SCHEME OF EVALUATION

SCORING INDICATOR

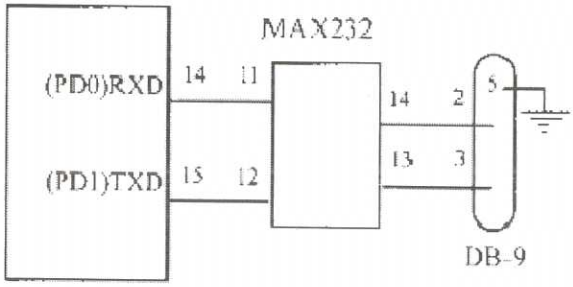
Revision: 2015 Course Code: 6132

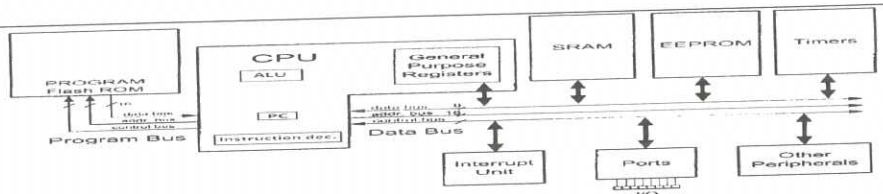
Course Title: MICROCONTROLLER

<table>
<tr><th>Qstn NO</th><th>Scoring indicator</th><th>Split up score</th><th>Sub total</th><th>Total</th></tr>
<tr><td>I</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1.</td><td>BRNE, BRLO, BRSH, BREQ etc..</td><td>1+1</td><td>2</td><td></td></tr>
<tr><td>2.</td><td>This instruction tests the bit and skips the instruction just below if the bit is low</td><td>2</td><td>2</td><td></td></tr>
<tr><td>3.</td><td>DDRx, PINx, PORTx</td><td>2</td><td>2</td><td></td></tr>
<tr><td>4.</td><td>Normal mode and Compare match mode</td><td>1+1</td><td>2</td><td></td></tr>
<tr><td>5.</td><td>In asynchronous serial communication, each character is placed between stop and start bit. It is known as framing</td><td>2</td><td>2</td><td>10</td></tr>
<tr><td>II</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1.</td><td>
<table>
<tr><th>Microcontroller</th><th>Microprocessor</th></tr>
<tr><td>It contains fixed amount of RAM, ROM and I/O ports and timer all on a single chip.</td><td>It contains no ROM, no RAM, no timer and no I/O ports on the chip itself.</td></tr>
<tr><td>Occupies less space, draw less power and cheap.</td><td>Needs more space consume more power less reliable and costly.</td></tr>
<tr><td>Microcontrollers have memory oriented architecture.</td><td>Most of the microprocessors have register oriented architecture.</td></tr>
<tr><td>Microcontrollers based systems are easy to troubleshoot and maintain.</td><td>Microprocessor based systems are not easy to troubleshoot and maintain.</td></tr>
<tr><td>Designer cannot add any external memory, I/O ports or timer.</td><td>Designer decides the amount of RAM, ROM and I/O ports needed to fit the task.</td></tr>
</table>
</td><td>6</td><td>6</td><td></td></tr>
<tr><td>2.</td><td>
<table>
<tr><td>I</td><td>T</td><td>H</td><td>S</td><td>V</td><td>N</td><td>Z</td><td>C</td></tr>
</table>
C - Carry flag N - Negative flag S - Sign flag T - Bit Copy Storage
Z - Zero flag V - Overflow flag H - Half flag I - Global Interrupt Enable
</td><td>6</td><td>6</td><td></td></tr>
</table>

3.	**DDRx register**
	
	To work the DDRX register an output port set DDRX register 1.
	
	LDI R16, OXFF
	OUT DDRB, R16
	
	To make a port an input port, we must put zeros into the DDRX register.
	
	LDI R16, 0X00
	
	OUT DDRB, R16
	
	PIN register
	To read the data present at the pins, we should read the PIN register.
	
	To bring the data into CPU from pins , we read the contents of the PINxregister.
	
