System I

# Computer System and Information Representation

Haoting Shen

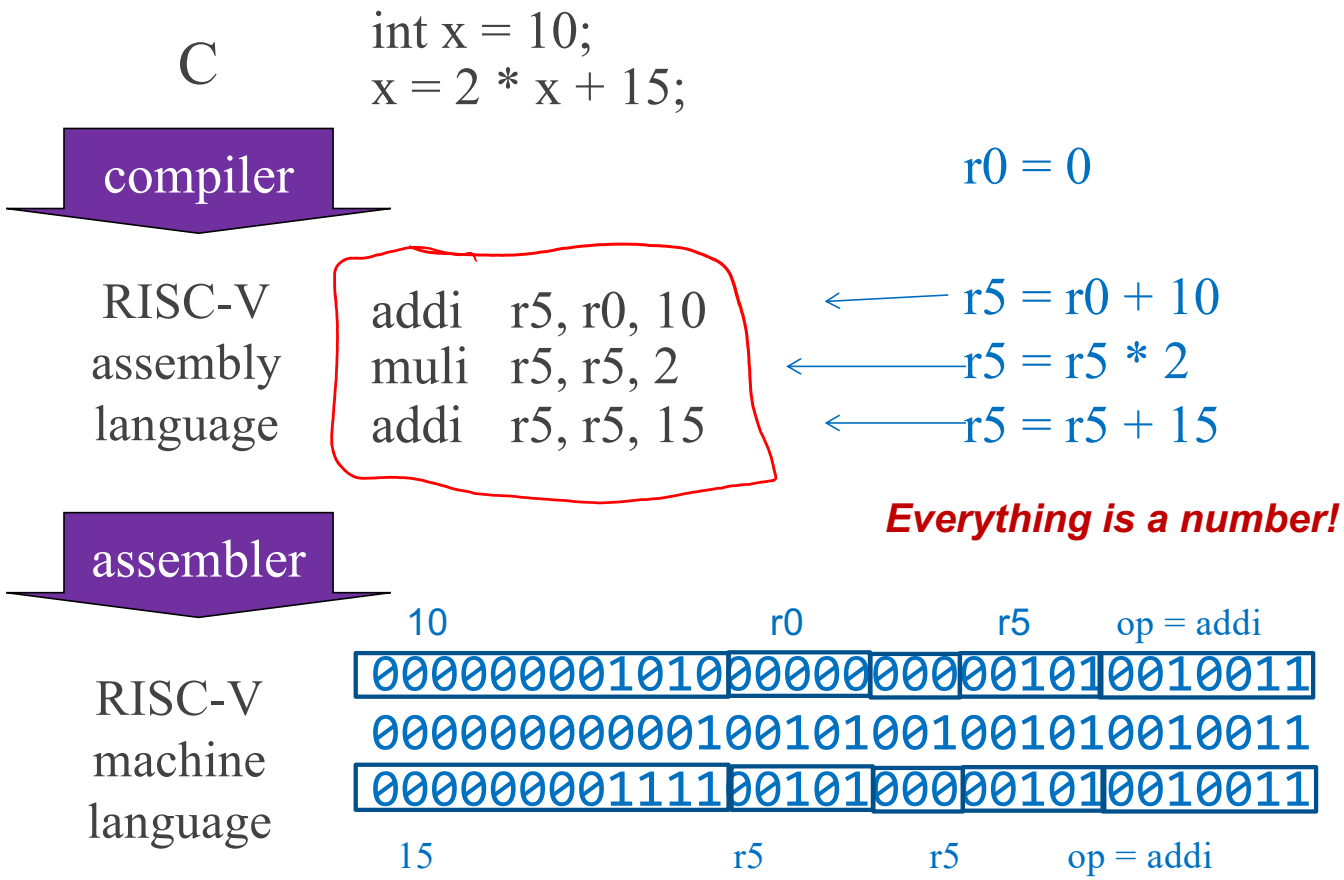
Zhejiang University

# Disclaimer

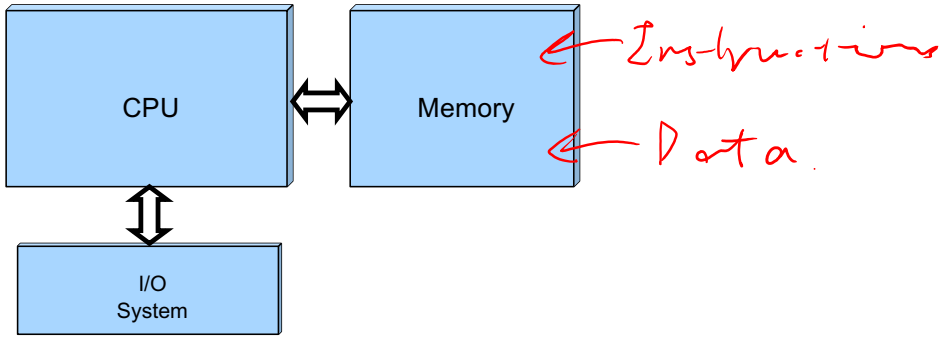
- **Many images and resources used in this lecture are collected from the Internet, and they are used only for the educational purpose. The copyright belong to the original owners, respectively.**

- **Part of slides credit to**
  - **CSAPP @ CMU**
  - **Prof. Yabo Dong @ ZJU**
  - **Dr. Jun Lu, Peiking Univ.**
  - **Coursera.org**

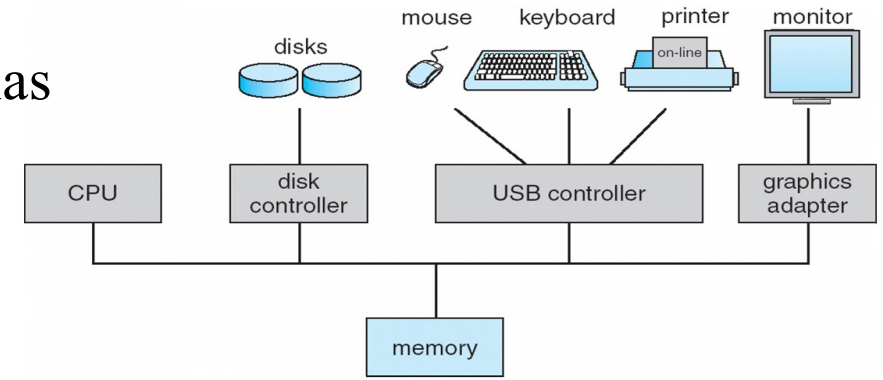
# Source Code -> Executable File



# Von-Neumann Model




- Amazingly, it's still possible to think of the computer this way at a conceptual level (model from ~80 years ago!!!)
- A computer today has some differences

