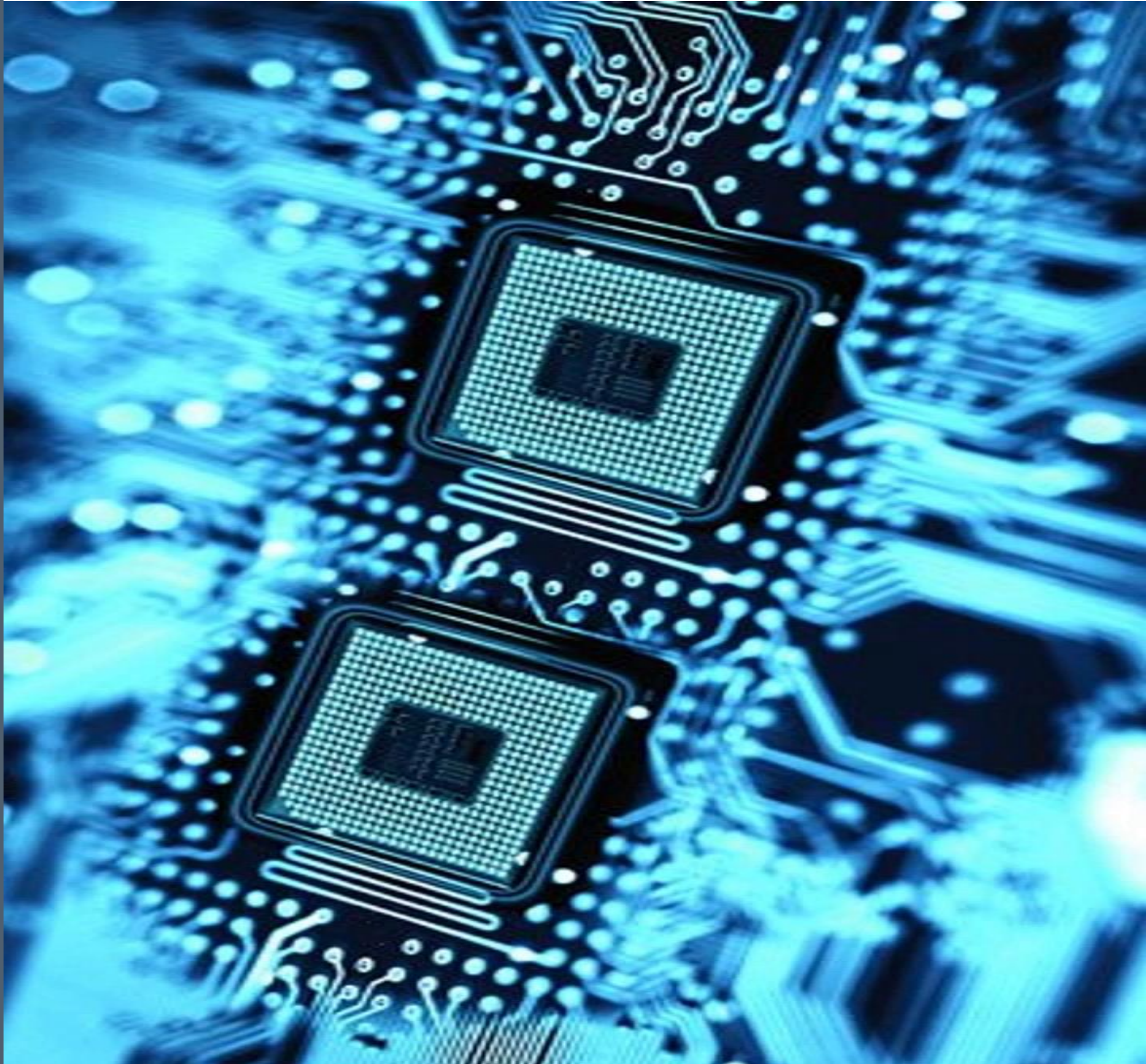
Computer organization & architecture

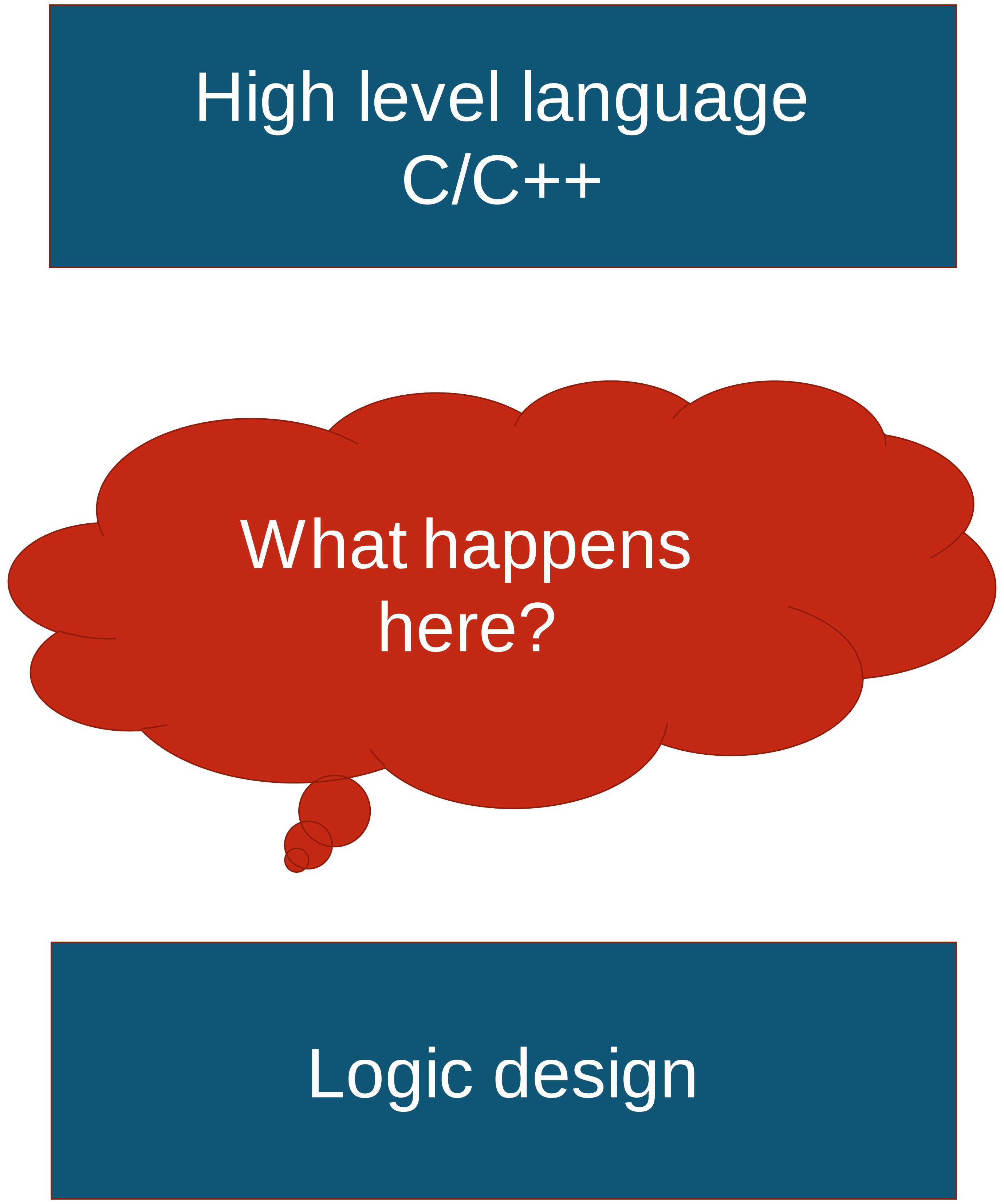
Course by: Dr. Ahmed Sadek

Computer organization & architecture

An introduction.

