# Chapter -0

## Introduction to computer organization

Reference: **Lyla B. Das, Embedded Systems: An Integrated Approach , Pearson**

# UCS704: EMBEDDED SYSTEMS DESIGN

| L | T | P | Cr |
|---|---|---|---|
| 2 | 0 | 2 | 3.0 |

**Course Objectives:** To learn the concepts of embedded system and services in addition with its implementation for assessment of understanding the course by the students

**Introduction to Embedded systems:** Application domain of embedded systems, Desirable features and general characteristics of embedded systems, Model of an Embedded System, Microprocessor vs. Micro-controller, Example of a Simple embedded system, Figures of merit for an embedded system, Classification of Scum: 4/8/16/32 Bits, History of embedded systems, Current trends.

**Embedded Systems – The hardware point of view:** Micro-controller Unit (MCU), A Popular8-bit MCU, Memory for embedded systems, Low power design, Pull-up and pull-down resistors.

**Sensors, ADCs and Actuators:** Sensors, Analog to Digital Converters, Actuators.

**Real – time Operating Systems:** Real-time tasks, Real-time systems, Types of Real-time tasks, Real-time operating systems, Real- time scheduling algorithms, Rate Monotonic Algorithm, The Earliest deadline first algorithm, Qualities of a Good RTOS.

**DSP Processor:** The Application Scenario, General features of Digital Signal Processors, SIMD Techniques

**Automated design of Digital IC's:** History of integrated circuit(IC) design, Types of Digital IC's, ASIC design, ASIC design: the complete sequence.

**Hardware Software Co-design and Embedded Product development lifestyle management:** Hardware Software Co-design, Modeling of Systems, Embedded Product Development Lifestyle Management, Lifestyle Models.

**Internet of Things:** Sensing and Actuation from Devices, Communication Technologies, Multimedia Technologies, Circuit Switched Networks, Packet Switched Networks.

**Self-Learning Content:**

**Basics of computer architecture and the binary number system:** Basics of computer architecture, Computer languages, RISC and CISC architectures, Number systems, Number format conversions, Computer arithmetic, Units of memory capacity.

**Examples of Embedded Systems:** Mobile Phone, Automotive Electronics, Radio frequency Identification (RFID), Wireless sensor networks (WISENET), Robotics, Biomedical Applications, Brain machine interface.

**Laboratory Work:**

To design and simulate list of combinational and sequential digital circuits using Modelsim& Xilinx – Verilog language. To design and simulate the operations of systems like Verilog using Modelsim& Toggle, Bitwise, Delay and any Control Logic Design in 8051.

**Course Learning Outcomes (CLOs) / Course Objectives (COs):**

On completion of this course, the students will be able:

1. Identify the need and usage of Embedded System.
2. Distinguish different type of real-time tasks and apply different real-time scheduling algorithms.
3. Describe the kind of memory and processor.
4. Design and verify different type of combinational and sequential digital circuits using Hardware Description Language.
5. Discuss field programmable gate array (FPGA) and its application.
6. Outline the concept of Internet of Things.

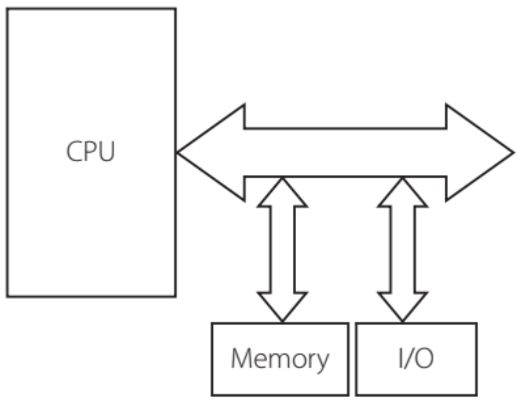
**Text Book:**

1. Das Lyla B., Embedded Systems: An Integrated Approach (2013) 1st ed.

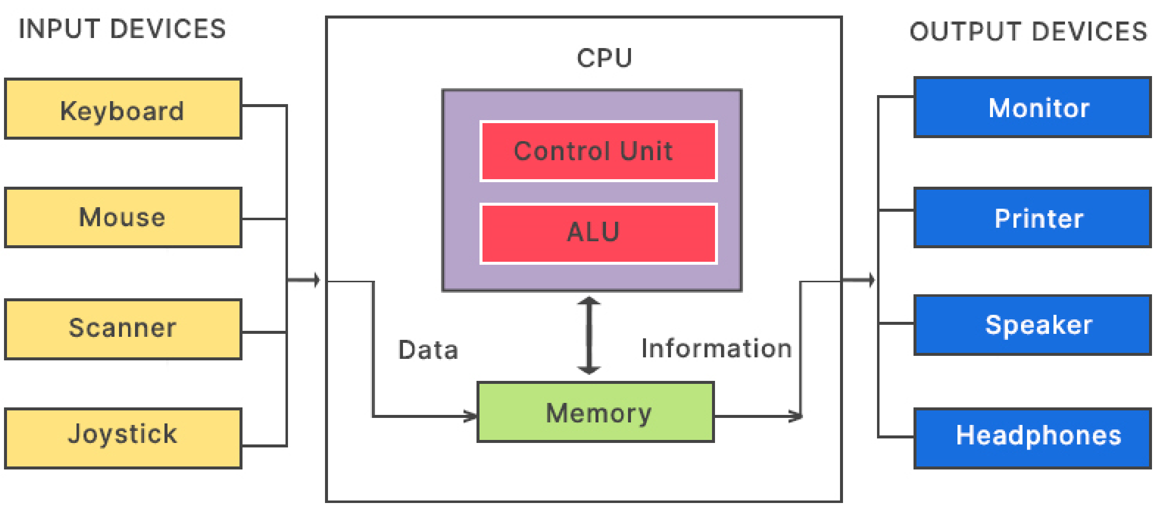
**Reference Book:**

1. KamalRaj, Embedded Systems Architecture, Programming and Design (2003)1st ed.

# Computer




The block diagram of a computer




Basic Architecture Of a Computer

# System Bus

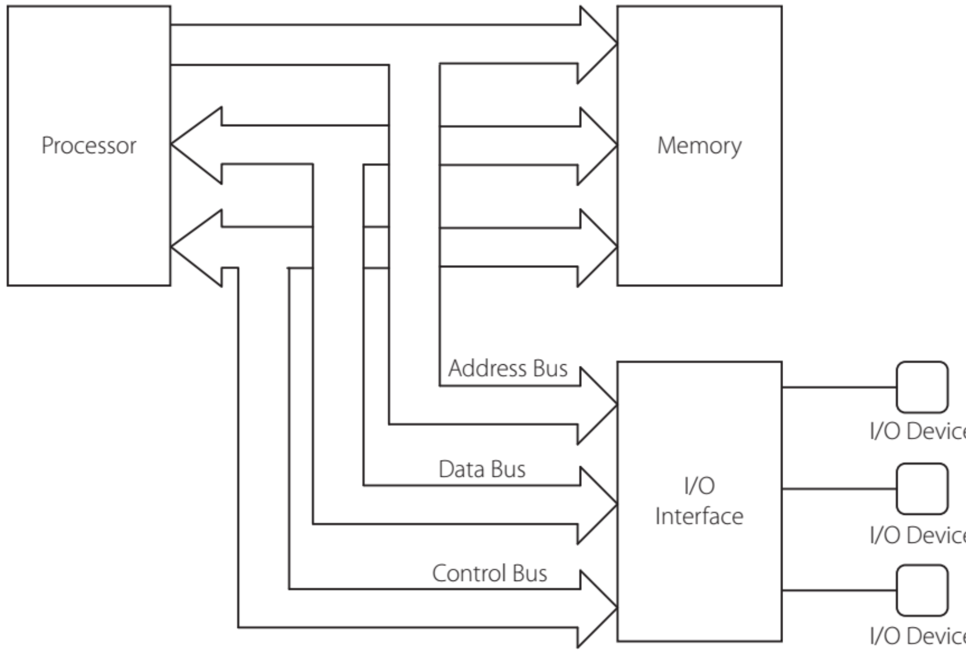



**Figure 0.2** | The system bus and its components

- A **bus** is **collection of signal wires** which connect between the components of the computer systems—Figure 0.2 shows that the CPU is connected to the memory as well as I/O through the system bus, but only one at a time—if the memory and I/O wants to use the bus at the same time, there is a conflict, as there is only one system bus.
- The system bus comprises of the **address bus**, **data bus** and the **control bus**.

# System Bus

***Data Bus:***

- The **set of lines used to transfer data is called the data bus**.
- It is a **bidirectional bus**, as data has to be sent from the CPU to memory and I/O, and has to be received as well by the CPU.
- The **width** of the data bus determines the **data transfer rate**, **size** of the internal registers of the CPU and the processing capability of the CPU.
- In short, it is a reflection of the complexity of the processor.
- As we see, the **8086** has a data bus width of **16-bits**, while the **80486** has a **32-bits** bus width.
- Thus the 80486 can process data of 32 bits at a time while the 8086 can only handle 16 bits.

# System Bus

***Address Bus:***

- The address bus width determines the maximum size of the physical memory that the CPU can access.
- With an address bus width of **20 bits**, the 8086 can address 220 different locations.
- It can use a memory size of **$2^{20}$ bytes or 1 MB.**
- For Pentium with an address bus width of **32 bits**, the corresponding numbers are **$2^{32}$ bytes i.e., 4 GB**.
- When a particular memory location is to be accessed, the corresponding address is placed on the address bus by the CPU.
- I/O devices also have addresses.
- In both cases, it is the CPU which supplies the address, and as such, the address bus is unidirectional.

# System Bus

***Control Bus:***

- The control bus is a set of control signals which needs to be activated for activities like **writing/reading to/from memory/I/O**, or **special activities of the CPU** like **interrupts** and **DMA**.
- Thus, we see signals like **Memory Read, I/O Read, Memory Write and Interrupt Acknowledge** as part of the control bus.
- These control signals dictate the actions taking place on the system bus that involve communications with devices like memory or I/O.
- **For example,**
    - **Memory Read signal** will be asserted for reading from memory. It is sent to memory from the processor.
    - A signal such as '**Interrupt**' is received by the processor from an I/O device.
- Hence in the control bus, we have signals traveling in either direction.
- Some control lines may be bidirectional too.

# Processor

- The **processor** or the **microprocessor** as we might call it, is the component responsible for controlling all the activity in the system.
- It performs the following **three actions** continuously:
  i) **Fetch** an instruction from memory.
  ii) **Decode** the instruction.
  iii) **Execute** the instruction.

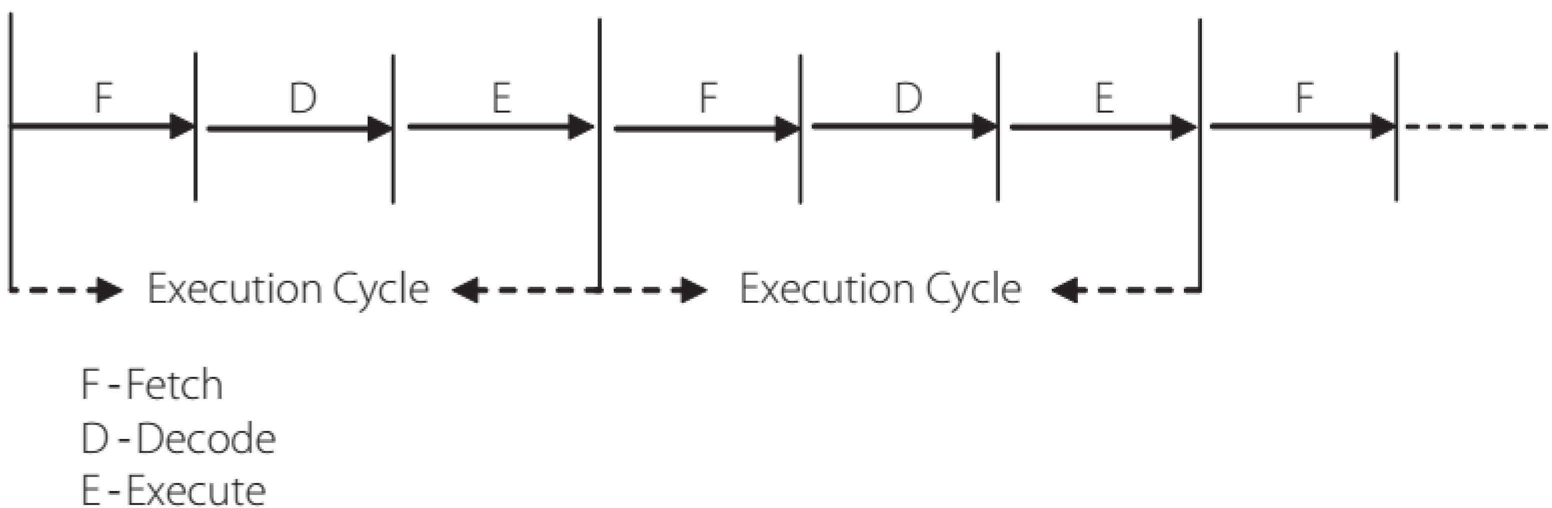



**Figure 0.3** | The execution cycle

# Processor

- When we write a program, it is stored in memory.
- Our code has to be brought to the processor for the required action to be performed.
  - The first step obviously, is to '**fetch**' it from memory.
  - The next step i.e., **decoding**, involves the interpretation of the code as to what action is to be performed.
  - After decoding, the **action required** is performed.
- This is termed '**instruction execution**' i.e. **execute.**
- The sequence of these three actions is called the '**execution cycle**.'
- Processor has two parts:
  - ALU
  - Control circuitry

# System Clock

- All activities of the processor is synchronized with a clock.
- A execution cycle may require many clock cycle.

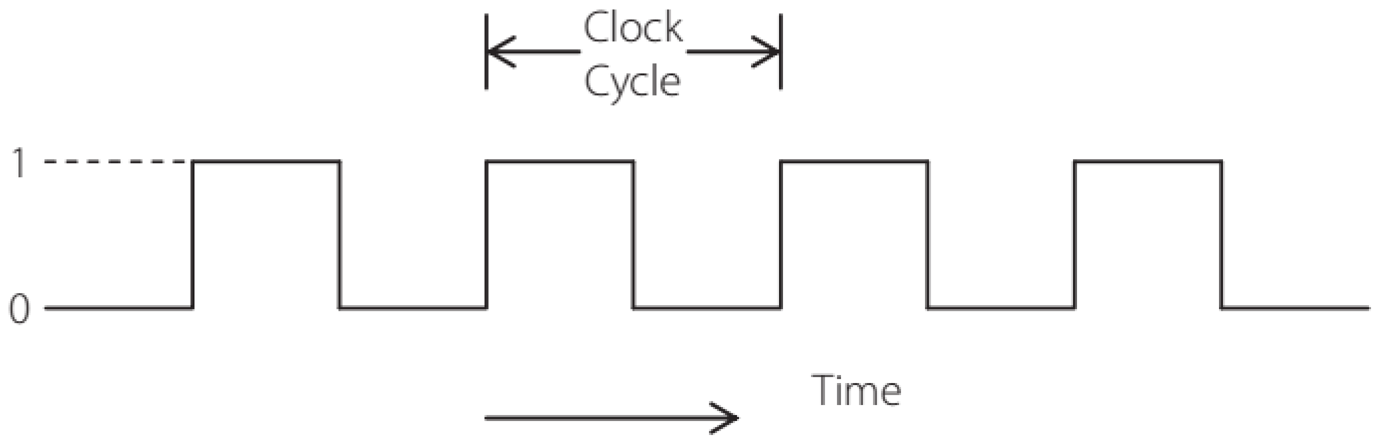



**Figure 0.4** | System clock

# Memory

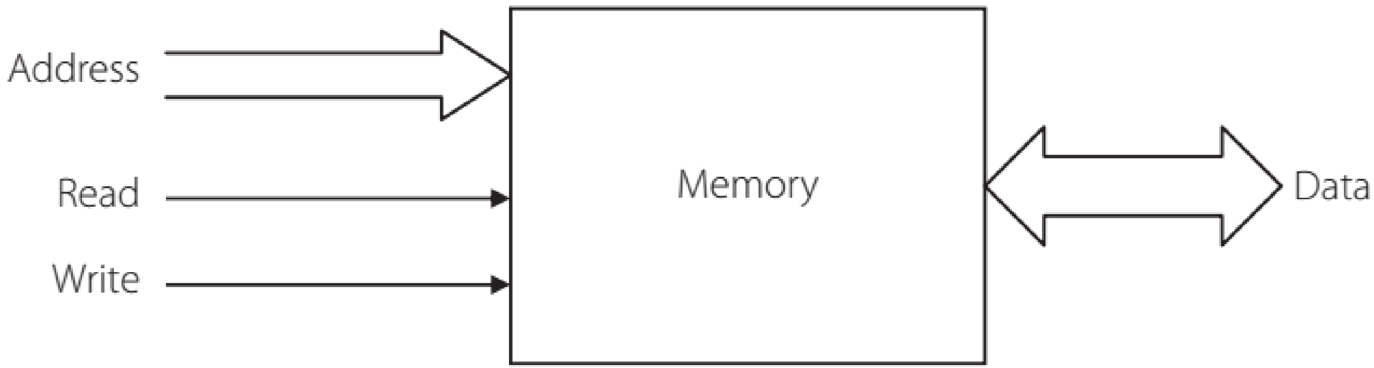



**Figure 0.5** | Memory and associated control signals

- The memory associated with a computer system includes the **primary memory (RAM)** as well as secondary memory.
- Memory is organized as **bytes**, and the capacity of a memory chip is stated in terms of the number of bytes it can store.
- Thus, we can have chips of size 256 bytes, 1KB, 1MB and so on.
- If a computer has a total memory space of 20 MB, it can use RAM chips of the available capacity to get that much of memory.
- There are two basic operations associated with memory— **read** and **write**.

# Memory Read Cycle

***Memory Read Cycle*** The steps involved in a typical read cycle are:

i) Place on the address bus, the address of the memory location whose content is to be read. This action is performed by the processor.
ii) Assert the memory read signal which is part of the control bus.
iii) Wait until the content of the addressed location appears on the data bus.
iv) Transfer the data on the data bus to the processor.
v) De-activate the memory read signal. The memory read operation is over and the address on the address bus is not relevant anymore.

# Memory Write Cycle

***Memory Write Cycle*** As a continuation, let us also examine the steps in a typical write cycle.

i) Place on the address bus, the address of the location to which data is to be written.
ii) On the data bus, place the data to be written.
iii) Assert the memory write signal which is part of the control bus.
iv) Wait until the data is stored in the addressed location.
v) De-activate the memory write signal. This ends the memory write operation.

- At this stage, we should remember that **these operations are synchronized with the system clock**.

- An **8086 processor** takes at least **four clock cycles for reading/writing**.
- These four cycles constitute the 'memory read' and 'memory write' cycles for the processor.
- Other processors may require more/less clock cycles for the same operations.

# IO subsystem

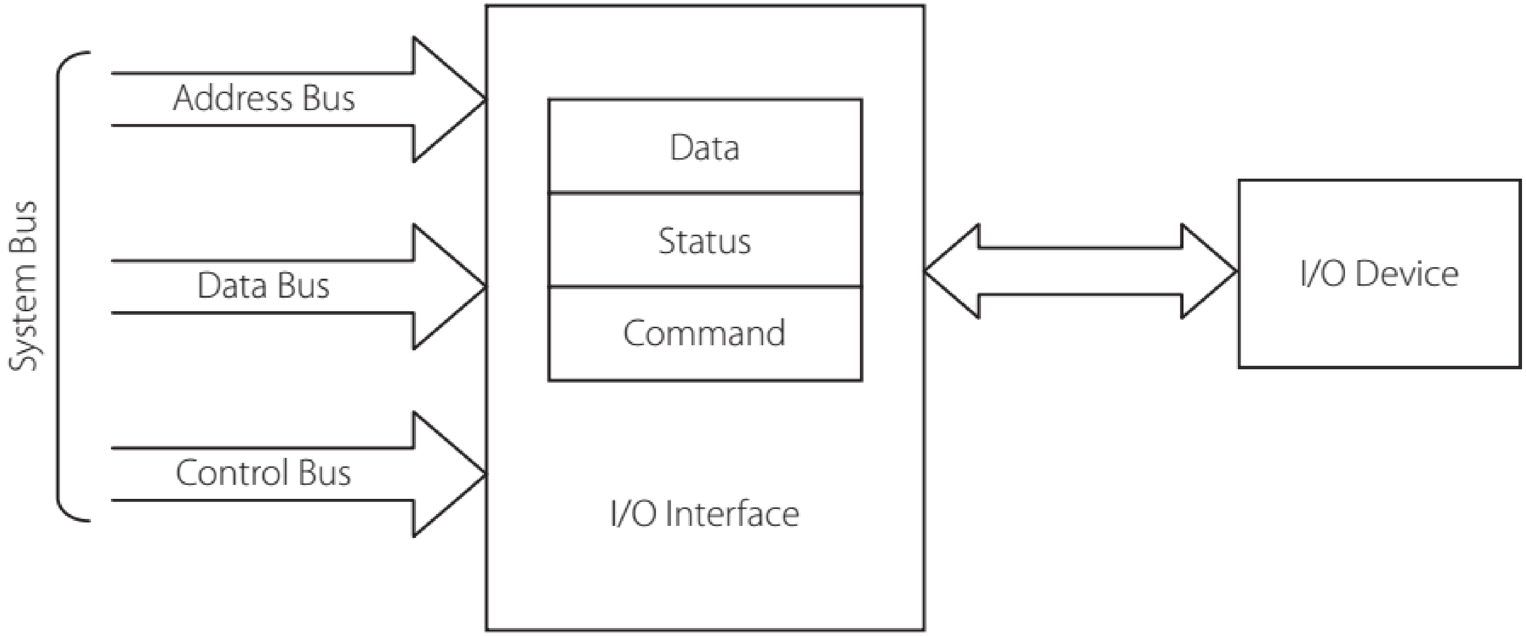



**Figure 0.6** | The I/O system

For a computer to communicate with the outside world there is the need for what are called peripherals.

**Examples:** keyboard, mouse, printer, and video monitor, some modem transfer data in both directions etc.

# High Level Language

- The high-level languages are much closer to human language.
- A programming language such as C, FORTRAN or Pascal that enables to write programs which is understandable to programmer (Human) and can perform any sort of task.
- High level language must use interpreter, compiler or translator to convert human understandable program to computer readable code (machine code).

## Examples:

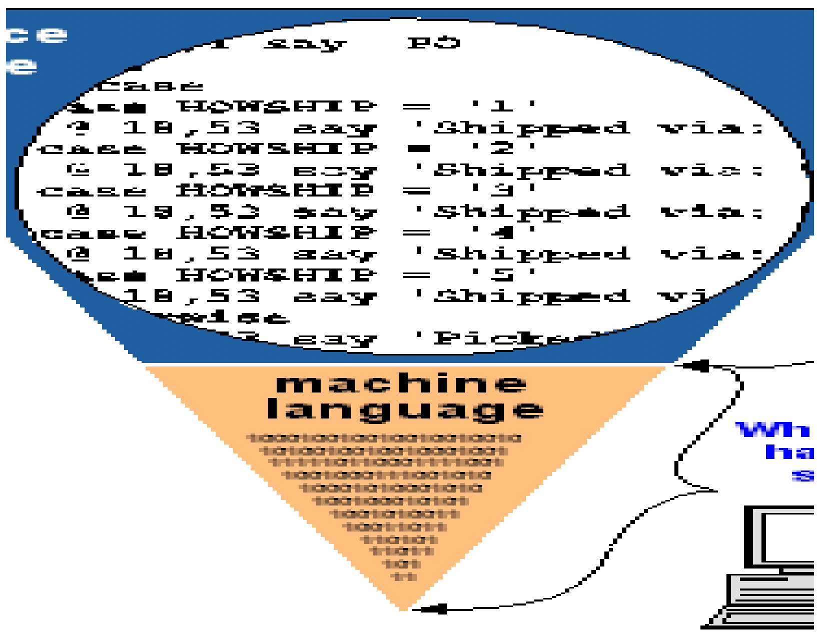
| | |
|---|---|
| **COBOL** | **Business applications** |
| **FORTRAN** | **Engg & Scientific Applications** |
| **PASCAL** | **General use and as a teaching tool** |
| **C & C++** | **General Purpose – currently most popular.** |
| **PROLOG** | **Artificial Intelligence** |
| **JAVA** | **General all purpose programming** |
| **.NET** | **General or web applications.** |

## Advantages of High over Low level language

- The main advantage of high-level languages over low-level languages is that they are easier to read, write and maintain.
- High-level languages make complex programming simpler.
- High level programming techniques are applicable everywhere even where computational resources are limited.
- Error ratio is less in high level language and debugging (locate and correct errors in program code ) is easier.
- Length of the program is also small compared with low level.
- Many real time problems can be easily solved with high level language.

