Embedded Software in a system and Programming Concepts

Embedded software in a system

- Embedded Software as ROM Image: Generally for embedded computing the software is placed in Ram before it runs.
- The embedded system needs a specific software or an application consists of large instruction sets and data which are generally placed in ROM or Flash memory of the system.
- Each code or datum is available only in bits and bytes format. The figure shows the ROM Image format used in Embedded System.

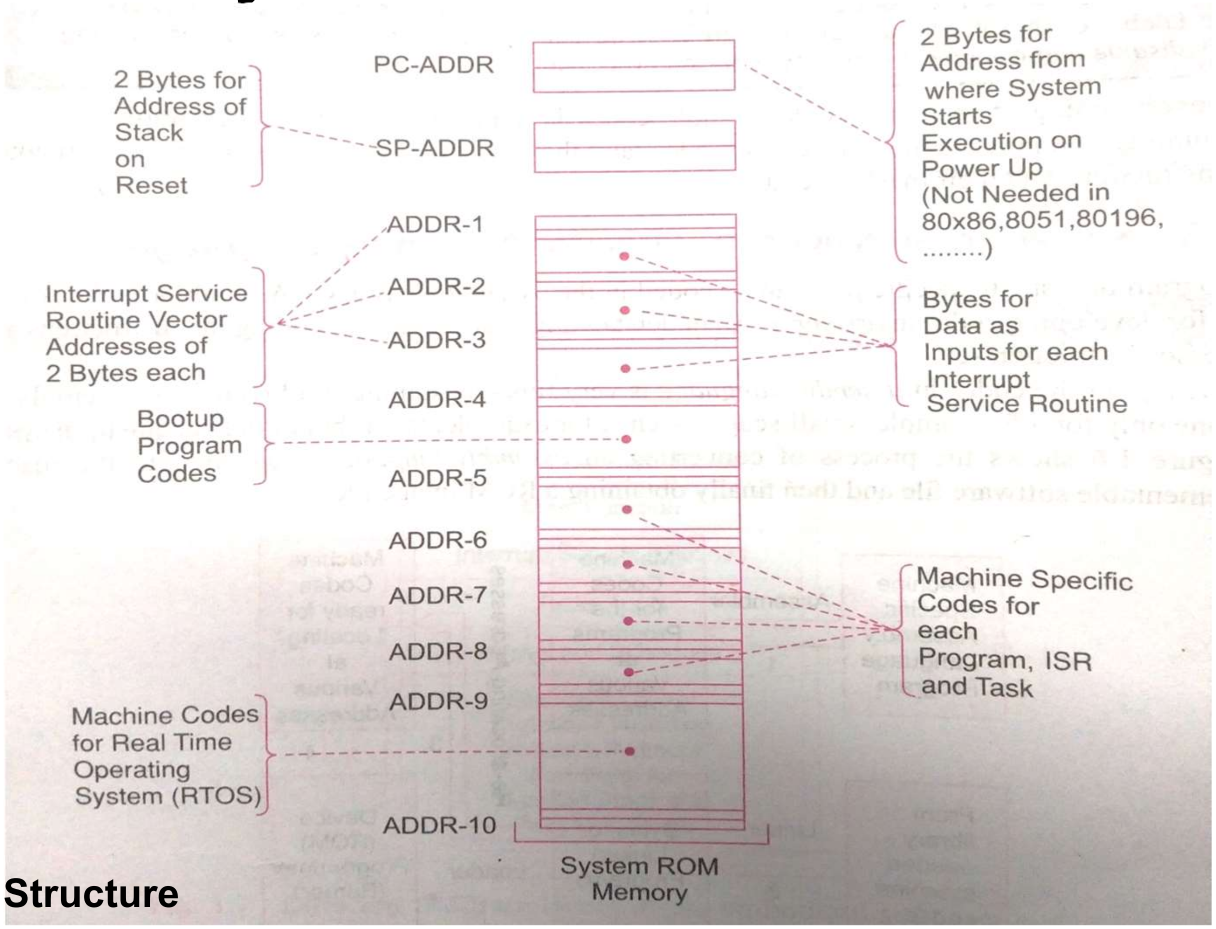

PC-ADDR
SP-ADDR
ADDR-1
ADDR-2
ADDR-3
ADDR-4
ADDR-5
ADDR-6
ADDR-7
ADDR-8
ADDR-9
ADDR-10
2 Bytes for Address of Stack on Reset
Interrupt Service Routine Vector Addresses of 2 Bytes each
Bootup Program Codes
Machine Codes for Real Time Operating System (RTOS)
2 Bytes for Address from where System Starts Execution on Power Up (Not Needed in 80x86,8051,80196, ...)
Bytes for Data as Inputs for each Interrupt Service Routine
Machine Specific Codes for each Program, ISR and Task
System ROM Memory

