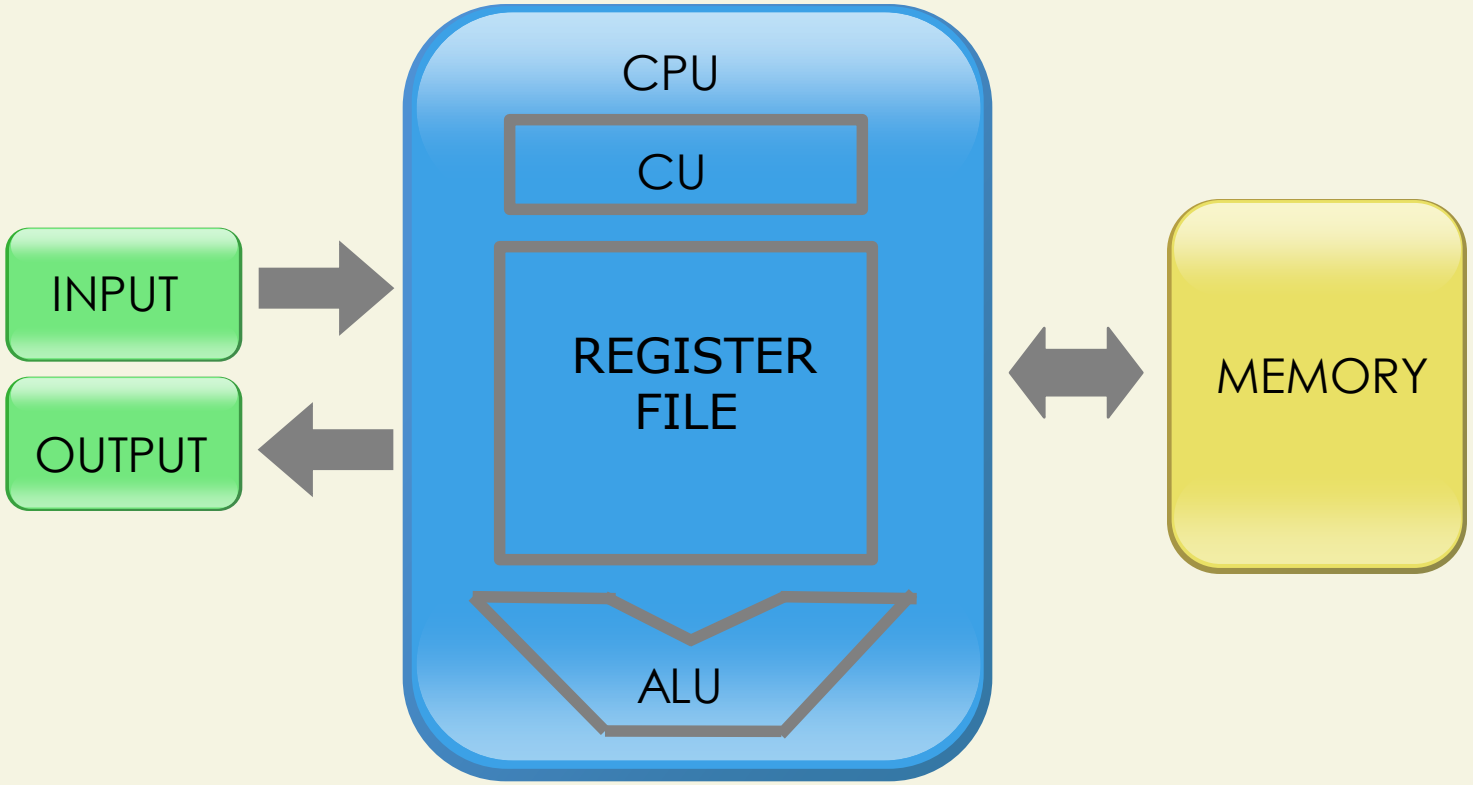
I. INTRODUCTION

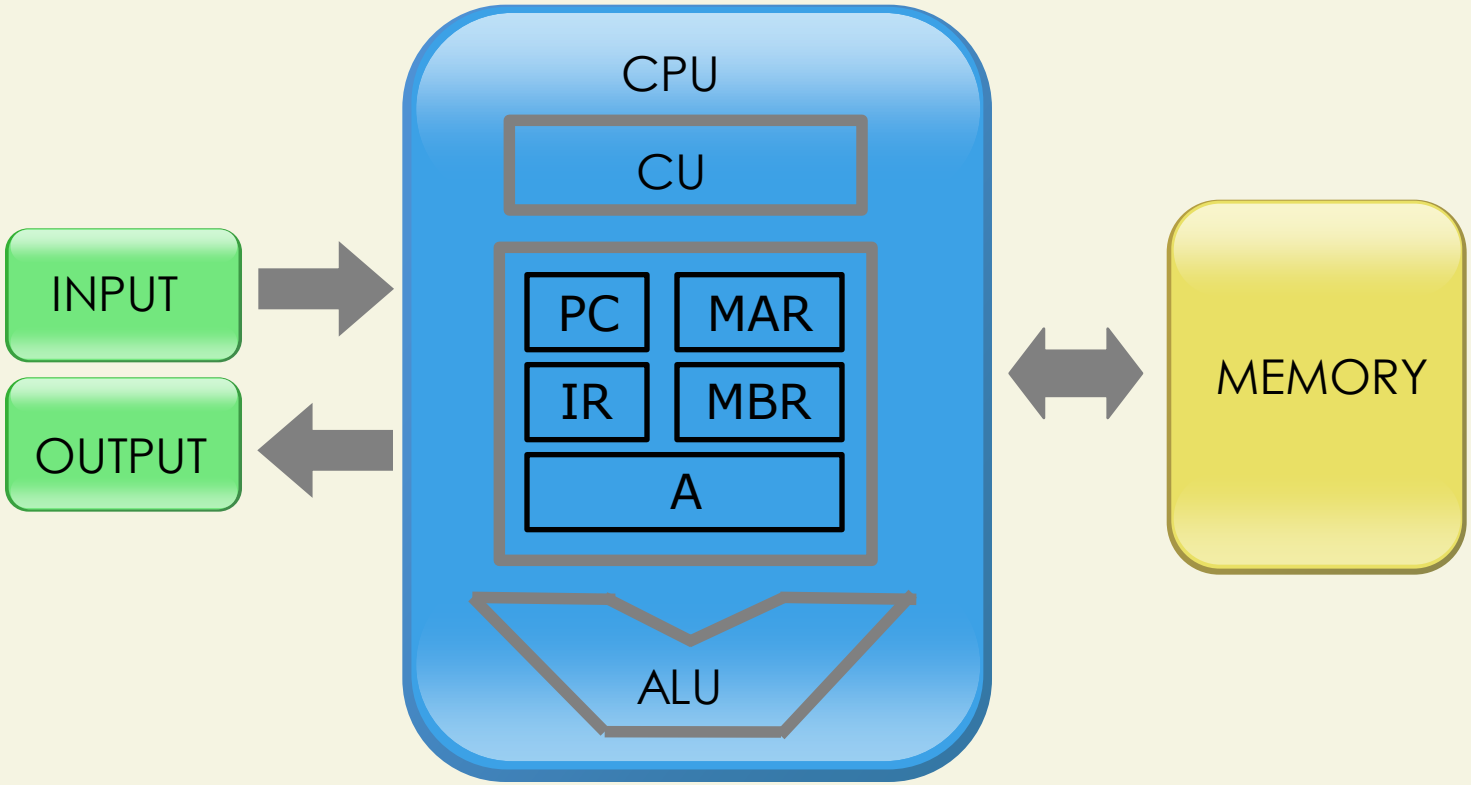
Microcomputer Systems:

Basic Computer Organization

The Basic Organization of a Microcomputer



Von Neumann's Simple Computer



The Fetch-Decode-Execute Cycle

1. Get the instruction from the memory using the address contained in PC.
2. Put the instruction into IR.
3. Increment the value in PC.
4. Decode the value in IR.
5. Execute the operation specified in the instruction.
6. Repeat step number 1.

