SCORING INDICATORS

COURSE NAME:EMBEDDED SYSTEMS
COURSE CODE: 6201B QID: 2102240185

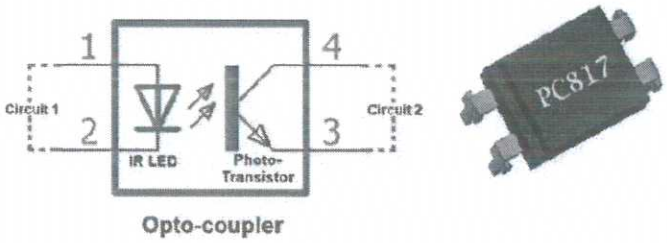
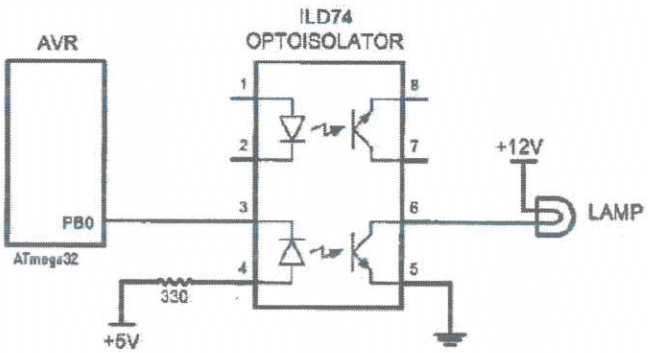
Q No	Scoring Indicators	Split score	Sub Total	Total score
	PART A		1	**9**
I.1	I2C, SPI, UART, 1-Wire etc and Parallel bus interface		1	
I.2	Temperature sensors, proximity sensors, Accelerometer sensors etc.		1	
I.3	ATMega32 consist of **two 8 bit(Timer0 & Timer2)**		1	
I.4	Classic AVR,Mega AVR,Tiny AVR,Special Purpose AVR (any two)		1	
I.5	Inter-integrated-circuit		1	
I.6	10- bit		1	
I.7	FreeRTOS,Zephyr,MQX,Keil RTX,µC/OS,LynxOS,Integrity,emboss,ThreadX,Neutrino. (any two)		1	
I.8	Multiprocessing		1	
I.9	A kernel is a central component of an operating system that acts as an interface between the user applications and the hardware.		1	
	PART B			**24**
II.1	• **Light emitted Diode (LED):** It is an output device & is used as an indicator for the status of various signals or situations. LED is a p-n junction diode and it contains an anode & a cathode. For proper functioning of the LED, the anode of it should be connected to +ve terminal of the supply voltage and cathode to the –ve terminal of supply voltage. A resistor is used in series between the power supply and the resistor to limit the current through the LED. • **7-Segment LED Display:** It is an output device for displaying alpha numeric characters & contains 8 LED segments arranged in a special form. Out of the 8 LED segments, 7 are used for displaying alpha numeric characters. LED segments are named A to G and the		3	

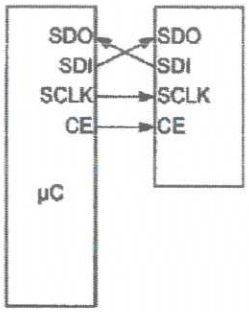
	decimal point LED segment is named as DP. The current flow through each of the LED segments should be limited to the maximum value supported by the LED display unit by connecting a current limiting resistor. • **Stepper Motor**: It is an electro mechanical device which generates discrete displacement (motion) in response to DC electrical signals. It differs from the normal DC motor in its operation. The DC motor produces continuous rotation on applying DC voltage whereas a stepper motor produces discrete rotation in response to the dc voltage applied to it. Stepper motors are widely used in industrial embedded applications, consumer electronic products and robotics control systems. • **Relay**: It is an electro mechanical device which acts as dynamic path selectors for signals and power. The 'Relay' unit contains a relay coil made up of insulated wire on a metal core and an armature with one or more contacts. It works on electromagnet principle. When a voltage is applied to the relay coil, current flows through the coil, which in turn generates a magnetic field. The magnetic field attracts the armature core and moves the contact point. The movement of the contact point changes the power/signal flow path. The Relay is normally controlled using a relay driver circuit connected to the port pin of the processor/controller. (any two)			
II.2	**Based on Complexity & Performance**: The embedded systems are classified into three types based on the complexity & performance of the systems. • Small Scale Embedded Systems: Small Scale Embedded Systems are built with a single 8 or 16-bit microprocessor or controller. The main programming tools used are an editor, assembler, cross assembler and integrated development environment (IDE). The hardware and software complexities in small-scale embedded system are very low. It may or may not contain an operating system for its functioning. An electronic toy is an example for a small- scale embedded system. • Medium Scale Embedded Systems: The Embedded system with		3	

