

THE AVR MICROCONTROLLER AND EMBEDDED SYSTEMS USING ASSEMBLY AND C



SECOND EDITION: BASED ON
ATMEGA328 AND ARDUINO BOARDS

MUHAMMAD ALI MAZIDI,
SEPEHR NAIMI, AND
SARMAD NAIMI

Lecture # 06M: Assembly Language Programming for ATmega328P



Course Teacher: **Prof. Dr. Engr. Muhibul Haque Bhuyan**
Professor, Department of EEE
American International University-Bangladesh (AIUB)
Dhaka, Bangladesh

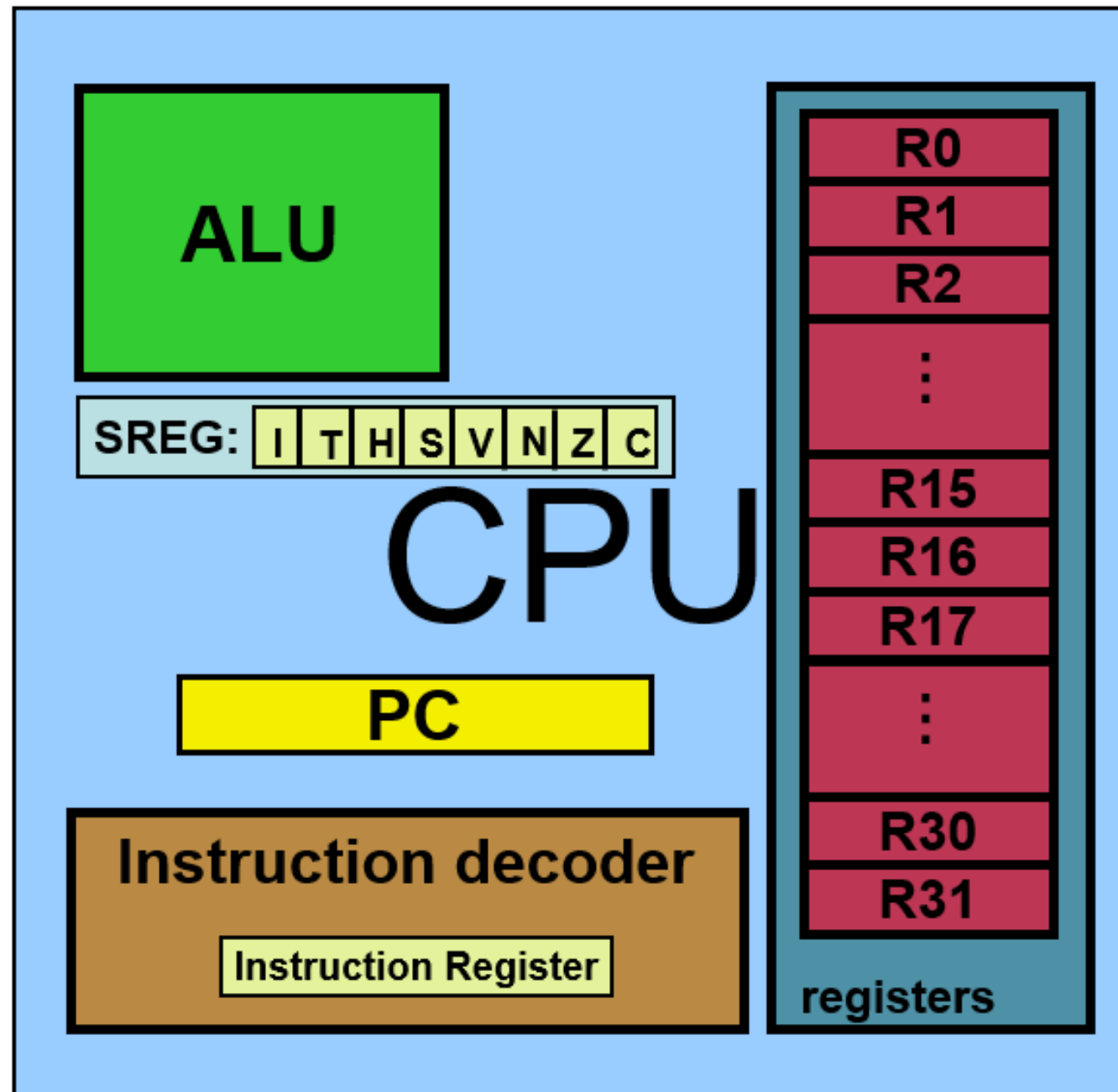


WHERE LEADERS ARE CREATED

Outline

- ❑ Arduino AVR Architecture
- ❑ Assembly Language Programming
- ❑ Examples of Assembly Programming
- ❑ Flow Chart Drawing and Algorithm Development

AVR Architecture of CPU



- Arithmetic Logic Unit (ALU)
- 32 General Purpose Register (GPR)
← each 8-bit
- Program Counter (PC)
- Instruction Decoder
- Status Register (SREG)



