QID – 1009

Revision : 15 **Course Code : 6132**

Course Title : Microcontrollers

Qst No		Scoring Indicator	Split Score	Sub Total	Total
PART A					
I	1	• Harvard & RISC architecture • Memory => 32KB (flash), 2KB (Data RAM) , 1KB (EEPROM)	1+1	2	10
	2	• **SBI** ioReg, bit_num • **CBI** ioReg, bit_num • **SBIS** ioReg, bit_num • **SBIC** ioReg, bit_num	0.5 x4	2	
	3	• Counter uses external source as clock and Timer uses oscillator signal as clock.	1+1	2	
	4	• DTE => Data Terminal Equipment i.e terminals & computers that send & receive data. • DCE => Data Communication Equipment eg modems that are responsible for transferring data.	1+1	2	
	5	<table><tr><th>Sl.No</th><th>Serial Communication</th><th>Parallel Communication</th></tr><tr><td>1</td><td>Slower because in 1 clock cycle 1 bit transferred</td><td>Faster because in 1 clock cycle n bits are transferred</td></tr><tr><td>2</td><td>Preferred in long distance communication eg serial communication via RS232</td><td>Preferred in short distance communication eg printer & pc communication</td></tr></table>	1+1	2	
PART B					
II	1	<table><tr><th>Sl.No</th><th>Microcontroller</th><th>Microprocessor</th></tr><tr><td>1</td><td>Single Purpose</td><td>General Purpose</td></tr><tr><td>2</td><td>Contains RAM, ROM, I/O ports etc on chip</td><td>Everything is off chip</td></tr><tr><td>3</td><td>Rapid movement of codes & datas within chip</td><td>Rapid movement of codes & datas from external address to chip</td></tr><tr><td>4</td><td>Low processing power, low speed & cheaper</td><td>high processing power, high speed & expensive</td></tr><tr><td>5</td><td>Single or 2 stage pipelining</td><td>Deep pipelining (5-20 stages)</td></tr><tr><td>6</td><td>Many bit handling instructions are available</td><td>Only 1 or 2 bit handling instructions are available</td></tr></table>	1.5x4	6	
	2	• Fixed length instructions so less clock cycles required so its faster • RISC architecture gives more importance to Software. Here compiler takes on most of the burden • Only "LOAD & STORE" instructions can access memory			

- Large code size
- Highly Pipelined
- Requires fewer transistors for implementation so cheaper to design & produce

1.5x4 | 6

3

- PORT X register => When port set as output port PORTx used to send data out to pins likewise when when port set as input port then the data which is read from PIN X register is brought into this register .
- DDRx register => Data Direction register. When DDRx =1 => Port set as output port & when DDRx =0 => port set as input port.
- PIN X register =>**Port Input Pin** .When port is input port then all datas are first brought into this register & then only it is carried to CPU.

3x2 | 6

DIFFERENT STATES OF A PIN

PORTx \ DDRx	0	1
0	Input & high impedance	Out 0
1	Input & pull-up	Out 1

4

Sl. No	Data type/ Features	Unsigned Char	Signed Char (char)	Unsigned Int	Signed int(int)
1	No of Bits	8	8	16	16
2	Range (hex)	00-FF	00-FF	0000-FFFF	0000-FFFF
3	Range (dec)	0-255	-128to+127	0 to 65535	-32768 to +32767
4	No of bit for magnitude	8	7	16	15
5	Application	Counter value	Temperature	Memory address	---

- Apart from these data types there are also other data types like unsigned long, long, float, double etc