	PORT register
	
	To send data out to pins, we use PORTx register.
		3*2	6	

4.	**DDRB=0xFF** **For(i=0; i<=255; i++)** **{** **Portb=i;** **}**	6	6	
5.	1.Load the TCNTO register with the initial count value 2. Load the value into the TCCR0. 3. Keep monitoring the timer over flow flag(TOV0) to see if it is raised. 4. Stop the timer by disconnecting the clock source. 5. Clear the TOVO flag for the next round. 6. Go back to Step 1 to load TCNTO again.	6	6	
6.	The ATmega32 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TX and RX and are part of the PortD group (PD0 and PD1) of the 40-pinpackage. Pin15 of the ATmega32 is assigned to TX and pin 14 is designated as RX. These pins are TTL compatible; therefore , they require a line driver to make them RS232 compatible. One such line driver is the MAX232chip.	Exp: 3 Fig: 3	6	

	ATmega32  40-Pin DIP Package ATmega32			
7.	Duplex – Data can be both transmitted and received. Simplex – only sends data. Duplex transmissions can be half or full duplex. Half duplex – Data is transmitted one way at a time. Full duplex – Data can go both ways at the same time.	6		30
III a	The **data memory** is composed of three parts: **1. GPRs(General Purpose Register)** **2. I/O Memory** **3. Internal SRAM** Students should explain with figure	Exp: 4.5 Fig: 4.5	9	
III b	1. AVR is an 8-bit RISC single chip microcontroller with Harvard architecture. 2. AVR's have some standard features such as on-chip program ROM, data RAM, data EEPROM, timers and I/O ports. 3. Most AVRs have some additional features like ADC (Analog to Digital Converter), PWM and different kinds of serial interfaces such as USART (Universal Synchronous Asynchronous Receiver Transmitter), CAN (Controller Area Network), USB(Universal Serial Bus). 4. The Program ROM size can vary from **1k to 256k**. 5. Uses on-chip **flash memory** for program storage. 6. Flash memory can be **erased easily in seconds** compared to UV-EEPROM. 7. AVR has maximum of 64k bytes of data RAM space. 8. AVR can have from **3 to 86 pins for I/O**. 9. AVR can have **up to 6 timers**.	Any 6 points 6*1	6	
IV a	In harvard architecture, there are separate buses for the code and data. Program bus provides access to the program flash ROM and data bus is used for bringing data to the CPU. In the program bus - data bus is 16 bit wide and address bus is as wide as the PC register. In the data bus - address bus is 16 bits wide and data bus is 8 bit wide. The memory locations are divided into lower byte and higher byte.	Exp: 4 Fig: 5	9	

IV b				

	Hex numbers
	There are two ways to show hex numbers:
	1. Put Ox (or 0X) in front of the number like this: LDI R16, Ox99
	2. Put$ in front of the number, like this: LDI R22, $ 99
	Binary numbers
	There is only one way to represent binary numbers in an AVR assembler. It is as follows:
	
	LDI Rl6,0bl0011001 ; R16 = 10011001 or 99 in hex
	Decimal numbers
	To indicate decimal numbers in an AVR assembler we simply use the decimal and nothing before or after it.
	LDI R17, 12
	ASCII characters
	To represent ASCII data in an AVR assembler we use single quotes as follows:
	LDI R23,'2'
		4*1.5	6	

V a	Unsigned char a= '3' Unsigned char b='5'; **a= a&0f;** **a= (a<<4);** **b=b&0f;** **a=a|b;** **bcd=a;**	9	9	
V b	1. Using a simple for loop 2. Using predefined C functions 3. Using AVR timers	3*2	6	

VI a	Unsigned char a=29
	b=a;
	x=a&0f;
	y=x|30;
	z=a&f0;
	z=z>>4;
	y1=z|30;
		9	9	

VI b	1)SBI (Set bit in I/O register) To set HIGH a single bit of a given I/O register, we use the following syntax. SBI ioReg, bit_num IoReg – can be lower 32 I/O registers (address 0 to 31) and bit_num is the desired bit number from 0 to 7. These instructions can be used for manipulation of bits D0-D7 of the lower 32 I/O registers. 2)CBI (Clear bit in I/O register) To clear a single bit of a given I/O register.	6	6	

Syntax is

CBI ioReg, bit _ number
Eg : CBI PORT B,2 ; bit clear PB2 =low.
PB2 is the 3 rd bit of port B.

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

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

Instruction format is SBIS a, b

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

To monitor the status of a single bit for low

This instruction tests the bit and skips the instruction right below it if the bit is low.
Instruction format is
SBIC a, b

VII a

1. **TCCR0(Timer/Counter Control Register) register**

TCCR0 is an 8-bit register used for control of Timer0..
The bits for TCCR0 are shown below:

7	6	5	4	3	2	1	0
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
W	RW	RW	RW	RW	RW	RW	RW
0	0	0	0	0	0	0	0

CS02:CS00 (Timer0 clock source)
These bits in the TCCR0 register are used to choose the clock source.