AVR Flags/Status Register

- ❑ C = Carry Flag, This flag is set whenever there is a carry out from the D7 bit after an arithmetic operation (Addition, subtraction, increment, decrement, etc.). This flag bit is affected after an 8-bit addition or subtraction.
- ❑ Z = Zero Flag, This flag is affected after an arithmetic or logic operation. If the result is zero then $Z = 1$, else $Z = 0$.
- ❑ N = Negative Flag, It reflects the result of an arithmetic operation. If the D7 bit of the result is zero, then $N = 0$ and the result is positive. Else $N = 1$ and the result is negative.
- ❑ V = Overflow Flag,
- ❑ S = Sign Flag,
- ❑ H = Half Carry Flag, this bit is set if there is a carry from D3 to D4 bit after ADD or SUB instruction.
- ❑ T = Bit Copy Storage. Used as a temporary storage for bit. It can be used to copy a bit from a GPR to another GPR.
- ❑ I = Global Interrupt Enable



Direct Data Register Method

- ☐ To assign input- **0**
- ☐ To assign output- **1**
- ☐ DDRB= 0x 05;
- ☐ Or DDRB= 0b 00000101; PB0 and PB2 are output pin while rest are input.
- ☐ To write High-1
- ☐ To write Low-0
- ☐ PORTB= 0x01;
- ☐ PORTB= 0b00000001; write PB0 as high.

PORTB Register

| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

DDRB Register

| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
|-----|-----|-----|-----|-----|-----|----------|----------|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |



WHERE LEADERS ARE CREATED

Instruction Set Summary

| Mnemonics | Operands | Description | Operations | Flags | Clocks |
|-----------|----------|---------------------------------|---------------------------------|-----------|--------|
| ADD | Rd,Rr | Add two Registers | $Rd \leftarrow Rd + Rr$ | Z,C,N,V,H | 1 |
| ADC | Rd,Rr | Add with Carry two Registers | $Rd \leftarrow Rd + Rr + C$ | Z,C,N,V,H | 1 |
| SUB | Rd,Rr | Subtract two Registers | $Rd \leftarrow Rd - Rr$ | Z,C,N,V,H | 1 |
| SUBI | Rd,K | Subtract Constant from Register | $Rd \leftarrow Rd - K$ | Z,C,N,V,H | 1 |
| AND | Rd,Rr | Logical AND Registers | $Rd \leftarrow Rd \cdot Rr$ | Z,N,V | 1 |
| OR | Rd,Rr | Logical OR Registers | $Rd \leftarrow Rd \vee Rr$ | Z,N,V | 1 |
| NEG | Rd | Two's Complement | $Rd \leftarrow 0x00 - Rd$ | Z,C,N,V,H | 1 |
| INC | Rd | Increment | $Rd \leftarrow Rd + 1$ | Z,N,V | 1 |
| DEC | Rd | Decrement | $Rd \leftarrow Rd - 1$ | Z,N,V | 1 |
| MUL | Rd,Rr | Multiply Unsigned | $R1:R0 \leftarrow Rd \times Rr$ | Z,C | 2 |