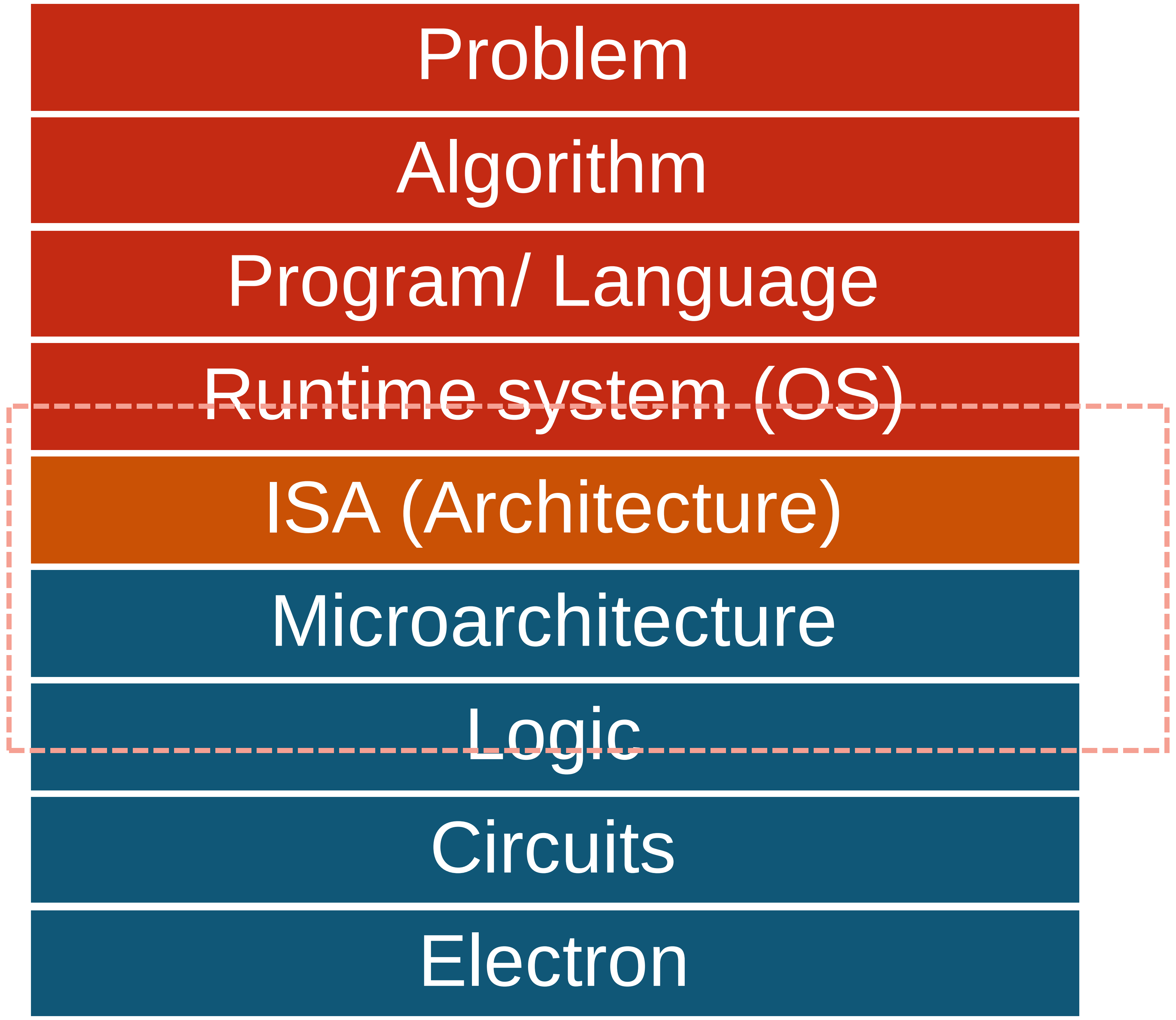


Introduction



- A **program** written by a **high level** language is translated into an **assembly language** before it can be executed by the **computer H/W**.

Introduction



- **Computer architecture** goals:
  - What is **ISA** and how can the computer H/W **understand** it.
  - How is a **computer designed** using **logic gates** and wires to satisfy specific **goals**? Automatic V.S Manual parallelism

Abstraction

- Regarding **binary search** algorithm:
  - Can **Java** implement it?
  - Can **C** implement it?
  - Can **Assembly** implement it?
  - Can **machine language** implement it?

- **Abstraction** reduces the effort!

Why we learn computer architecture?

Why we learn computer architecture?

Three reasons:

Hardware perspective

You must learn this course to create your **own** computer **hardware**. **Computer** includes all **types** like PC, laptop, mobile, tablet, etc.

Why we learn computer architecture?

Three reasons:

What if:

- You’ve created a **program** correctly but it **runs slowly**?
- You’ve created a **program** that takes so **much energy**?
- You’ve created a **program** that **doesn’t run correctly**?

Software perspective

Why we learn computer architecture?

