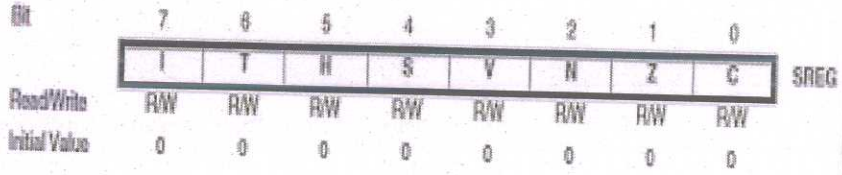
SCHEME OF VALUATION
(Scoring Indicators)

Revision: 2015

Course Title: Embedded Systems **Course Code: 5041**

Qst No	Scoring Indicator	Split up score	Sub total	Total
	PART A			
I (1)	Any 4 features. 32K ROM,2K RAM, 1K EEPROM, 8bit controller, RISC, Harvard architecture.	0.5*4	2	
I (2)	16K ROM, 1K RAM	1+1	2	
I (3)	ADD,SUB,MUL,INC,DEC	0.5*4	2	10
I (4)	INT0,INT1,INT2 (any2)	1+1	2	
I (5)	In RTOS implementation of a design, the program is divided into different independent functions what we call as a *task*.	2	2	
	PART B			
II(1)		3	6	30
	Bit 7 – I: Global Interrupt Enable The Global Interrupt Enable bit must be set for the interrupts to be enabled.	3		
	Bit 6 – T: Bit Copy Storage The Bit Copy instructions BLD (Bit LoaD) and BST (BitSTore) use the T-bit as source or destination			
	Bit 5 – H: Half Carry Flag The Half Carry Flag H indicates a half carry in some arith- metic operations. Half Carry is useful in BCD arithmetic.			
	Bit 4 – S: Sign Bit, $S = N \oplus V$ The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V			
	Bit 3 – V: Two's Complement Overflow Flag The Two's Complement Overflow Flag V supports two's complement arithmetics.			
	Bit 2 – N: Negative Flag The Negative Flag N indicates a negative result in an arithmetic or logic operation.			
	Bit 1 – Z: Zero Flag The Zero Flag Z indicates a zero result in an arithmetic or logic operation.			
	Bit 0 – C: Carry Flag			

II(2)	The Carry Flag C indicates a carry in an arithmetic or logic operation. Assembler directives- These are the statements that direct the assembler to do something. As the name says, it directs the assembler to do a task. INCLUDE- This directive is used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code. Its format is: .INCLUDE path: file name EQU - The equate directive is used to substitute values for symbols or labels. The format is .EQU label = expression ORG - The origin directive sets the location counter to the value specified. Subsequent statements are assigned memory locations starting with the new location counter value DEF - The DEF directive allows the registers to be referred to through symbols. A defined symbol can be used in the rest of the program to refer to the register it is assigned to. Syntax: .DEF Symbol=Register	2 any 4 1*4	6	
~~II(3)~~ II (3)	DW, DD, DQ, MACRO, ENDMACRO Assembler macros can be powerful tools that allow you to reuse sequences of code over and over. Creating a library of macros for your most commonly used functions will allow you to code faster and make less mistakes. Macros are defined by placing code between the assembler directives .macro and .endmacro. For example, a macro myMacro can be defined as .macro myMacro ... .endmacro ; macro code	2 2 2	6	

II(4)	Example
	
	#include <avr/io.h>
	int main(void)
	{
	while (1)
	{
	unsigned char x='5';
	unsigned char z='9';
	DDRB=0xff;
	x = x & 0x0f;
	z = z & 0x0f;
	z = z<<4;
	PORTB=x|z;
		6	6	