I. INTRODUCTION

Assembly Programming Process

Outline

1. Assembly Programming Environment
2. Number Systems Conversion
3. Developing Assembly Programs

Assembly Programming Environment

- Assembler
  - a computer program for translating assembly language (a mnemonic representation of machine language) into object code.
  - TASM, MASM, **NASM**

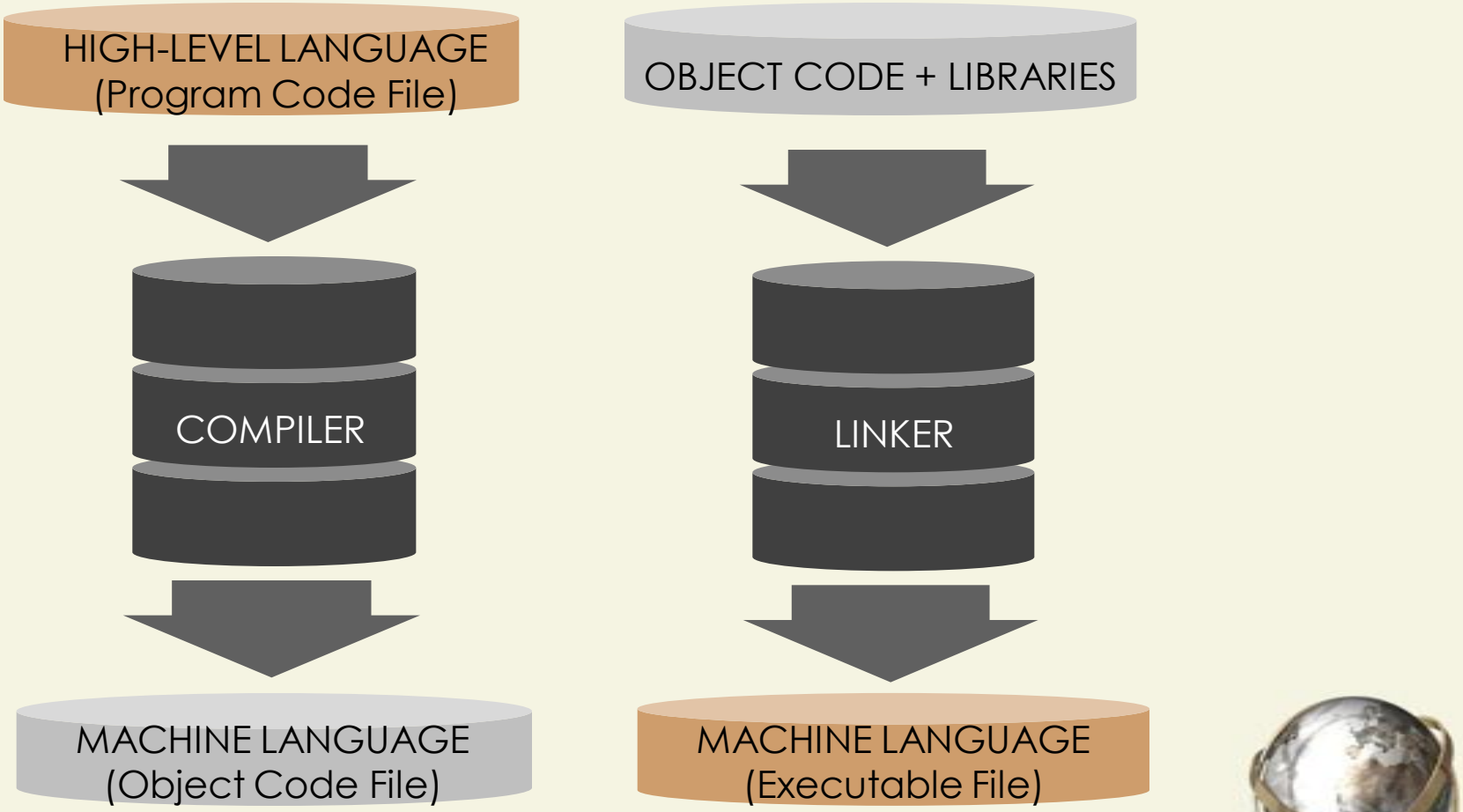
Assembly Programming Environment

- Linker
  - a program that combines libraries (modules) together to form an executable file
  - TLINK, MLINK, ALINK, **LD**

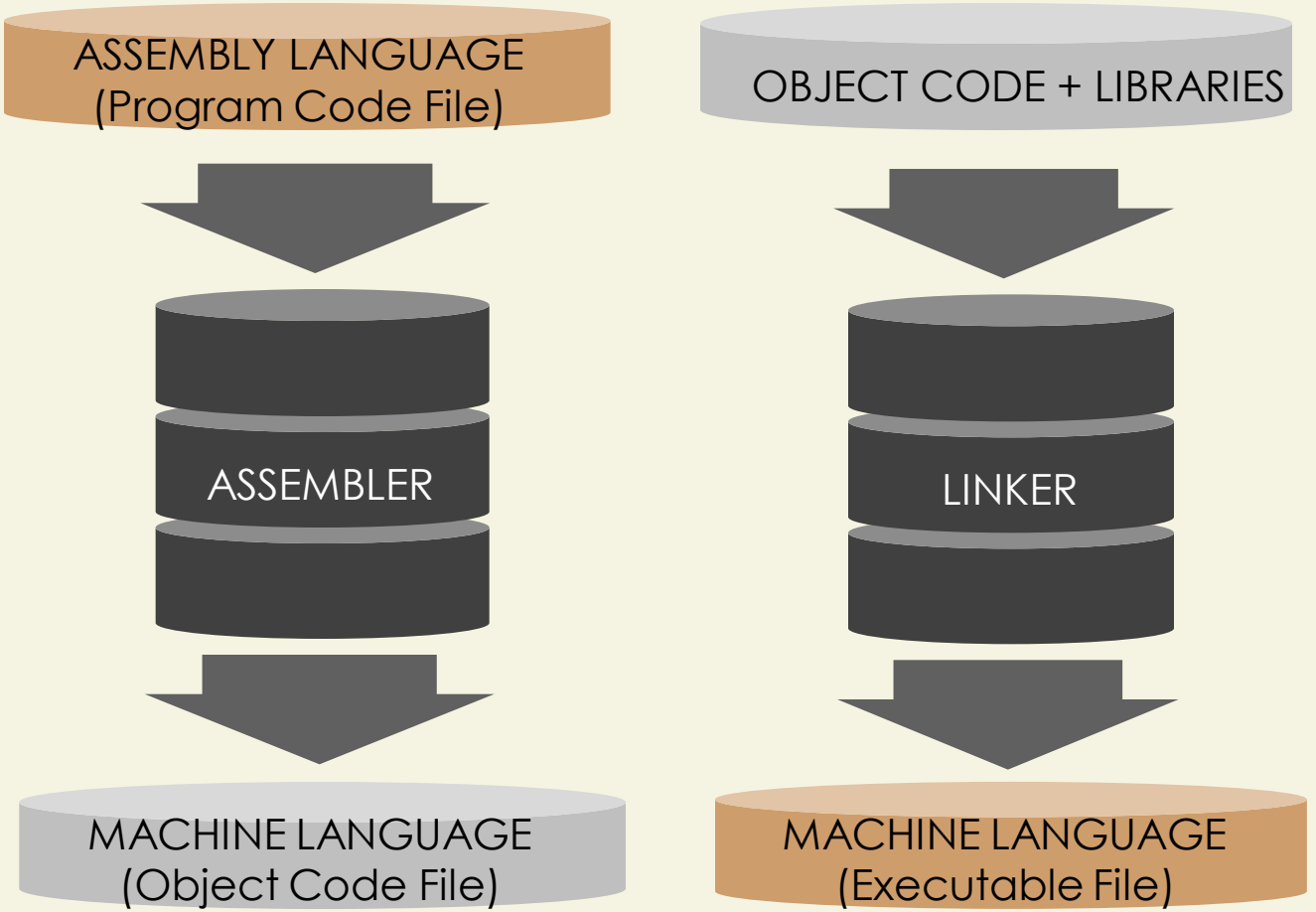
Assembly Programming Environment

- Disassembler
  - a computer program which translates machine language into assembly language, performing the inverse operation to that of an assembler