If CS02:CS00=000, then the counter is stopped. If CS02-CS00 have values between001 and 101, the oscillator is used as clock source and the timer/counter acts as a timer. In this case, the timers are often used for time delay generation.

If CS02-CS00 are 110 or 111, the external clock source is used and it acts as a counter

CS02:00	D2	D1	D0	Timer0 clock selector
	0	0	0	No clock source (Timer/Counter stopped)
	0	0	1	clk (No Prescaling)
	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.

WGM01 :WGM00
Timer0 can work in four different modes:

1. Normal mode 3. CTC

2. Fast PWM 4. Phase correct PWM

The WGM01 and WGM00 bits are used to choose one of them.

WGM00, WGM01

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM01:00 D5 D4 Compare Output Mode:
These bits control the waveform generator •

(1 and 2) *3=6
(3 and 4)*1.5=3

9

5

2. TIFR(Timer/counter Interrupt Flag Register) register

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.

TOV1 D2 Timer1 overflow flag bit
OCF1B D3 Timer1 output compare B match flag
OCF1A D4 Timer1 output compare A match flag
ICF1 D5 Input Capture flag
TOV2 D6 Timer2 overflow flag
OCF2 D7 Timer2 output compare match flag

3. TCNTn (timer/Counter)

InATmega32we have TCNTO, TCNTI, and TCNT2.

The TCNTn register i s a counter.

Upon reset, the TCNTn contains zero. It counts up with each pulse. You can load a value into the TCNTn register or read its value.

4. OCRn(Output Compare Register)

The content of the OCRn is compared with the content of the TCNTn. When they are equal the OCFn (Output Compare Flag)flag will be set.

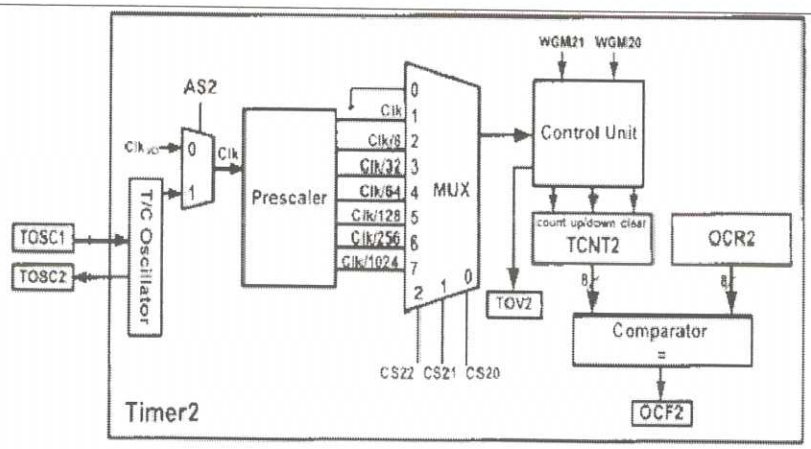
The timer registers are located in the I/O register memory. Therefore, you can read or write from timer registers using IN and OUT instructions, like the other I/O registers

VII b

Interrupts	Polling
Whenever any device needs the microcontroller's service, the device notifies it by sending an interrupt signal.	The microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service.
Each device is serviced based on the priority as signed to it.	It checks all devices in a round-robin fashion.
It can ignore a device request.	It cannot ignore a device request.
It avoids the tying down the microcontroller.	It wastes much of the time by polling devices that do not need service.

4*1.5 6

VIII a



Exp:3
Fig:6

9

VIII b

There are three external hardware interrupts in the ATmega32:

1. INTO(occur at pin PD2 ie, PORTD.2)
2. INTI (occur at pin PD3 ie, PORTD.3)
3. INT2(occur at pin PB2 ie, PORTB.2)

When these pins are activated, then the AVR is interrupted in whatever it is doing.

Then it jumps to the vector table to perform the associated interrupt service routine.

The interrupt vector table locations $2,$4, and $6 are set aside for INT0, INT1, and INT2, respectively.

INT0

The INT0 is a low-level-triggered interrupt. When a low signal is applied to pinPD2(PORTD.2),the controller will be interrupted and jump to location $0002 in the vector table to service the ISR.