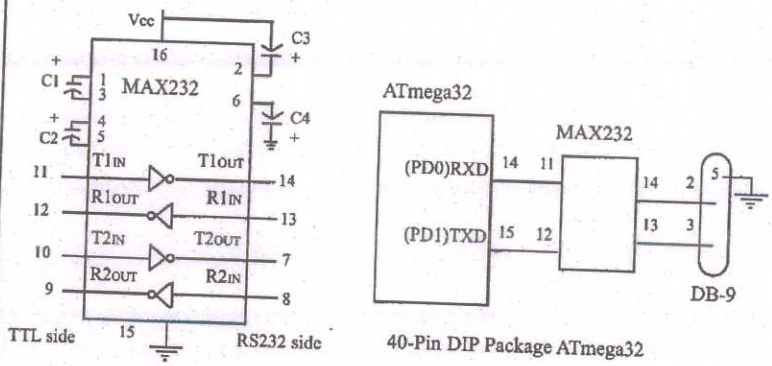
~~II(5)~~

```
}
    return 0;
}
```

Output PORTB=0x59

fig 4 expl 2 — 6

II (5)



In general, a kernel's memory

~~II(6)~~

II (6) management responsibilities include:

6*1 — 6

Managing the mapping between logical (physical) memory and task memory references.

Determining which processes to load into the available memory space.

Allocating and deallocating of memory for processes that make up the system.

Supporting memory allocation and deallocation of code requests (within a process),
such as the C language "alloc" and "dealloc" functions,
or specific buffer allocation and
deallocation routines.

Tracking the memory usage of system components.

~~II(7)~~ Ensuring cache coherency (for systems with cache).

Ensuring process memory protection.

II (7)

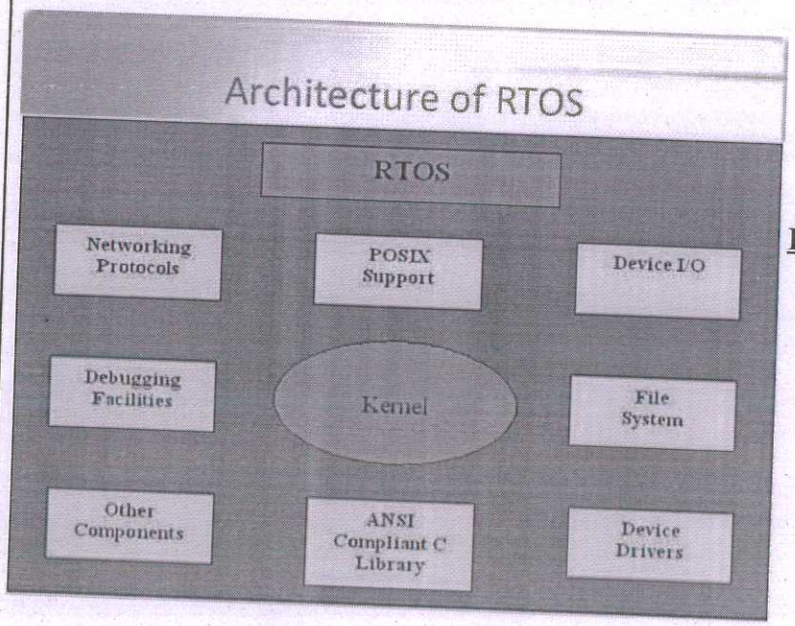
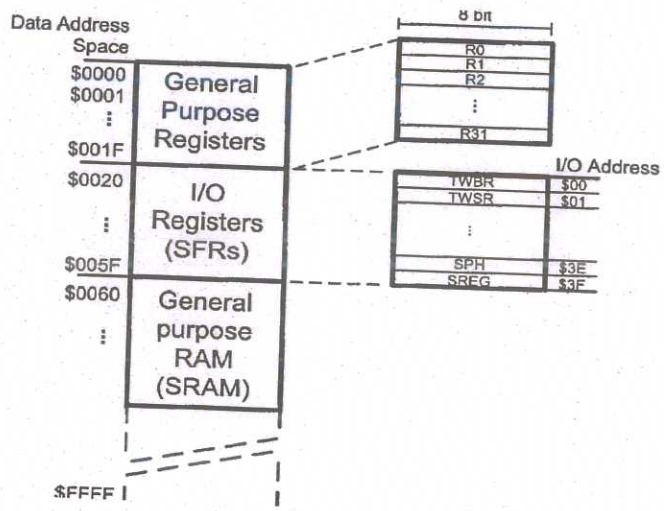
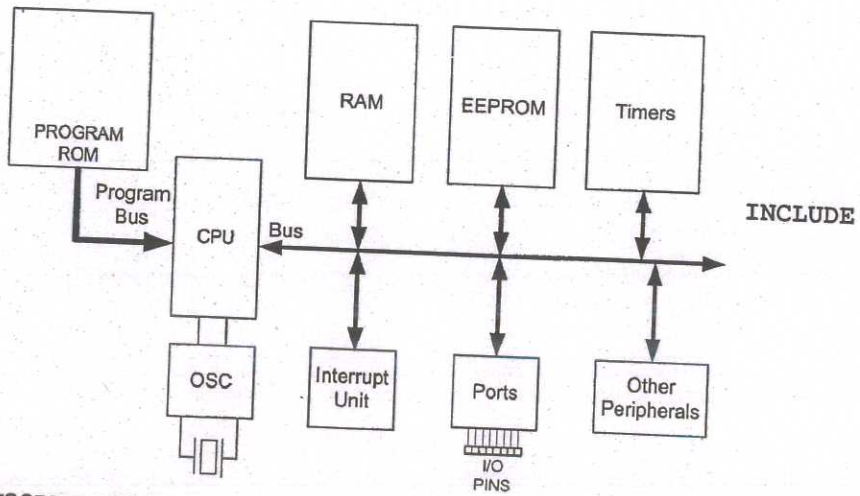


