUT SREG, A ; Clear status register

DONE: RJMP DONE
```

$S = N \oplus V$

AVR Flags: N / V / S Flags

Zero (Z) Flag

- It indicates the result is zero.

Carry (C) Flag

- It indicates unsigned overflow.
  - Addition: carry out of MSB
  - Subtraction: borrow occurred
  - Shift/rotate: bit shifted out

Negative (N) Flag

- N = bit7 of the result (MSB)
- It indicates whether the result “appears” negative when interpreted as signed.

Signed Overflow (V) Flag

- Signed overflow occurs when performing an operation such as ADD or SUB:
  - Two positive numbers produce a negative result.
  - Two negative numbers produce a positive result.

Sign (S) Flag

- S = N ⊕ V
- It represents the true sign of the result for signed arithmetic.
- The S (Sign) flag is used only for signed comparisons and branches.
  - Normally used after: CP, CPC, SUB, SBC.
  - Branch instructions that use S: BRLT, BRGE
  - Branch instructions that use S and Z: BRGT, BRLE.

AVR Flags: N / V / S Flags

```
   LDI A, 0x7F ; A=127 (unsigned) = +127 (signed)
   LDI B, 1    ; B=1   (unsigned) = +1   (signed)
   ADD A, B    ; A=A+B
               ; Result: A=0x80, Flags: V=1, N=1, Z=0, C=0, S=0
```

```
  0b0111_1111 (A = 127)
+ 0b0000_0001 (B = 1)
  -----------------
  0b1000_0000 (+128 unsigned or -128 signed)
```

C=0: no carry out

V=1: positive + positive => negative (signed overflow)

N=1: MSB is 1 (the result appears negative).

S=0: the result should be considered positive (+128), not negative (-128).

AVR Flags: N / V / S Flags

```
  LDI A, 0x7F ; A=127 (unsigned) = +127 (signed)
  LDI B, 0x80 ; B=128 (unsigned) = -128 (signed)
  ADD A, B    ; A=A+B
              ; Result: A=0xFF, Flags: V=0, N=1, Z=0, C=0, S=1
```

```
  0b0111_1111 (A = 127)
+ 0b1000_0000 (B = 128 unsigned or -128 signed)
  -----------------
  0b1111_1111 (+255 unsigned or -1 signed)
```

C=0: No carry out

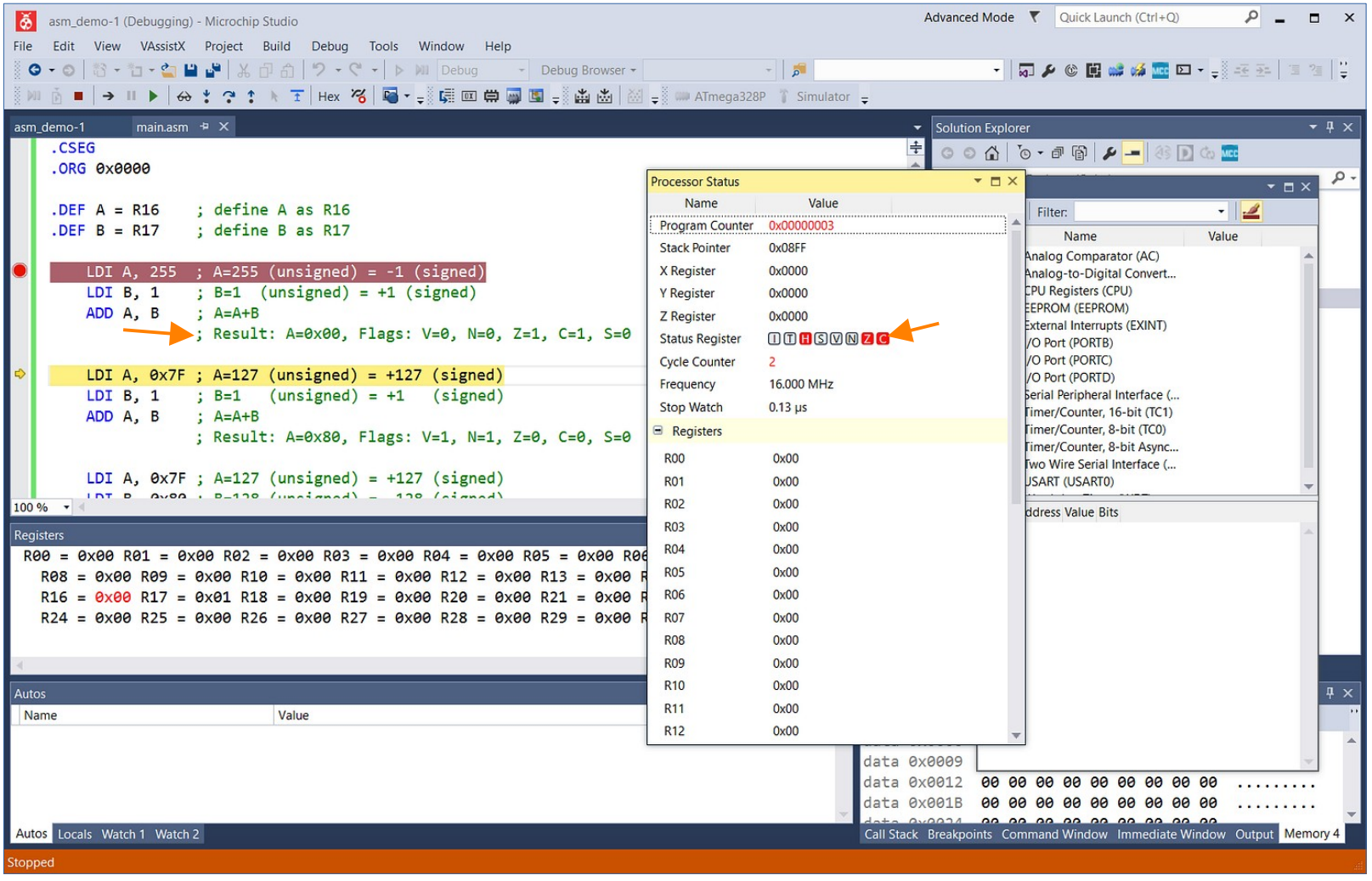
V=0: positive + negative => negative (no signed overflow)

N=1: MSB is 1 (the result appears negative).

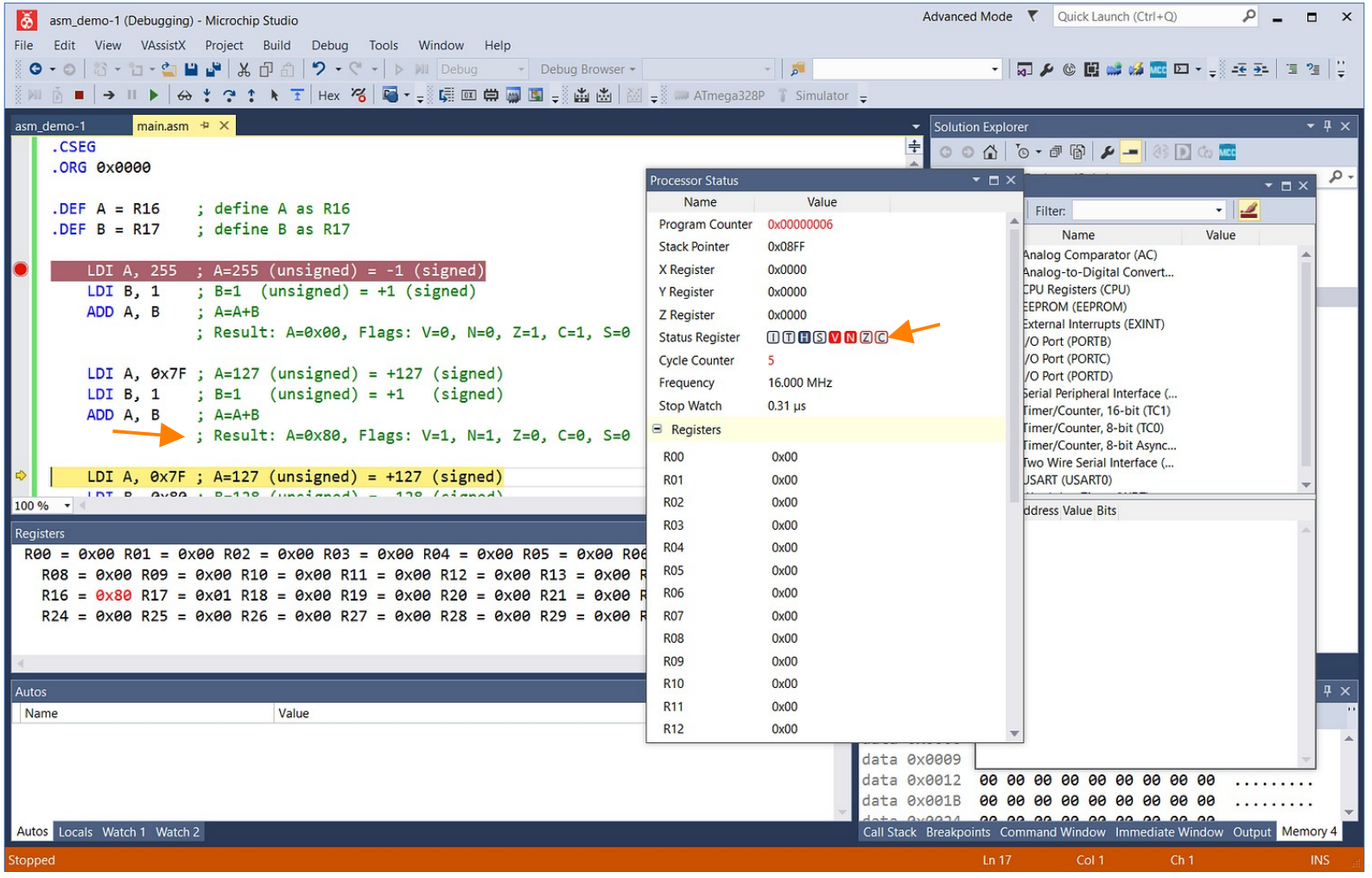
S=1: the result can be considered negative (-1) for signed operation OR positive (255) for unsigned operation.

Simulation

Use “Step Over” to execute the instructions and observe the SREG.



Simulation


asm_demo-1 (Debugging) - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
asm_demo-1
main.asm
.CSEG
.ORG 0x0000
.DEF A = R16 ; define A as R16
.DEF B = R17 ; define B as R17
LDI A, 255 ; A=255 (unsigned) = -1 (signed)
LDI B, 1 ; B=1 (unsigned) = +1 (signed)
ADD A, B ; A=A+B
; Result: A=0x00, Flags: V=0, N=0, Z=1, C=1, S=0
LDI A, 0x7F ; A=127 (unsigned) = +127 (signed)
LDI B, 1 ; B=1 (unsigned) = +1 (signed)
ADD A, B ; A=A+B
; Result: A=0x80, Flags: V=1, N=1, Z=0, C=0, S=0
LDI A, 0x7F ; A=127 (unsigned) = +127 (signed)
Processor Status
Name Value
Program Counter 0x00000006
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register I T H S V N Z C
Cycle Counter 5
Frequency 16.000 MHz
Stop Watch 0.31 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
Solution Explorer
Filter:
Name Value
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Address Value Bits
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00
R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00
R16 = 0x80 R17 = 0x01 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00
R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00
Autos
Name Value
data 0x0009
data 0x0012 00 00 00 00 00 00 00 00 00 .....
data 0x001B 00 00 00 00 00 00 00 00 00 .....
Autos Locals Watch 1 Watch 2
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 17 Col 1 Ch 1 INS

Example 2: ALU Operations and SREG Flags

```
.CSEG
.ORG 0x0000
.DEF A = R16      ; define A as R16
.DEF B = R17      ; define B as R17

    LDI A, 1      ; A=1
    LDI B, 1      ; B=1
    SUB A, B      ; A=A-B
                  ; Result: A=0x00, V=0, N=0, Z=1, C=0, S=0

    LDI A, 0x00   ; A=0
    LDI B, 0x01   ; B=1
    SUB A, B      ; A=A-B
                  ; Result: A=0xFF, V=0, N=1, Z=0, C=1 (borrow), S=1

    LDI A, 0x00   ; A=0   (unsigned) = +0 (signed)
    LDI B, 0xFF   ; B=255 (unsigned) = -1 (signed)
    SUB A, B      ; A=A-B
                  ; Result: A=0x01, V=0, N=0, Z=0, C=1 (borrow), S=0

    LDI A, 0x00   ; A=0   (unsigned) = +0   (signed)
    LDI B, 0x80   ; B=128 (unsigned) = -128 (signed)
    SUB A, B      ; A=A-B
                  ; Result: A=0x80, V=1, N=1, Z=0, C=1 (borrow), S=0
    CLZ           ; Clear the Z flag
    CLC           ; Clear the C flag
    CLN           ; Clear the N flag
    CLV           ; Clear the V flag
DONE: RJMP DONE
```

AVR Flags: N / V / S Flags

```
LDI A, 0x00  ; A=0
LDI B, 0x01  ; B=1
SUB A, B     ; A=A-B
             ; Result: A=0xFF, V=0, N=1, Z=0, C=1 (borrow), S=1
```