6

6

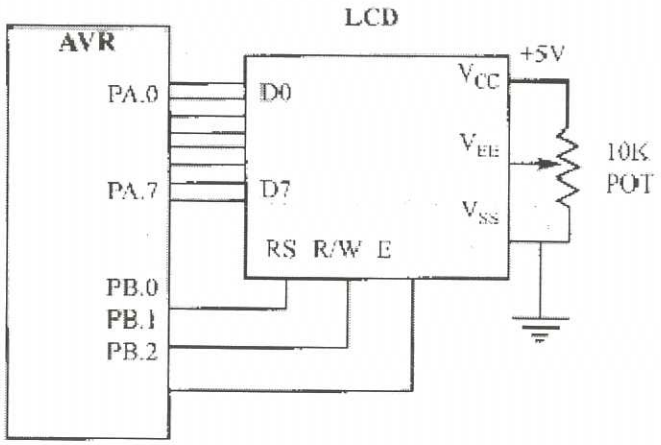
INT1

INT1 is either low-level-triggered or edge triggered interrupt.

INT2

INT2 isedge triggered interrupt. The hardware interrupts must been abled before they can take effect. This is done using the INTx bit located in the GICR register.

IX



LCD pin descriptions

LCD has 14pins..

The function of each pins are:

Vcc, VEE,and Vss

Vcc : +5 V

Vss :ground

VEE :used for controlling LCD contrast

Fig:5
Exp:10

15

RS (Register Select)

There are two very important registers inside the LCD.-command code register, data register

The RS pin is used for their selection as follows.

If RS=0, the instruction command code register is selected, allowing the user to send commands such as clear display, cursor at home, and so on.

If RS= 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W (Read/Write)

R/W input allows the user to write information to the LCD or read information from it.

R/W=1 when reading; R/W=0 when writing.

E (Enable)

The enable pin is used by the LCD to latch information presented to its data pins.

D0-D7

The 8-bit data pins, DO-D7,are used to send information to the LCD or read the contents of the LCD's internal registers.

Sending commands and data to LCDs

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

1. Initialize the LCD .

To initialize the LCD for 5x 7matrix and 8-bit operation, the following sequence of commands should be sent to the LCD: Ox38, Ox0E , and 0x01. Next we will show how to send a command to the LCD. After power-up you should wait about 15ms before sending initializing commands to the LCD.

2. Send any of the commands to the LCD.

To send any of the commands to the LCD , make pins RS and *RIW*=0 and put the command number on the data pins (D0-D7). Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. Notice that after each command you should wait about 100µs. After the 0x01 and Ox02 c o m m a n d s you should wait for about 2ms.

3. Send the character to be shown on the LCD.

To send data to the LCD, make pins RS= 1 and *RIW*=0. Then put the data on the data pins (DO-D7)and send a high-to-low pulse to the E pin to enable the internal latch of the LCD. After sending data you should wait about 100 µs to let the LCD module write the data on the screen.

X

In the AVR microcontroller five major registers are associated with the ADC.

They are ADCH (high data),ADCL (low data),ADCSRA (ADC Control and Status Register), ADMUX (ADC multiplexer selection register), and SPIOR (Special Function I/O Register).

1. ADCH
2. ADCL
4. ADMUX
5. SPIOR

3. ADCSRA

1. ADMUX register

It is ADC multiplexer selection register.

This register is used to select the reference voltage, select whether the result is left adjusted or right adjusted and select the analog inputs.

The bits of ADMUX register is shown below:

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Register:8
Polling steps:7

15

2. ADCH:ADCL registers

After the *AID* conversion is complete, the result sits in registers ADCL *(A/*D Result Low Byte) and ACDH (A/D Result High Byte).

3. ADCSRA register

The ADCSRA register is the status and control register of ADC.

Bits of this register control or monitor the operation of the ADC.

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN: This bit enables or disables ADC

ADSC: To start each conversion, have to set this bit to one.

ADATE: Auto triggering of the ADC is enabled when you set this bit to one

ADIF: This bit is set when an ADC conversion completes and the data registers are updated

ADIE: Setting this bit to one enables the ADC conversion complete interrupts

ADPS2:0: These bits determine the division factor between the XTAL frequency and the input clock to the ADC

Steps in programming the A/D converter using polling

To program the *A*/D converter of the AVR, the following steps must be taken:

1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module of the AVR because it is disabled upon power-on reset to save power.
3. Select the conversion speed. We use registers ADPS2:0 to select the conversion speed.
4. Select voltage reference and ADC input channels. We use the REFS0 and REFS1 bits in the ADMUX register to select voltage reference and the MUX4:0 b i t s in ADMUX to select the ADC input channel.
5. Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6. Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7. After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output. Notice that you have to read ADCL before ADCH; otherwise, the result will not be valid.
8. If you want to read the selected channel again, go back to step5.

Description of each bits of the ADCSRA register is shown below:
