# MOVING DATA | SEC204

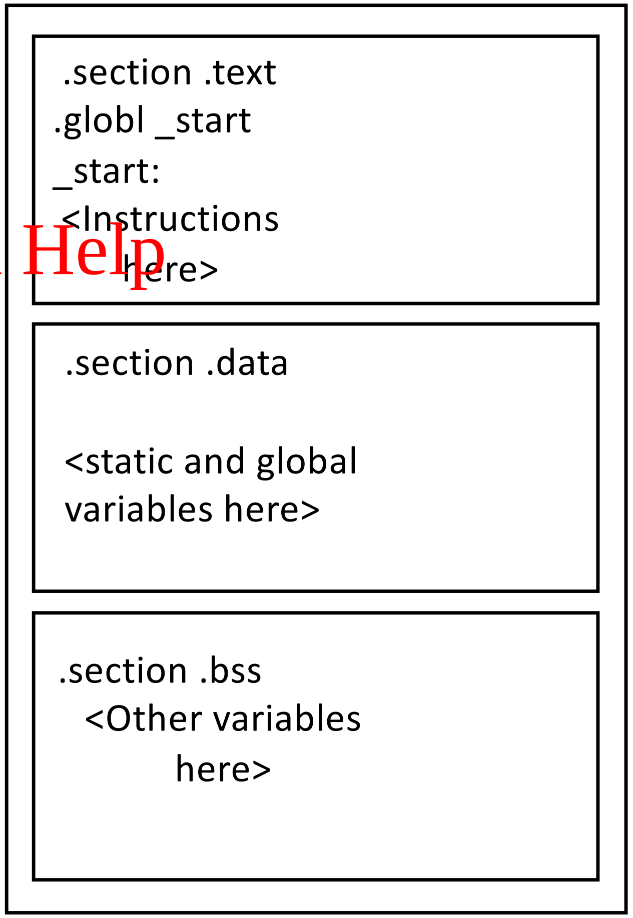
## Overview

- Sections of a program
- Move instruction
- Indexed memory mode

# INTRODUCTION

# SECTIONS OF A PROGRAM

- **.section .text**
- The text section contains instructions
- Start of the program is defined by the **_start** label.
  - This indicates the first instruction from which the program should run. If the linker cannot find it, it will produce an error
- **.section .data**
- The data section contains static and global variables (data elements with a static value, variables accessible to all program functions)
- **.section .bss**
- The bss section contains other variables
- We'll talk about the stack and heap later on



```
.section .text
.globl _start
_start:
<Instructions
here>

.section .data

<static and global
variables here>

.section .bss
<Other variables
here>
```

## THE DATA SECTION

To define elements in the data section, you need label and directive

| Directive | Data Type |
|---|---|
| .ascii | Text string |
| .asciz | Null-terminated text string |
| .byte | Byte value |
| .double | Double-precision floating point number |
| .float | Single-precision floating point number |
| .int | 32-bit integer number |
| .long | 32-bit integer number (same as .int) |
| .short | 16-bit integer number |
| .single | Single-precision floating point number (same as .float) |

```
 .section .text
.globl _start
_start:
```

