CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test – II

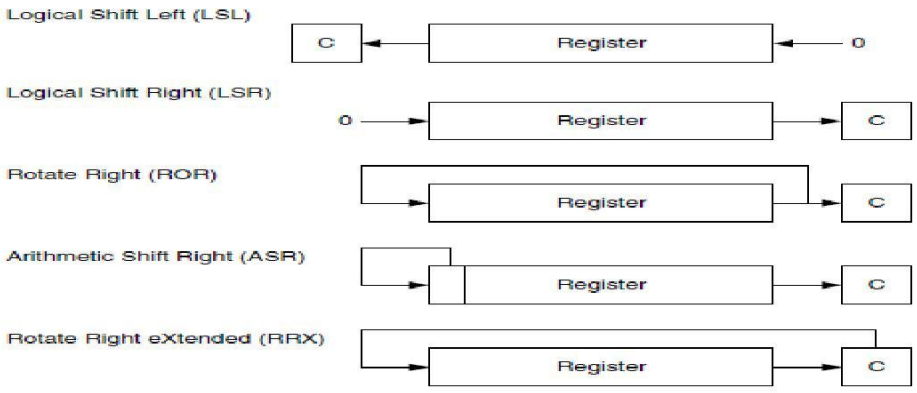
Sub:	ARM Microcontroller and Embedded Systems							Code:	15EC62
Date:	20/4/2019	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch/Section	ECE A ,B,C,D section

Answer Any FIVE FULL Questions

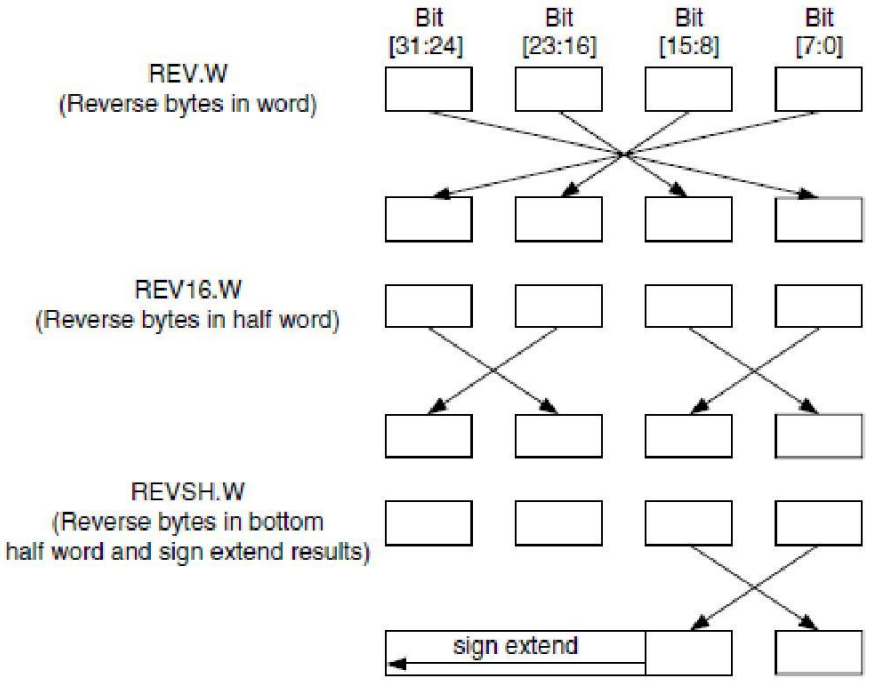
		Marks	OBE	
			CO	RBT
1 .	Explain shift and rotate operations available in ARM Cortex M3 instruction set. Why is there rotate right instruction but no rotate left instruction in cortex M3?	[10]	CO1	L2
2.	List and Explain the function of any four data processing and branch instructions in Cortex M3 with example	[10]	C01	L2
3.a)	Write a cortex M3 program to perform logical AND operations on two 32 bit data which is already stored in two memory locations. The memory locations are 8000 h and 9000 h. store the result in to 5000h	[5]	C01	L3
3b)	Perform unsigned division on two 32 bit datas which is to be stored in R0 and R1 register and store the result in R6 register	[5]	C01	L3
4.a)	Write a note on interfacing between assembly and C programming	[5]	C02	L1
4.b)	Explain the different types used to store the data in ARM cortex M3 processor	[5]	CO4	L4
5a.)	Write a C program to toggle an LED with small delay in Cortex M3	[5]	C02	L2
5b.)	Write a C program using ARM processor to display the seven segment LED	[5]	CO2	L2
6	Write Memory map and explain memory access attributes in ARM cortex M3	[10]	C02	L4
7.a)	Differentiate between Embedded system and General purpose systems	[5]	C03	L2
7.b)	List the applications of Embedded systems	[5]	C03	L1
8.a)	Explain the different types of memory used in Embedded systems	[5]	C03	L4
8.b)	List and Explain the classification of Embedded Systems	[5]	C03	L1