```
  0b0000_0000 (A = 0)
- 0b0000_0001 (B = 1)
  -----------------
  0b1111_1111 (+255 unsigned or -1 signed)
```

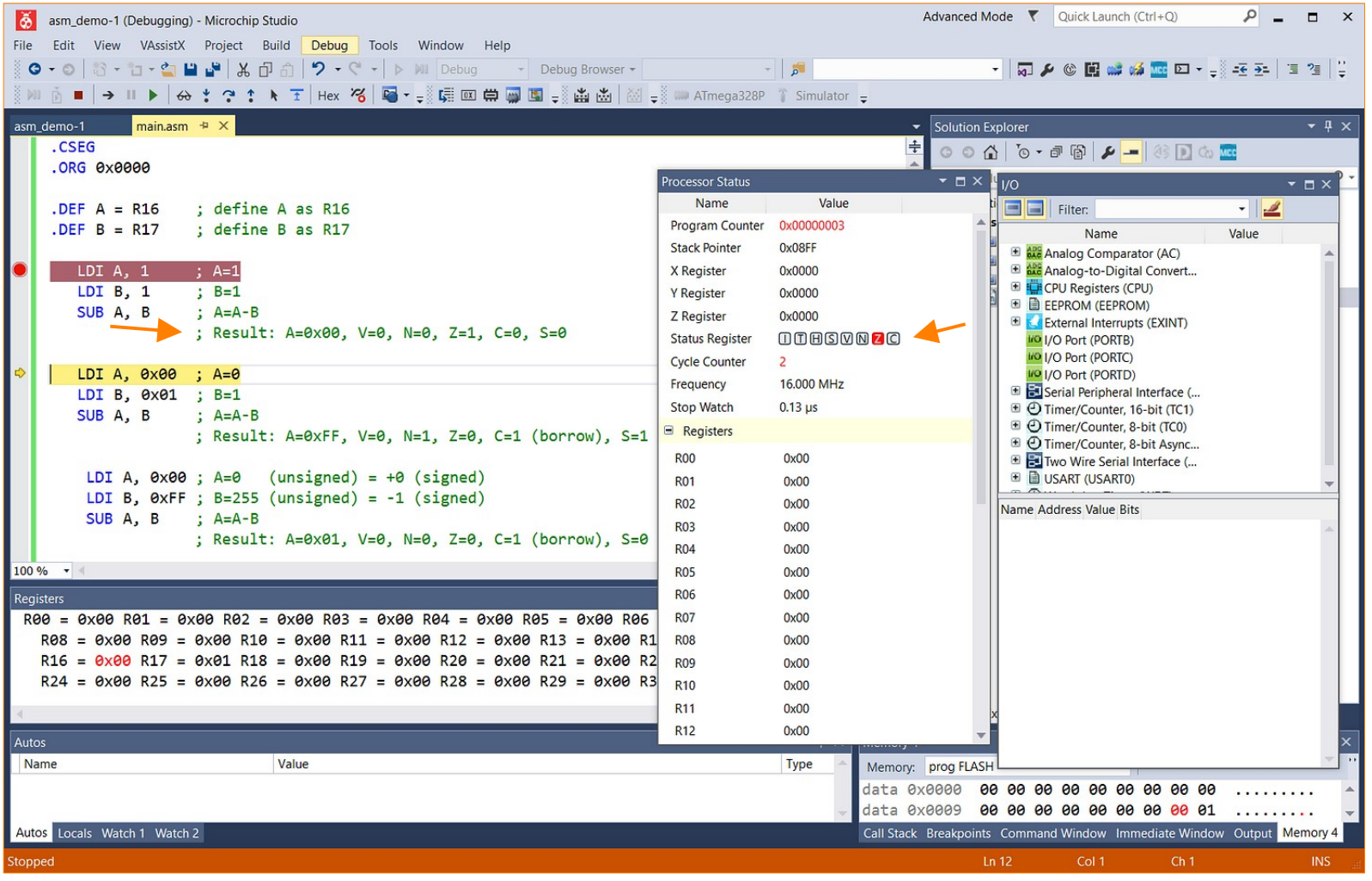
C=1: Borrow (for unsigned operation)

V=0: positive - positive => negative (no signed overflow)

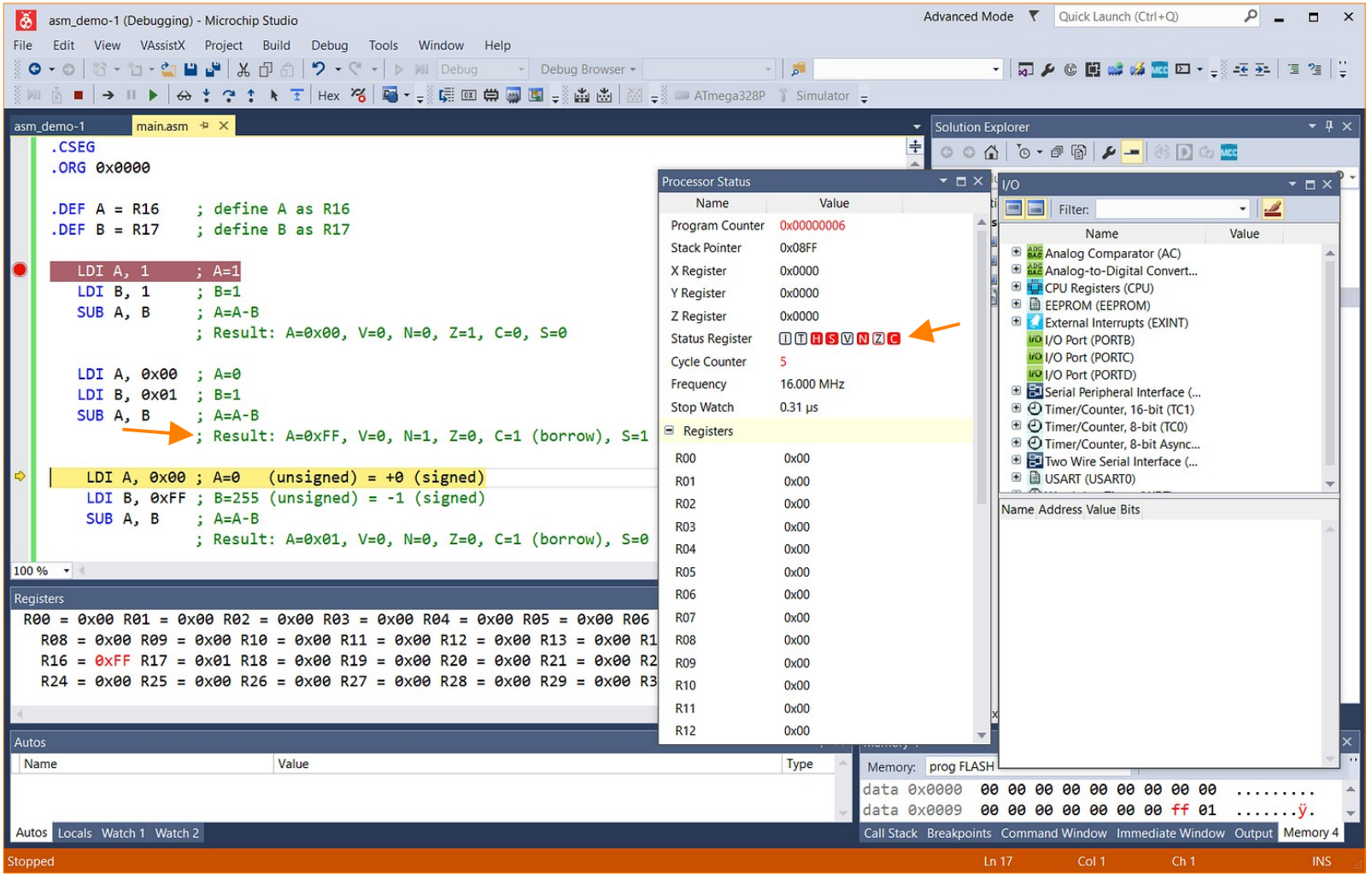
N=1: MSB is 1 (the result appears negative).

S=1: the result should be considered negative (-1) for signed operation.

Simulation


asm_demo-1 (Debugging) - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
asm_demo-1
main.asm
.CSEG
.ORG 0x0000
.DEF A = R16 ; define A as R16
.DEF B = R17 ; define B as R17
LDI A, 1 ; A=1
LDI B, 1 ; B=1
SUB A, B ; A=A-B
; Result: A=0x00, V=0, N=0, Z=1, C=0, S=0
LDI A, 0x00 ; A=0
LDI B, 0x01 ; B=1
SUB A, B ; A=A-B
; Result: A=0xFF, V=0, N=1, Z=0, C=1 (borrow), S=1
LDI A, 0x00 ; A=0 (unsigned) = +0 (signed)
LDI B, 0xFF ; B=255 (unsigned) = -1 (signed)
SUB A, B ; A=A-B
; Result: A=0x01, V=0, N=0, Z=0, C=1 (borrow), S=0
Processor Status
Program Counter 0x00000003
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register
Cycle Counter 2
Frequency 16.000 MHz
Stop Watch 0.13 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
Solution Explorer
I/O
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06
R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R1
R16 = 0x00 R17 = 0x01 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R2
R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00 R3
Autos
Name Value Type
Memory: prog FLASH
data 0x0000 00 00 00 00 00 00 00 00 00
data 0x0009 00 00 00 00 00 00 00 00 01
Autos Locals Watch 1 Watch 2
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 12 Col 1 Ch 1 INS

Simulation


asm_demo-1 (Debugging) - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
asm_demo-1
main.asm
.CSEG
.ORG 0x0000
.DEF A = R16 ; define A as R16
.DEF B = R17 ; define B as R17
LDI A, 1 ; A=1
LDI B, 1 ; B=1
SUB A, B ; A=A-B
; Result: A=0x00, V=0, N=0, Z=1, C=0, S=0
LDI A, 0x00 ; A=0
LDI B, 0x01 ; B=1
SUB A, B ; A=A-B
; Result: A=0xFF, V=0, N=1, Z=0, C=1 (borrow), S=1
LDI A, 0x00 ; A=0 (unsigned) = +0 (signed)
LDI B, 0xFF ; B=255 (unsigned) = -1 (signed)
SUB A, B ; A=A-B
; Result: A=0x01, V=0, N=0, Z=0, C=1 (borrow), S=0
Processor Status
Name Value
Program Counter 0x00000006
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register I T H S V N Z C
Cycle Counter 5
Frequency 16.000 MHz
Stop Watch 0.31 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
Solution Explorer
I/O
Filter:
Name Value
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Name Address Value Bits
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06
R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R1
R16 = 0xFF R17 = 0x01 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R2
R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00 R3
Autos
Name Value Type
Memory: prog FLASH
data 0x0000 00 00 00 00 00 00 00 00 00 .....
data 0x0009 00 00 00 00 00 00 00 ff 01 .....ÿ.
Autos Locals Watch 1 Watch 2
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 17 Col 1 Ch 1 INS

Example 3: AVR Assembly Code vs. Pseudo C Code

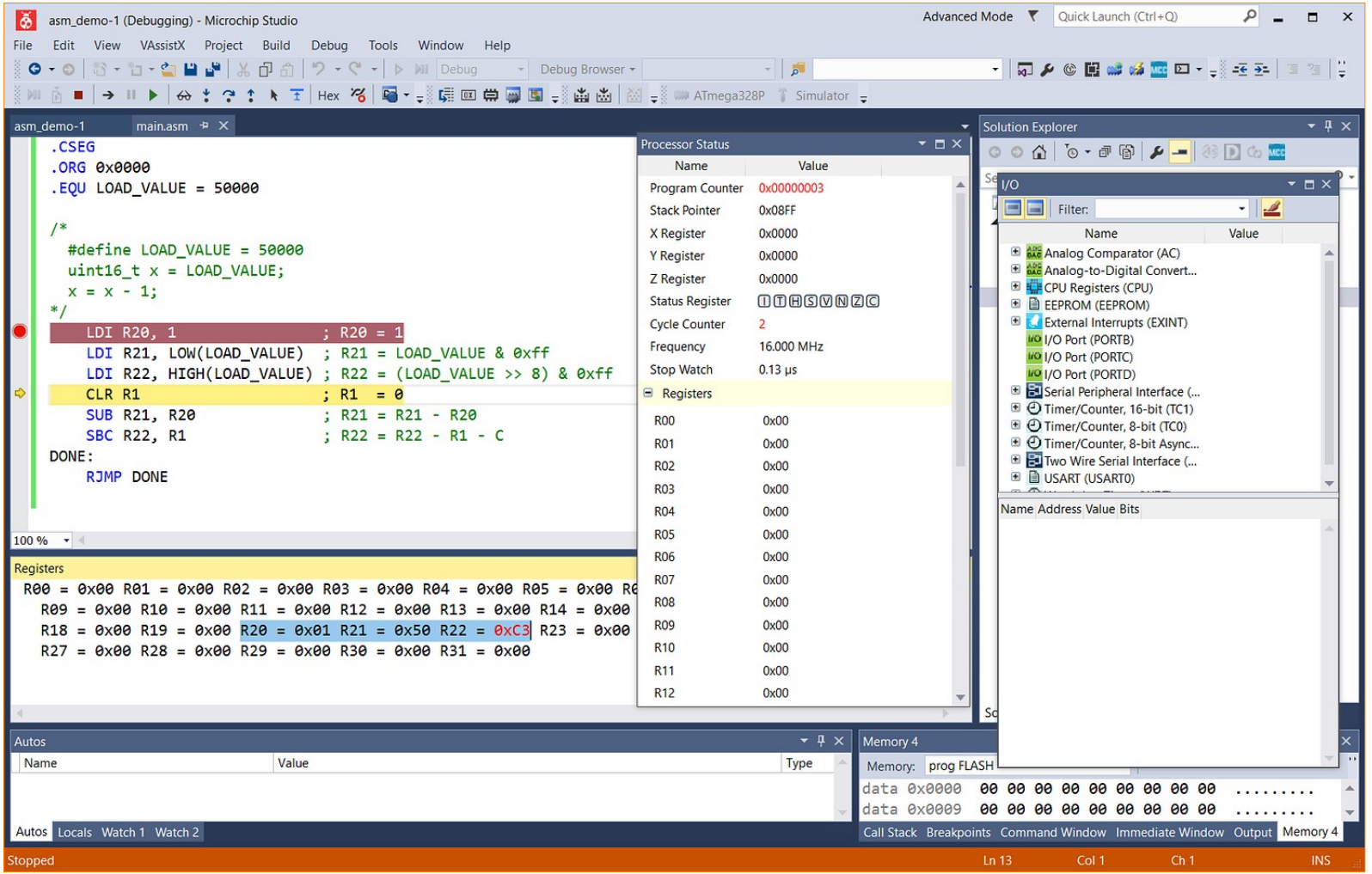
```
.CSEG
.ORG 0x0000
.EQU LOAD_VALUE = 50000      ; define a constant 0xC350 (hex)

/*
  #define LOAD_VALUE = 50000
  uint16_t x = LOAD_VALUE;
  x = x - 1;
*/
    LDI R20, 1                ; R20 = 1
    LDI R21, LOW(LOAD_VALUE)  ; R21 = low byte of LOAD_VALUE
    LDI R22, HIGH(LOAD_VALUE) ; R22 = high byte of LOAD_VALUE
    CLR R1                    ; R1  = 0
    SUB R21, R20              ; R21 = R21 - R20
    SBC R22, R1               ; R22 = R22 - R1 - C
DONE:
    RJMP DONE
```

Note: In this example, the variable **x** (16-bit unsigned) in C code is stored in R22:R21, not stored in SRAM.

50 000 (dec) = 0xC350 (hex)

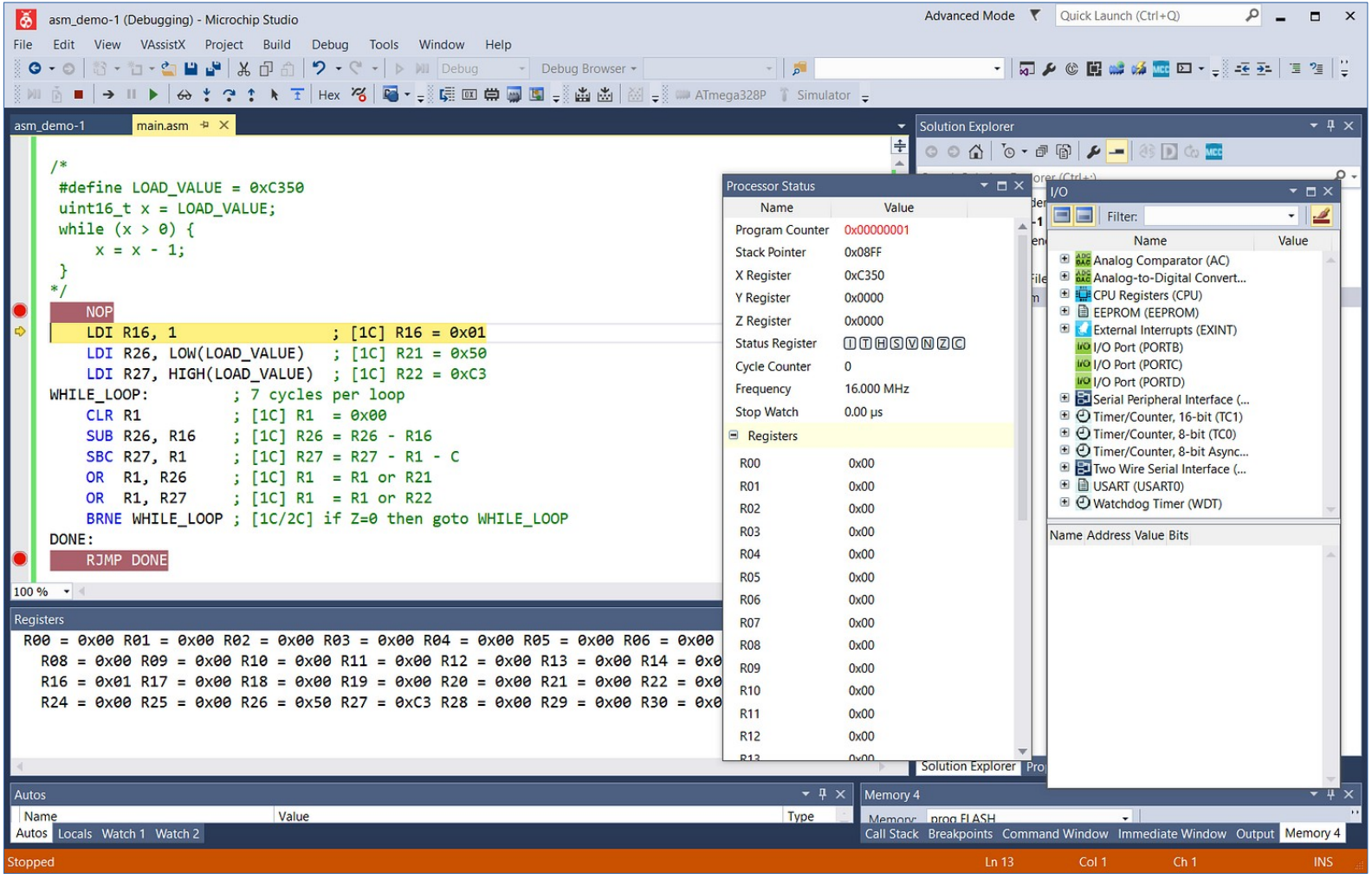
Simulation


asm_demo-1 (Debugging) - Microchip Studio
Advanced Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
ATmega328P
Simulator
asm_demo-1
main.asm
.CSEG
.ORG 0x0000
.EQU LOAD_VALUE = 50000
/*
#define LOAD_VALUE = 50000
uint16_t x = LOAD_VALUE;
x = x - 1;
*/
LDI R20, 1 ; R20 = 1
LDI R21, LOW(LOAD_VALUE) ; R21 = LOAD_VALUE & 0xff
LDI R22, HIGH(LOAD_VALUE) ; R22 = (LOAD_VALUE >> 8) & 0xff
CLR R1 ; R1 = 0
SUB R21, R20 ; R21 = R21 - R20
SBC R22, R1 ; R22 = R22 - R1 - C
DONE:
RJMP DONE
100 %
Processor Status
Name Value
Program Counter 0x00000003
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register ITHSVNZC
Cycle Counter 2
Frequency 16.000 MHz
Stop Watch 0.13 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
Solution Explorer
I/O
Filter:
Name Value
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Name Address Value Bits
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R0
R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x00
R18 = 0x00 R19 = 0x00 R20 = 0x01 R21 = 0x50 R22 = 0xC3 R23 = 0x00
R27 = 0x00 R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Memory: prog FLASH
data 0x0000 00 00 00 00 00 00 00 00 00 .....
data 0x0009 00 00 00 00 00 00 00 00 00 .....
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 13 Col 1 Ch 1 INS

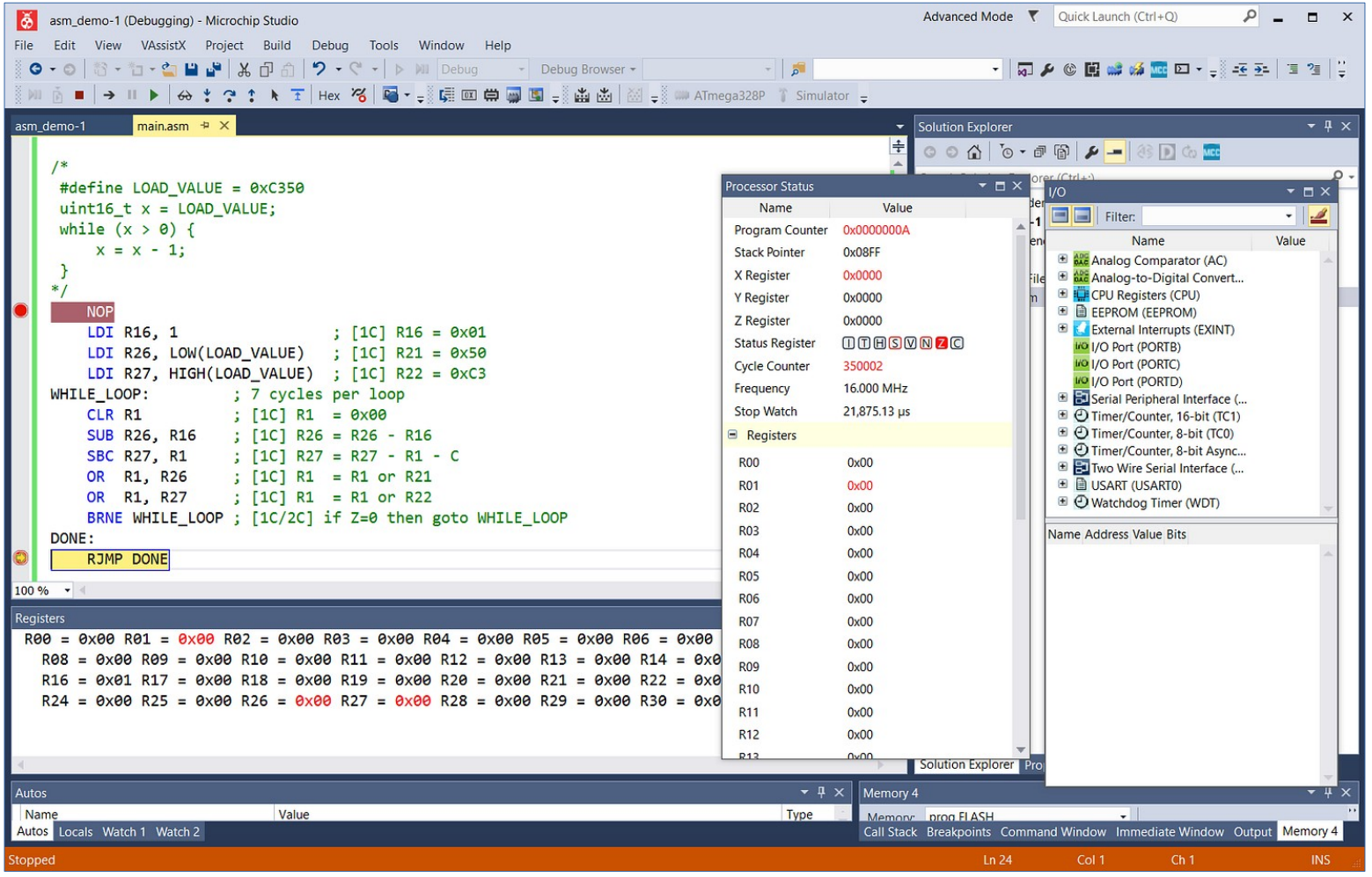
Example 4: AVR Assembly Code vs. Pseudo C Code

```
.CSEG
.ORG 0x0000
.EQU LOAD_VALUE = 50000        ; 0xC350 (hex)
/*
 #define LOAD_VALUE = 0xC350
 uint16_t x = LOAD_VALUE;
 while (x > 0) {
     x = x - 1;
 }
*/
    NOP                         ; [1C] no operation
    LDI R16, 1                  ; [1C] R16 = 0x01
    LDI R26, LOW(LOAD_VALUE)    ; [1C] R21 = 0x50 (low byte)
    LDI R27, HIGH(LOAD_VALUE)   ; [1C] R22 = 0xC3 (high byte)
WHILE_LOOP:                     ; 7 machine cycles per loop
    CLR R1                      ; [1C] R1  = 0x00
    SUB R26, R16                ; [1C] R26 = R26 - R16
    SBC R27, R1                 ; [1C] R27 = R27 - R1 - C
    OR  R1, R26                 ; [1C] R1  = R1 or R26
    OR  R1, R27                 ; [1C] R1  = R1 or R27
    BRNE WHILE_LOOP             ; [1C/2C] if Z=0 then goto WHILE_LOOP
    ; Total cycles: 3 + 50000 * 7 - 1 = 350002
DONE:
    RJMP DONE
```

Simulation


asm_demo-1 (Debugging) - Microchip Studio
Advanced Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
ATmega328P
Simulator
asm_demo-1
main.asm
/*
#define LOAD_VALUE = 0xC350
uint16_t x = LOAD_VALUE;
while (x > 0) {
x = x - 1;
}
*/
NOP
LDI R16, 1 ; [1C] R16 = 0x01
LDI R26, LOW(LOAD_VALUE) ; [1C] R21 = 0x50
LDI R27, HIGH(LOAD_VALUE) ; [1C] R22 = 0xC3
WHILE_LOOP: ; 7 cycles per loop
CLR R1 ; [1C] R1 = 0x00
SUB R26, R16 ; [1C] R26 = R26 - R16
SBC R27, R1 ; [1C] R27 = R27 - R1 - C
OR R1, R26 ; [1C] R1 = R1 or R21
OR R1, R27 ; [1C] R1 = R1 or R22
BRNE WHILE_LOOP ; [1C/2C] if Z=0 then goto WHILE_LOOP
DONE:
RJMP DONE
100 %
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x0
R16 = 0x01 R17 = 0x00 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R22 = 0x0
R24 = 0x00 R25 = 0x00 R26 = 0x50 R27 = 0xC3 R28 = 0x00 R29 = 0x00 R30 = 0x0
Processor Status
Name Value
Program Counter 0x00000001
Stack Pointer 0x08FF
X Register 0xC350
Y Register 0x0000
Z Register 0x0000
Status Register ITHSVNZC
Cycle Counter 0
Frequency 16.000 MHz
Stop Watch 0.00 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
Solution Explorer
I/O
Filter:
Name Value
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Watchdog Timer (WDT)
Name Address Value Bits
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Memory: prog FLASH
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 13 Col 1 Ch 1 INS

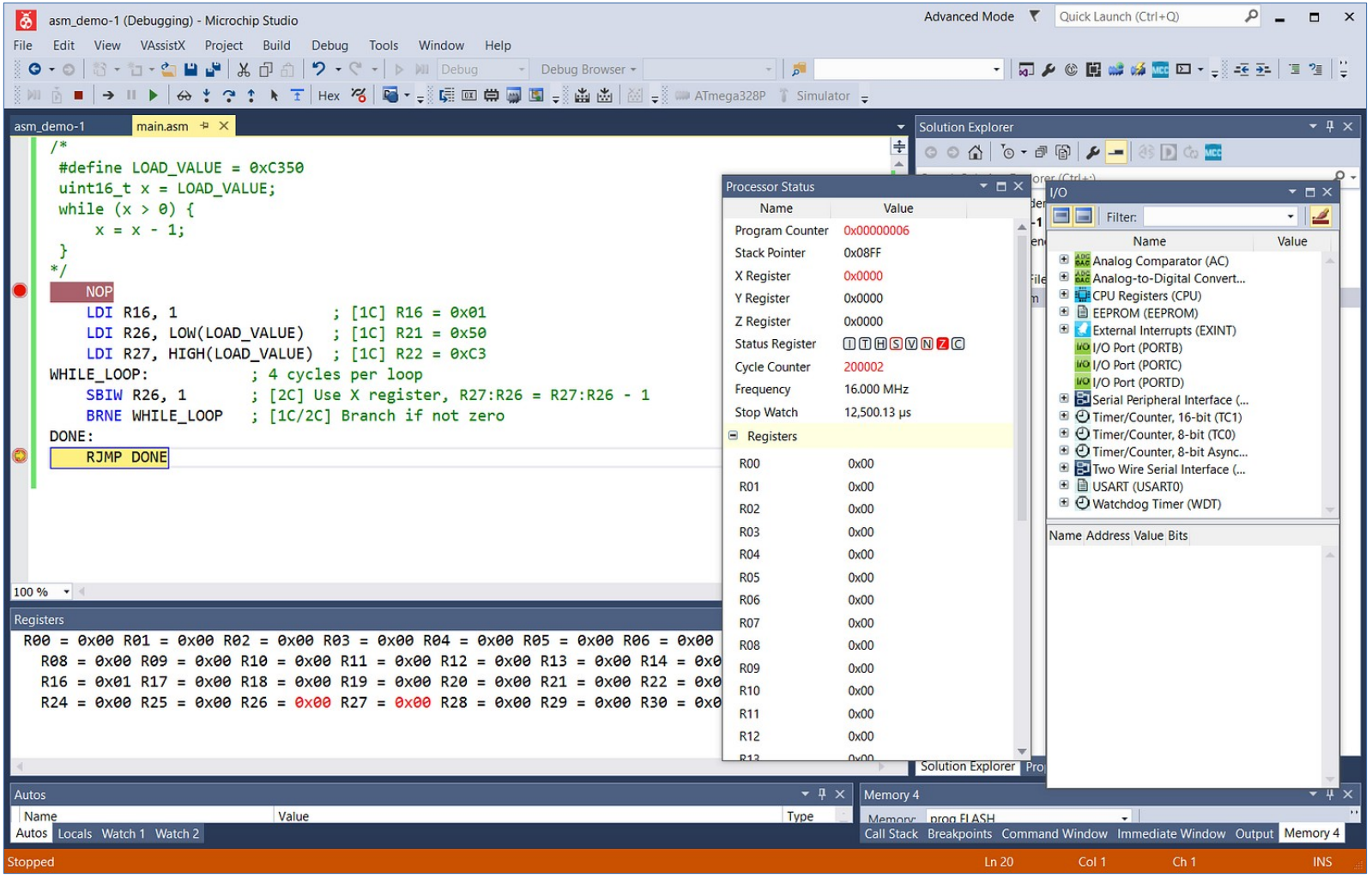
Simulation


asm_demo-1 (Debugging) - Microchip Studio
Advanced Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
ATmega328P
Simulator
asm_demo-1
main.asm
/*
#define LOAD_VALUE = 0xC350
uint16_t x = LOAD_VALUE;
while (x > 0) {
x = x - 1;
}
*/
NOP
LDI R16, 1 ; [1C] R16 = 0x01
LDI R26, LOW(LOAD_VALUE) ; [1C] R21 = 0x50
LDI R27, HIGH(LOAD_VALUE) ; [1C] R22 = 0xC3
WHILE_LOOP: ; 7 cycles per loop
CLR R1 ; [1C] R1 = 0x00
SUB R26, R16 ; [1C] R26 = R26 - R16
SBC R27, R1 ; [1C] R27 = R27 - R1 - C
OR R1, R26 ; [1C] R1 = R1 or R21
OR R1, R27 ; [1C] R1 = R1 or R22
BRNE WHILE_LOOP ; [1C/2C] if Z=0 then goto WHILE_LOOP
DONE:
RJMP DONE
100 %
Solution Explorer
Processor Status
Name Value
Program Counter 0x0000000A
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register I T H S V N Z C
Cycle Counter 350002
Frequency 16.000 MHz
Stop Watch 21,875.13 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
I/O
Filter:
Name Value
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Watchdog Timer (WDT)
Name Address Value Bits
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x0
R16 = 0x01 R17 = 0x00 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R22 = 0x0
R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00 R30 = 0x0
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Memory: prog FLASH
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 24 Col 1 Ch 1 INS

Example 5: AVR Assembly Code vs. Pseudo C Code

