CMR INSTITUTE OF TECHNOLOGY

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

Internal Assesment Test - II

Sub:	EMBEDDED SYSTEM DESIGN						Code:	10EC74
Date:	03 / 11 / 2016	Duration:	90 mins	Max Marks:	50	Sem: VII	Branch:	ECE- A&B

Answer Any FIVE FULL Questions

		Marks	OBE CO	OBE RBT
1 (a)	What is Context switching? Describe the sequence of steps that are necessary to handle an occurrence of an interrupt?	[4]	CO5	L2
(b)	A microprocessor based system has 8 address lines and 8 data lines, requires an SRAM system that can store upto 4K 16 bit words. But largest available memory device is 1K x 18. Design memory system that supports above said data.	[6]	CO3	L3
2 (a)	Write the inside and outside diagram for DRAM along with read & write operation. Also explain refresh operation	[6]	CO3	L3
(b)	Define: a) Bandwidth b) Cycle time c) Latency d) Page	[4]	CO3	L3
3 (a)	Explain Direct mapping cache implementation. Write its advantages and disadvantages	[6]	CO3	L3
(b)	Explain: a) Swapping b) Overlays	[4]	CO3	L3
4 (a)	Explain the different types of Stacks used in the memory management	[4]	CO3	L3
(b)	Discuss the significance of Duplicate hardware context with a suitable diagram	[6]	CO3	L3
5 (a)	List the different types of Operating system services and explain them briefly. Also discuss the significance of the architecture of an Operating system	[8]	CO5	L2
(b)	Define: a) Persistence b) Real-time operating system	[2]	CO5	L2
6 (a)	Briefly explain the state transitions in the Task control block module system	[4]	CO5	L2
(b)	With a suitable schematic & program, explain the Task Control Block	[6]	CO5	L2
7 (a)	Write the differences between: i) Light weight thread and Heavy weight thread ii) Task and Thread iii) Program and Process iv) Foreground and Background task	[8]	CO5	L2
(b)	What is an Interrupt Service Routine?	[2]	CO5	L2

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Introduction to Embedded Systems and description of the design flow along with development process of an embedded system	3	-	2	-	-	-	-	-	-	-	-	--
CO2:	Discuss the hardware side of an Embedded system and associate it with time constraints using Finite State Machines.	3	2	3	1	-	-	-	-	-	-	-	-
CO3:	Classification of Embedded memory types, it's interfaces and their description in detail.	-	-	3	2	1	-	-	-	-	-	-	1
CO4:	Analyze the Embedded system design and devlopment process in detail and compare their lifecycle models.	3	-	3	1	-	-	-	-	-	-	-	-
CO5:	Select Operating systems and Real Time Kernel for Embedded Systems and explain Task and Task Control Blocks.	-	-	3	3	-	-	-	-	-	-	-	-
CO6:	Integrate Performance Analysis and Optimization techniques for Embedded applications with necessary trade offs.	3	3	3	-	-	-	-	-	-	-	-	-

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

IAT - 2 Solution

(2 marks)

1 a) A task's context comprises the important information about the state of the task such as the values of any variables, the value of the PC and so forth.

Each time a running task is stopped- preempted or blocked- and the CPU is given to another task that is ready, a switch to new context is executed.

During context switch, the state of the currently active task must be saved.

If the task that is scheduled next had been running previously, its state is restored and it continues where it had let off. Otherwise, it starts from its initial state.

Sequence of steps that are necessary to handle an occurrence of an interrupt: (2 marks)

1. Foreground code is running, Interrupts are enabled.
2. Interrupt sends an interrupt request to CPU.
3. CPU begins the interrupt response:

    a. Saves the current PC
    b. Saves some status (depending upon the CPU)
    c. Jump to the correct ISR for this request.
    d. ISR code saves any register and flags that it will modify
    e. Services the interrupt
    f. ISR code restores saved register and flags
    g. ISR executes return from interrupt instruction or sequence.

4. Return automatically restores saved status.
5. Recovers the PC
6. Foreground code continues to run from the point it responded to the interrupt.

1. The requirement specifies support for 4K 16-bit words. (1 mark)

The available memory chips only support 1K 8-bit words. We can use two 1K 8-bit memory chips to hold 1K 16- bit words.