WHERE LEADERS ARE CREATED

Instruction Set Summary

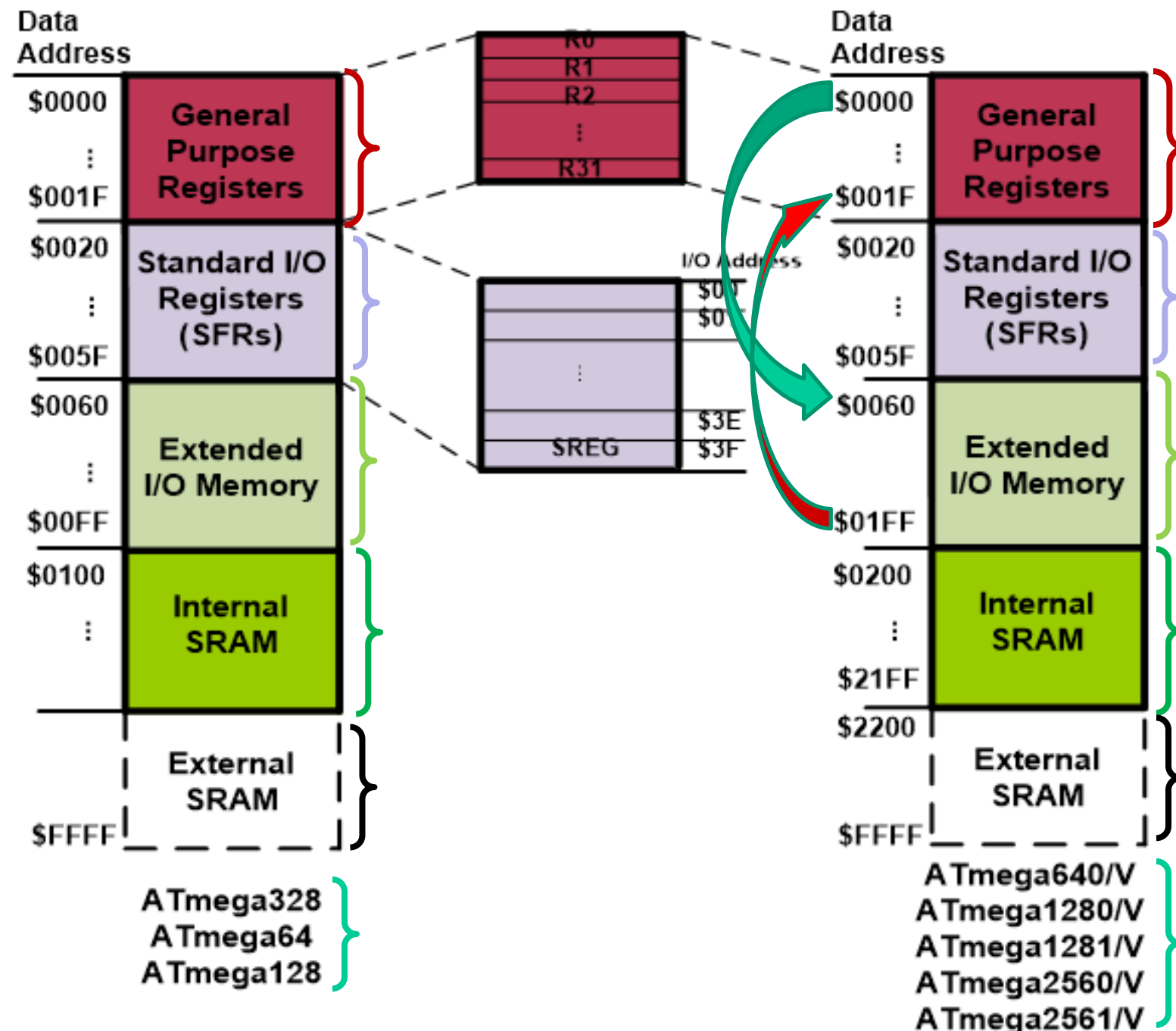
| Mnemonics | Operands | Description | Operations | Flags | Clocks |
|-----------|----------|---|------------------------------|-------|--------|
| TST | Rd | Test for Zero or Minus | $Rd \leftarrow Rd \cdot Rd$ | Z,N,V | 1 |
| CLR | Rd | Clear Register | $Rd \leftarrow Rd \oplus Rd$ | Z,N,V | 1 |
| MOV | Rd, Rr | Move Between Registers | $Rd \leftarrow Rr$ | None | 1 |
| LDI | Rd, K | Load Immediate | $Rd \leftarrow K$ | None | 1 |
| LDS | Rd, K | Load Direct from data space location K | $Rd \leftarrow [K]$ | None | 1 |
| IN | Rd, P | From In Port P/ address to Rd register | $Rd \leftarrow P$ | None | 1 |
| OUT | P, Rr | From Rr register to Out Port P/ address | $P \leftarrow Rr$ | None | 1 |
| PUSH | Rr | Push Register on Stack | $STACK \leftarrow Rr$ | None | 2 |
| POP | Rd | Pop Register from Stack | $Rd \leftarrow STACK$ | None | 2 |
| NOP | | No Operation | None | 1 | |
| BREAK | | Break | For On-chip Debug Only | None | N/A |



Instruction Set Summary

| Mnemonics | Operands | Description | Operations | Flags | Clocks |
|-----------|----------|--|---|-------|--------|
| STS | K, Rr | Store Rr to data space location K. | $[K] \leftarrow Rr$ | None | 1 |
| SBI | A, b | Sets a specified bit in an I/O Register | Sets the b no. bits of the A register | None | 1 |
| CBI | A, b | Resets a specified bit in an I/O Register | Resets the b no bits of the A register | None | 1 |
| SBIC | A, b | Skip if Bit in I/O Register is Cleared/Reset | If b no bit of A register is 0, then skip PC by 2 or 3. | None | 1 |
| SBIS | A, b | Skip if Bit in I/O Register is set | If b no bit of A register is 1, then skip PC by 2 or 3. | None | 1 |
| RJUMP | K | Relative jump | $PC \leftarrow PC + K + 1$ | None | 1 |

Memory Locations



STS 0x60, R0

LDS R31, 0x01FF



WHERE LEADERS ARE CREATED

Register Contents

SBI DDRB, 3;

DDRB Register

| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

CBI DDRB, 5;

DDRB Register

| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

SBIC R7, 5;

R7 Register

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

PC = PC+2 or 3

SBIS R8, 5;