High-level PL to Executable Programs



Assembly to Executable Programs



Assembly Programming Environment

32-bit Assembly Programming

- x86 machine instructions

Linux

- Use Linux services and system calls

Executing Assembly Programs

nasm –f elf <file>.asm

- produces <file>.o

ld -o <file> <file>.o

- produces <file>.exe

./<file>

- run <file>

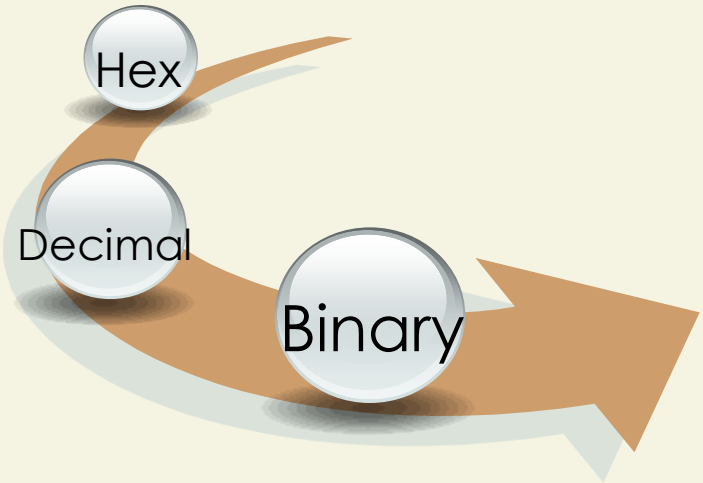
Review

Number Systems Conversion

Binary to Decimal

Binary → 0 0 1 0 0 1 1 0

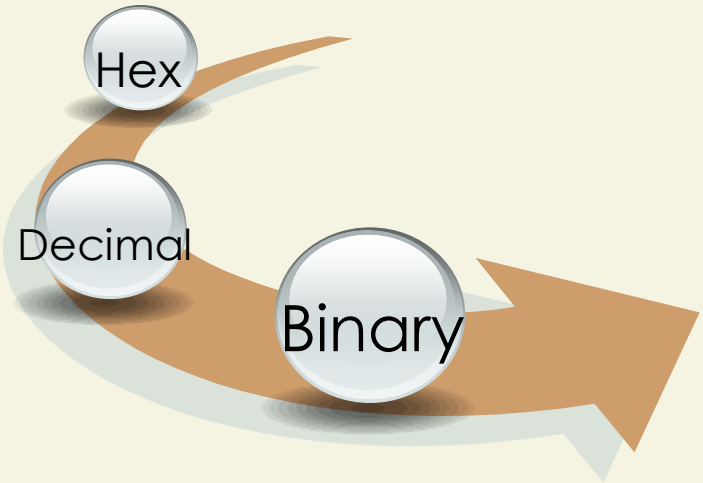
Decimal →



Binary to Decimal

128	64	32	16	8	4	2	1
Binary → 0	0	1	0	0	1	1	0

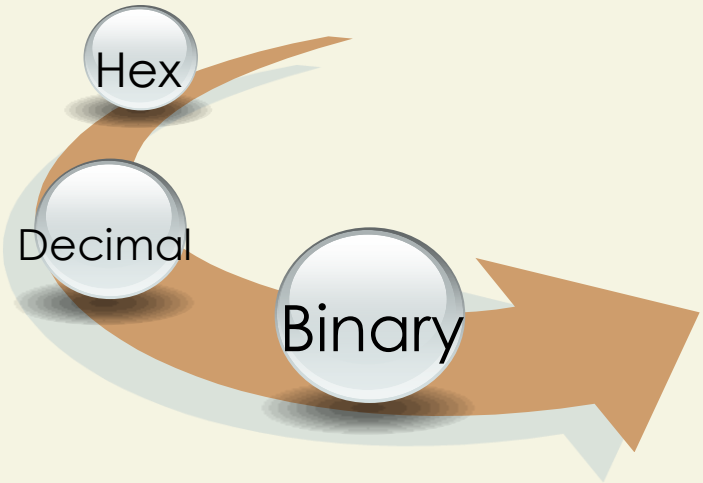
Decimal →



Binary to Decimal

128	64	32	16	8	4	2	1
Binary →	0	0	1	0	0	1	1	0

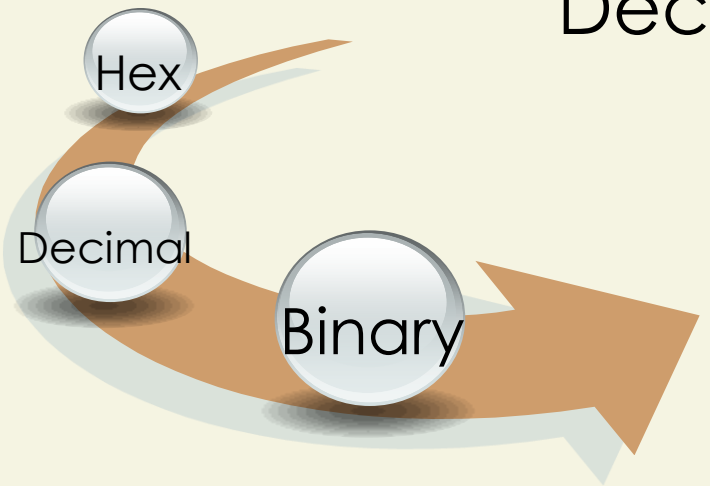
Decimal → 3 8



Binary to Decimal

128	64	32	16	8	4	2	1
Binary →	0	0	0	1	1	0	0	0

Decimal →

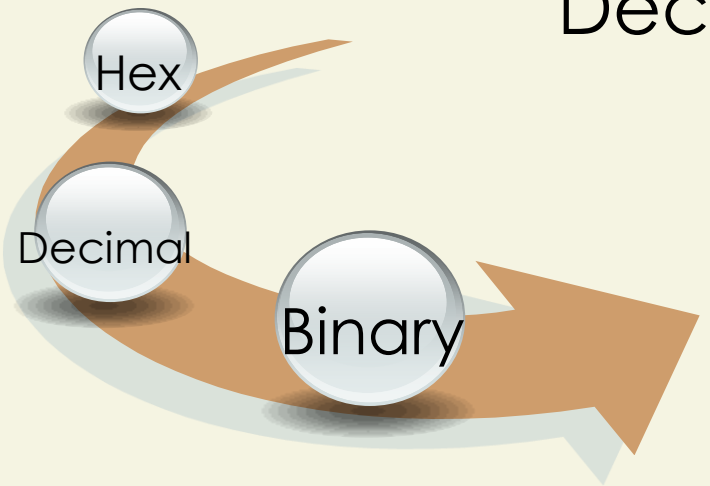


Binary to Decimal