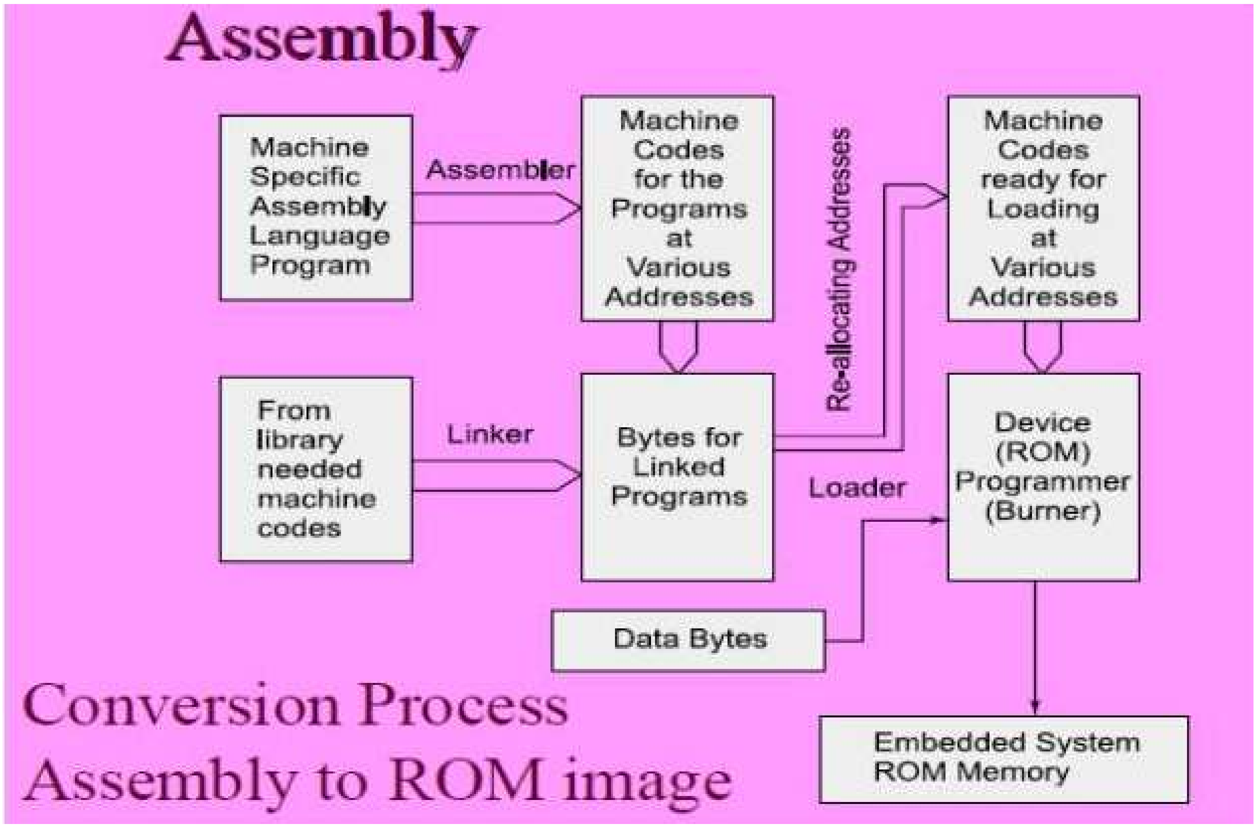
Figure 1:Rom Image Structure

Building Software for Embedded System

- The software for an embedded system can be prepared in following languages:
  - Machine code
  - Assembly line code
  - High level language
- **Machine coding**
  - For machine coding designer should have knowledge of processor and devices to be attached to system
  - In this programmer defines the address and corresponding bytes or bits at each addressing during coding
  - It is used for coding or interfacing specific physical device or sub-system.
  - It is very tedious process and time consuming.