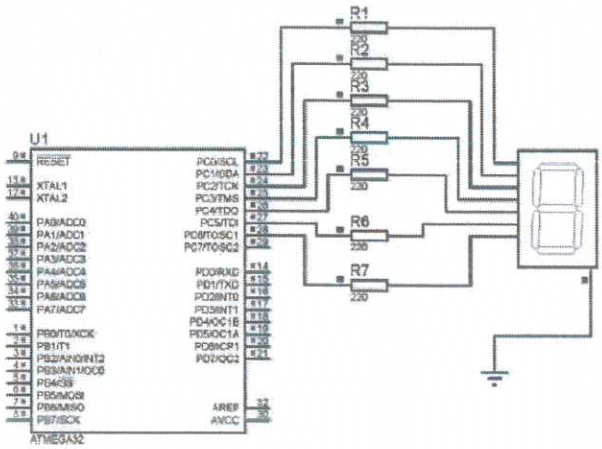
	medium performance 16- bit or 32-bit microprocessor or controller, ASICs or DSPs fall under the medium scale embedded systems. They have both hardware and software complexities. The main programming tools used are C, C++, JAVA, Visual C++, RTOS, debugger, source code engineering tool, simulator and IDE. • Large scale Embedded Systems: The embedded systems have highly complex hardware and software, built around 32-bit or 64-bit processors/controllers, RISC processors, SoC, scalable and configurable processors. They are also called sophisticated embedded systems.			
II.3	**Applications of Embedded Systems**: Today, embedded systems play a critical role in various sectors.Here are a few examples that make use of embedded technology. • Telecommunications systems: Switches, Routers, Bridges • Consumer electronics: MP3 players, Digital cameras, Mobile phones, Digital watches • Household appliances : Washing machines, Television, Set top Boxes ,Microwave ovens, Dish washer, refrigerators • Home automation & Security: Air condition, CCTV Systems, Intruder alarms, Fire alarms • Healthcare: Different Kinds of Scanners(MRI, CT, PET), EEG & ECG machines • Automobiles: Anti-Lock Braking System (ABS), Traction Control (TCS) and Electronic Stability Control (ESP) • Banking & retails: Automated Teller Machine(ATM), Currency counter • Measurements & Instrumentation: Digital Multimeter, Digital Storage Oscilloscope (DSO), Programmable Logic Controller(PLC) systems. • Computer peripherals: Printer, Scanner, Fax machine • Card Readers: Barcode, Smart Card Readers, Hand held Devices etc.	Any six	3	
II.4	**Data Types:** In C programming, data types are declarations for variables. This determines the type and size of data associated with variables.		3	

	<table><tr><th>Data Type</th><th>Size in Bits</th><th>Data Range/Usage</th></tr><tr><td>unsigned char</td><td>8-bit</td><td>0 to 255</td></tr><tr><td>char</td><td>8-bit</td><td>–128 to +127</td></tr><tr><td>unsigned int</td><td>16-bit</td><td>0 to 65,535</td></tr><tr><td>int</td><td>16-bit</td><td>–32,768 to +32,767</td></tr><tr><td>unsigned long</td><td>32-bit</td><td>0 to 4,294,967,295</td></tr><tr><td>long</td><td>32-bit</td><td>–2,147,483,648 to +2,147,483,648</td></tr><tr><td>float</td><td>32-bit</td><td>±1.175e-38 to ±3.402e38</td></tr><tr><td>double</td><td>32-bit</td><td>±1.175e-38 to ±3.402e38</td></tr></table> **1) Unsigned char**: It is an **8-bit** data type that takes the value in the range of **0-255(00-FFH)**. It is one of the most widely used data types for AVR like setting counter values. The C-compiler by default use the signed char unless the keyword '**unsigned**' is specified. **2) Signed char:** It is an **8-bit** data type that uses the Most Significant Bit(**MSB**) i.e **D7** bit to represent **+ve** or **-ve** value. As a result only 7 bits are available for use, giving the rang -128 to +127. **3) Unsigned int:** It is an **16-bit** data type that takes the value in the range of **0-65,535(0000-FFFFH)**. It is used to define 16-bit variable such as memory address, set counter values more than 256. The C-compiler by default use the signed int unless the keyword '**unsigned**' is specified. Since it is a 2-byte data type, **int** is used only in situations that demands, because the use of int data type will result in the creation of larger hex files, slower execution and more memory usage. **4) Signed int:** It is an **16-bit** data type that uses the Most Significant Bit(**MSB**) i.e **D15** bit to represent **+ve** or **-ve** value. As a result only 15 bits are available for use, giving the rang **-32,768** to **+32,767**. **5) Other data types:** • If we want values greater than **16-bit** use **long** data types. • If we want deal with fractional numbers use **float** & **double** data types.			
II.5	**I/O ports:** AT Mega32 has four ports(Port A,Port B, port C and Port D) having 32 pins. **Oscillator:** AT Mega32 has an internal oscillator for its clock. By default it is set to operate an internal caliberated oscillator of 1 MHz. **Timer/counter:** It is used to count an event or to generate time		3	