128	64	32	16	8	4	2	1
0	0	0	1	1	0	0	0

Binary →

Decimal → 24

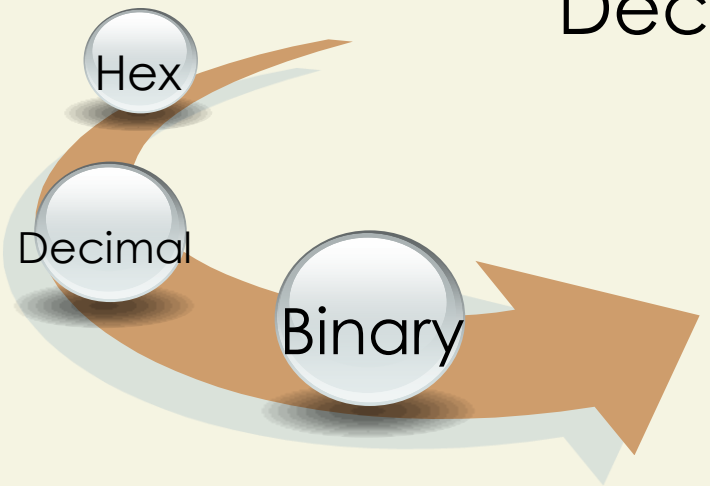


Decimal to Binary

128 64 32 16 8 4 2 1

Binary →

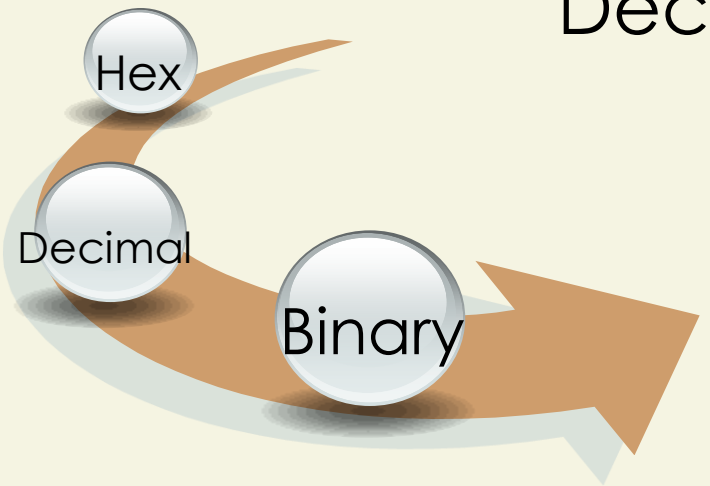
Decimal → 5 7



Decimal to Binary

128	64	32	16	8	4	2	1
Binary →	0	0	1	1	1	0	0	1

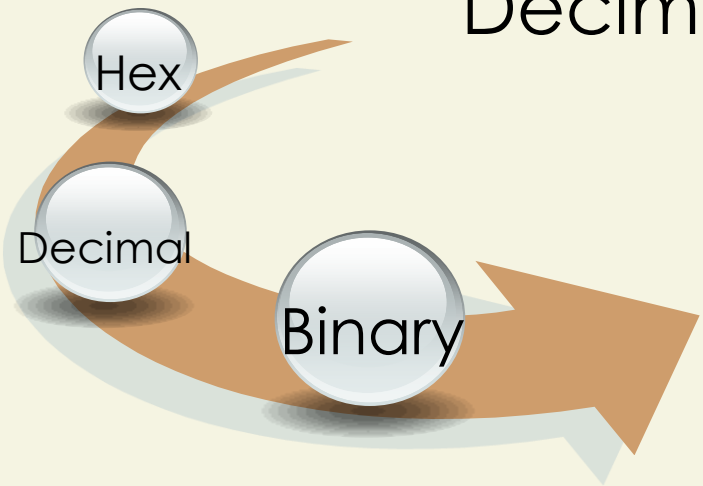
Decimal → **5** **7**



Binary to Decimal

128	64	32	16	8	4	2	1
Binary →	1	0	0	0	0	0	1	1

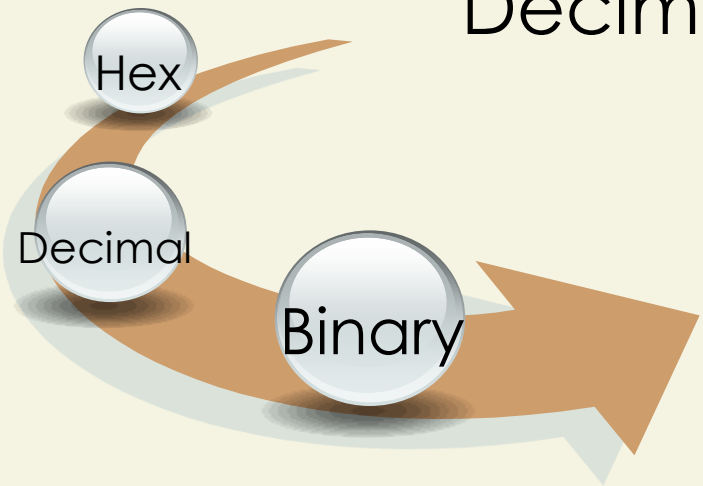
Decimal →



Binary to Decimal

128	64	32	16	8	4	2	1
Binary →	1	0	0	0	0	0	1	1

Decimal → **1** **3** **1**

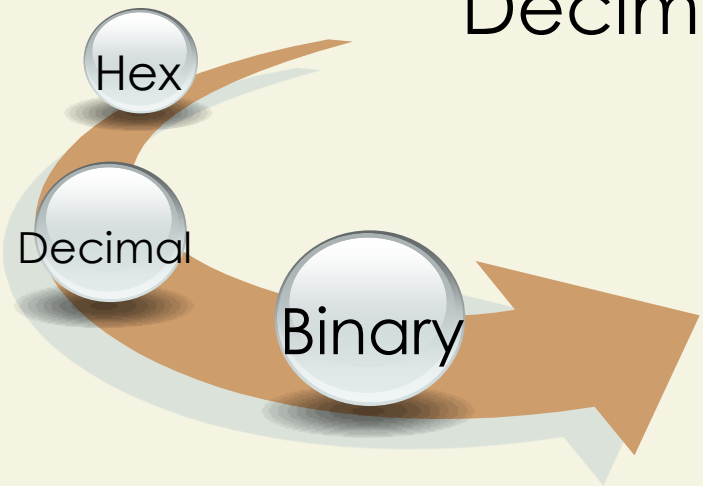


Decimal to Binary

128 64 32 16 8 4 2 1

Binary →

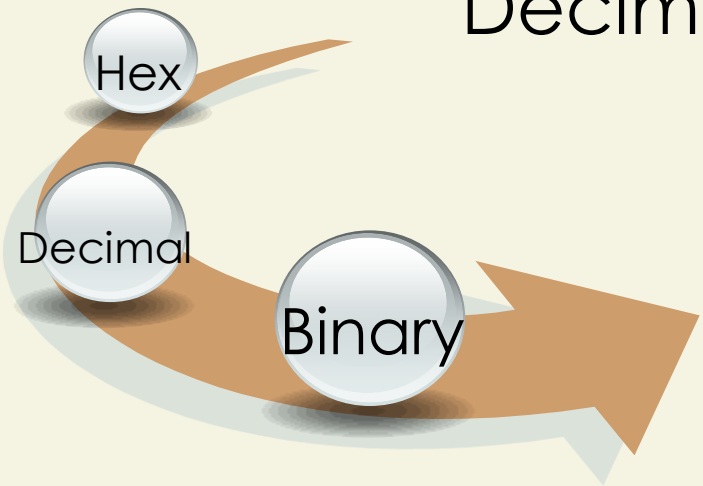
Decimal → 1 5 0



Decimal to Binary

128	64	32	16	8	4	2	1
Binary →	1	0	0	1	0	1	1	0

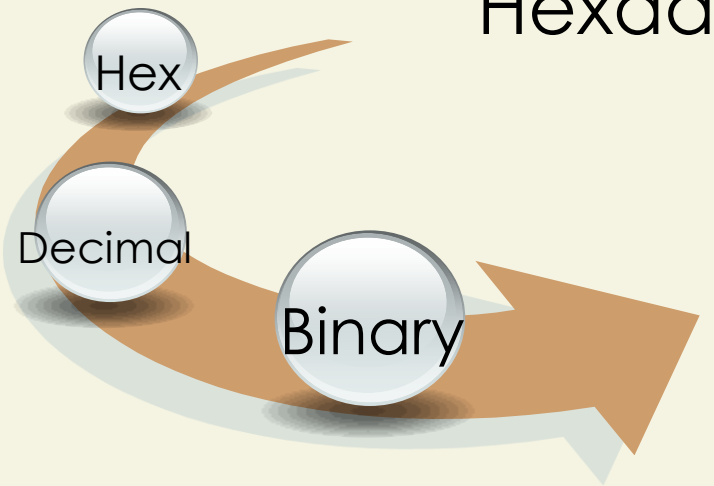
Decimal → **1 5 0**



Binary to Hexadecimal

Binary → 0 0 1 0 0 1 1 0