1.5x4 | 6

5

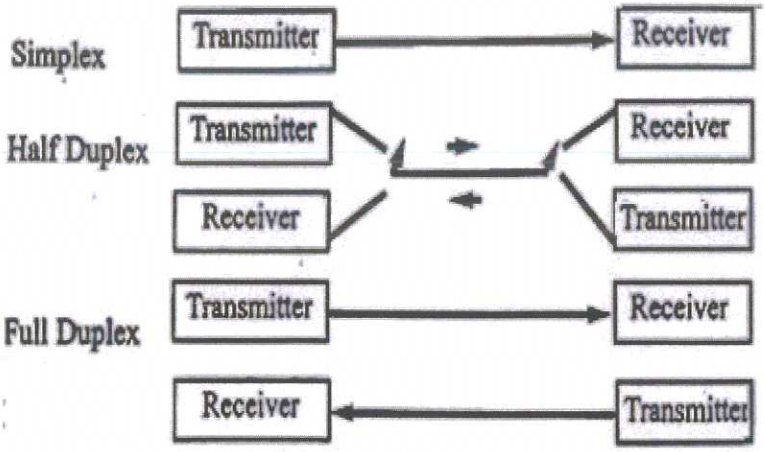
i. TCNT n

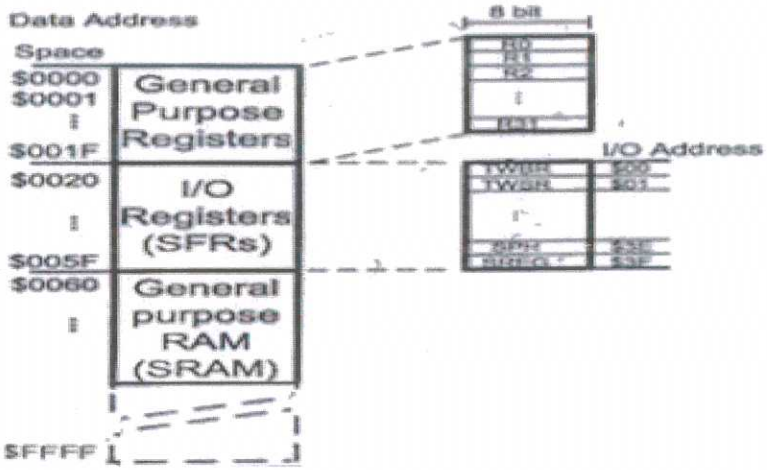
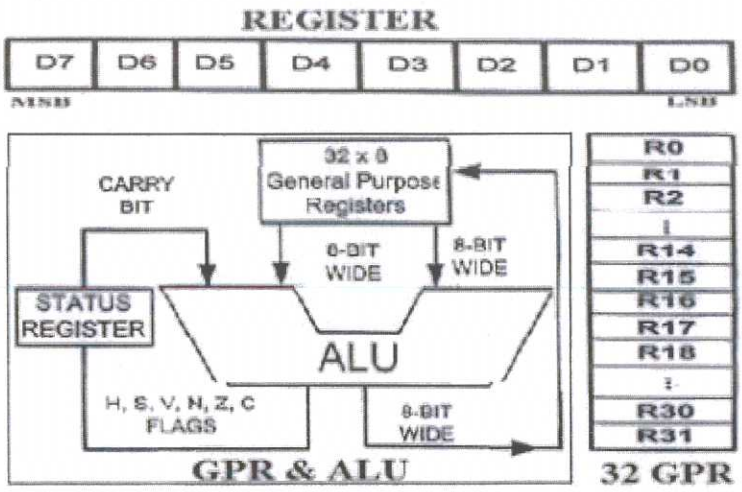
=>3 registers TCNT 0, TCNT 1, TCNT 2

