Chapter - 7

Why Program the AVR in C?

→ Compiler produces hex file.

→ Hex file is downloaded into the ROM of AVR.

→ Size of the hex file produced by compiler is one of the main concerns of microcontroller programmer.

① All μc (micro controller) limited on chip ROM.

② The code space for ATMEGA32 is limited to 32K bytes.

Microcontroller and Embedded Processors →

Microcontroller :

What they are: Tiny computer chips that do specific tasks.

Inside Them : A small brain (processor), a memory to remember things, and tools to connect with other parts (peripherals)

Used for : Controlling things like your TV remote, washing machine, or car engine

Embedded Processors :

What they are: Like a general computer brain that can do many things.

Inside them : A flexible brain (processor) that

can handle different tayks, but not always with built-in tools like microcontrollers.

Used For: Running the show in devices, like your phone, computer or smart thermostat.

Key Differences:

Scope of Application

Microcontrollers are specialized for specific application.

Embedded processors are more general-purpose and can be used in a vaity of application.

Integration:

Microcontrollers integrate peripherals for specific applications.

Embedded processors may or may not include integrated peripherals and may rely on external components for specific function.

Complexity and Power:

Microcontrollers are often simpler and more power-efficient.

Embedded processors can be more complex and powerful, suitable for diverse computational tasks.

Microcontroller verses general purpose micro processor

Microcontrollers:

Job: They are like specialists with a specific job.

Tools: Come with built-in tools for that one job.

Specific Code: Programmed with specific code for its particular tasks.

All-in-one: They have a processor (the brain), memory (for storing info.), and tools to connect with other parts all in one.

small package.

Examples: Think of them like a chef with all necessary ingredients and tools to make one specific dish.

General Purpose Microprocessors:

Versatile: They are like all-around helpers.

Tools: Don't have specific tools; they use external tools as needed.

Used Everywhere: Found in computers, smartphones and devices that need to do many different things.

Key Differences:

Microcontrollers are like specialists tailored for specific tasks, with everything they need built-in.

Or,

Microcontrollers: Specialized tools for one specific task.

General Purpose Microprocessors: Versatile, but they might need additional tools for specific tasks.

Jobath

1(a)

What is the difference between RET and IRET instruction?

In x86 assembly language, 'RET' and 'IRET' are instructions used for managing the flow of control in a program, particularly in the context of function calls and interrupts.

1. RET (Return)

Purpose: Used to return control from a ~~subroutin~~ or function to the calling code.

Usage: 'RET' is often used at the end of a function to transfer control back to the calling function.

Stack: Before 'RET' is executed, it typically retrives the return address from the stack and jumps to that address.

assembly

```
; Example of RET
  CALL  MyFunction  ; Call a function
; ...
MyFunction:
  ; Function code here
  RET               ; Return from the function
```

2. IRET (Interrupt Return)

Purpose: Used to return from an interrupt service routine (ISR) and restore the processor state after handling an interrupt.

Usage: IRET is used at the end of an interrupt handler to return from the interrupt.

Stack: Similar to 'RET', 'IRET' retrives the return addresses and other information from the stack.

```
; Example of IRET
ISR:
    ; Interrupt service routine code here
    IRET ; Return from interrupt and restore processor state.
```

Key Difference :

RET : Used for normal function returns, where the return address is typically the only information retrive from the stack.

IRET : Used specifically for returning from interrupt service routines, where the processor state (including flags) is restored from the stack in addition to the return address.

In Summary, 'RET' is for returning from regular functions, while 'IRET' is for returning from interrupt service routines and involves additional state restoration.

Chapter - 2

Components of embadded systems.

Sol^n: An embedded system is like a smart and dedicated computer designed for a specific job. Here's a simple breakdown of its main parts:

1. Brain (Microcontroller): Thinks and follows instructions.

2. Memory: Stores both the instructions and temporary data.

3. Sensors: Gathers information from the environment (like touch or temperature)

4. Outputs: Does something based on the

gathered information (like turning on a light or a motor)

5. Communication: Talks to other devices, like sending or receiving data.

6. Clock: keeps track of time for time-dependent

7. Power Source: Provides energy for everything to work.

All these parts work together to make the embedded system perform its special task efficiently

SRAM Vs. EEPROM

SRAM (Static Random Access Memory):

Volatility: ~~Volatility~~ Volatile (loses data when power is off.