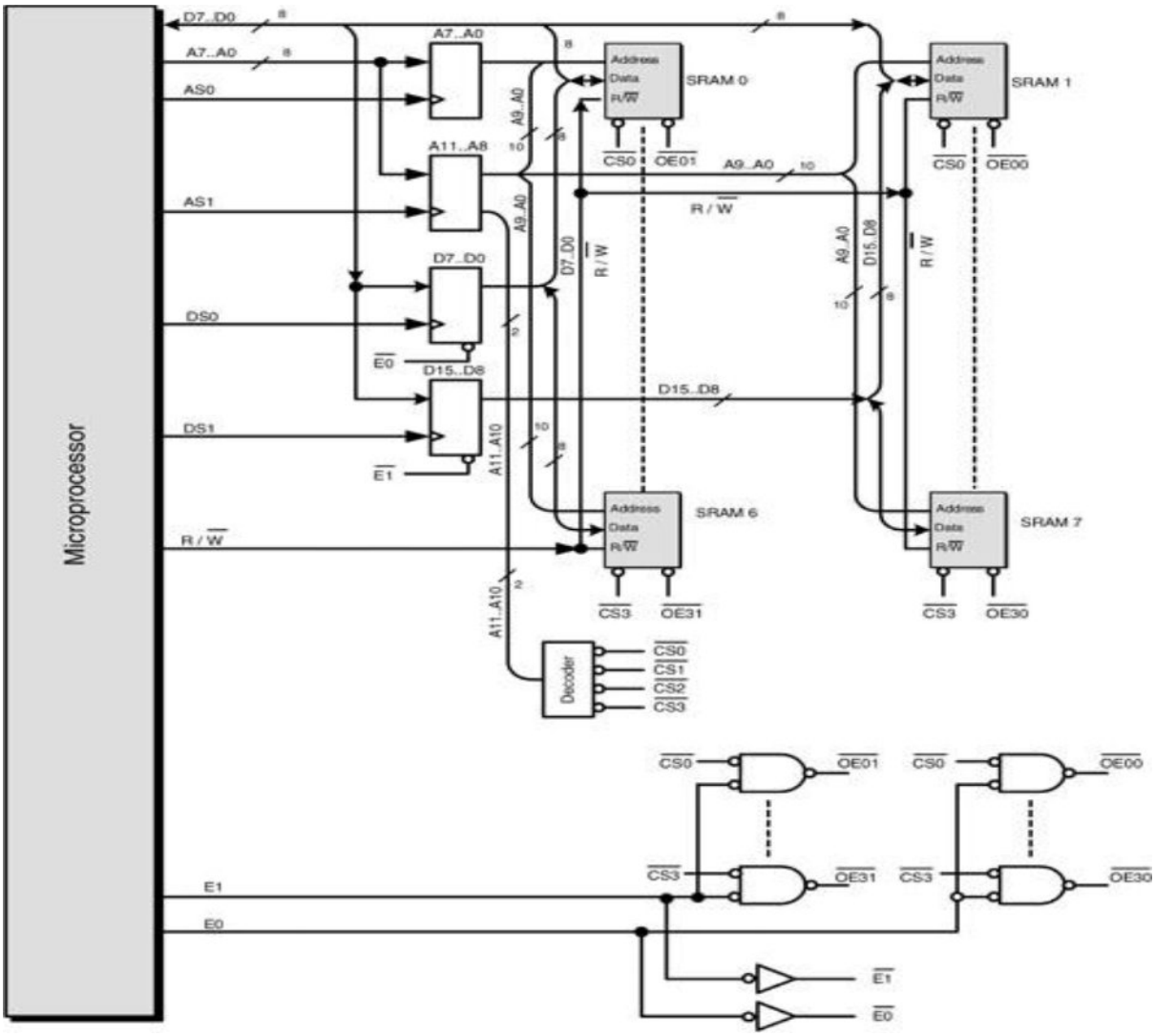
By duplicating such a configuration four times we will have sufficient storage for the 4K 16-bit words.

