MODULE – II AVR Programming in C

I/O port programming in AVR – I/O bit Manipulation Programming.
Data types and Time Delays in C - I/O programming in C – Logic Operations in C- Data Conversion Programs in C – Data Serialization in C

I/O port Programming in AVR

- In the AVR family, there are many ports for I/O operation. A total of 32 pins are set aside for the 4 ports PORTA, PORTB, PORTC & PORTD.
- The rest of pins designated as:
  - ✓ Vcc → This pin provides supply voltage to the chip typical voltage is +5V.
  - ✓ Avcc → It is the supply voltage for PORT A and the A/D converter.
  - ✓ AREF → It is the analog reference pin for ADC.
  - ✓ GND, AGND → Two pins are also used for ground.
  - ✓ XTAL1 & XTAL2 → Used to connect the clock source.
  - ✓ RESET → Pin 9, If this pin goes to low the micro controller will reset and terminate all activities.

I/O port pins and their functions

- The number of ports in the AVR family varies depending on the number of pins on the chip.
  - ✓ 8 pin AVR has – PORT B only
  - ✓ 64 pin – PORT A through PORT F (6)
  - ✓ 100 pin – PORT A through PORT L (12)
  - ✓ 40 pin – PORTA,PORTB,PORTC,PORTD (4)
- To use any of these ports, it must be programmed.
- In addition to simple I/O each port has other functions such as ADC, timers, interrupts & serial communication.
- ***Each port has THREE I/O registers associated with it***.
- They are designated as **PORTx, DDRx** & **PINx**.
- Eg:- For port B, we have PORT B,DDRB & PINB
  - ✓ DDR(Data Direction Register), PIN (Port Input Pin)
- Each I/O registers have 8 bit wide and each port has a maximum of 8 pins.

How to access I/O registers associated with the ports

DDRx register

- To work the DDRX register an output port set DDRX register 1.
- Eg:
    - ✓ For this the command is

      LDI R16, OXFF

      OUT DDRB, R16

  - ✓ Then the PORT B act as a output register, then we can output data to the AVR I/O pins.
- To make a port an input port, we must put zeros into the DDRX register.
- Eg:

  LDI R16, 0X00

  OUT DDRB, R16

PIN register

- To read the data present at the pins, we should read the PIN register.
- To bring the data into CPU from pins , we read the contents of the PINxregister.
- To send data out to pins, we use PORTx register.

PORT register

PORT A

- It has 8 pins (PA0-PA7)
- To make PORT A an output port, DDRA must be set

```
LDI   R16,0xFF    ;R16 = 11111111 (binary)
OUT   DDRA,R16    ;make Port A an output port
```

- To make PORT A an input port, DDRA must be cleared

```
LDI   R16,0x00    ;R16 = 00000000 (binary)
OUT   DDRA,R16    ;make Port A an input port (0 for In)
```

PORT B

- It has 8 pins (PB0-PB7)
- To make PORT B an output port, DDRB must be set

```
LDI   R16,0xFF    ;R16 = 11111111 (binary)
OUT   DDRB,R16    ;make Port B an output port (1 for Out)
```

- To make PORT B an input port, DDRB must be cleared

```
LDI   R16,0x00    ;R16 = 00000000 (binary)
OUT   DDRB,R16    ;make Port B an input port (0 for In)
```

PORT C

- It occupies a total of 8 pins(PC0-PC7)
- to use the pins of pairs of PORT C as input port, DDRC must be cleared

```
LDI   R16,0x00    ;R16 = 00000000 (binary)
OUT   DDRC,R16    ;make Port C an input port (0 for In)
```

- To use the pins of PORT C as output port DDRC must be set

```
LDI   R16,0xFF    ;R16 = 11111111 (binary)
OUT   DDRC,R16    ;make Port C an output port (1 for Out)
```

PORT D

- It occupies a total of 8 pins PD0-PD7
- To use the pins of PORTD as input or output ports each bits of the DDRD register must be cleared or set respectively.
- to use the pins of pairs of PORT D as input port, DDRD must be cleared

```
LDI   R16,0x00    ;R16 = 00000000 (binary)
OUT   DDRD,R16    ;make Port D an input port (0 for In)
```

- To use the pins of PORT D as output port DDRD must be set

```
LDI   R16,0xFF    ;R16 = 11111111 (binary)
OUT   DDRD,R16    ;make Port D an output port (1 for Out)
```

I/O BIT Manipulation Programming

I/o ports and bit -- addressability

- Some times we need to access only one or two bits of the ports instead of 8 bits.
- For all AVR ports , we can access either all 8 bits or any single bits without altering the rest.