	delays between two operations. ATMega32 consist of two 8 bit(Timer0 & Timer2) and one 16 bit timer/counter(Timer1). **Watchdog timer**: A watchdog timer is a simple countdown timer which is used to reset a microprocessor after a specific interval of time. System reset is required for preventing failure of the system in a situation of a hardware fault or program error. **Interrupt unit:** AVR ATmega32 consist of 21 interrupt sources out of which three are external. The remaining are internal interrupts which supports the peripherals like USART, ADC, timer etc. The three external hardware interrupts are on pins PD2, PD3, and PB2 which are referred to as INT0, INT1, and INT2 respectively. **Memory:** AT Mega32 consist of three different memory sections. • **Program ROM:** It is used to store program or codes. AT Mega32 has 32KB as Program ROM • **EEPROM:** It is also a non volatile memory used to store data. AT Mega32 has 1 KB of EEPROM. • **Data RAM:** It is a volatile memory used to store data. AT Mega32 has 2 KB of Data RAM. **Other Pheripherals:** It includes: • ADC interface: AT Mega32 has an 8 channel ADC with resolution of 10 bits for Analog to Digital signal conversions. • USART(universal synchronous asynchronous receiver transmitter): USART interface is available for interfacing with external device capable of communicating serially. • SPI(Serial Peripheral Interface): It is used for serial communication between two devices on a common clock source. The data transmission rate of SPI is more than that of USART. • TWI(Two Wire Interface): Can be used to attach low speed peripherals to a motherboard, embedded systems, cell phones or other electronic devices. It is a multi master serial single ended computer bus.			
II.6	**OptoIsolator or Optocoupler:** • It is a semiconductor device that allows an electrical signal to be transmitted between two isolated circuits. • Two parts are used in an optocoupler: **a LED that emits infrared light & a photosensitive device(Photodiode, Phototransistor)**		3	

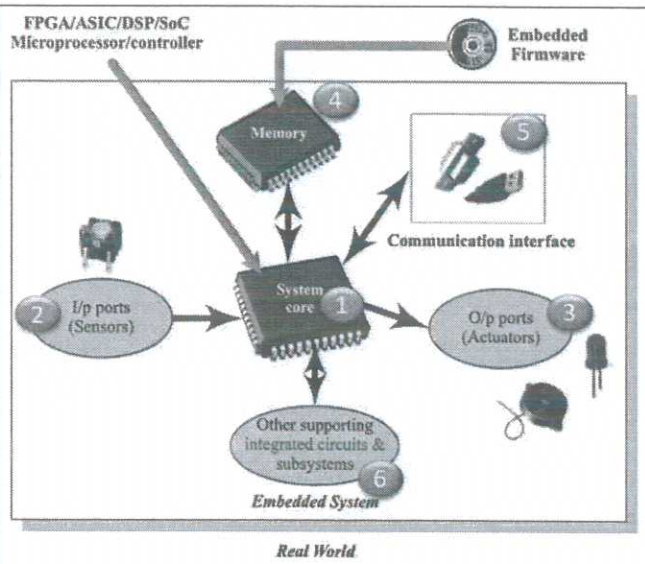
	that detects light from the LED. Both parts are contained within a black box with pins for connectivity. The input circuit takes the incoming signal, whether the signal is AC or DC, and uses the signal to turn on the LED.  Opto-coupler • The port pin to which the optoisolator input control voltage is connected has to be configured as an output port. • The microcontroller will send a logic high signal(5V) to the optoisolator to drive a load(like bulb, motor).  			
II.7	**Serial Peripheral Interface (SPI):** • It is a **bus interface connection protocol** originally started by **Motorola Corp(Freescale).** • SPI is a **synchronous serial interface** in which data in an 8-bits can be shifted in and/or out one bit at a time. • It can be used to communicate with a serial peripheral device(flash, EEPROM, sensors) or with another microcontroller with an SPI interface • **It uses four pins for communication.** ▪ **SDI** (Serial Data Input) ▪ **SDO** (Serial Data Output) ▪ **SCLK** (Serial Clock) ▪ **CE**(Chip Enable)/**CS** (Chip Select)		3	