# Assembly Language

- A computer low level language that deals with hardware registers by name known as **assembly language**.
- **Assembly language** is the best example of low-level language, this i s in between machine language and high-level language.
- A low-level language does not need a compiler or interpreter to run the program, the processor run low-level code directly.



# Assembly Language

- Assembly languages enable a programmer to use **names instead** of numbers.
- Programmers still use assembly language when **speed is essential** or when they need to perform an operation that isn't possible in a high-level language.
- It uses **mnemonic codes** (short forms) for instructions and allows the programmer to introduce names for blocks of memory that hold data.

**Example:**

```
mov edx, [esp+8]
cmp edx,
```

# Machine language

- Machine code or machine language is a system of instructions and data executed directly by a computer's CPU, The lowest-level programming language that only be understood by computers.
- Computer language that is directly executable by a computer without the need for translation by a compiler or an assembler.
- **The native language of the computer,** The set of symbolic instructions in binary that is used to represent operations and data in a machine called machine code

# Program code converters

**INTERPRETER:**

Interpreter can convert a source code , usually on a step-by-step, line-by-line and unit-by-unit basis into machine code.

**COMPILER:**

Compiler is a program that compile source code into executable instructions that a computer can understand, it check the entire program for Syntex and semanti c errors.

**ASSEMBLER:**

Assembler normally converts assembly language's source code into machine lang uage.

**TRANSLATOR:**

Translator is a computer program that translates one programming language instru ction(s) into another programming language instruction(s)

# RISC and CISC

# RISC and CISC

- when it comes to understanding and designing microprocessor architectures, three concepts are in center:
  - Instruction Set, RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer).

# Instruction set

- There are certain instructions that the CPU knows and when we give them those instructions, different transistors inside it switch ON and OFF to perform those instruction.
- The instructions that we input are in the form of 1’s and 0’s, or opcode.
- we generally  use shorthand’s for those instructions, called assembly language, and a assembler converts it into opcode.
- The number of instructions that a particular CPU can have is limited and the collection of all those instructions is called **the Instruction Set.**
- The Instruction Set is very important. A proper design of hardware and instruction set can determine how fast the CPU is.

# CPU Performance

- The performance of a CPU is the number of programs it can run in a given time. The more the number of programs it can run in that time, the faster the CPU is.

- The performance is determined by the number of instructions that a program has:
  - more instructions, more time to perform them.

- It also depends upon the number of cycles (clock cycles) per instructions.

- This means that there are only two ways to improve the performance:
  - Either minimize the number of instructions per program,
  - or reduce the number of cycles per instruction.

# CISC ARCHITECTURE

- **CISC** is the shorthand for **Complex Instruction Set Computer**.
- The CISC architecture tries to reduce the number of Instructions that a program has, thus optimizing the Instructions per Program part of the above equation.
- This is done by combining many simple instructions into a single complex one.

# Example: MUL instruction

- This instruction takes two inputs: the memory location of the two numbers to multiply, it then performs the multiplication and stores the result in the first memory location.

**MUL 1200, 1201**

- This reduces the amount of work that the compiler has to do as the instructions themselves are very high level.

- The instructions take very little memory in the RAM and most of the work is done by the hardware while decoding instructions.

- Since in a CISC style instruction, the CPU has to do more work in a single instruction, so clock speeds are slightly slower.

- Moreover, the number of general purpose registers are less as more transistors need to be used to decode the instructions.

# RISC ARCHITECTURE

- **Reduced Instruction Set Computer or RISC** architectures have more instructions, but they reduce the number of cycles that an instruction takes to perform.

- Generally, a single instruction in a RISC machine will take only one CPU cycle.

- Multiplication in a RISC architecture cannot be done with a single MUL like instruction.

- Instead, we have to first load the data from the memory using the LOAD instruction, then multiply the numbers, and the store the result in the memory.
  - **Load A, 1200**
  - **Load B, 1201**
  - **Mul A, B**
  - **Store 1200, A**

- in RISC architectures, we can only perform operations on Registers and not directly on the memory.

- This might seem like a lot of work, but in reality, since each of these instructions only take up one clock cycle, the whole multiplication operation is completed in fewer clock cycles.

- RISC has simpler instruction sets, complex High-Level Instructions needs to be broken down into many instructions by the compiler.

- This puts a lot of stress on the software and the software designers, while reducing the work needed to be done by the hardware.

- The decoding logic is simple, transistors required are lesser and more number of general purpose registers can be fit into the CPU.

# comparison

- CISC tries to complete an action in as few lines of assembly code as possible, RISC tries to reduce the time taken for each instruction to execute.

- the MUL operation on two 8-bit numbers in the register, in 8086 which is a CISC device can take as much as 77 clock-cycles, whereas the complete multiplication operation in a RISC device like a PIC takes only 38 cycles

- Since CISC instructions take a more number of cycles to execute, parallelism and pipelining of instructions is much harder. In RISC however, since all instructions take one cycle, pipelining instructions is easier.
- the compiler plays an important role in RISC systems, and its ability to perform this “code expansion” can hinder performance.

# Final word: which is better

- CISC is most often used in automation devices whereas RISC is used in video and image processing applications.

- When microprocessors and microcontroller were first being introduced, they were mostly CISC. This was largely because of the lack of software support present for RISC development.

- Later a few companies started delving into the RISC architecture, most notable, Apple, but most companies were unwilling to risk it (pun intended) with an emerging technology.

# 1. Number Systems

# Common Number Systems

| System | Base | Symbols | Used by humans? | Used in computers? |
|---|---|---|---|---|
| Decimal | 10 | 0, 1, … 9 | Yes | No |
| Binary | 2 | 0, 1 | No | Yes |
| Octal | 8 | 0, 1, … 7 | No | No |
| Hexa-decimal | 16 | 0, 1, … 9, A, B, … F | No | No |

# Quantities/Counting (1 of 3)

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |

# Quantities/Counting (2 of 3)

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

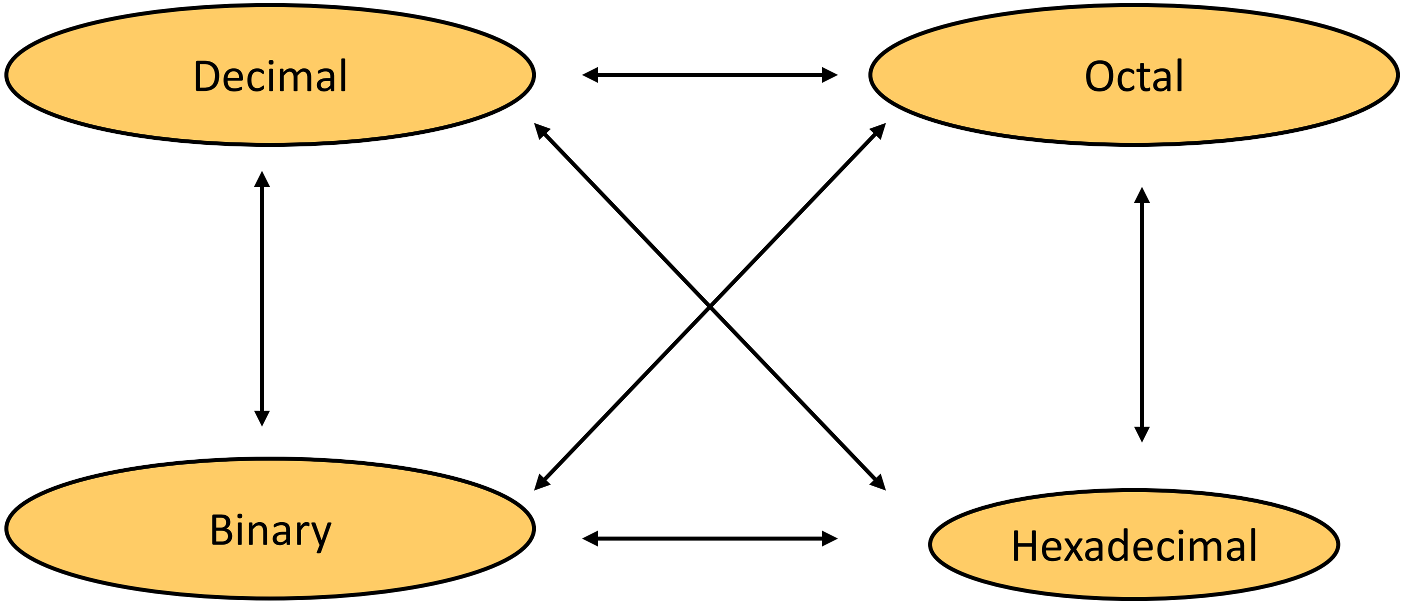
# Quantities/Counting (3 of 3)

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 16 | 10000 | 20 | 10 |
| 17 | 10001 | 21 | 11 |
| 18 | 10010 | 22 | 12 |
| 19 | 10011 | 23 | 13 |
| 20 | 10100 | 24 | 14 |
| 21 | 10101 | 25 | 15 |
| 22 | 10110 | 26 | 16 |
| 23 | 10111 | 27 | 17 |

Etc.

# Conversion Among Bases

- The possibilities:

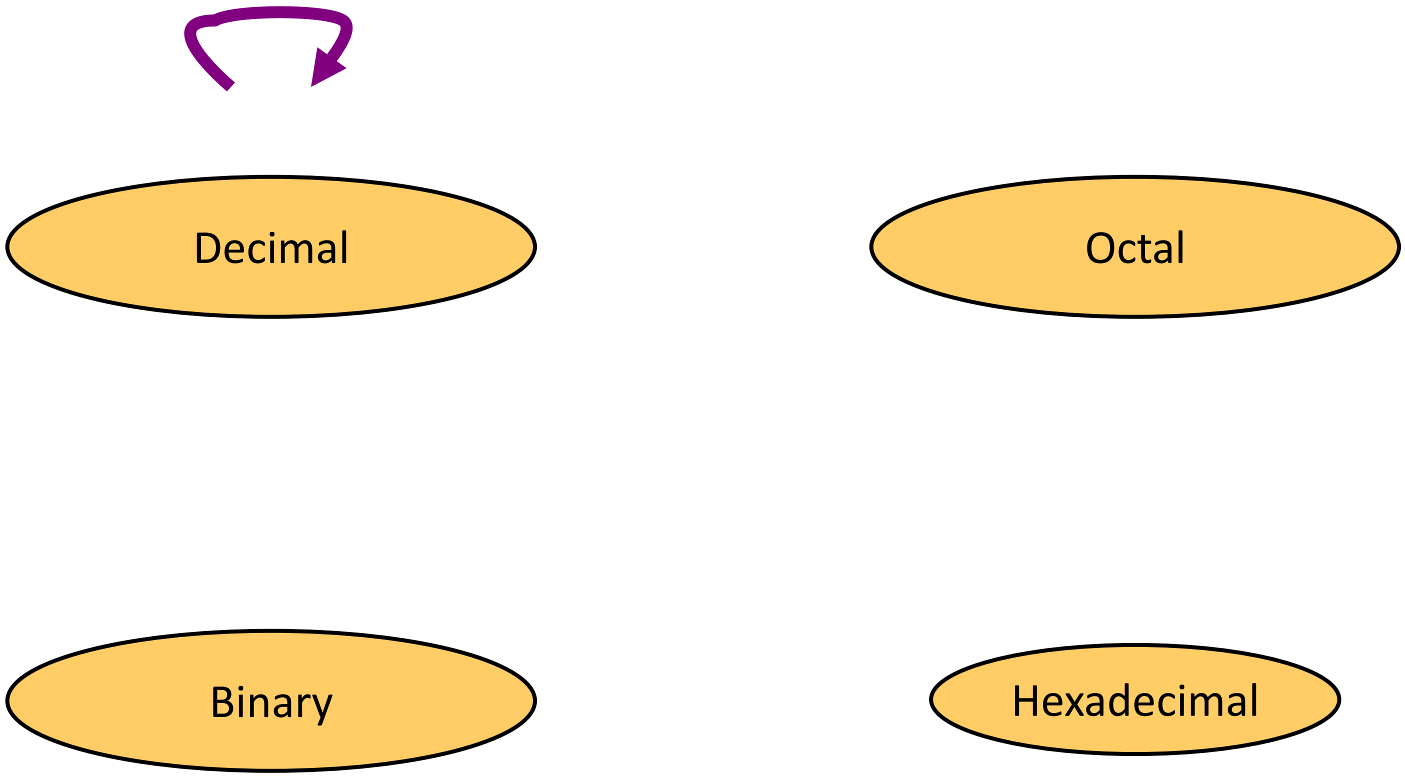


# Quick Example

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

Base

# Decimal to Decimal (just for fun)

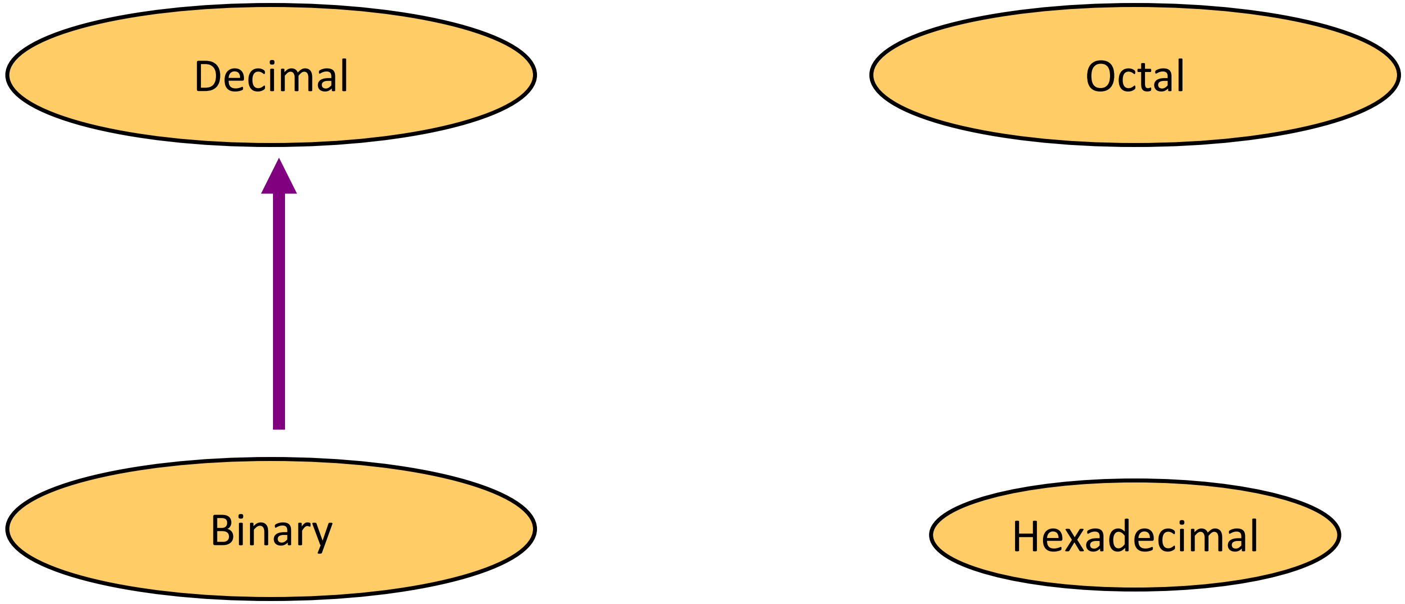


Weight

$$\begin{array}{lrcr} 125_{10} \Rightarrow & 5 \times 10^{0} & = & 5 \\ & 2 \times 10^{1} & = & 20 \\ & 1 \times 10^{2} & = & 100 \\ \cline{4-4} & & & 125 \end{array}$$

Base

# Binary to Decimal



# Binary to Decimal

- Technique
  - Multiply each bit by $2^n$, where $n$ is the "weight" of the bit
  - The weight is the position of the bit, starting from 0 on the right
  - Add the results

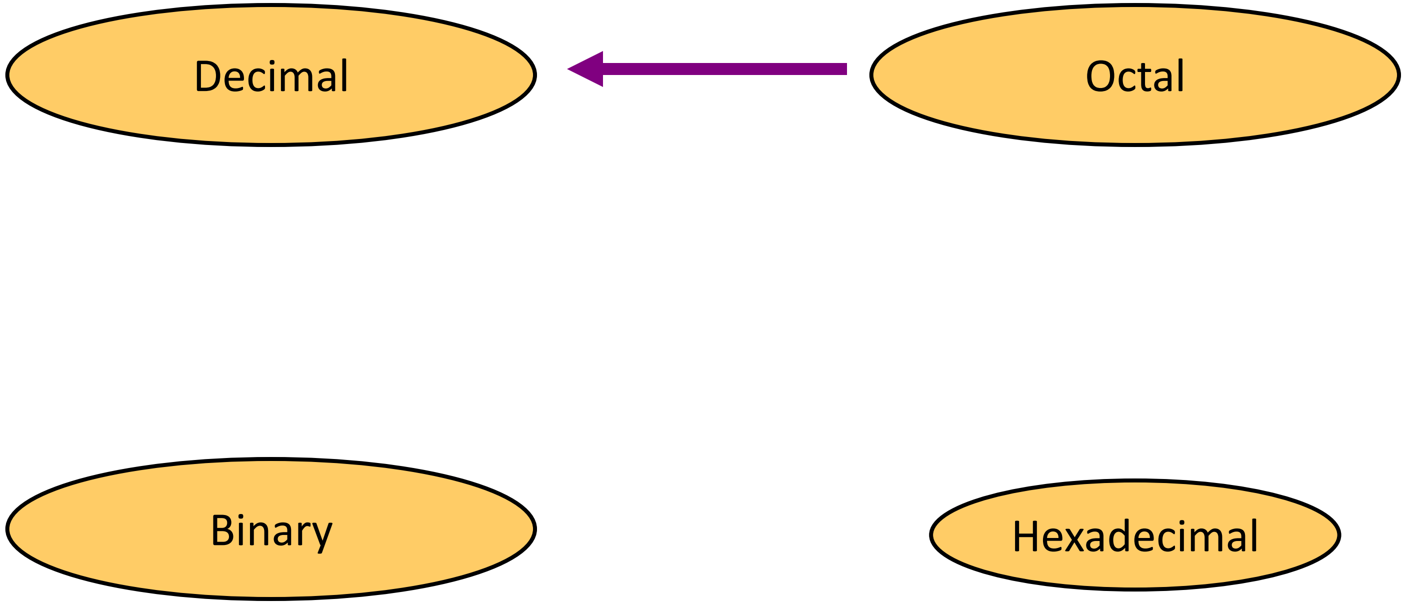
# Example

Bit "0"

$101011_2$ =>

$$
\begin{array}{rcl}
1 \times 2^0 & = & 1 \\
1 \times 2^1 & = & 2 \\
0 \times 2^2 & = & 0 \\
1 \times 2^3 & = & 8 \\
0 \times 2^4 & = & 0 \\
1 \times 2^5 & = & 32 \\
\hline
 & & 43_{10}
\end{array}
$$

# Octal to Decimal



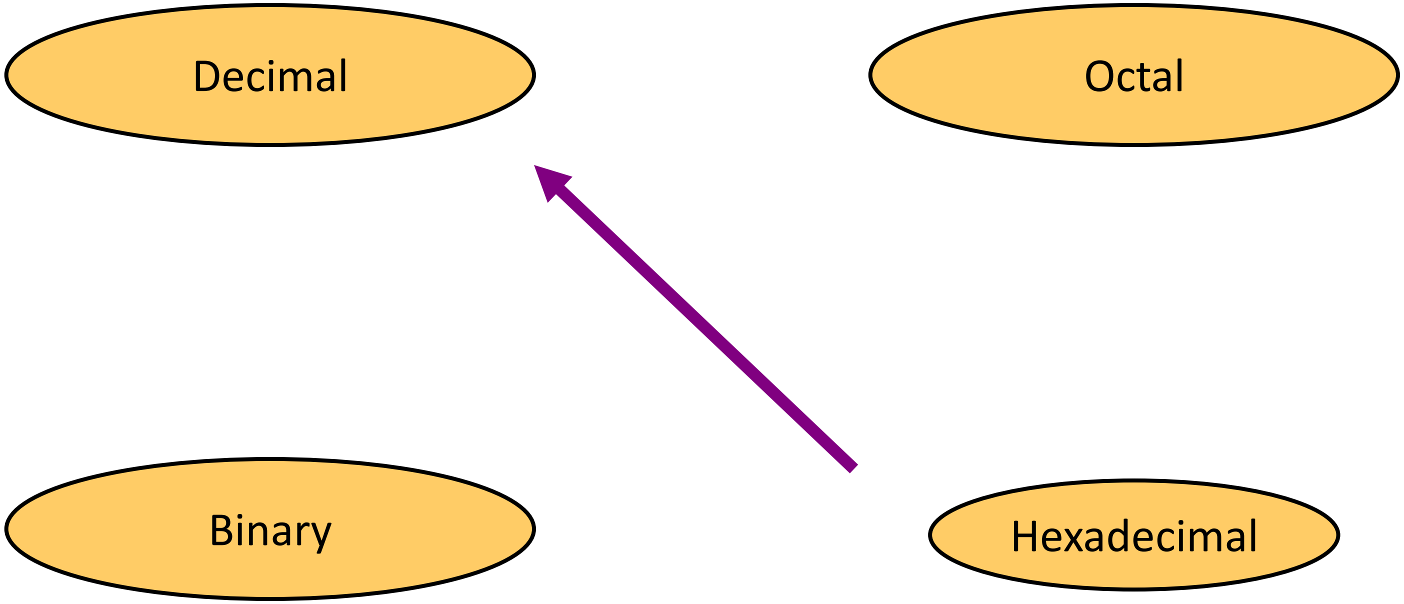
# Octal to Decimal

- Technique
  - Multiply each bit by $8^n$, where *n* is the "weight" of the bit
  - The weight is the position of the bit, starting from 0 on the right
  - Add the results

# Example

$$
\begin{array}{llcrl}
724_8 \Rightarrow & 4 \times 8^0 & = & 4 \\
 & 2 \times 8^1 & = & 16 \\
 & 7 \times 8^2 & = & 448 \\
\hline
 & & & 468_{10}
\end{array}
$$

# Hexadecimal to Decimal



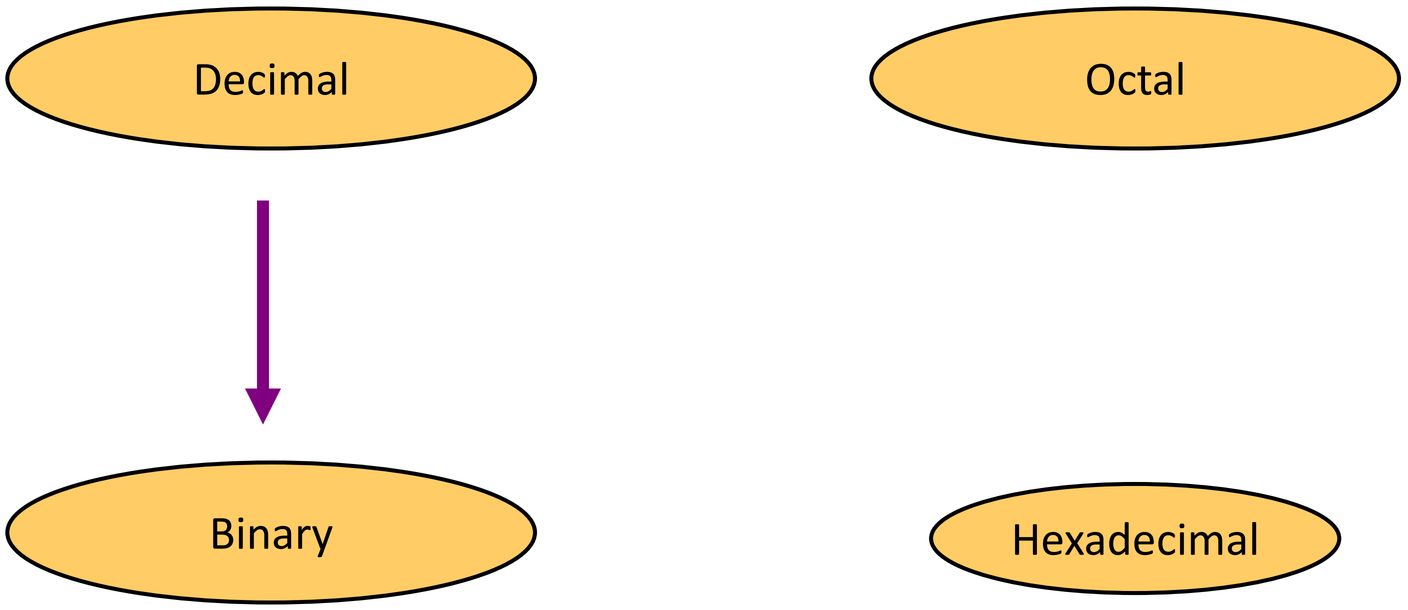
# Hexadecimal to Decimal

- Technique
  - Multiply each bit by $16^n$, where $n$ is the “weight” of the bit
  - The weight is the position of the bit, starting from 0 on the right
  - Add the results

# Example

$$
\begin{array}{llrcrcrcr}
ABC_{16} \Rightarrow & C \times 16^0 & = & 12 & \times & 1 & = & 12 \\
 & B \times 16^1 & = & 11 & \times & 16 & = & 176 \\
 & A \times 16^2 & = & 10 & \times & 256 & = & \underline{2560} \\
 & & & & & & & 2748_{10}
\end{array}
$$

# Decimal to Binary



# Decimal to Binary

- Technique
  - Divide by two, keep track of the remainder
  - First remainder is bit 0 (LSB, least-significant bit)
  - Second remainder is bit 1
  - Etc.