1 .	Explain shift and rotate operations available in ARM Cortex M3 instruction set. Why is there rotate right instruction but no rotate left instruction in cortex M3?	[10]
	4 shift and rotate operations each carry 2 marks-8 marks Explanation-2 marks The rotate right can be operated as left if we increase the rotation value.	

Instruction			Operation
ASR	Rd, Rn,#immed	; Rd = Rn » immed	Arithmetic shift right
ASRRd,	Rn	; Rd = Rd » Rn	
ASR.W	Rd, Rn, Rm	; Rd = Rn » Rm	
LSLRd,	Rn,#immed	; Rd = Rn « immed	Logical shift left
LSLRd,	Rn	; Rd = Rd « Rn	
LSL.W	Rd, Rn, Rm	; Rd = Rn « Rm	
LSRRd,	Rn,#immed	; Rd = Rn » immed	Logical shift right
LSRRd,	Rn	; Rd = Rd » Rn	
LSR.W	Rd, Rn, Rm	; Rd = Rn » Rm	
ROR	Rd, Rn	; Rd rot by Rn	Rotate right
ROR.W	Rd, Rn,#immed	; Rd = Rn rot by immed	
ROR.W	Rd, Rn, Rm	; Rd = Rn rot by Rm	
RRX.W	Rd, Rn	; {C, Rd} = {Rn, C}	Rotate right extended



Instruction	Operation
`REV   Rd, Rn ; Rd = rev(Rn)`	Reverse bytes in word
`REV16 Rd, Rn ; Rd = rev16(Rn)`	Reverse bytes in each half word
`REVSH Rd, Rn ; Rd = revsh(Rn)`	Reverse bytes in bottom half word and sign extend the result



2	List and Explain the function of any four data processing and branch instructions in Cortex M3 with example	[10]

4 data processing Instruction sets each carry 1.25 marks-5 marks

4 Branch Instruction sets each carry 1.25 marks-5 marks

Instruction	Function
B	Branch
B<cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR (this is actually a 32-bit instruction, but it is also available in Thumb in traditional ARM processors)
BLX	Branch with link and change state (BLX <reg> only)[1]
BX <reg>	Branch with exchange state
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

Instruction	Operation

ADD Rd, Rn, Rm ; Rd = Rn + Rm
ADD Rd, Rd, Rm ; Rd = Rd + Rm
ADD Rd, #immed ; Rd = Rd + #immed
ADD Rd, Rn, #immed ; Rd = Rn + #immed
	ADD operation

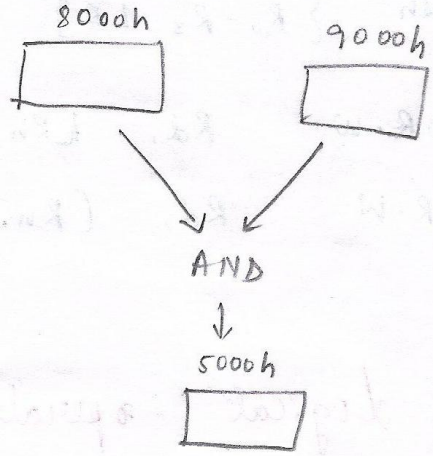
ADC Rd, Rn, Rm ; Rd = Rn + Rm + carry ADC Rd, Rd, Rm ; Rd = Rd + Rm + carry ADC Rd, #immed ; Rd = Rd + #immed + carry	ADD with carry
ADDW Rd, Rn,#immed ; Rd = Rn + #immed	ADD register with 12-bit immediate value