Single bit(Bit Oriented) instructions for AVR

1)SBI 2)CBI 3)SBIS 4)SBIC

SBI (Set Bit in I/O register)

- To set HIGH a single bit of a given I/O register, we use the following syntax.

  SBI ioReg, bit_num

- IoReg – can be lower 32 I/O registers (address 0 to 31) and bit_num is the desired bit number from 0 to 7.
- These instructions can be used for manipulation of bits D0-D7 of the lower 32 I/O registers.

CBI (Clear Bit in I/O register)

- To clear a single bit of a given I/O register.

- Syntax is

  CBI ioReg, bit _ number

  Eg : CBI PORT B,2 ; bit clear PB2 =low.

  PB2 is the 3 rd bit of port B.

SBIS (Skip if Bit in I/O register Set)

- It is used for monitoring the status of a single bit for high.
- This instruction tests the bit and skips the next instruction if it is high.
- Instruction format is

  SBIS a, b

SBIC (Skip if Bit in I/O register Cleared)

- To monitor the status of a single bit for low.
- This instruction tests the bit and skips the instruction right below it if the bit is low.
- Instruction format is

  SBIC a, b

ARITHMETIC INSTRUCTIONS

ADD

- It has 2 general purpose register as input and the result is stored in first register.
- Format is

  ADD Rd, Rr ; Rd=Rd+Rr
- This instruction will change any of the Z,C,N,V,H or S bits of the status register
- It does not support direct memory access.i.e we cannot add a memory location to another memory location or register

ADC(Add with Carry)

- When adding two 16 bit data operands, we use ADC instruction
- Format is

  ADC Rd,Rr ;Rd=Rd+Rr+C
- It will add two registers and the contents of the C flag and places the result in destination register Rd
- It will change the H,S,V,N,Z,C bits of status register
- E.g. add 3CE7 H+ 3B8D h

  Suppose R1=8D, R2=3B, R3=E7, R4=3C

  ADD R3, R1

ADC R4, R2

Subtraction

- In subtraction AVR use 2's complement method
- In AVR we have 5 instructions for subtraction

1) SUB

- Subtracts two registers and places the result in the destination register.
- Format is

  SUB Rd, Rr

  - It will change H, S, V,N,Z, C bits of status register

2) SUBI

- Subtracts a register and a constant and places the result in destination register
- Format is
  - SUBI Rd,K ;Rd=Rd-K
- It will change H, S, V,N,Z, C bits of status register

3) SBC (Subtract with borrow)

- Subtracts two register and subtracts with the C flag and places the result in destination register
- Format is
  - SBC Rd,Rr ;Rd=Rd-Rr-C
- It will change H, S, V,N,Z, C bits of status register

4) SBCI

- Subtracts a constant from a register and subtracts with the C flag and places the result in destination register
- Format is
  - SBCI Rd,K ;Rd=Rd-K-C
- It will change H, S, V,N,Z, C bits of status register

5) SBIW

- Subtracts an immediate value from a register pair and places the result in register pair
- Format is
  - SBIW Rd+1: Rd, K ;Rd+1:Rd=Rd+1:Rd-K
- It will change S, V,N,Z, C bits of status register

Multiplication

- There are 3 multiplication instructions

1) MUL

- It is a byte by byte multiply instruction
- The operands must be a register
- After multiplication, the 16 bit unsigned product is placed in R1(high byte) and R0(low byte)
- Format is

  MUL Rd, Rr ;R1:R0=Rd*Rr

- It will change Z, C bits of status register

2) MULS

- It will multiply the signed no. in Rd with the signed no. in Rr and the result is placed in R1 and r0
- Format is

  MULS Rd, Rr ;R1:R0=Rd*Rr
- It will change Z, C bits of status register

3) MULSU

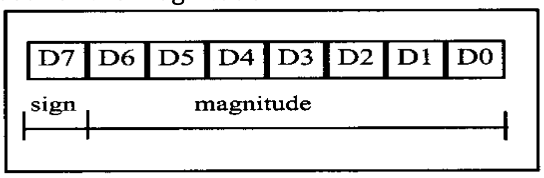
- It performs 8 bit* 8 bit -> 16 bit multiplication of a signed and an unsigned number.
- The multiplicand Rd is a signed no. and the multiplier Rr is an unsigned no. result is placed in R1 and R0
- Format is

  MULS Rd, Rr ;R1:R0=Rd*Rr
- It will change Z, C bits of status register

- AVR has no instruction for division operation.but we can write a program to perform division by repeated subtraction.