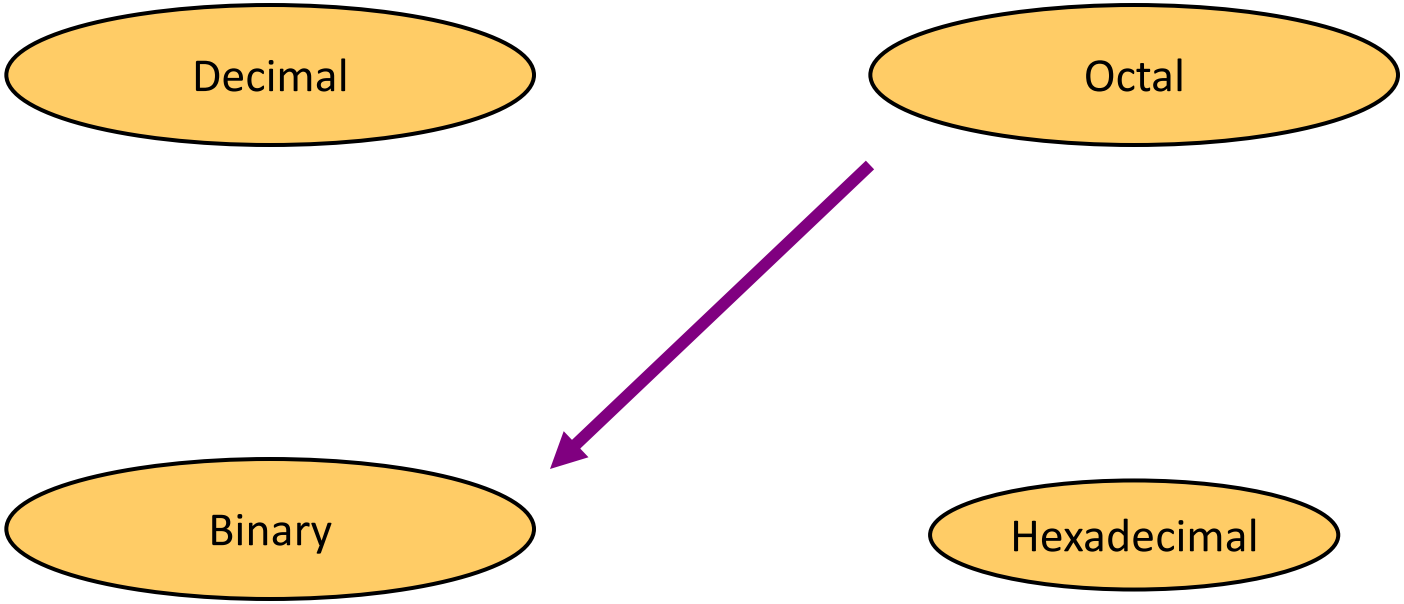
# Example

$125_{10} = ?_2$

| | | |
|---|---|---|
| 2 | 125 | |
| 2 | 62 | 1 |
| 2 | 31 | 0 |
| 2 | 15 | 1 |
| 2 | 7 | 1 |
| 2 | 3 | 1 |
| 2 | 1 | 1 |
| | 0 | 1 |

$125_{10} = 1111101_2$

# Octal to Binary



# Octal to Binary

- Technique
  - Convert each octal digit to a 3-bit equivalent binary representation

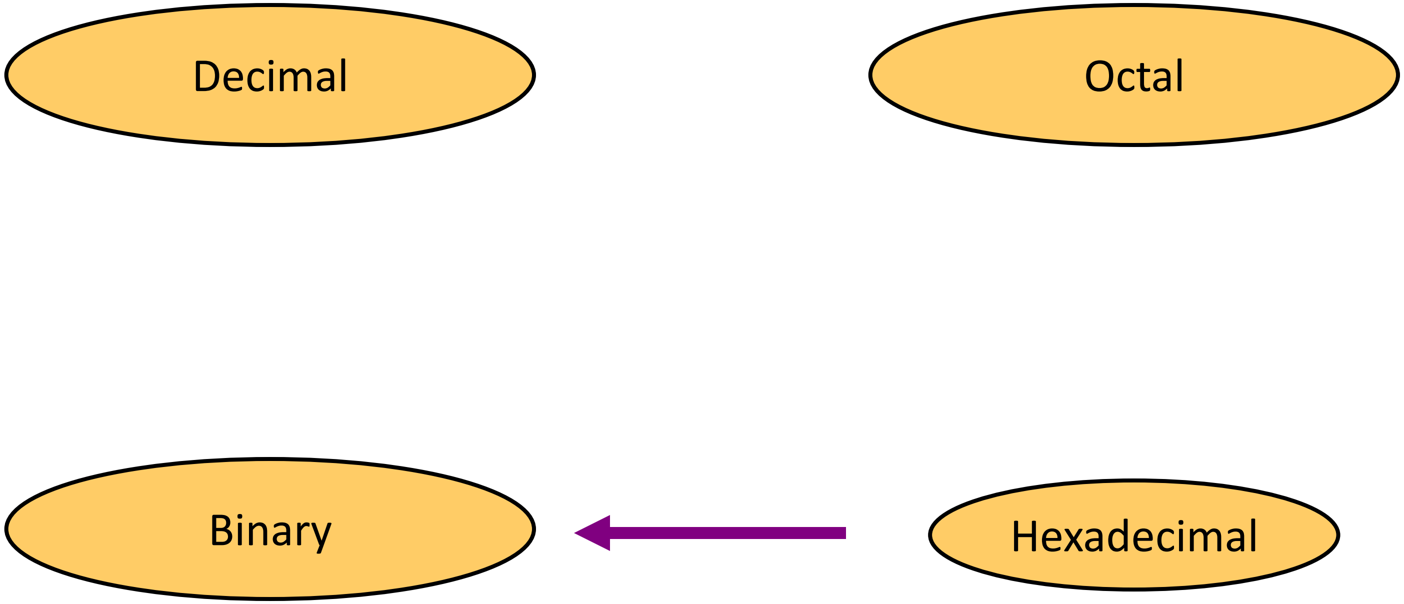
# Example

$705_8 = ?_2$

$$\begin{array}{ccc} 7 & 0 & 5 \\ \downarrow & \downarrow & \downarrow \\ 111 & 000 & 101 \end{array}$$

$705_8 = 111000101_2$

# Hexadecimal to Binary



# Hexadecimal to Binary

- Technique
  - Convert each hexadecimal digit to a 4-bit equivalent binary representation

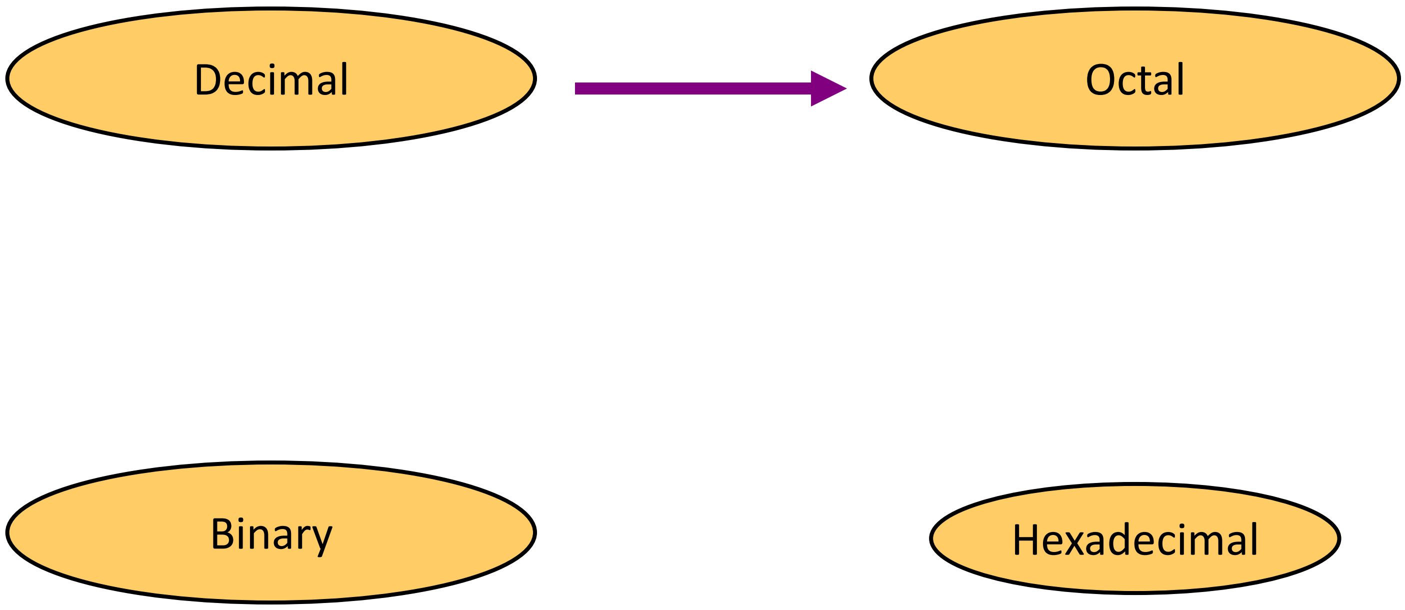
# Example

$10AF_{16} = ?_2$

| 1 | 0 | A | F |
|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ |
| 0001 | 0000 | 1010 | 1111 |

$10AF_{16} = 0001000010101111_2$

# Decimal to Octal



# Decimal to Octal

- Technique
  - Divide by 8
  - Keep track of the remainder

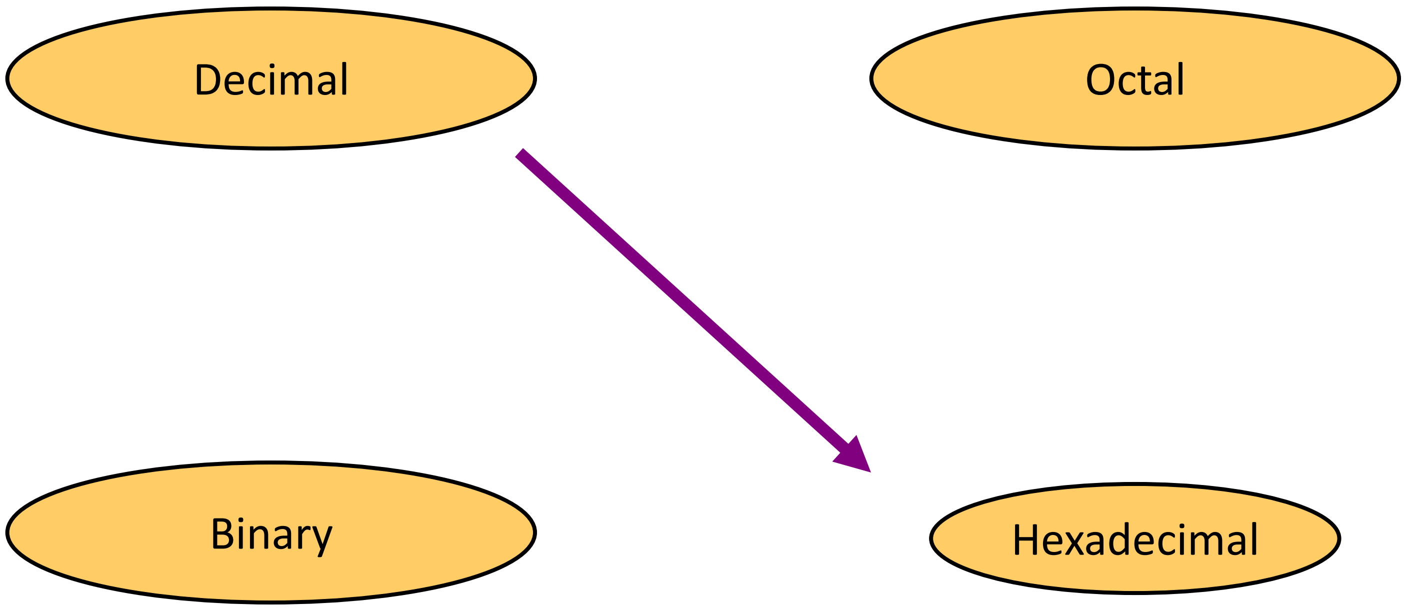
# Example

$1234_{10} = ?_8$

| | | |
|---|---|---|
| 8 | 1234 | |
| 8 | 154 | 2 |
| 8 | 19 | 2 |
| 8 | 2 | 3 |
| | 0 | 2 |