	• It has two pins for **data transfer** called **SDI (Serial Data Input)** and **SDO (Serial Data Output)**. **SCLK (Serial Clock)** pin is used to synchronize data transfer and Master provides this clock. **CS (Chip Select)** pin is used by the master to select the slave device.  SPI peripherals in AVR can operate in **Master or Slave modes**. If **master mode** is selected, then it takes care of generating the clock signal and starting the transfer. The slave only waits for the CS line to pull down to get ready for transfer. • ATmega32 has an inbuilt SPI module. It can act as a master and slave SPI device. SPI communication pins in AVR ATmega are: ▪ MISO(Master In Slave Out): The Master receives data and the slave transmits data through this pin. ▪ MOSI(Master Out Slave In): The Master transmits data and the slave receives data through this pin. ▪ SCK(Synchronization Clock): The Master generates this clock for the communication, which is used by the slave. The only master can initiate a serial clock. ▪ SS(Slave Select): Master can select slaves through this pin.			

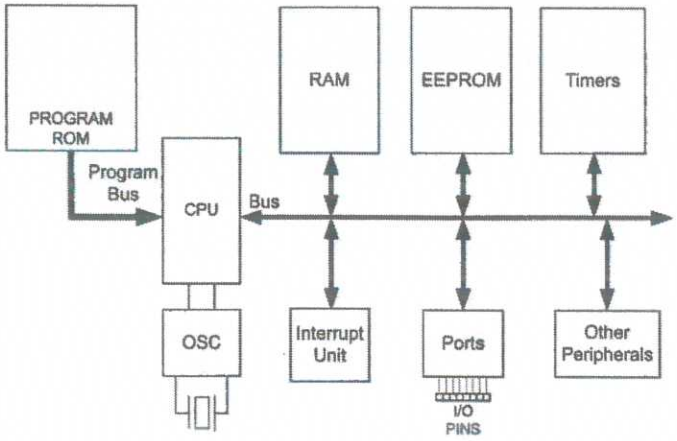
II.8			3	
II.9	**Task Communication:** The mechanism through which processes/task communicate each other is known as Inter Process/Task Communication(IPC). It is essential for process coordination. Various IPC mechanisms adopted by Kernel • **Shared Memory**: Processes share some area of memory to communicate among them. Information to be communicated by the process is written to this shared memory area. Other process which require this information can read the same from the shared memory area. Implementation of shared memory can be done through Pipes & Memory mapped Objects. • **Message Passing:** Its a synchronous information exchange mechanism used for the Inter Process/thread communication. Major difference from shared memory is that through shared memory, lots of data can be shared whereas only limited amount of info/data is passed through message passing. Message passing is relatively fast and free from synchronisation overheads compared to shared memory. Message passing methods includes Message Queue, Mail box, Signalling **Remote Procedure Call(RPC) and sockets**: Inter Process Communication mechanism used by a process to call a procedure of another process running on the same CPU or on a different CPU which is interconnected in a network. Used for distributed applications like client-server applications. With RPC it is possible to communicate over a heterogeneous network(Network where		3	

	client and server running on different OS). The CPU/process containing the procedure which needs to be invoked remotely is known as server. The CPU/process which initiates an RPC request is known as client.			
II.10	**Types of task scheduling:** Based on scheduling algorithm, the scheduling can be classified as: **1) Non preemptive scheduling**: • Employed in systems which implement non-preemptive multitasking. The currently executing task/process is allowed to run until it terminates or enters the 'Wait' state waiting for an I/O or system source. **Various types of non-preemptive scheduling**: • **First come First Served(FCFS)/FIFO scheduling:** Allocates CPU time to processes based on the order in which they enter the 'Ready' queue. First entered process is served first. **Last Come First Served(LCFS)/LIFO Scheduling:** Allocates CPU time to the processes based on the order in which they are entered in the 'Ready' queue. Last entered process is served first. **Shortest Job First(SJF)scheduling:** Sorts the 'Ready' queue each time to pick the process with shortest estimated completion/run time. In SJF, the process with shortest estimated run time is scheduled first, followed by the next shortest process and so on. **Priority based scheduling:** A process with high priority is serviced first in the 'Ready' queue. One way of priority assigning is giving priority to the task/process at the time of creation of the task/process. SJF(shortest job first) algorithm can be viewed as a priority scheduling where each task is prioritised in the order of the time required to complete the task. **2) Preemptive Scheduling:** •Employed in systems which implement preemptive multitasking. •In this scheduling every task in the 'Ready' queue gets a chance to execute. •When and how often each process gets a chance to execute depend on the type of preemptive scheduling algorithm called 'Scheduler' used for scheduling processes.		3	