SUB Rd, Rn, Rm ; Rd = Rn - Rm
SUB Rd, #immed ; Rd = Rd - #immed
SUB Rd, Rn,#immed ; Rd = Rn - #immed
	SUBTRACT

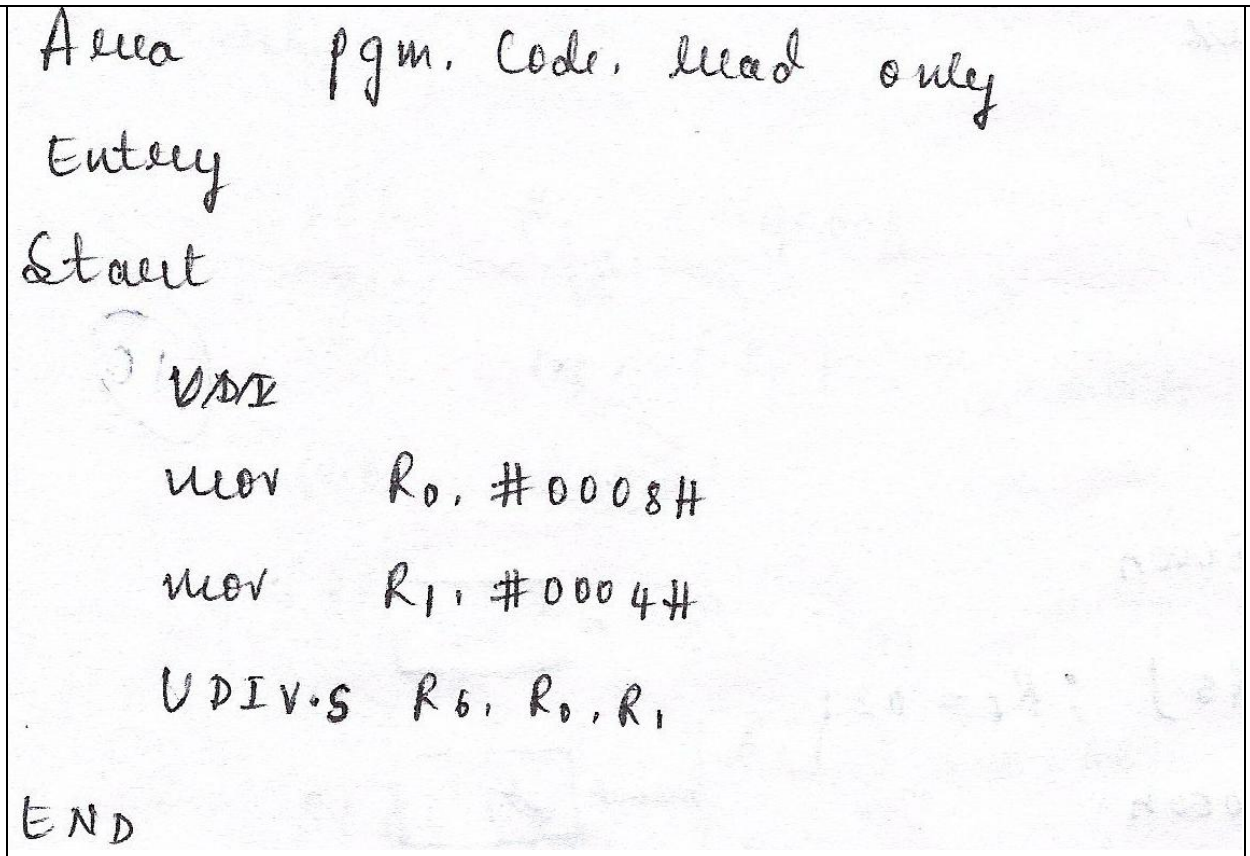
SBC Rd, Rm ; Rd = Rd - Rm - borrow SBC.W Rd, Rn, #immed ; Rd = Rn - #immed - borrow SBC.W Rd, Rn, Rm ; Rd = Rn - Rm - borrow	SUBTRACT with borrow (not carry)
RSB.W Rd, Rn, #immed ; Rd = #immed - Rn RSB.W Rd, Rn, Rm ; Rd = Rm - Rn	Reverse subtract
MUL Rd, Rm ; Rd = Rd * Rm MUL.W Rd, Rn, Rm ; Rd = Rn * Rm	Multiply
UDIV Rd, Rn, Rm ; Rd = Rn/Rm SDIV Rd, Rn, Rm ; Rd = Rn/Rm	Unsigned and signed divide