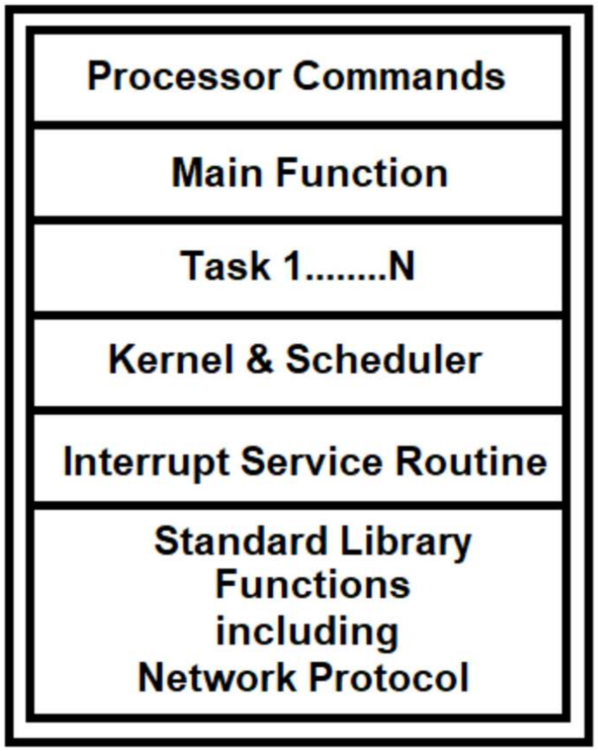
Building Software for Embedded System- Assembly Language

- Assembler is used to write software for embedded system in assembly code.
- Designer must have knowledge of processor and its instruction sets used in embedded system.
- The process of converting the assembly code into ROM image is shown in figure

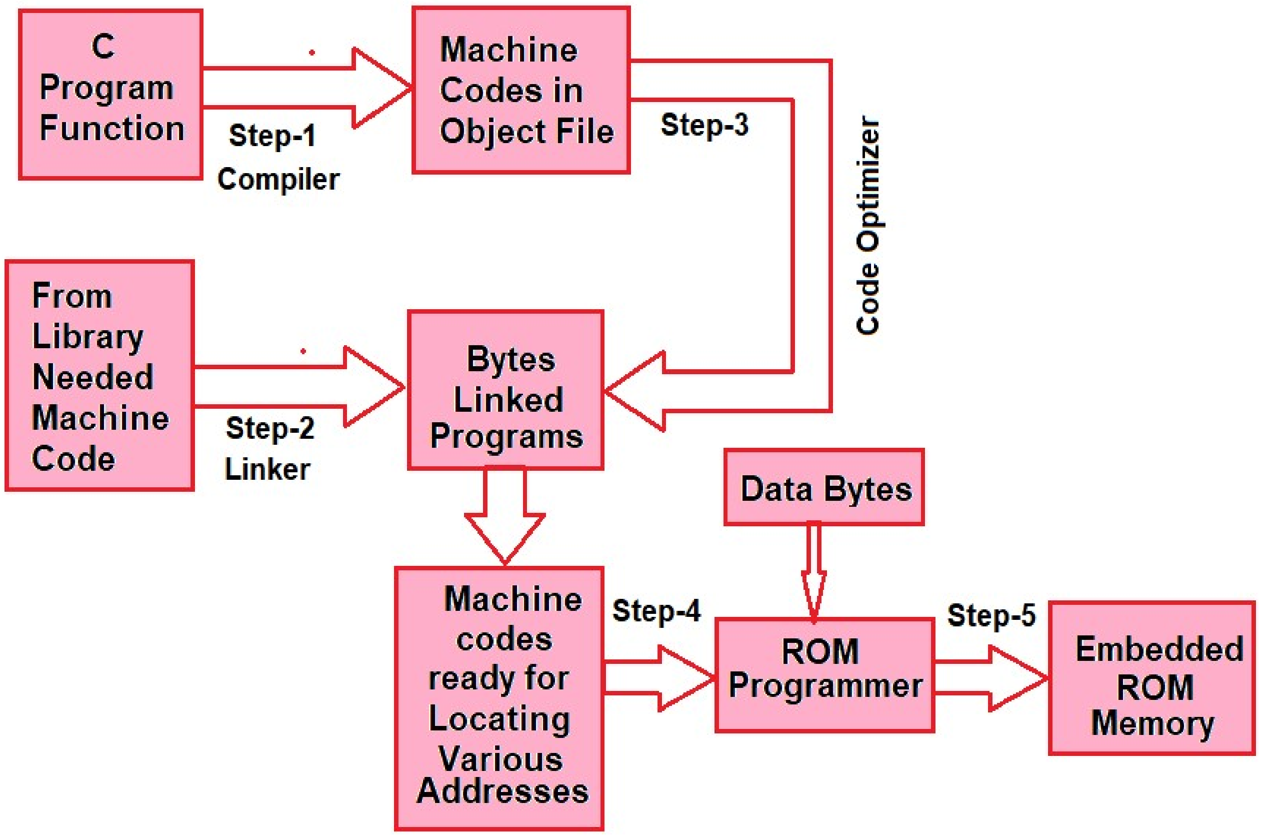

Assembly
Machine Specific Assembly Language Program
Assembler
Machine Codes for the Programs at Various Addresses
Re-allocating Addresses
Machine Codes ready for Loading at Various Addresses
From library needed machine codes
Linker
Bytes for Linked Programs
Loader
Device (ROM) Programmer (Burner)
Data Bytes
Embedded System ROM Memory
Conversion Process
Assembly to ROM image

Software in High Level Language

- As coding in Assembly language is time consuming, designer generally use high level language like C/C++Visual C++ / Java /Python etc.
- The coding can be implemented without prior knowledge of Processor.
- Figure shows the different programming layers using Embedded C programming.
- The Process of converting high level programming in C to ROM image is shown in figure in next slide.


Processor Commands
Main Function
Task 1........N
Kernel & Scheduler
Interrupt Service Routine
Standard Library Functions including Network Protocol

The Process of converting high level programming in C to ROM image



Building Software for Embedded System- High Level Language

- The various high level language used in embedded software development are
  - C/C++ Language
  - Embedded C language
  - HDL
- HDL- Hardware Description Language- Specifically used for system design using FPGA devices and its associated SoC.
- We will discuss briefly in C/C++ language.

Assembly Language Programming

- Foe Assembly language programming of embedded system, a designer should have knowledge of followings
    - Processor architecture details
    - Register architecture
    - Device architecture for external connections and their communication methods
- Assembly language has following advantages
- Gives precise control over processor internal devices, and enables all type of processor-specific features.
- The assembly codes are compact and requires smaller memory.
- Assembly code for device drivers are very small in size also requires less memory.
- Bottom-up approach is easily used.

HIGH LEVEL LANGUAGE PROGRAMMING

- High-level language for source files may be written in C, C++, C#, Visual C++, Java which provides great ease of writing program. It offers following advantages
  - Program development cycle very small
  - Top-down approach is used.
  - It can take various data types during programming and simple to declare different data types.
  - The program supports ‘type checking’ for easy for debugging.
  - This allows to use different program flow control structures such as loop, case etc.
  - These programs basically not processor specific. For any hardware changes, only modules for the device drivers and device management, initialization etc needs to be rewritten. So these coding schemes offer more portability.

‘C’ Programming Elements

- The various ‘C’ programming elements are
  - Preprocessor declarations, definitions and statements
    - Include directives for file inclusion
    - Preprocessor global variables
    - Constraint definitions
    - global variable data types, type declarations and data structures, macros, functions
  - Main functions
  - Functions, exceptions and ISRs

Include directives

- Include is a pre-processor directive that allows us to include the contents of a file.
- The various files are included as specific header file as shown in example below.

  Example-1

  ```
  #include “VxWorks.h”    //include vxworks function
  #include “semLib.h”   // includes semaphore functions
  #include “taskLib.h”    // includes multitasking functions
  ```
- **Including coding** file as   #include “prctlHandler.c”
- **Including constant** data files : this file includes various constants to be used in programming and may have extension “.const”
- includes string data files: : this file includes various strings to be used in programming and may have extension “.strings” or “.str” or “.txt”
- Including initial datafiles: initial or default data files are for locating in ROM of embedded system.
- Including basic variable files
- Pre-processor global variables are declared by using “#define” syntax as #define time
- Pre-processor constants are declared by using #define as #define pi 3.147

Programming elements: Macros & Functions & others

- The various programming elements are

Program elements	uses
Macro	Executes a named small collection of codes. It does not saves context after execution.
Function	Executes a named set of codes with values or references to the values passed from calling functions through its arguments. It does saves context after.
Main function	A function which runs first. It does not saves context after execution.
Reentrant function	The function is reentrant which it enters into same state of variables and processor registers. It does saves context after.
ISRs or device drivers	Declarations of the functions and data types, typedef, and executes a named set of codes. It does saves context after.
Process or task or thread	A set of instruction which run under control of OS. It does saves context after.
Recursive functions	A function that calls itself. It does saves context after.

Difference between macros and function

- A macro is a collection of codes that is defined in a program by a name. it differs from function that once the macro is defined, the compiler puts the corresponding code for it at every place.
- The codes of a function are compiled once, processor saves the context on calling and after returning the processor restores the context. Sometime functions may not return any value.
- The codes for macro are compiled in every function wherever that macro name is used. Before compilation, compiler puts the code at the places wherever the macro is used. When the macro runs inside a function, the processor does not save the context as there is no return in macro.
- Macros are used for short codes only. This is due to fact that function call is used for short codes instead of macro, the overheads take additional time.

