***GOVERNMENT POLYTECHNIC COLLEGE PERUMBAVOOR***

***LAB MANUAL OF EMBEDDED SYSTEMS REVISION 2021***

***CODE 5137***

EX NO:3 DATE:

### ADDITION OF DIFFERENT DATA FORMATS

### 

AIM:

### 

Write an AVR assembly language program to perform the Addition of numbers of various data formats.

OBJECTIVE

After completing this experiment the student will be able to understand various data formats of AVR microcontrollers. THEORY AVR microcontroller supports various data formats like binary, hexadecimal and decimal.

* Binary - A binary number is a combination of 0’s and 1’s. E.g:0B00101101
* Decimal - A decimal number system consists of numbers between 0-9. E.g: 10
* HexaDecimal - A hexadecimal number system consisting of numbers 0-9 and Alphabets A-F. E.g: 22H, 0x22
* ASCII - ASCII means American standard code for information interchange Is a character encoding standard for electronics communication. ASCII (American Standard Code for Information Interchange) is the most common format for text files in computers and on the Internet. In an ASCII file, each alphabetic, numeric, or special character is represented with a 7- bit binary number (a string of seven 0s or 1s). 128 possible characters are defined. E.g: ’32’

### SOURCE CODE

### Write Source code here

### SAMPLE OUTPUT:

INPUT:

OUTPUT

EX NO: 4 DATE :

## ARITHMETIC OPERATIONS ON NUMBERS

AIM

Write an AVR assembly language program to perform various arithmetic operations.

OBJECTIVE

After completing this experiment the student will be able to understand various arithmetic operations in AVR microcontrollers, and a generalized concept of arithmetic operators in AVR.

THEORY

AVR microcontroller supports various arithmetic operations like addition, subtraction, multiplication etc.some of the important arithmetic operators are given below:

Arithmetic operators : +, -, \* /, %

### SOURCE CODE

**Write Source code here**

### SAMPLE OUTPUT

INPUT:

OUTPUT:

ADDITION RESULT

SUBTRACTION RESULT

MULTIPLICATION RESULT

QUOTIENT:

REMAINDER:

EX NO: 5 DATE:

AIM

## SHIFT INSTRUCTIONS IN AVR C

Write an AVR C program to count the number of 1’s and 0’s in a given binary

number.

OBJECTIVE

After completion of this experiment the student is able to understand the usage of various shift instructions.

THEORY

Shift and Rotate instructions shift the bits in the destination operand by one or more positions either right or left.

Various Shift instructions

* + Shift Left
* Instruction Format: Data << Number of bits to be shifted left.

•This instruction shifts the data to a left by a specified number of times

* Eg: 0B00000010 << 3 = 0B00010000
  + Shift Right
* Instruction format: Data >> Number of bits to be shifted right.
* This instruction shifts the data to right by a specified number of times.
* E.g : 0B00010000 >> 3 = 0B00000010

### SOURCE CODE

**Write Source code here**

**SAMPLE OUTPUT** INPUT:

OUTPUT:

EX NO: 6 DATE:

# ASCII TO PACKED BCD

**AIM :**

Write an AVR C program to convert the given ASCII number into Packet

BCD.

OBJECTIVE

After completion of this experiment the student will be able to understand the concept simple code conversion

THEORY

To convert ASCII to packed BCD, you first convert it to unpacked BCD, and then combine it to make packed BCD. For example, for 4 and 7 the keyboard gives 34 and 37, respectively. The goal is to produce 4711 or "0100 0111", which is packed BCD. This process is illustrated next.

| Key | ASCII | Unpacked BCD | Packed BCD |
| --- | --- | --- | --- |
| 4 | 34 | 00000100 |  |
| 7 | 37 | 00000111 | 0100 0111 which is 47H |

### SOURCE CODE

**Write Source code here**

### SAMPLE OUTPUT

INPUT:

OUTPUT:

EX NO: 7 DATE:

**AIM**

# PACKED BCD TO ASCII

Write an AVR C program to convert the given Packed BCD number into ASCII.