3.a)	Write a cortex M3 program to perform logical AND operations on two 32 bit data which is already stored in two memory locations. The memory locations are 8000 h and 9000 h. store the result in to 5000h. Instruction syntax-2 Marks	[5]

	Program-3 Marks	

	Area pgm, Code, read only
	entry
	Start
	ldr R0, =8000h
	mov R1, [R0]
	ldr R2, =9000h
	mov R3, [R2]
	AND R1, R3
	ldr R5, =5000h
	str R1, [R5]
	END
	 	

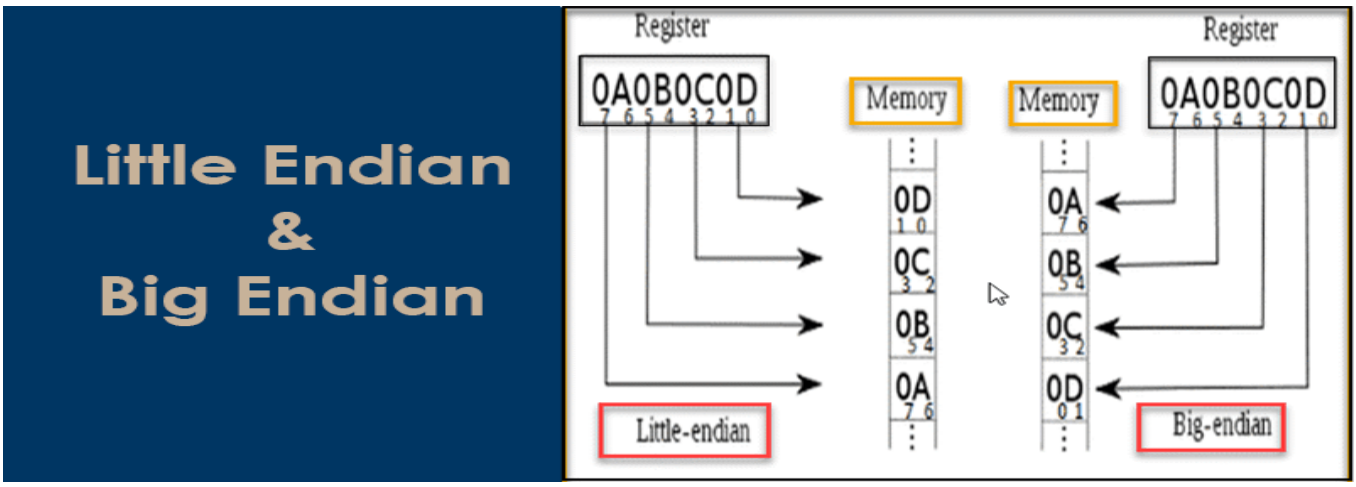
3.b)	Perform unsigned division on two 32 bit datas which is to be stored in R0 and R1 register and store the result in R6 register Instruction Syntax-2 Marks Program -3 Marks	[5]



4.a)	Write a note on interfacing between assembly and C programming Explanation-5 Marks	[5]

In various situations, assembly code and the C program interact. For example,
• When embedded assembly (or inline assembler, in the case of the GNU tool chain) is used in C program code
• When C program code calls a function or subroutine implemented in assembler in a separate file
• When an assembly program calls a C function or subroutine
In these cases, it is important to understand how parameters and return results are passed between the calling program and the function being called. The mechanisms of these interactions are specified in the *ARM Architecture Procedure Call Standard* [*AAPCS*, [Ref. 5]].
For simple cases, when a calling program needs to pass parameters to a subroutine or function, it will use registers R0–R3, where R0 is the first parameter, R1 is the second, and so on. Similarly, R0 is used for returning a value at the end of a function. R0–R3 and R12 can be changed by a function or subroutine whereas the contents of R4–R11 should be restored to the previous state before entering the function, usually handled by stack PUSH and stack POP.
To make them easier to understand, the examples in this book do not strictly follow AAPCS practices. If a C function is called by an assembly code, the effect of a possible register change to R0–R3 and R12 will need to be taken into account. If the contents of these registers are needed at a later stage, these registers might need to be saved on the stack and restored after the C function completes. Since the example codes mostly only call assembly functions or subroutines that affect a few registers or restore the register contents at the end, it's not necessary to save registers R0–R3 and R12.