	•Scheduler can preempt(stop temporarily) the currently executing task and select another task from the 'Ready' queue for execution. Types of Preemptive Scheduling: • **Preemptive SJF Scheduling/Shortest Remaining Time (SRT):** The Preemptive SJF scheduling algorithm sorts the 'Ready' queue when a new process enters the 'Ready' queue and checks whether the execution time of the new process is shorter than the remaining of the total estimated time for the currently executing process. If the execution time of the new process is less, currently executing process is pre-empted and the new process is scheduled for execution. The Non-Preemptive SJF scheduling algorithm sorts the 'Ready' queue only after completing the execution of the current process or when the process enters 'Wait' state. **Round Robin (RR) Scheduling**: Means 'Equal chance to all'. Each process in the 'Ready' queue is executed for a predefined time slot. Execution starts with picking up the first process in 'Ready' queue. Its executed for a predefined time and when the predefined time elapses or the process complete(before the predefined time slice), the next process in the 'Ready' queue is selected for execution. This is repeated for all the process in the 'Ready' queue. **Priority based Scheduling:** Priority based Preemptive Scheduling is same as Priority based Non-premptive Scheduling except for the switching of execution between tasks. In Preemptive scheduling, any high priority process entering the 'Ready' queue is immediately scheduled for execution whereas in non preemptive any high priority process entering the 'Ready' queue is scheduled only after the currently executing process completes its execution or only when it voluntarily relinquishes the CPU.			
	PART C			**42**

III	 **Hardware components:** 1) **System Core**: A typical embedded system core contains a single chip controller/ Processors which acts as the master brain of the system. The core of the system performs some predefined operations on input data with the help of embedded firmware and sends some actuating signals to the actuator. 2) **Input port and Sensors**: The input signal provided by the end users or sensors which are connected to the input ports. Keyboards, push button, switches, etc. are examples of input devices 3) **Output port and Actuators**: The processed control signals is send to the actuators or devices connected to the output port. LEDs, LCDs, Piezoelectric buzzers, etc. are examples for output devices. 4) **Memory**: The memory of the system is responsible for holding the proram and data. Some controllers may contain internal memory or on-chip memory, while other requires external memory or off-chip memory. There are two types of memory present: • **Fixed memory:** It is a Read Only Memory(ROM) & is used for storing code or program. The user cannot do any modifications in this type of memory. Common types used are PROM, EPROM, EEPROM & Flash memory. • Temporary memory or Working memory: It is a Random Access Memory(**RAM)** & is used for performing arithmetic operations. Common types used are SRAM, DRAM and NVRAM. 5) **Communication interface**: They are used in embedded systems to establish communication with other sub-systems or embedded		7	

	systems in the external world. There are several communication ports including USB, UART, USB, I2C, SPI, and RS-485. 6) **Other supporting Integrating Circuits(ICs) & Sub-Systems:** It includes timers, clock circuits, power supply, Interrupt controller, etc.. for the proper funtioning of an embedded system.			
IV	**RAM(Random Access Memory) or Temporary memory:** It is the data or working memory of the controller/processor. It retains the content as long as the power is applied to the chip. If the power is turned off then its contents will be lost forever. It is a volatile momory • **SRAM (Static RAM):** It stores data in the form of voltage. They are made up of flip flops. A typical SRAM cell is made of 6 transistors. It is fast in operations. • **DRAM (Dynamic RAM):** It stores data in the form of charges. They are made up of MOS transistor gates. A typical DRAM cell is made of one capacitor and one transistor. It has high memory density compared to SRAM. It is slow in operations. A DRAM controller • **NVRAM(Non - Volatile RAM):** It is usually a SRAM with battery backup. When power is turned on, the NVRAM operates just like any other SRAM but when power is off, the **ROM(Read Only Memory) or Fixed memory:** It is the code or program memory of the controller/processor. It retains the content even the power applied to the chip turns off. It is a non - volatile momory • **Masked ROM:** It is an one time programmable memory. These are hardwired memory devices found on system & are factory programmed. It contains pre-programmed set of instruction and data and it cannot be modified by end user. • **PROM (PROGRAMMABLE ROM):** It is also an one time programmable memory. This memory device comes in an un-programmed state i.e. at the time of purchased it is in an un-programmed state and it allows the user to write his/her own program or code into this. • **EPROM (ERASABLE-AND-PROGRAMABLE ROM):** It is same as PROM and is programmed in same manner as a PROM. It can be erased and reprogrammed repeatedly as the name suggests. • **EEPROM**(Electrically Erasable and Programmable ROM). It is same as EPROM, but the erase operation is performed electrically. Any byte in EEPROM can be erased and rewritten as desired. The erasing process is faster compared to EPROM. • **Flash:** Flash memory is the most recent advancement in memory technology. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write)		7	
V	**Simplified Block Diagram of ATMEGA32 Microcontroller**		7	



I/O ports: AT Mega32 has four ports(Port A,Port B, port C and Port D) having 32 pins.