```
.CSEG
.ORG 0x0000
.EQU LOAD_VALUE = 50000          ; 0xC350 (hex)
/*
 #define LOAD_VALUE = 0xC350
 uint16_t x = LOAD_VALUE;
 while (x > 0) {
     x = x - 1;
 }
*/
    NOP
    LDI R16, 1                   ; [1C] R16 = 0x01
    LDI R26, LOW(LOAD_VALUE)     ; [1C] R21 = 0x50 (low byte)
    LDI R27, HIGH(LOAD_VALUE)    ; [1C] R22 = 0xC3 (high byte)
WHILE_LOOP:                      ; 4 machine cycles per loop
    SBIW R26, 1                  ; [2C] Use X register, R27:R26 = R27:R26-1
    BRNE WHILE_LOOP              ; [1C/2C] Branch if not zero (Z=0)
    ; Total cycles 3 + 50000 * 4 - 1 = 200002
DONE:
    RJMP DONE
```

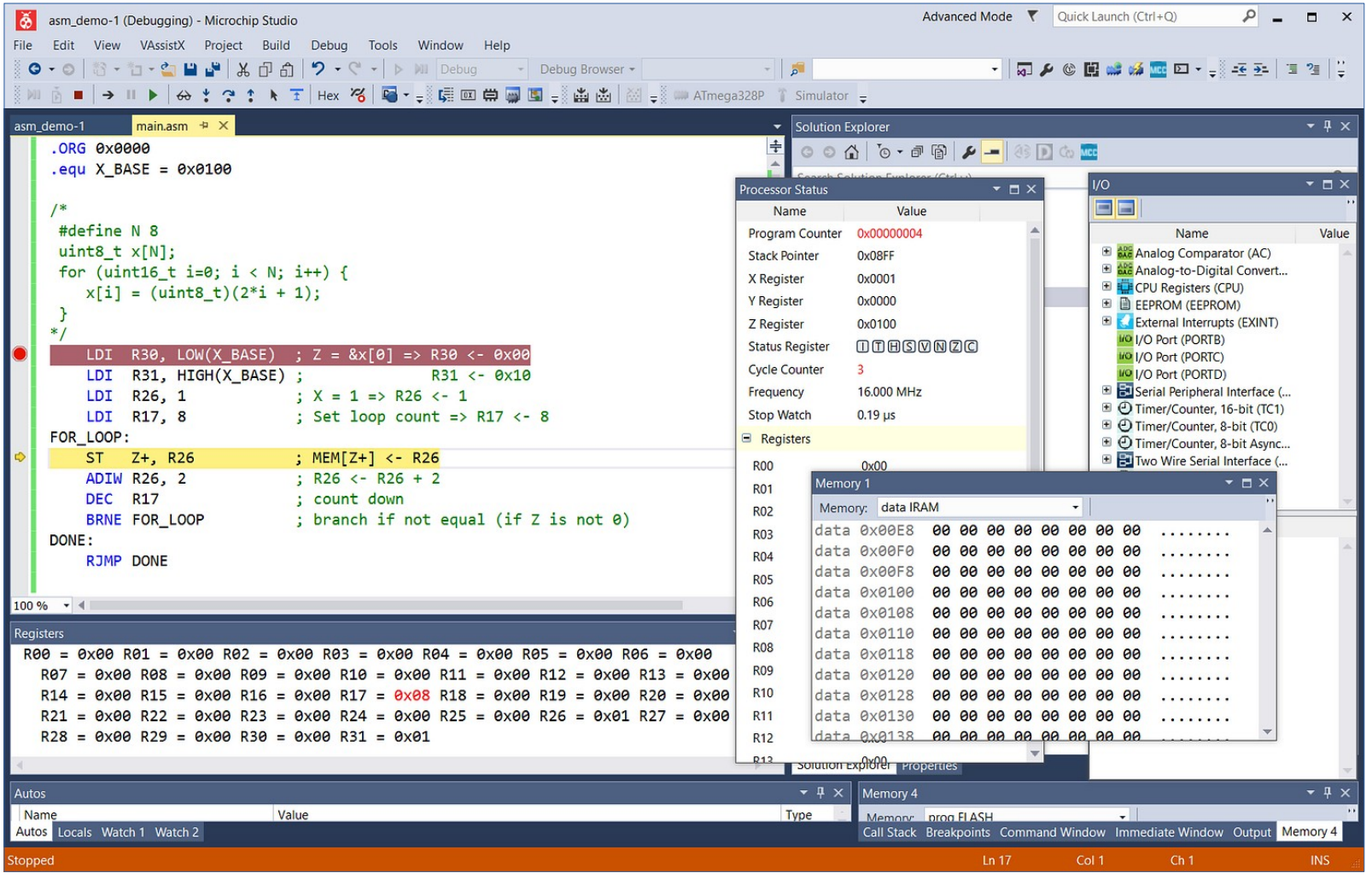
Simulation


asm_demo-1 (Debugging) - Microchip Studio
Advanced Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
ATmega328P
Simulator
asm_demo-1
main.asm
/*
#define LOAD_VALUE = 0xC350
uint16_t x = LOAD_VALUE;
while (x > 0) {
x = x - 1;
}
*/
NOP
LDI R16, 1 ; [1C] R16 = 0x01
LDI R26, LOW(LOAD_VALUE) ; [1C] R21 = 0x50
LDI R27, HIGH(LOAD_VALUE) ; [1C] R22 = 0xC3
WHILE_LOOP: ; 4 cycles per loop
SBIW R26, 1 ; [2C] Use X register, R27:R26 = R27:R26 - 1
BRNE WHILE_LOOP ; [1C/2C] Branch if not zero
DONE:
RJMP DONE
100 %
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00 R14 = 0x0
R16 = 0x01 R17 = 0x00 R18 = 0x00 R19 = 0x00 R20 = 0x00 R21 = 0x00 R22 = 0x0
R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00 R28 = 0x00 R29 = 0x00 R30 = 0x0
Processor Status
Name Value
Program Counter 0x00000006
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0000
Status Register ITHSVNZC
Cycle Counter 200002
Frequency 16.000 MHz
Stop Watch 12,500.13 µs
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
Solution Explorer
I/O
Filter:
Name Value
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
USART (USART0)
Watchdog Timer (WDT)
Name Address Value Bits
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Memory: prog FLASH
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 20 Col 1 Ch 1 INS

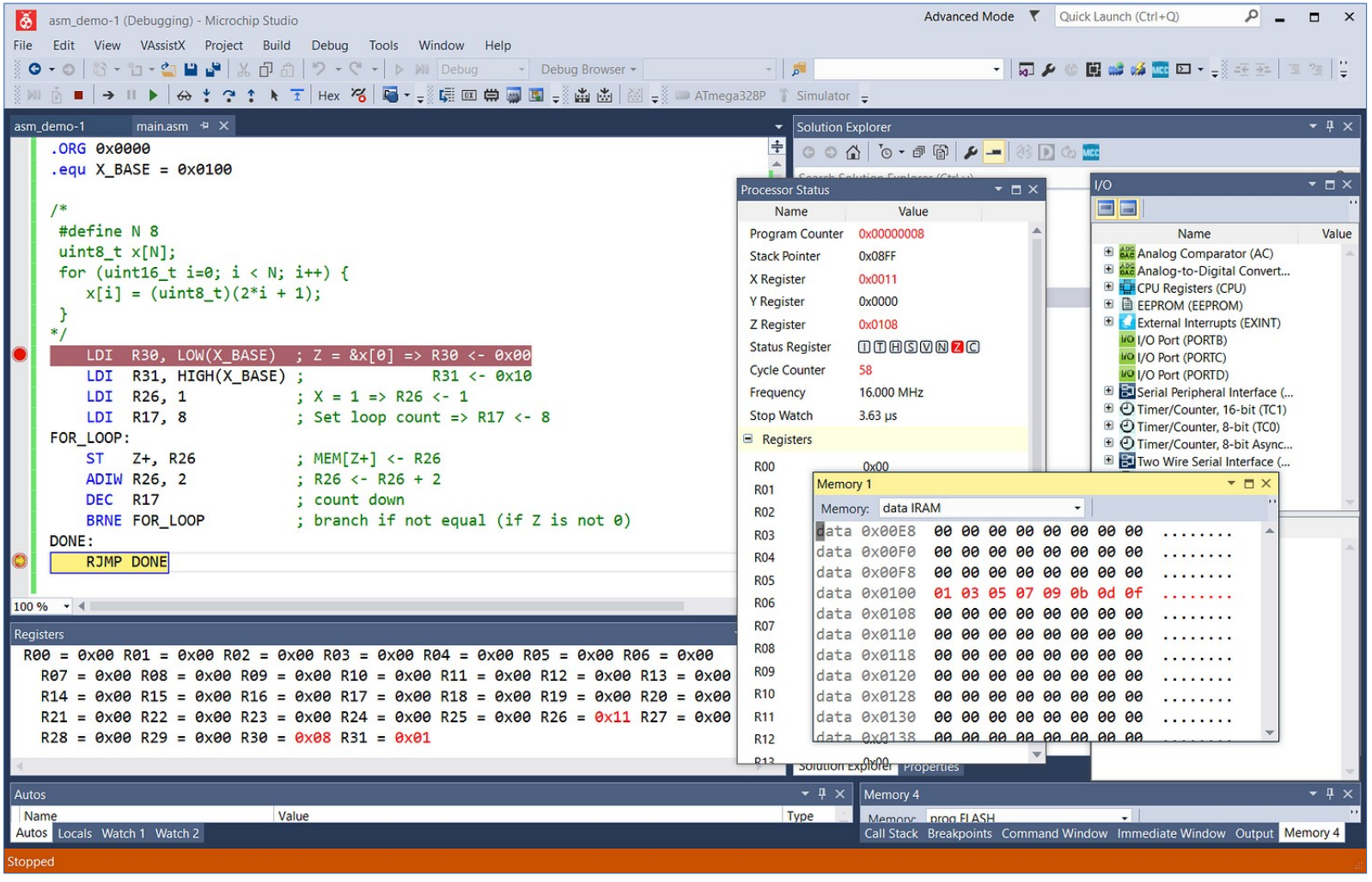
Example 6: AVR Assembly Code vs. Pseudo C Code