R8 Register

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

PC = PC+2 or 3



Programming Examples

Example: State the contents of R20, R21, and data memory location of 0x120 after executing the following program.

```
LDI R20, 5;  
LDI R21, 2;  
ADD R20,R21;  
ADD R20,R21;  
STS 0x120, R20;
```

Solution:

```
LDI R20, 5;      R20=5  
LDI R21, 2;      R21=2  
ADD R20, R21;    R20=5+2=7  
ADD R20, R21;    R20= 7+2=9  
STS 0x120, R20;  0x120 = 9
```

R20= 9

R21= 2

0x120= 9

Example: State the contents of RAM locations of \$212, \$213, \$214, \$215, and \$216 after executing the following program.

```
LDI R16, 0x99  
STS 0x212, R16  
LDI R16, 0x85  
STS 0x213, R16  
LDI R16, 0x3F  
STS 0x214, R16  
LDI R16, 0x63  
STS 0x215, R16  
LDI R16, 0x12  
STS 0x216, R16
```

Solution:

| | |
|-------|------|
| 0x212 | 0x99 |
| 0x213 | 0x85 |
| 0x214 | 0x3F |
| 0x215 | 0x63 |
| 0x216 | 0x12 |



Difference Between Assembly and C

Assembly Code Example

sei ; *set Global Interrupt Enable*
sleep; *enter sleep, waiting for interrupt*
; note: will enter sleep before any pending interrupt(s)

C Code Example

```
__enable_interrupt(); /* set Global Interrupt Enable */  
__sleep(); /* enter sleep, waiting for interrupt */  
/* note: will enter sleep before any pending interrupt(s) */
```




EEPROM Control Register (EECR)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|-----------|---|-------|-------|------|------|
| | | EEPM(1:0) | | EERIE | EEMPE | EEPE | EERE |

SBIC EECR, EEPE; If EEPE bit of EECR is 0 then the Program Counter (PC) will skip 2 or 3 steps.



Difference Between Assembly and C

Assembly Code Example

```
EEPROM_write:
; Wait for completion of previous write
sbic EECR,EEPE
rjmp EEPROM_write
; Set up address (r18:r17) in address register
out EEARH, r18
out EEARL, r17
; Write data (r16) to Data Register
out EEDR,r16
; Write logical one to EEMPE
sbi EECR,EEMPE
; Start eeprom write by setting EEPE
sbi EECR,EEPE
ret
```

C Code Example

```
void EEPROM_write(unsigned int uiAddress,
unsigned char ucData)
{
    /* Wait for completion of previous write */
    while((EECR & (1<<EEPE))
    ;
    /* Set up address and Data Registers */
    EEAR = uiAddress;
    EEDR = ucData;
    /* Write logical one to EEMPE */
    EECR |= (1<<EEMPE);
    /* Start eeprom write by setting EEPE */
    EECR |= (1<<EEPE);
}
```



Functions of Assembly and C

The next code examples show assembly and C functions for reading the EEPROM.

The examples assume that interrupts are controlled so that no interrupts will occur during the execution of these functions.

Assembly Code Example

```
EEPROM_read:
; Wait for completion of previous write
sbic EECR,EEPE
rjmp EEPROM_read
; Set up address (r18:r17) in address
register
out EEARH, r18
out EEARL, r17
; Start eeprom read by writing EERE
sbi EECR,EERE
; Read data from Data Register
in r16,EEDR
ret
```

C Code Example

```
unsigned char
EEPROM_read(unsigned int uiAddress)
{
/* Wait for completion of previous write */
while((EECR & (1<<EEPE))
;
/* Set up address register */
EEAR = uiAddress;
/* Start eeprom read by writing EERE */
EECR |= (1<<EERE);
/* Return data from Data Register */
return EEDR;
}
```



Functions of Assembly and C

Bit 0 – IVCE: Interrupt Vector Change Enable

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts,

Assembly Code Example

```
Move_interrupts:
; Enable change of Interrupt
Vectors
ldi r16, (1<<IVCE)
out MCUCR, r16
; Move interrupts to Boot Flash
section
ldi r16, (1<<IVSEL)
out MCUCR, r16
ret
```

C Code Example

```
void Move_interrupts(void)
{
/* Enable change of Interrupt Vectors */
MCUCR = (1<<IVCE);
/* Move interrupts to Boot Flash section */
MCUCR = (1<<IVSEL);
}
```




Functions of Assembly and C

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

If an interrupt occurs between the two instructions accessing the 16-bit register, the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

Assembly Code Example

```
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
...
```

C Code Example

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```



Functions of Assembly and C

The following code examples show how to do an atomic read of the TCNT1 Register contents.

Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle

Assembly Code Example

```
TIM16_ReadTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore global interrupt flag
out SREG,r18
ret
```

C Code Example

```
unsigned int TIM16_ReadTCNT1( void )
{
    unsigned char sreg;
    unsigned int i;
    /* Save global interrupt flag */
    sreg = SREG;
    /* Disable interrupts */
    _CLI();
    /* Read TCNT1 into i */
    i = TCNT1;
    /* Restore global interrupt flag */
    SREG = sreg;
    return i;
}
```



Functions of Assembly and C

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing any of the OCR1A/B or ICR1 Registers can be done by using the same principle

Assembly Code Example

```
TIM16_ReadTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore global interrupt flag
out SREG,r18
ret
```

C Code Example

```
unsigned int TIM16_ReadTCNT1( void )
{
    unsigned char sreg;
    unsigned int i;
    /* Save global interrupt flag */
    sreg = SREG;
    /* Disable interrupts */
    _CLI();
    /* Read TCNT1 into i */
    i = TCNT1;
    /* Restore global interrupt flag */
    SREG = sreg;
    return i;
}
```



Functions of Assembly and C

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

Assembly Code Example

```
TIM16_WriteTCNT1:
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Set TCNT1 to r17:r16
out TCNT1H,r17
out TCNT1L,r16
; Restore global interrupt flag
out SREG,r18
ret
```

C Code Example

```
void TIM16_WriteTCNT1( unsigned int i )
{
    unsigned char sreg;
    unsigned int i;
    /* Save global interrupt flag */
    sreg = SREG;
    /* Disable interrupts */
    _CLI();
    /* Set TCNT1 to i */
    TCNT1 = i;
    /* Restore global interrupt flag */
    SREG = sreg;
}
```


Difference Between Assembly and C

| | Assembly Code Example | C Example | Comments |
|---|---|---|--|
| 1 | <pre>ldi r16, (1<<TWINT) (1<<TWSTA) (1<<TWEN) out TWCR, r16</pre> | <pre>TWCR = (1<<TWINT) (1<<TWSTA) (1<<TWEN)</pre> | Send START condition |
| 2 | <pre>wait1: in r16,TWCR sbrs r16,TWINT rjmp wait1</pre> | <pre>while (!(TWCR & (1<<TWINT))) ;</pre> | Wait for TWINT Flag set. This indicates that the START condition has been transmitted |
| 3 | <pre>in r16,TWSR andi r16, 0xF8 cpi r16, START brne ERROR</pre> | <pre>if ((TWSR & 0xF8) != START) ERROR();</pre> | Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR |
| | <pre>ldi r16, SLA_W out TWDR, r16 ldi r16, (1<<TWINT) (1<<TWEN) out TWCR, r16</pre> | <pre>TWDR = SLA_W; TWCR = (1<<TWINT) (1<<TWEN);</pre> | Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address |
| 4 | <pre>wait2: in r16,TWCR sbrs r16,TWINT rjmp wait2</pre> | <pre>while (!(TWCR & (1<<TWINT))) ;</pre> | Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received. |

Difference Between Assembly and C

| | | | |
|---|--|--|---|
| 5 | <pre>in r16,TWSR andi r16, 0xF8 cpi r16, MT_SLA_ACK brne ERROR</pre> | <pre>if ((TWSR & 0xF8) != MT_SLA_ACK) ERROR();</pre> | Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR. |
| | <pre>ldi r16, DATA out TWDR, r16 ldi r16, (1<<TWINT) (1<<TWEN) out TWCR, r16</pre> | <pre>TWDR = DATA; TWCR = (1<<TWINT) (1<<TWEN);</pre> | Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data. |
| 6 | <pre>wait3: in r16,TWCR sbrs r16,TWINT rjmp wait3</pre> | <pre>while (!(TWCR & (1<<TWINT))) ;</pre> | Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received. |
| 7 | <pre>in r16,TWSR andi r16, 0xF8 cpi r16, MT_DATA_ACK brne ERROR</pre> | <pre>if ((TWSR & 0xF8) != MT_DATA_ACK) ERROR();</pre> | Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR. |
| | <pre>ldi r16, (1<<TWINT) (1<<TWEN) (1<<TWSTO) out TWCR, r16</pre> | <pre>TWCR = (1<<TWINT) (1<<TWEN) (1<<TWSTO);</pre> | Transmit STOP condition |



Example 1 of Assembly Program

Determine the contents of the memory locations in binary format in Table 1 based on the following assembly language instructions:

```
LDI R17, 0xE6;  
STS 0x21FB, R17;  
LDI R22, 0xE0;  
LDI R30, 0xD2;  
ADD R30, R22;  
STS 0x21FC, R30;  
SUBI R30, 0xA1;  
STS 0x21FD, R30;  
AND R30, R17;  
STS 0x21FE, R30;  
OR R30, R17;  
STS 0x21FF, R30;
```

| Memory Location | Contents in Binary |
|-----------------|--------------------|
| 0x21FB | |
| 0x21FC | |
| 0x21FD | |
| 0x21FE | |
| 0x21FF | |