4.b)	Explain the different types used to store the data in ARM cortex M3 processor Two types each carry 2.5 marks-5 Marks	[5]



Endianness

The endianness is the bytes order in which data stored in the memory of the computer and it also describes the order of byte transmission over a digital link. In memory data store in which order it depends on the endianness of the system, if the system is big-endian then the MSB byte store first (means at lower address) and if the system is the little endian then LSB byte store first (means at lower address).

Some examples of little-endian and big-endian system.

Little Endian	Big Endian
• Intel x86 and x86-64 series • Zilog Z80 (including Z180 and eZ80) • MOS Technology 6502	• Motorola 68000 series • Xilinx Microblaze, SuperH, • IBM z/Architecture, Atmel AVR32.

Big-endian

The most significant byte of data stored at the lowest memory address.

Address	Value
00	0x11
01	0x22
02	0x33
03	0x44

Little-endian

The least significant byte of data stored at the lowest memory address

Address	Value
00	0x44
01	0x33
02	0x22
03	0x11

5.a)	Write a C program to toggle an LED with small delay in Cortex M3 Steps-2 Marks Program-3 Marks	[5]

```
#define LED *((volatile unsigned int *)(0xDFFF000C))
int main (void)
{
int i; /* loop counter for delay function */
volatile int j; /* dummy volatile variable to prevent
C compiler from optimize the delay away */
while (1)
{
LED = 0x00; /* toogle LED */
for (i=0;i<10;i++) {j=0;} /* delay */
LED = 0x01; /* toogle LED */
for (i=0;i<10;i++) {j=0;} /* delay */
}
return 0;
}
```

5.b)	Write a C program using ARM processor to display the seven segment LED	[5]

Steps 2 Marks
Program 3 Marks

```
#include<lpc214x.h>          //Header file for LPC214x Series microcontrollers
void delay(int );            //Function declaration for delay
int i;                       //Variable declared as integer
unsigned int a[]={0xf3,0x12,0x163,0x133,0x192,0x1b1,0x1f1,0x13,0x1f3,0x1b3}; //integer array with
numbers for display
int main()
{
   IO0DIR=IO0DIR|0xffffffff;          //Sets direction as output for PORT 0 pins
   while(1)
   {
      for(i=0;i<=9;i++)
      {
         IO0SET=IO0SET|a[i];         //sets corresponding pins HIGH
         delay(9000);                //Calls delay function
         IO0CLR=IO0CLR|a[i];         //Sets corresponding pins LOW
      }
   }
   return 0;
}
void delay(int k)            //Function for making delay
```

```
{
   int i,j;
   for(i=0;i<k;i++)
   for(j=0;j<=1000;j++);
}
```

6)	Write Memory map and explain memory access attributes in ARM cortex M3 Memory map-4 Marks Memory Access Attributes-6 Marks	[10]

