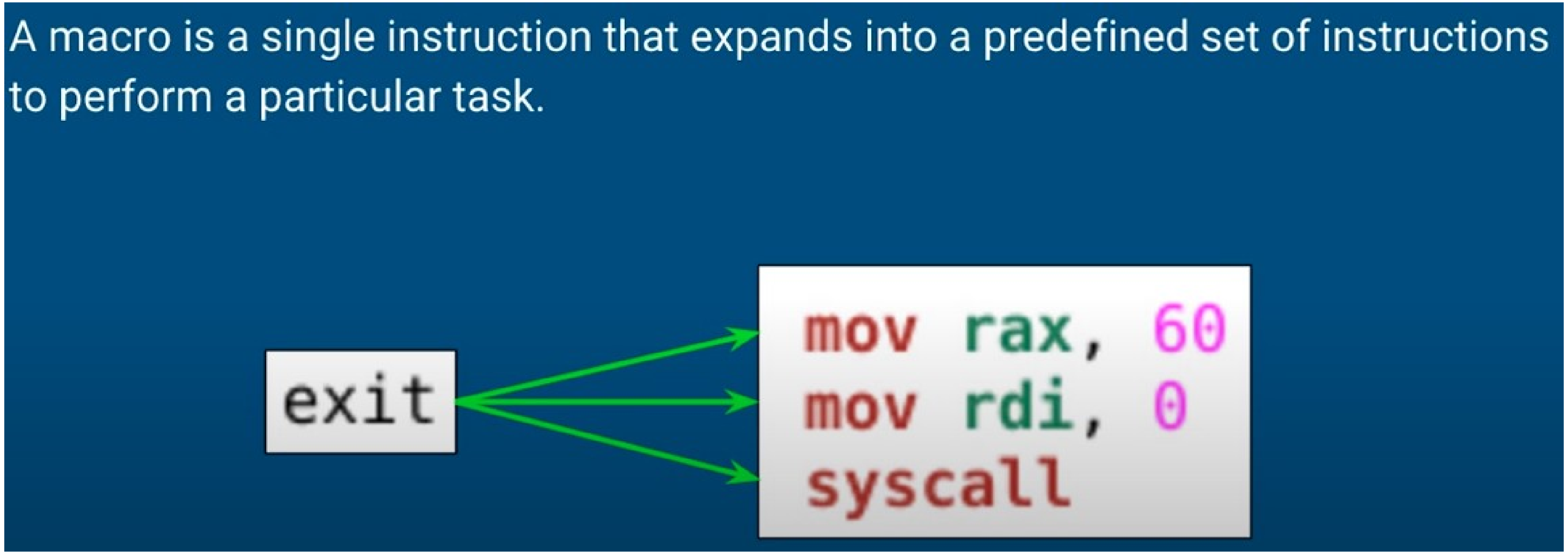
Linux Assembly

Armine Hayrapetyan

Outline

- Macros
- Arguments for Macros
- Local Labels in Macros
- Including External Files

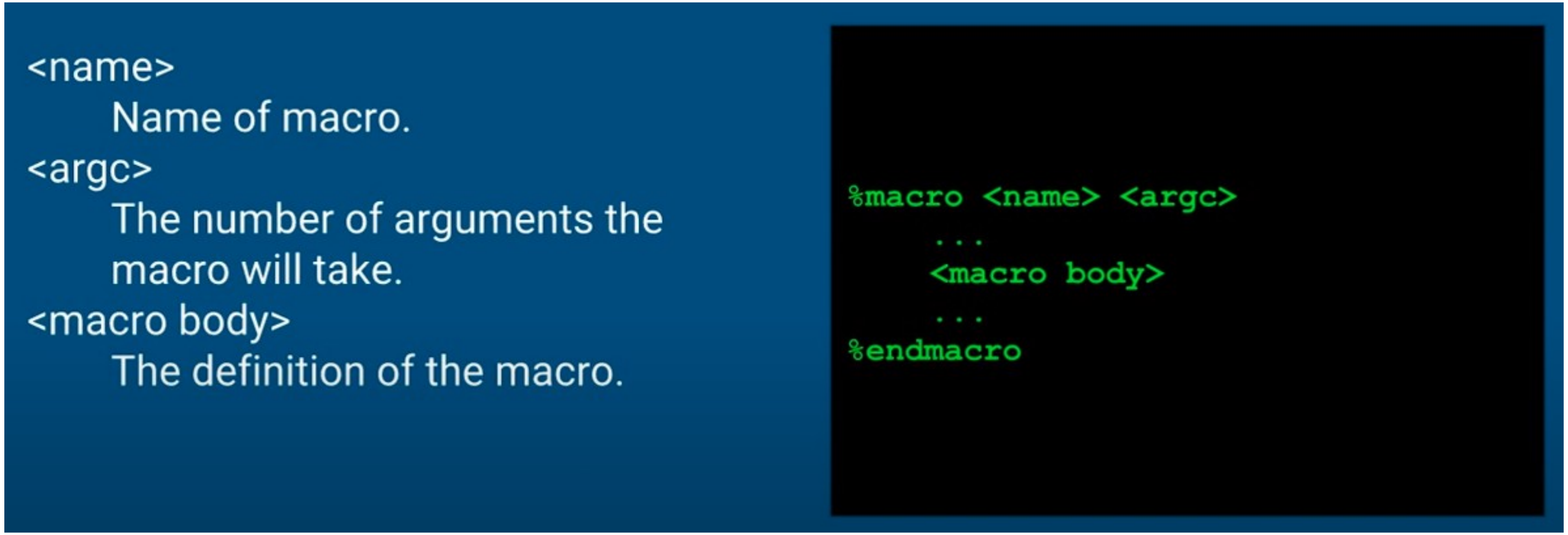
Macros



A macro is a single instruction that expands into a predefined set of instructions to perform a particular task.