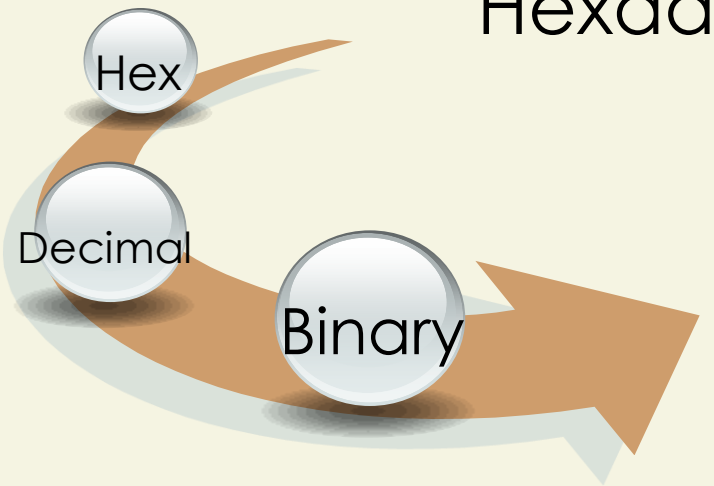
Hexadecimal →



Binary to Hexadecimal

8	4	2	1	8	4	2	1
Binary →	0	0	1	0	0	1	1	0

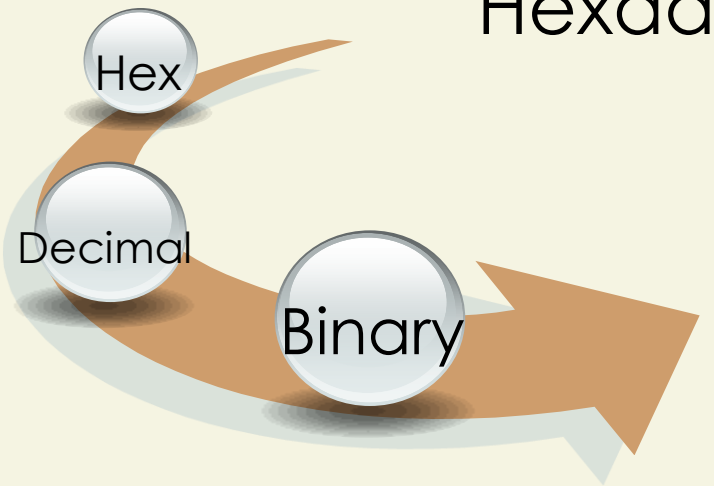
Hexadecimal →



Binary to Hexadecimal

8	4	2	1	8	4	2	1
Binary →	0	0	1	0	0	1	1	0

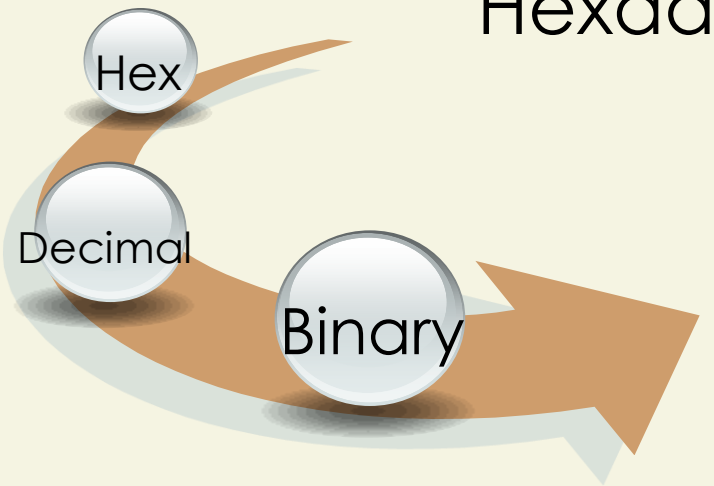
Hexadecimal → **2** **6**



Binary to Hexadecimal

8	4	2	1	8	4	2	1
Binary →	0	0	1	1	1	0	1	1

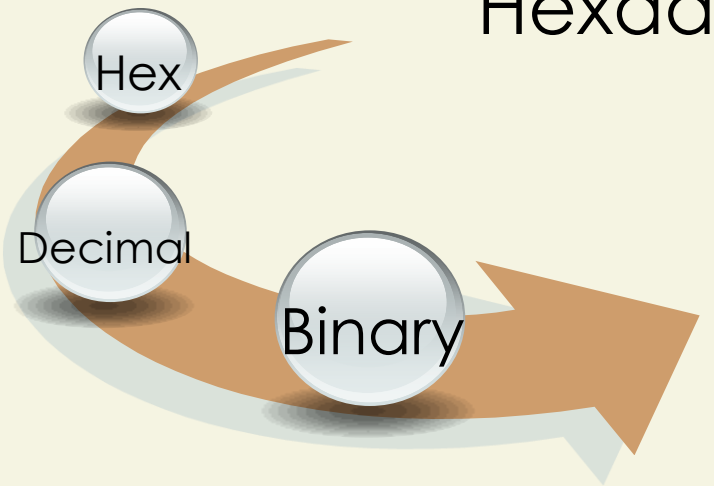
Hexadecimal →



Binary to Hexadecimal

8	4	2	1	8	4	2	1
Binary →	0	0	1	1	1	0	1	1

Hexadecimal → **3** **B**

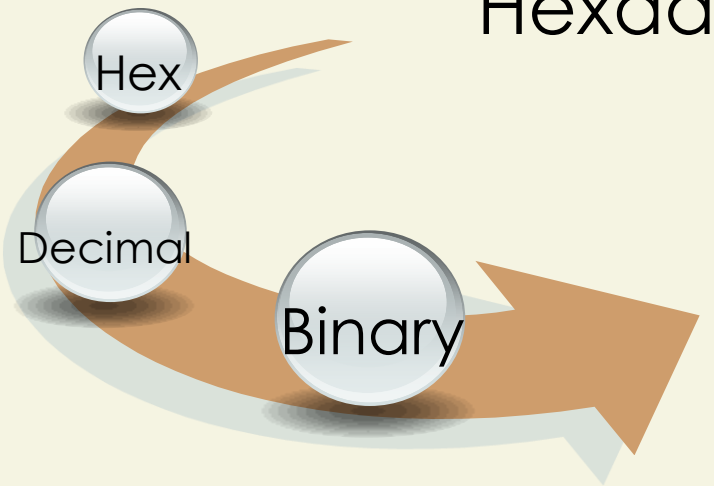


Hexadecimal to Binary

8 4 2 1 8 4 2 1

Binary →

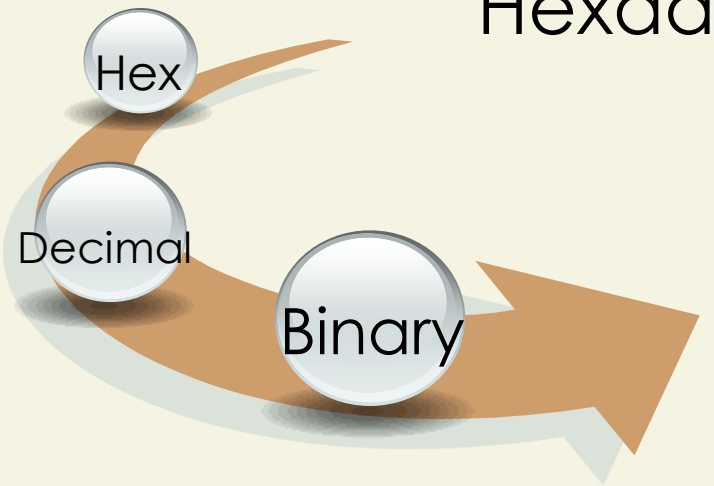
Hexadecimal → F 4



Hexadecimal to Binary

	8	4	2	1	8	4	2	1
Binary →	1	1	1	1	0	1	0	0

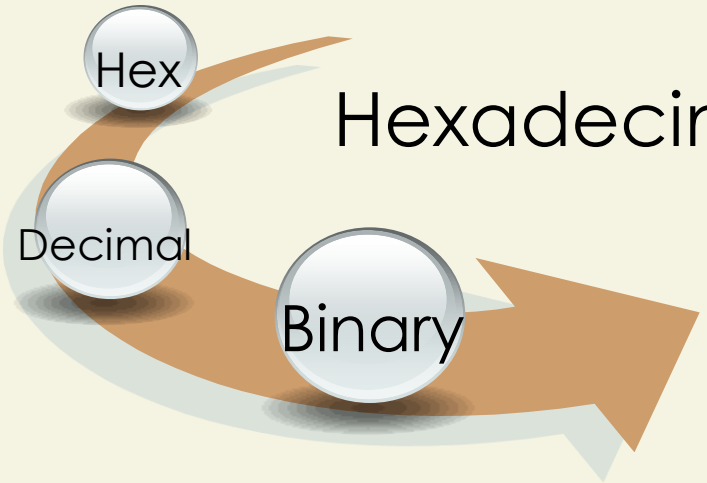
Hexadecimal → **F** **4**



Hexadecimal to Binary

Binary→

8 4 2 1 8 4 2 1 8 4 2 1

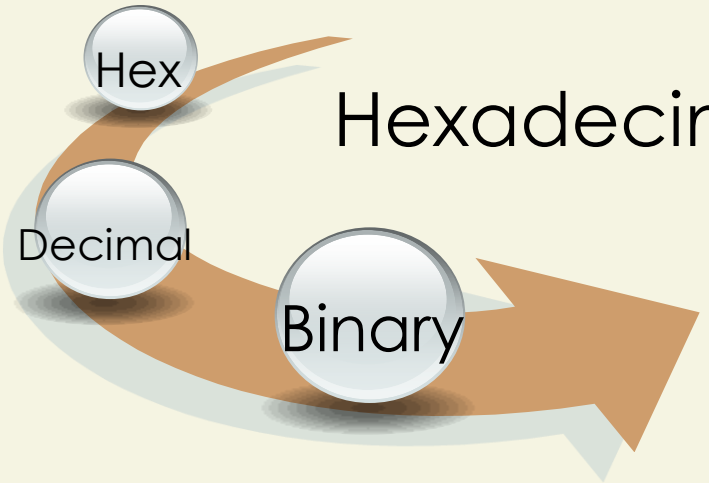