$1234_{10} = 2322_8$

# Decimal to Hexadecimal



# Decimal to Hexadecimal

- Technique
  - Divide by 16
  - Keep track of the remainder

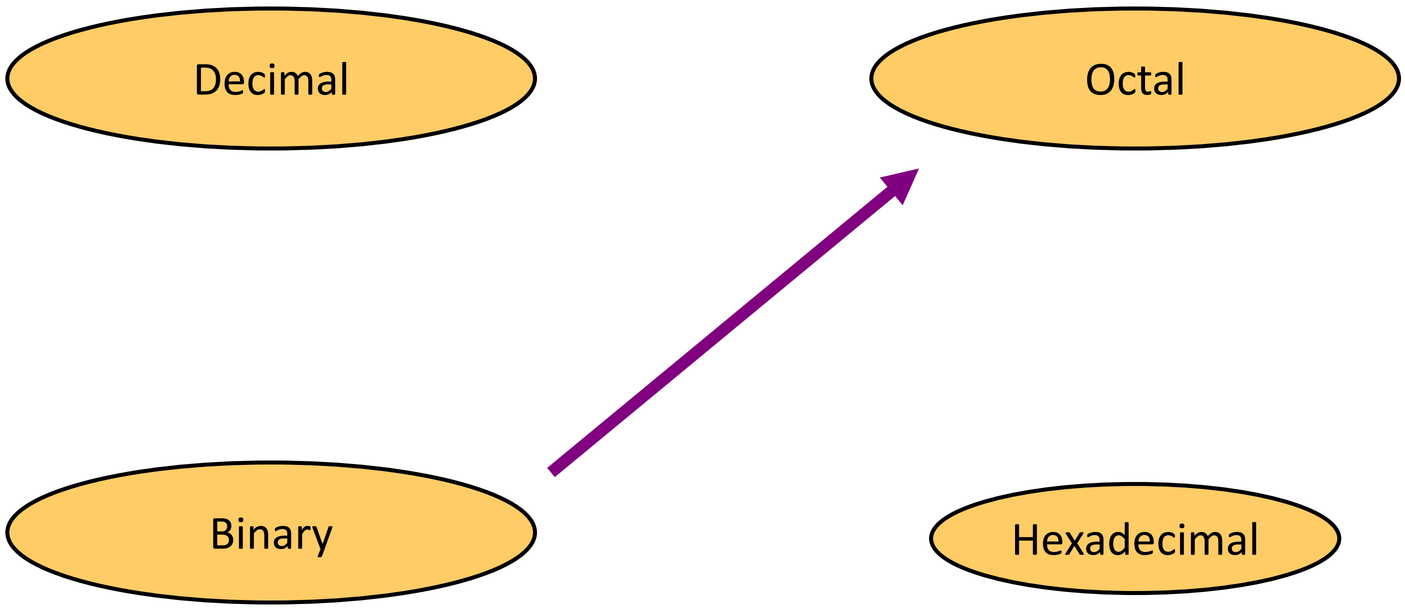
# Example

$1234_{10} = ?_{16}$

$$\begin{array}{r|l l}
16 & 1234 & \\ \hline
16 & 77 & 2 \\ \hline
16 & 4 & 13 = D \\ \hline
 & 0 & 4
\end{array}$$

$1234_{10} = 4D2_{16}$

# Binary to Octal



# Binary to Octal

- Technique
  - Group bits in threes, starting on right
  - Convert to octal digits

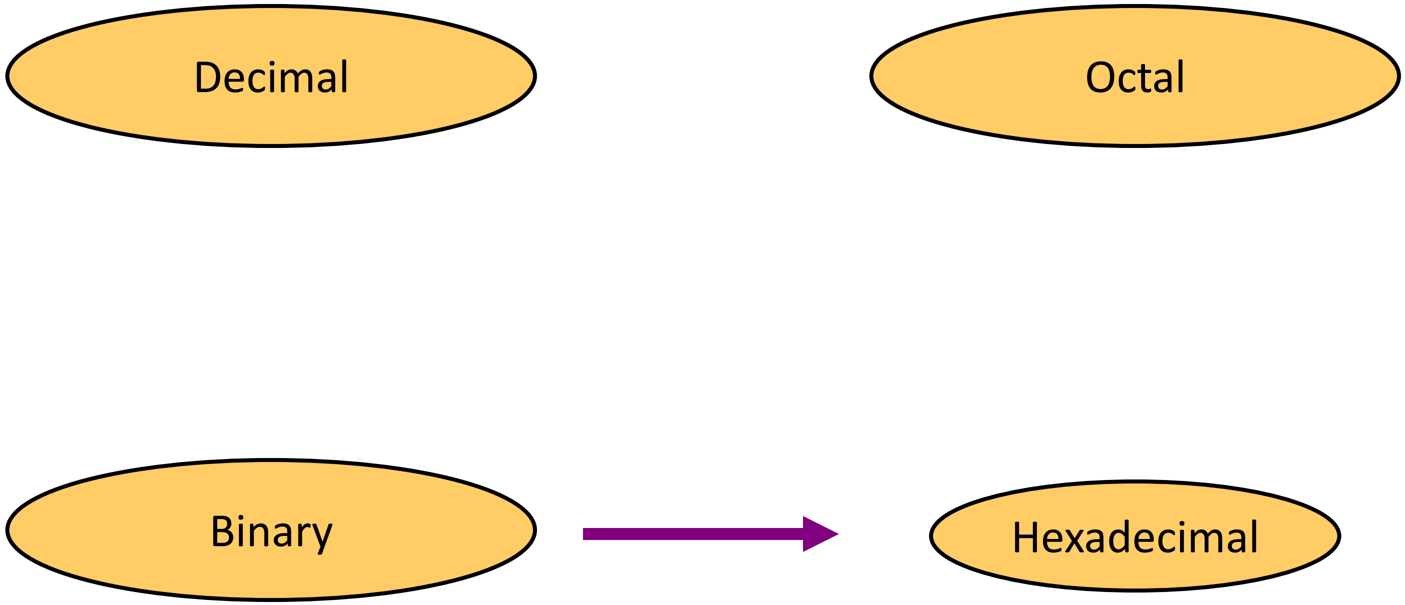
# Example

$1011010111_2 = ?_8$

$$\begin{array}{cccc} 1 & 011 & 010 & 111 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 3 & 2 & 7 \end{array}$$

$1011010111_2 = 1327_8$

# Binary to Hexadecimal



# Binary to Hexadecimal

- Technique
  - Group bits in fours, starting on right
  - Convert to hexadecimal digits

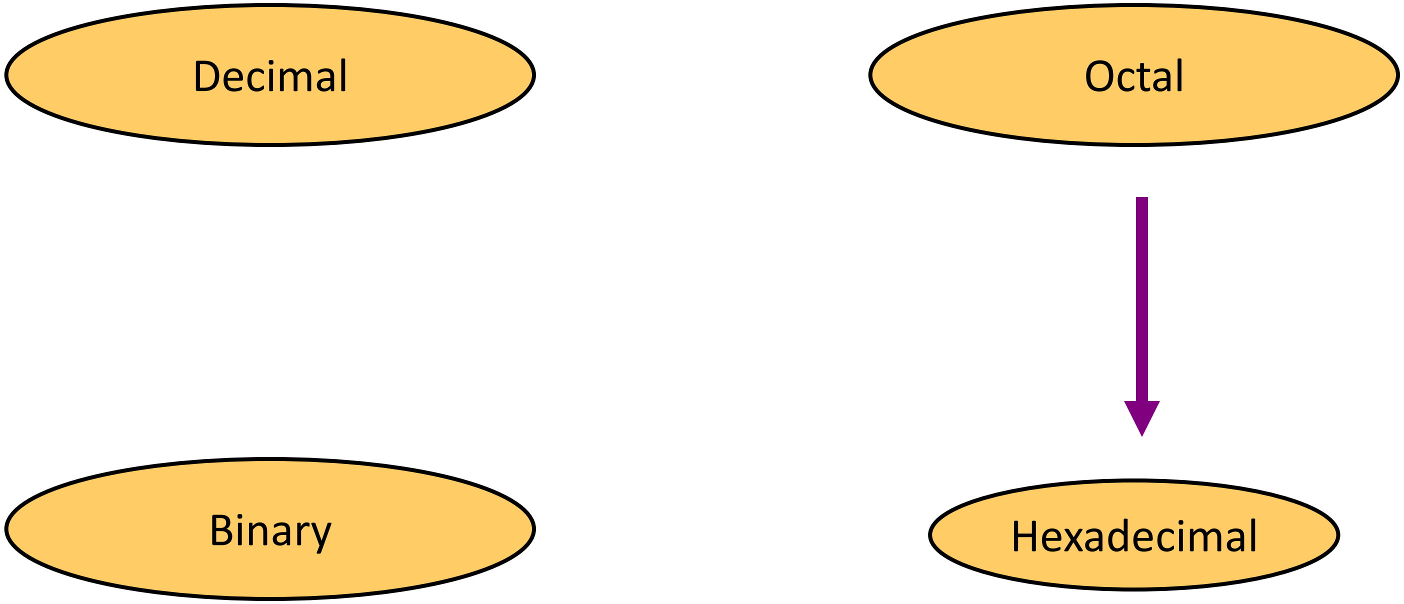
# Example

$1010111011_2 = ?_{16}$

$$\begin{array}{ccc} 10 & 1011 & 1011 \\ \downarrow & \downarrow & \downarrow \\ 2 & B & B \end{array}$$

$1010111011_2 = 2BB_{16}$

# Octal to Hexadecimal



# Octal to Hexadecimal

- Technique
  - Use binary as an intermediary

# Example

$1076_8 = ?_{16}$

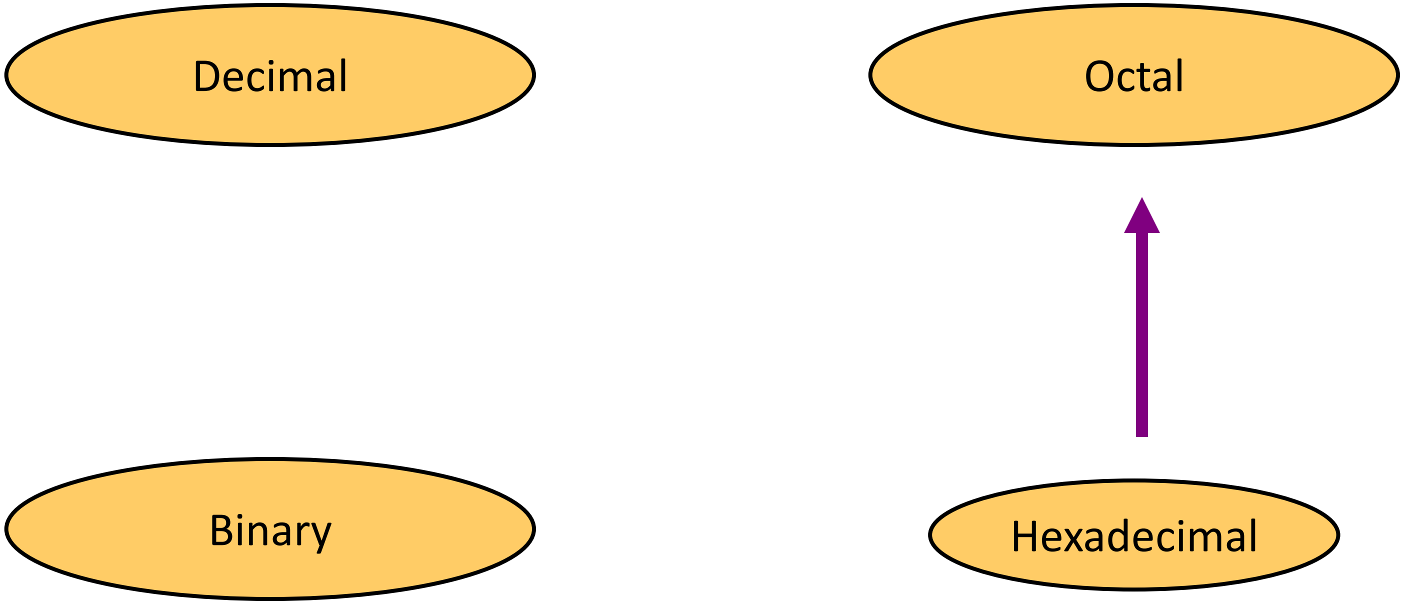
1 0 7 6

001 000 111 110

2 3 E

$1076_8 = 23E_{16}$

# Hexadecimal to Octal



# Hexadecimal to Octal

- Technique
  - Use binary as an intermediary

# Example

$1F0C_{16} = ?_8$

1 F 0 C

0001 1111 0000 1100

1 7 4 1 4

$1F0C_{16} = 17414_8$

# Exercise – Convert ...

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 33 | | | |
| | 1110101 | | |
| | | 703 | |
| | | | 1AF |

Don’t use a calculator!