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

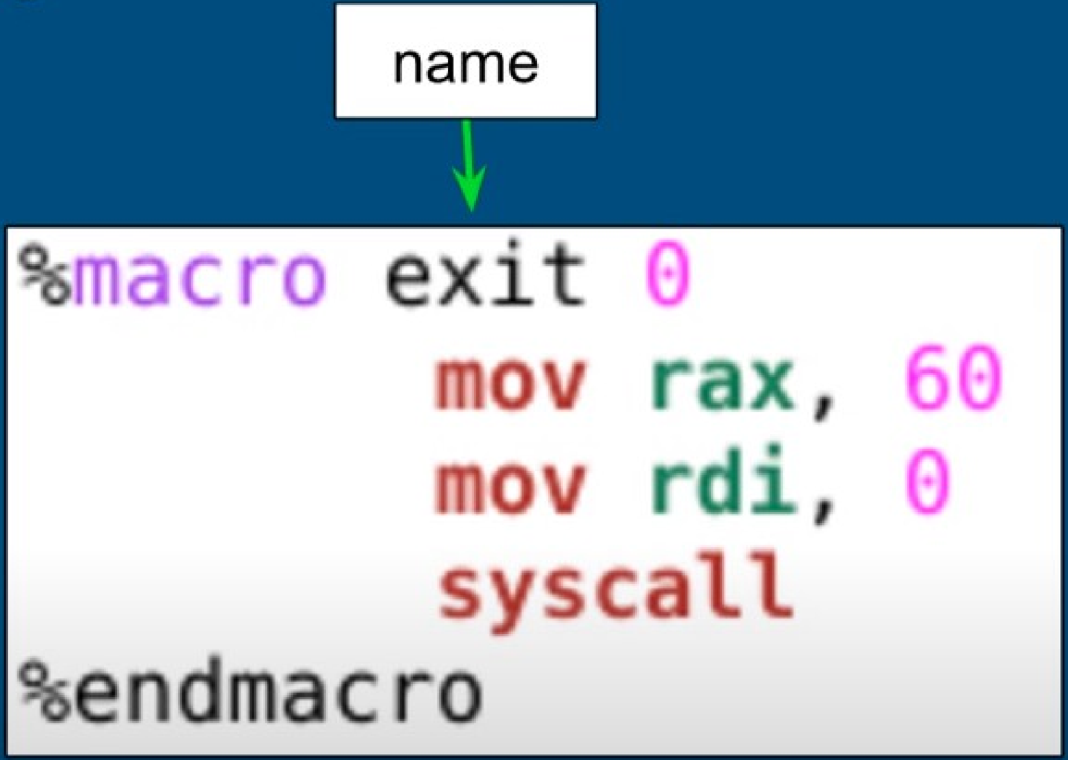
Defining Macros



<name>
    Name of macro.
<argc>
    The number of arguments the macro will take.
<macro body>
    The definition of the macro.

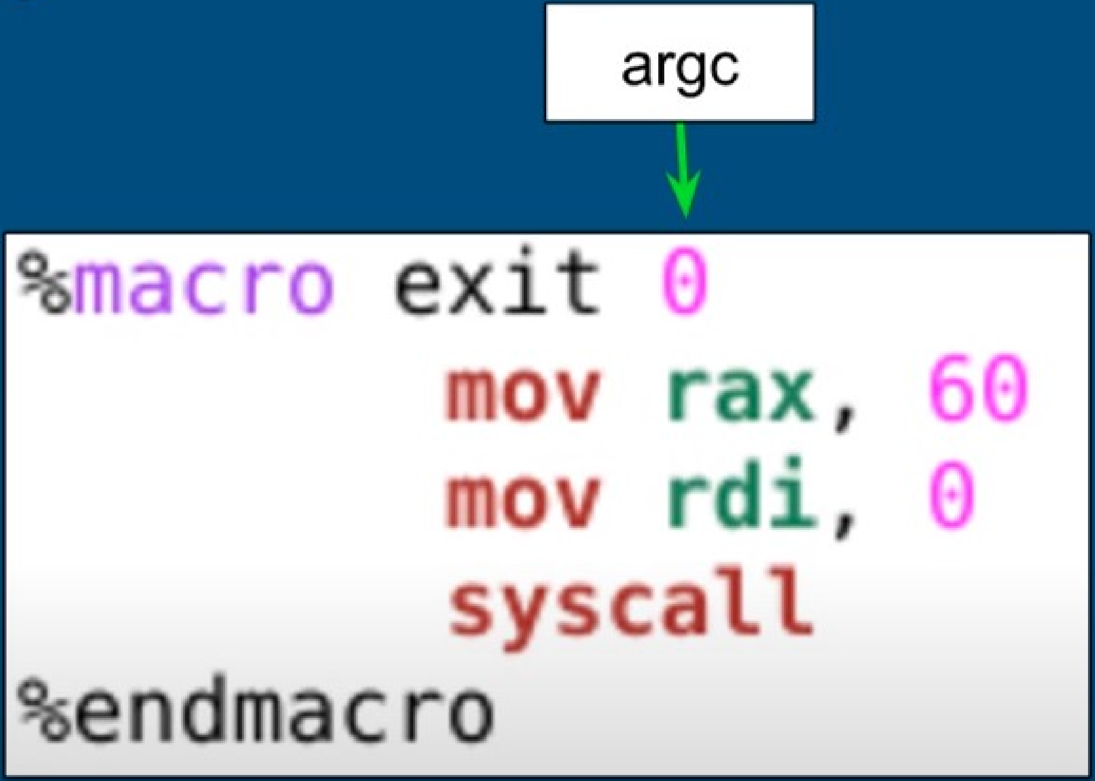
```
%macro <name> <argc>
    ...
    <macro body>
    ...
%endmacro
```