Example 1 of Assembly Program

Answers:

```
LDI R17, 0xE6;      // R17 register gets an immediate value of 0xE6
STS 0x21FB, R17;     // Memory location, 0x21FB gets a value of 0xE6 from the register R17
LDI R22, 0xE0;      // R22 register gets an immediate value of 0xE0
LDI R30, 0xD2;      // R30 register gets an immediate value of 0xD2
ADD R30, R22;        // R30 register gets a value of addition results of R22 and R30 registers' contents, i.e., R30 = R30 + R22 = 0xD2 + 0xE0
                    // = 0xB2; carry would be neglected
STS 0x21FC, R30;     // Memory location, 0x21FC gets a value of 0xB2 from the register R30
SUBI R30, 0xA1;      // R30 register gets a value of subtraction results of 0xA1 from R30 register's contents, i.e., R30 = R30 - 0xA1 = 0xB2 +
                    // 0xA1 = 0x11; borrow won't be generated
STS 0x21FD, R30;     // Memory location, 0x21FD gets a value of 0x11 from the register R30
AND R30, R17;        // R30 register gets a value of logical AND operation results
                    // R17 = 0x11 AND 0xE6 = 0x00010001 AND 0x111100110 = 0x00
STS 0x21FE, R30;     // Memory location, 0x21FE gets a value of 0x00 from the register R30
OR R30, R17;         // R30 register gets a value of logical OR operation results of
                    // = 0x00 OR 0xE6 = 0x00000000 OR 0x111100110 = 0xE6
STS 0x21FF, R30;     // Memory location, 0x21FF gets a value of 0xE6 from the register R30
```

| Memory Location | Contents in Binary |
|-----------------|--------------------|
| 0x21FB | 0b11100110 |
| 0x21FC | 0b10110010 |
| 0x21FD | 0b00010001 |
| 0x21FE | 0b00000000 |
| 0x21FF | 0b11100110 |

Example 2 of Assembly Program

Prepare an assembly program to set the Sleep Mode Control Register (SMCR) so that the system remains in power-save mode as a whole and power reduction mode (PRR) for its PRTIM0 and PRSPI.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------------|---|---|---|---|-----|-----|-----|-----|
| 0x33 (0x53) | – | – | – | – | SM2 | SM1 | SM0 | SE |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

SMCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------------|-------|--------|--------|---|--------|-------|----------|-------|
| (0x64) | PRTWI | PRTIM2 | PRTIM0 | – | PRTIM1 | PRSPI | PRUSART0 | PRADC |
| Read/Write | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

PRR

| SM2 | SM1 | SM0 | Sleep Mode |
|-----|-----|-----|---------------------------------|
| 0 | 0 | 0 | Idle |
| 0 | 0 | 1 | ADC Noise Reduction |
| 0 | 1 | 0 | Power-down |
| 0 | 1 | 1 | Power-save |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Standby ⁽¹⁾ |
| 1 | 1 | 1 | External Standby ⁽¹⁾ |



Example 2 of Assembly Program

Answers:

To set the Sleep Mode Control Register (SMCR) in the power save mode, we need to set various bits of SMCR as follows as per the bit setting of the given table (SE bit, i.e. bit 0 must be set to logical HIGH to enable the Sleep Mode):

0000 0111b = 07h

To set the Power Reduction Register (PRR) for saving the power of PRTIM0 and PRSPI, we must enable bit 5 and bit 2 of the PRR register as follows:

0010 0100b = 24h

Therefore, the assembly program should be as follows:

; Set SMCR to 0x07

LDI R15,0x07; // Sleep Mode Control Register control data is loaded into the register R15

OUT SMCR, R15; // R15 sends the control data to the SMCR to set it in power save mode

; Set PRR to 0x24

LDI R14, 0x24; // Power Reduction Register (PRR) control data is loaded into the register R14

OUT PRR, R14; // R14 sends the control data to the PRR to set it in the power reduction modes of Timer0 and SPI



Flowchart

The first design of flowchart goes back to 1945 which was designed by John Von Neumann. Unlike an algorithm, Flowchart uses different symbols to design a solution to a problem. It is another commonly used programming tool. By looking at a Flowchart, one can understand the operations and sequence of operations performed in a system.

A flowchart is often considered a blueprint of a design used for solving a specific problem.

A flowchart is a diagrammatic representation of an algorithm. Flowcharts are very helpful in writing programs and explaining programs to others.

Though, flowcharts are useful in efficient coding, debugging, and analysis of a program, drawing flowcharts for very complicated in case of complex programs is often ignored.

Flowcharts use special shapes to represent different types of actions or steps in a process. Lines and arrows show the sequence of the steps and the relationships among them. These are known as flowchart symbols. So, flowchart symbols are specific shapes used to create a visual representation of a program.