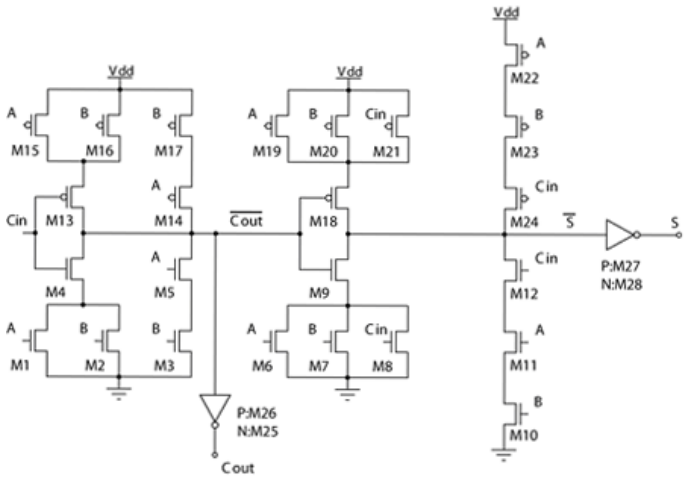


# Conceptual View of Memory

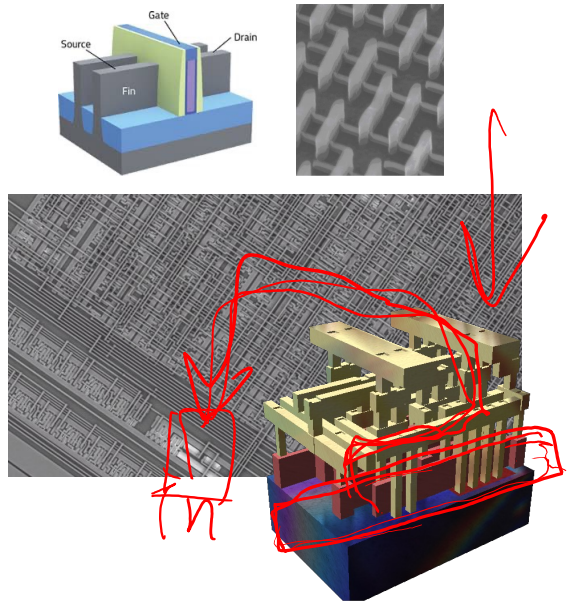
| address | content |
|---|---|
| 0000 0000 0000 0000 | 0110 1110 |
| 0000 0000 0000 0001 | 1111 0100 |
| 0000 0000 0000 0010 | 0000 0000 |
| 0000 0000 0000 0011 | 0000 0000 |
| 0000 0000 0000 0100 | 0101 1110 |
| 000( | |
| 000( | |
| 0000 0000 0000 0111 | 0100 0000 |
| 0000 0000 0000 1000 | 1111 0101 |
| ... | ... |