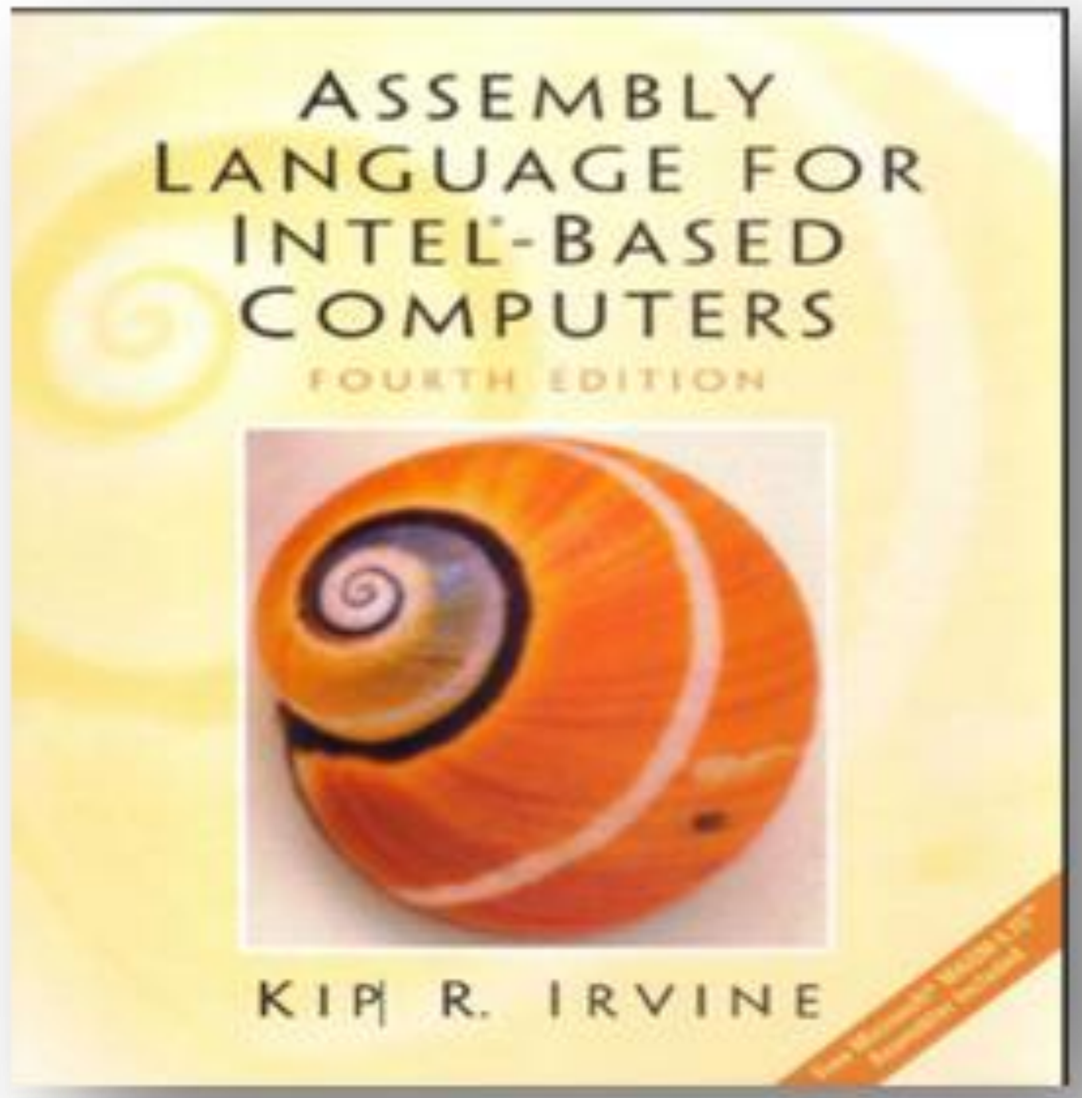
Three reasons:

Assembly Language

Creating an **operating system** or a **compiler**.
Creating **embedded systems**.
Games developers that need to take care of all **audio** and **video** drivers.
Debugging a **program** that you don't have its **source code**.
Cracking programs (**Illegal** :D)
Assembly language programmer can **earn more** than programmers who can not write assembly language (in those **applications** where **assembly language** is **required**). As there are few assembly programmers their salary are high.
And much more!

Course information

Course information



- **Instructor**: Dr. Ahmed Sadek
- **Assistant**: Mahmoud Badry, Nedaa
- **Lecture book**: “Computer Organization And Architecture” by William Stallings
- **Lab book**: “Assembly language for INTEL-based computers”, 4th edition , by KIP R. I RV I N E
  - book **focuses** on programming **Intel** microprocessors, specifically members of the Intel **IA-32 processor family [Intel 80386** to **Pentium 4]**
- **Lab software**: Emu8086.
- **Project will be on**: MDA 8086 trainer kit
- **Github link**: https://github.com/mbadry1/FCIFayoum-Computer-architecture-2018
- Assignments **deadline** are **one week**.
- **Quizzes will be reported before they can be taken.**