Speed: Fast read and write operation.

Size: Medium sized (smaller than flash, larger than EEPROM)

Usage: Temporary storage during program execution.

EEPROM (Electrically Erasable Programmable Read only memory).

Volatility: Non-volatile (retains data even when power is off)

Speed: Slower compared to SRAM.

Size: Medium-sized (smaller than SRAM, larger than Flash)

Usage: Persistent storage for data that survives power cycles.

Key difference between RISC and CISC

Aspect	RISC (Reduce Instruction set Comput	CISC (Complex Instruction Set Comput
Instructions	fewer, simpler instructions	Many, some complex instruction.
speed	Generally faster execution per instruction.	Some instruction may take longer to execute.
Complexity	Emphasizes simplicity	Emphasizes versatility.
Registers	Uses more registers, simple instructions.	May use fewer registers, complex instruction.
Memory Usage	Typically requires more memory.	Can be more memory-efficient depending on the task.
Examples	ARM, MIPS, PowerPC	x86 (Intel, AMD)

What is the purpose of the pseudo instruction;

The purpose of the pseudo- instructions in assembly language programming is to provide a convenient and human-readable way for programmers to write code while allowing the assembler or compiler to perform specific tasks during the assembly process. Pseudo-instructions are not actual machine instructions executed by the CPU; instead, they searve the following pupposes.

1. Simplifying Programming
2. Memory Allocation
3. Constant Definition
4. Assembler Directives
5. Code Organization
6. Abstraction of Low-Level Operations.
7. Portability.

What do RISC and CISC stand for?

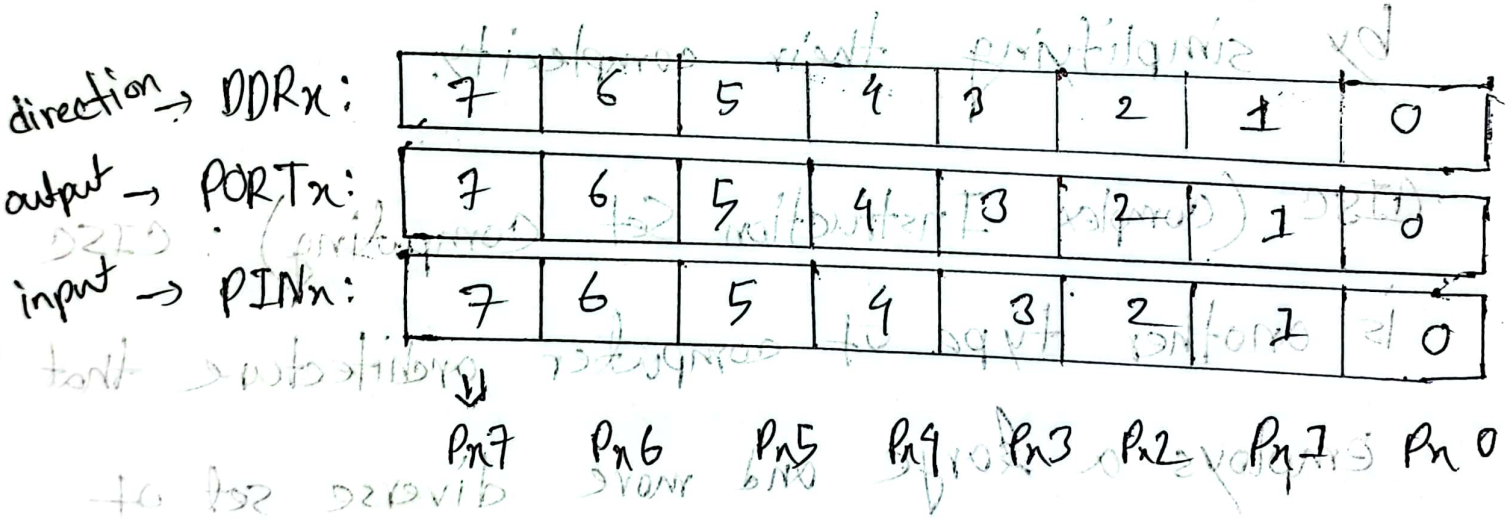
RISC (Reduced Instruction Set computing): RISC is a type of computer architecture that uses a small set of a simple and highly optimized instructions. Each instruction in RISC typically performs a single, basic operation. The goal is to execute instructions quickly by simplifying their complexity.