“Exit” Macro



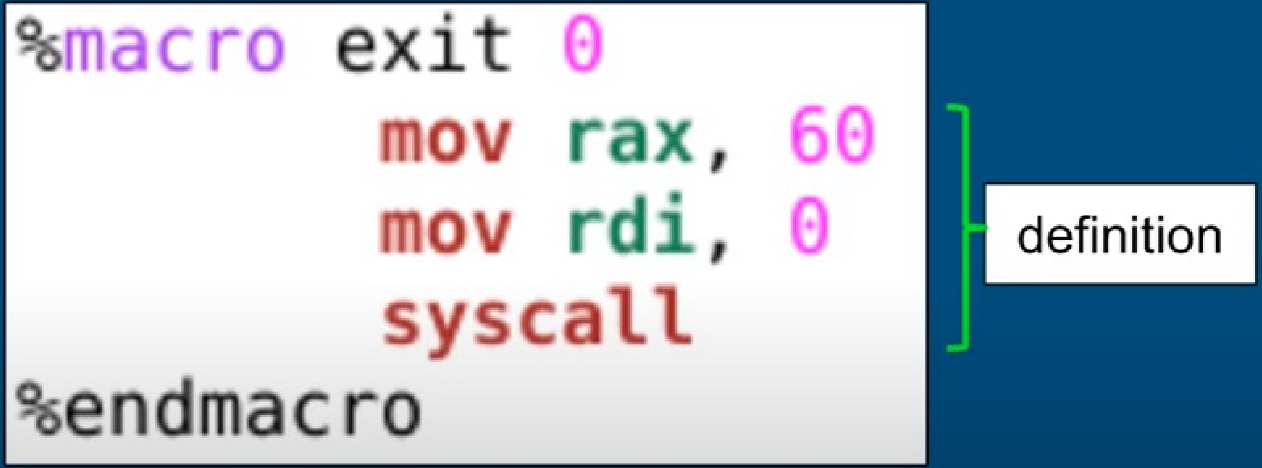
```
%macro exit 0
        mov rax, 60
        mov rdi, 0
        syscall
%endmacro
```

“Exit” Macro



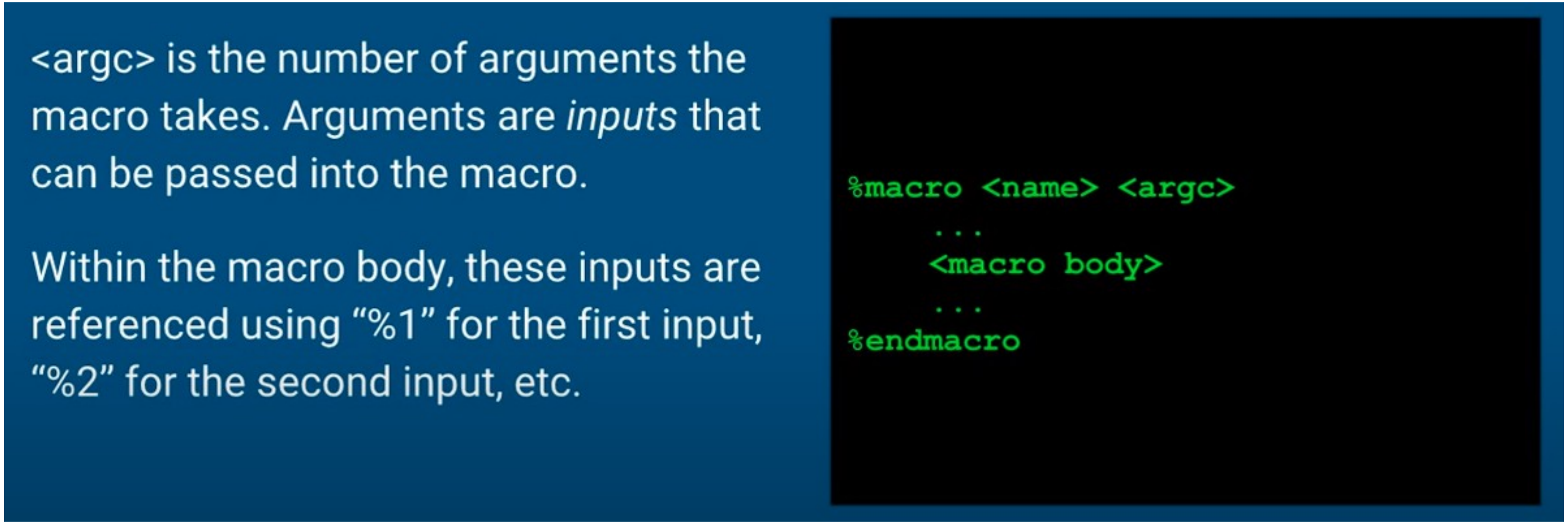
```
%macro exit 0
        mov rax, 60
        mov rdi, 0
        syscall
%endmacro
```