```
.CSEG
.ORG 0x0000
.EQU X_BASE = 0x0100
/*
 #define N 8
 uint8_t x[N];
 for (uint16_t i=0; i < N; i++) {
    x[i] = (uint8_t)(2*i + 1);
 }
*/
    LDI  R30, LOW(X_BASE)  ; Z = &x[0] => R30 = 0x00
    LDI  R31, HIGH(X_BASE) ;              R31 = 0x10
    LDI  R26, 1            ; X = 1 => R26 = 1
    LDI  R17, 8            ; Set loop count => R17 = 8
FOR_LOOP:
    ST   Z+, R26           ; MEM[Z+] = R26
    ADIW R26, 2            ; R26 = R26 + 2
    DEC  R17               ; count down
    BRNE FOR_LOOP          ; branch if not equal (if Z is not 0)
DONE:
    RJMP DONE
```

Simulation


asm_demo-1 (Debugging) - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
asm_demo-1
main.asm
.ORG 0x0000
.equ X_BASE = 0x0100
/*
#define N 8
uint8_t x[N];
for (uint16_t i=0; i < N; i++) {
x[i] = (uint8_t)(2*i + 1);
}
*/
LDI R30, LOW(X_BASE) ; Z = &x[0] => R30 <- 0x00
LDI R31, HIGH(X_BASE) ; R31 <- 0x10
LDI R26, 1 ; X = 1 => R26 <- 1
LDI R17, 8 ; Set loop count => R17 <- 8
FOR_LOOP:
ST Z+, R26 ; MEM[Z+] <- R26
ADIW R26, 2 ; R26 <- R26 + 2
DEC R17 ; count down
BRNE FOR_LOOP ; branch if not equal (if Z is not 0)
DONE:
RJMP DONE
Solution Explorer
Processor Status
Name Value
Program Counter 0x00000004
Stack Pointer 0x08FF
X Register 0x0001
Y Register 0x0000
Z Register 0x0100
Status Register ITHSVNZC
Cycle Counter 3
Frequency 16.000 MHz
Stop Watch 0.19 µs
Registers
I/O
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
Memory 1
Memory: data IRAM
data 0x00E8 00 00 00 00 00 00 00 00 ....
data 0x00F0 00 00 00 00 00 00 00 00 ....
data 0x00F8 00 00 00 00 00 00 00 00 ....
data 0x0100 00 00 00 00 00 00 00 00 ....
data 0x0108 00 00 00 00 00 00 00 00 ....
data 0x0110 00 00 00 00 00 00 00 00 ....
data 0x0118 00 00 00 00 00 00 00 00 ....
data 0x0120 00 00 00 00 00 00 00 00 ....
data 0x0128 00 00 00 00 00 00 00 00 ....
data 0x0130 00 00 00 00 00 00 00 00 ....
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00
R14 = 0x00 R15 = 0x00 R16 = 0x00 R17 = 0x08 R18 = 0x00 R19 = 0x00 R20 = 0x00
R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x01 R27 = 0x00
R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x01
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped
Ln 17 Col 1 Ch 1 INS

Simulation

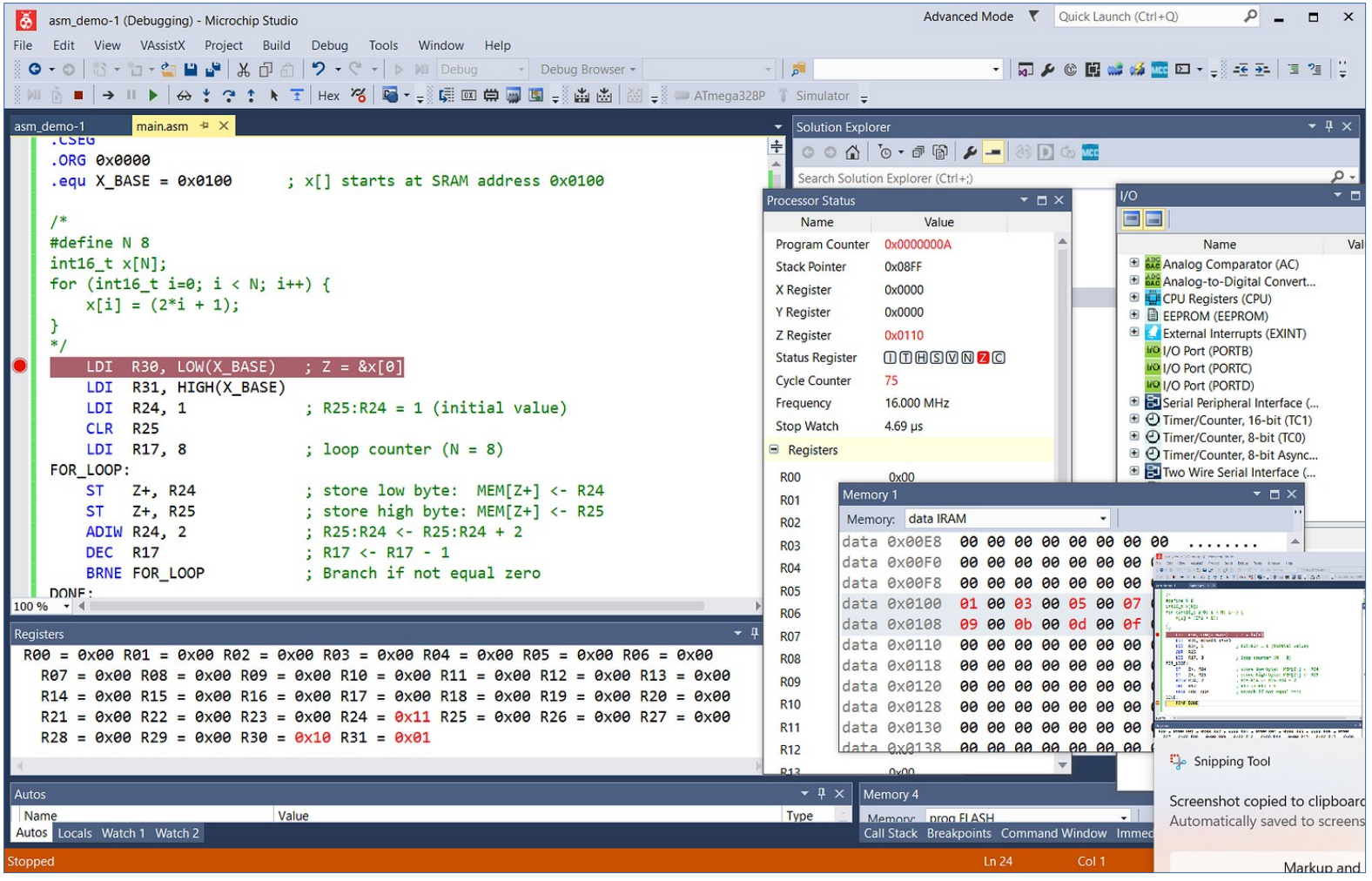

asm_demo-1 (Debugging) - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
asm_demo-1
main.asm
.ORG 0x0000
.equ X_BASE = 0x0100
/*
#define N 8
uint8_t x[N];
for (uint16_t i=0; i < N; i++) {
x[i] = (uint8_t)(2*i + 1);
}
*/
LDI R30, LOW(X_BASE) ; Z = &x[0] => R30 <- 0x00
LDI R31, HIGH(X_BASE) ; R31 <- 0x10
LDI R26, 1 ; X = 1 => R26 <- 1
LDI R17, 8 ; Set loop count => R17 <- 8
FOR_LOOP:
ST Z+, R26 ; MEM[Z+] <- R26
ADIW R26, 2 ; R26 <- R26 + 2
DEC R17 ; count down
BRNE FOR_LOOP ; branch if not equal (if Z is not 0)
DONE:
RJMP DONE
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00
R14 = 0x00 R15 = 0x00 R16 = 0x00 R17 = 0x00 R18 = 0x00 R19 = 0x00 R20 = 0x00
R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x00 R25 = 0x00 R26 = 0x11 R27 = 0x00
R28 = 0x00 R29 = 0x00 R30 = 0x08 R31 = 0x01
Solution Explorer
Processor Status
Program Counter 0x00000008
Stack Pointer 0x08FF
X Register 0x0011
Y Register 0x0000
Z Register 0x0108
Cycle Counter 58
Frequency 16.000 MHz
Stop Watch 3.63 µs
Registers
I/O
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
Memory 1
Memory: data IRAM
data 0x00E8 00 00 00 00 00 00 00 00 ....
data 0x00F0 00 00 00 00 00 00 00 00 ....
data 0x00F8 00 00 00 00 00 00 00 00 ....
data 0x0100 01 03 05 07 09 0b 0d 0f ....
data 0x0108 00 00 00 00 00 00 00 00 ....
data 0x0110 00 00 00 00 00 00 00 00 ....
data 0x0118 00 00 00 00 00 00 00 00 ....
data 0x0120 00 00 00 00 00 00 00 00 ....
data 0x0128 00 00 00 00 00 00 00 00 ....
data 0x0130 00 00 00 00 00 00 00 00 ....
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped

Example 7: AVR Assembly Code vs. Pseudo C Code

```
.CSEG
.ORG 0x0000
.EQU X_BASE = 0x0100          ; x[] starts at SRAM address 0x0100
/*
#define N 8
int16_t x[N];
for (int16_t i=0; i < N; i++) {
    x[i] = (2*i + 1);
}
*/
    LDI  R30, LOW(X_BASE)     ; Z = R31:R30 = &x[0]
    LDI  R31, HIGH(X_BASE)
    LDI  R24, 1               ; load initial value into R24
    CLR  R25                  ; R25 = 0
    LDI  R17, 8               ; loop counter (N = 8)
FOR_LOOP:
    ST   Z+, R24              ; store low byte:  MEM[Z+] = R24
    ST   Z+, R25              ; store high byte: MEM[Z+] = R25
    ADIW R24, 2               ; R25:R24 = R25:R24 + 2
    DEC  R17                  ; R17 = R17 - 1
    BRNE FOR_LOOP             ; Branch if not equal zero
DONE:
    RJMP DONE
```

AT mega328P SRAM Range : Start: 0x0100, End: 0x08FF (Size: 2 KB or 2048 bytes)

Simulation


asm_demo-1 (Debugging) - Microchip Studio
Advanced Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
ATmega328P
Simulator
asm_demo-1
main.asm
.CSEG
.ORG 0x0000
.equ X_BASE = 0x0100 ; x[] starts at SRAM address 0x0100
/*
#define N 8
int16_t x[N];
for (int16_t i=0; i < N; i++) {
x[i] = (2*i + 1);
}
*/
LDI R30, LOW(X_BASE) ; Z = &x[0]
LDI R31, HIGH(X_BASE)
LDI R24, 1 ; R25:R24 = 1 (initial value)
CLR R25
LDI R17, 8 ; loop counter (N = 8)
FOR_LOOP:
ST Z+, R24 ; store low byte: MEM[Z+] <- R24
ST Z+, R25 ; store high byte: MEM[Z+] <- R25
ADIW R24, 2 ; R25:R24 <- R25:R24 + 2
DEC R17 ; R17 <- R17 - 1
BRNE FOR_LOOP ; Branch if not equal zero
DONE:
100 %
Registers
R00 = 0x00 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00
R14 = 0x00 R15 = 0x00 R16 = 0x00 R17 = 0x00 R18 = 0x00 R19 = 0x00 R20 = 0x00
R21 = 0x00 R22 = 0x00 R23 = 0x00 R24 = 0x11 R25 = 0x00 R26 = 0x00 R27 = 0x00
R28 = 0x00 R29 = 0x00 R30 = 0x10 R31 = 0x01
Solution Explorer
Search Solution Explorer (Ctrl+;)
Processor Status
Name Value
Program Counter 0x0000000A
Stack Pointer 0x08FF
X Register 0x0000
Y Register 0x0000
Z Register 0x0110
Status Register I T H S V N Z C
Cycle Counter 75
Frequency 16.000 MHz
Stop Watch 4.69 µs
Registers
R00 0x00
R01
R02
R03
R04
R05
R06
R07
R08
R09
R10
R11
R12
R13 0x00
I/O
Name
Analog Comparator (AC)
Analog-to-Digital Convert...
CPU Registers (CPU)
EEPROM (EEPROM)
External Interrupts (EXINT)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)
Timer/Counter, 8-bit (TC0)
Timer/Counter, 8-bit Async...
Two Wire Serial Interface (...
Memory 1
Memory: data IRAM
data 0x00E8 00 00 00 00 00 00 00 00 ....
data 0x00F0 00 00 00 00 00 00 00 00
data 0x00F8 00 00 00 00 00 00 00 00
data 0x0100 01 00 03 00 05 00 07
data 0x0108 09 00 0b 00 0d 00 0f
data 0x0110 00 00 00 00 00 00 00 00
data 0x0118 00 00 00 00 00 00 00 00
data 0x0120 00 00 00 00 00 00 00 00
data 0x0128 00 00 00 00 00 00 00 00
data 0x0130 00 00 00 00 00 00 00 00
data 0x0138 00 00 00 00 00 00 00 00
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screens
Markup and
Autos
Name Value Type
Autos Locals Watch 1 Watch 2
Memory 4
Memory: prog FLASH
Call Stack Breakpoints Command Window Immed
Stopped
Ln 24 Col 1

Example 8: AVR Assembly Code vs. Pseudo C Code (1)

```
.CSEG
.ORG 0x0000
.EQU X_BASE = 0x0100        ; 16-bit x at 0x0100 and 0x0101
.EQU Y_BASE = 0x0102        ; 8-bit  y at 0x0102
.EQU VALUE  = 0xF355
/*
 uint16_t x = 0x3055;  // at 0x0100
 uint8_t  y;           // at 0x0102
 uint8_t cnt = 0;
 for (uint8_t i=0,; i < 16; i++) {
   cnt += (x & 1); x >>= 1;
 }
 y = cnt;
*/
DATA_INIT:
      LDI  R16, LOW(VALUE)
      STS  X_BASE, R16          ; low byte of x
      LDI  R16, HIGH(VALUE)
      STS  X_BASE+1, R16        ; high byte of x
      CLR  R18                  ; R18 (cnt) = 0

      LDS  R16, X_BASE          ; R16 = low byte of x
      LDS  R17, X_BASE+1        ; R17 = high byte of x
      LDI  R19, 16              ; loop counter i = 16
; Code continues on the next page.
```

Example 8: AVR Assembly Code vs. Pseudo C Code (2)

```
COUNT_LOOP:
      ; cnt += (x & 1)
      MOV  R20, R16              ; copy low byte
      ANDI R20, 0x01             ; get the LSB
      ADD  R18, R20              ; accumulate

      ; x >>= 1 (shift 16-bit)
      LSR  R17                   ; high byte
      ROR  R16                   ; low byte

      DEC  R19
      BRNE COUNT_LOOP

      ; y = cnt (only now write result)
      STS  Y_ADDR, R18

DONE:
      RJMP DONE
```

LSR – Logical Shift Right

Description

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C flag of the SREG. This operation effectively divides an unsigned value by two. The C flag can be used to round the result.

Operation:

(i)

	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

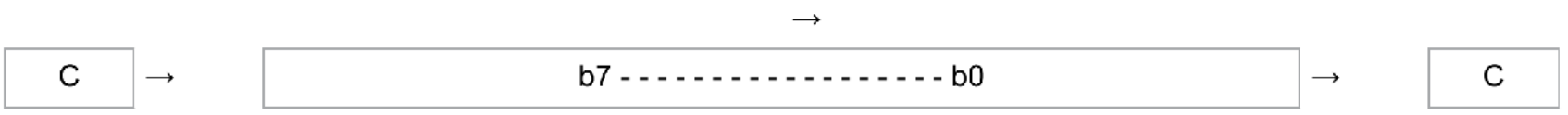
I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

ROR – Rotate Right through Carry

Description

Shifts all bits in Rd one place to the right. The C flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR, it effectively divides multi-byte unsigned values by two. The Carry flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	ROR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0111

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

Example 9: Using Data in SRAM

- **How to reserve the memory in SRAM for a variable of one byte?**
- **How to reserve the memory in SRAM for a variable of one word (16 bits)?**

```
.DSEG
A: .BYTE 1  ; reserve one byte in SRAM (data segment) for A.
B: .BYTE 2  ; reserve two bytes (a word) in SRAM for B (little endian).

.CSEG
.ORG 0x0000
    LDI  R16, 0x55   ; load a constant (0x55) into r16
    STS  A, R16      ; write r16 to A

    LDS  R3, A       ; load A into r3
    MOV  R4, R16     ; r4 = r16
    STS  B, R3       ; write r3 as the low byte of B
    STS  B+1, R4     ; write r4 as the high byte of B
```

Example10: Checking Zero Flag

How to use conditional branch to check whether the Z (Zero) flag is 0 or 1 ?

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
    LDI R16, 0      ; r16 = 0 (no impact on SREG)
    BRNE LABEL_1    ; Branch if not equal (if Z = 0)
    RJMP DONE
LABEL_1:
    INC R16
    BREQ LABEL_2    ; Branch if equal (if Z = 1)
    CPI R16, 1      ; Compare r16 with 1 (r16-1)
    BREQ LABEL_2    ; Branch if equal (if Z = 0)
    RJMP DONE
LABEL_2:
    DEC R16         ; R16 = R16 - 1
    TST R16         ; Test if r16 is zero
    BRNE LABEL_2
DONE: RJMP DONE
```

Example 11: 8-bit Unsigned Comparison

How to compare A and B (both unsigned 8-bit) in AVR assembly?

```
.DSEG
A: .BYTE 1
B: .BYTE 1

.CSEG
.ORG 0x0000
    LDI  R16, 100  ; r16 = 100
    STS  A, R16    ; write r16 to A in SRAM
    DEC  R16       ; r16 = r16-1
    STS  B, R16    ; write r16 to B in SRAM

    LDS  R4, A     ; load A from SRAM into r4
    LDS  R5, B     ; load B from SRAM into r5
    CP   R5, R4    ; compare A with B (both unsigned): compute (B - A)
    BRLO LABEL_1   ; branch if lower (unsigned): C (borrow) is 1 (A > B)
    RJMP DONE
LABEL_1:
    CP R4, R5      ; compare A with B (both unsigned): compute (A - B)
    BRSH LABEL_2   ; branch if same or higher: C is 0 (A >= B)
    RJMP DONE
LABEL_2:
    NOP
DONE: RJMP DONE
```

Example 12: 8-bit Signed Comparison

How to compare A and B (both signed 8-bit) in AVR assembly?

```
.DSEG
A: .BYTE 1
B: .BYTE 1

.CSEG
.ORG 0x0000
    LDI  R16, -1    ; r16 = -1
    STS  A, R16     ; write r16 to A in SRAM
    DEC  R16        ; r16 = r16-1
    STS  B, R16     ; write r16 to B in SRAM

    LDS  R4, A      ; load A into r4
    LDS  R5, B      ; load B into r5
    CP   R5, R4     ; compare A with B (both signed): compute (B - A)
    BRLT LABEL_1    ; branch if less than (signed): S = 1, B-A < 0
    RJMP DONE
LABEL_1:
    CP R4, R5       ; compare A with B (both signed): compute (A - B)
    BRGE LABEL_2    ; branch if greater or equal: S is 0 (A >= B)
    RJMP DONE
LABEL_2:
    NOP
DONE: RJMP DONE
```

AVR Instructions for 8-bit Comparisons

Unsigned

Mnemonic	Meaning
`BRLO`	lower (`C = 1`)
`BRSH`	same or higher (`C = 0`)
`BRHI`	higher
`BRLS`	lower or same

Signed

Mnemonic	Meaning
`BRLT`	less than (signed)
`BRGE`	greater or equal (signed)
`BRGT`	greater than (signed)
`BRLE`	less or equal (signed)

8-bit UNSIGNED comparisons (uint8_t)

CP Rd, Rr computes Rd − Rr

CPI Rd, K computes Rd − K, K is 8-bit constant.

