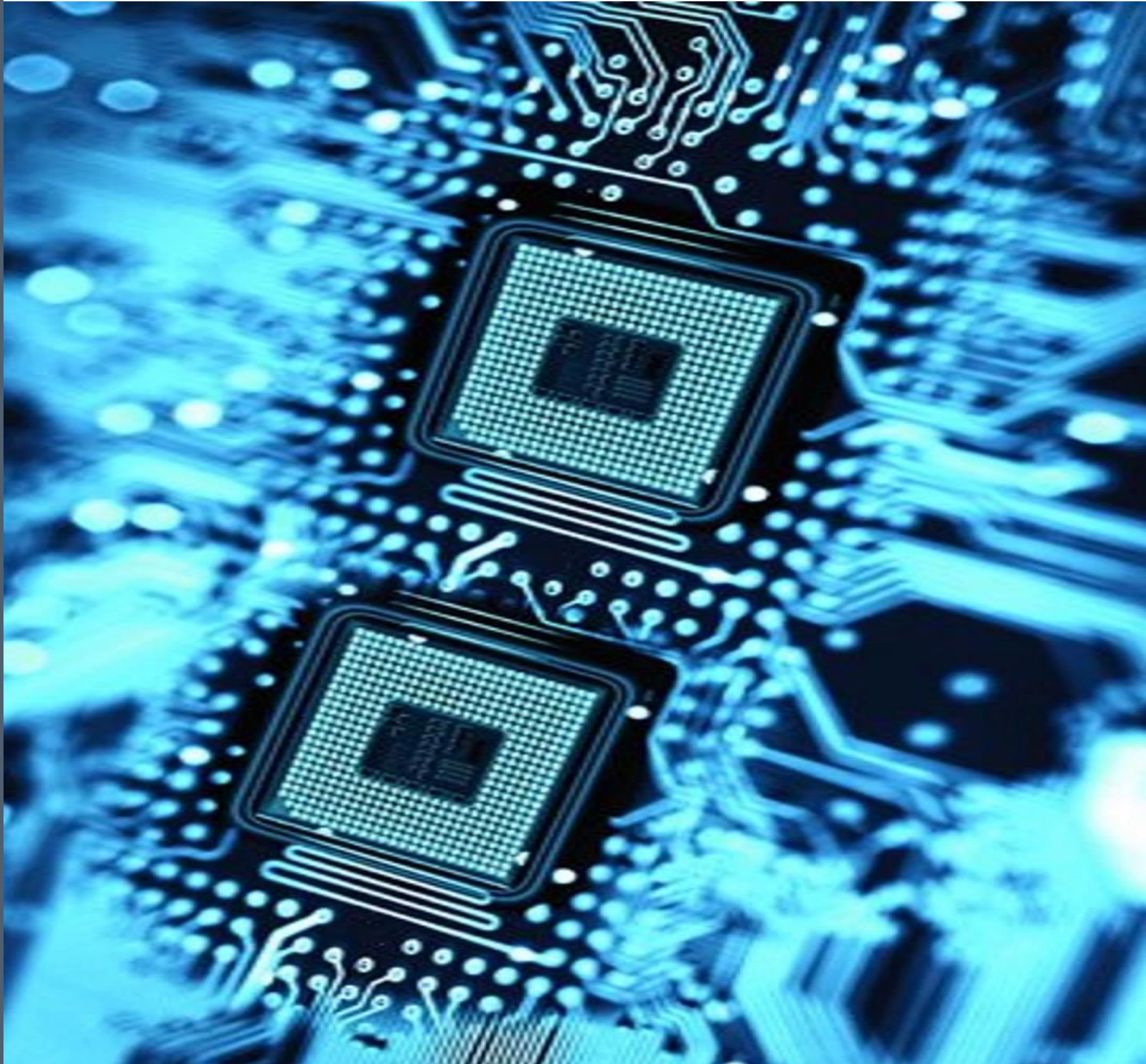
Computer organization & architecture

Course by: Dr. Ahmed Sadek

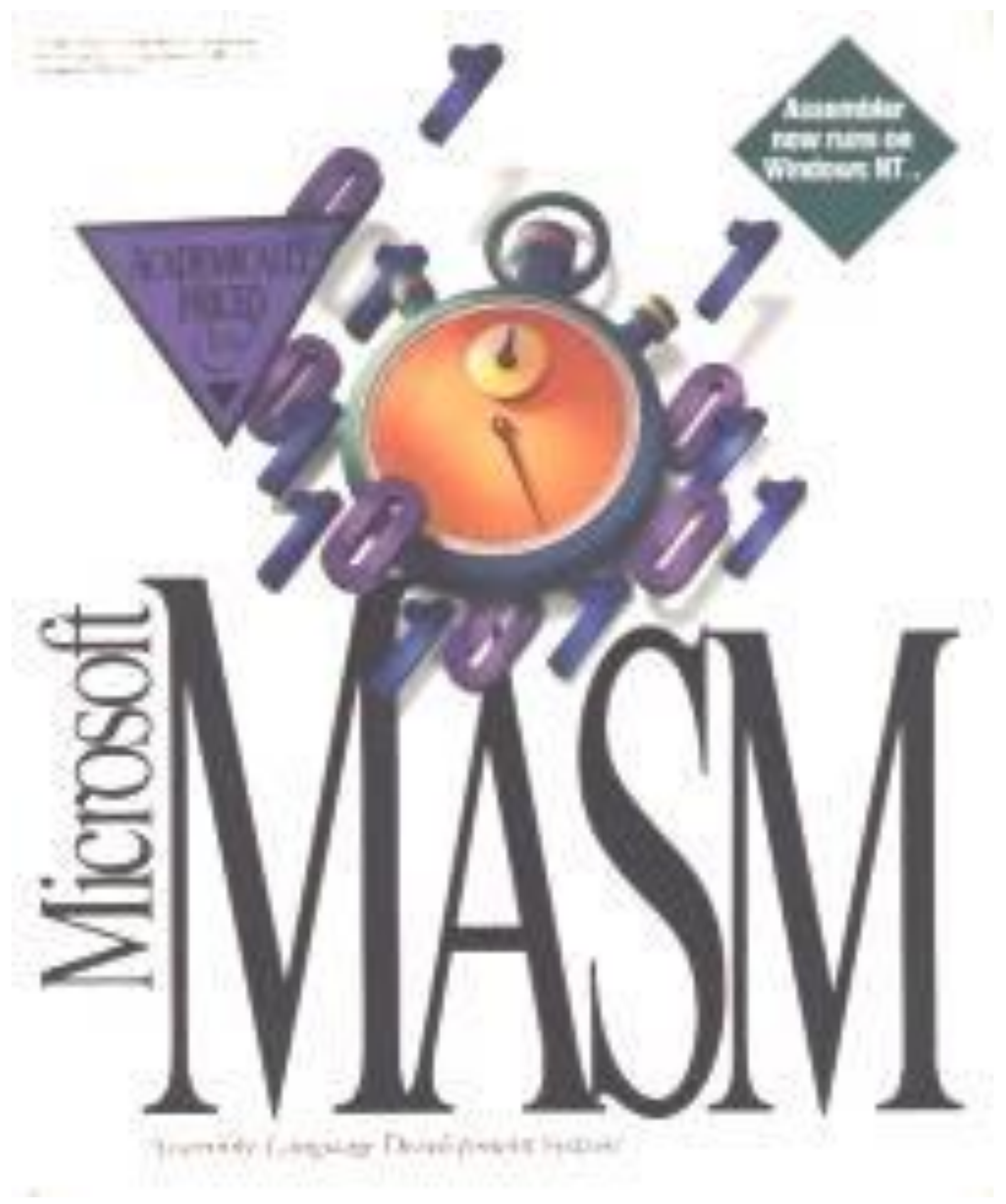
Lab By: Mahmoud Badry

Assembly Language Fundamentals

Chapter 3



About Chapter



- In this chapter, you will learn how to **define** and **declare variables** and **constants**, using Microsoft Assembler **(MASM)** syntax.
- We also can use **Emu8086** but some **difference** occurs.