Hexadecimal → 1 3 1

Hexadecimal to Binary

Binary →

8	4	2	1	8	4	2	1	8	4	2	1
0	0	0	1	0	0	1	1	0	0	0	1



Hexadecimal → **1 3 1**

I. INTRODUCTION

Developing Assembly Language Programs

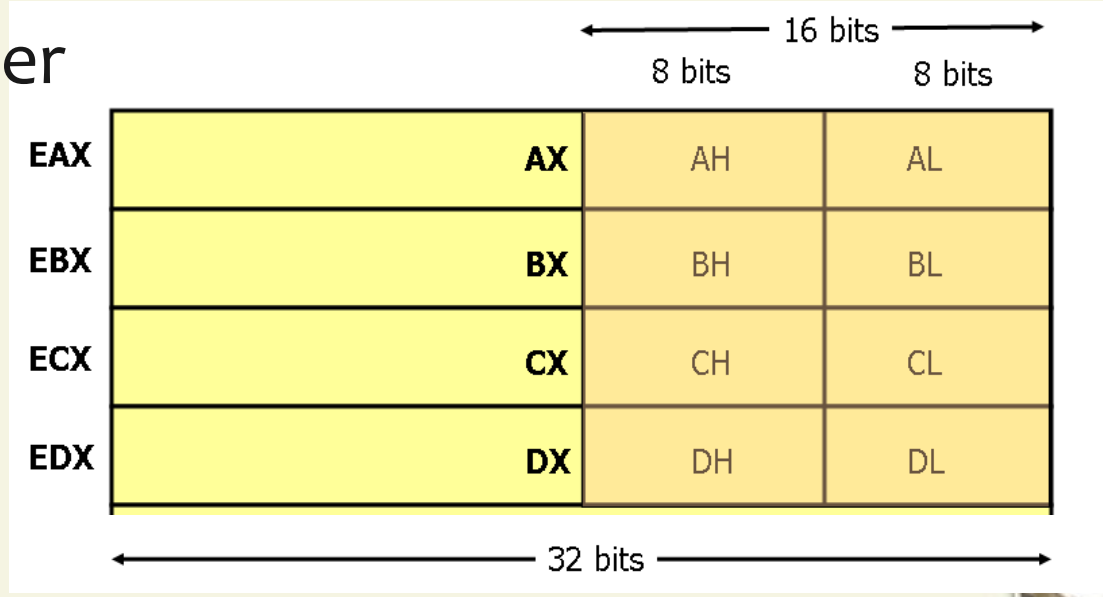
Objectives

At the end of this section, we should be able to:

- Discuss the parts of an assembly program, and
- Develop a simple assembly program implementing basic input/output and other sequential statements.

The 32-bit Registers

- General Purpose Registers
  - EAX – Accumulator
  - EBX – Base
  - ECX – Counter
  - EDX – Data



The 32-bit Registers

- Segment Registers (16 bits)
  - CS – Code Segment
  - DS, ES, FS, GS – Data Segment
  - SS – Stack Segment
- Index Registers
  - ESI – Source Index
  - EDI – Destination Index

The 32-bit Registers

- Pointer Registers
  - EBP – Base Pointer
  - ESP – Stack Pointer
- EIP – Instruction Pointer (a.k.a. PC)
- eFlags – Flag Registers