Oscillator: AT Mega32 has an internal oscillator for its clock. By default it is set to operate an internal caliberated oscillator of 1 MHz.

Timer/counter: It is used to count an event or to generate time delays between two operations. ATMega32 consist of two 8 bit(Timer0 & Timer2) and one 16 bit timer/counter(Timer1).

Watchdog timer: A watchdog timer is a simple countdown timer which is used to reset a microprocessor after a specific interval of time. System reset is required for preventing failure of the system in a situation of a hardware fault or program error.

Interrupt unit: AVR ATmega32 consist of 21 interrupt sources out of which three are external. The remaining are internal interrupts which supports the peripherals like USART, ADC, timer etc. The three external hardware interrupts are on pins PD2, PD3, and PB2 which are referred to as INT0, INT1, and INT2 respectively.

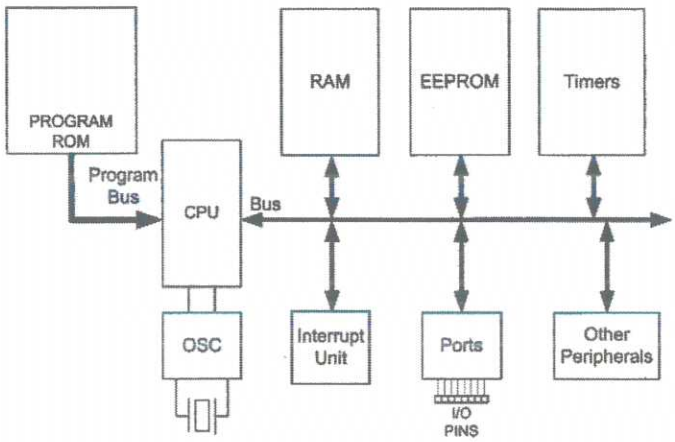
Memory: AT Mega32 consist of three different memory sections.

• Program ROM: It is used to store program or codes. AT Mega32 has 32KB as Program ROM

• EEPROM: It is also a non volatile memory used to store data. AT Mega32 has 1 KB of EEPROM.

• Data RAM: It is a volatile memory used to store data. AT Mega32 has 2 KB of Data RAM.

Other Pheripherals: It includes:

	• **ADC interface:** AT Mega32 has an 8 channel ADC with resolution of 10 bits for Analog to Digital signal conversions. • **USART(universal synchronous asynchronous receiver transmitter):** USART interface is available for interfacing with external device capable of communicating serially. • **SPI(Serial Peripheral Interface):** It is used for serial communication between two devices on a common clock source. The data transmission rate of SPI is more than that of USART. • **TWI(Two Wire Interface):** Can be used to attach low speed peripherals to a motherboard, embedded systems, cell phones or other electronic devices. It is a multi master serial single ended computer bus.			
VI	**Simplified Block Diagram of ATMEGA32 Microcontroller**  **I/O ports:** AT Mega32 has four ports(Port A,Port B, port C and Port D) having 32 pins. **Oscillator:** AT Mega32 has an internal oscillator for its clock. By default it is set to operate an internal caliberated oscillator of 1 MHz. **Timer/counter:** It is used to count an event or to generate time delays between two operations. ATMega32 consist of two 8 bit(Timer0 & Timer2) and one 16 bit timer/counter(Timer1). **Watchdog timer**: A watchdog timer is a simple countdown timer which is used to reset a microprocessor after a specific interval of time. System reset is required for preventing failure of the system in a situation of a hardware fault or program error. **Interrupt unit:** AVR ATmega32 consist of 21 interrupt sources		7	

	out of which three are external. The remaining are internal interrupts which supports the peripherals like USART, ADC, timer etc. The three external hardware interrupts are on pins PD2, PD3, and PB2 which are referred to as INT0, INT1, and INT2 respectively. **Memory**: AT Mega32 consist of three different memory sections. • Program ROM: It is used to store program or codes. AT Mega32 has 32KB as Program ROM • EEPROM: It is also a non volatile memory used to store data. AT Mega32 has 1 KB of EEPROM. • Data RAM: It is a volatile memory used to store data. AT Mega32 has 2 KB of Data RAM. Other Pheripherals: It includes: • **ADC interface:** AT Mega32 has an 8 channel ADC with resolution of 10 bits for Analog to Digital signal conversions. • **USART(universal synchronous asynchronous receiver transmitter):** USART interface is available for interfacing with external device capable of communicating serially. • **SPI(Serial Peripheral Interface):** It is used for serial communication between two devices on a common clock source. The data transmission rate of SPI is more than that of USART. • **TWI(Two Wire Interface):** Can be used to attach low speed peripherals to a motherboard, embedded systems, cell phones or other electronic devices. It is a multi master serial single ended computer bus.			