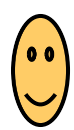
Skip answer

Answer

# Exercise – Convert …

Answer

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 33 | 100001 | 41 | 21 |
| 117 | 1110101 | 165 | 75 |
| 451 | 111000011 | 703 | 1C3 |
| 431 | 110101111 | 657 | 1AF |



# Common Powers (1 of 2)

- Base 10

| Power | Preface | Symbol | Value |
|---|---|---|---|
| $10^{-12}$ | pico | p | .000000000001 |
| $10^{-9}$ | nano | n | .000000001 |
| $10^{-6}$ | micro | μ | .000001 |
| $10^{-3}$ | milli | m | .001 |
| $10^{3}$ | kilo | k | 1000 |
| $10^{6}$ | mega | M | 1000000 |
| $10^{9}$ | giga | G | 1000000000 |
| $10^{12}$ | tera | T | 1000000000000 |

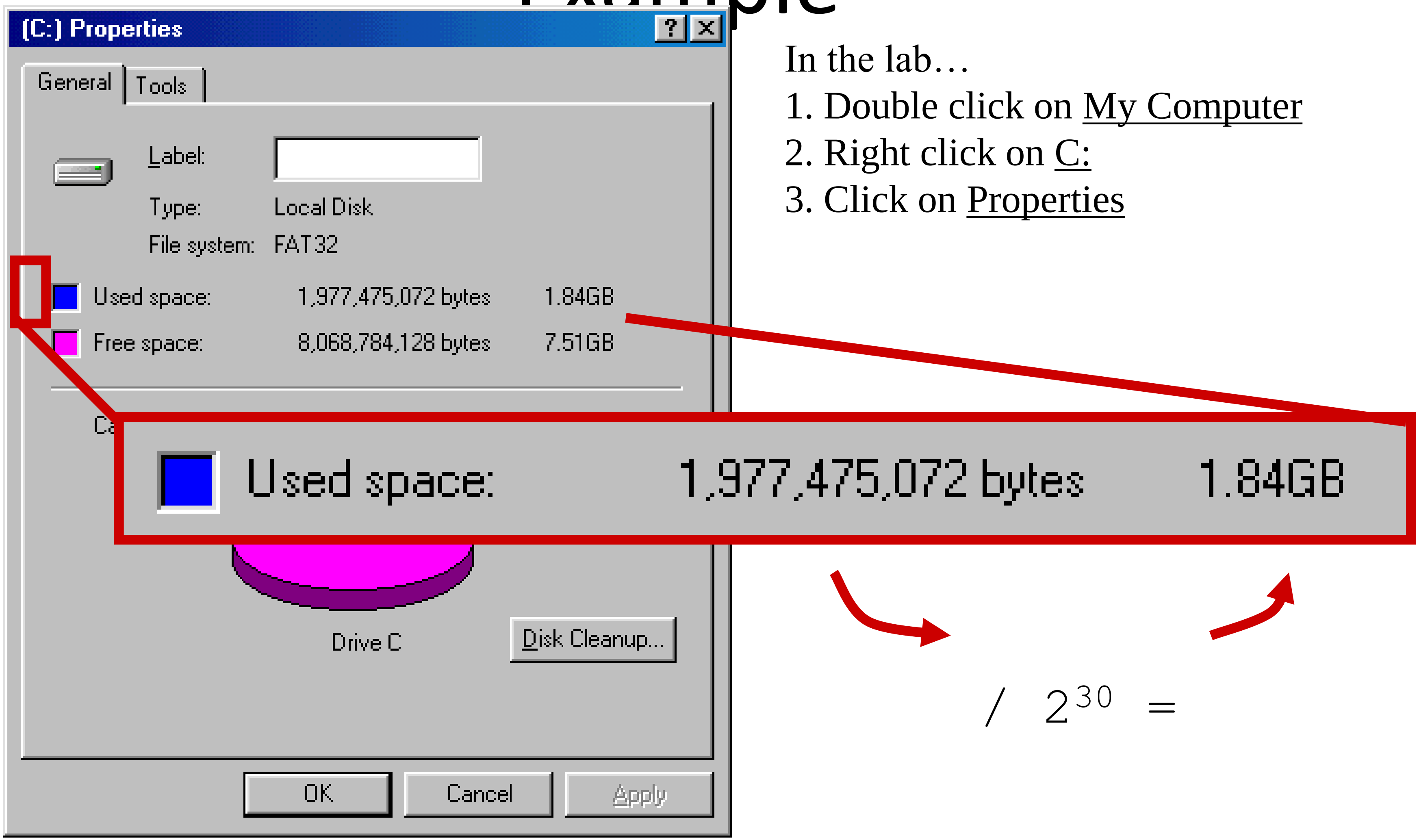
# Common Powers (2 of 2)

- Base 2

| Power | Preface | Symbol | Value |
|---|---|---|---|
| $2^{10}$ | kilo | k | 1024 |
| $2^{20}$ | mega | M | 1048576 |
| $2^{30}$ | Giga | G | 1073741824 |

- What is the value of “k”, “M”, and “G”?
- In computing, particularly w.r.t. <u>memory</u>, the base-2 interpretation generally applies

# Example




In the lab…

1. Double click on My Computer
2. Right click on C:
3. Click on Properties

$/ \ 2^{30} =$

# Exercise – Free Space

- Determine the “free space” on all drives on a machine in the lab

| Drive | Free space | |
|---|---|---|
| | Bytes | GB |
| A: | | |
| C: | | |
| D: | | |
| E: | | |
| etc. | | |

# Review – multiplying powers

- For common bases, add powers

$$a^b \times a^c = a^{b+c}$$

$$2^6 \times 2^{10} = 2^{16} = 65{,}536$$

or…

$$2^6 \times 2^{10} = 64 \times 2^{10} = 64k$$

# Fractions

- Decimal to decimal (just for fun)

$$
\begin{array}{lrcl}
3.14 \Rightarrow & 4 \times 10^{-2} & = & 0.04 \\
 & 1 \times 10^{-1} & = & 0.1 \\
 & 3 \times 10^{0} & = & \underline{3\phantom{.14}} \\
 & & & 3.14
\end{array}
$$

# Fractions

- Binary to decimal

$$
\begin{array}{llcl}
10.1011 \Rightarrow \quad & 1 \times 2^{-4} & = & 0.0625 \\
& 1 \times 2^{-3} & = & 0.125 \\
& 0 \times 2^{-2} & = & 0.0 \\
& 1 \times 2^{-1} & = & 0.5 \\
& 0 \times 2^{0} & = & 0.0 \\
& 1 \times 2^{1} & = & \underline{2.0} \\
& & & 2.6875
\end{array}
$$

# Fractions

- Decimal to binary

```
3.14579                                 .14579
                                       x     2
                                       0.29158
                                       x     2
                                       0.58316
                                       x     2
                                       1.16632
                                       x     2
                                       0.33264
                                       x     2
                                       0.66528
                                       x     2
                                       1.33056

11.001001...                           etc.
```

# Exercise – Convert ...

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 29.8 | | | |
| | 101.1101 | | |
| | | 3.07 | |
| | | | C.82 |

Don't use a calculator!

Skip answer

Answer

# Exercise – Convert ...

Answer

| Decimal | Binary | Octal | Hexa-decimal |
|---|---|---|---|
| 29.8 | 11101.110011… | 35.63… | 1D.CC… |
| 5.8125 | 101.1101 | 5.64 | 5.D |
| 3.109375 | 11.000111 | 3.07 | 3.1C |
| 12.5078125 | 1100.10000010 | 14.404 | C.82 |

# BCD, ASCII, Negative Binary Numbers, Addition, Subtraction

# Unsigned Numbers

- Unsigned number implies that the sign of the number is irrelevant
- We consider the numbers as having no sign bit
- All the bits allotted for the data are used for magnitude alone
- Refers to positive numbers
- With 8 bits, numbers from 0 to 255 can be used

# Representation of Negative Numbers

- Computers use 2’s complement for representation for negative numbers.
- 2’s complement= complement each bit of number and add ‘1’ to this.
- Eg 4-bit representation of -6 is
  i) Write 4 bit binary value of 6: 0110
  ii) Complement each bit: 1001
  iii) Add ‘1’ to this: 1010
- So, -6 is ‘1010’, for computers

Negative binary numbers: 2’s complement

Flip zeros and ones and add 1

| decade number | binary II K |
|---|---|
| -8 | 1000 |
| -7 | 1001 |
| -6 | 1010 |
| -5 | 1011 |
| -4 | 1100 |
| -3 | 1101 |
| -2 | 1110 |
| -1 | 1111 |
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |

# Observations

- The range of numbers that can be represented by 4 bits is -8 to +7.
- For an n-bit number, this range is $(-2^{n-1})$ to $(+2^{n-1}-1)$.
- The most significant bit (MSB) is considered as sign bit. The MSB for positive numbers is ‘0’ and for negative numbers is ‘1’.
- There is a unique representation for 0.

Q) Find 2's complement number corresponding to -6 when 6 is represented in 8 bits as

0000 0110.

A) Steps:

Write 8bit binary value of 6 (as given): 0000 0110

Complement each bit: 1111 1001

Add '1' to this: 1111 1010

F A H

Thus, -6 is FAH in 8- bit form, while it is AH in 4-bit form ( where H is notation for 'hexadecimal')

# Conversion from 2’s complement form

Given 2’s complement representation of a decimal number, how do we find the decimal number which it represents?

A) Take its 2’s complement again.

Eg. Given 2’s complement is 1110. What decimal number it represents?

A) 2’s complement of 1110 is 0010. (which is 2)

Thus, 1110 is the negative representation of -2

# Another Example

- We know that 1011 is the representation in binary for -5.
- But if we are only given 1011, how can we identify that it of magnitude 5?
- Solution is take 2’s complement of 1011.

  1011→ 0100 + 1 → 0101 which is 5.

# Binary Coded Decimal Numbers

- BCD Numbers
- Binary representation of decimal numbers
- Decimal numbers– 0 to 9 digits

- When we represent one decimal digit as byte, it is called ' Unpacked BCD'.
- Eg. 9 is written as 00001001 in unpacked BCD

# Unpacked BCD

- Eg. 98 in unpacked BCD is represented in two bytes (one byte for each digit)

| 9 | 8 |
|---|---|
| 00001001 | 00001000 |

Thus the binary code of each decimal digit is in one byte

# Packed BCD

- When each digit is packed into 4 binary digit, it is packed BCD.

  Eg. 98 in packed BCD is represented in four digits

| 9 | 8 |
|---|---|
| 1001 | 1000 |

- Packed BCD form of 675 is 0110 0111 0101
- Since there is no digit greater than 9, no BCD nibble can have a code greater than ‘1001’

# BCD numbers in hexadecimal form

- Decimal number 675 when written as 675H represents the packed BCD, in hex form.
- Steps:
  - Write binary equivalent of each decimal number, as a nibble,
  - Write the hex equivalent of each nibble

| | | | | |
|---|---|---|---|---|
| 675 is | 0110 | 0111 | 0101 | |
| | 6 | 7 | 5 | H |

Note: No digit of BCD in hex form will ever take value of A to F ( as decimal digits are limited to 9)

Q) Find binary, hex and packed BCD representation of decimal number 126. Also write packed BCD in hex format

# Solution

| | |
|---|---|
| Number | 126 |
| Binary | 0111 1110 |
| Hex | 7EH |
| BCD | 0001 0010 0110 |
| BCD in hex | 126H |

Q) Find binary, hex and packed BCD representation of decimal number 245. Also write packed BCD in hex format

# Solution

| | |
|---|---|
| Number | 245 |
| Binary | 1111 0101 |
| Hex | F5H |
| BCD | 0010 0100 0101 |
| BCD in hex | 245H |

Q) Find the packed BCD value of decimal number 2347654, and represent BCD in hex format.