"Exit" Macro



```
%macro exit 0
        mov rax, 60
        mov rdi, 0
        syscall
%endmacro
```

Arguments for Macros


<argc> is the number of arguments the macro takes. Arguments are *inputs* that can be passed into the macro.

Within the macro body, these inputs are referenced using "%1" for the first input, "%2" for the second input, etc.

```
%macro <name> <argc>
     . . .
     <macro body>
     . . .
%endmacro
```

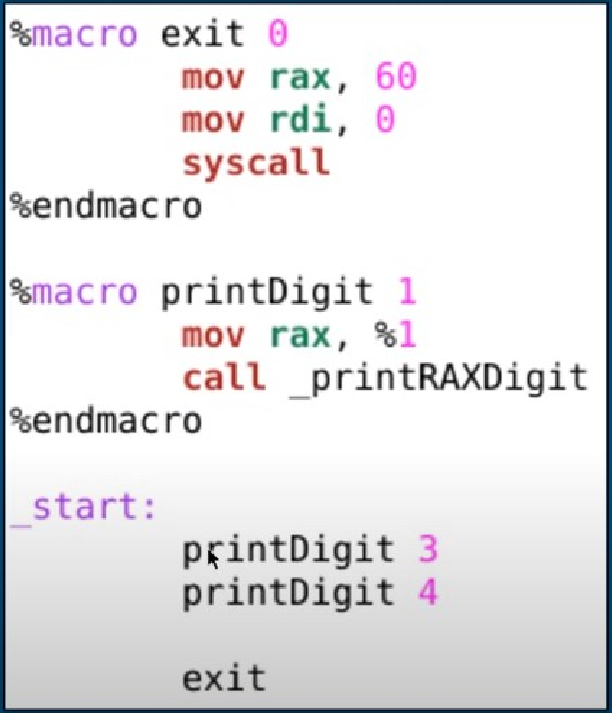
Arguments for Macros

For the “printDigit” macro, argc is 1 because it takes 1 argument (the digit).

For the “exit” macro, argc is 0 because it takes no arguments.

When we use “printDigit” in code under _start, we specify a number after it, that is our first argument.

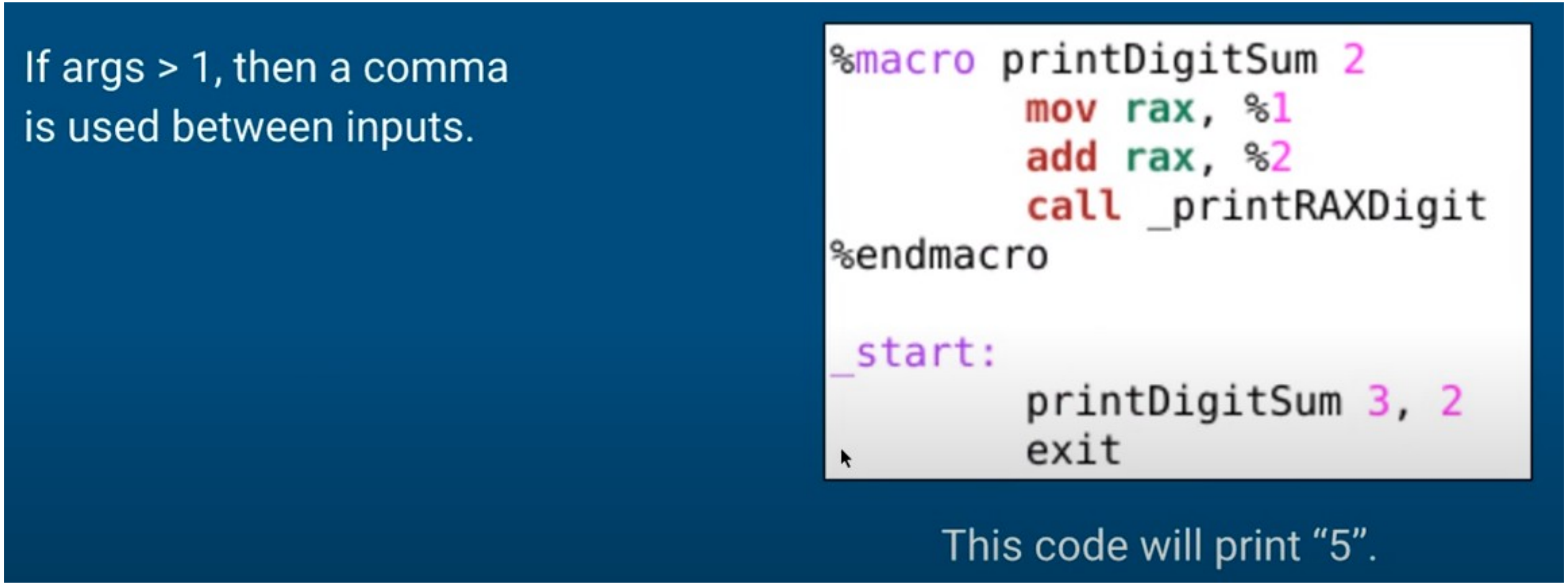
When we use “exit” we specify no numbers after it because it takes no arguments.



```
%macro exit 0
        mov rax, 60
        mov rdi, 0
        syscall
%endmacro

%macro printDigit 1
        mov rax, %1
        call _printRAXDigit
%endmacro

_start:
        printDigit 3
        printDigit 4

        exit
```

This code will print “3” then “4”.

Arguments for Macros

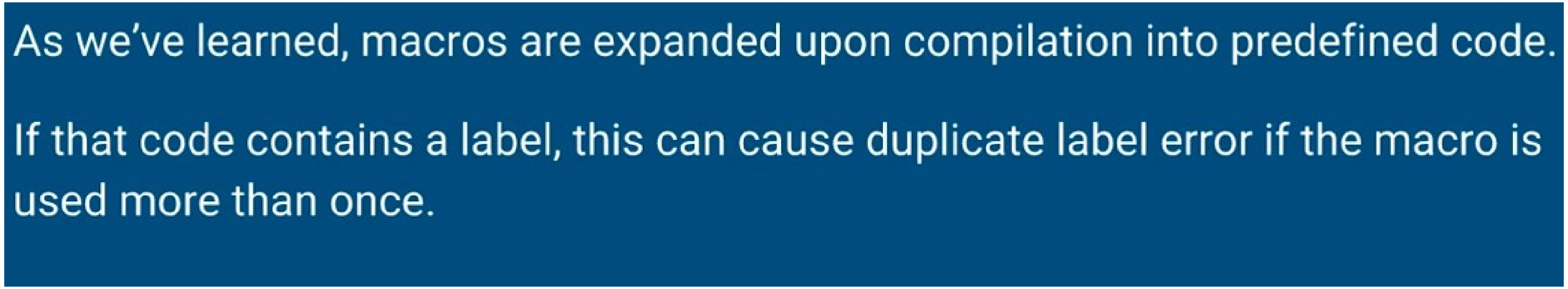


If args > 1, then a comma is used between inputs.

```
%macro printDigitSum 2
        mov rax, %1
        add rax, %2
        call _printRAXDigit
%endmacro

_start:
        printDigitSum 3, 2
        exit
```

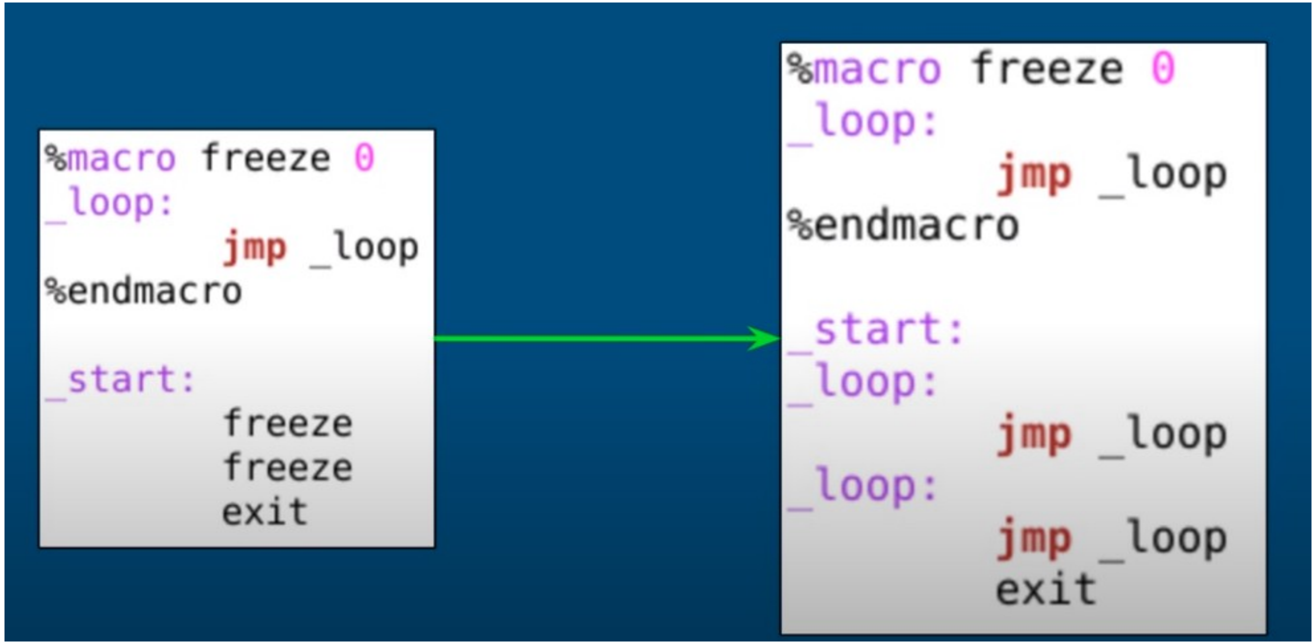
This code will print “5”.