fig 4 expl 2 — 6

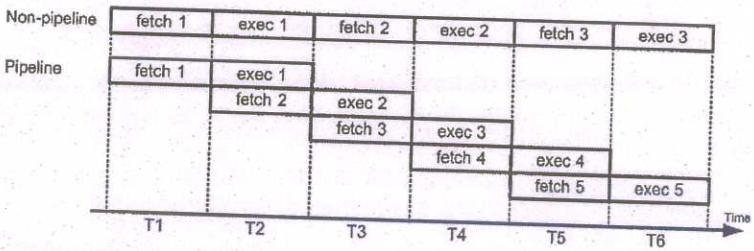
PART C

III(a)		any 5 1*5	5	15
III(b)	Microchip company, Texas instruments, Silicon Lab, Zilog, Intel corporation, Dallas Semiconductor, Renesas, National, Freescale semiconductor.	list. 5 eg. 5	10	
III(b)	1. Single-Register (Immediate) 2. Register 3. Direct 4. Register indirect 5. Flash Direct 6. Flash Indirect			
IV(a)		fig 5 expl 2	7	15
IV(b)				
V(a)		fig 6 expl 2	8	
V(a)	"M32DEF.INC" LDI R20,0x00 ;R20 = 00000000 (binary) INCLUDE	7	7	15

```
    OUT  DDRB,R20  ;DDRA = R20
LDI  R20,0xFF  ;R20 = 11111111 (binary)
    OUT  DDRD,R20  ;DDRB = R20
L1: IN   R20,PINB  ;R20 = PINA
    OUT  PORTD,R20 ;PORTB = R20
    RJMP L1
```

V (b) fig4 expl4 — 8

V (b) We can use a pipeline to speed up execution of instructions. In pipelining, the process of executing instructions is split into small steps that are all executed in parallel. In this way, the execution of many instructions is overlapped. One lim-



VI (a) — 7 — 7 — 15

VI (a)
```
INCLUDE "M32DEF.INC"
    LDI  R20,0x8A   ;R20=LSB of num1
    LDI  R21,0x3B   ;R21=LSB of num2
    LDI  R22,0x8A   ;R22=MSB of num1
    LDI  R23,0x3B   ;R23=MSB of num2
    ADD  R20,R21    ;ADD LSBs
    ADC  R22,R23    ;ADD MSBs with carry
    STS  $301,R20   ;LSB of result in $301
    STS  $302,R22   ;MSB of result in $302
    RJMP L1
```