**At address 0000 0000 0000 0010**
**the content is 0000 0000**

## Transistor Schematic

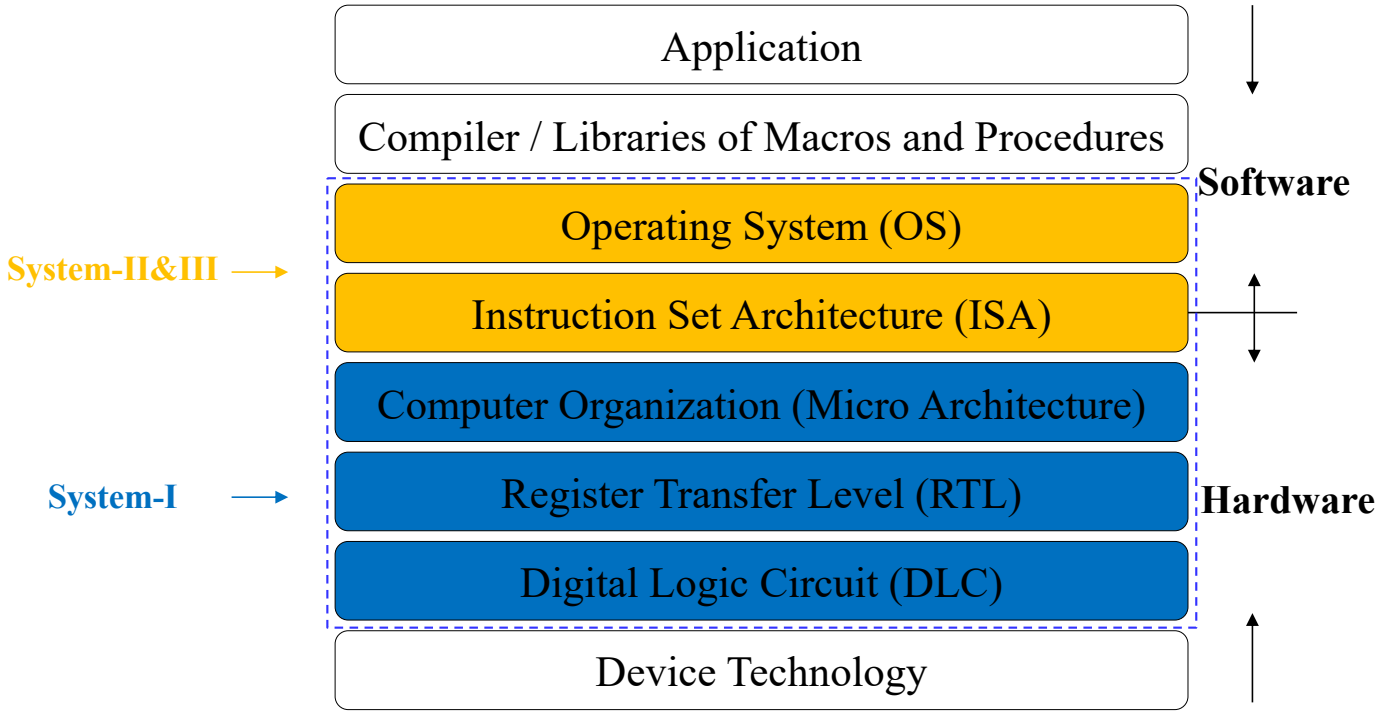



Full adder

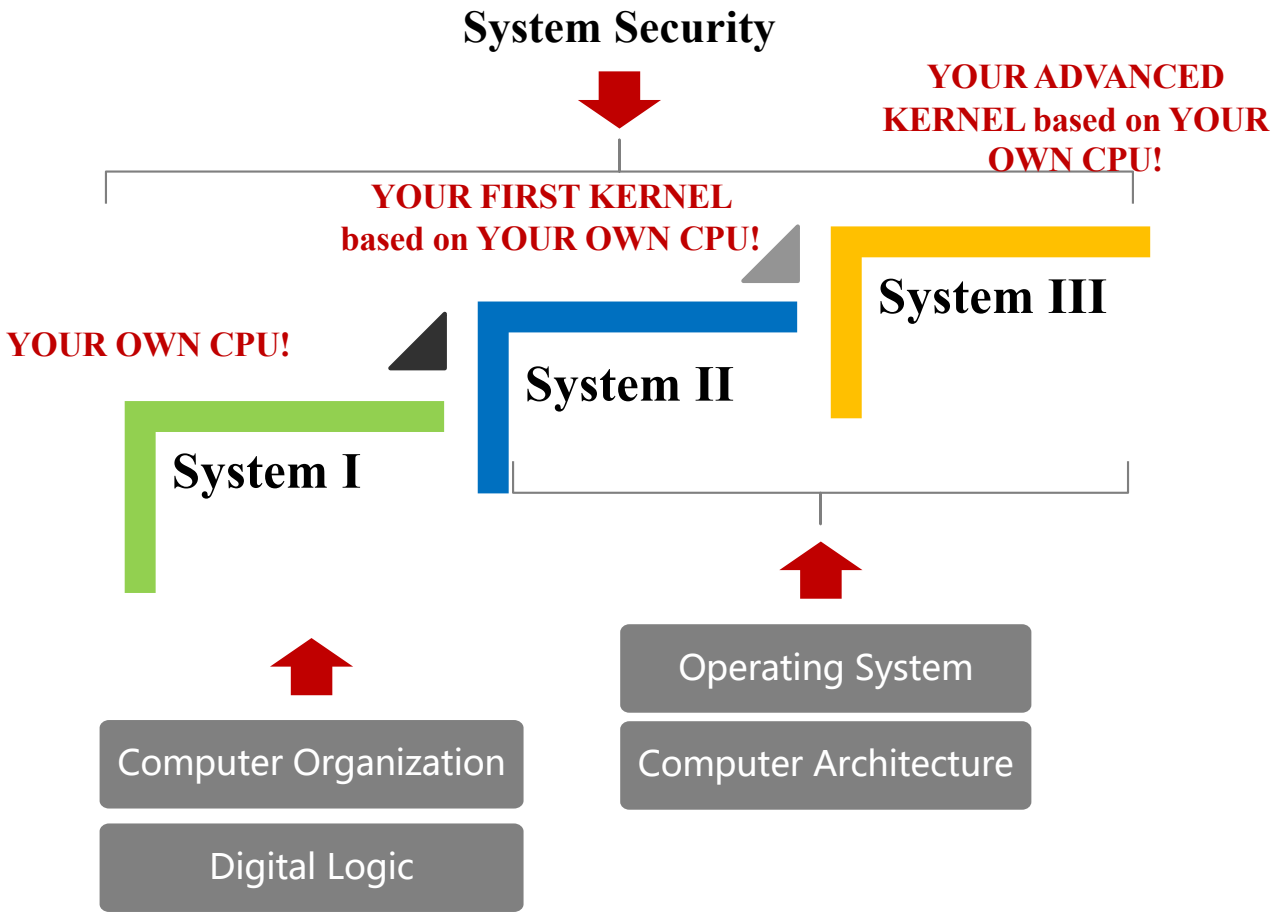



Very Large Scale Integrated (VLSI) Circuit

# Covered in ZJU Computer System Courses



# ZJU Computer System Courses

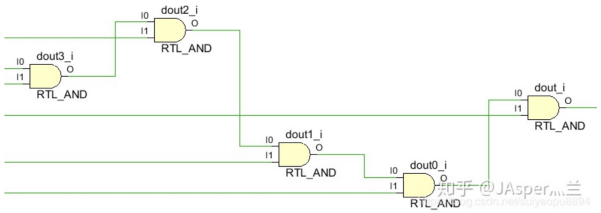




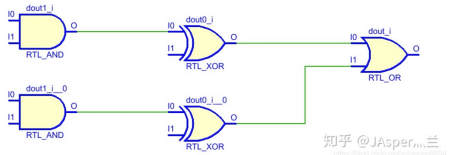

8

# LUT

```
//以下是例1
module top(
    input clk,
    input rst,
    input data0,
    input data1,
    input data2,
    input data3,
    input data4,
    input data5,
    output dout
    );

 assign dout = data0&data1&data2&data3&data4&data5;

endmodule
```




```
//以下是例2
module top(
    input clk,
    input rst,
    input data0,
    input data1,
    input data2,
    input data3,
    input data4,
    input data5,
    output dout
    );

 assign dout = ((data0&data1)^data2|data3&data4^data5);

endmodule
```



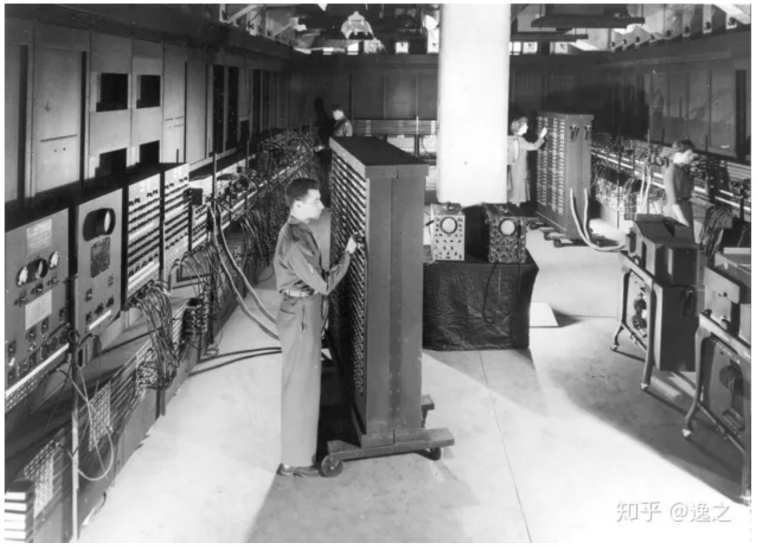
# Overview

- **Introduction to computer systems**
- Binary number representation
- Arithmetic operations
- Representation of numeric data
- Representation of non-numeric data
- Data width and storage

# ENIAC

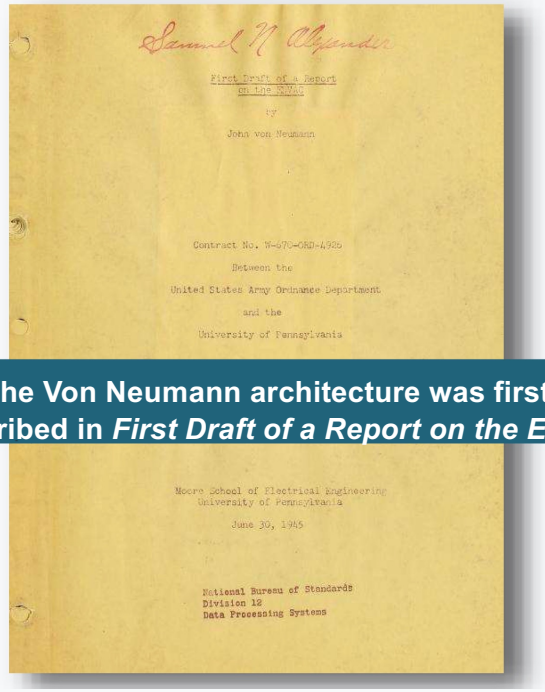
- ENIAC (Electronic Numerical Integrator and Computer)
- EE@Upenn
- Funded by the U.S. Army's Ballistics Research Laboratory to calculate missiles

**Use mechanical switches, manual wire connections, and other settings to implement instructions.**




ENIAC（图片来自维基百科）

# Von Neumann and EDVAC


Samuel N Alexander

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W-670-ORD-4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

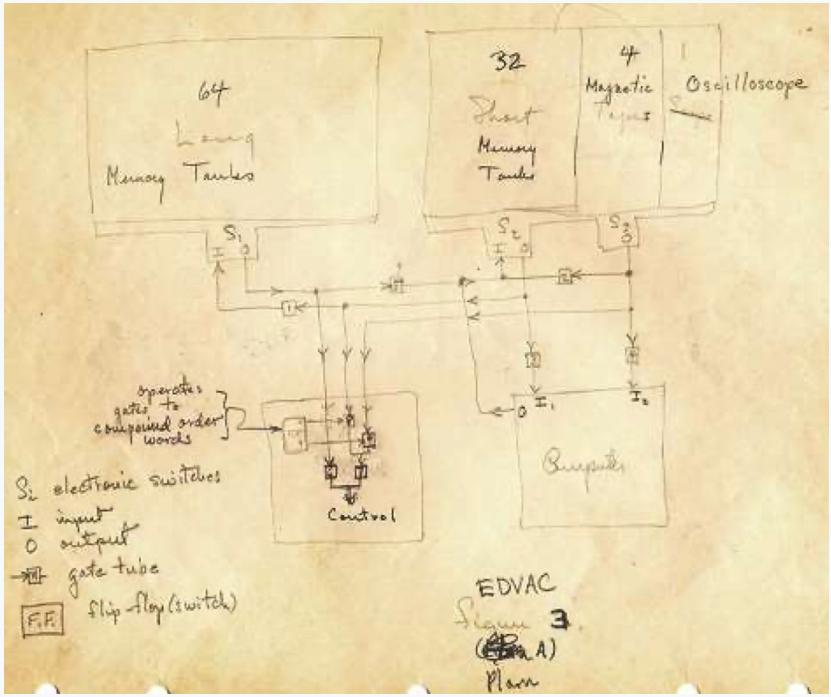
National Bureau of Standards
Division 12
Data Processing Systems