Programming elements: data types,

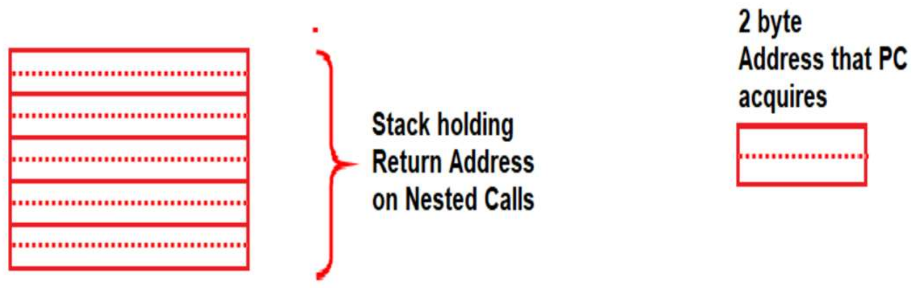
- Various **data types** that can be used in Embedded C programming are
  - char(8 bit)
  - Unsigned short(16 bit)
  - Unsigned int(32bit)
  - Int (32)
  - Long double(64 bit)
  - Float (32 bit)
  - Double(64 bit)
- **Modifiers**
  - A modifier is an action data type. Modifier ‘auto’ or no modifier inside function block, Ram is allocated otherwise ROM is allocated for various program locator.

Programming elements: Pointers, data structures

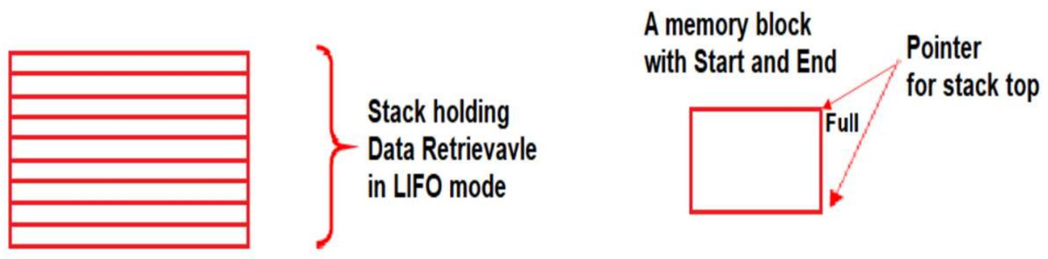
- Use of **pointers** and **Null pointers**
  - Pointers are powerful tool when used correctly and according to certain basic principle. A pointer is a reference to a starting memory address. The pointer can refer to a variavle, data structure or function.
  - This can be written as : char *0x1000 where '*' denotes the pointer used before a variable.
- **Data structure**
  - Data structure is a way to organising a set of data elements of same type or different types. Data in data structures can be accessed by the help of pointers or indices or functions.
  - It serves as a set of memory addresses in a organised way so that any data can be retrieved easily. Few data structures are stack, one dimensional array, queue, circular queue, pipe, table, look-up table, and list.

Programming elements: Data structures(Stack)

- A pointer which points to stack top is needed for handling each stack. This is called as stack pointer which always points to stack top location.
- As shown in figure a processor must have one stack pointer and this is used when there is a call fo subroutine in main program.
- A value from the stack is retrieved using LIFO mode.



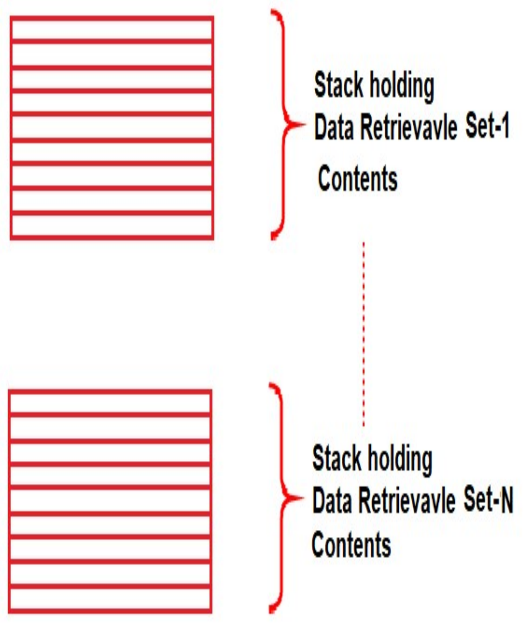
(a) A stack due to nestedfunction calls and pushing of program counter



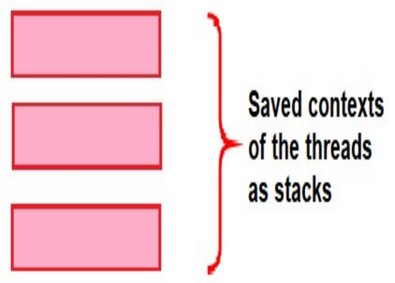
(b) Stack of pointers and parameter pushed on to stack before the next context switch

Programming elements: Data structures(Multiple Stack)

- Various stack structures are created during processing.
- When a processor has a only a SP, then memory address are used as stack pointers for each stack structure.
- A processor may have several SP and some advance processor may have SP, RIP(return instruction pointer), FP(data frame pointer), and FEP(previous Program Frame pointer).



