Topic 11
Lecture 11
Memory Management

CSCI 150

Assembly Language / Machine Architecture

Prof. Dominick Atanasio

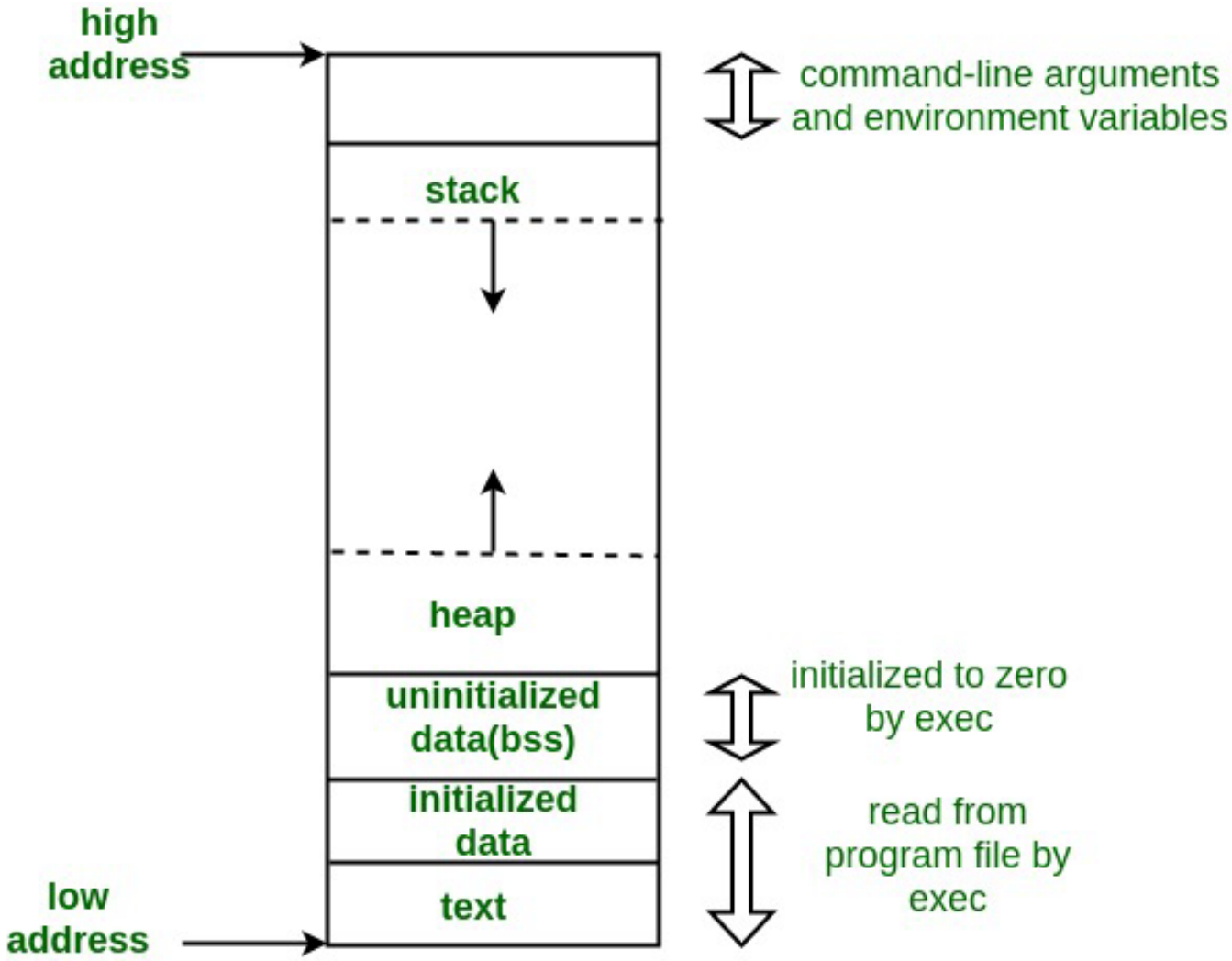
Introduction

- Thus far we have primarily worked with the .data, .bss, and the program stack.
- The memory allocation reqrirements of the *data* and *bss* segments are known at compile time.
  - Space for these two sections are reserved when the program starts
- While the use of the stack is dynamic during runtime, the data structure of every function that can be called is known at compile-time as well.
  - We can push and pop based on user input, but it is preferred to use local variables instead.
- In this topic we will learn how to request heap space for runtime allocated storage.

Memory Layout

Sections of memory from lowest to highest address

- text: holds the machine code for all instructions
- data: holds global initialized values. Capacity does not change.
- bss (block starting symbol): global uninitialized data segment. Reserved space for values not known at compile time.
- heap: storage for data that is not known at compile time (dynamically allocated). Capacity does not change.
- run-time stack: stores information about the active subroutines of the program. stores local variables, return address, parameter arguments, etc.



Dynamic Memory (heap) Allocation

- We allocate space on the heap by changing the address that marks the beginning of the heap space.
- To change that address we must first know the address.
  - To get the address of the beginning of the heap we perform a syscall (the brk syscall) with the value 0 in EBX
    - The value in EBX represents the new address of the brk.
  - The system call is 0x2d
  - It returns the address of the “program break“, the end of the statically reserved storage space.

```
mov eax, 0x2d    ; setup syscall for brk
xor ebx, ebx     ; set ebx to 0
int 80h          ; interupt
```

Dynamic Memory (heap) Allocation

- This address should be stored in a local or global memory location
- To reserve space on the heap we add to the address of the *program break*
  - Remember, the heap grows up while the stack grows down
- Example: assume that we have stored the *program break* address at a label called *prog_brk* that was that was reserved in the *bss* section. This will reserve 4KB of space.

```
mov     eax, 0x2d
lea     ebx, [prog_brk + 4096]
int     80h
```

- There is now 4096 bytes available to our program starting at [prog_brk]
- We can "deallocate that space by calling *brk* with the original address

Usage

- There is a function in C called *malloc* which performs the brk syscall
- Because this syscall is expensive in terms of CPU time, the C compiler will consolidate all of the malloc instructions into one large reservation.
- It will then be managed by code built by the compiler. When the *free* function is called, the space is not deallocated but marked as available.
  - Assembly language programmers must write this management code.
- There is a limit on how much space can be allocated. This limit is set by the syscall, *setrlimit* (set resource limit).

Linked Data Elements

- Often, data elements that are stored in the heap space include an extra field.
  - This field holds the address of the next element in a chain. the last element's address is set to 0;
- This dynamically allocated data structure is known as a linked list.
  - It is a collection of elements that can dynamically grow
  - It requires memory management because space cannot be deallocated.

Example

- Write a program that creates a linked list of Dwords