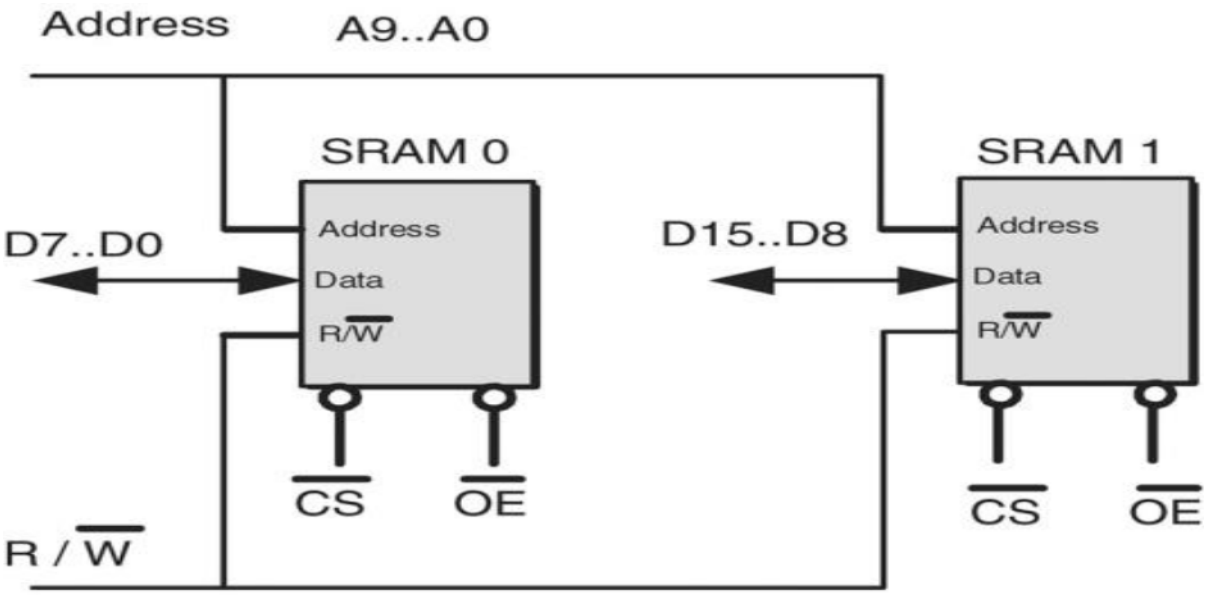
10 address bits are required since 2^{10}= 1024 combinations or 1K.

Two additional bits to activate Chip select (CS) control input.

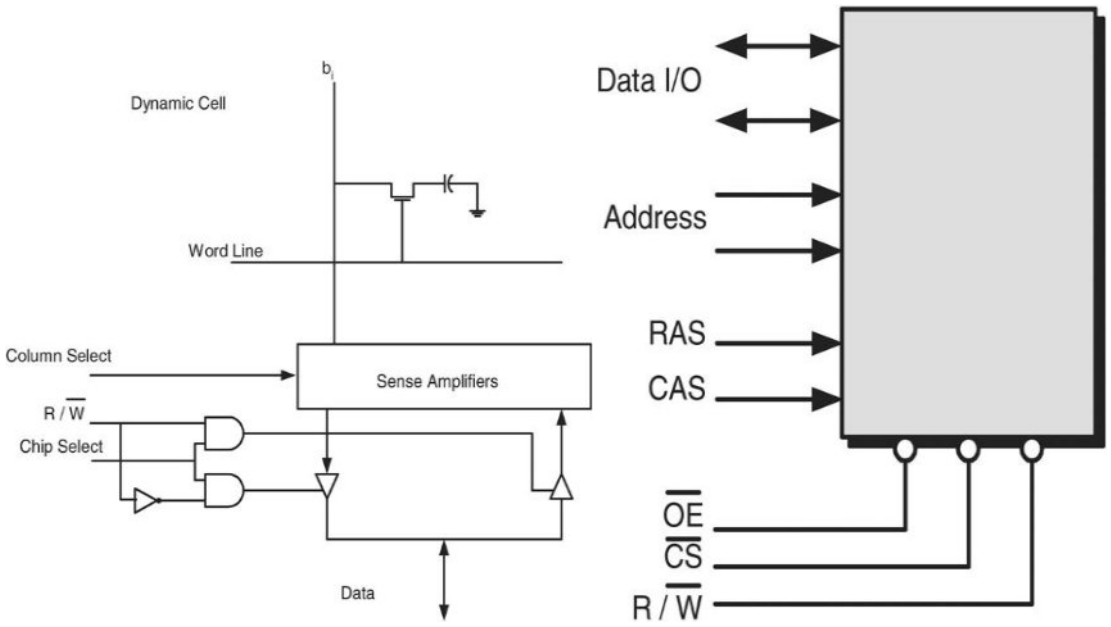
Output enable (OE) must be kept at logical 1 during a write and at logical 0 during read cycle.