Welcome to Assembly Language

Chapter 1, Section 1

Some Good Questions to Ask

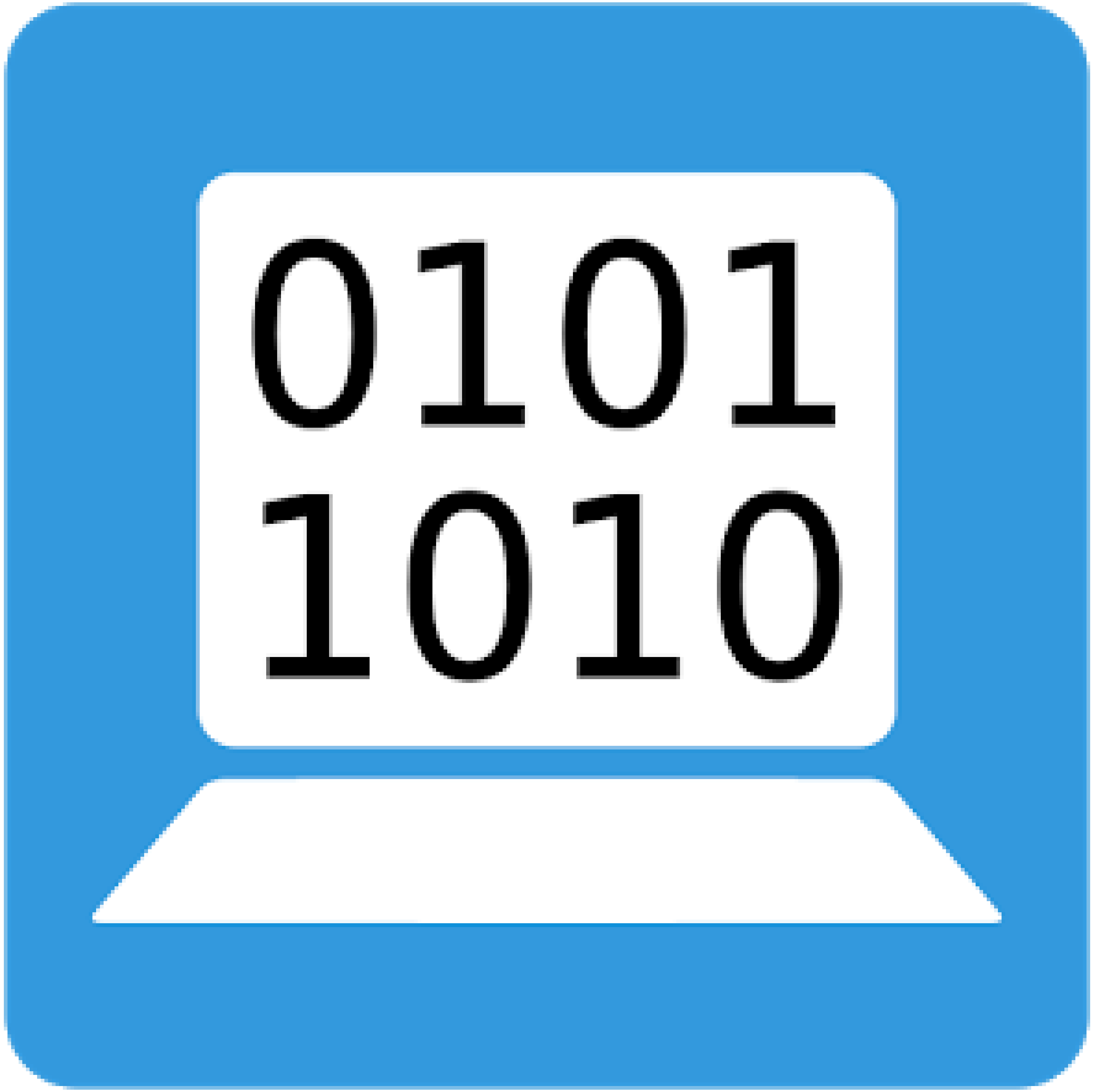
- **What is Assembly language?**
- Assembly language is the **oldest programming language**, and of all languages.
- It provides **direct access** to a computer's **hardware**, making it necessary for you to understand a great **deal** about your **computer's architecture.**
- **What background should I have?**
- you should have completed a single college course or its equivalent in **computer programming**. Like c/c++, C#, Java
- **How do C++ and Java relate to assembly language?**
- A single **statement** in C++ (Or Java) **expands** into **multiple assembly** language or **machine instructions**.
- **What is an assembler?**
- An **assembler** is a program that **converts source-code** programs from **assembly language** into machine language.
- Two of the most popular **assemblers** for the Intel family are **MASM** (Microsoft Assembler) and **TASM** (Borland Turbo Assembler)
- **What is a Linker?**
- A linker **combines individual files** created by an assembler into a **single executable** program.
- A **third program**, called a **debugger** provides a way for a programmer to **trace** the **execution** of a program and **examine** the contents of **memory**.