OBJECTIVE

After completion of this experiment the student will be able to understand the concept simple code conversion

THEORY

In many systems we have what is called a real-time clock (RTC). The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off (see Chapter 16). This data, however, is provided in packed BCD. For this data to be displayed on a device such as an LCD, or to be printed by the printer, it must be in ASCII format. To convert packed BCD to ASCII, you must first convert it to unpacked BCD. Then the unpacked BCD is tagged with 011 0000 (30H).

The following demonstrates converting packed BCD to ASCII.

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

### SOURCE CODE

**Write Source code here**

### SAMPLE OUTPUT

INPUT:

OUTPUT:

EX NO: 8 DATE:

**AIMI/O PORT PROGRMMING AVR C**

Write an AVR C program to get data from PINB and send it to the I/O register of PORTC continuously.

OBJECTIVE :

After completion of this experiment the student should be able to understand the operation of ATMEGA PORT as output. The operations of PORT are visualized by interfacing LEDs to PORT pins. In addition, this experiment will provide preliminary experience on programming microcontrollers in Embedded C.

THEORY

ATMEGA ports are 8 bit wide. Each port has 3 eight bit registers associated. Each bit in these registers configures pins of the associated port. Bit 0 of these registers is associated with Pin 0 of the port; Bit1 of these registers is associated with Pin1 and so on.

These three registers are

* DDRx register, PORTx register, PINx register X may be replaced by A,B,C or D based on the PORT you are using.
* DDRx register DDRx (Data Direction Register) configures data direction of the port pins. Writing 0 to a bit in DDRx makes the corresponding port pin as input, while writing 1 to a bit in DDRx makes the corresponding port pin as output.

Example: • to make all pins of port B as input, DDRA = 0b00000000;

* to make all pins of port A as output pins : DDRB= 0b11111111;
* to make lower nibble of port B as output and higher nibble as input : DDRB = 0b00001111; In hexadecimal representation, it can be written as DDRB = 0x0F;

### SOURCE CODE

**Write Source code here**

### SAMPLE OUTPUT

INPUT:

OUTPUT:

EX NO: 9 DATE:

**BIT MANIPULATION-I**

A switch is connected to pin PB2. Write a program to check the status of the switch and perform followings:

1. If Switch=0, send the letter ‘N’ to PORTD.
2. If Switch=1, send the letter ‘Y’ to PORTD.

### OBJECTIVE

After completing this experiment the student is able to understand the various bit manipulation operators in AVR microcontrollers.

### THEORY

AVR microcontroller has functionality for single bit manipulation. The status of a bit position in a variable ‘y’ can be checked by setting the bit position to 1 in binary 0b00000000 and check using the statement if (y & 0b00000001). if it is true 1 is set in the bit position otherwise 0 in bit position.

### SOURCE CODE

**Write Source code here**

**SAMPLE OUTPUT** INPUT:

OUTPUT:

EX NO:10 DATE:

## BIT MANIPULATION -II

**AIM**

Aim : To blink LED connected to PB1 AT 1sec interval while keeping LED on PB7 on using AND, OR operations and with generalized approach..

### OBJECTIVE

After completing this experiment the student is able to understand the various bit manipulation operators in AVR microcontrollers.

### THEORY

AVR microcontroller has functionality for single bit manipulation. Previous concept of PORT setting by moving values to the entire port cannot be used in

practical cases as moving can change values of pins which are nor desired to be affected Hence usually AND and OR logic operations are used to manipulate bits.

In order to set PB7 in this case PORTB |= 0B10000000; // setting PB7

In order to clear PB1 PORTB &= 0B11111101; // clear PB1

This can be rewritten as, in order to set PB7 in this case

PORTB |= (PORTB <<7); // setting PB7

Or as PORTB |= (PORTB <<PB7); // setting PB7

PORTB &= ~(PORTB <<1); // clear PB1

Or as PORTB &= ~(PORTB <<PB1); // clear PB1

SOURCE CODE

**Write Source code here**

### 

### SAMPLE OUTPUT

INPUT:

OUTPUT:

EX NO: 11 DATE:

**TIME DELAY IN AVR C**

Write an AVR C program to toggle bits 1 and 3 of PORTB in every 200ms.

OBJECTIVE

After completion of this experiment the student should be able to understand the operation of ATMEGA PORT as output. The operations of PORT are visualized by interfacing LEDs to PORT pins. In addition, this experiment will provide preliminary experience on programming microcontrollers on Embedded C.