VI (b) — 8 — 2

VI (b) The unconditional branch is a jump in which control is transferred unconditionally to the target location. In the AVR there are three unconditional branches: JMP (jump), RJMP (relative jump), and IJMP (indirect jump). Deciding which one to use depends on the target address. Each instruction is explained next. — 2

JMP (JMP Is a long jump)

JMP is an unconditional jump that can go to any memory location in the 4M (word) address space of the AVR. It is a 4-byte (32-bit) instruction in which 10 bits are used for the opcode, and the other 22 bits represent the 22-bit address of the target location. The 22-bit target address allows a jump to 4M (words) of memory locations from 000000 to $3FFFFF. So, it can cover the entire address

RJMP (relative jump)

In this 2-byte (16-bit) instruction, the first 4 bits are the opcode and the rest — 2

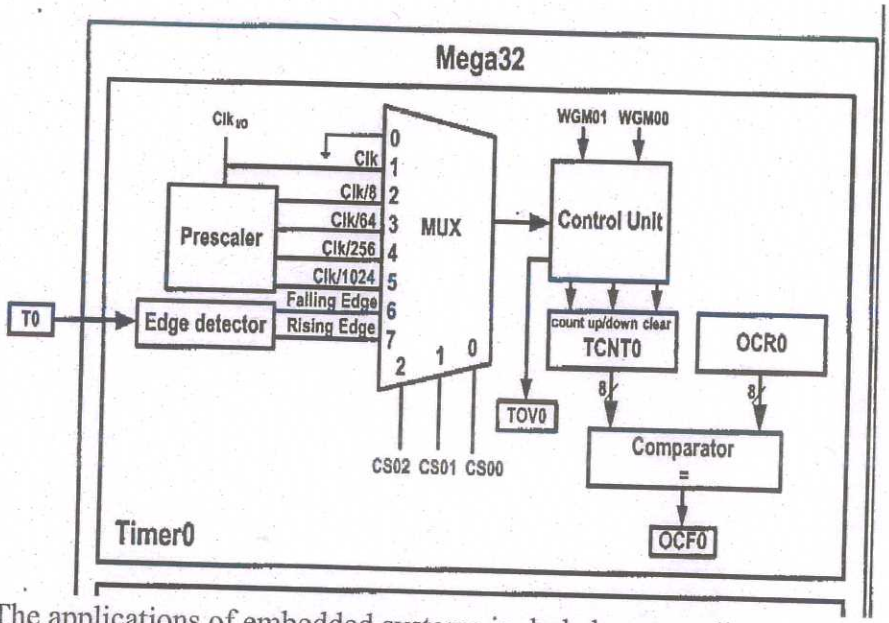
		2		
VII (a)	***IJMP (indirect jump)*** IJMP is a 2-byte instruction. When the instruction executes, the PC is loaded with the contents of the Z register, so it jumps to the address pointed to by the Z register. As you will see in Chapter 6, Z is a 2-byte register, so IJMP can jump within the lowest 64K words of the program memory.	7	7	15

VII (b)	`#include <avr/io.h>` // standard avr header
	`int main(void)`
	`{`
	`DDRB = (1<<3);` // PORTB,3 pin as output
	`unsigned char z,i;` //declaration of variables
	`z = 0X35; //data input;`
	`for (i=0; i < 8; i++)`
	`{`
	`if (z & 0x01)// check the d0 bit of z`
	`PORTC=PORTC |0b00001000;` //make PORTC,pin3 high
	`else`
	`PORTC =PORTC&0b11110111;` //make PORTC,pin3 low
	`z=z>>1;` // shift the data towards right
	`}`
	`return 0;`
	`}`
		4	8	