Some Good Questions to Ask

- What **hardware** and **software** do I need?
- You need a computer with an **Intel386**, **Intel486**, or One of the **Pentium** processors.
- Software:
  - **OS**: Windows, MS-DOS, or even Linux with DOS emulator.
  - **Editor**: To write assembly code.
  - **Assembler**: Like MASM
  - **Linker**.
  - **Debugger**: MASM supplies a good 16-bit debugger named **CodeView**

Data Representation

Chapter 1, Section 2

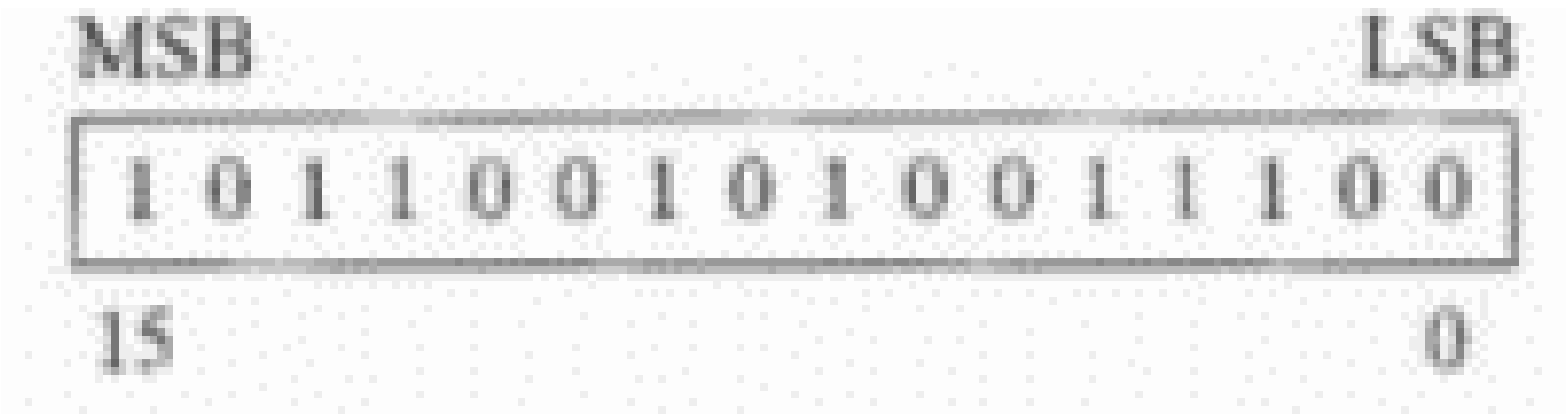


Number systems

System	Base	Possible Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

- **Before** we can begin to **discuss** computer organization and **assembly** language , we need a common mode of **communication** with **numbers**.
- The following **table** defines **number systems** and it **basis**

