# Assembly Level Machine Organization

*Just to let you know* –The topics related to Assembly level machine organization are:

1. Basic organization of the von Neumann machine
2. Control unit; instruction fetch, decode, and execution
3. Instruction sets and types (data manipulation, control, I/O)
4. Assembly/machine language programming
5. Instruction formats
6. Addressing modes
7. Subroutine call and return mechanisms (xref PL/Language Translation and Execution)
8. I/O and interrupts
9. Heap vs. Static vs. Stack vs. Code segments
10. Shared memory multiprocessors/multicore organization
11. Introduction to SIMD vs. MIMD and the Flynn Taxonomy

Since, it's not feasible to cover all the above mentioned topics in 2 Lecture Hours.

As per the **Course Plan,** We'll have to discuss:

1. Von  Neumann Machine
2. X86 Assembly Instructions
3. Heap, Stack, Code

# Von Neumann Machine.

The Content for “Von Neumann Model” shall be found in the TEXT BOOK (PFA) -

**The Essentials of Computer Organization and Architecture - Linda Null - BOOK**

**(PDF Page 57 - 60)**

**If you find any more stuff that could be added or replaced instead - Kindly let us know

# X86 Assembly Instructions

The Content for “Assembly Language Instructions in context to Operating Systems” shall be found in the TEXT BOOK (PFA) -

**Computer Systems- A Programmers Perspective - BOOK (PDF Page 206 - 213)**

- Embedding Assembly Code in C Programs

**If you find any more stuff that could be added or replaced instead - Kindly let us know

# Stack & Heap

The Content for “Assembly Language Instructions in context to Operating Systems” shall be found in the TEXT BOOK (PFA) -

**Computer Systems- A Programmers Perspective - BOOK (PDF Page 34 - 35)**

- Virtual Memory

**Computer Systems- A Programmers Perspective - BOOK (PDF Page 119 - 124)**

- Accessing Information
    a. Data Movement Instructions

**If you find any more stuff that could be added or replaced instead - Kindly let us know

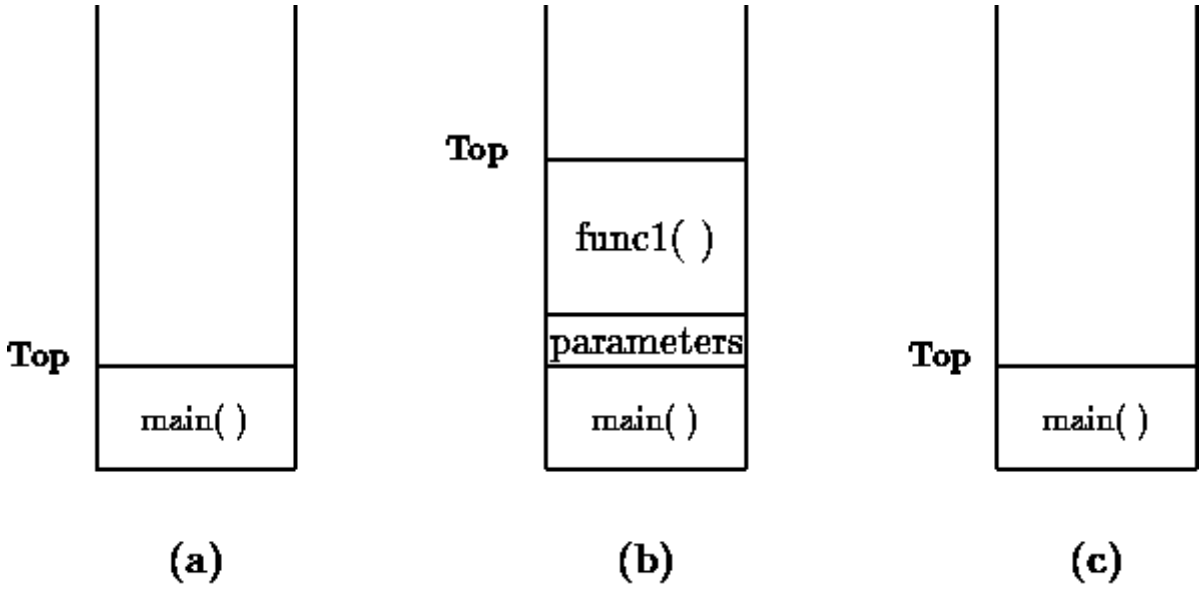
# Stack Vs Heap Allocation

**The Content below are extracted for “Stack versus Heap” from the WWW**

When a program is loaded into memory, it is organized into three areas of memory, called *segments*: the *text segment*, *stack segment*, and *heap segment*. The text segment (sometimes also called the code segment) is where the compiled code of the program itself resides. This is the machine language representation of the program steps to be carried out, including all functions making up the program, both user defined and system.

The remaining two areas of system memory is where storage may be allocated by the compiler for data storage. The stack is where memory is allocated for automatic variables within functions.

A stack is a *Last In First Out* (LIFO) storage device where new storage is allocated and deallocated at only one ``end'', called the Top of the stack. This can be seen in Figure below




When a program begins executing in the function main(), space is allocated on the stack for all variables declared within main(), as seen in Figure(a). If main() calls a function, func1(), additional storage is allocated for the variables in func1() at the top of the stack as shown in Figure(b).

Notice that the parameters passed by main() to func1() are also stored on the stack. If func1() were to call any additional functions, storage would be allocated at the new Top of stack as seen in the figure. When func1() returns, storage for its local variables is deallocated, and the Top of the stack returns to to position shown in Figure above.

If main() were to call another function, storage would be allocated for that function at the Top shown in the figure. As can be seen, the memory allocated in the stack area is used and reused during program execution. It should be clear that memory allocated in this area will contain garbage values left over from previous usage. The heap segment provides more stable storage of data for a program; memory allocated in the heap remains in existence for the duration of a program.

Therefore, global variables (storage class external), and static variables are allocated on the heap. The memory allocated in the heap area, if initialized to zero at program start, remains zero until the program makes use of it. Thus, the heap area need not contain garbage.

**If you find any more stuff that could be added or replaced instead - Kindly let us know