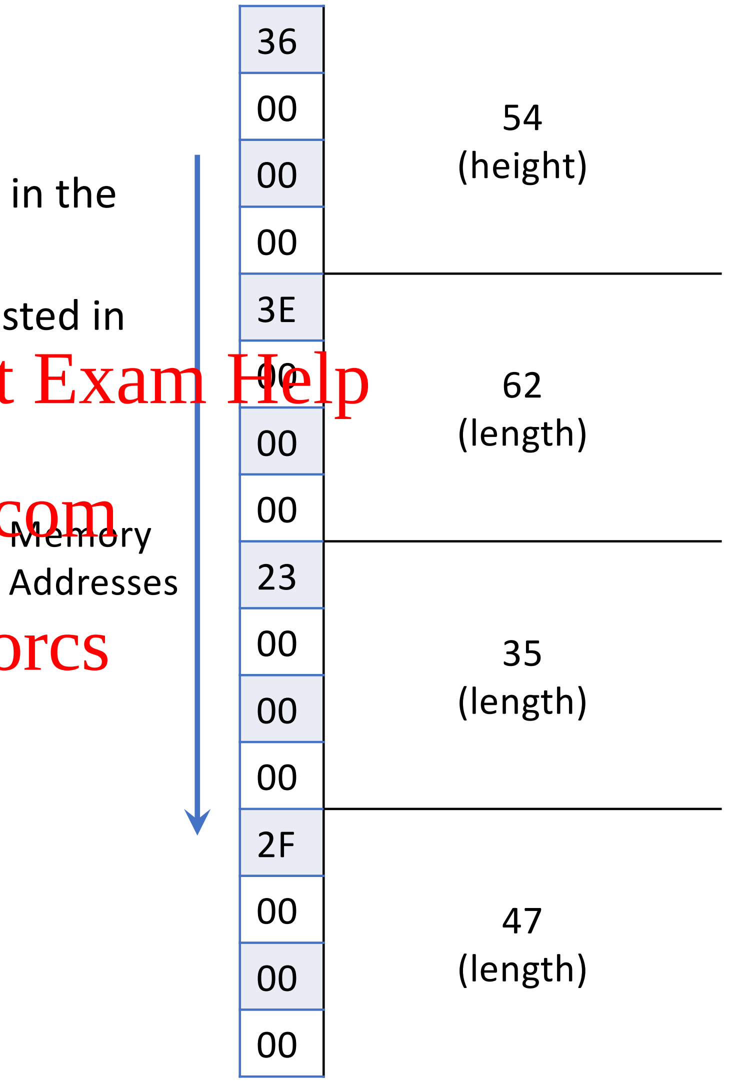
```
.section .data
msg:
   .ascii “This is a test”
factors:
   .double 37.45, 45.33, 12.30
height:
   .int 54
length:
   .int 62, 35, 47
```

```
 .section .bss
```

## THE DATA SECTION

- Each data is placed in memory in the order it is defined in the data section
- Elements with multiple values are placed in the order listed in the directive

```
.section .data
msg:
   .ascii "This is a test"
factors:
   .double 37.45, 45.33, 12.30
height:
   .int 54
length:
   .int 62, 35, 47
```



## STATIC SYMBOLS

- To declare constants (static data symbols), we  use .equ directive
- To reference it, you use the $  symbol

```
.section .data
msg:
     .ascii “This is a test”
factors:
     .double 37.45, 45.33, 12.30
height:
     .int 54
length:
     .int 62, 35, 47
.equ factor, 3
.equ LINUX_SYS_CALL, 0x80
```

```
movl $LINUX_SYS_CALL, %eax
```

Moves 0x80 to the eax register

## THE BSS SECTION

To define elements in the bss section, you declare raw segments of memory

| Directive | Data Type |
|---|---|
| .comm | Declares a common memory area for data that is not initialised |
| .lcomm | Declares a local common memory area for data that is not initialised |

Create sizetest1.s (using code below). Then assemble, and link it to view its size

```
.section .text
  .globl _start
  _start:
      movl $1, %eax
      movl $0, %ebx
      int $0x80
```

Create sizetest2.s (using code below), adding a 10,000-byte buffer. Then assemble, and link it to view its size

```
.section .bss
    .lcomm buffer, 10000
.section .text
  .globl _start
  _start:
      movl $1, %eax
      movl $0, %ebx
      int $0x80
```

Create sizetest3.s (using code below), adding a 10,000-byte buffer. Then assemble, and link it to view its size

```
.section .data
 buffer:
      .fill 10000
.section .text
  .globl _start
  _start:
      movl $1, %eax
      movl $0, %ebx
      int $0x80
```

## MOV INSTRUCTION FORMATS

- MOV source, destination
  - Source and destination can be memory addresses, data values stored in memory, data values defined in the instruction, or registers