THEORY

There are three ways to create a time delay in AVR C :

* Using a simple for loop
  + Factors affecting the accuracy of delay:
    - The Crystal Frequency connected to the XTAL1- XTAL2 input pins.
    - Compiler used to compile the C program
* Using pre defined C functions
  + \_delay\_ms()
  + \_delay\_us()
* Using AVR Timers.

### SOURCE CODE

**Write Source code here**

### 

### SAMPLE OUTPUT

INPUT:

OUTPUT:

EX NO: 12 DATE:

## DATA SERIALIZATION IN C

AIM: Write an AVR C program to send out the value 44H serially one bit at a time via PORTC, The LSB should go out first.

OBJECTIVE

After completing this experiment the student will be able to understand the data serialization in AVR microcontroller, and a generalized concept of various bitwise operators in AVR C.

THEORY

Serializing data is a way of sending a byte of data one bit at a time through a single pin of the microcontroller. There are two ways to transfer a byte of data serially:

1. Using the Serial Port.
2. The second method of serializing data is to transfer data one bit at a time and control the sequence of data and spaces between them.
3. The data serialization can be done in either bit wise or byte wise
   * We can send the data bit-wise to a particular pin and also we can store a byte of data in continuous bits of an i/o port.
     + The AVR C supports various bitwise operators such as,

* Shift Right It shifts the bits to right by a specified number of times. Format: data

>> number of bits to be shifted right. E.g: 0b00010000 >>3 =0b00000010

* Shift left It shifts the bits to left by a specified number of times. Format: data << number of bits to be shifted left. E.g: 0b00010000 <<3 = 0b10000000

### SOURCE CODE

**Write Source code here**

### SAMPLE OUTPUT

INPUT:

OUTPUT:

EX NO: 13 DATE:

**TIMER0/COUNTER0 PROGRAMMING**

AIM : Write an AVR C program to toggle all bits of PORTB alternatively, with some delay. Use TIMER0 normal mode and pre scalar option to generate delay.

OBJECTIVE

After completing this experiment the student will be able to understand the Usage of timers in AVR microcontrollers.

THEORY

A Timer is a simple counter. The input clock of the microcontroller and operation of the timer is independent of the program execution. The Timers are mainly classified into three:

* TIMER 0

It is an 8-bit timer.

The register where the counting takes place is the TCNTn register, where n is become 0, 1, 2 etc. It counts automatically and overflows and restarts again.

CS02, CS00 bits in the TCCR0 are used to choose the clock source. The various modes of clock source are:

| D2 | D1 | D0 |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | - Timer/counter Stops. |
| 0 | 0 | 1 | - No Pre scalar |
| 0 | 1 | 0 | - Clk / 8 |
| 0 | 1 | 1 | - Clk / 64 |
| 1 | 0 | 0 | - Clk /256 |
| 1 | 0 | 1 | - Clk /1024 |
| 1 | 1 | 0 | - External source to falling edge. |
| 1 | 1 | 1 | - External Source to raising edge. |

WGM is used for Timer mode selection. Various modes are:

| D6 | D3 |  |
| --- | --- | --- |
| 0 | 0 | - Normal mode |
| 0 | 1 | - CTC (Clear Timer on Compare Match) |
| 1 | 0 | - PWM, phase correct |
| 1 | 1 | - Fast PWM |

TIFR(Timer/counter flag register) – contains flags of different timers. Important flags are:

1. TOV0 D0 –

Timer0 overflow flag bit to 0 – Timer0 did not overflow. Timer0 overflow flag bit to 1 – Timer0 has overflowed.

1. OCF0 D1 –

Timer0 output compare flag to 0 – Compare match did not occur. Timer0 output compare flag to 1 – Compare match occurred.

* TIMER 1
* It is a 16-bit timer.

The control register is split into 2 8-bit registers TCCR1A and TCCR1B.

* TIMER 2

It is an 8-bit timer.

It cannot be used as a counter, because it does not support external clock.

It can be used as a real time clock.

### SOURCE CODE

**Write Source code here**

### 

### SAMPLE OUTPUT