**The Von Neumann architecture was firstly described in *First Draft of a Report on the EDVAC***

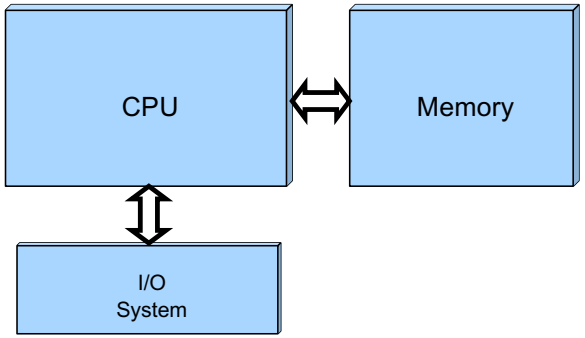
John Von Neumann
1903~1957

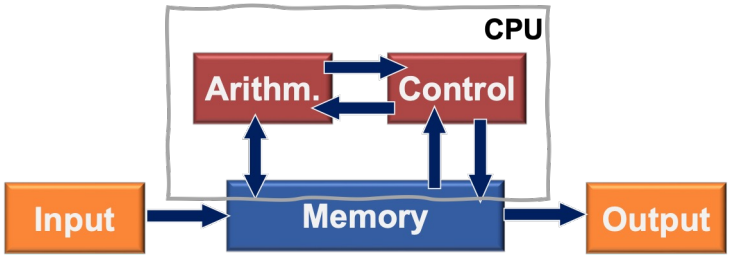
# Von Neumann and EDVAC

- 101 pages, unfinished manuscript
- Tow important conceptions
  - **Stored program**
  - Binary
- 5 parts
  - Central Arithmetical (CA)
  - Central Control (CC)
  - Memory (M)
  - Input (I)
  - Output (O)

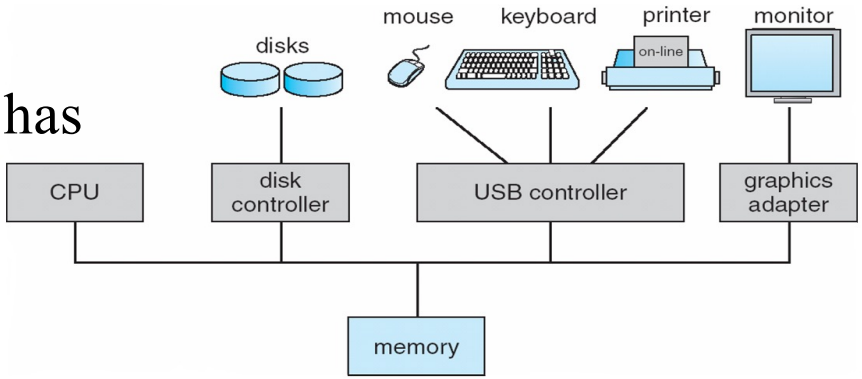


# Von-Neumann Model



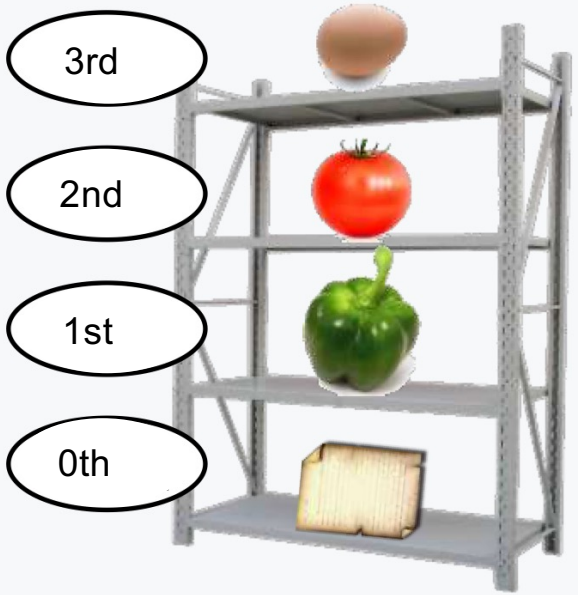




- Amazingly, it's still possible to think of the computer this way at a conceptual level (model from ~80 years ago!!!)
- A computer today has some differences