CISC (Complex Instruction Set Computing): CISC is another type of computer architecture that employs a large and more diverse set of instructions, including complex instructions that can perform multiple operations. The idea is to provide a ~~varity~~ variety of instructions to handle various tasks, potentially reducing the number of instructions a programmer needs to writ

How many instructions does the ATmega have?

Soln: The ATmega microcontrollers have a simple and efficient instruction set, typically around 131 instructions.

Chapter – 4



Port A and Port B how does it work?

Port A:

Poart A occupies a total 8 pins (PA0 - PA7).

To use the pins of Port A as input or output ports, each bit of the DDRA register must be set of proper value.

Port A as input:

In order to make all the bits of Port A an input, DDRA must be cleared by writing 0 to all the bits.

How does Port A and Port B works in AVR I/O port programming?

In AVR microcontrollers, the I/O ports are labeled as Port A, Port B, Port C etc. Each port consists of a set of pins, and each pin is can be individually configured as an input or

an output. Here is a general overview of how port A and port B work in AVR I/O port programming:

PortA and PortB Overview:

1. Register Configuration:

Each port has 3 associated registers: PORTx, PINx, and DDRx

→ PORTx: (Port output register) - used to set the output state of the port pins.

→ PINx: (Port input register) - Used to read the ~~output~~ input state of the port pins.

→ DDRx: (Data Direction register) - Used to configure pin as an input and an output

2. Setting pins as Input or Outputs:

→ To set a pin as an ~~input~~ output, you write a '1' to the corresponding bit in 'DDRx' register.

→ To set a pin as an input, you write a '0' to the corresponding bit in the 'DDRx' register.

3. Reading and Writing to Pins:

To read the state of the input pin, you read the corresponding bit from 'PINx' register.

To write a value to an output pin, you write the desired value to the corresponding bit in the 'PORTx' register.

Example Code for PortB:

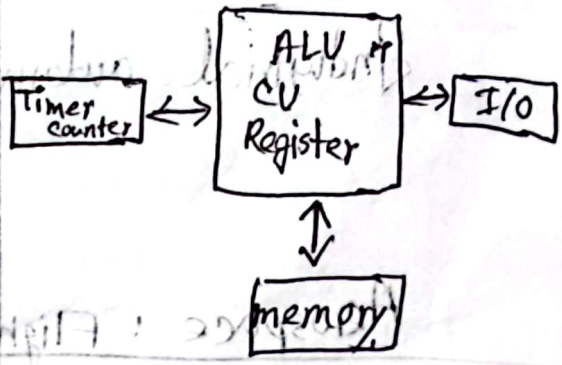
```
#include <avr/io.h>
#define LED_PORT PORTB
#define LED_PORT PINB
#define LED_PORT DDRB
#define LED_PORT PB0

int main(void) {
    // set PB0 as an output
    LED_DDR |= (1 << LED_BIT);

    while(1) {
        // Toggle the state of PB0
        LED_PORT ^= (1<< LED_BIT);
    }
    return 0;
}
```

Difference between Microprocessor & Microcontroller.

Feature	Microprocessor	Microcontroller.
Application	General-Purpose	Specific, dedicated tasks
Computational Power	High	Moderate
Integration	Requires external components.	Integrated components.
Power Consumption	High	Lower
Cost	Generally Higher	Generally Lower.
Examples	Intel Core i7, AMD Ryzen	Atmel AVR, PIC, Arduino



In summary, while a microprocessor is suitable for tasks requiring high computational power and flexibility, a microcontroller is designed for

What is an Embedded System?

An embedded system is a specialized computer system designed to perform a specific task. It is typically composed of a microprocessor, memory, and input/output (I/O) peripherals. Embedded systems are found in a wide variety of devices, including:

Medical devices: pacemakers, insulin pumps,

Consumer electronics: smartphones, tablets, TV, camera

Automotive systems: engin control modules, ant-lock breake systems, airbags

Industrial automation: robotics, sensors, programmable logic controllers (PLCs).

Aerospace: Flight control systems, navigation systems, communication systems.

Embedded systems are often design to operate in real time, meaning that they must respond to events within a very short period of time.