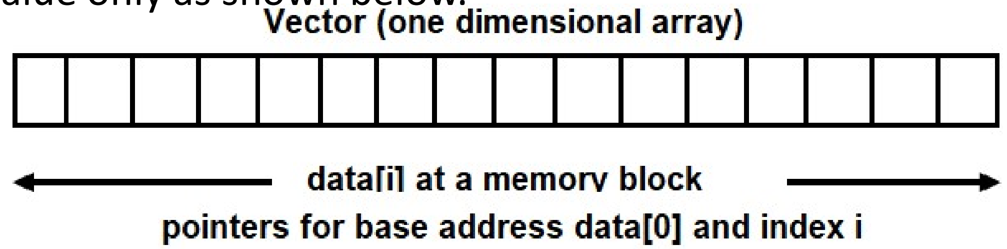
(a) Multiple stacks pushed on the stacks each having separate pointer



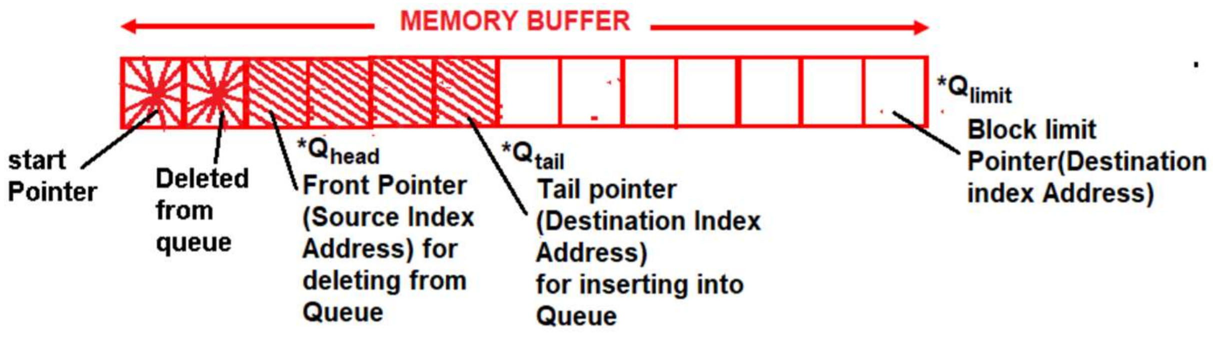
(b) Multiple stacks of CPU registers for multiple tasks which are pushed to stack before context switch

Programming elements: Arrays & Queue

- An array is an important part of the C programming that has multiple data elements.
- Each element is identified by an index or by a set of indices.
- Index is an integer starts from 0 to array length-1 in one dimension. Data can be retrieved from the array through index value only as shown below.