=>Contents of Timer/Counter accessed using TCNTn

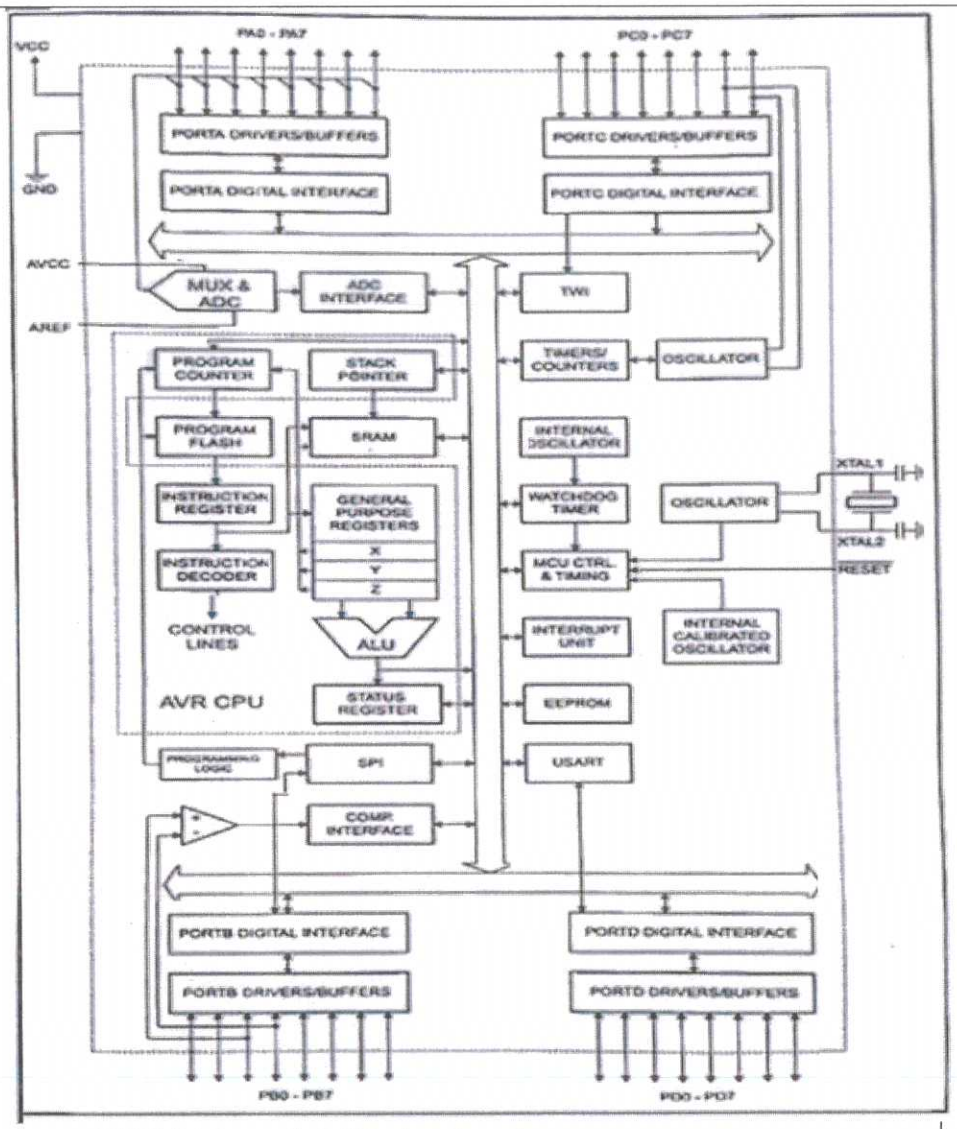
=>Upon reset it contains 0 & it counts up with each pulse.

30

		ii. TCCR n =>Timer/Counter control register for setting modes of operation for eg Timer 0 can be made to work as Timer or Counter by loading proper value into TCCR0 **iii. TIFR** =>Timer/Counter Interrupt flag register =>Contains TOVn (Timer Overflow flags), OCFn (O/P compare flags etc) **iv. OCR n** =>Output compare register . The contents of OCR n is compared with contents of TCNT n. => Timer registers can read/write using IN and OUT instructions like other I/O registers.	1.5x4	6	
	6	An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention i. Timers (2) => 1 Interrupt for overflow & 1 Interrupt for compare match ii. External hardware interrupts (3) => INT0 (PORTD.2), INT1(PORTD.3), INT2(PORTB.2) iii. Serial Communications USART (2) =>1 for receive & 1 for transmit iv. SPI Interrupts (1) v. ADC Interrupts (1) Apart from these interrupts there are many other interrupts also present	1 1x5	6	
	7	 • Simplex => Only 1 side data transmission eg computer to printer • Half Duplex => Bi-directional data transmission possible in either direction but not simultaneously • Full Duplex => Bi-directional data transmission possible in either direction simultaneously	3+3	6	