The key characteristics of embedded systems

1) Dedicated function 2) Real time operation
3) Low-power consumption 4) Low cost 5) small-size.

General-Purpose Computing System vs. Embedded System.

Characteristic	General-Purpose Computing System	Embeded System
functionality	Wide range of tasks and applications.	Specific, dedicated functions
flexibility	Highly flexible and adaptable.	Limited flexible, specialized.
Components	Processor, memory, storage, I/O.	Microcontroller, memory interfaces.
Operating System	Contains a general purpose operating system (GPOS)	May or may not contain an operating system for functioning.

Examples	Personal computers, laptops, servers.	Microcontrollers in appliances, automotive control system
Real-time operation	Generally not optimized for real-time operation	Often optimized for real-time operation (depend on the purpose)
Update and modification	Can be easily updated and modified.	Updates may involve firmware or software updates.

Major Applications areas embedded systems.

Embedded systems find applications in a wide range of industries and everyday devices, contributing to their efficiency, functionality, and intelligence. Here are some major applicati areas of embedded systems:

1. Consumer Electronics: Camera, smart-phones, tablets, smart TVs,

2. Household Appliances Applications: Television, DVD players, washing machine, fridge, etc

3. Home automation and security systems: Air conditioners, fire alarm, intruder detection alarm.

4. Automated Industry: Anti-lock breaking system (ABS), engin control, automatic nerigation system

5. Telecom: Cellular telephoney, telephone switches, handset multimedia applications, etc.

6. Computer Peripherals: Printers, scanners, fax machine etc.

7. Computer networking systems: Network raders, switches, hubs, etc.

8. Healthcare: Different kinds of scanners, EEG, ECG machines etc.

9. Measurements & Instrumentation: Digital multimeters, digital CROs. etc.

10. Banking & Retail: Automatic Teller Machine (ATM) and currency counters

Purpose of EMBEDDED SYSTEMS

Each embedded system is designed to serve the purpose of any one or combination of following tasks

1. Data collection / Storage / Representation
2. Data communication
3. Data processing
4. Monitoring
5. Control
6. Application specific user interfa

7. Real time operation

8. Low power consumption

9. Automation 10. Security

Chapter-2

Core of the Embedded system

Embedded systems are domain and application specific and are built around a central core. The core of the embedded system are falls into any one of the following categories:

1. General Purpose and Domain specific Processors.
   - 1.1. Microprocessors
   - 1.2. Microcontrollers
   - 1.3. Digital signal Processors
2. Application Specific Integrated Circuits (ASICs)

3. Programmable Logic Devices (PLDs)

4. Commercial off-the-shelf components (COTS)

If you examine any embedded system you will find that it is built around any of the core units mentioned above.

~~hi~~

What is Sensors?

A sensor is transducer device that converts energy from one from to another for any measurement or control purpose

R/Q

1. Explain the components of a typical embedded system.

A typical embedded system ~~is~~ consist of several key components that works together

⊕ to perform specific functions within a large system. Embedded systems are specialized computing systems designed to perform ~~specific~~ dedicated functions or tasks and are often embedded as part of a large device. Here are typical components of an embedded system:

1. Microprocessor / Microcontroller
2. Memory
   - → RAM
   - → ROM
   - → Flash memory
3. Input devices.
   - → sensors
   - → interfaces (receive Input from external devices)
4. Output devices
   - → Actuators
   - → Interfaces (send output to external devices)

5. Communication Interfaces
6. Real-time clock (RTC)
7. Power supply
8. Operating system

Chapter - 3

Characteristics of a embedded systems

Embedded systems are specilized computer systems designed to perform a specific tasks. They are typically found in devices that require real-time operation and low power conyumption.

Here are some key characteristics of embedded systems:

1. Real-time operation
2. Low-power consumption
3. Reliability

4. Cost - effectiveness
5. Small size- and weight.
6. Distributed
7. Power concerny.
8. Application & domain specific
9. Operates in harsh environment

Operational quality Attributes.