Parts of an Assembly Program

Section .data

- initialized variables

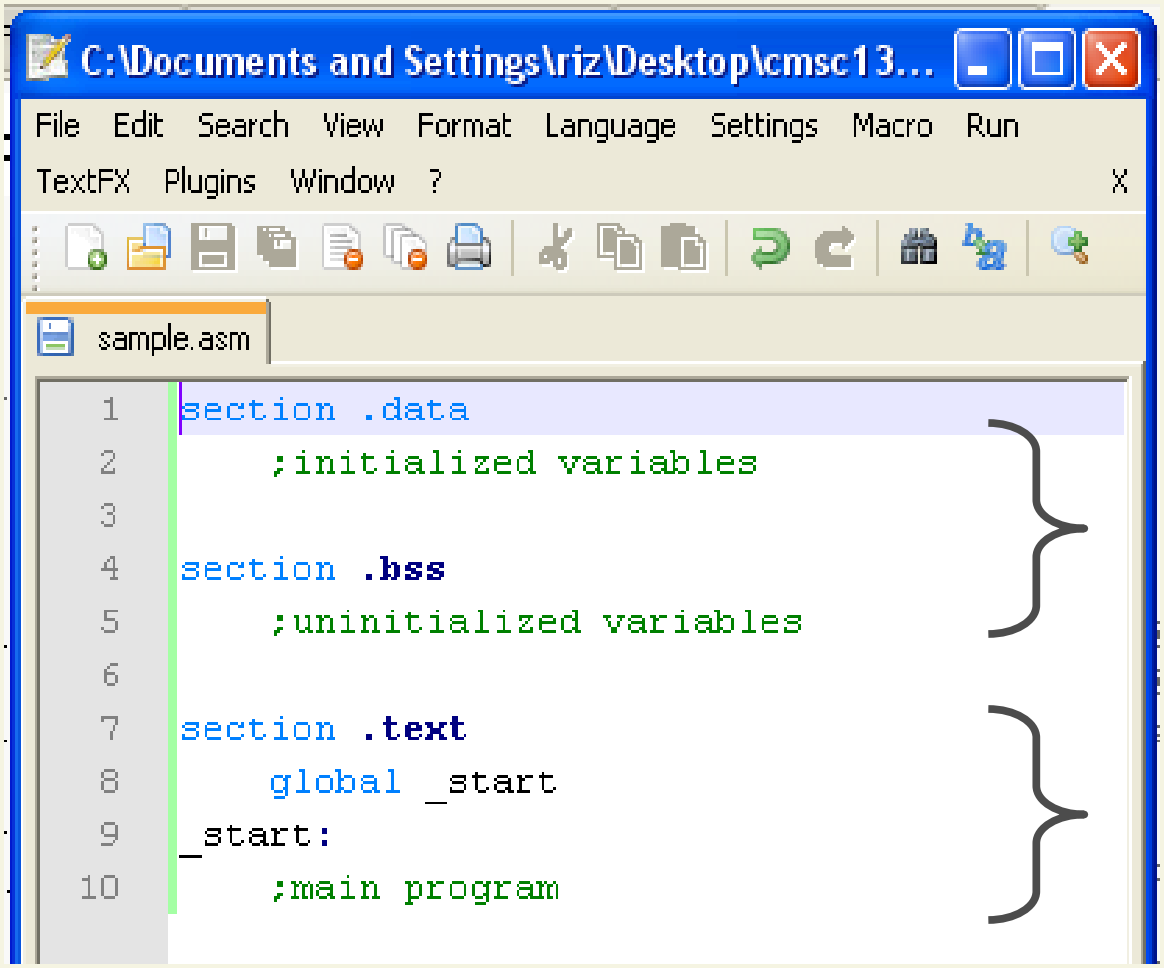
Section .bss

- unintialized variables

Section .text

- instructions
- program code

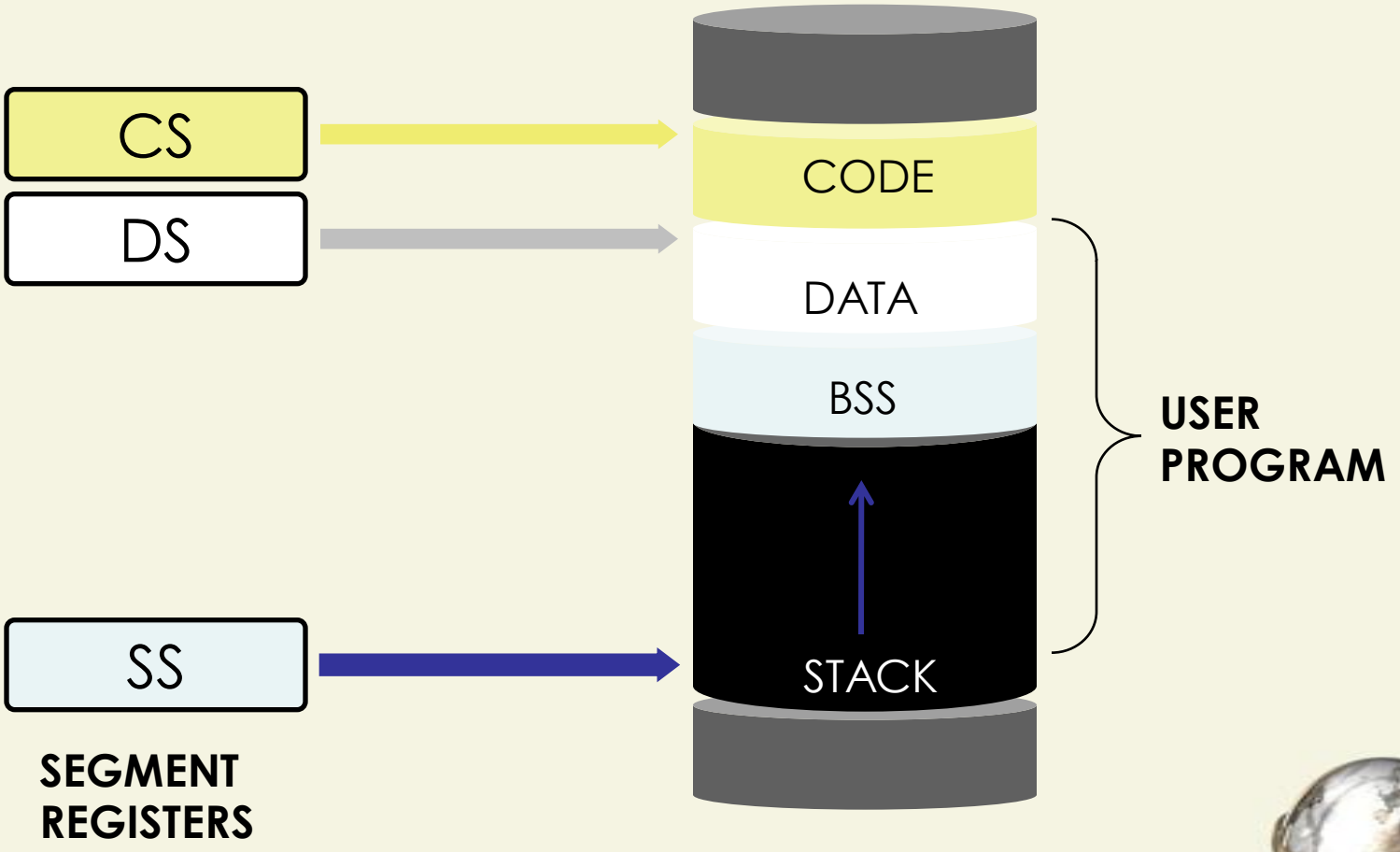
Parts of an Assembly Program



Data Segment

Code Segment

Segments and Segment Registers



Data and Data Types

High-level Programming Languages

- numeric
  - signed integer
  - unsigned integer
  - float or real
- non-numeric
  - characters and strings
  - boolean
  - sets

Data and Data Types

Computers

- only know numbers (bits)
- data types are human abstractions
- data types depend on size of data and human interpretation

Instructions and Directives

Instructions

- tell processor what to do
- assembled into machine code by assembler
- executed at runtime by the processor
- from the Intel x86 instruction set

Instructions and Directives

Directives

- tell assembler what to do
- commands that are recognized and acted upon by the assembler
- not part of the instruction set
- used to declare code and data areas, define constants and memory for storage
- different assemblers have different directives