PART C					
III	(a)	Memory is of 2 types Data Memory (data stored) & Code memory (program stored) **DATA MEMORY**  • Data Memory = GPR (32 byte)+I/O Memory (64 byte)++Internal Data SRAM  • Majority of AVR registers=> 8 bit. 32 GPRs (R0-R31) present which are 8 bit registers. • ALU performs operations such as bit, arithmetic and logic upon the contents of the registers and write back the result into the register file into the designated registers. • Each ALU operation affects the flags in the STATUS registers depending upon the instructions • I/O Memory is the gateway to all the peripherals of the AVR processor.	4 5	9	60

		• It is implemented as SRAM and can be accessed in 2 ways as **SRAM ($20 to 5F) **I/O registers ($00 to $3F)- To access this AVR uses IN & OUT instructions. SBI & CBI can be used for address ($00 to &1F) • I/O memory is made up on 8 bit registers. These registers are also called SFR (Special function registers).They are dedicated to specific functions like I/O ports, ADC, serial Communication, Timers, status registers etc • Actual size of I/O depends on pin number & peripherals anyway minimum space of 64 bytes is definitely saved. This minimum 64bytes is called ***Standard I/O memory***			
	(b)	Data type can be positive or negative but only 8 bit. The AVR microcontroller supports the following types of data formats • Hex Data **Use $ or 0x eg LDI R16, 0x99 **LDI R16, $99 • Binary Data **eg LDI R16, 0b1001101 **LDI R16, 0B1001101 • Decimal Data **eg LDI R16,12 • ASCII formats- **LDI R16, '2' => R16=0011 0010 (or 32 in hex) • To represent a string we use double quotes & for defining ASCII strings we use .DB	1.5x4	6	

IV	(a)		6	9	
		• RISC & Harvard architecture based • GPR => 32 registers R0-R31 (32 bytes),I/O memory => 64Bytes ,Data RAM => 2KB, Flash Memory => 32KB, EEPROM => 1KB • 3 Timers (apart from watch dog timer) , 8 channel 10 bit ADC, 4 ports => port A, port B, port C, port D • Serial Interfaces like USART, SPI, I2C etc present. • PC => 14 bit, SP=> 2byte, SREG=> 8bit	3		
	(b)	Flag register is also called status register D7 D0 I T H S V N Z C • I-Global Interrupt enable. Setting this enables all interrupts & clearing it disables all.	3		

		• T-Bit copy storage. Used with BLD (Bit load) & BST (Bit store) instruction for loading & storing bits from 1 register to another. • H-Half carry (conditional flag)- H=1 when carry from D3 to D4.used for BCD • S-Sign flag (conditional flag)- S= exclusive or of V&N • V-Overflow flag (conditional) -V=1 when result too large & num overflows into D7 • N-Negative flag (conditional flag)-If D7=1 of result then -ve no so N=1 • Z-Zero flag (conditional flag)- If result =0 then Z=1 • C-Carry Flag (conditional flag)-carry out from D7	3	6	

V	(a)	#include <avr/io.h> //standard AVR header file
		int main (void)
		{
		unsigned char x, binbyte, d1,d2,d3;
		DDRB=DDRC=DDRD=0XFF; // port B,C,D output
		binbyte=0XFD; //Binary (hex) byte
		x=binbyte/10; // divide by 10 (0A)
		d1=binbyte%10; //find remainder (LSD)
		d2=x%10; //middle digit
		d3=x/10; //Most significant digit (MSD)
		PORTB=d1;
		PORTC=d2;
		PORTD=d3;
		return 0;
		}
			9	9	

	(b)	Three ways to create time delay in C **1) For loop** Two factors which affect the time delay created are • Crystal connected b/w XTAL1 & XTAL2. It must be accurate • Compiler Used. It converts C statements & functions into assembly language instructions. • Oscilloscope must be used to check the exact duration because often compilers does optimization before generating hex file. Here they often omit delay loop as it simply wastes CPU time. **2) Pre-defined C function** • Eg _delay_ms(), _delay_us() functions in delay.h • Problem=> different compilers use different names for delay so we have to change the name everywhere. To overcome it use macro or wrapper function. The later just	3x2	6	

does the work of calling the predefined delay function.