The operational quality attribute represents the relevent quality attribute, related to embedded system when it is in an operational mode or 'online' mode. The importants attributes coming under the catagory are listed below:

1. Response
2. Throughput
3. Reliability
4. Maintainability
5. Security
6. Safety

PPT

MPU (Microprocessor) Vs. MCU (Microcontroller)

MPUs and MCUs are both types of integrated circuits (ICs) that are used to control electronic devices. However, there are some key differences between two →

Feature	MPU (Microprocessor)	MCU (Microcontroller)
Purpose	General-purpose	Specialized
Application	personal computers, servers, smartphones, tablets	consumer electronics, industrial automation, medical devices.
Power consumption	Higher	Lower
Cost	Higher	Lower
Complexity	More-complex	Less complex
Performance	Higher	Lower

	GPU	
Flexibility	More flexible	Less flexible

Why use Microcontroller

→ simpler construction and control

→ Lower cost

⇒ Low total area are occupied on the circuit board.

→ ~~Change~~ Flexibility - Changes can be made by simple changing the software. (code used to program the microcontroller)

→ Software implementation is usually easier then its hardware counterpart.

Different type of AVR Microcontroller

Tiny AVR – Less memory, small size, suitable only for simpler applications.

Marge AVR – These are the most popular ones having good amount of memory (upto 256k higher number of inbuilt peripherals and suitable for moderate to complex applicati

XmegaAVR – Used commercially for compl applications, which require large program memory and high speed.

I/O Programming –

I/O port has 3 registers associated with each port. These three registers are –

→ DDRx (data direction register)