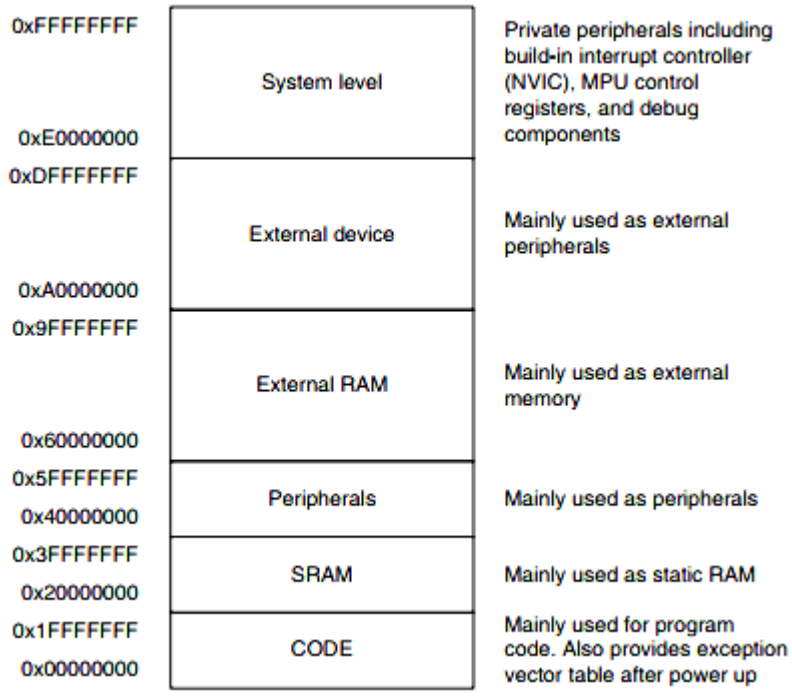
Memory map :5 marks

Memory access attributes:5 marks

The Memory Map:

The Cortex-M3 has a predefined memory map. This allows the built-in peripherals, such as the interrupt controller and the debug components, to be accessed by simple memory access instructions. Thus, most system features are accessible in C program code. The predefined memory map also allows the Cortex-M3 processor to be highly optimized for speed and ease of integration in system-on-a-chip (SoC) designs.

The Cortex-M3 design has an internal bus infrastructure optimized for this memory usage. In addition, the design allows these regions to be used differently. For example, data memory can still be put into the CODE region, and program code can be executed from an external Random Access Memory (RAM) region.



Memory Access Attributes

- The CORTEX M3 memory attributes (Method) are
  - Bufferable:
    - Write to memory can be carried out by a write buffer while the processor continues on next instruction execution. Example : UART
  - Cacheable:
    - Data obtained from memory read can be copied to a memory cache so that next time it is accessed the value can be obtained from the cache to speed up the program execution
  - Executable
    - Processor can execute code from this memory region
  - Sharable
    - Data in this memory region could be shared by multiple bus masters

Memory regions of attributes

- Code Memory region (0x00000000 to 0x1FFFFFFF)
  - Region is executable
  - Cache is write through i.e. can have data in this memory region
  - Data is transferred using the data bus interface
  - Write transfers is bufferable

CONTI.

- SRAM memory region *(0x20000000–0x3FFFFFFF)*
  - Meant for on-chip RAM
  - Executing the code is feasible.
  - Write transfer are bufferable.
- Peripheral Region *(0x40000000–0x5FFFFFFF)*
  - Reserved to use only by peripherals
  - Can't execute program here
- External RAM *(0x60000000–0x7FFFFFFF)*
  - Used for on or off chip memory
  - Can execute the program in this region
  - Write data are bufferable

Conti.

- External RAM (0x80000000–0x9FFFFFFF)
  - Reserved only for external devices
  - Memory is non-bufferable to access
  - Can't execute code in this region
- Same for 0xC0000000–0xDFFFFFFF
- System Region (0xE0000000–0xFFFFFFFF)
  - Reserved for the usage of private peripherals and vendor specific peripherals
  - For private peripherals access are non-bufferable
  - Vendor specific access are bufferable

Default Memory Access Permission

- Prevents the user programs to access some locations which are controlled by MPU

Memory Region	Address	Access in User Program
Vendor Specific	0xE0100000–0xFFFFFFFF	Full access
ROM table	0xE00FF000–0xE00FFFFF	Blocked ; if user tries to access a bus fault is generated
External Private Peripheral Bus	0xE0042000–0xE00FEFFF	Blocked; if user tries to access a bus fault is generated
ETM (Embedded Trace Macrocell)	0xE0041000–0xE0041FFF	Blocked; if user tries to access a bus fault is generated

7.a)	Differentiate between Embedded system and General purpose systems	[5]