(5marks)

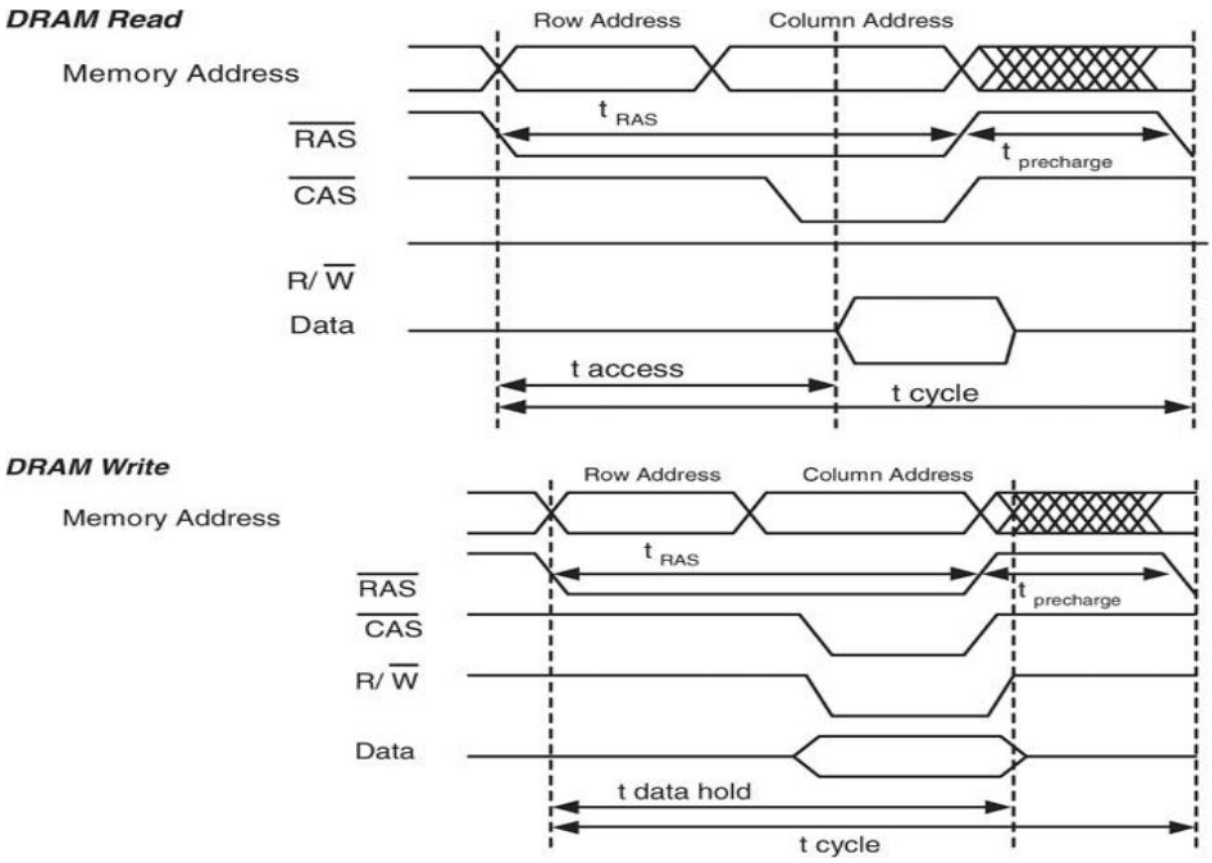

Microprocessor
D7..D0
A7..A0
AS0
AS1
DS0
DS1
R / W
E1
E0
A11..A8
A9..A0
D15..D8
A11..A10
Address
Data
R/W
SRAM 0
SRAM 1
SRAM 6
SRAM 7
CS0
CS3
OE01
OE00
OE31
OE30
Decoder
CS1
CS2


Address
A9..A0
SRAM 0
SRAM 1
D7..D0
D15..D8
Address
Data
R/W
CS
OE
R / W

2 a)



Inside and outside diagrams of a DRAM (2 marks)



(2 Marks)

Refresh operation: (2 Marks)

Since DRAM memory cells are capacitors, the charge they contain can leak away over time. If the charge is lost, so is the data. To prevent this from happening, DRAMs must be refreshed -- that is, the charge on the individual memory cells must be restored periodically. The frequency with which refresh must occur depends on the silicon technology used to manufacture the memory chip and the design of the memory cell itself.

Reading or writing a memory cell has the effect of refreshing the selected cell. Unfortunately, not all cells are read or written within the time limitations. Thus each cell in the array must be accessed and restored during the refresh interval.

2 b) (1 mark each)

a) **Bandwidth** : Memory bandwidth is a measure of the word transmission rate to and from memory via the memory I/O bus. When the data is read from memory, the pattern on each data line will be a square wave. The highest frequency of that square wave is the memory bandwidth.