- Can define the size of data element to be moved
  - **`movl`**: `l for 32-bit long word value`
  - **`movw`**: `w for 16-bit word value`
  - **`movb`**: `b for 8-bit byte value`

```
movl %eax, %ebx
Moves 32-bits %eax to the %ebx register
```

## MOVING DATA

1. Between registers

```
movl %eax, %ecx
movb %al, %bl
movw %ax, %bx
```

2. Between memory and registers

```
movl value, %eax
movl $10, %eax
movl %eax, value
```

Create file movtest1.s with the following content. Assemble, debug.

```
.section .data
   value:
      .int 1
.section .text
.globl _start:
   nop
```

…cont…

```
    movl value, %ecx
    movl $1, %ebx
    movl $0, %ebx,
    int $0x80
```

## INDEXED MEMORY MODE - TABLES

- When you specify more than one value on a directive in memory:

```
values:
   .int 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60
```

- A sequential series of data values are placed in memory

- Each value occupies one memory unit
- To determine the memory location, we need:

  - A base address
  - An Offset address to add to the base address
  - The size of the data element
  - An index to determine which data element to select

```
base_address (offset_address, index, size)
```

```
memory location = (base_address + offset_address + (index * size))
```

## INDEXED MEMORY MODE

- Example: how to access value 20 from array values

```
values:
   .int 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60
```

```
movl $2, %edi
movl values(, %edi, 4), %eax
```

```
 base_address (offset_address, index, size)

Base_address: values
Offset_address: (null)
Index: %edi (2=third value)
Size: 4 (int size)
```

# INDEXED MEMORY EXAMPLE

Create file movtest3.s with the following content.

```
.section .data
output:
   .asciz "The value is %d\n"
values:
   .int 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60
.section .text
.globl _start:
   nop
   movl $0, %edi
loop:
   movl values(,%edi,4),%eax
   pushl %eax
   pushl $output
   call printf
   addl $8, %esp
   inc %edi
   cmpl $11, %edi
```

…cont…

```
   jne loop
   movl $0, %ebx
   movl $1, %eax
   int $0x80
```

## Indexed memory example

1. Assemble, link, then run the code. What do you see?

```
$as -o movtest3.o movtest3.s
$ld -dynamic-linker /lib/ld-linux.so.2 -lc -o movtest3 movtest3.o
$./movtest3
```

Pay particular attention to instruction:

```
movl values(, %edi, 4), %eax
```

# INDEXED MEMORY - POINTERS

- Besides holding data, registers can be used to hold memory addresses
  - When a register holds a memory address, it is referred to as a pointer
  - Accessing data stored in the memory location using the pointer is called indirect addressing

- To access the memory location address of a data value, we prepend it with $

```
movl $values, %edi
```

$values: memory address of values. Moves the memory address of values to the EDI register

## INDEXED MEMORY - POINTERS

- To use a register as a pointer we use parenthesis

```
movl %ebx, (%edi)
```

Moves the value of EBX to the memory location contained in the EDI register

```
movl %edx, 4(%edi)
```

Moves the value of EDX to the memory location 4 bytes after the location pointed to by the EDI register

```
movl %edx, -4(%edi)
```

Moves the value of EDX to the memory location 4 bytes before the location pointed to by the EDI register

## POINTERS EXAMPLE

Create file movtest4.s with the following content. Assemble it with gstabs and run it in gdb

```
.section .data
values:
   .int 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60
.section .text
.globl _start:
   nop
   movl values, %eax
   movl $values, %edi
   movl $100, 4(%edi)
   movl $1, %edi
   movl values(, %edi, 4), %ebx
   movl $1, %eax
   int $0x80
```

In GDB:

- `x/4d &values`
- `step`
- `print $eax`
- `step`
- `print/x $edi`
- `step`
- `x/4d &values`

## FURTHER READING

- Professional Assembly Language, chapter 5, pg 91-106