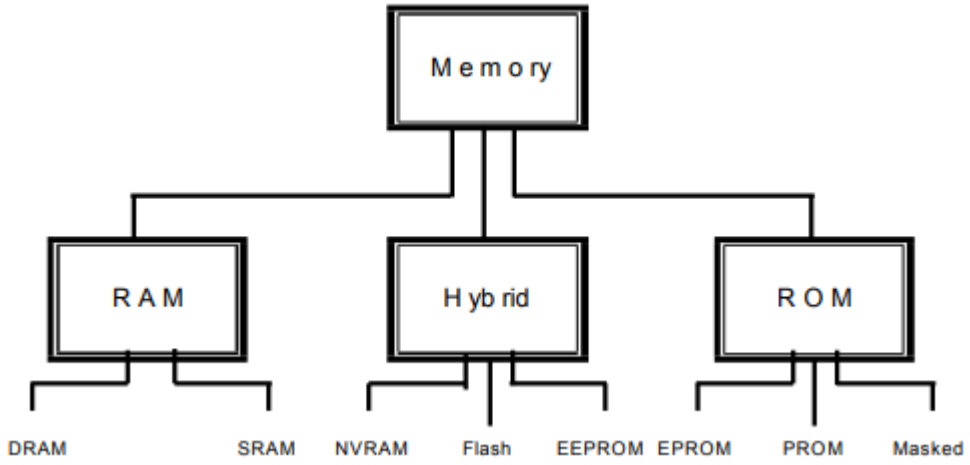
	Any Five differences each carry one mark-5 M	

Criteria	General Purpose Computer	Embedded system
Contents	It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Operating System	It contains general purpose operating system	It may or may not contain operating system.
Alterations	Applications are alterable by the user.	Applications are non-alterable by the user.
Key factor	Performance" is key factor.	Application specific requirements are key factors.
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications

7.b)	List the applications of Embedded Systems. Any Five applications each carry one mark-5 M	[5]
	APPLICATION OF EMBEDDED SYSTEM The application areas and the products in the embedded domain are countless. 1. Consumer Electronics: Camcorders, Cameras. 2. Household appliances: Washing machine, Refrigerator. 3. Automotive industry: Anti-lock breaking system(ABS), engine control. 4. Home automation & security systems: Air conditioners, sprinklers, fire alarms. 5. Telecom: Cellular phones, telephone switches. 6. Computer peripherals: Printers, scanners.	

	7. Computer networking systems: Network routers and switches. 8. Healthcare: EEG, ECG machines. 9. Banking & Retail: Automatic teller machines, point of sales. 10. Card Readers: Barcode, smart card readers.	

8.a)	Explain the different types of memory used in Embedded systems Primary memory-2.5 Marks Secondary memory-2.5 Marks	[5]
	MEMORIES There are different types of memories available to be used in computers as well as embedded system. This chapter guides the reader through the different types of memories that are available and can be used and tries to explain their differences in simple words. TYPES OF MEMORY There are three main types of memories, they are a) **RAM** (Random Access Memory) It is read write memory. · Data at any memory location can be read or written. · It is volatile memory, i.e. retains the contents as long as electricity is supplied. · Data access to RAM is very fast b) **ROM** (Read Only Memory) It is read only memory. · Data at any memory location can be only read. · It is non-volatile memory, i.e. the contents are retained even after electricity is switched off and available after it is switched on. Data access to ROM is slow compared to RAM. c) **HYBRID** It is combination of RAM as well as ROM · It has certain features of RAM and some of ROM · Like RAM the contents to hybrid memory can be read· and written Like ROM the contents of hybrid memory are non volatile The following figure gives a classification of different types of memory	



TYPES OF RAM
There are 2 important memory device in the RAM family.
a) SRAM (Static RAM)
b) DRAM (Dynamic RAM)

SRAM (Static RAM)

- 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.

DRAM (Dynamic RAM)

- DRAM has extremely short Data lifetime(usually less than a quarter of second). This is true even when power is applied constantly.
- b) A DRAM controller is used to make DRAM behave more like SRAM.
- c) The DRAM controller periodically refreshes the data stored in the DRAM. By refreshing the data several times a second, the DRAM controller keeps the contents of memory alive for a long time.

TYPES OF ROM
There are three types of ROM described as follows:
Masked ROM
a. These are hardwired memory devices found on system. b. It contains pre-programmed set of instruction and data and it cannot be modified or appended in any way.
b. (it is just like an Audio CD that contains songs pre-written on it and does not allow to write any other data)
c. The main advantage of masked ROM is low cost of production.
PROM (PROGRAMMABLE ROM)
a) 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 ROM.
b) In the un-programmed state the data is entirely made up of 1's. c) PROMs are also known as one-time-programmable (OTP) device because any data can be written on it only once. If the data on the chip has some error and needs to be modified this memory chip has

to be discarded and the modified data has to be written to another new PROM.