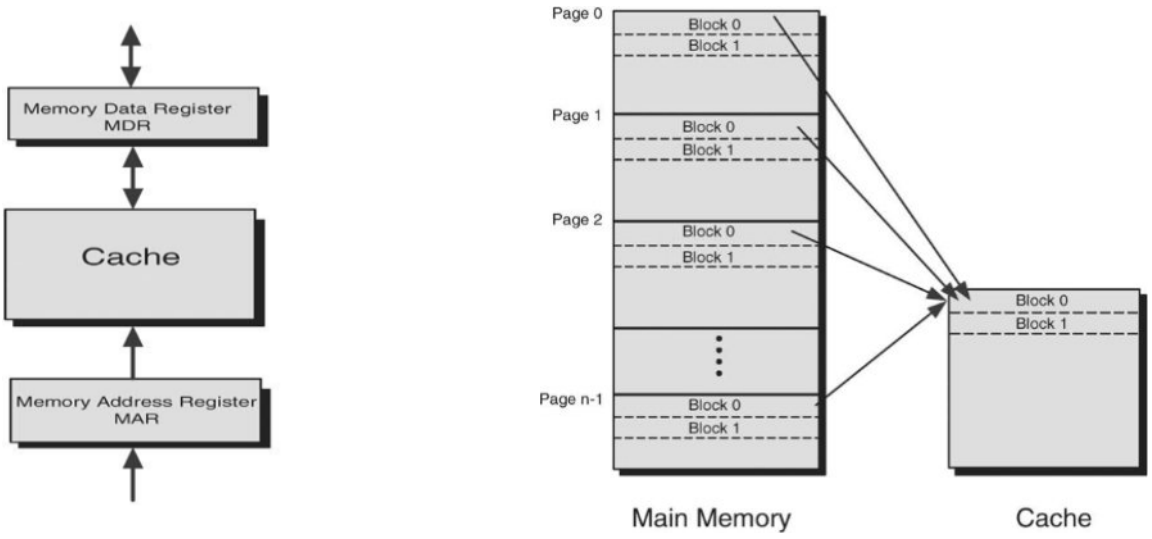
b) **Cycle time**: It is defined as the "time interval from the start of one read or write operation until the start of the next".

c) **Latency** : It is defined as the amount of time required to access the first of a sequence of words. Latency measures the time necessary to compute the address of that sequence and then locate its first block of words in memory.

d) **Page** : It is a logical view placed on larger collections of words in memory. Pages are generally comprised of blocks; the size of a page can be given in words or in blocks.

3 a)

Direct Cache mapping: (2 marks)



Cache and main memory will store 32-bit words, cache holds 64K words. (2 marks)

The cache will be organized as 128 0.5K word blocks.

Memory addresses = 32 bits

Main memory will be organized as 128M words; 2K pages, each holds 128 blocks.

Each location in RAM has one specific place in cache where the data will be held.

Consider the cache to be like an array. Part of the address is used as index into the cache to identify where the data will be held.

Since a data block from RAM can only be in one specific line in the cache, it must always replace the one block that was already there. There is no need for a replacement algorithm.

Direct cache mapping algorithm is one of the simpler replacement algorithm where a complete block of required data is brought into the cache. Mapping here is many-one.

The main memory page size is set equal to the cache size, therefore each page will contain the corresponding number of blocks. Thus the main memory will contain

$$\text{size}_{MM} = \text{mod size}_{cache} \quad \text{pages.}$$

When a block is brought into the cache from the main memory, it is placed into the corresponding numbered block in the cache. Thus, main memory *block0* will always be placed into the cache *block0* slot, a main memory *block1* will always be placed into the cache *block1* and so on.

Mapping process: (2 marks)

The system first checks for the corresponding block number in the cache. Use tag to see if a desired word is in cache – If there is no match, it is cache miss, the block containing the required word must first be read from the memory.

If there is a match, then the block offset is referred to fetch the desired word from the cache.

Advantage – Simplest replacement algorithm. The mapping scheme is easy to implement. You don't have to simulataneously match tags with all slots.

Disadvantage – Not flexible. There is contention problem even when cache is not full i.e each block of main memory maps to a fixed location in the cache; therefore, if two different blocks map to the same location in cache and they are continually referenced, the two blocks will be continually swapped in and out (known as thrashing). This would degrade the performance of the system.