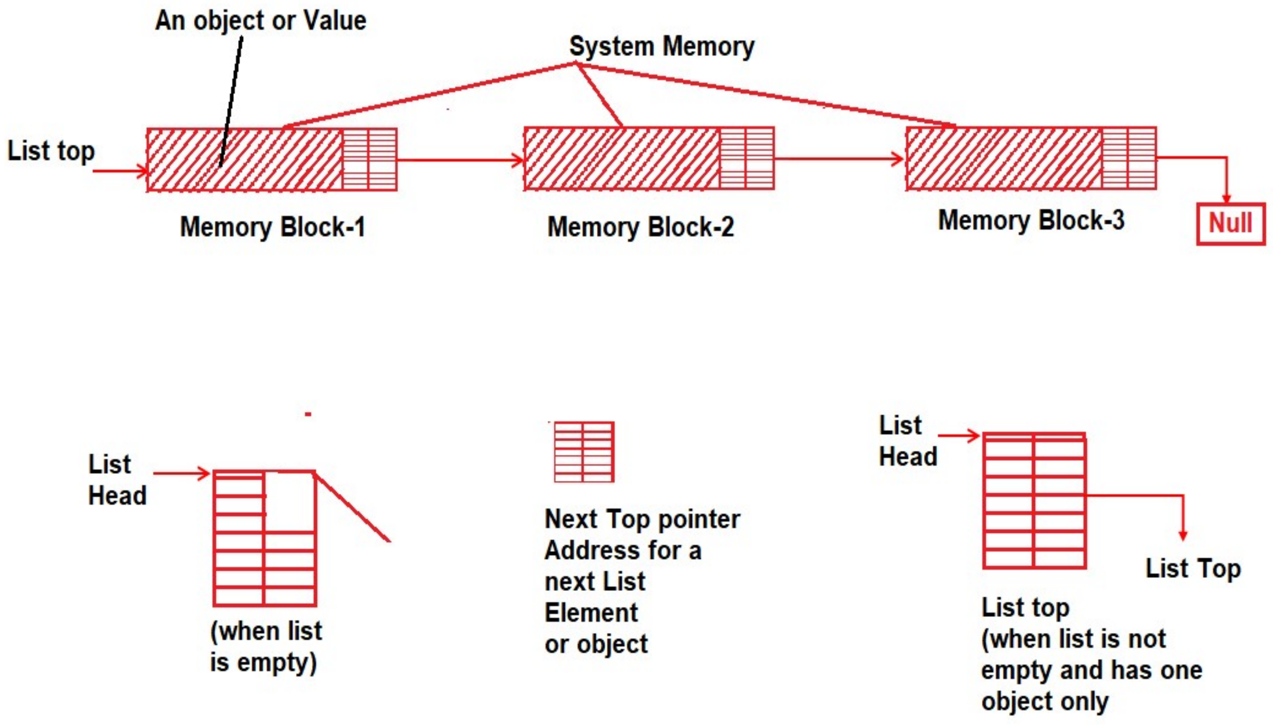

Vector (one dimensional array)
data[i] at a memory block
pointers for base address data[0] and index i

- Array are read with indices value and each element is read in queue from the address of next to the address from where queue element was last read.


MEMORY BUFFER
start Pointer
Deleted from queue
*Q_head Front Pointer (Source Index Address) for deleting from Queue
*Q_tail Tail pointer (Destination Index Address) for inserting into Queue
*Q_limit
Block limit Pointer(Destination index Address)

Programming elements: Queue

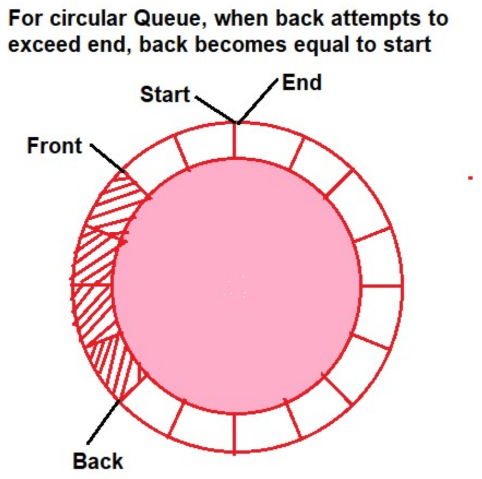
- A list for non-consecutively located objects at the memory. A list is a data structure with a set of number of distinct sets of memory addresses, one for each element.
- A list has a top (head) pointer for the memory address from where the list starts.
- Each list element at the memory also stores pointer to the next element at next set of addresses. The last element in list points to null



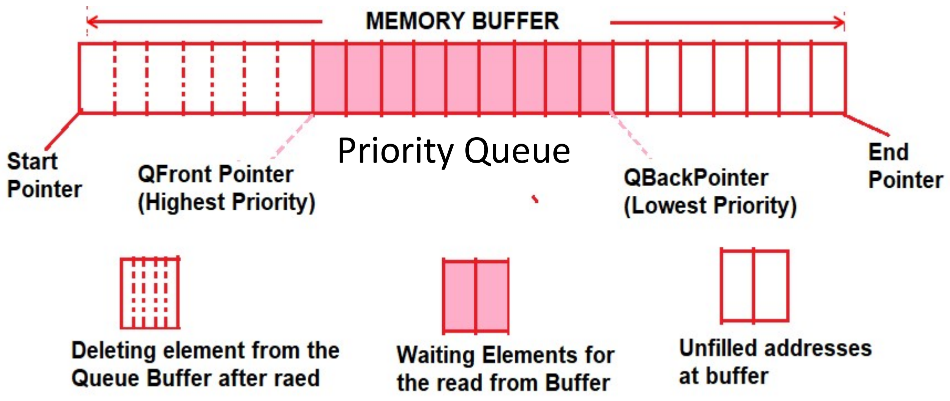
Pointers in a list with elements at a set of memory addresses

Programming elements: Circular Queue and Priority Queue

- **Circular Queue**
  - A circular queue is a special queue. A pointer on reaching $*Q_{limit}$, returns to it starting value Q_{start}. A print buffer and display buffer are example of circular queue.
  - Each character or element read in FIFO mode and figure shows the memory block with a circular queue with its two pointers needed for insertion and deletion.

