Assembly - Memory Segments

We have already discussed the three sections of an assembly program. These sections represent various memory segments as well.

Interestingly, if you replace the section keyword with segment, you will get the same result. Try the following code –

Live Demo

```
segment .text       ;code segment
   global _start    ;must be declared for linker

_start:             ;tell linker entry point
   mov edx,len      ;message length
   mov ecx,msg      ;message to write
   mov ebx,1        ;file descriptor (stdout)
   mov eax,4        ;system call number (sys_write)
   int 0x80         ;call kernel

   mov eax,1        ;system call number (sys_exit)
   int 0x80         ;call kernel

segment .data       ;data segment
msg     db 'Hello, world!',0xa   ;our dear string
len     equ     $ - msg          ;length of our dear string
```

When the above code is compiled and executed, it produces the following result –

```
Hello, world!
```

Memory Segments

A segmented memory model divides the system memory into groups of independent segments referenced by pointers located in the segment registers. Each segment is used to contain a specific type of data. One segment is used to contain instruction codes, another segment stores the data elements, and a third segment keeps the program stack.

In the light of the above discussion, we can specify various memory segments as –

- **Data segment** – It is represented by **.data** section and the **.bss**. The .data section is used to declare the memory region, where data elements are stored for

the program. This section cannot be expanded after the data elements are declared, and it remains static throughout the program.

The .bss section is also a static memory section that contains buffers for data to be declared later in the program. This buffer memory is zero-filled.

- **Code segment** – It is represented by **.text** section. This defines an area in memory that stores the instruction codes. This is also a fixed area.
- **Stack** – This segment contains data values passed to functions and procedures within the program.