Basic Elements of Assembly Language

Chapter 3, Section 1

Integer Constants

- An **integer constant** is made up of an **optional** leading **sign**, one or more **digits** and an **optional suffix** character(called a **radix**) indicating the number's **base**:

```
[{+I- }] digits [radix]
```

- IF there's no radix, **decimal** is **default**.
- **Radix**:

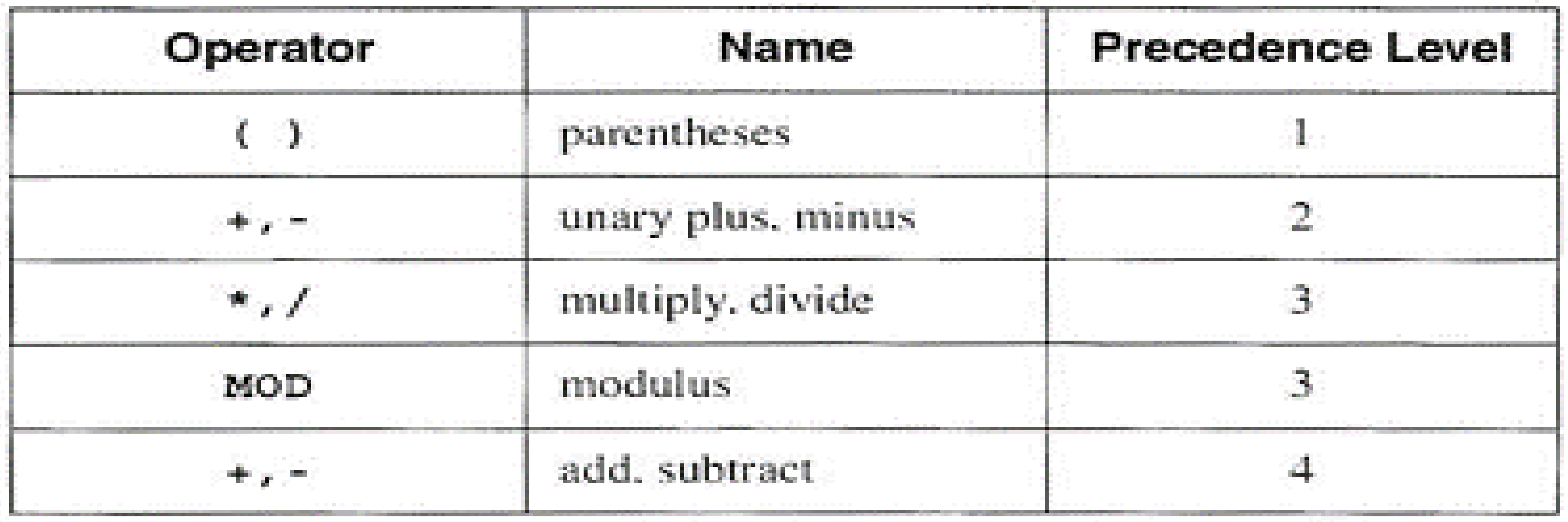
H | h	hexadecimal
(G|q)|(o|O)	octal
D | d	decimal
B | b	binary

Integer Constants

- A **hexadecimal constant** beginning with a **letter** must have a **leading zero** to prevent the **assembler** from interpreting it as an **identifier**.
- Examples:

```
26
26d
10101111b
1Ah
0A3h
```

Integer Expressions



Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

- An **integer expression** is a mathematical expression **involving integer** values and **arithmetic** operators. The expression must **evaluate** to an **integer** which can be stored in **32** bit (**16bit** in our case).