- Wrapper function consumes some microsecond time as delay

3) Using AVR timer

- Timer produces delay using a clock pulse to tick. The clock source can be internal or external. Internal clock source => feed frequency of crystal oscillator into timer. External clock source => feed clock pulse externally through pins. The later is called counter

VI (a)

```
#include <avr/io.h>                    //standard AVR header file
    int main (void)
{
    unsigned char x, y;
    unsigned char mybyte =0x29;
    DDRB=DDRC =0XFF;                   // makes ports B and C output
    x= mybyte & 0xFF;                  //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;
}
```

9 9

(b)

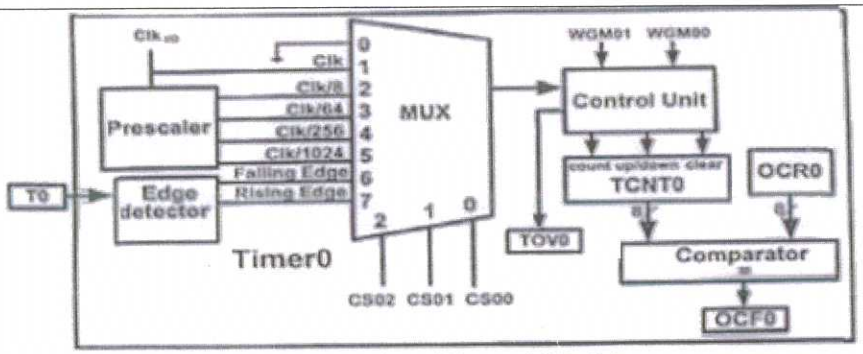
The different logic operations in C are as follows

- **Bit-wise operations in C**
  - BIT-WISE AND -$
  - BIT-WISE OR- |
  - BIT-WISE NOT-~
  - SHIFT RIGHT ->>
  - SHIFT LEFT -<<
  - Eg 0x35 & 0x0F=0X05, ~0X55=0XAA
- **Compound Assignment operators 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 operations in C & bit manipulation**
  - Shift right (>>)- eg 0b00010000>>3 = 0b00000010
  - Shift Left (<<) – eg 0b00010000<<3 =0b10000000
  - Numbers like 0b00100000 can be written as "1<<5" & numbers like 0b11101111 can be written as ~(1<<5)
  - To generate more complicated numbers OR operations can be used eg (1<<7)(1<<4)

3x2 6

VII	(a)	 **Steps For Timer 0 Programming (Normal Mode)** **1)** Load **TCNT0** with Initial count value **2)** Load value into **TCCR0** (indicate mode i.e 8 or 16 bit & the pre-scalar option). When you select the clock source (i.e Timer/Counter) then it starts to count & each tick causes contents of Timer/Counter to increase by 1. **3)** Get out of loop when **TOV0 (within TIFR register)** is raised. **4)** Stop Timer by disconnecting clock as shown LDI R20, 0x00 then OUT TCCR0, R20 **5)** Clear TOV0 for next round. **6)** Go back to step 1 to load TCNT0 again.	5 4	9	
	(b)	TIFR or Timer/ Counter Interrupt Flag Register has the following bits **OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0** • OCF2=>Output compare flags of Timer 2 • OCF1A, OCF1B =>Output compare flags of Timer 1 • OCF0 =>Output compare flags of Timer 0. For OCFx "1" means compare match occurred. • TOV2=> Timer Overflow Flags of Timer 2 • TOV1=> Timer Overflow Flags of Timer 1 • TOV0 => Timer Overflow Flags of Timer 0. Overflow occurs when count goes from $FF to $00 and then flag s set. Cleared by software i.e write 1 to it & 0 to all other flags . LDI R20, 0x01 then OUT TIFR, R20 • ICF1 => Input capture flag	3 3	6	
VIII	(a)	**TCCR0** (Timer/Counter 0 Control Register) **FOC0 | WGM00 | COM01 | COM00 | WGM01 | CS02 | CS01 | CS00** • **FOC0** => Force Compare Match. It's a write only bit which can be used while generating a wave. Writing 1 to it will cause the generator to act as if a compare match had occurred.	3		

