

The slide features a white central area surrounded by decorative geometric patterns. In the top-left corner, there are several parallel blue lines forming a triangular shape. In the top-right corner, there are several parallel dark teal lines forming a triangular shape. In the bottom-left corner, there are several parallel teal lines forming a triangular shape. In the bottom-right corner, there are several parallel gold lines forming a triangular shape. The title "Linux Assembly" is centered in the white area in a brown font, and the author's name "Armine Hayrapetyan" is centered below it in a smaller brown font.

Linux Assembly

Armine Hayrapetyan

Outline

- “Hello, World” Source Code
- Registers
- System Calls

“Hello, World” Source Code

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

db

```
text db "Hello, World!",10
```



“db” stands for “define bytes”.
It essentially means that we are going
to *define* some raw *bytes* of data to
insert into our code.

db

```
text db "Hello, World!",10
```



This is the bytes of data we are
defining.

Each character in the string of text is a
single byte. The "10" is a newline
character, which I often denote as "\n".

db

```
text db "Hello, World!",10
```

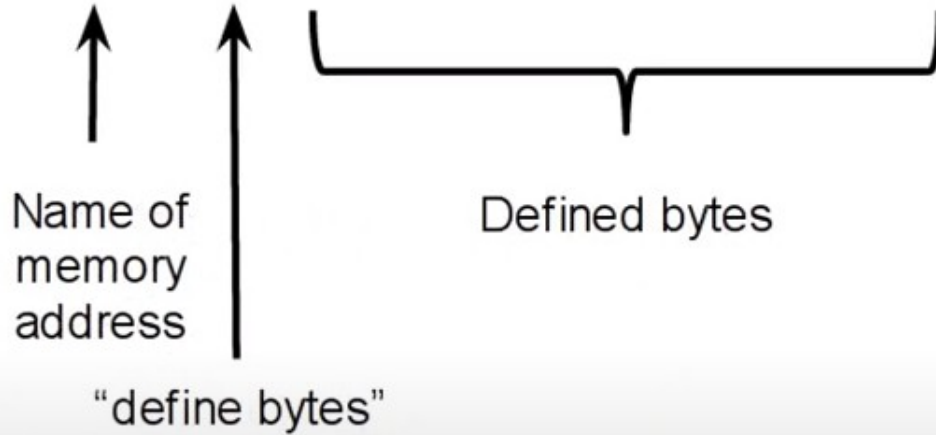


This is a name assigned to the address in memory that this data is located in.

Whenever we use “text” later in the code, when the code is compiled, the compiler will determine the actual location in memory of this data and replace all future instances of “text” with that memory address.

db

```
text db "Hello, World!",10
```



Registers

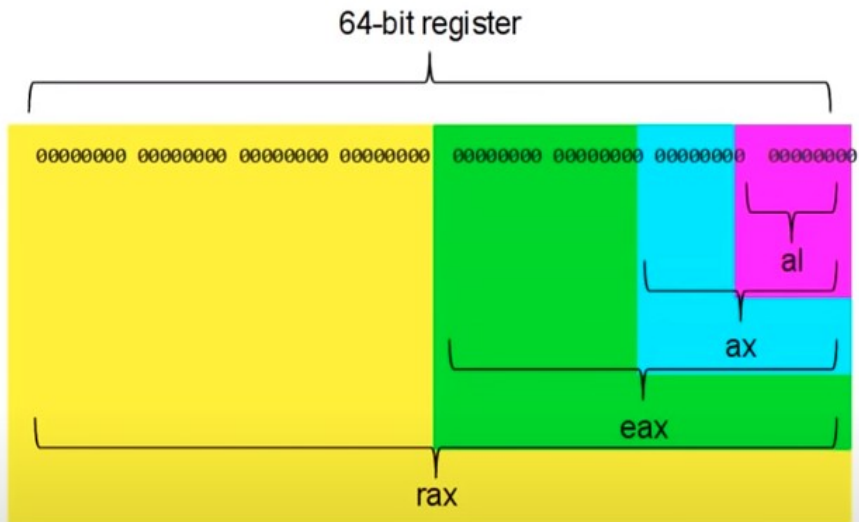
Registers are a part of the processor that temporarily holds memory.