Character Constants

- A **character constant** is a **single character** enclosed in either **single** or **double quotes**.
- The **assembler** converts it to the **binary ASCII** code matching the **character**.
- Examples:

  'A‘

  “d”

String Constants

- A **string constant** is a string of **characters** enclosed in either **single** or **double quotes**.
- **Embedded quotes** are **permitted** when used in the **manner**.
- Examples:

  "Goodnight , Gracie"

  '4096 ‘

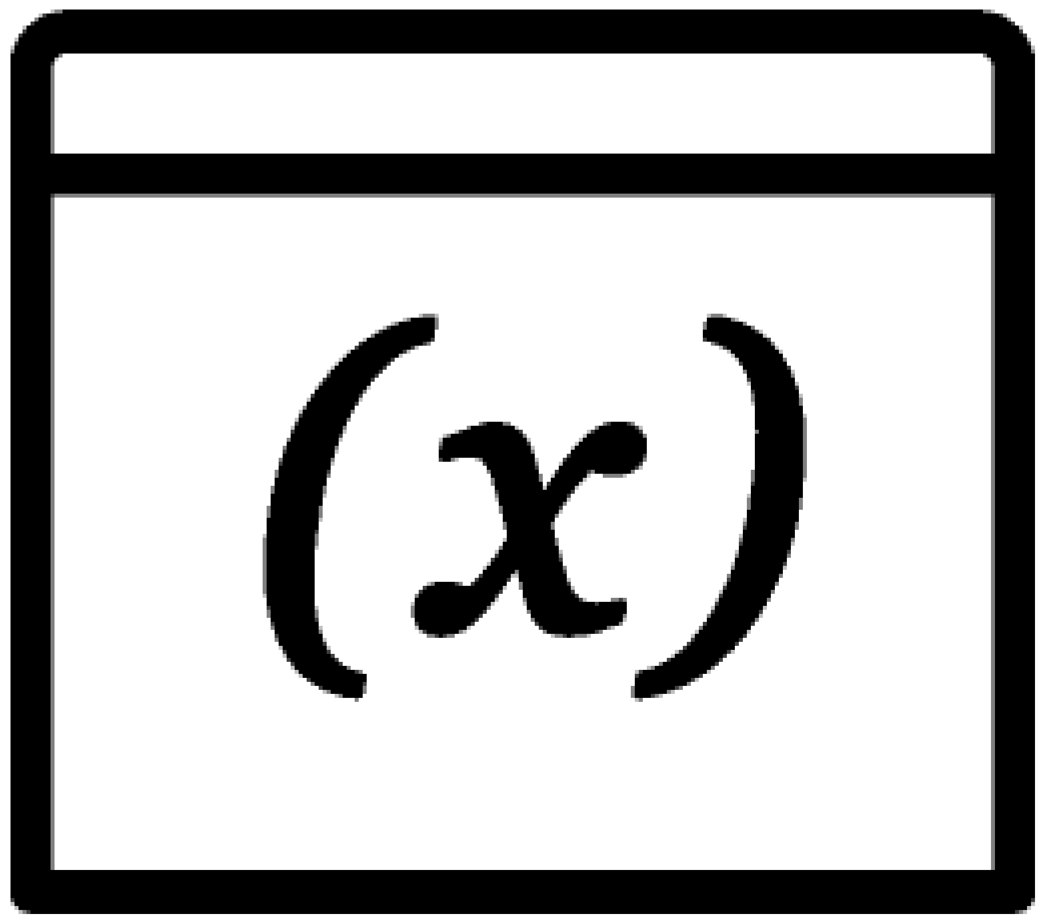
  "This isn't a test"

  'Say "Goodnight," Gracie’

Reserved Words

- Are **List** of **words** that have **special** meaning and can only be used in their **correct context**. Some of these words:
  - Instruction **mnemonics**, such as **MOV**, **ADD** or **MUL**, which correspond to **built-in** operations performed by Intel processors.
  - **Directives** ,which tell MASM how to **assemble** programs or a specific command that can only run on that assembler.
  - **Attributes**, which provide size and usage information for **variables** and **operands**. Examples are **BYTE** and **WORD**.
  - **Operators**, used in constant **expressions**.
  - **Predefined symbols** such as @data, which return constant integer values at assembly time.