Advantages of Flowchart

1. A flowchart is an excellent way of communicating the logic of a program.
2. Easy and efficient to analyze the problem using a flowchart.
3. During the program development cycle, the flowchart plays the role of a blueprint, which makes the program development process easier.
4. After the successful development of a program, it needs continuous timely maintenance during the course of its operation. The flowchart makes a program or system maintenance easier.
5. It is easy to convert the flowchart into any programming language code.

Commonly used Operators in Flowchart

While creating flowchart, the following mathematical operators are used.

| Operator | Meaning | Example | Remarks |
|----------|----------------|----------|--|
| + | Addition | $A + B$ | Values in A and B will be added |
| - | Subtraction | $A - B$ | Values of B will be subtracted from that of A |
| * | Multiplication | $A * B$ | Values of B will be multiplied by that of A |
| / | Division | A/B | Values of A will be divided by that of B |
| ^ | Power | A^3 | Values of A will be raised by a power of 3 |
| % | Remainder | $A \% B$ | Values of A will be divided by that of A to get a remainder (if any) |

Commonly used Operators in Flowchart

While creating flowchart, the following relational operators are used.

| Operator | Meaning | Example | Remarks |
|----------|--------------------------|-----------------------|--|
| < | Less than | $A < B$ | Value in A is less than that of B |
| <= | Less than or equal to | $A \leq B$ | Value in A is less than or equal to that of B |
| = or == | Equal to | $A = B$ or $A == B$ | Values of A and B are equal |
| # or != | Not Equal to | $A \# B$, $A \neq B$ | Values of A and B are not equal |
| > | Greater than | $A > B$ | Value in A is greater than that of B |
| >= | Greater than or equal to | $A \geq B$ | Value in A is greater than or equal to that of B |




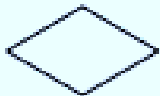
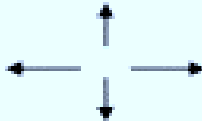




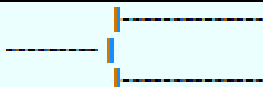


Commonly used Operators in Flowchart

While creating flowchart, the following logical operators are used.

| Operator | Meaning | Example | Remarks |
|----------|-------------|--------------|---------------------------------|
| AND | Logical AND | $A \wedge B$ | Values in A and B will be ANDed |
| OR | Logical OR | $A \vee B$ | Values in A and B will be ORed |
| NOT | Logical NOT | $\neg A$ | Values in A will be inverted |

Commonly used Symbols in Flowchart

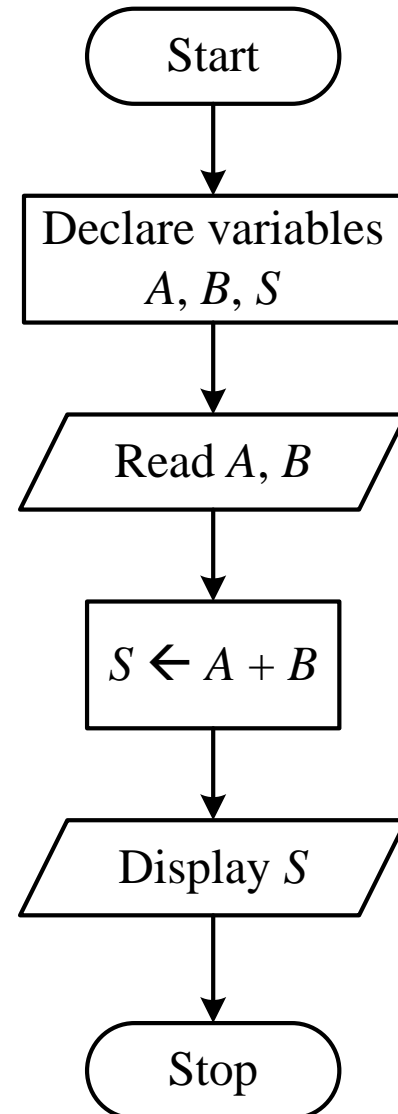
| Symbol Name | Symbol | function |
|---------------|---|--|
| Oval |  | Used to represent start and end of flowchart |
| Parallelogram |  | Used for input and output operation |
| Rectangle |  | Processing: Used for arithmetic operations and data-manipulations |
| Diamond |  | Decision making. Used to represent the operation in which there are two/three alternatives, true and false etc |
| Arrows |  | Flow line Used to indicate the flow of logic by connecting symbols |
| Circle |  | Page Connector |
| |  | Off Page Connector |
| |  | Predefined Process /Function Used to represent a group of statements performing one processing task. |
| |  | Preprocessor |
| |  | Comments |

While creating a flowchart, the following symbols are used.

- Rectangle Shape - Represents a process
- Oval or Pill Shape - Represents the start or end
- Diamond Shape - Represents a decision
- Parallelogram - Represents input/output
- Circle - Represents reference/page connector
- Arrow - Represents a connection between shapes

Examples of Flowchart

Draw a flowchart along with its algorithm to add two numbers entered by the user.