Signed number concepts

- The numbers could be positive or negative.
- For representing the signed no.s , the MSB is set aside for sign(+ or -)and the rest of bits are used for the magnitude



. **Figure 8-Bit Signed Operand**

- Using 8 bit no.,the range of +ve no is in the range of 0 to +127.(D7=0)
- For negative no.s,D7=1.the magnitude is represented in its 2's complement form .
- If the result of an operation on signed number is too large for the register, an overflow has occurred.
- Overflow is possible only when we ADD two operands with the same sign.

Logic instructions

1. AND

- Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register RD.
- Format is AND Rd, Rr
- It will change the S,V,N,Z bits of status register
- AND instruction is used to mask(set to 0) certain bits of an operand.
- for e.g.

```
LDI     R20,0x35      ;R20 = 35H
ANDI    R20,0x0F      ;R20 = R20 AND 0FH (now R20 = 05)
```

```
        35H    0011 0101
AND     0FH    0000 1111
        ---------------
        05H    0000 0101          ;35H AND 0FH = 05H, Z = 0,
```

- Here 3 is masking.

2. **OR**

- Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register RD.
- Format is

  OR Rd, Rr
- It will change the S,V,N,Z bits of status register
- OR instruction is used to set certain bits of an operand to one.
- for e.g.

```
        04H       0000 0100
OR      30H       0011 0000
        --------------------
        34H       0011 0100
```

3. **COM**

- Performs 1's complement of register Rd
- Format is COM Rd
- It will change the S,V<-0,N,Z<-1,C bits of status register

4. **EOR or EX-OR**

- Performs the logical Exclusive OR between the contents of register Rd and register Rr and places the result in the destination register Rd.
- Format is

  EOR Rd, Rr
- It will change the S,V,N,Z bits of status register

Show the results of the following:

```
LDI   R20, 0x54
LDI   R21, 0x78
EOR   R20, R21
```

Solution:

```
        54H   0101 0100
XOR     78H   0111 1000
        ----------------
        2CH   0010 1100          54H XOR 78H = 2CH, Z = 0, N = 0
```

5. **NEG**

- Replace the contents of register Rd with its two's complement
- Format is   NEG Rd
- It will change the S,V,N,Z,C,H bits of status register

Compare instruction

CP

- Performs compare between the contents of register Rd and register Rr.
- None of the registers are changed some conditional branches can be used after CP instruction.
- Format is

  CP Rd, Rr
- It will change the S,V,N,Z,C,H bits of status register
- Conditional instructions that are used with CP,

AVR Compare Instructions

BREQ	Branch if equal	Branch if Z = 1
BRNE	Branch if not equal	Branch if Z = 0
BRSH	Branch if same or higher	Branch if C = 0
BRLO	Branch if lower	Branch if C = 1
BRLT	Branch if less than (signed)	Branch if S = 1
BRGE	Branch if greater than or equal (signed)	Branch if S = 0
BRVS	Branch if Overflow flag set	Branch if V = 1
BRVC	Branch if Overflow flag clear	Branch if V = 0

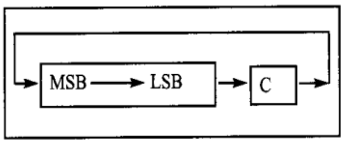
Rotate instructions

- There are 2 rotate instructions. They involve carry flag.

1. ROR

- 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.
- Format is

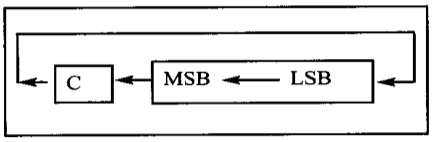
  ROR Rd
- It will change the S,V,N,Z,C bits of status register



2. ROL

- Shifts all bits in Rd one place to the left. the c flag is shifted into bit 0 of Rd. bit 7 is shifted into the C flag
- Format is

  ROL Rd
- It will change the S,V,N,Z,C,H bits of status register



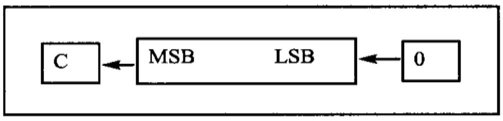
Shift instructions

1. LSL

- Shifts all bits in Rd one place to the left. Bit 0 is cleared.bit 7 is loaded into the c flag .
- Format is