Identifiers



- An **identifier** is a **programmer chosen** name. It might identify a **variable**, A **constant**, a **procedure**, or a code **label**.
- Rules:
  - between **I** and **247 characters**.
  - They are **not case-sensitive**.
  - The **first character** must be either a letter **(A..Z.a..z), _, @@, or $.** Subsequent characters may also be digits.
  - An **identifier cannot** be the same as an assembler **reserved word**.
- Examples:

```
var1
main
@@myfile
$first
```

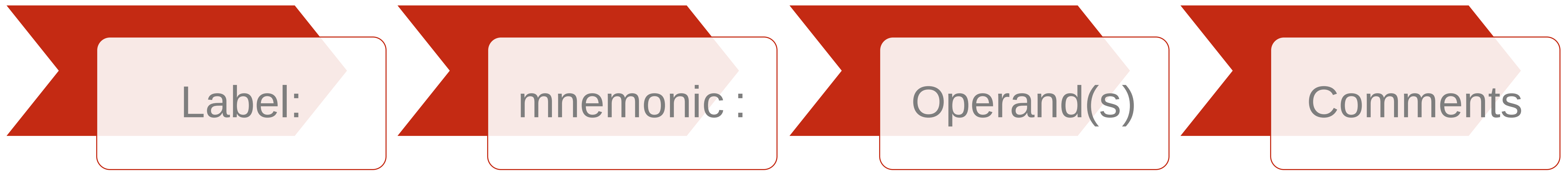
Directives

- A **directive** is a **command** that is **recognized** and **acted** upon by the **assembler** as the program's source code is **being assembled**.
- **Directives** are **part** of the **assembler's syntax**, but are **not related** to the Intel **instruction set**.
- Directives **aren’t case sensitive**.
- Example:
    - The **.DATA** directive **identifies** the **area** of a program that **contains variables**.
    - The **.CODE** directive **identifies** the **area** of a program that **contains instructions**.
    - The **PROC** directive **identifies** the **beginning** of a **procedure**. Name may be any identifier:

```
name  PROC
```

Instructions

- An **instruction** is a **statement** that is **executed** by the **processor** at runtime after the **program** has been **loaded** into **memory** and **started**.
- An **instruction** contains **four** basic parts:
  - **Label** (optional)
  - **Instruction mnemonic** (required)
  - **Operand(s)** (usually required)
  - **Comment** (optional)



Label

- A **label** is an **identifier** that acts as a **place marker** for either **instructions** or **data**.
- In the **process** of **scanning** a source program, the **assembler assigns** a **numeric address** to each program **statement**. A label placed just before an instruction implies the **instruction's address**. Similarly, a label placed just before a variable implies the **variable's address**.

Label

- **Code Labels**:
  - A **label** in the **code** area of a program **must end** with a **colon** (:) character.

```
target: mov ax,bx
……
jmp target
```

- **Data Labels**:
  - If a **label** is used in the **data** area of a program, it **cannot end** with a **colon**.

```
first BYTE 10
```