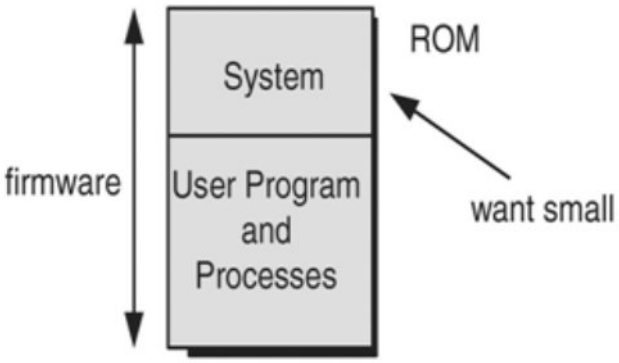
3 b) (4= 2 marks each)

Swapping : Swapping is a simple memory/process management technique used by the operating system(os) to increase the utilization of the processor by moving some blocked process from the main memory to the secondary memory (hard disk); thus forming a queue of temporarily suspended process and the execution continues with the newly arrived. It is a simplest method for accommodating multiple programs in memory. Such scheme assumes that only a single-user program is resident in memory at a time.

Program consists of several tasks. The first task is executing when a second must be run. The first task is suspended and swapped into a secondary storage device. Secondary storage device could also be a ROM since we are dealing with firmware in Embedded systems.

In the swapping the processes those are on waiting state and those are on suspend or temporary suspend will be stored from out side the memory locations so that the speed of process will be high.

Such a scheme can be deadly in time critical systems. One should ensure that the task/Program execution time is long compared to the swap time.

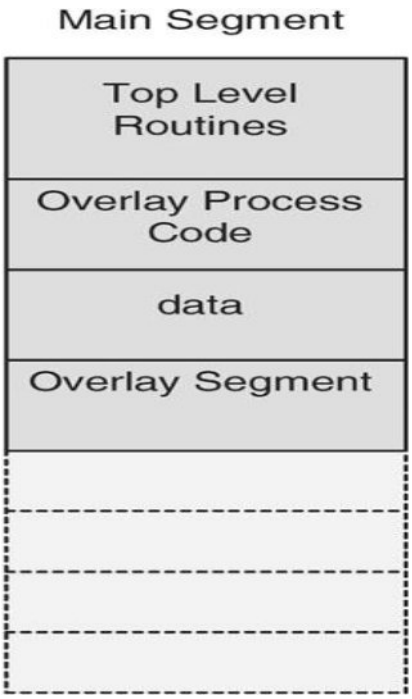


Overlays: To enable a process to be larger than the amount of memory allocated to it, we can use overlays. The program is segmented into a number of sections called overlays.

Segment program into one main section and a set of overlays. The main section consists of :

i. Top level routine
ii. Code to perform overlay process
iii. Data segment for shared data
iv. Overlay segment

When the overlay process is executed, a new overlay segment replaces the current one in main memory. Choose segmentation carefully to prevent ***thrashing.***



4 a) Different types of stacks: (1 mark each and 1 mark for diagram= 4)

i) Run time
ii) Application Stack
iii) Multiprocessing

Run time Stack

It is a defensive design. Runtime stack is under system control and it may be shared by other processes or threads. There is usually no dynamic allocation. Size of the stack is fixed, thus at runtime, one must ensure that not too many stack frames are pushed on to the stack. Otherwise there is a potential overflow leading to system crash. Single runtime stack can work in foreground/background mode.

Application Stacks

The single stack model can be extended by incorporating several additional stacks. The design consists of runtime stack as well as multiple application stacks to simplify the management of multiple tasks in a pre-emptive environment. The runtime stack consists of a pointer to the application stack associated with the initial or pre-empted thread or task, so the pre-empting task works with the new stack.

The save and restore interface functions must be modified to store/restore with respect to the current context as the runtime stack is unwound. Such a scheme provides a fast context switch. Refer to fig 11.21 in the text book

Multiprocessing stacks

Multiprocessing refers to working with multiple processes rather than multiple processors. When a task is started, it is allocated its own space along with other resources. This stack is managed by owner processes. It is allocated from a heap when the process is created and returned back to the heap when the process exits.

4 b)

Duplicate hardware context: (diagram 4 marks & explanation= 2 marks)

Refer to fig 11.18 & 11.19 from text

Architecture of most of the embedded applications is built around multiple tasks/threads. If the design supports the ability to pre-empt or block a running task/thread and initiate the other, there must be a switch to a new context. During context switch the process of saving the existing context, switching to the new context and restoring the old one consumes a significant amount of time. This time required to affect the switch can be critical to the success or failure of the real time systems. One way to accomplish such saving of information is by the use of duplicate hardware context where some microprocessors provide some hardware support for the context switch by substantially increasing the number of available general purpose registers.

At software level, several contexts will be defined and a subset of registers will be allocated to each context. Eg: If there are **4** different contexts and **64** general purpose registers, each context with **16** general purpose registers can be defined. When a switch is to occur, rather than saving the contents of current set of registers, the system simply switches to a new hardware context and the time required for the switching between the contexts is saved.