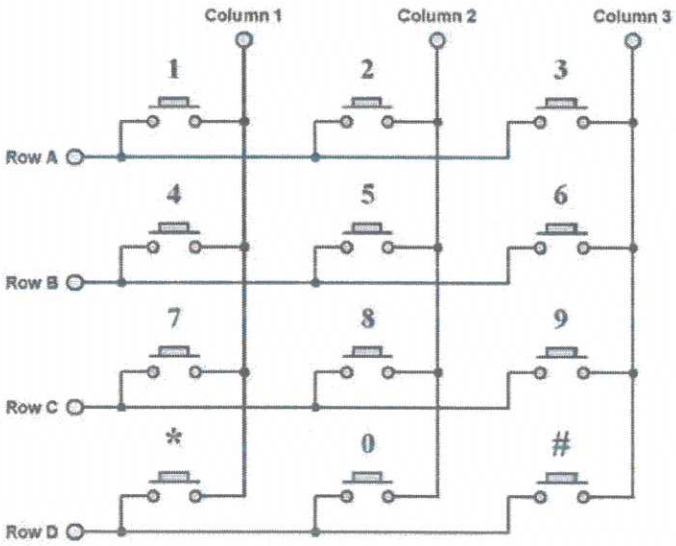
VII	#include <avr/io.h>
	int main(void)
	{
	DDRB = 0xFF; //PORT B is output
	DDRC = 0x00; //PORT C is input
	PORTB = 0x00;
	while(1)
	{
	if (PINC & (1<<2))
	{
	PORTB = 0x00;
	}			

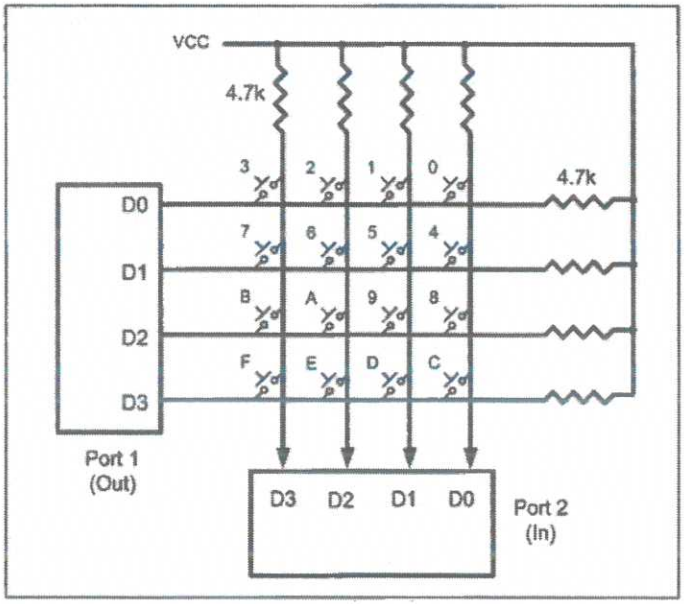
	else
	{
	PORTB = 0xFF;
	}
	}
	return 0;
	}			

VIII	**Write an AVR C program to convert packed BCD 0x29 to ASCII and display the bytes on PORTB and PORTC.**
	
	Solution:
	
	`#include <avr/io.h>` `//standard AVR header`
	`int main(void)`
	`{`
	`unsigned char x, y;`
	`unsigned char mybyte = 0x29;`
	
	`DDRB = DDRC = 0xFF;` `//make Ports B and C output`
	`x = mybyte & 0x0F;` `//mask upper 4 bits`
	`PORTB = x | 0x30;` `//make it ASCII`
	`y = mybyte & 0xF0;` `//mask lower 4 bits`
	`y = y >> 4;` `//shift it to lower 4 bits`
	`PORTC = y | 0x30;` `//make it ASCII`
	
	`return 0;`
	`}`
			7	

IX	#include <avr/io.h>
	#include <util/delay.h>
	int main(void)
	{
	DDRC = 0xFF; // PORT C is output
	while(1)
	{
	PORTC = 0x09;
	_delay_ms(100);
	PORTC = 0x0C;
	_delay_ms(100);
	PORTC = 0x06;
	_delay_ms(100);
			7	