Local Labels in Macros


As we've learned, macros are expanded upon compilation into predefined code.

If that code contains a label, this can cause duplicate label error if the macro is used more than once.

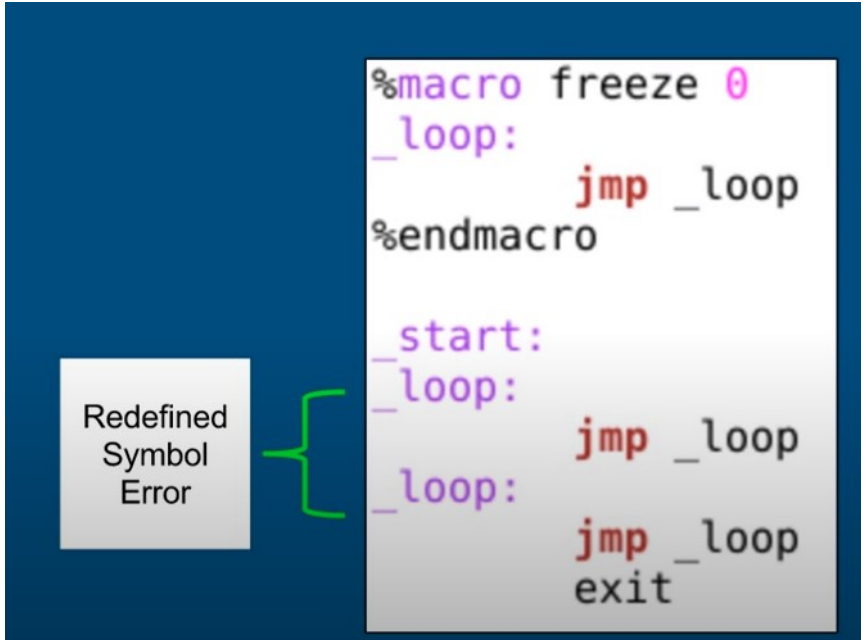
Local Labels in Macros



```
%macro freeze 0
_loop:
        jmp _loop
%endmacro

_start:
        freeze
        freeze
        exit
```

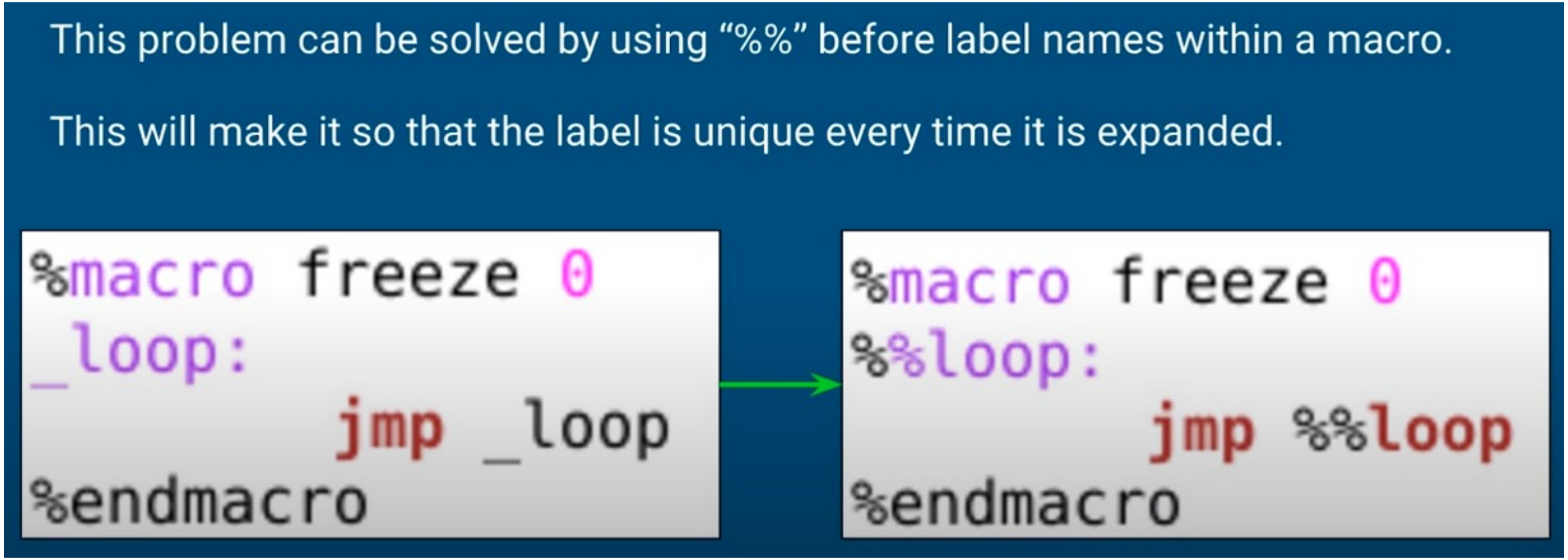
```
%macro freeze 0
_loop:
        jmp _loop
%endmacro

_start:
_loop:
        jmp _loop
_loop:
        jmp _loop
        exit
```

Local Labels in Macros



```
%macro freeze 0
_loop:
        jmp _loop
%endmacro

_start:
_loop:
        jmp _loop
_loop:
        jmp _loop
        exit
```