- **WGM 01, 00**=>For Selecting the mode of Timer 0

WGM0	WGM01	TIMER 0 MODE SELECT BITS
0	0	Normal (content of timer increments with each clock)
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM Phase Correct
1	1	Fast PWM

- **COM01, 00** =>Compare Output Mode. These bits control the waveform generator

COM01	COM00	Description
0	0	Normal Port operation, OC0 disconnected
0	1	Reserved
1	0	Clear OC0 on compare match, set oc0 at TOP
1	1	Set OC0 on compare match, clear oc0 at TOP

- **CS02:00** => Used for selecting Timer 0 clock source

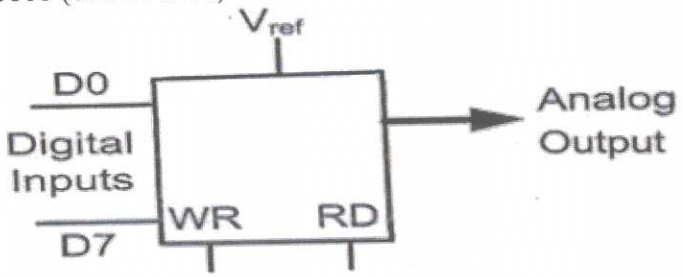
CS0	CS01	CS00	TIMER 0 CLOCK SELECT
0	0	0	NO Clock Source (Timer/Counter Stopped)
0	0	1	Clk (No Pre-scaling)
0	1	0	Clk/8
0	1	1	Clk/64
1	0	0	Clk/256
1	0	1	Clk/1024
1	1	0	External Clock source on T0 pin. Clock on falling edge
1	1	1	External Clock source on T0 pin. Clock on rising edge

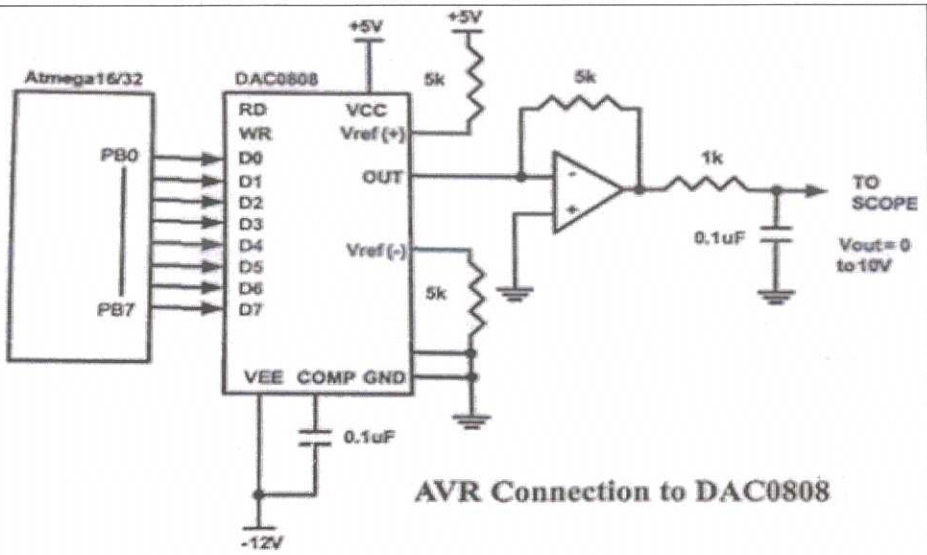
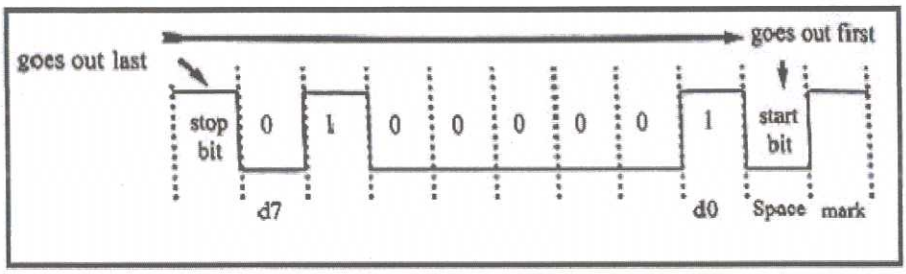
9