Assembler Directives

EQU directive

- defines constants
  - *label* equ *value*
  - *count* equ *100*

Data definition directive

- defines memory for data storage
- defines size of data

Assembler Directives

Initialized Data

- db – define byte
- dw – define word
- dd – define double

– *label*	directive	*initial value*
– *int*	db	*0*
– *num*	dw	*100*

Assembler Directives

Character constants

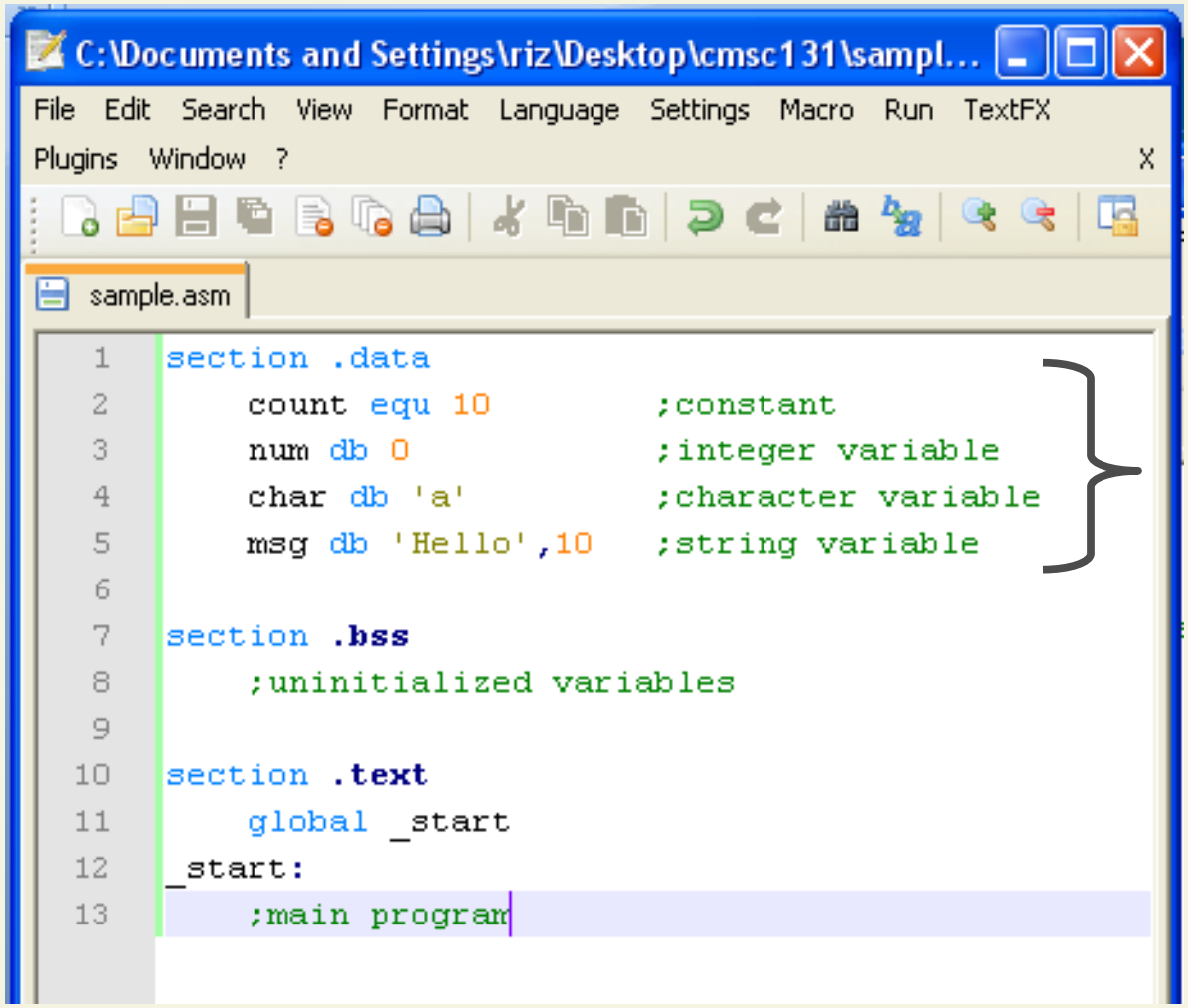
- single quote delimited
- ‘A’
  - *char* db *‘!’*

Assembler Directives

String constants

- single quote delimited or sequence of characters separated by commas
- each character is one byte each
- ‘hello’
- ‘h’, ’e’, ’l’, ’l’, ’o’
  - *prompt1* db *‘Please enter number: ’*
  - *prompt2* db *‘Please enter number: ’,10*

Assembler Directives



Declarations

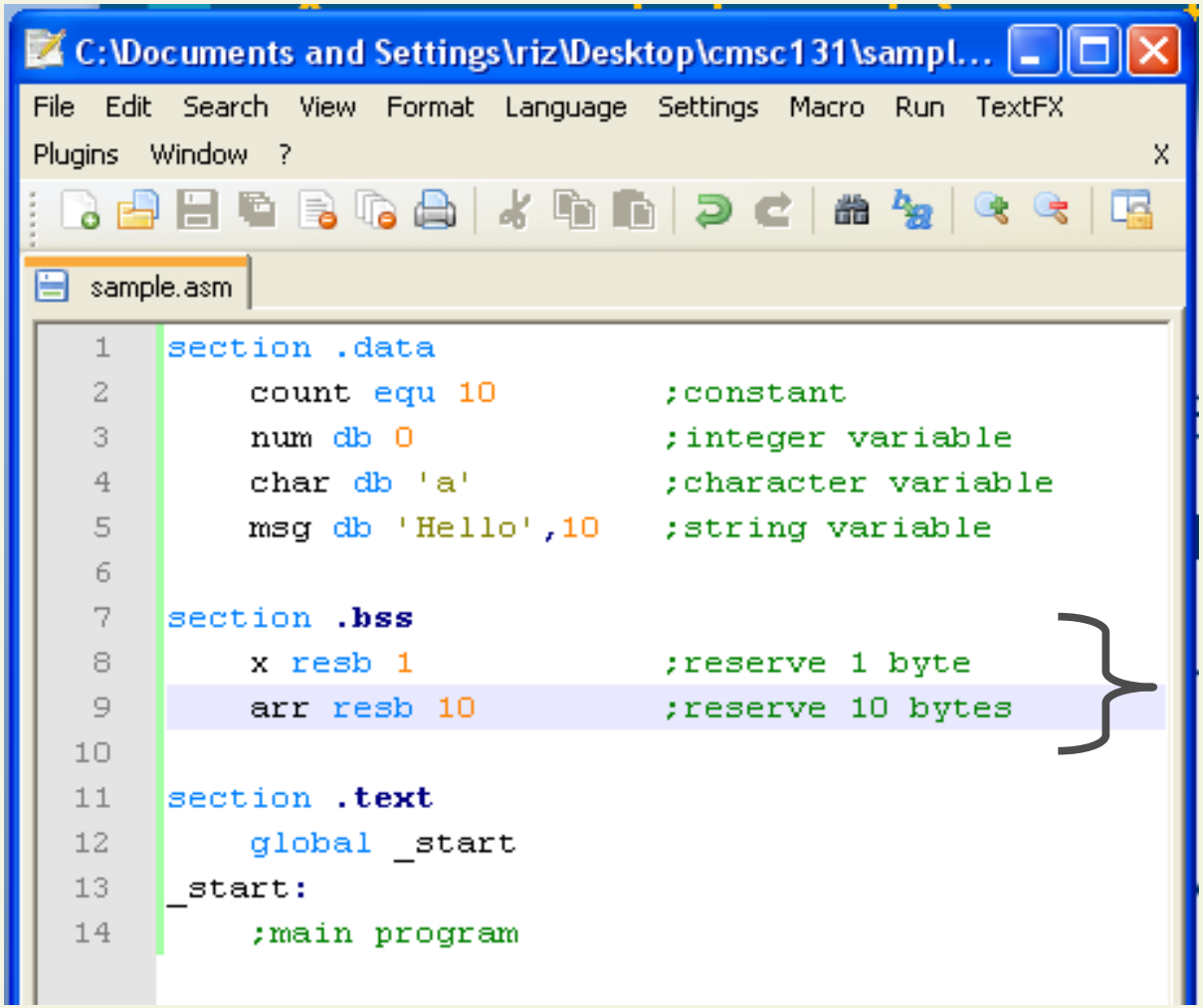
Assembler Directives

Uninitialized Data

- resb – reserve byte
- resw – reserve word
- resd – reserve double word

  - *label* directive *value*
  - *num* resb *1* *; reserves 1 byte*
  - *nums* resb *10* *; reserves 10 bytes*

Assembler Directives



Uninitialized variables

Assembly Instructions

Basic Format

instruction operand1, operand2

Operand

- Register
- Immediate
- Memory

Instruction Operands

Register

- eax, ax, ah, al
- ebx, bx, bh, bl
- ecx, cx, ch, cl
- edx, dx, dh, dl

Instruction Operands

Immediate

- character constants
  - character symbols enclosed in quotes
  - character ASCII code
- integer constants
  - begin with a number
  - ends with base modifier (B, O or H)
- ‘A’ = 65 = 41H = 01000001B

Instruction Operands

Memory

- when using the value of a variable, enclose the variable name in square brackets
- [num] - value of num
- num - address of num

Instruction Operands

- If the operands are registers or memory locations, they must be of the same type.
- Two memory operands are not allowed in the instruction.