→ PINx

→ PORTx

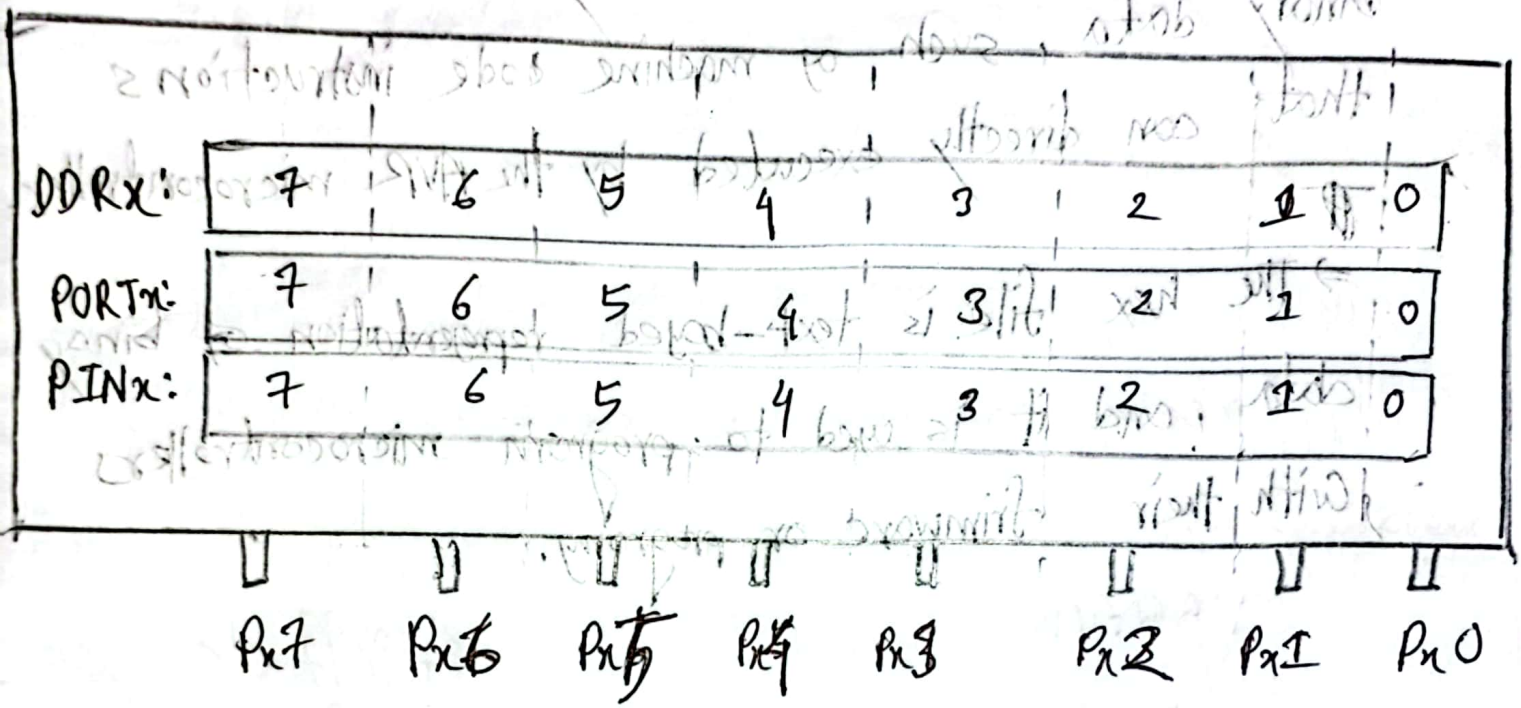


Fig: Relations Between the Registers and the Pins of AVR.

PORTx / DDRx	0	1
0	Input & high impedance	Out 0
1	Input & Pull-up	Out 1

Different States of a Pin in the AVR Microcontroller.

* Explaining the hex file for AVR

In the context of AVR (Alf and Vegard's RISK processor) microcontrollers, a hex file is a type of file format commonly used to store binary data, such as machine code instructions that can directly executed by the AVR microcontroller.

⇒ The hex file is text-based representation of binary data, and it is used to program microcontrollers with their firmware or programs.

: BRAAAA
[colon] [Data size] [start address] [record type]
[Data checksum]

Chapter 7

Why Program the AVR in C?

The following are some reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than in Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

Time Delay:

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

1. Using a simple for loop.
2. Using predefined C functions
3. Using AVR timers

Chapter 7

Why Program the AVR in C?

The following are some reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than in Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

Time Delay:

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

1. Using a simple for loop.
2. Using predefined C functions
3. Using AVR timers

Ex: Write an AVR C program to send values 00-FF to Port B.

Soln:

```
#include <avr/io.h>  //standard AVR header.
int main(void)
{
  unsigned char z;
  DDRB = 0xFF;          // PORTB is output
  for (z=0; z<=255; z++)
    PORT = z;
  return 0;
}
```

Ex: Write an AVR C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C and D to port B.

Soln:

```
#include <avr/io.h>
int main(void)
{
    unsigned char mylist[]="012345ABCD";
    unsigned char z;
    DDRB = 0xFF;
    for(z=0; z<10; z++) {
        PORTB = mylist[z];
    }
    while(1);
    return 0;
}
```

Write an AVR C program to toggle all the bits of Port B 200 times.

```
#include <avr/io.h>
int main(void)
{
    DDRB = 0xFF;
    PORTB = 0xAA;
```

```
unsigned char z;
for ( z=0 ; z<200; z++)
{
PORTB = ~PORTB;
}
while (1);
return 0;
}
```

Write an AVR C program to send 0 to -4 to PORTB

```
#include <avr/io.h>
int main (void)
{
char mynum[] = {-4,-3,-2,-1,0,1,2,3,4};
unsigned char z;
DDRB = 0xFF;
for (z=0 ; z<9 ; z++)
{
PORT = mynum[z]
while(1);
}
```

write an AVR C program to toggle all bits of port B 50,000 times.

soln:

```
#include <avr/io.h>
int main(void)
{
  unsigned int z;
  DDRB = 0xFF;
  for (z=0; z<50000 ; z++)
  {
    PORT = 0x55;
    PORT = 0xAA;
  }
  while (1);
  return 0;
}
```

Write an AVR C program to toggle all bits of PORTB 100,000 times.

```
#include <avr/io.h>
int main(void)
{
  unsigned char z;
  DDRB = 0xFF;
```

```
    for(z=0 ; z<100,000 ; z++){
        PORTB = 0x55;
        PORTB = 0xAA;
    }
    while(1);
    return 0;
}
```

Ex: Write an AVR C program to toggle all the bits of Port B continuously with a 100ms delay. Assume that the system is ATmega 32 with XTAL = 8MHz.