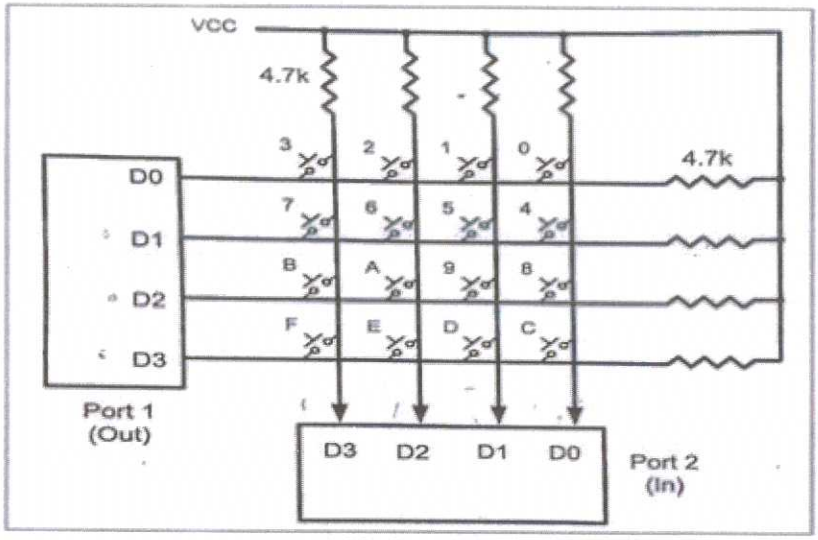
6

(b) An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention . When an interrupt gets activated then

1) Finish current task
    a. MC finishes the instruction its currently executing and saves the next instruction address (PC) on the stack.
2) IVT

		a. MC jumps to Interrupt vector table (**IVT**) from. Interrupt vector table is a place where the address of the program corresponding to the activated interrupt is stored. 3) ISR a. ISR or interrupt service routine is the program which needs to be executed so that the purpose of the currently activated interrupt is met 4) RETI • RETI or return from interrupt is the last instruction of the ISR program. • RETI execution causes loading of PC value from stack & continuing from the left over instruction.	1.5x4	6	
IX	(a)	• DAC Converts digital signal to analog signal.Common DAC's are DAC 0808 (or MC 1408)  **DAC Block Diagram** • Types of DAC are o Binary weighted DAC & o R/2R ladder DAC => Mostly used because of greater degree of precision. • The number of output levels of a "n" bit DAC is => 2^n .An 8 bit DAC provides 256 levels • In DAC 0808 (or MC 1408) the digital inputs are converted to current (Iout) & by connecting a resistor (feedback resistor=5KΩ) to the Iout pin, the result is converted to voltage. • The total current produced by DAC is given by Iout =Iref ((D7/2)+ (D6/4)+ (D5/8)+ (D4/16)+ (D3/32)+ (D2/64)+ (D1/128)+ (D0/256)) • Here Iref (usually 2mA)is the current which is to be applied to pin 14 & D0-D7 are digital binary bits. If all the inputs are high and Iref =2mA then Iout=1.99mA	4	9	