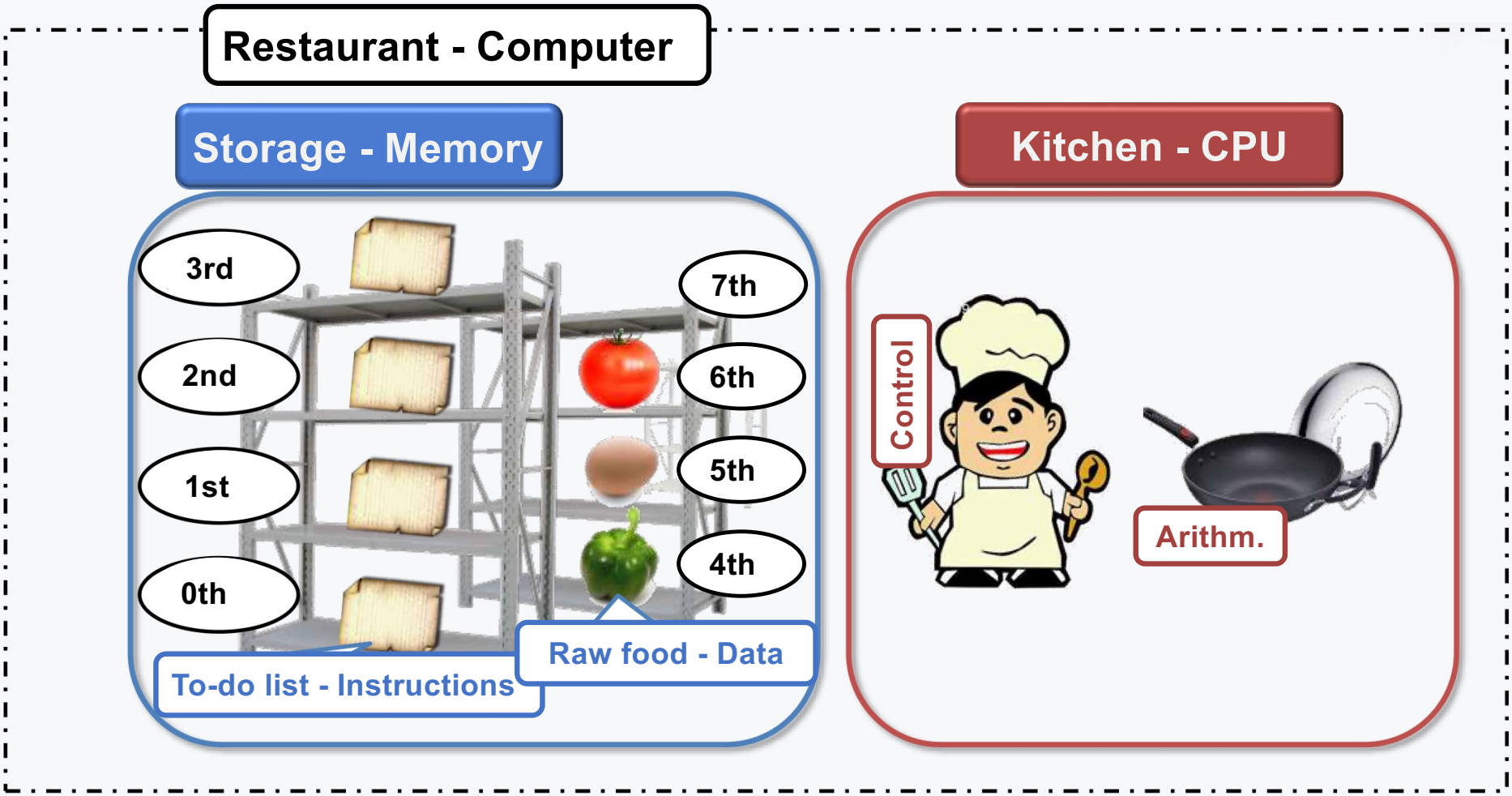


# Main Memory (also called RAM)

| | Address | Content |
|---|---|---|
| High address | ... | ..... |
| | 0011 | 00001100 |
| | 0010 | 00100010 |
| | 0001 | 00000000 |
| Low address | 0000 | 01101101 |



# Von Neumann Computer Vs. Restaurant



# Von Neumann Computer Vs. Restaurant

Need Task on 1st

Task

Memory

3rd

2nd

1st

0th

7th

6th

5th

4th

To-do list - Instructions

Raw food - Data

Next step: on 1st

Current task

Kitchen - CPU

Control

Arithm.

General Purpose Registers (GPRs)

# Von Neumann Computer Vs. Restaurant

How to cook: fry
Raw food: 6th and disk A
Where to leave: disk A

Task

Next step: on 2nd

1. Fetch 6th
2. Fetch disk A
3. Fry
4. Put into disk A

Memory

3rd
2nd
1st
0th
7th
6th
5th
4th

To-do list - Instructions

Raw food - Data

Control

Arithm.

CPU

General Purpose Registers (GPRs)

# Von Neumann Computer Vs. Restaurant

How to cook: fry
Raw food: 6th and disk A
Where to leave: disk A

Task

Next step: on 2nd

1. Fetch 6th
2. Prepare disk A
3. Fry
4. Put into disk A

Memory

3rd
2nd
1st
0th
7th
6th
5th
4th

To-do list - Instructions

Raw food - Data

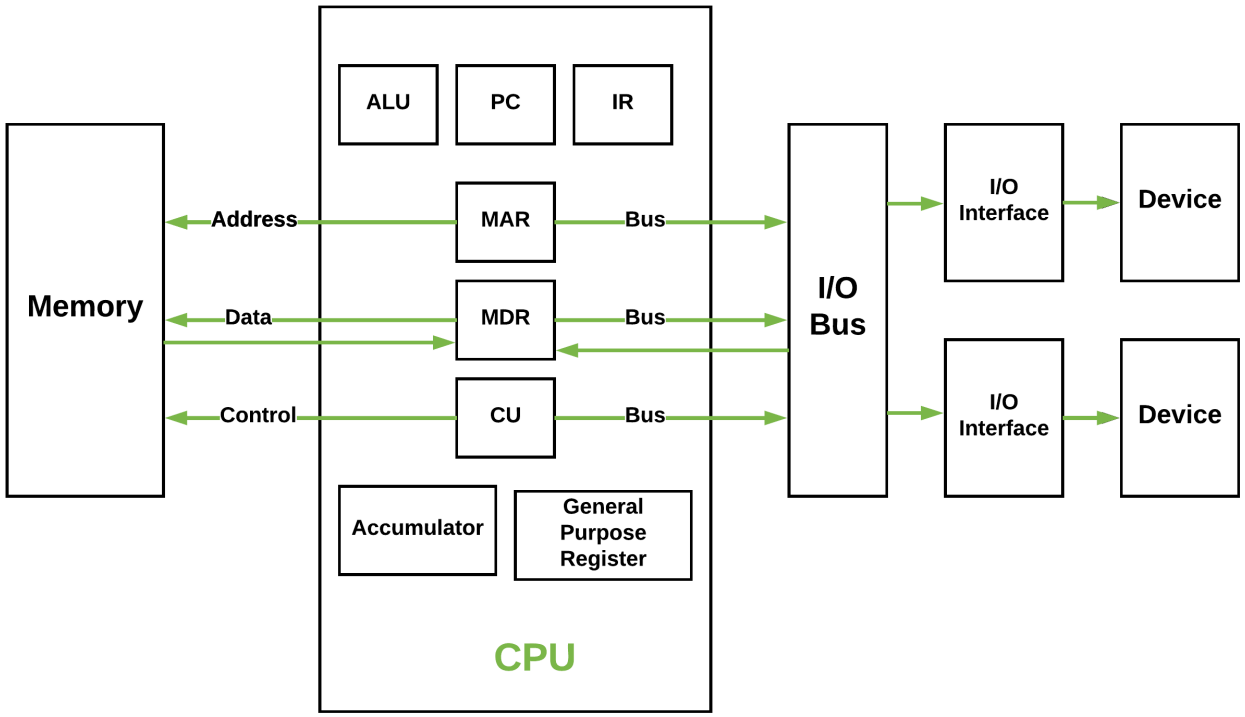
Control

Arithm.

CPU

General Purpose Registers (GPRs)

# Details in Von-Neumann Model




https://media.geeksforgeeks.org/wp-content/uploads/vn_cpu.png

# Program Representation and Execution

- **Program Representation**
- **Program Execution**

# Program Representation

- **Instructions**
  - **0/1 binaries**

- **Program Language**

| Low Level Language |
| --- |
| Machine Language |
| Assembly Language |
| |

| High Level Language |
| --- |
| C |
| Java |
| …… |

| Translator |
| --- |
| Assembler |
| Interpreter |
| Compiler |

# Program Execution

- **Execute Instructions Automatically**
  - **Instruction Fetch**
  - **Instruction Decode**
  - **Execute**
  - **Memory Access**
  - **Write Back**