5 a) (1 mark each= 4)

Operating system services:

An operating system must provide or support 3 specific functions:

- Schedule task execution.
- Dispatch a task to run.
- Ensure communication and synchronization among tasks.

In an embedded operating system, such functions are captured in the following types of services.

1. **Process or task management** –

i. Creation and deletion of user and system processes as well as the suspension and resumption of such processes.

ii. Management of interprocess communication and of deadlocks.

2. **Memory management** –

i. Tracking and control of which tasks are loaded into memory.

ii. Monitoring the usage of different parts of memory.

iii. Administering dynamic memory and managing caching schemes.

3. **I/O system management** – Management of the interaction of the embedded application with variety of external devices and users or tasks in the application. Operating systems of a complex systems and well designed systems use device drivers and application programmer's interface (API) for this purpose.

4. **File system management** –

i. This responsibility is directed towards the creation, deletion and management of files and directories considering the available memory limitations.

ii. Backup of data that is to be saved as well as emergency backup either as power is failing or as some other catastrophic event is occurring to the system.

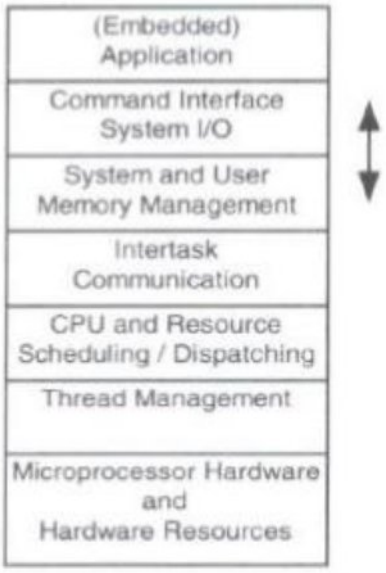
5. **System protection** – Ensuring the protection of data and resources in the context of concurrent processes is an important and essential duty of operating system.

6. **Networking** – Operating system must take on the responsibility of managing distributed intrasystem communication and the remote scheduling of tasks.

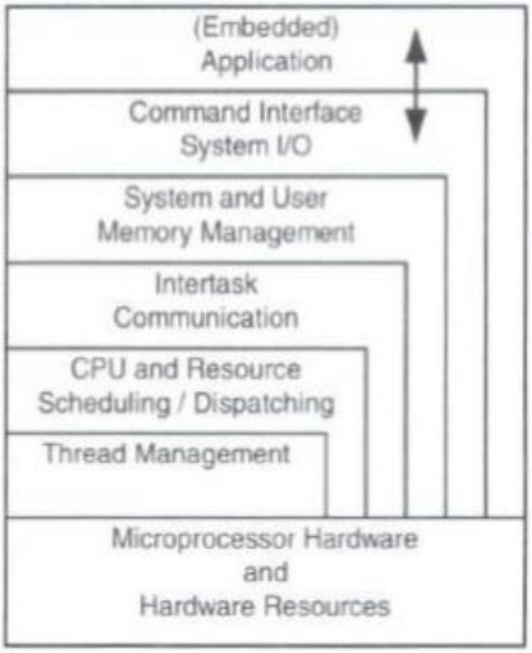
7. **Command interpretation** – Commands and directives that come into the system must be parsed, checked for grammatical accuracy and directed to the target task as embedded applications provides support provisions for user interaction via variety of software drivers and interaction with the hardware I/O devices.

Significance of OS architectures: (2 mark each)

OS are designed and implemented as a hierarchy of ***virtual machines.***



The hierarchy in the above model is designed in such a way that each layer uses the functions/operations and services of lower layers. The primary advantage here is increased modularity. Organized like the onion model, the only real machine that the various pieces of functionality within the OS see is the underlying physical microprocessor.



In the high level architecture, the higher level layers have access to lower levels through system calls and hardware instructions.

Existing calling interface between the levels is retained while providing the access to the below physical hardware.

Interface is made to appear as if it is a machine executing a set of instructions as defined by the API.

The idea can be logically extended to create the illusion that the tasks are running on its own machine, where each level is called a virtual machine.

With such an approach, one could run entirely different OS as an application within primary OS.

5 b) (1mark each= 2)

Persistence: It is defined as the duration of the time when a task enters the system until it terminates.

Real time OS : It is a special type of OS which ensures that rigid time constraints can be met. If such constraints are not met, the performance of the application is inaccurate or compromised in some way.