Binary Numbers



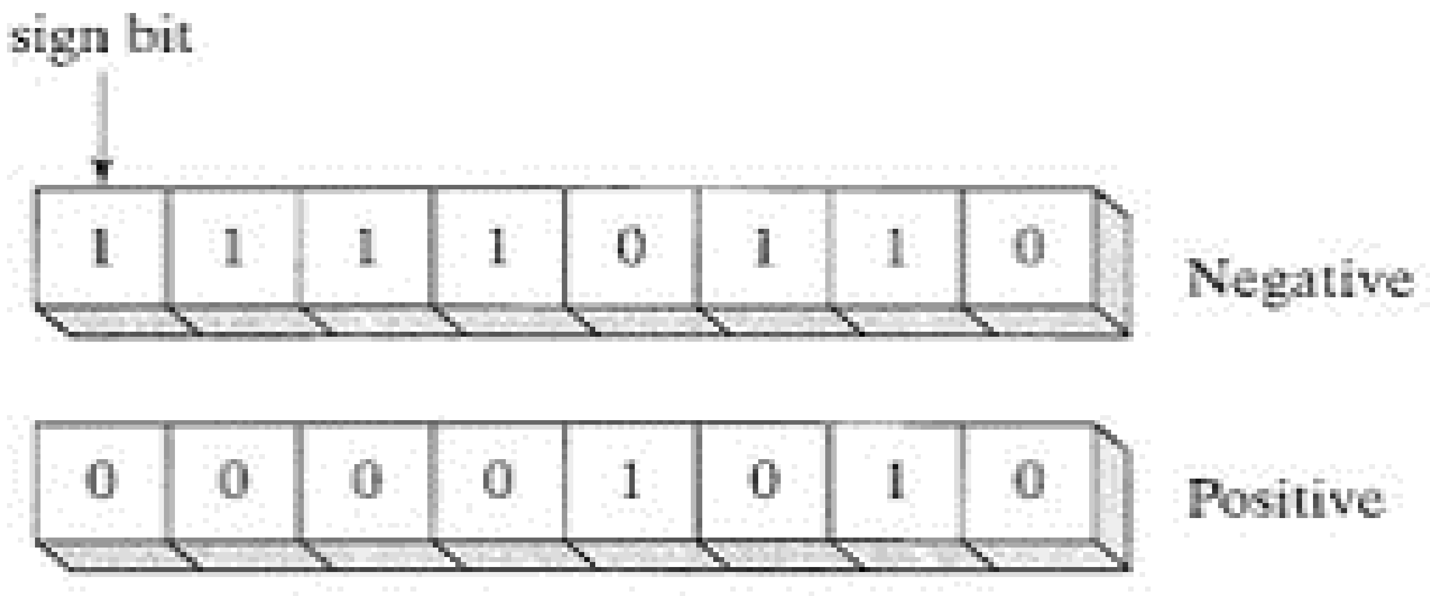
- **Binary** numbers are base **2** numbers in which each binary digit (called a **bit)** is either a **0** or a **1**.
- The **bit** on the left is called the most significant bit (**MSB**), and the bit on the right is the least significant bit (**LSB**).
- **Binary** integers can be either **signed** or **unsigned**, also it can represent **real numbers**.
- Now lets try:
  - Translating **Unsigned Binary** Integers to **Decimal**.
  - Translating **Unsigned Decimal** Integers to **Binary**.
  - Binary **Addition**.
  - Converting Unsigned **Hexadecimal** to **Decimal**
  - Converting Unsigned **Decimal** to **Hexadecimal**
  - Converting **Hex** to **binary** and **reverse**

Integer Storage Sizes

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65.535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4.294.967.295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18.446.744.073.709.551.615	0 to $(2^{64} - 1)$

- One **kilobyte** (KB) is equal to 2^10 or 1,024 bytes.
- One **megabyte(MB**) is equal to 2^20 1,048,576 bytes.
- One **gigabyte(GB**) is equal to 2^ 30, or 1024^3, or 1.073,741.824 bytes.
- One **terabyte** (TB) is equal to 2^40 or 1024^4 or 1.099,511.627.776 bytes.
- One **petabyte** is equal to 2^50 or I.125.899 .906,842.624bytes.
- One **Exabyte** is equal to 2^60 or 1,152.921.504.606,846,976 bytes.
- One **zettabyte** is equal to 2^70
- One **yottabyte** is equal to 2^80

Signed Integers



- **MSB** indicates the **number's sign**, **0** indicates that the integer is **positive**, and **1** indicates that it is **negative**.
- **Negative integers** are represented using what is called **two's complement** representation.

Two's Complement Notation

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two's complement representation	11111111