	PORTC = 0x03;
	_delay_ms(100);
	}
	return 0;
	}			

X	**Interfacing of Matrix Keyboard:** • It is one of the most widely used **input device** interfaced with a micro controller. • A **matrix keypad** is an arrangement of push button switches arranged in **rows** and **columns** i.e. in matrix fashion.  • The CPU accesses both rows and columns through ports, therefore it can be connected with a microcontroller. • When a key is pressed, a column and a row make a contact, otherwise, there is no connection between rows and columns. • The **rows** are connected to an **output port pins** and the **columns** are connected to an **input port pins**. • To **detect which key is pressed** from the matrix keyboard, **the row lines are to be made low one by one and read the columns.** **Pull-up resistors** are connected to the column lines to limit the current that flows to the row line on a key press • If no key has been pressed, reading the input port will give **1s** for all columns since they are connected to high (VCC). **If all the rows are grounded and key is pressed, one of the columns.** have 0 since the key is pressed provides the path to ground. It is the function of microcontroller to scan the keyboard continuously to		7	

	detect and identify the key pressed. • If the data read from the columns is D3-D0=1111, no key has been pressed and the process continues until a key pressed is detected by system. • However, if one of the column bits has a zero, this means that a key press has occurred. For example, if D3 - D0 = 1101, it means that a key in the D1 column has been pressed. After a keypress is detected, the microcontroller will pass through the process of identifying the key. • The output can be displayed using LEDs/7-segment display/ LCD display & port to which these output devices are connected are configured as an output port.  			

XI	#include <avr/io.h>
	int main(void)
	{
	DDRA = 0;
	DDRC = 0xFF;
	PORTC &= 0xF8; // initially disable L298, L298 IN1 = 0, L298 IN2 = 0
	while(1)
	{
	PORTC |= (1<<0); //enable L298
	If
	((PINA & 0x01) == 0)
	{
	PORTC &= (~(1<<1)); //L298 IN1 = 0
	PORTC |= (1<<2); //L298 IN2 = 1
	}
			7	

	else
	{
	PORTC |= (1<<1); //L298 IN1 = 1
	PORTC &= (~(1<<2)); //L298 IN2 = 0
	}
	}
	return 0;
	}			

XII	#include <avr/io.h>
	int main(void)
	{
	DDRB = 0xFF; //PORT B is output
	DDRC = 0x00; //PORT C is input
	PORTB = 0x00;
	while(1)
	{
	if (PINC & (1<<2))
	{
	PORTB = 0x00;
	}
	else
	{
	PORTB = 0xFF;
	}
	}
	return 0;
	}
			7	

XIII	**µC/OS-II services:** It includes the following: **1) Task Creation and Management:** MicroC/OS-II is a multi-tasking(manage up to 64 tasks) operating system. Each task is an infinite loop and can be in any one of the following 5 states: • **Dormant:** State where task is created but no resources are allocates. **Ready:** State where task is launched into the memory & awaiting the processor time for execution. **Running:** State at which the process execution happens in CPU. **Waiting/Pending:** State where a running process is temporarily suspended from execution and does not have immediate access to resources.		7	

	Interrupted: State at which an interrupt is occurred & CPU execute ISR. **2) Kernel Functions and Initialization:** The kernel needs to be initialized & started before executing a task. MicroC/OS-II kernel initialization & OS kernel start is a written as a part of the start up code which is executed before the execution of user tasks. **3) Task Scheduling:** MicroC/OS-II support priority based scheduling. Each task has a unique priority ranging from 0 to 63, with 0 being highest priority. Task rescheduling occur whenever a higher priority task becomes 'Ready' to run or when a task enters 'Wait' state or an 'Interrupt' occurs. **4) Inter - task communication**: MicroC/OS-II kernel supports inter - task communication for data sharing & interfacing through 'Mailbox' for handling single message & 'Message Queue' for handling multiple messages. **5) Mutual Exclusion and Task synchronization:** In multi-tasking environment multiple tasks are executed simultaneously & shared the system resources & data. The access to shared resources should be made mutually exclusive to prevent data corruption. Synchronisation techniques are used to synchronise the their execution. **6) Timing & Reference**: The 'Clock Tick' act as the time source for providing timing referencer for time delays & timeouts. It generates periodic interrupts. **7) Memory Management**: MicroC/OS-II uses runtime memory allocation & deallocation on a needed basis. The memory is divided into multiple sectors(partitions). Each sector is further divided into blocks of fixed memory size. Each memory sector associates a 'Memory Control Block' with it.			
XIV	**Selection criteria for RTOS:** The factors include the functional & non-functional requirements for selection an RTOS. **Functional Requirements:** These are requirements that specify the desired functionality or operations of a RTOS. It includes: **Processor Support**: It is essential to ensure that the processor should support the OS under consideration. **Memory Requirement:** Since embedded system is memory constrained, it is essential to have a minimal ROM & RAM		7	

	requirements for an OS under consideration. **Real time capabilities:** It is essential to analyse the real time capabilities of an OS under consideration, as task/process scheduling policies play an important role in real time behaviour.			