LSL Rd

- It will change the S,V,N,Z,C,H bits of status register

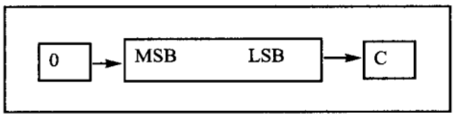


2. LSR

- Shifts all bits in Rd one place to the right. Bit 7 is cleared. bit 0 is loaded into the c flag .
- Format is

LSR Rd

- It will change the S,V,N,Z,C bits of status register

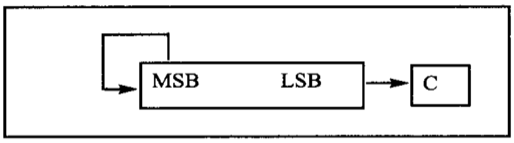


3. ASR

- means arithmetic shift right
- Shifts all bits in Rd one place to the right. Bit 7 is held constant. bit 0 is loaded into the c flag .
- Format is

ASR Rd

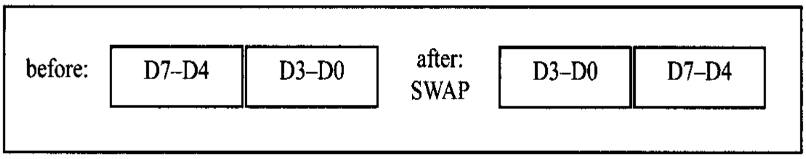
- It will change the S,V,N,Z,C bits of status register

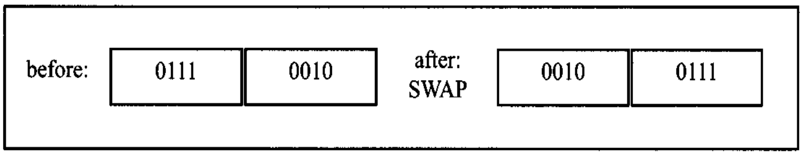


SWAP Instructions

- It swaps the lower nibble and the higher nibble.
- It works on R0-R31
- Format is

SWAP Rd





BCD and ASCII conversion

BCD

- It stands for Binary coded decimal
- Binary representation of 0 to 9 is called BCD
- There are 2 terms for BCD numbers
- Unpacked BCD and Packed BCD

Unpacked BCD

- ✓ The lower 4 bits of the no. represent the BCD number and the rest of the bits are 0
- ✓ For e.g. "0000 1001" and "0000 0101" are unpacked BCD for 9 and 5 respectively

Packed BCD

- ✓ A single byte has two BCD numbers in it. One in the lower 4 bits and one in the upper 4 bits.
- ✓ For e.g. "0101 1001" is packed BCD for 59H.

ASCII

- It is a 7 bit code

Packed BCD to ASCII conversion

- To convert packed BCD to ASCII,first convert it to unpacked BCD
- Then the unpacked BCD is added with 011 0000(30H)

Packed BCD	***Unpacked BCD***	***ASCII***
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 &	0011 0010 &
	0000 1001	0011 1001

ASCII to Packed BCD conversion

- To convert ASCII to packed BCD ,first convert it to unpacked BCD
- Then combine it to make packed BCD

Key	***ASCII***	***Unpacked BCD***	***Packed BCD***
4	34	00000100	
7	37	00000111	01000111 which is 47H

AVR Programming in C

- Compliers produce hex files that we downloaded into flash of microcontroller.
- The size of microcontroller is one of the main concerns of microcontroller programmers because microcontrollers have limited on chip-flash.
- Major reasons for writing programs in C:
  - ✓ It is easier and less time consuming to write in C than in Assembly.
  - ✓ C easier to modify and update.
  - ✓ You can use code available in function libraries.

- ✓ C code is portable to other microcontrollers with little or no modification.

C data type for the AVR C

- Good understanding of C data type create smaller hex file.

Some Data types widely used by C compilers are :

Data type	Size in Bits	Data Range/Usage
Unsigned char	8-bit	0 to 255
char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	-32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	-2,147,483,648 to +2,147,483,648
float	32-bit	+-1.175e-38 to +-3.402e38
double	32-bit	+-1.175e-38 to +-3.402e38

Unsigned char

- The unsigned char is an 8-bit data type the value range of 00-FFH(0-255).
- Most widely used data type.
- avoid the use of int if possible, because AVR microcontroller have limited registers and data RAM locations.

Signed char

- Is an 8-bit data type that use most significant bit to represent - or + value.
- We have only 7 bits for the magnitude of the signed number values from -128 to +127.