Local Labels in Macros



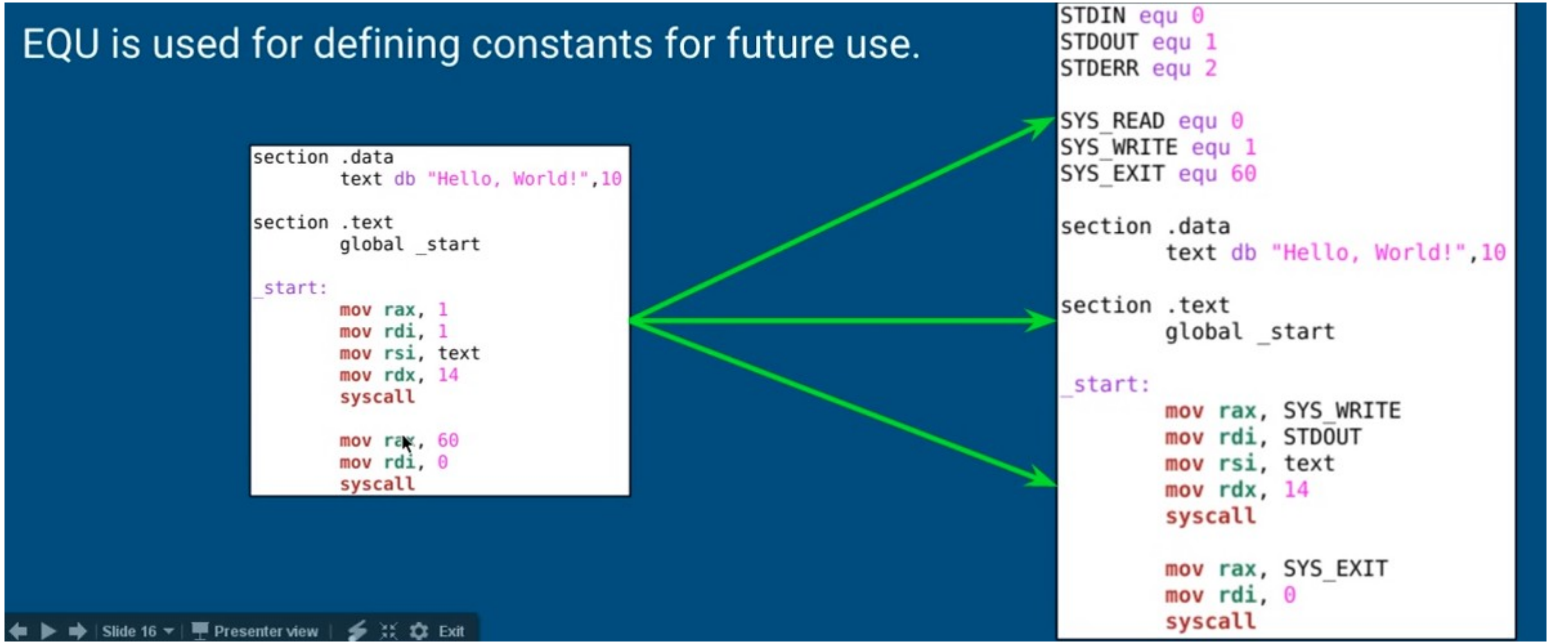
This problem can be solved by using “%%” before label names within a macro.

This will make it so that the label is unique every time it is expanded.

```
%macro freeze 0
_loop:
        jmp _loop
%endmacro
```

```
%macro freeze 0
%%loop:
        jmp %%loop
%endmacro
```