6 a) Refer fig 11.4 from the text

State transitions in the Task control block system: (3 mark= diagram and 1 mark= explanation)

In the case of static allocation, when a task control block is created, memory is allocated at system generation time and placed in **Dormant or unused** state.

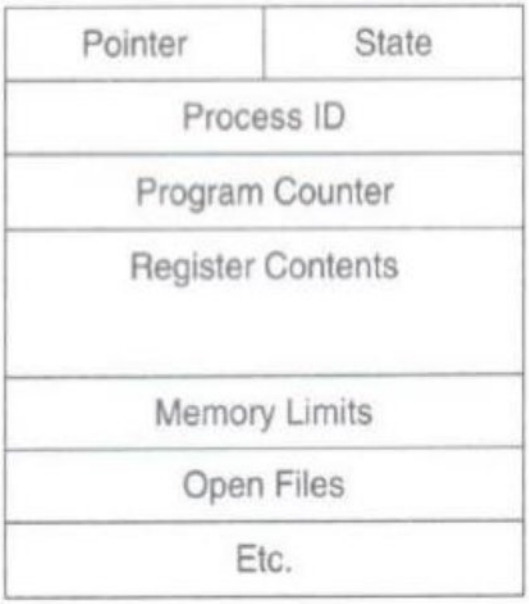
When the task is initiated, a TCB is allocated to it and the appropriate information is entered. It is then placed in the **Ready state** by the scheduler.

From the ready state it is moved to the **execute state** by the dispatcher.

When a task **terminates**, the associated TCB is returned to the **dormant state**.

With dynamic allocation, when a task is terminated, the TCB is returned to heap storage.

6 b) Refer fig 11.11 from text (2 mark= diagram, 2 mark= program, 2 mark= Explanation)



In a task-based approach, each process is represented in the OS by a data structure called a Task control block. It contains all the information about the task, such as

- Pointer
- Process ID and state
- Program counter
- CPU registers
- Scheduling information
- Memory management information
- I/O status information

i. The creation, deletion and allocation of TCB's is taken care by the memory management unit of OS. When a task enters a system, it will typically be placed in a queue called job queue or entry queue, which is implemented using linked list. Thus, the last entries in the TCB contains pointers to the preceeding and succeeding TCBs in the queue.
ii. In C, the TCB is implemented as a struct containing pointers to all relevant information as seen in the code.
iii. The first entry in the TCB is a pointer to a function- **taskPtr**, that function embodies the functionality associated with the task.
iv. The function's parameter list comprises the single argument of type ***void**.
v. **Struct** is a means used to pass data into the task.
vi. The type information associated with the data struct is removed by referencing it through a **void*** pointer.
vii. Each task will have its own stack, the third entry in the TCB is pointer to the stack.
viii. The fourth entry gives the priority for the task
ix. The fifth and sixth entries are the pointers used to link the TCB to the next and previous TCBs in any of the mentioned queues.

7 a) (2 mark each = 8)

i) **Light weight threads** : Subset of resources is called light weight thread. They share the same address space. Communication between the threads is easy and simple. The resources are conserved. They are not always thread safe. Does not rely on native code.

Heavy weight threads : Process itself can be referred to as a heavy weight thread. They contain their own address space. Communication between these processes would involve additional communications mechanisms. Resources are not conserved. They are thread safe. They rely on the native code.

ii) **Task** : Task is characterised by a collection of resources that are utilized to execute a program. It does not share the memory space. Switching between the task is more expensive. It is a heavyweight operation

Threads : Small subset of these resources that are utilized to execute a program is called a thread. They run on the same memory space. Switching between the threads is less expensive. It is a light weight operation

iii)**Program** : It is a passive(static) entity. It is an algorithm expressed in a suitable notation. Program is independent. Stored on a disk or a secondary memory.

Process : It is a active(dynamic) entity. It is a program in execution. Process is dependant on program. Stored on a primary memory.

iv) **Foreground task**: It is a process that happens on-screen. They have the access to the terminal standard I/Os. It is given a lesser priority. Controlled by user.

Background task : It runs off- screen. They run with little or no interaction at all. It is given high priority. Controlled by operating systems.

7b) **Interrupt service routine**: It is a code written to handle an interrupt and is located in memory. An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt. ISRs examine an interrupt and determine how to handle it. ISRs handle the interrupt, and then return a logical interrupt value.

ISR is also called device driver in case of the devices and called exception or signal or trap handler in case of software interrupts. ISRs often need to happen quickly as the hardware can have a limited buffer, which will be overwritten by new data if it's not pulled off quickly enough. (2 mark)