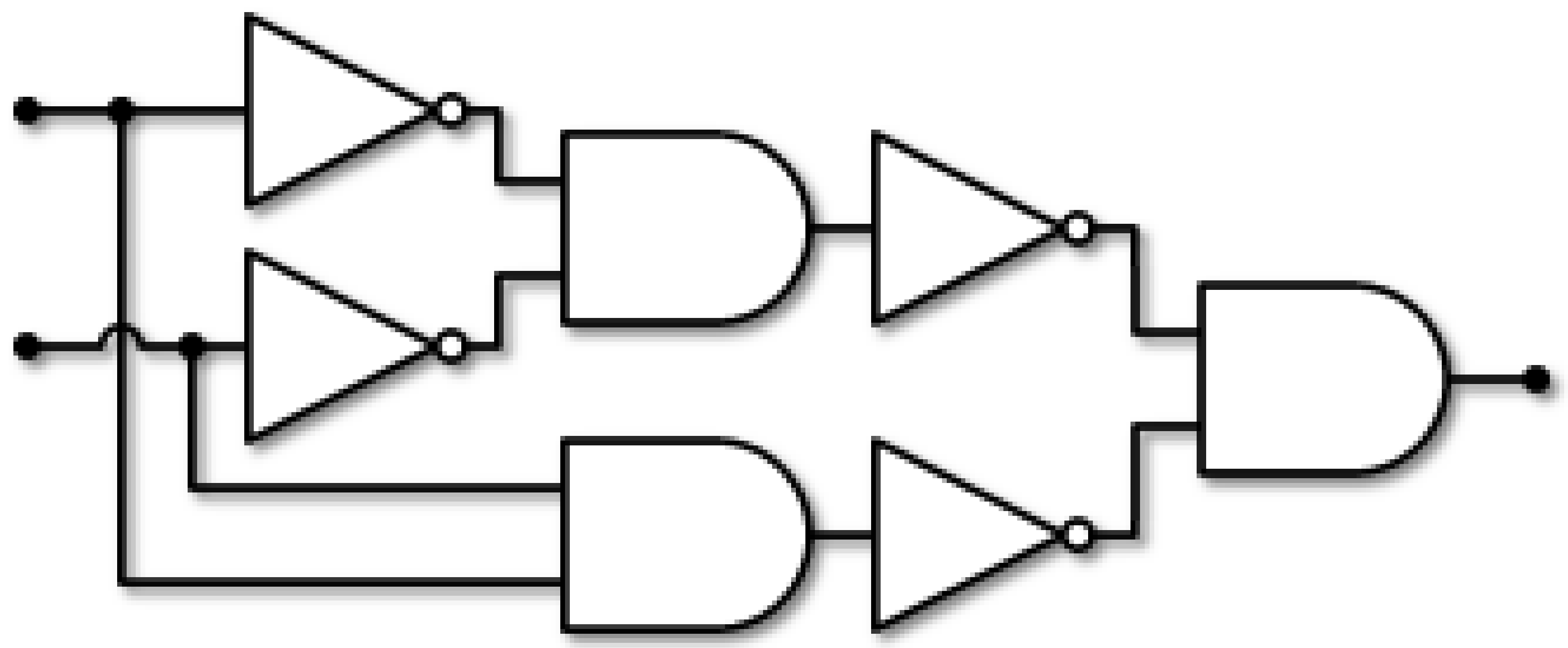
- The two 's complement of a **binary integer** is formed by **reversing** its **bits** and **adding 1**.
- What about **hexadecimal**?
- To convert **Signed Binary** or **Hexadecimal**, first detect the **sign**, if its **negative** make two's complement and then convert the remaining.

Character storage

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@
33	21	041	!	!	65	41	101	A	A
34	22	042	"	"	66	42	102	B	B
35	23	043	#	#	67	43	103	C	C
36	24	044	$	$	68	44	104	D	D
37	25	045	%	%	69	45	105	E	E
38	26	046	&	&	70	46	106	F	F
39	27	047	'	'	71	47	107	G	G
40	28	050	(	(	72	48	110	H	H
41	29	051	)	)	73	49	111	I	I
42	2A	052	*	*	74	4A	112	J	J
43	2B	053	+	+	75	4B	113	K	K
44	2C	054	,	,	76	4C	114	L	L
45	2D	055	-	-	77	4D	115	M	M
46	2E	056	.	.	78	4E	116	N	N
47	2F	057	/	/	79	4F	117	O	O
48	30	060	0	0	80	50	120	P	P
49	31	061	1	1	81	51	121	Q	Q
50	32	062	2	2	82	52	122	R	R
51	33	063	3	3	83	53	123	S	S
52	34	064	4	4	84	54	124	T	T
53	35	065	5	5	85	55	125	U	U
54	36	066	6	6	86	56	126	V	V
55	37	067	7	7	87	57	127	W	W
56	38	070	8	8	88	58	130	X	X
57	39	071	9	9	89	59	131	Y	Y
58	3A	072	:	:	90	5A	132	Z	Z
59	3B	073	;	;	91	5B	133	[	[
60	3C	074	<	<	92	5C	134	\	\
61	3D	075	=	=	93	5D	135	]	]
62	3E	076	>	>	94	5E	136	^	^
63	3F	077	?	?	95	5F	137	_	_

- Assuming that a **computer** can only store **binary data**, one might **wonder** how it could also **store characters**.
- To do this, it must **support** a certain **character set**, which is a **mapping** of **characters** to **integers**.
- **Character** sets used only **8 bits**. Because of the great **diversity** of **languages** around the world, the 16-bit **Unicode character** set was created.
- **ASCII** codes use only the **lower 7 bits** of every byte, Sometimes the **extra bit** is used to indicate that the **byte** is not an **ASCII character**, but is a **graphics symbol**, however this is **not defined** by **ASCII**.

Boolean Operations



- We all know about **NOT**, **AND** and **OR** operators and their **truth tables**.
- Operator **Precedence** is () ➔ NOT ➔ AND ➔ OR

Assignment

From the book Solve:

Section 1.3.7 (Pages 23-24-25):

- Points : 3, 5, 9, 11, 13, 16, 17, 21

Section 1.4.2 (page 29):

- Points: 6, 9

THANKS