In the x86_64 architecture, registers hold 64 bits.

This means each register can hold the values:

Unsigned:	0	—	18,446,744,073,709,551,616
Signed:	-9,223,372,036,854,775,808	—	9,223,372,036,854,775,807

Registers



8-bit	16-bit	32-bit	64-bit
al	ax	eax	rax
bl	bx	ebx	rbx
cl	cx	ecx	rcx
dl	dx	edx	rdx
sil	si	esi	rsi
dil	di	edi	rdi
bpl	bp	ebp	rbp
spl	sp	esp	rsp
r8b	r8w	r8d	r8
r9b	r9w	r9d	r9
r10b	r10w	r10d	r10
r11b	r11w	r11d	r11
r12b	r12w	r12d	r12
r13b	r13w	r13d	r13
r14b	r14w	r14d	r14
r15b	r15w	r15d	r15

System Call

A **system call**, or a **syscall**, is when a program requests a service from the **kernel**.

System calls will differ by operating system because different operating systems use different kernels.

All syscalls have an ID associated with them (a number).

Syscalls also take **arguments**, meaning, a list of inputs.

System Call Inputs by Register

Argument	Registers
ID	rax
1	rdi
2	rsi
3	rdx
4	r10
5	r8
6	r9

System Call List

syscall	ID	ARG1	ARG2	ARG3	ARG4	ARG5	ARG6
sys_read	0	#filedescriptor	\$buffer	#count			
sys_write	1	#filedescriptor	\$buffer	#count			
sys_open	2	\$filename	#flags	#mode			
sys_close	3	#filedescriptor					
...
pwritev2	328

sys_write

Argument Type	Argument Description
File Descriptor	0 (Standard Input), 1 (Standard Output), 2 (Standard Error)
Buffer	Location of string to write
Count	Length of string

sys_write

Argument	Registers
ID	rax
1	rdi
2	rsi
3	rdx
4	r10
5	r8
6	r9

sys_write

Suppose we want to write “Hello, World!\n” to the screen...

Argument Type	Argument Description
File Descriptor	0 (Standard Input), 1 (Standard Output), 2 (Standard Error)
Buffer	Location of string to write
Count	Length of string

syscall	rax	rdi	rsi	rdx	r10	r8	r9
sys_write	1	1	ADDR	14			

sys_write

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```



syscall	rax	rdi	rsi	rdx	r10	r8	r9
sys_write	1	1	ADDR	14			

sys_exit

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```



syscall	rax	rdi	rsi	rdx	r10	r8	r9
sys_exit	60	0					

“Hello, World” Source Code Overview

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```


Defines bytes
“Hello, Word!\n” and
labels the memory
address “text”.

“Hello, World” Source Code Overview

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```



sys_write(1,text,14)

“Hello, World” Source Code Overview

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

} sys_exit(0)

Sections

All x86_64 assembly files have three sections, the “.data” section, the “.bss” section, and the “.text” section.

The data section is where all data is defined before compilation.

The bss section is where data is allocated for future use.

The text section is where the actual code goes.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Label

A “label” is used to *label* a part of code.

Upon compilation, the compiler will calculate the location in which the label will sit in memory.

Any time the name of the label is used afterwards, that name is replaced by the location in memory by compiler.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

The “Start” Label

The “_start” label is essential for all programs.

When your program is compiled and later executed, it is executed first at the location of “_start”.

If the linker cannot find “_start”, it will throw an error.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Global

The word “global” is used when you want the linker to be able to know the address of some a label.

The object file generated will contain a link to every label declared “global”.

In this case, we have to declare “_start” as global since it is required for the code to be properly linked.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```


References

- <https://www.youtube.com/watch?v=BWRR3Hecjao&list=PLetF-YjXm-sCH6FrTz4AQhfH6INDQvQSn&index=2>
- <https://www.youtube.com/watch?v=VQAKkuLL31g&list=PLetF-YjXm-sCH6FrTz4AQhfH6INDQvQSn&index=1>