Soln:

```
#include <avr/io.h>
int main(void)
{
    void delay100ms(void)
    {
        unsigned int i;
        for(i=0; i<42150; i++);
    }
```

```
int main(void)
{
    DDRB = 0xFF;
    while (1)
    {
      PORTB = 0xAA;
      delay100ms();
      PORTB = 0x55;
      delay100ms();
    }
    return 0;
}
```

Good

=> Write an AVR program to toggle all the pins of Port C continuously 10ms delay. Use a predefined delay function in Win AVR.

soln:

```
#include<util/delay.h>      //delay loop function
#include<avr/io.h>
```

```
int main(void)
{
    void delay_ms(int d)
    {
        _delay_ms(d);
    }
  DDRB = 0xFF;

  while(1) {
      PORTB = 0xFF;
      delay_ms(10);
      PORTB = 0x55;
      delay_ms(10);
  }
   return 0;
}
```

Time Delay

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

1. Using a simple for loop.
2. Using predefined C functions
3. Using AVR timers.

Give two factors that can affect the delay

size →

The term "delay size" can ~~reference~~ to different things depending on the context. Two factors that can affect delay size are:

1. Propagation Delay:

This is the time it takes for a signal to travel from the source to the destination. The speed of ~~time~~ light in the transmission medium plays a ~~cir~~ crucial role in determining the propagation delay

2. Processing Delay: This includes the time taken by devices or systems to process and handled the signal. For example, in a network, routers and introduce some processing delay as they examine and forward packet.

7-10) Write an AVR C program to get a byte of dat from Port B, and then send it to Port C.

Soln:

```
#include <avr/io.h>
int main (void)
{
  unsigned char temp;
   DDRB = 0x00;
   DDRC = 0xFF;
while(1)
{  temp = PINB;
   PINC = temp;
}
return 0;
}
```

7-11 (write an AVR C program to get a byte of data from Port C. If it is less than 100, send it to Port B; otherwise send it to Port D.

Soln:

```
#include <avr/io.h>
int main (void)
{
   DDRC = 0x00;
   DDRB = 0x FF;
   DDRD = 0xFF;
   unsigned char temp;
   while (1)
   {
     temp = PINB ;
     if (temp < 100)
       PORTB = temp ;
     else
       PORTD = temp;
   }
   return 0;
}
```

Ex-7-13 Write an AVR C program to toggle only bit 4 of port B continuously without distributing the rest of the pins of Port B.

```
#include <avr/io.h>
int main (void)
{
    DDRB = 0xFF;          // PORTB is output
    while (1)
    {
        PORTB = PORTB | 0b00010000;
        PORTB = PORTB & 0b11101111;
    }
return 0;
}
```

Ex: 7-14 Write an AVR C program to monitor bit 5 of port C. If it is HIGH, send 55H to PORTB; otherwise, send AAH to port B.

soln:

```
#include <avr/io.h>   // standard AVR header
int main (void)
{
    DDRB = 0xFF;
    DDRC = 0x00;
    DDRD = 0xFF;
    if (PINC & 0b00100000)
    {
        PORTB = 0x55;
    }
    else
        PORTB = 0xAA;
}
return 0;
}
```

Ex-7.5 A door sensor is connected to bit 1 of PortB, and an LED is connected to bit 7 of Port C. Write an AVR C Program to monitor the door sensor and when it opens, turn on the LED.

```
#include <avr/io.h>
int main (void)
{
    DDRB = DDRB & 0b11111101;
    DDRC = DDRC | 0b00000000;
    while (1)
    { if (PINB & 0b00000010)
        PORTC = PORTC | 0b00000000;
      else
        PORTC = PORTC & 0b11111111;
    }
    return 0;
}
```

Ex: The data pins of an LCD are connected to Port B. The information is latched into the LCD whenever its Enable pin goes from HIGH to Low

Soln:

```
#include <avr/io.h>
int main (void)
{
    unsigned char message[] = "The Earth is but
                One Cuntry";
    Unsigned char z;
    DDRB = 0xfff;
    DDRC = DDRC | 0b00100000;
    for (z=0; z<28; z++)
    {
        PORTB = message [z];
        PORTC = PORTC | 0b00100000;
        PORTC = PORTC & 0b11011111;
    }
    while (1);
    return 0;
}
```

Shifting:

1. 0b000010000 >> 3 = 0b000000010
2. 0b000010000 << 3 = 0b010000000
3. 1 << 3 = 0b000001000

Chapter-12

LCD operation (l

In recent years the LCD is finding widespread use replacing LEDs (seven segment LEDs or other multisegment LEDs). This is due to the following reasons:

1. The declining price of LCDs
2. The ability to display number, characters and graphics. This is incontrast to LEDs, which are limited to number and few characters.
3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU by the task of refreshing the LCD. In contrast, the LED

must be refreshed by the CPU to keep displaying the data.