Defining Values with EQU

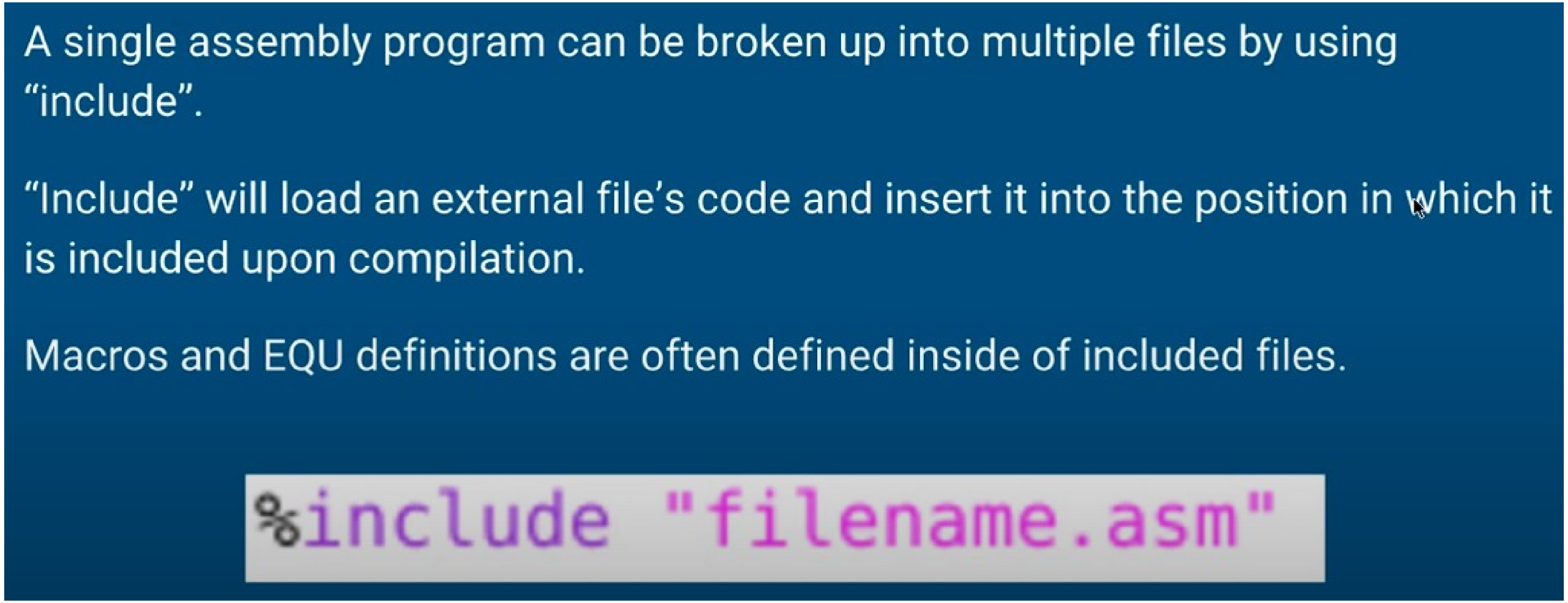


EQU is used for defining constants for future use.

```
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
```

```
STDIN equ 0
STDOUT equ 1
STDERR equ 2

SYS_READ equ 0
SYS_WRITE equ 1
SYS_EXIT equ 60

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

section .text
        global _start

_start:
        mov rax, SYS_WRITE
        mov rdi, STDOUT
        mov rsi, text
        mov rdx, 14
        syscall

        mov rax, SYS_EXIT
        mov rdi, 0
        syscall
```

Including External Files



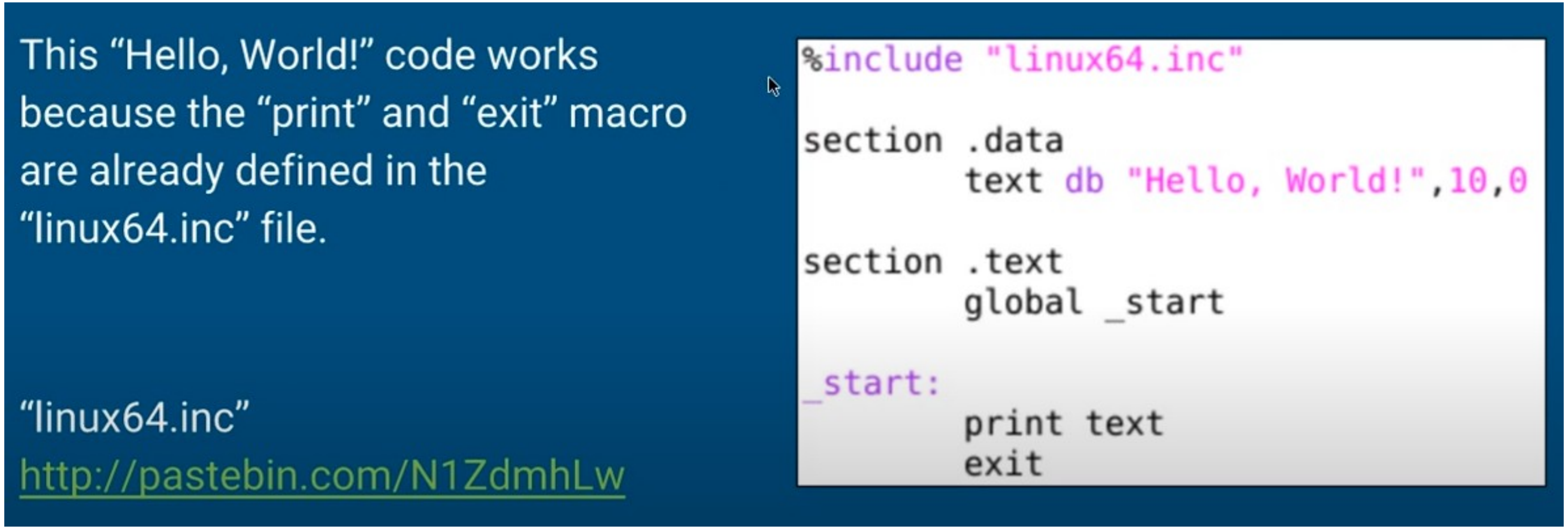
A single assembly program can be broken up into multiple files by using "include".

"Include" will load an external file's code and insert it into the position in which it is included upon compilation.

Macros and EQU definitions are often defined inside of included files.

```
%include "filename.asm"
```

Including External Files



This “Hello, World!” code works because the “print” and “exit” macro are already defined in the “linux64.inc” file.

“linux64.inc”
http://pastebin.com/N1ZdmhLw

```
%include "linux64.inc"

section .data
        text db "Hello, World!",10,0

section .text
        global _start

_start:
        print text
        exit
```

References

- https://www.youtube.com/watch?v=mRTax0MLaok&list=PLetF-YjXm-sCH6FrTz4AQhfH6INDQvQSn&index=7