Unsigned int

- Is 16-bit data type, value in the range of 0 to 65,535(0000-FFFFH).
- Used to represent 16-bit variables like memory addresses.
- Used to set counter values of more than 256.
- It takes 2 bytes of RAM.
- The misuse of int variables will result in larger hex file, slower execution of program.

Signed int

- Is a 16-bit data type that use most significant bit to represent + or - value.
- 15-bits for magnitude, range of -32,768 to +32,767.

Other data types

- AVR C compiler supports long data types, if want values greater than 16-bit.

Time Delay

- There are 3 ways to create a time delay in AVR C.
    1. Using a simple for loop

2. Using predefined C functions
3. Using AVR timers

Using a simple for loop

- In creating time delay using for loop, we must be mindful of two factors that can affect the accuracy of delay.
    1. The crystal frequency connected XTAL1-XTAL2 input pins is the most important factor in time delay calculation.
    2. Affects the time delay is the complier used to compile the C program.

Using predefined C functions

- Use predefined functions such as delay_ms() and delay_us() defined in delay.h in WinAVR.
- Drawback is probability problem, because different compliers do not use same name for delay functions.
- Overcome by using wrapper or macro function.
- Wrapper call the predefined delay function.

Using AVR timers

- One way to generate a time delay is to clear the counter at the start time and wait until the counter reaches a certain number.
- The content of the counter register represents how much time has elapsed.

I/O Programming in C

- All port registers of the AVR are both byte accessible and bit accessible.

Byte Programming

- To access a PORT register as a byte, we use PORTx label, where x indicate the name of register.
- Access the DDRx register , x indicate data direction of port.
- Access the PINx register, x indicate name of port.

Bit Programming

- The I/O ports of ATmega32 are bit-accessible.
- We can access a single bit of I/O port registers.
- For eg: PORTB.0=1; Here we set the 0th bit of PORTB is equal to 1.
- Also we can use the AND and OR bit wise operations to access a single bit of a given register.

Logic Operations in C

- One of the most powerful feature of C language is its ability to perform bit manipulation.
- The logic operators are
    - ✓ AND(&&)
    - ✓ OR(||)
    - ✓ not(!)

- C also supports bit-wise operators. Those are
  - ✓ AND(&)
  - ✓ OR(|)
  - ✓ EX-OR(^)
  - ✓ Inverter(~)
  - ✓ Shift right(>>)
  - ✓ Shift left(<<)
- **eg:** 0x35&0x0f=0x05 /*ANDing*/

Bit-wise logic operators for C

		AND	OR	EX-OR	Inverter
A	**B**	**A&B**	**A|B**	**A^B**	**Y=~B**
0	0	0	0	0	1
0	0	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

Compound assignment operators in C

- To reduce coding.

Compound assignment operator in C

Operation	Abbreviated expression	Equal C expression
And assignment	a &= b	a =a & b
OR assignment	a|=b	a =a | b

Bit-wise shift operators in C

Bit-wise shift operators for C

Operation	Symbol	Format of shift operation
Shift right	>>	data>>number of bits to be shifted right
Shift left	<<	data<<number of bits to be shifted left

- eg: 0b00010000 >> 3 = 0b00000010 /* shifting right 3 times */

Data Conversion Programs in C

- Many compliers have some predefined functions to convert data types.
- To use these functions the stdlib.h file should be included.
- Data type conversion function in C are:
  - ✓ int atoi(char *str) - Converts the string str to integer
  - ✓ long atol(char *str) - Converts the string str to long
  - ✓ void itoa(int n,char *str) - Converts the integer n to characters in string str

- ✓ void ltoa(int n,char *str) - Converts the long n to characters in string str
- ✓ float atof(char *str) - Converts the characters from string str to float

- In newer microcontrollers have a real-time clock(RTC) where the time and date are kept even when power is off.
- Very often RTC provides data and time packed in BCD.
- To display them we must convert them to ASCII.

Checksum byte in ROM

- When you transmit data from one device to another or when you save and restore data to a storage device.
- To ensure data integrity, the checksum process uses checksum bytes.
- Checksum byte is an extra byte that is tagged to the end of a series of bytes of data.

Data Serialization in C

- Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller.
- There are 2 ways to transfer a data serially:
    1. Using a serial port. The programmer has very limited control over the sequence of data transfer.
    2. Transfer data one bit a time and control the sequence of data and spaces between them.
- Devices such as LCD,ADC they take up less space on printed circuits.