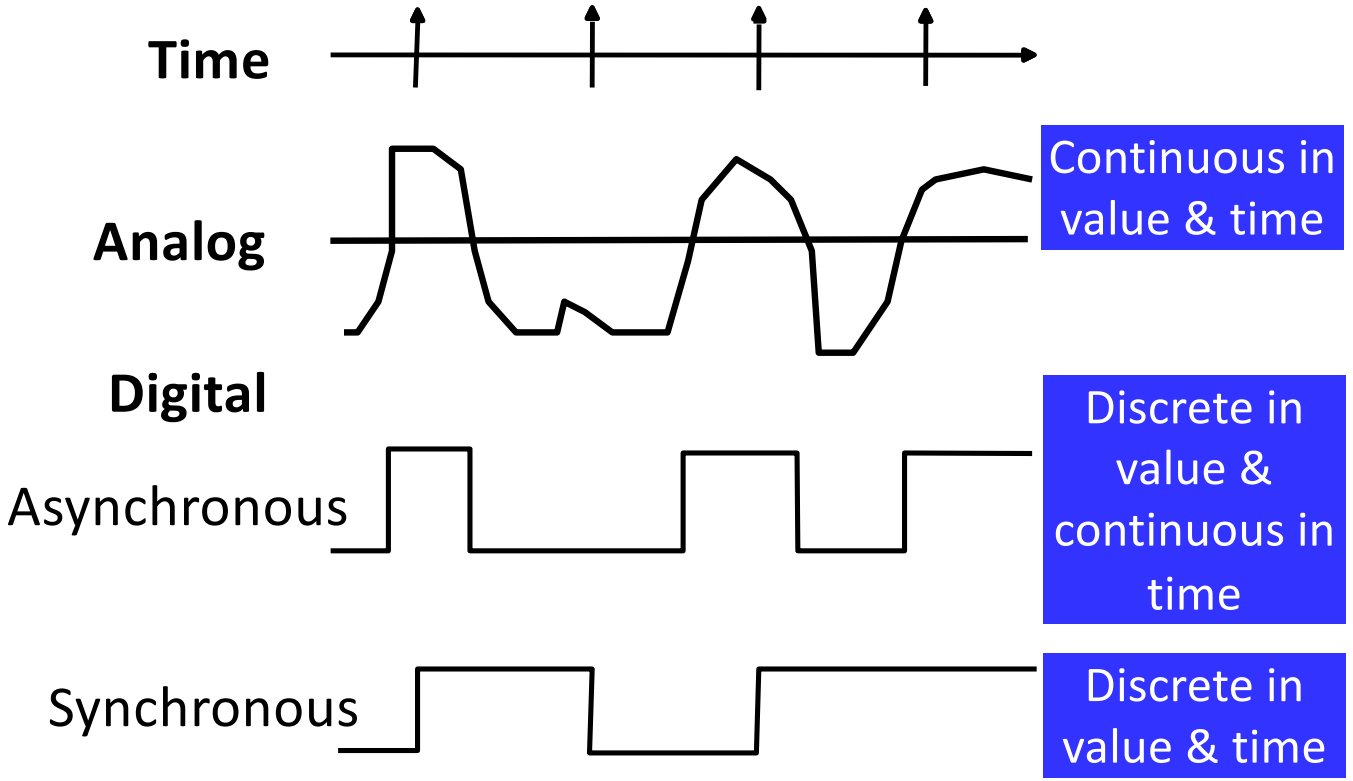
# Overview

- Introduction to computer systems
- **Binary number representation**
- Arithmetic operations
- Representation of numeric data
- Representation of non-numeric data
- Data width and storage

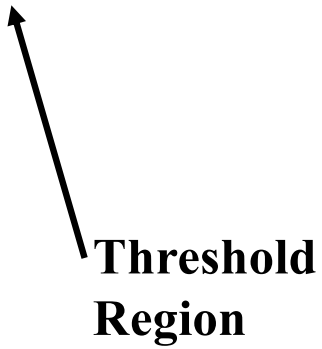
# INFORMATION REPRESENTATION - Signals

- Information variables represented by **physical quantities**.
- For digital systems, the variables take on **discrete** values.
- Two level, or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.
- Binary values are represented by values or **ranges of values** of physical quantities

# Signal Examples Over Time

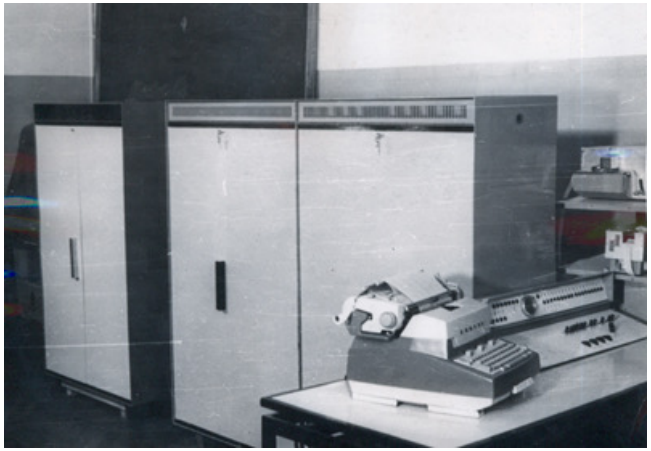


# Signal Example – Physical Quantity: Voltage



# Why Binary?

- **Setun (Сетунь)**
  - **A balanced ternary computer developed in 1958 at Moscow State University**
  - **The only modern ternary computer, using three-valued ternary logic**