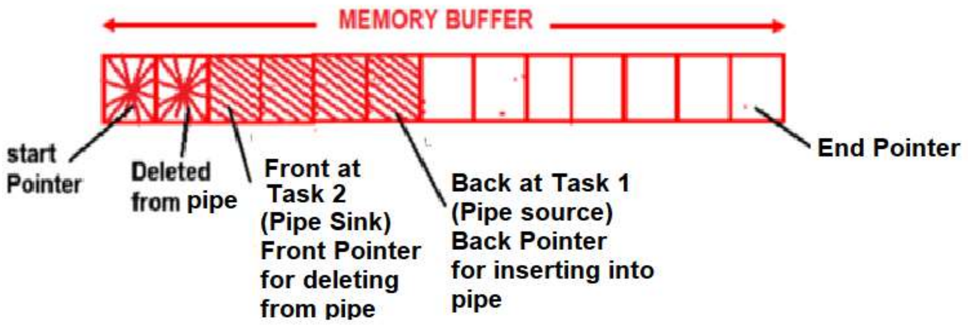

For circular Queue, when back attempts to exceed end, back becomes equal to start
Start
End
Front
Back

- **Priority Queue**
  - A priority queue is a special queue in which insertion takes place in $*Q_{head}$ if priority of the element is higher than previously inserted element, else at the $*Q_{tail}$. Message posted from the arrange in priority queue.
  - Figure shows the priority Queue with elements arranged in priority order between memory addresses between head and tail.

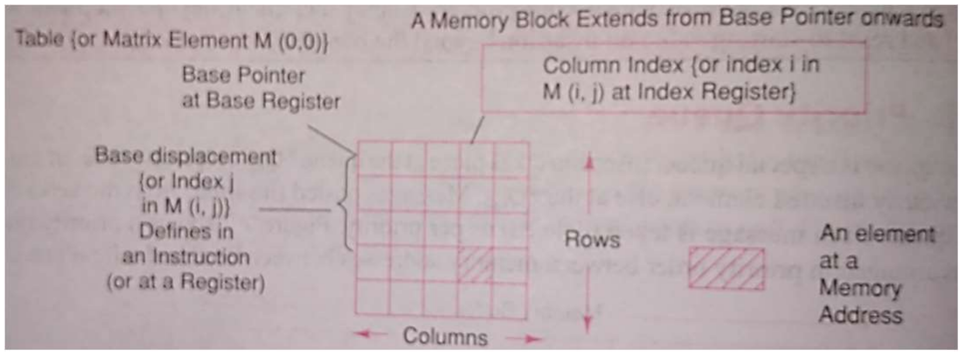

MEMORY BUFFER
Priority Queue
Start Pointer
QFront Pointer (Highest Priority)
QBackPointer (Lowest Priority)
End Pointer
Deleting element from the Queue Buffer after raed
Waiting Elements for the read from Buffer
Unfilled addresses at buffer

Programming elements: Pipe, Table, Hash Table

- **Pipe**
  - A pipe is a device which uses driver function in which insertions are made from source-end and deletions from the destination end. Sources and destinations are connected by a function called pipe as shown in figure.


MEMORY BUFFER
start Pointer
Deleted from pipe
Front at Task 2 (Pipe Sink) Front Pointer for deleting from pipe
Back at Task 1 (Pipe source) Back Pointer for inserting into pipe
End Pointer

- **Table**
  - A table is 2 dimensional array and it contains a base pointer which points to the first column first row of the table. There are two indices- row indices, column indices. Figure shows the table


A Memory Block Extends from Base Pointer onwards
Table {or Matrix Element M (0,0)}
Base Pointer at Base Register
Column Index {or index i in M (i, j) at Index Register}
Base displacement {or Index j in M (i, j)} Defines in an Instruction (or at a Register)
Rows
An element at a Memory Address
Columns

Loops, Infinite Loops, conditions

- A set of statements is repeated in a loop each time with index changed value. In this case loop starts from an initial value or variable or condition and it executes until the limiting value in the loop is satisfied.
- This limiting value may be a array index value or a condition.
- The various type of loops are .
- Finite loop

```
        For (i=0;i<=10;i++){
                Statement1;
                Statement 2;
                }
        i=0;
                While(i<100){statement 1;
                        Statement 2;
                        i++;}
```

- Infinite loop: generally infinite loops are the features of an embedded system. The infinite loop is written as
- While(1) {statement 1;
- Statement 2;}