		 **AVR Connection to DAC0808**	5		
	(b)	 • Framing => placing of data to be transmitted between start & stop bit. • Start bit & stop bit => start bit is always one bit (usually 0) while stop bit can be 1 or more bits (usually 1). In older systems 2 or more stop bits were used but now only 1 stop bit is used. • Fig shows transmission of letter "A" whose ASCII is "0100 0001". Now ASCII is 8 bit • Total Bits for 1 character => 10 bits. Sometimes parity bit is also included. In odd parity the number of 1's in data bits including parity bit is odd. • Data Transfer rate => expressed in bps (or baud rate).	3 3	6	
X	(a)	• 4x4 keyboard means => 4 Rows & 4 Columns. • Rows are connected to PORT 1 (O/P Port) of ATmega32 & Columns are connected to PORT 2 (I/P Port) of ATmega32 • Columns are connected to VCC via a 4.7 K resistor & the Rows are connected to PORT 1 which are grounded. • There is no connection between ROW & COLUMN but when a key is pressed the corresponding row and column gets connected.			



Matrix Keyboard Connection to Ports

3+6 | 9

Procedure for Key Press Detection

(i) When no key press then Port 2, $D_3D_2D_1D_0$= 1111 & Port 1, $D_3D_2D_1D_0$=0000

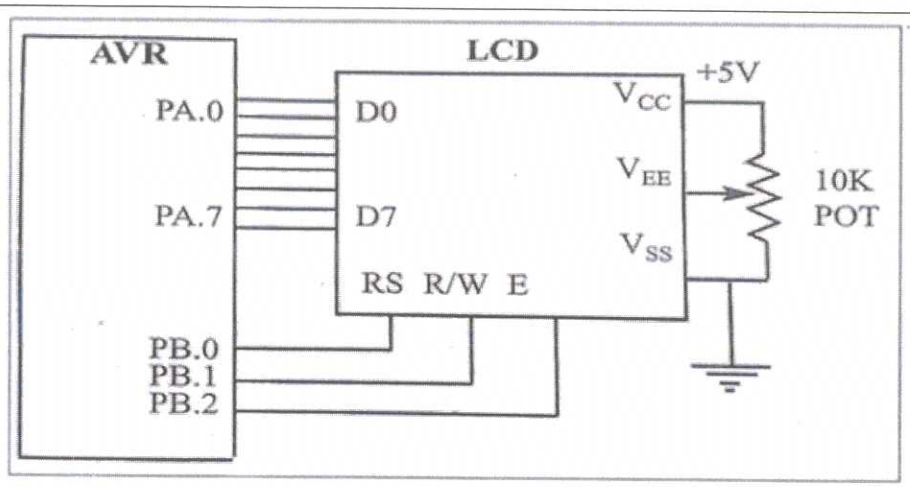
(ii) When a key is pressed (say "**9**") then at Port 2, D1=0 occurs & is detected by Microcontroller when it scans Port 2. Thisway Microcontroller understands that 1 of the key in column D1 is pressed.

(iii) Prior to conforming key detection Microcontroller waits for 20ms because it can be erroneous key press & also it can be a Multiple key press event so to confirm that it is a valid press it must remain pressed for min 20ms.

(iv) After detecting the column of key press now the Microcontroller needs to find the row of key press, for which it grounds only row 1 (i.e D0=0 of Port 1) & check if the value of $D_3D_2D_1D_0$ of Port 2 is same as that of earlier obtained value (i.e $D_3D_2D_1D_0$ = 1101 in case of "**9**" press). If its not, then the next row (i.e D1=0 of Port 1) is made so keeping other $D_3D_2D_0$=111 .

(v) This process is repeated until a low value is obtained in $D_3D_2D_1D_0$ of Port 2

(vi) When Row & Column are detected then it sets up the starting address for the lock-up table holding the scan codes (or ASCII values) for that row.

	(b)	 **LCD Connections for 8-bit Data** 1. Initialize LCD =>for 5x7 matrix and 8 bit operation use commands like 38, 01,02,06,80. 2. Send commands (as per need) to the LCD • Make RS=0 & R/$\overline{W}$=0 • Put the command number on data bus • Send a high to low pulse on enable pin to enable the internal latch of LCD. 3. Send the character to be shown on the LCD • Make RS=1 & R/$\overline{W}$=0 • Put the data on data bus • Send a high to low pulse on enable pin to enable the internal latch of LCD.	3 3	6	

picture
quality low