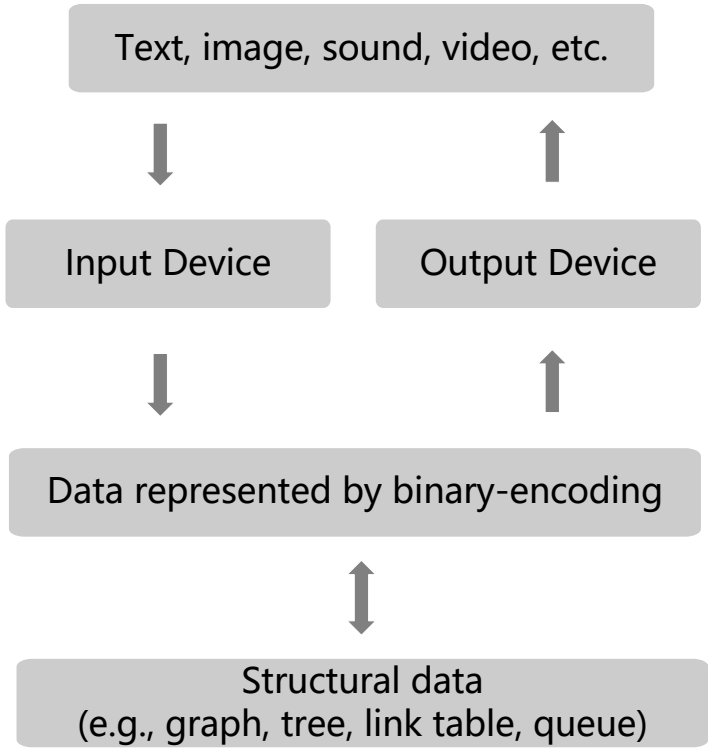
Setun-70 in 1970

**Is binary the most efficient one in digital systems?**

# External information and internal data

- **Continuous vs. discrete**
- **Advanced types vs. primitive types**

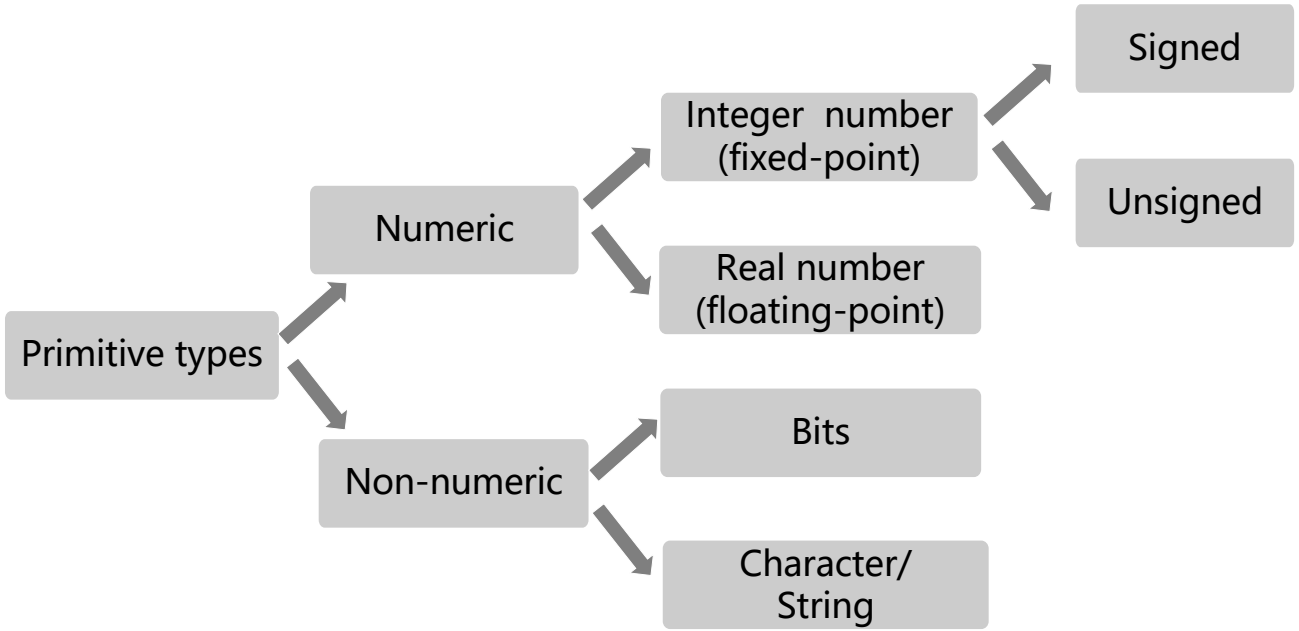
# External Information




User's perspective

High-level programmer's perspective

# Internal Data




Low-level programmer's/ hardware system designer's perspective

## Carry Counting System

- Positive radix, positional number systems
- A number with *radix* ***r*** is represented by a string of digits:

  $$A_{n-1}A_{n-2}\ldots A_1A_0 . A_{-1}A_{-2}\ldots A_{-m+1}A_{-m}$$

  in which $\mathbf{0} \le A_i < r$ and **.** is the *radix point*.
- The string of digits represents the power series:

$$(\text{Number})_r = \left(\sum_{i=0}^{i=n-1} A_i \cdot r^i\right) + \left(\sum_{j=-m}^{j=-1} A_j \cdot r^j\right)$$

**(Integer Portion) + (Fraction Portion)**

# About the Decimal Number

**123.456**

$$1*10^{2}+2*10^{1}+3*10^{0}+4*10^{-1}+5*10^{-2}+6*10^{-3}$$

# What About the Binary Number?

$$(100101.01)_2$$

$$1*2^5+0*2^4+0*2^3+1*2^2+0*2^1+1*2^0+0*2^{-1}+1*2^{-2}$$

# Different Radixes

| Name | Radix | 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 |

# Different Radixes

| Decimal (Base 10) | Binary (Base 2) | Octal (Base 8) | Hexadecimal (Base 16) |
|---|---|---|---|
| 00 | 00000 | 00 | 00 |
| 01 | 00001 | 01 | 01 |
| 02 | 00010 | 02 | 02 |
| 03 | 00011 | 03 | 03 |
| 04 | 00100 | 04 | 04 |
| 05 | 00101 | 05 | 05 |
| 06 | 00110 | 06 | 06 |
| 07 | 00111 | 07 | 07 |
| 08 | 01000 | 10 | 08 |
| 09 | 01001 | 11 | 09 |
| 10 | 01010 | 12 | 0A |
| 11 | 01011 | 13 | 0B |
| 12 | 01100 | 14 | 0C |
| 13 | 01101 | 15 | 0D |
| 14 | 01110 | 16 | 0E |
| 15 | 01111 | 17 | 0F |
| 16 | 10000 | 20 | 10 |

# Converting R-base to Decimal

- **To convert to decimal, use decimal arithmetic to form Σ (digit × respective power of R).**
- **Convert $(100101.01)_2$ to $N_{10}$:**
  - $1*2^5+0*2^4+0*2^3+1*2^2+0*2^1+1*2^0+0*2^{-1}+1*2^{-2}=(37.25)_{10}$
- **Convert 3A.CH to $N_{10}$:**
  - $3*16^1+10*16^0+12*16^{-1}=(58.75)_{10}$

# Converting Decimal to R-base

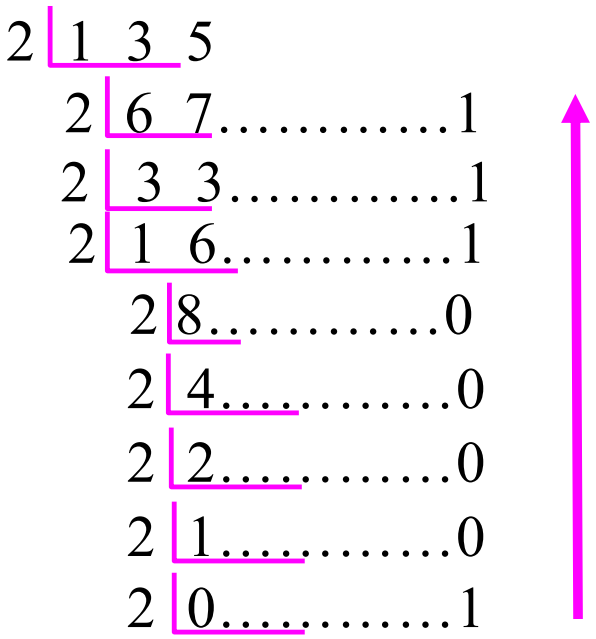
- **To convert from one base to another:**
  - **Convert the integer part**
  - **Convert the fraction part**
  - **Join the two results together**

# Convert the Integral Part

- **Repeatedly divide the number by the new radix and save the remainders. The digits for the new radix are the remainders in *reverse order* of their computation.**
  - **If the new radix is > 10, then convert all remainders > 10 to digits A, B, …**

# Example: Convert $135_{10}$ To Base 2

$$(135)_{10} = (1000\ 0111)_2$$



# Convert the Fractional Part

- **Repeatedly multiply the fraction by the new radix and save the integer digits that result. The digits for the new radix are the integer digits in *order* of their computation.**
  - **If the new radix is > 10, then convert all remainders > 10 to digits A, B, …**

# Example: Convert $0.6875_{10}$ To Base 2

$$(0.6875)_{10} = (0.1011)_2$$

$2 \times 0.6875 \ldots\ldots = 1.375$

$2 \times 0.375 \ldots\ldots = 0.75$

$2 \times 0.75 \ldots\ldots = 1.5$

$2 \times 0.5 \ldots\ldots = 1$

## Convert $135.6875_{10}$ To Base2

- **Convert 135 to Base 2**

$$(135)_{10} = (10000111)_2$$

- **Convert 0.6875 to Base 2:**

$$(0.6875)10 = (0.1011)_2$$

- **Join the results together**

$$(135.6875)10 = (10000111.1011)_2$$

# Additional Issue - Fractional Part

- **Note that in this conversion, the fractional part can become 0 as a result of the repeated multiplications.**
- **In general, it may take many bits to get this to happen or it may never happen.**
- **Example Problem: Convert $0.65_{10}$ to $N_2$**
  - **0.65 = 0.1010011001001 …**
  - **The fractional part begins repeating every 4 steps yielding repeating 1001 forever!**
- **Solution: Specify number of bits to right of radix point and round or truncate to this number.**

# Why Do Repeated Division and Multiplication Work?

- **Divide the integer portion of the power series by radix r. The remainder of this division is $A_0$, represented by the term $A_0/r$.**
- **Discard the remainder and repeat, obtaining remainders $A_1$, …**
- **Multiply the fractional portion of the power series by radix r. The integer part of the product is $A_{-1}$.**
- **Discard the integer part and repeat, obtaining integer parts $A_{-2}$, …**
- **This demonstrates the algorithm for any radix r >1.**

# Octal (Hexadecimal) to Binary and Back

- **Octal (Hexadecimal) to Binary:**
  - **Restate the octal (hexadecimal) as three (four) binary digits starting at the radix point and going both ways.**
- **Binary to Octal (Hexadecimal):**
  - **Group the binary digits into three (four) bit groups starting at the radix point and going both ways, padding with zeros as needed in the fractional part.**
  - **Convert each group of three bits to an octal (hexadecimal) digit.**