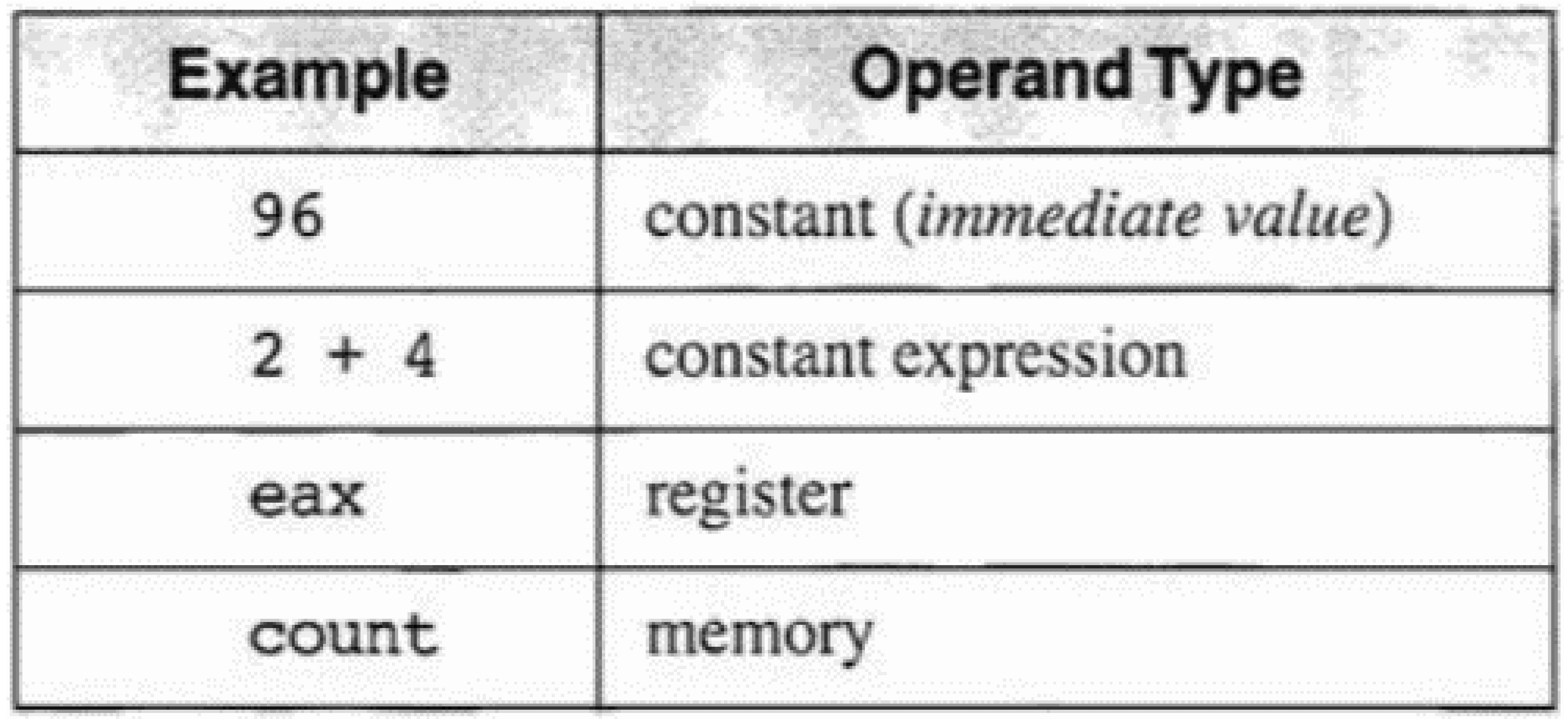
Instruction Mnemonic

- An **instruction mnemonic** is a **short** word that **identifies** the **operation** carried out by an **instruction**.
- Some Mnemonics:

mov	Move (assign) one value to another
add	Add two values
sub	Subtract one value from another
mul	Multiply two values
jmp	Jump to a new location
call	Call a procedure

- We will **talk** about each one **soon**.

Operands



Example	Operand Type
96	constant (*immediate value*)
2 + 4	constant expression
eax	register
count	memory

- An **assembly language** instruction can **have** between zero and **three** operands, each of which can be a **register**, **memory** operand, **constant** expression, or **I/O port**.
- Example:

```
Stc                     ; set Carry flag
inc ax                  ;add 1 to ax
mov count,bx            ;move BX to count
```

Comments

- **Comments**, as you **probably know**, are an important way for the **writer** of a **program** to communicate **information** about how the program works to a **person reading** the source **code**.
- **Comments** can be specified in **two** ways:
    - **Single-line** comments, beginning with a semicolon character (**;**)

```
;This is a comment
```

- **Block** comments, beginning with the **COMMENT** directive and a **user-specified symbol**.

```
COMMENT !
This line is a comment.
This line is also a comment. !
```

Adding three integers program

```
org 100h
; This program adds and
subtracts 32-bit integers.
.code
jmp main

main proc
    mov ax,100h
    add ax,400h
    add ax,400h
    jmp exit
main endp
exit: ret
END
```

Program template

```
;Program Description:
;Author:
;Creation date:
;Revisions:
;Date:                ;Modified by:
.data
;Insert variables here
.code
JMP main
main PROC
;Insert your code here
JMP Exit
main ENDP
;(insert additional procedures here)
Exit: ret
END
```

THANKS