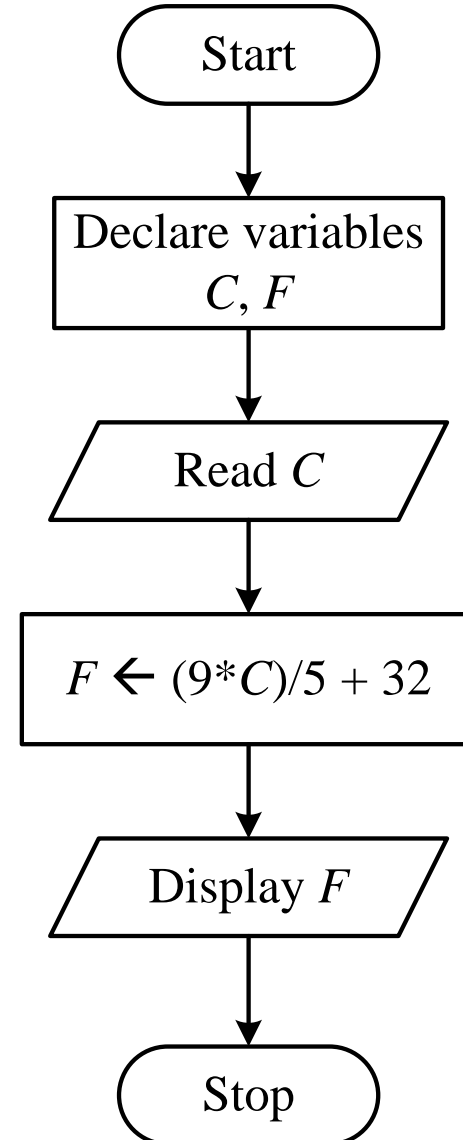


An algorithm to find the sum of two numbers:

- Step 1: Start
- Step 2: Declare three variables
- Step 3: Read numbers and store them in the declared variables
- Step 4: Sum these two numbers and assign them to the declared variable
- Step 5: Display the sum
- Step 6: Stop

Examples of Flowchart

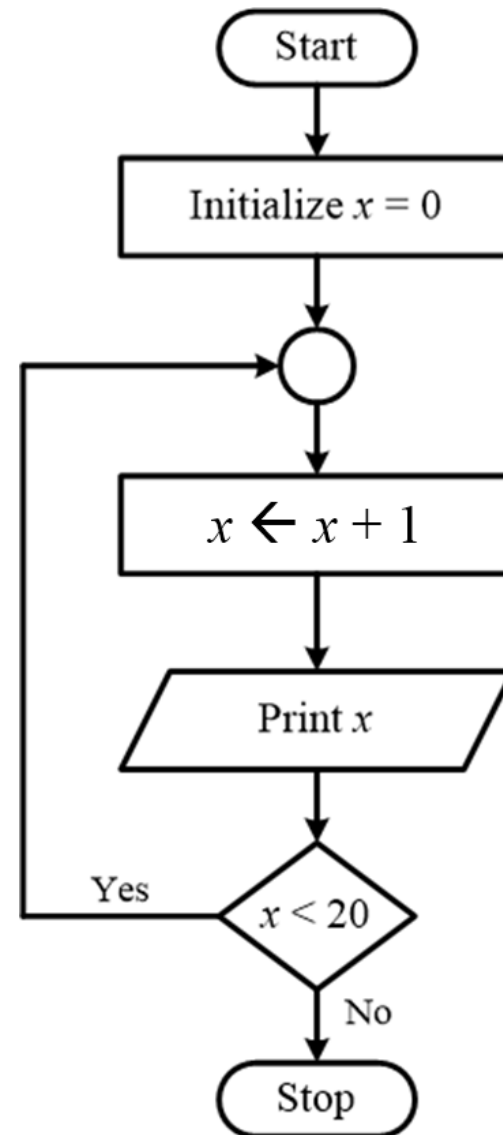
Draw a flowchart along with its algorithm to convert temperature from Celsius to Fahrenheit.



- Step 1: Start
- Step 2: Declare two variables
- Step 3: Read the temperature value in Celsius scale and store it in one of the declared variables
- Step 4: Apply the formula of temperature conversion i.e., $F = (9 * C) / 5 + 32$ to the convert Celsius value in Fahrenheit scale and store it in the declared variable
- Step 5: Display the temperature in Fahrenheit scale
- Step 6: Stop

Examples of Flowchart

Draw a flowchart along with its algorithm to print numbers from 1 to 20.



- Step 1: Start
- Step 2: Declare and initialize a variable with a 0 value
- Step 3: Increment the variable value by 1 and store it in the same declared variable
- Step 4: Print the value stored in the variable
- Step 5: If the variable is < 20 then go to step 3, otherwise go to the next step
- Step 6: Stop



Thanks for attending....