## Hexadecimal to Binary and Back

| | |
|---|---|
| (0)H = 0000 | (1)H = 0001 |
| (2)H = 0010 | (3)H = 0011 |
| (4)H = 0100 | (5)H = 0101 |
| (6)H = 0110 | (7)H = 0111 |
| (8)H = 1000 | (9)H = 1001 |
| (A)H = 1010 | (B)H = 1011 |
| (C)H = 1100 | (D)H = 1101 |
| (E)H = 1110 | (F)H = 1111 |

# Some Examples

$(67.731)_8$ = （110 111 .111 011 001)$_2$

$(312.64)_8$=(011 001 010 . 110 1)$_2$

(11 111 101 . 010 011 11)$_2$=$(375.236)_8$

(10 110.11)$_2$=$(26.6)_8$

$(2B.5E)_{16}$ = （0010 1011.0101 1110)$_2$

$(21A.5)_{16}$=(0010 0001 1010 . 0101)$_2$

$(1001101.01101)_2$= ( 0100 1101. 0110 1000 )$_2$=$(4D.68)_{16}$

(110 0101.101)$_2$=$(65.A)_{16}$

# Octal to Hexadecimal via Binary

- **Convert octal to binary.**
- **Use groups of four bits and convert as above to hexadecimal digits.**
- **Example: Octal to Binary to Hexadecimal**

**6 3 5 . 1 7 7 $_8$**

→ **110 | 011 | 101 . 001 | 111 | 111$_2$**

→ **1 | 1001 | 1101 . 0011 | 1111 | 1(000)$_2$**

→ **1 9 D . 3 F 8$_{16}$**

**Why do these conversions work?**

## Simple BASE CONVERSION(From Decimal to Binary) - Positive Powers of 2

- Useful for Base Conversion

| Exponent | Value |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |

| Exponent | Value |
|---|---|
| 11 | 2,048 |
| 12 | 4,096 |
| 13 | 8,192 |
| 14 | 16,384 |
| 15 | 32,768 |
| 16 | 65,536 |
| 17 | 131,072 |
| 18 | 262,144 |
| 19 | 524,288 |
| 20 | 1,048,576 |
| 21 | 2,097,152 |

## Special Powers of 2

- $2^{10}$ (1024) is Kilo, denoted "K"
- $2^{20}$ (1,048,576) is Mega, denoted "M"
- $2^{30}$ (1,073, 741,824)is Giga, denoted "G"
- $2^{40}$ (1,099,511,627,776 ) is Tera, denoted “T"

# $2^X$ vs. $10^Y$ Bytes Ambiguity

| Decimal term | Abbreviation | Value | Binary term | Abbreviation | Value | % Larger |
|---|---|---|---|---|---|---|
| kilobyte | KB | $10^3$ | kibibyte | KiB | $2^{10}$ | 2% |
| megabyte | MB | $10^6$ | mebibyte | MiB | $2^{20}$ | 5% |
| gigabyte | GB | $10^9$ | gibibyte | GiB | $2^{30}$ | 7% |
| terabyte | TB | $10^{12}$ | tebibyte | TiB | $2^{40}$ | 10% |
| petabyte | PB | $10^{15}$ | pebibyte | PiB | $2^{50}$ | 13% |
| exabyte | EB | $10^{18}$ | exbibyte | EiB | $2^{60}$ | 15% |
| zettabyte | ZB | $10^{21}$ | zebibyte | ZiB | $2^{70}$ | 18% |
| yottabyte | YB | $10^{24}$ | yobibyte | YiB | $2^{80}$ | 21% |

# Overview

- Introduction to computer systems
- Binary number representation
- **Arithmetic operations**
- Representation of numeric data
- Representation of non-numeric data
- Data width and storage

## ARITHMETIC OPERATIONS - Binary Arithmetic

- Single bit addition with carry
- Multiple bit addition
- Single bit subtraction with borrow
- Multiple bit subtraction
- Multiplication
- Division

# Single Bit Binary Addition with Carry

**Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):**

**Carry in (Z) of 0:**

| | | | | |
|---|---|---|---|---|
| Z | 0 | 0 | 0 | 0 |
| X | 0 | 0 | 1 | 1 |
| + Y | + 0 | + 1 | + 0 | + 1 |
| C S | 0 0 | 0 1 | 0 1 | 1 0 |

**Carry in (Z) of 1:**

| | | | | |
|---|---|---|---|---|
| Z | 1 | 1 | 1 | 1 |
| X | 0 | 0 | 1 | 1 |
| + Y | + 0 | + 1 | + 0 | + 1 |
| C S | 0 1 | 1 0 | 1 0 | 1 1 |

## Multiple Bit Binary Addition

- **Extending this to two multiple bit examples:**

| | | |
|---|---|---|
| **Carries** | **0** | **0** |
| **Augend** | **01100** | **10110** |
| **Addend** | **+10001** | **+10111** |
| **Sum** | **11100** | **101101** |

- **Note: The 0 is the default Carry-In to the least significant bit.**

# Single Bit Binary Subtraction with Borrow

- **Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):**
- **Borrow in (Z) of 0:**

| | | | | |
|---|---|---|---|---|
| **Z** | **0** | **0** | **0** | **0** |
| **X** | **0** | **0** | **1** | **1** |
| **- Y** | **-0** | **-1** | **-0** | **-1** |
| **B S** | **0 0** | **1 1** | **0 1** | **0 0** |

- **Borrow in (Z) of 1:**

| | | | | |
|---|---|---|---|---|
| **Z** | **1** | **1** | **1** | **1** |
| **X** | **0** | **0** | **1** | **1** |
| **- Y** | **-0** | **-1** | **-0** | **-1** |
| **B S** | **1 1** | **1 0** | **0 0** | **1 1** |

# Multiple Bit Binary Subtraction

- **Extending this to two multiple bit examples:**

| | | |
|---|---|---|
| **Borrows** | **0** | **0** |
| **Minuend** | **10110** | **10110** |
| **Subtrahend** | **- 10010** | **- 10011** |
| **Difference** | **00100** | **00011** |

- **Notes: The 0 is a Borrow-In to the least significant bit. If the Subtrahend > the Minuend, interchange and append a – to the result.**

## Binary Multiplication

**The binary multiplication table is simple:**

**0 * 0 = 0 | 1 * 0 = 0 | 0 * 1 = 0 | 1 * 1 = 1**

**Extending multiplication to multiple digits:**

| | |
|---|---|
| **Multiplicand** | **1011** |
| **Multiplier** | **x 101** |
| **Partial Products** | **1011** |
| | **0000 -** |
| | **1011 - -** |
| **Product** | **110111** |

# Binary Division

**The binary multiplication table is simple:**

**0 / 0 | 1 / 0** **meaningless**

**0 / 1 = 0 | 1 / 1 = 1**

**Extending division to multiple digits:**

**Dividend** **11001**

**Divisor** **/ 101**

```
101) 11001
     101
     ---
      101
      101
      ---
        0
```

**Quotient** **101**

# A Final Conversion Note

- **You can use arithmetic in other bases if you are careful:**
- **Example: Convert $101110_2$ to Base 10 using binary arithmetic:**

**Step 1 101110 / 1010 = 100 r 0110**

**Step 2 100 / 1010 = 0 r 0100**

**Converted Digits are $0100_2$ | $0110_2$**

**or 4 6 $_{10}$**