Condition	Instruction	Branch
`x > y`	`CP y, x`	`BRLO`
`x ≥ y`	`CP y, x`	`BRSH`
`x < y`	`CP x, y`	`BRLO`
`x ≤ y`	`CP x, y`	`BRSH`
`x == y`	`CP x, y`	`BREQ`
`x ≠ y`	`CP x, y`	`BRNE`

8-bit SIGNED comparisons (int8_t)

CP Rd, Rr computes Rd − Rr

CPI Rd, K computes Rd − K, K is 8-bit constant.

Condition	Instruction	Branch
`x > y`	`CP y, x`	`BRLT`
`x ≥ y`	`CP y, x`	`BRGE`
`x < y`	`CP x, y`	`BRLT`
`x ≤ y`	`CP x, y`	`BRGE`
`x == y`	`CP x, y`	`BREQ`
`x ≠ y`	`CP x, y`	`BRNE`

Example 13: 16-bit Unsigned Comparison

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
    ; Use the Z to point to A and B in Flash
    LDI  ZL, LOW(A*2)
    LDI  ZH, HIGH(A*2)
    LPM  R24, Z+        ; A low byte
    LPM  R25, Z         ; A high byte

    LDI  ZL, LOW(B*2)
    LDI  ZH, HIGH(B*2)
    LPM  R26, Z+        ; B low byte
    LPM  R27, Z         ; B high byte

    ; Compare B with A (B - A)
    CP   R26, R24       ; low byte
    CPC  R27, R25       ; high byte with carry
    BRLO LABEL_1        ; branch is if lower: B < A (unsigned)
    RJMP DONE
LABEL_1:
    NOP                 ; This code executes if B < A is true
DONE:
    RJMP DONE
; Constant data in Flash
A: .DW 1000             ; 0x03E8 (low=E8, high=03)
B: .DW 500              ; 0x01F4 (low=F4, high=01)
```

Example 14: 16-bit Signed Comparison

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
    ; Load A (16-bit) from Flash
    LDI  ZL, LOW(A*2)
    LDI  ZH, HIGH(A*2)
    LPM  R24, Z+        ; A low byte
    LPM  R25, Z         ; A high byte
    ; Load B (16-bit) from Flash
    LDI  ZL, LOW(B*2)
    LDI  ZH, HIGH(B*2)
    LPM  R26, Z+        ; B low byte
    LPM  R27, Z         ; B high byte
    ; Compare B with A (Calculates B - A)
    CP   R26, R24       ; Compare low bytes
    CPC  R27, R25       ; Compare high bytes with Carry from low byte
    ; BRLT checks the S flag (S = N ⊕ V)
    BRLT LABEL_1        ; Branch if less than: B < A (Signed)
    RJMP DONE
LABEL_1:
    NOP                 ; This code executes if B < A
DONE:
    RJMP DONE
; Constant data in Flash
A: .DW 1000             ; +1000 (16-bit, stored as 0x03E8)
B: .DW -500             ;  -500 (16-bit, stored as 0xFE0C)
```

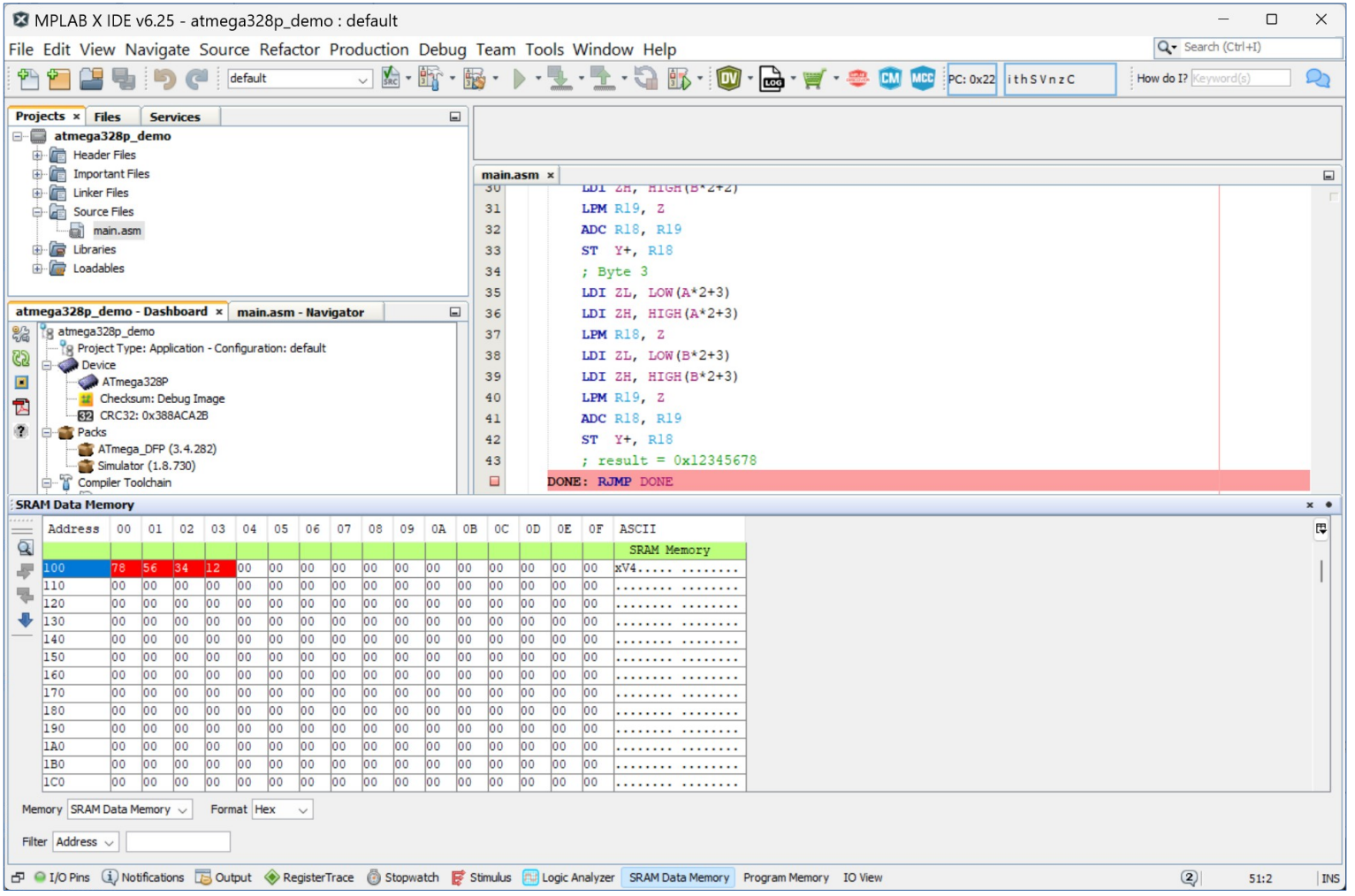
Example 15: 32-bit Signed Addition (1)

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
    ; use Y pointer to store RESULT in SRAM
    LDI YL, LOW(RESULT)
    LDI YH, HIGH(RESULT)
    ; use Z pointer to load A and B from Flash; Byte 0
    LDI ZL, LOW(A*2)
    LDI ZH, HIGH(A*2)
    LPM R18, Z
    LDI ZL, LOW(B*2)
    LDI ZH, HIGH(B*2)
    LPM R19, Z
    ADD R18, R19
    ST  Y+, R18
    ; use Z pointer to load A and B from Flash; Byte 1
    LDI ZL, LOW(A*2+1)
    LDI ZH, HIGH(A*2+1)
    LPM R18, Z
    LDI ZL, LOW(B*2+1)
    LDI ZH, HIGH(B*2+1)
    LPM R19, Z
    ADC R18, R19
    ST  Y+, R18
; Code continues on the next page.
```

Example 15: 32-bit Signed Addition (2)

```
    ; use Z pointer to load A and B from Flash; Byte 2
    LDI ZL, LOW(A*2+2)
    LDI ZH, HIGH(A*2+2)
    LPM R18, Z
    LDI ZL, LOW(B*2+2)
    LDI ZH, HIGH(B*2+2)
    LPM R19, Z
    ADC R18, R19
    ST  Y+, R18
    ; use Z pointer to load A and B from Flash; Byte 3
    LDI ZL, LOW(A*2+3)
    LDI ZH, HIGH(A*2+3)
    LPM R18, Z
    LDI ZL, LOW(B*2+3)
    LDI ZH, HIGH(B*2+3)
    LPM R19, Z
    ADC R18, R19
    ST  Y+, R18
    ; result = 0x12345678
DONE:
    RJMP DONE
; Constants in Flash
A: .DW 0x0000, 0x8000    ; 0x80000000
B: .DW 0x5678, 0x9234    ; 0x92345678
.DSEG
.ORG 0x0100
RESULT: .BYTE 4 ; Reserve 4 bytes in SRAM
```

Simulation


MPLAB X IDE v6.25 - atmega328p_demo : default
File Edit View Navigate Source Refactor Production Debug Team Tools Window Help
Projects
Files
Services
atmega328p_demo
Header Files
Important Files
Linker Files
Source Files
main.asm
Libraries
Loadables
atmega328p_demo - Dashboard
main.asm - Navigator
Project Type: Application - Configuration: default
Device
ATmega328P
Checksum: Debug Image
CRC32: 0x388ACA2B
Packs
ATmega_DFP (3.4.282)
Simulator (1.8.730)
Compiler Toolchain
main.asm
LPM R19, Z
ADC R18, R19
ST Y+, R18
; Byte 3
LDI ZL, LOW(A*2+3)
LDI ZH, HIGH(A*2+3)
LPM R18, Z
LDI ZL, LOW(B*2+3)
LDI ZH, HIGH(B*2+3)
LPM R19, Z
ADC R18, R19
ST Y+, R18
; result = 0x12345678
DONE: RJMP DONE
SRAM Data Memory
Address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ASCII
SRAM Memory
100 78 56 34 12 00 00 00 00 00 00 00 00 00 00 00 00 xV4..... ....
110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
1A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
1B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
Memory SRAM Data Memory
Format Hex
Filter Address
I/O Pins
Notifications
Output
RegisterTrace
Stopwatch
Stimulus
Logic Analyzer
SRAM Data Memory
Program Memory
IO View
51:2
INS

Example 16: Subroutine – 8x8 Unsigned Multiply (1)

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
    ; Stack Initialization
    LDI R16, HIGH(RAMEND)
    OUT SPH, R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16

    ; Load A and B and compute 8x8 multiply ((unsigned)
    LDI R22, 0x64     ; Argument A
    LDI R23, 0xC8     ; Argument B
    CALL MULT         ; Call the subroutine (Result in R25:R24)
DONE:
    RJMP DONE

; Code continues on the next page.
```

Example 16: Subroutine – 8x8 Unsigned Multiply (2)

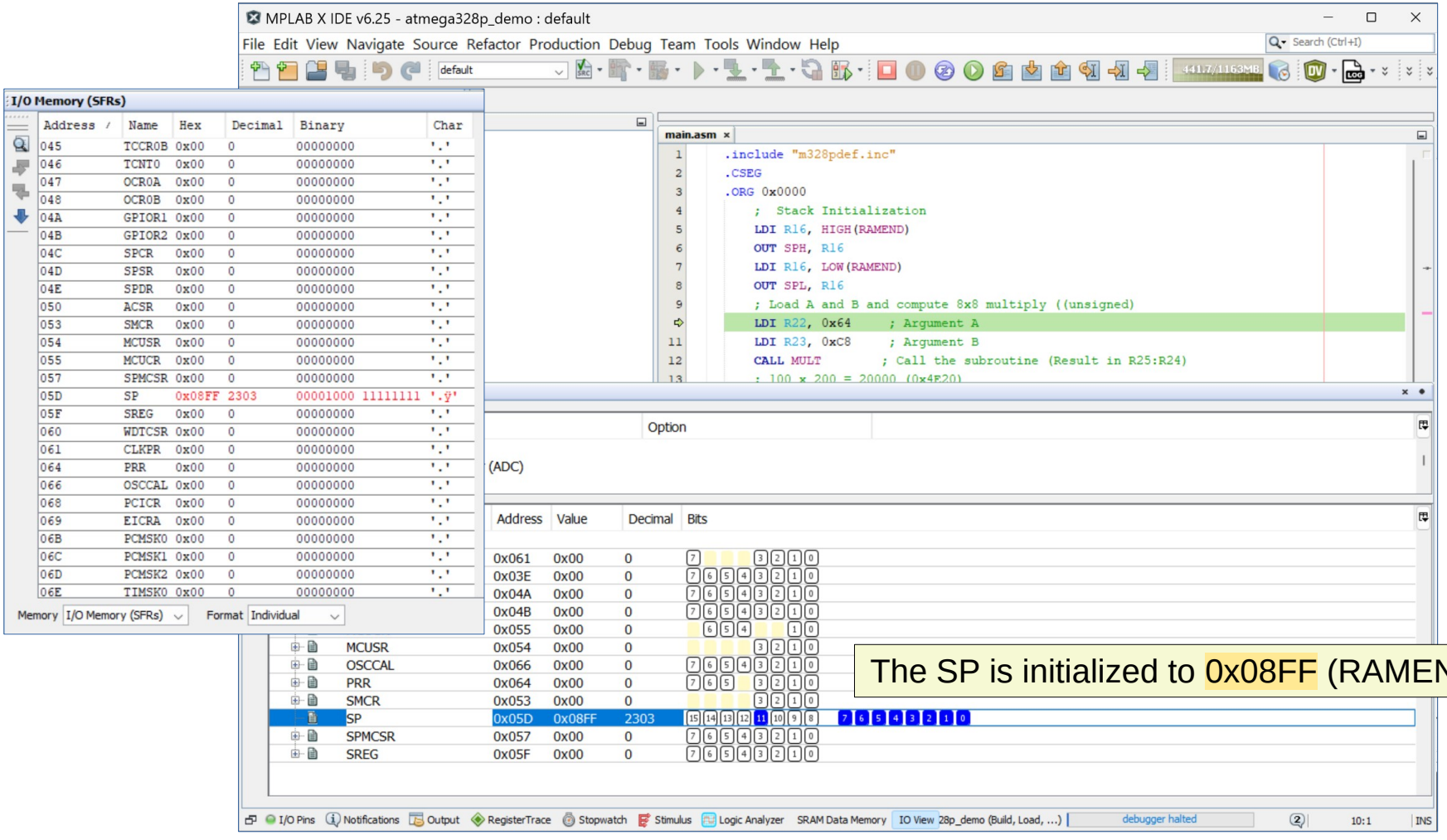
```
; Subroutine
; Inputs: Arguments are passed and stored in R23 and R23.
; Outputs: Multiplication is stored in R25:R24 (16-bit).
MULT:
    ; Save registers R0 and R1 to stack
    PUSH R0          ; [2C]
    PUSH R1          ; [2C]
    ; perform the operation
    MUL  R22, R23    ; [2C] Hardware Multiply (Unsigned)
    ; MULS R22, R23  ; [2C] Hardware Multiply (Signed)
    ; Result R1:R0 = R24 * R22
    MOV  R24, R0     ; [1C] Copy result low byte to R24
    MOV  R25, R1     ; [1C] Copy result high byte to R25
    ; Store registers from stack
    POP  R1          ; [2C]
    POP  R0          ; [2C]
    ; Return from subroutine call
    RET
```