EPROM (ERASABLE-AND-PROGRAMABLE ROM)

a) It is same as PROM and is programmed in same manner as a PROM.
b) It can be erased and reprogrammed repeatedly as the name suggests.
c) The erase operation in case of an EPROM is performed by exposing the chip to a source of ultraviolet light.
d) The reprogramming ability makes EPROM as essential part of software development and testing process.

TYPES OF HYBRID MEMORY

There are three types of Hybrid memory devices: EEPROMs
a. EEPROMs stand for Electrically Erasable and Programmable ROM.
b. It is same as EPROM, but the erase operation is performed electrically.
c. Any byte in EEPROM can be erased and rewritten as desired

Flash

a. Flash memory is the most recent advancement in memory technology.
b. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable.
c. Flash is much more popular than EEPROM and is rapidly displacing many of the ROM devices.
d. Flash devices can be erased only one sector at a time, not byte by byte.

NVRAM

a. NVRAM is usually just a SRAM with battery backup.
b. When power is turned on, the NVRAM operates just like any other SRAM but when power is off, the NVRAM draws enough electrical power from the battery to retain its content.
c. NVRAM is fairly common in embedded systems.
d. It is more expensive than SRAM.

DIRECT MEMORY ACCESS (DMA)

DMA is a technique for transferring blocks of data directly between two hardware devices. In the absence of DMA the processor must read the data from one device and write it to the other one byte or word at a time. DMA Absence Disadvantage: If the amount of data to be transferred is large or frequency of transfer is high the rest of the software might never get a chance to run.

DMA Presence Advantage: The DMA Controller performs entire transfer with little help from the Processor. Working of DMA The Processor provides the DMA Controller with source and destination address & total number of bytes of the block of data which needs transfer. After copying each byte each address is incremented & remaining bytes are reduced by one. When number of bytes reaches zeros the block transfer ends & DMA Controller sends an Interrupt to Processor.

8.b)	List and Explain the classification of Embedded Systems	[5]
	Based upon the performance of functional requirements-2.5 Marks Based upon the performance of Microcontroller-2.5 Marks	

CLASSIFICATION OF EMBEDDED SYSTEMS

The classification of embedded system is based on following criteria's:

- On generation
- On complexity & performance
- On deterministic behavior
- On triggering
- **On generation**:

1. First generation (1G):

- Built around 8bit microprocessor & microcontroller.
- Simple in hardware circuit & firmware developed.
- Examples: Digital telephone keypads.

2. Second generation (2G):

- Built around 16-bit µp & 8-bit µc.
- They are more complex & powerful than 1G µp & µc.
- Examples: SCADA systems

3. Third generation (3G):

- Built around 32-bit µp· & 16-bit µc.
- Concepts like Digital Signal Processors (DSPs), Application Specific Integrated Circuits(ASICs) evolved. Examples: Robotics, Media, etc.

4. Fourth generation:

- Built around 64-bit µp & 32-bit µc.
- The concept of System on Chips (SoC), Multicore Processors evolved.
- Highly complex & very powerful. Examples: Smart Phones.
- **On complexity & performance:**

1. Small-scale:

- Simple in application need
- Performance not time-critical.

- Built around low performance& low cost 8 or 16 bit μp/μc. Example: an electronic toy

2. Medium-scale:

- Slightly complex in hardware & firmware requirement.
- Built around medium performance & low cost 16 or 32 bit μp/μc.
- Usually contain operating system.
- Examples: Industrial machines.

3. Large-scale:

- Highly complex hardware & firmware.
- Built around 32 or 64 bit RISC μp/μc or PLDs or Multicore-Processors.
- Response is time-critical.
- Examples: Mission critical applications.
- **On deterministic behavior:**
- This classification is applicable for “Real Time” systems.
- The task execution behavior for an embedded system may be deterministic or nondeterministic.
- Based on execution behavior Real Time embedded systems are divided into Hard and Soft.
- **On triggering**
- Embedded systems which are “Reactive” in nature canbe based on triggering.
- Reactive systems can be:
- Event triggered
- Time triggered