	Interrupt priority If two interrupts are activated at the same time, the interrupt with the higher priority is served first. The priority of each interrupt is related to the address of that interrupt in the interrupt vector. The interrupt that has a lower address, has a higher priority. See Table 10-1. For example, the address of external interrupt 0 is 2, while the address of external interrupt 2 is 6; thus, external interrupt 0 has a higher priority, and if both of these interrupts are activated at the same time, external interrupt 0 is served first.	4		
VIII (a)	**Interrupt latency** The time from the moment an interrupt is activated to the moment the CPU starts to execute the task is called the *interrupt latency*. This latency is 4 machine cycle times. During this time the PC register is pushed on the stack and the I bit of the SREG register clears, causing all the interrupts to be disabled. The duration of an interrupt latency can be affected by the type of instruction that the CPU is executing when the interrupt comes in, since the CPU finishes the execution of the current instruction before it serves the interrupt. It takes slightly longer in cases where the instruction being executed lasts for two (or more) machine cycles (e.g., MUL) compared to the instructions that last for only one instruction cycle (e.g., ADD). See the AVR datasheet for the timing.	7	7	15

VIII (b)	`#include <avr/io.h> // standard avr header`
	`#include <util/delay.h>`
	`int main(void)`			

VIII (b)	{
	DDRB = 0xFF; // PORTB as output
	PORTB = 0X55; // initializing PORTB with 55h
	while(1) // indefinite loop
	{
	PORTB-0x55; // PORTB=55h
	_delay_ms(1000); // predefined delay of 5000ms
	PORTB-0xAA; // PORTB = AAh
	_delay_ms(1000); // predefined delay of 5000ms
	}
	return 0;
	}
		fig 5
		expl 3
			8	

VIII (b)				
IX (a)		5x1	5	15
IX (b) (a)	The applications of embedded systems include home appliances, office automation, security, telecommunication, instrumentation, entertainment, aerospace, banking and finance, automobiles personal and in different embedded systems projects. ,	6	6	
IX (b)	Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible			

	knowledge that can be of great help to novices and experts alike All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide. the Arduino project provides an integrated development environment (IDE) based on the Processing language project.			
IX (c)	The *kernel* is the fundamental part of an operating system. It is responsible for managing the resources and the communication between hardware and software components. The *kernel* offers hardware abstraction to the applications and provides secure access to the system memory. The main tasks of the kernel are : Process management Device management Memory management Interrupt handling I/O communication File system... Etc	5	5	
X(a)	Intertask Commucication Different tasks in an embedded system typically must share the same hardware and software resources or may rely on each other in order to function correctly. For these reasons, embedded OSs provide different mechanisms that allow for tasks in a multitasking system to intercommunicate and synchronize their behavior so as to coordinate their functions, avoid problems, and allow tasks to run simultaneously in harmony. Embedded OSs with multiple intercommunicating processes commonly implement interprocess communication (IPC) and synchronization algorithms based upon one or some combination of memory sharing, message passing, and signaling mechanisms.	4	12	15
	Context switching Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes. There are three situations that a context switch is necessary, as shown below.	4		

	Multitasking - When the CPU needs to switch processes in and out of memory, so that more than one process can be running. **Kernel/User Switch** - When switching between user mode to kernel mode, it may be used . **Interrupts** - When the CPU is interrupted to return data from a disk read. **Mutual exclusion** **Mutual exclusion** is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time. Like when you talk about cpu scheduling the operating system decides which process would get to use cpu first. That is done by allocating cpu for a particular time interval between all the processes running parallel. So when cpu is busy doing one task, the other process is in a suspended state. But this time is usually very small so we see all processes running parallel on a multi-tasking os.	4		
X (b)	VxWorks ,QNX ,eCos ,RTLinux ,Symbian OS,,eCos, LynxOS, QNX, RTAI, RTLinux, VxWorks, Windows CE, MontaVista Linux	any 3 3x1	3	