AVR Multiply Instructions

- The MUL instruction has a fixed destination for its output.
  - `MUL Rd, Rr` (Unsigned x Unsigned)
  - `MULS Rd,Rr` (Signed x Signed)
  - `MULSU Rd, Rr` (Signed x Unsigned)
- Source Registers: Any two registers from the general-purpose register file (R0..R31).
- Destination Registers: The result is always a 16-bit value placed in the R1:R0 register pair:
  - R1 (High Byte):R0 (Low Byte)

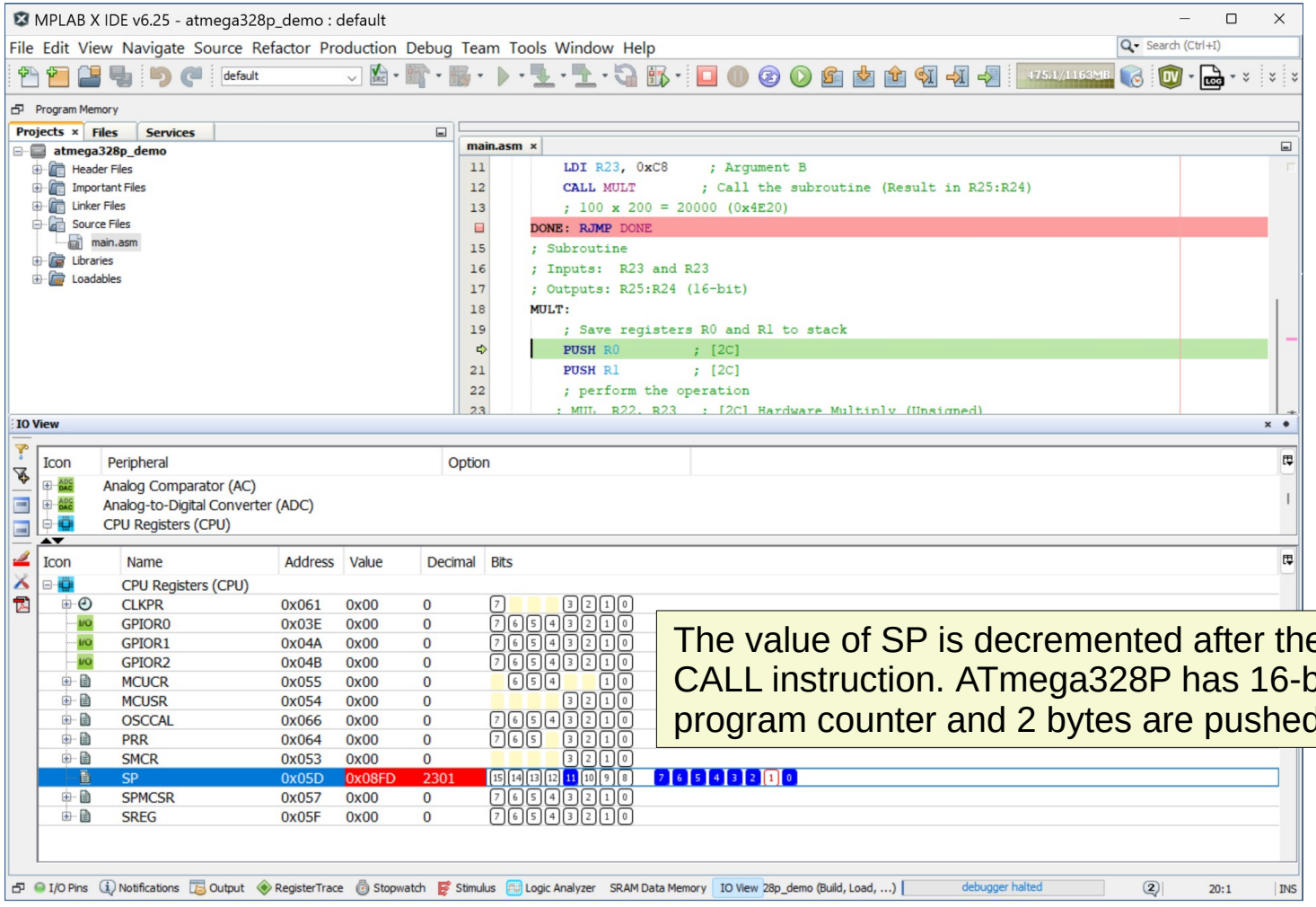
```
  0x64   (A: unsigned=100, signed=100)
x 0xC8   (B: unsigned=200, signed=-56)
--------------
0x4e20   (MUL  Result: signed=20,000)
0xea20   (MULS Result: unsigned=-5,600)
```

Simulation



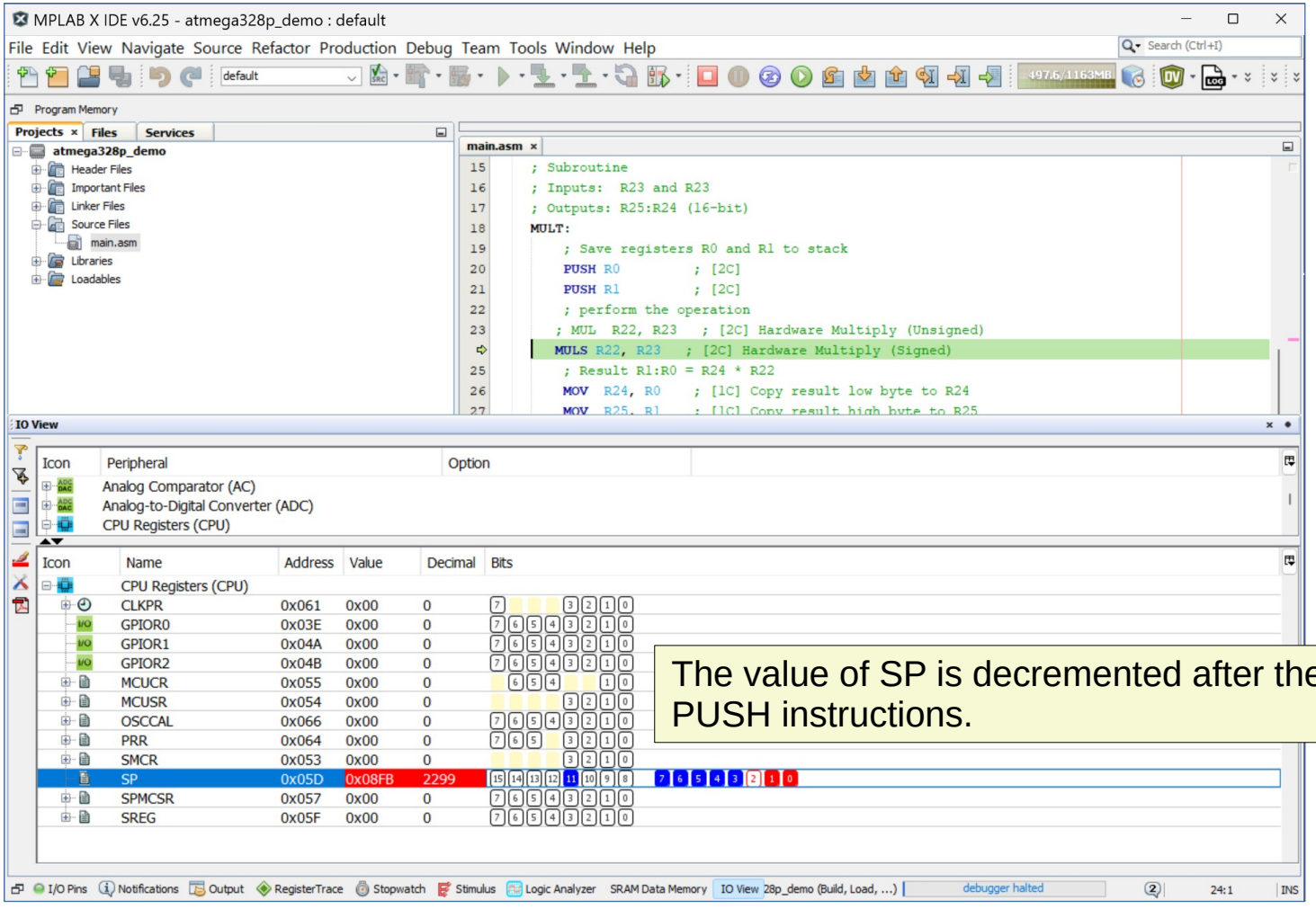
The SP is initialized to 0x08FF (RAMEND).

Simulation



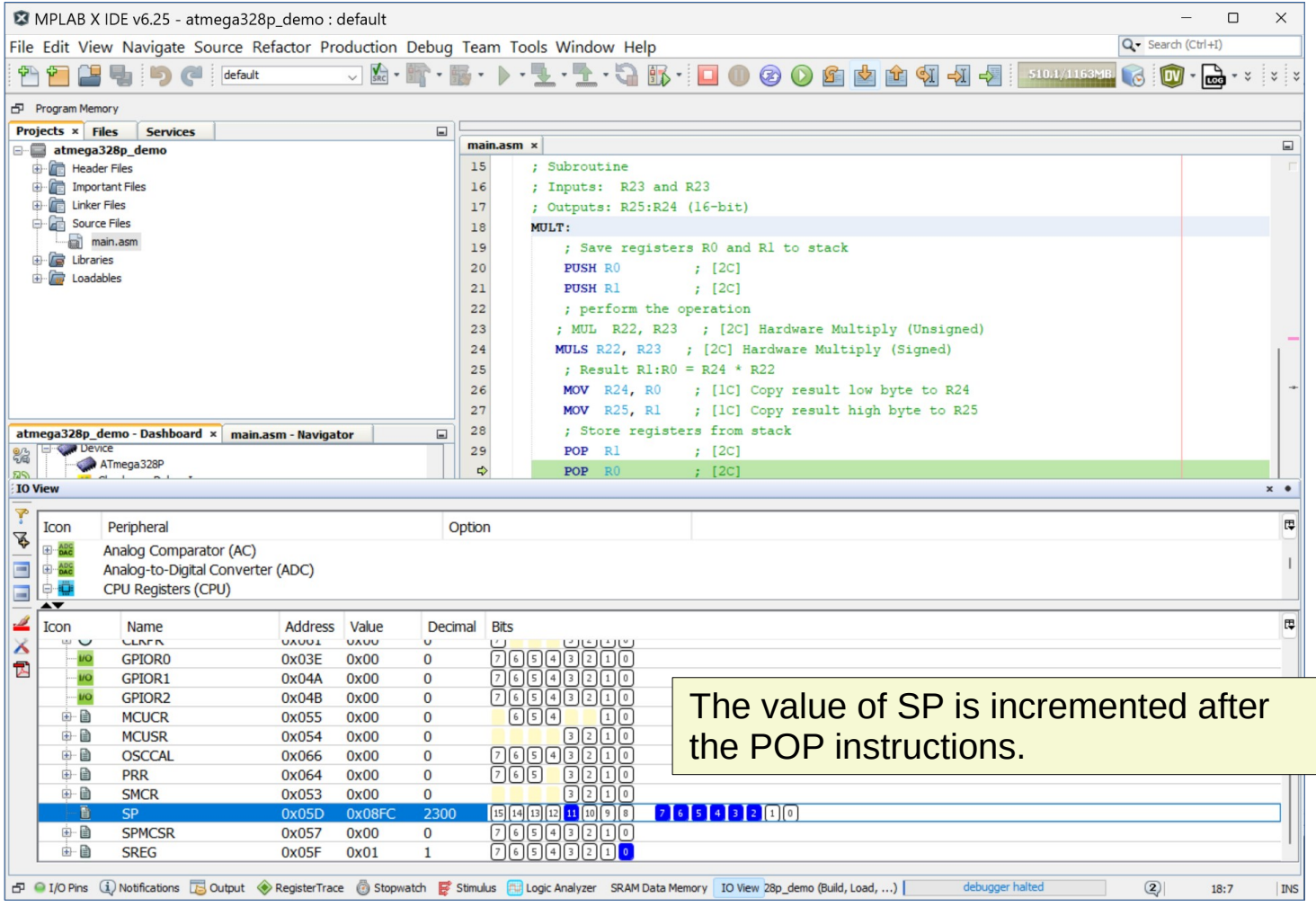
The value of SP is decremented after the CALL instruction. ATmega328P has 16-bit program counter and 2 bytes are pushed.

Simulation



The value of SP is decremented after the PUSH instructions.

Simulation



The value of SP is incremented after the POP instructions.

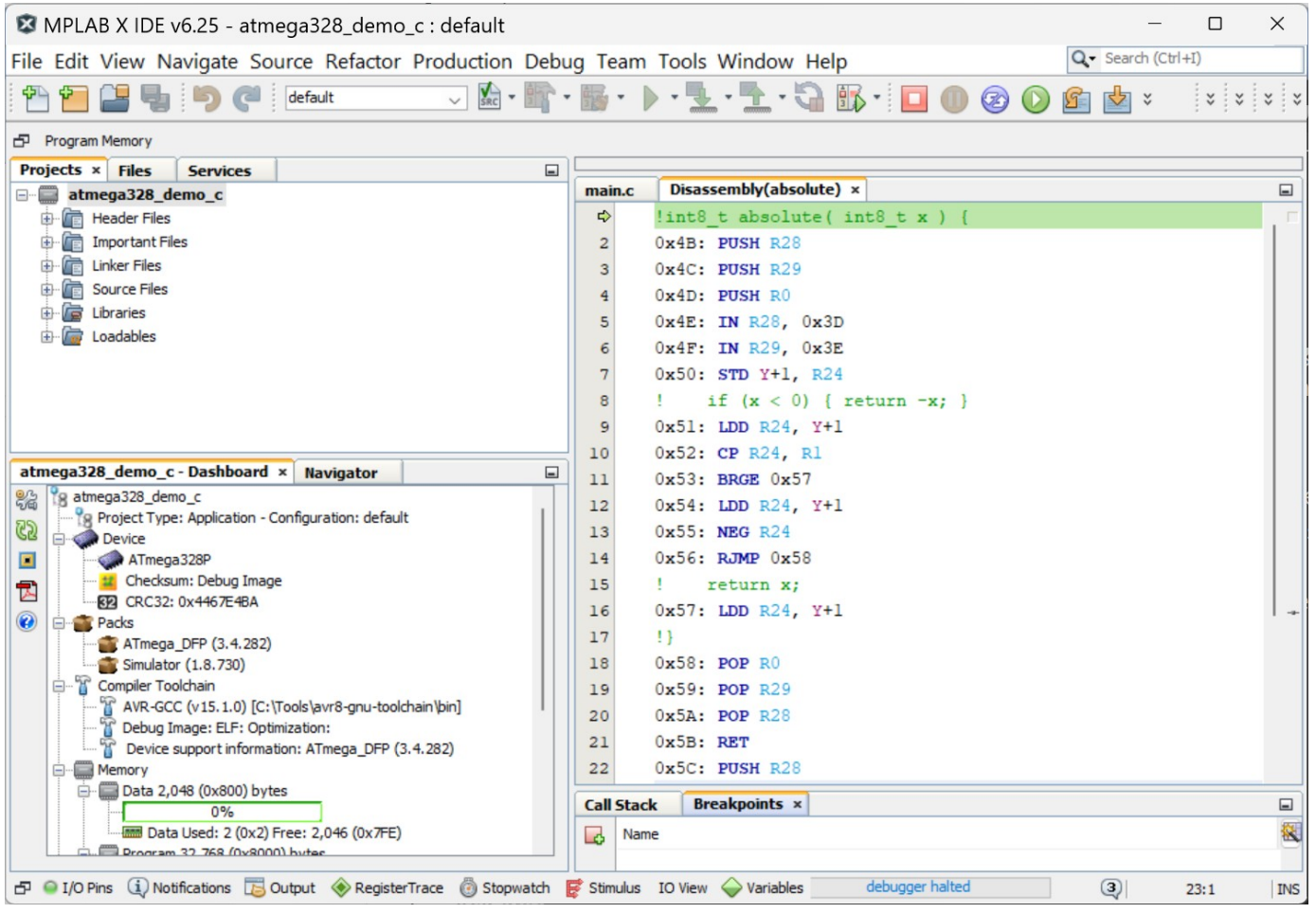
Subroutine Calls: Arguments and Return Values

- For AVR assembly programming, a subroutine call can return one or more values.
  - They can be returned in registers or via memory (including the stack).
  - Examples:
    - 8-bit: use R24
    - 16-bit: use R25:R24
- Other approaches to return values from a function call.
  - A pointer (call by reference in C) can be used to store the result in SRAM, with the pointer passed as an argument to the function.

C Function Calls and Stack Frames

- C compilers, such as AVR-GCC, define a **calling convention** that specifies how the stack is used for **C function calls**, including argument passing, return values, and register preservation.
- A **stack frame** defines the layout and usage of the stack memory for a single function call. It defines:
  - Where the return address is stored
  - Where function arguments are located (if passed on the stack)
  - Which registers are saved and where
  - Where local variables are allocated
  - How the stack pointer and frame pointer are used during the function execution
- The **stack frame pointer** remains constant, while the stack pointer changes as data is pushed and popped.
  - For AVR-GCC, the **Y register pair (R29:R28)** is commonly used as the stack frame pointer.

Example 17: AVR C Code and Disassembly




```c
#include <avr/io.h>

// global variable
int8_t number = -10;

// User-defined function
int8_t absolute( int8_t x ) {
    if (x < 0) { return -x; }
    return x;
}

// The main function
void main(void) {
    number = absolute(number);
    while (1);
    return;
}
```

Example 17: AVR C Code and Disassembly

- In this example, the C code is compiled using the avr-gcc compiler with no optimization (-O0) enabled.
- The generated .hex file is then analyzed and disassembled.
- An excerpt of the resulting AVR assembly code is shown below.
- Frame pointer: use the Y register pair (R29:R28)
- ATmega328P Stack Pointer registers.
  - SPL: I/O address 0x3D
  - SPH: I/O address 0x3E
- 8-bit argument: use R24 and 8-bit return value: use R24

```
; main() { ... }
0x5C:    PUSH R28         ; Save YL (R28) and YH (R29) on stack
0x5D:    PUSH R29
0x5E:    IN R28, 0x3D     ; Set the stack frame pointer: YH:YL (R29:R28) = SPH:SPL
0x5F:    IN R29, 0x3E
;-----------------------------------------------------------------
0x60:    LDS R24, 0x100   ; Load the argument: R24 = MEM[0x100] = -10
0x62:    CALL absolute    ; Call absolute
0x64:    STS 0x100, R24   ; Save the result: MEM[0x100] = R24
0x66:    NOP
0x67:    RJMP 0x66        ; Endless while loop
```

Example 17: AVR C Code and Disassembly

```
; absolute( int8_t x ) {...}
0x4B:    PUSH R28        ; Save YL (R28) and YH (R29) on stack
0x4C:    PUSH R29
0x4D:    PUSH R0         ; Save R0 on stack
0x4E:    IN R28, 0x3D    ; Set the stack frame pointer: YH:YL (R29:R28) = SPH:SPL
0x4F:    IN R29, 0x3E
;-----------------------------------------------------------------
0x50:    STD Y+1, R24    ; Save R24 at MEM[Y+1]
0x51:    LDD R24, Y+1    ; Load the argument from memory: R24 = MEM[Y+1]
0x52:    CP R24, R1      ; Signed compare: R24 - R1
0x53:    BRGE 0x57       ; Branch if greater than: R24 > 0
0x54:    LDD R24, Y+1    ; R24 = MEM[Y+1]
0x55:    NEG R24         ; R24 = -R24 (negate)
0x56:    RJMP 0x58
0x57:    LDD R24, Y+1    ; Load the argument from memory: R24 = MEM[Y+1]
;-----------------------------------------------------------------
0x58:    POP R0          ; Restore R0
0x59:    POP R29         ; Restore R29
0x5A:    POP R28         ; Restore R28
0x5B:    RET             ; Return from subroutine
```

Integer Division

How to compute integer division?

Dividend = Divisor * Quotient + Remainder

```
#include <avr/io.h>
#include <stdint.h>

uint16_t dividend = 201;
uint8_t  divisor  = 30;
uint16_t q, r;

void main(void) {
    q = dividend / divisor;
    r = dividend % divisor;
    while(1);
    return;
}
```

Restoring Division (Shift–Subtract) Algorithm

```
dividend = 201
divisor  = 30

r = 0  // Remainder
q = 0  // Quotient

// operate one per bit, from MSB to LSB.
for i = n-1 downto 0:
    r = (r << 1) | ((dividend >> i) & 1)
    r = r – divisor      // subtract
    if r >= 0:
        q[i] = 1         // keep subtraction
    else:
        r = r + divisor // restore remainder
        q[i] = 0
```

Restoring Division Algorithm

Pseudo code: non-restoring-style variant

```
dividend = 201
divisor  = 30

r = 0  // Remainder
q = 0  // Quotient

// operate one per bit, from MSB to LSB.
for i = n-1 downto 0:
    r = (r << 1) | ((dividend >> i) & 1)
    if r >= divisor:
        r = r – divisor // subtract
        q[i] = 1
    else:
        q[i] = 0
```

```
  00110010 (remainder)
- 00011110 (divisor)
  --------
  00010100
```

Example

```
Dividend = 11001001
Divisor  = 00011110
    r     | r >= divisor | q[i] | r (new)
00000001 |      no       |  0   | 00000001
00000011 |      no       |  0   | 00000011
00000110 |      no       |  0   | 00000110
00001100 |      no       |  0   | 00001100
00011001 |      no       |  0   | 00011001
00110010 |      yes      |  1   | 00010100
00101000 |      yes      |  1   | 00001010
00010101 |      no       |  0   | 00010101
q = 00000110 = 6
r = 00010101 = 21
```

Example 18: Restoring Division (Shift–Subtract) Algorithm

```
#include <avr/io.h>
#include <stdint.h>

// conditions: divisor is not zero and remainder is not NULL.
uint8_t udiv(uint8_t dividend, uint8_t divisor, uint8_t *remainder) {
    uint16_t q = 0, r = 0;
    for (int8_t i = 7; i >= 0; i--) { // starting from MSB to LSB
        // Shift remainder left and bring down next dividend bit
        r = (r << 1) | ((dividend >> i) & 1);
        if (r >= divisor) {
            r -= divisor;
            q |= (1U << i);
        }
    }
    *remainder = r; // Save the remainder
    return q; // Return the quotient
}

uint8_t q, r; // global variables

void main(void) {
    q = udiv(201, 30, &r);
    while(1);
    return;
}
```

Example 19: AVR Assembly Code – Unsigned 8-bit Division

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
     LDI R24, 201    ; dividend
     LDI R22, 30     ; divisor
     CALL udiv8
 DONE:
     RJMP DONE
```

```
; udiv8: Unsigned 8-bit division (shift–subtract)
; R24 = dividend, R22 = divisor
‘ R21 = quotient (q), R20 = remainder (r)
udiv8:
    CLR R21         ; q = 0
    CLR R20         ; r = 0
    LDI R18, 8      ; 8 bits to process
BIT_LOOP:
    LSL R20         ; remainder: arithmetic shift-left
    LSL R24         ; dividend: arithmetic shift-left
    ADC R20, R1     ; add with carry, R1 is zero
    CP  R20, R22    ; signed compare: r - divisor
    BRLO NO_SUB     ; branch if lower (r < divisor)
    SUB R20, R22    ; r = r - divisor
    LSL R21         ; q = q << 1
    ORI R21, 1      ; q |= 1
    RJMP NEXT_BIT
NO_SUB:
    LSL R21         ; q = q << 1
NEXT_BIT:
    DEC R18         ; R18 = R18 - 1
    BRNE BIT_LOOP   ; branch if not equal zero
    RET             ; return from subroutine
```

Example 20: Greatest Common Divisor (C Code)

```
#include <avr/io.h>
#include <stdint.h>

uint16_t gcd( uint16_t a, uint16_t b ) {
  do {
    if (a < b) { uint16_t t = a; a = b; b = t; }
    a = a - b;
  } while (b > 0);
  return a;
}

uint16_t x = 1025, y = 250, result; // gcd(1025,500) = 25

void main(void) {
    result = gcd(x, y);
    while (1);
    return;
}
```

Example 21: Greatest Common Divisor (AVR Assembly)

```
.include "m328pdef.inc"
.CSEG
.ORG 0x0000
    LDI R25, 0x04 ; a = 1025 (0x0401)
    LDI R24, 0x01
    LDI R23, 0x00 ; b =  250 (0x00FA)
    LDI R22, 0xFA
    CALL gcd
 DONE:
    NOP
    RJMP DONE
```

```
; uint16_t gcd(uint16_t a, uint16_t b)
; a = R25:R24, b = R23:R22, gcd in R25:R24
gcd:
gcd_loop:
    CP  R24, R22    ; Signed compare (low byte)
    CPC R25, R23    ; Signed compare (high byte)
    BRSH skip_swap  ; Branch if same or higher (a>=b)
    ; Swap a and b
    MOV R21, R24    ; t (low byte)  = a (low byte)
    MOV R20, R25    ; t (high byte) = a (high byte)
    MOV R24, R22    ; a (low byte)  = b (low byte)
    MOV R25, R23    ; a (high byte) = b (high byte)
    MOV R22, R21    ; b (low byte)  = t (low byte)
    MOV R23, R20    ; b (high byte) = t (high byte)
skip_swap:
    ; a = a - b
    SUB R24, R22    ; R24 = R24 - R22
    SBC R25, R23    ; R25 = R25 - R24 – C
    ; test if b (R23:R22) is zero
    TST R22         ; test R22 if zero
    BRNE gcd_loop   ; branch if not equal zero
    TST R23         ; test R23 if zero
    BRNE gcd_loop   ; branch if not equal zero
    RET
```

Example 21: Inline AVR Assembly

```
int8_t add( int8_t a, int8_t b ) {
  int8_t result;
  // result = a + b;
  asm volatile (
    // AVR instructions
    "add %[reg_a], %[reg_b] \n\t"     // add b to a
    : [reg_a] "=r" (result)           // the output list
    : "[reg_a]" (a), [reg_b] "r" (b)  // the input list
    : "cc"                            // the clobber list
  );
  return result;
}
```

```
uint8_t x = 125, y = 20;

void main(void) {
   X = add(x, y);
   while (1);
   return;
}
```

- The **input list** specifies which registers are used as input operands.
- The **output list** specifies which registers are used to store the result.
- The **clobber list** tells the compiler which registers or memory locations the inline assembly modifies.
- The **"cc"** flag refers to the condition code register. It tell the compiler that the status flags of AVR in SREG are affected by instructions.
- The **"r"** constraint indicates that the operand should be placed in a general-purpose register.

Example 22: Inline AVR Assembly

```
#include <avr/io.h>
#include <stdint.h>


void swap(uint8_t *a, uint8_t *b) {
   // Inline AVR assembly
   asm volatile (
      // AVR instructions
      "LD r16, %a[a_ptr]      \n\t"  // Load *a into r16
      "LD r17, %a[b_ptr]      \n\t"  // Load *b into r17
      "ST %a[a_ptr], r17      \n\t"  // Store r17 into *a
      "ST %a[b_ptr], r16      \n\t"  // Store r16 into *b
      :                                 // the output list
      : [a_ptr] "e" (a), [b_ptr] "e" (b) // the input list
      : "r16", "r17", "memory"          // the clobber list
   );
}
```

```
uint8_t x = 125, y = 20;

void main(void) {
   swap(&x, &y);
   while (1);
   return;
}
```

- For AVR, the **"e"** constraint specifies the pointer will be in a register pair:
  - X (R27:R26)
  - Y (R29:R28)
  - Z (R31:R30)

Example 23: Argument Passing in C/C++

1) Pass by pointer (C style)

- The arguments a and b of the function are pointers to variables.
- Addresses of the variables must be passed to the function call explicitly.

```
void swap(uint8_t *a, uint8_t *b);
```

```
uint8_t x = 10, y = 20;
swap(&x, &y); // pass addresses
```

2) Pass by reference (C++ style)

- The arguments a and b are references to variables.
- The function works directly on the original variables, no copying is needed.
- Syntax is C++ only (not standard C).

```
void swap(uint8_t &a, uint8_t &b);
```

```
uint8_t x = 10, y = 20;
swap(x, y); // pass references
```