4. Ease of programming for characters & graphics.

Pin	Symbol	I/O	Description
1	Vss	--	Ground
2	Vcc	+	+5V power supply
3	VEE	--	Power supply to control contrast
4	RS	I	RS=0 to select command register RS=1 to select data register.
5	R/W	I	R/W=0 for write, R/W=1 for read.
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data by
8	DB1	I/O	The 8-bit data by
9	DB2	I/O	The 8-bit data by.

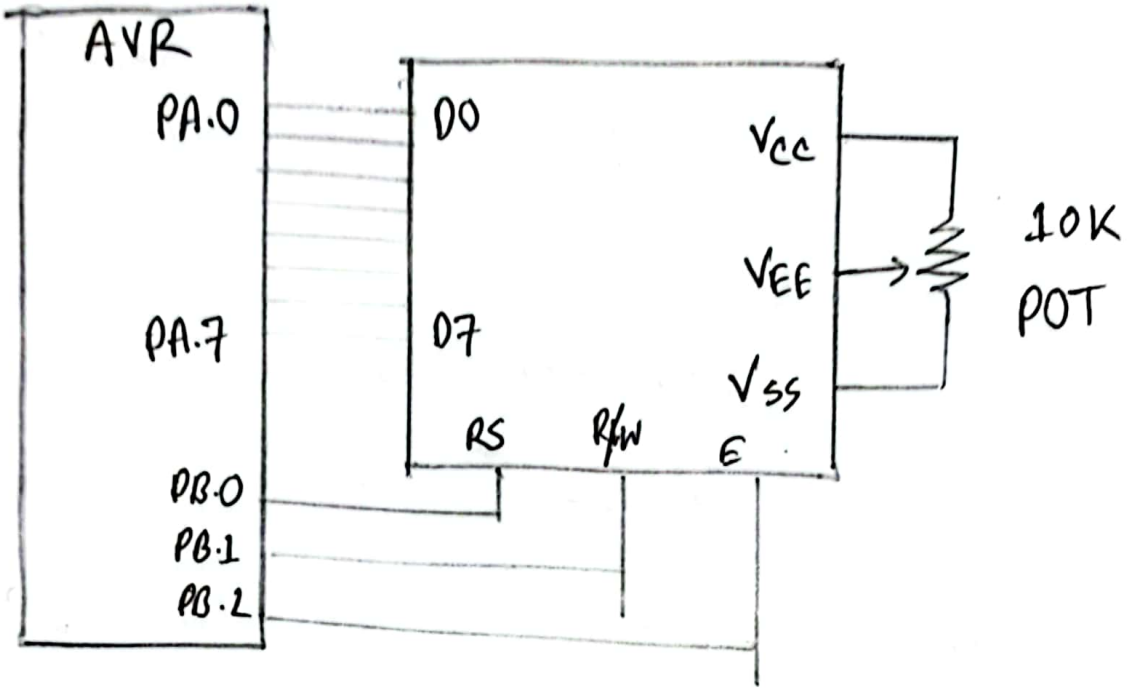
10	DB3	I/O	The 8-bit data by
11	DB4	I/O	The 8-bit data by
12	DB5	I/O	The 8-bit data by
13	DB6	I/O	The 8-bit data by
14	DB7	I/O	The 8-bit data by.

Sending commands and data to LCDs

To send data and commands to LCDs you should do the following steps. Step 2 and 3 can be repeated many times.

1. Initialize the LCD.
2. Send any commands from Table 12-2 to the LCD.
3. Send the character to be shown on the LCD.

LCD connection for 8-bit data.



LCD interfacing.

The following shows how to set DD RAM address locations.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

LCD Type	Line	Address Range				
16X2 LCD	Line1:	80	81	82	83	through 8F
	Line2:	C0	C1	C2	CB	through CF
20X4 LCD	Line1:	80	81	82	83	through 93
	Line2:	C0	C1	C2	C3	through D3
	Line3:	94	95	96	97	through A7
		D4	D5	D6	D7	through E7
40X2 LCD	Line1:	80	81	82	83	through A7
	Line2:	C0	C1	C2	C3	through E7