A) 2347654

0010 0011 0100 0111 0110 0101 1001

2347654H

# Addition

- Binary
- Hexadecimal
- BCD
- Negative Number

# Binary Addition (1 of 2)

- Two 1-bit values

| A | B | A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 10 |

“two”

# Addition of Unsigned Numbers Case1:

- Decimal , binary and hex

| | | |
|---|---|---|
| 0101 1001 | Decimal: 89 | Hex: 59H |
| + 0110 1001 | +105 | 69H |
| 1100 0010 | 194 | C2H |

Since the result lies between 0 to 255, there is no special problem

# Addition of Unsigned Numbers: Case2

| | | |
|---|---|---|
| 0111 1000 | 120 + | 78H + |
| 1001 1001 | 153 | 99H |
| 10001 0001 | 273 | 111H |

Here, extra bit, beyond 8 bits is called ‘carry’. It indicates the insufficiency of the space allocated for the result. In microprocessors, there is a flag that indicates this condition.

# Binary Addition (2 of 2)

- Two *n*-bit values
  - Add individual bits
  - Propagate carries
  - E.g.,

```
 1   1
  10101       21
+ 11001     + 25
 ------     ----
 101110       46
```

# Binary addition

10101010 + 11001100 =

110011+001100=

# Binary addition

10101010 + 11001100 = **0101110110**

110011+001100= **0111111**

# Hexadecimal addition

| | | |
|---|---|---|
| 59H | 8ABH | 999H |
| 69H | B78H | ABCDH |
| ----- | --------- | --------- |
| C2H | 1423H | B566H |

# Hexadecimal addition

| | | |
|---|---|---|
| 79H | 898 | FFF |
| 79H | ABA | EEE |
| ----- | --------- | --------- |

# Hexadecimal addition

| | | |
|---|---|---|
| 79H | 898H | FFFH |
| 79H | ABAH | EEEH |
| ----- | --------- | --------- |
| F2H | 1352H | 1EEDH |

# Addition of Negative Numbers

- Negative numbers are represented in 2’s complement notation

Q) Add -43 and -56

A) Calculate 2’s complement of 43 and 56

[ 43= 00101011 , 2’s complement is 1101 0101]

[ 56= 00111000, 2’s complement is 1100 1000]

Now add both of them

# Addition of Negative Numbers

| | | |
|---|---|---|
| - 43 | | 1101 0101 |
| - 56 | + | 1100 1000 |
| -99 | | 1 1001 1101 |

Ignore this carry and look at the eight bits of the sum.

{ This is the rule for 2' complement addition}

Also, MSB is '1', represents that result is negative

# Addition of Negative Numbers

But magnitude of result 1001 1101 is not 99 (as we can see from decimal no. addition)

To find the decimal number whose 2’s complement representation this is, take 2’s complement of the result.

This comes to be 0110 0011 ie. 99

# Example1

## Q) Add +90 and -26

A) Calculate 2's complement of 26

[ 26= 00011010 , 2's complement is 1110 0110]

Now add this in +90

| | |
|---|---|
| +90 | 0101 1010 |
| - 26 | 1110 0110 |
| 64 | 1 0100 0000 |

Ignore this carry

Also, MSB is '0', represents that result is positive

So, convert it into decimal which comes out to be 64

# Example2

## Q) Add -120 and 45

A) Calculate 2's complement of -120

[ 120 = 01111000 , 2's complement is 1000 1000]

Now add this in +90

| | |
|---|---|
| -120 | 1000 1000 |
| + 45 | 0010 1101 |
| -75 | 1011 0101 |

There is no carry

Also, MSB is '1', represents that result is negative

So, take 2's complement. Then result will be 0100 1011 i.e 75. So result is -75

# Addition BCD Case:1

| Packed BCD | Packed BCD in hex form | Decimal |
|---|---|---|
| 0100 0101 + | 45H + | 45 + |
| 0010 0010 | 22H | 22 |
| 0110 0111 | 67H | 67 |

# Addition BCD Case:2

When lower nibble of the sum is greater than 9

| Packed BCD | Packed BCD in hex form | Decimal |
|---|---|---|
| 0100 0101 + | 45H + | 45 + |
| 0010 0111 | 27H | 27 |
| 0110 1100 | 6CH | 72 |

**Correction**

| | |
|---|---|
| 0110 1100 | + |
| 0000 0110 | |
| **0111 0010** | |

When upper nibble of the sum is greater than 9

| | | | | |
|---|---|---|---|---|
| 0111 0110 | + | 76H | + | |
| 0110 0010 | | 62H | | |
| 1101 1000 | + | D8H | + | Now adding 6 to the upper nibble, |
| 0110 0000 | | 60H | | |
| **1 0011 1000** | | **138H** | | |

# Addition BCD Case:2

When both lower and upper nibbles of the sum are greater than 9

| | | | |
|---|---|---|---|
| 1000 1001 | + | 89H | + |
| 0111 0010 | | 72H | |
| 1111 1011 | | FBH | add 06 to both nibbles |
| 0110 0110 | | 66H | |
| 1 0110 0001 | | 1 61H | |

# Subtraction

- Binary
- Hexadecimal
- BCD

# Subtraction (2 of 3)

- Binary, two 1-bit values

| A | B | A – B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 (borrow 1) |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Subtraction

```
  10101010        10101100
  10100010         1010100
  --------        --------
  00001000         1011000
```

# Subtraction

| | |
|---|---|
| 1110 0110 | 1100 1001 |
| 0011 1000 | 1011 0110 |

# Subtraction

| | |
|---|---|
| 1110 0110 | 1100 1001 |
| 0011 1000 | 1011 0110 |
| 1010 1110 | 0001 0011 |

# Hexadecimal subtraction

| | | |
|---|---|---|
| E6H | 898 | ABC |
| 38H | ABA | 12D |
| ----- | --------- | --------- |
| AEH | -222 | 98F |

# Hexadecimal subtraction

| | |
|---|---|
| ABCH | FAC |
| 4FDH | DEA |
| --------- | --------- |

# Hexadecimal subtraction

| | |
|---|---|
| ABCH | FAC |
| 4FDH | DEA |
| --------- | --------- |
| 5BF | 1C2 |

# Subtraction

- Binary, Decimal and Hexadecimal

| | | |
|---|---|---|
| 230 – | 1110 0110 – | E6H – |
| 56 | 0011 1000 | 38H |
| 174 | 1010 1110 | AEH |

# Subtraction

- Packed BCD

| Decimal | | Packed BCD | | | |
|---|---|---|---|---|---|
| 230 | – | 0010 | 0011 | 0000 | – |
| 56 | | 0000 | 0101 | 0110 | |
| 174 | | 0001 | 0111 | 0100 | |

First Step

| | | | |
|---|---|---|---|
| 0010 | 0010 | 1010 | – |
| | | 0110 | |
| | | **0100** | |

Second Step

| | | |
|---|---|---|
| 0001 | 1100 | 1010 |
| | 0101 | 0110 |
| | **0111** | **0100** |

Third Step

| | | | |
|---|---|---|---|
| 0001 | 1101 | 1010 | – |
| 0000 | 0101 | 0110 | |
| **0001** | **0111** | **0100** | |

# BCD subtraction in 9’s compliment method.

- Here the method is very simple. At first the decimal equivalent of the given Binary Coded Decimal (BCD) codes are found out.
- Then the 9’s compliment of the subtrahend is done and then that result is added to the number from which the subtraction is to be done.
- If there is any carry bit then the carry bit may be added to the result of the subtraction.

# Let (0101 0001) – (0010 0001)

(0101 0001)-(0010 0001)

51   21 [Decimal Equivalent]

1's complement of 21 = 99

$$\begin{array}{r} 99 \\ -\ 21 \\ \hline 78 \end{array}$$

Add 78 with 51

$$\begin{array}{r} 78 \\ +\ 51 \\ \hline 129 \\ +\ 1 \\ \hline 30 \end{array}$$

0011 0000

∴ (0101 0001)-(0010 0001) = 0011 0000

# More examples

**Regular Subtraction** | **9's Complement Subtraction**

(a)

$$\begin{array}{r} 8 \\ -\ 2 \\ \hline 6 \end{array}$$

$$\begin{array}{rrl} & 8 & \\ + & 7 & \text{9's complement of 2} \\ \hline 1 & 5 & \\ + & 1 & \text{Add carry to result} \\ \hline & 6 & \end{array}$$

(b)

$$\begin{array}{r} 28 \\ -\ 13 \\ \hline 15 \end{array}$$

$$\begin{array}{rrl} & 28 & \\ + & 86 & \text{9' complement of 13} \\ \hline 1 & 14 & \\ + & 1 & \text{Add carry to the result} \\ \hline & 15 & \end{array}$$

(c)

$$\begin{array}{r} 18 \\ -\ 24 \\ \hline -6 \end{array}$$

$$\begin{array}{rl} 18 & \\ +\ 75 & \text{9' complement of 24} \\ \hline 93 & \text{9's complement of result} \\ \downarrow & \text{(No carry indicates that the} \\ & \text{answer is negative and in} \\ -\ 06 & \text{complement form)} \end{array}$$

# ASCII

- American Standard Code for Information Interchange.
- It is a 7-bit code/ 8-bit code, which is written as a byte.
- This is the code used when entering data through keyboard and displaying text on the video display.

# ASCII

- It has representation for numbers, lower case and upper case English alphabets, special characters (like #, ^ . &) and control characters.

- When we type a character on the keyboard, it is ASCII value of the key that is read in. The computer must convert it from this form to binary form, for processing.

## ASCII control characters

| | | |
|---|---|---|
| 00 | NULL | (Null character) |
| 01 | SOH | (Start of Header) |
| 02 | STX | (Start of Text) |
| 03 | ETX | (End of Text) |
| 04 | EOT | (End of Trans.) |
| 05 | ENQ | (Enquiry) |
| 06 | ACK | (Acknowledgement) |
| 07 | BEL | (Bell) |
| 08 | BS | (Backspace) |
| 09 | HT | (Horizontal Tab) |
| 10 | LF | (Line feed) |
| 11 | VT | (Vertical Tab) |
| 12 | FF | (Form feed) |
| 13 | CR | (Carriage return) |
| 14 | SO | (Shift Out) |
| 15 | SI | (Shift In) |
| 16 | DLE | (Data link escape) |
| 17 | DC1 | (Device control 1) |
| 18 | DC2 | (Device control 2) |
| 19 | DC3 | (Device control 3) |
| 20 | DC4 | (Device control 4) |
| 21 | NAK | (Negative acknowl.) |
| 22 | SYN | (Synchronous idle) |
| 23 | ETB | (End of trans. block) |
| 24 | CAN | (Cancel) |
| 25 | EM | (End of medium) |
| 26 | SUB | (Substitute) |
| 27 | ESC | (Escape) |
| 28 | FS | (File separator) |
| 29 | GS | (Group separator) |
| 30 | RS | (Record separator) |
| 31 | US | (Unit separator) |
| 127 | DEL | (Delete) |

## ASCII printable characters

| | | | | | |
|---|---|---|---|---|---|
| 32 | **space** | 64 | **@** | 96 | ` |
| 33 | **!** | 65 | **A** | 97 | **a** |
| 34 | **"** | 66 | **B** | 98 | **b** |
| 35 | **#** | 67 | **C** | 99 | **c** |
| 36 | **$** | 68 | **D** | 100 | **d** |
| 37 | **%** | 69 | **E** | 101 | **e** |
| 38 | **&** | 70 | **F** | 102 | **f** |
| 39 | **'** | 71 | **G** | 103 | **g** |
| 40 | **(** | 72 | **H** | 104 | **h** |
| 41 | **)** | 73 | **I** | 105 | **i** |
| 42 | * | 74 | **J** | 106 | **j** |
| 43 | **+** | 75 | **K** | 107 | **k** |
| 44 | **,** | 76 | **L** | 108 | **l** |
| 45 | **-** | 77 | **M** | 109 | **m** |
| 46 | **.** | 78 | **N** | 110 | **n** |
| 47 | **/** | 79 | **O** | 111 | **o** |
| 48 | **0** | 80 | **P** | 112 | **p** |
| 49 | **1** | 81 | **Q** | 113 | **q** |
| 50 | **2** | 82 | **R** | 114 | **r** |
| 51 | **3** | 83 | **S** | 115 | **s** |
| 52 | **4** | 84 | **T** | 116 | **t** |
| 53 | **5** | 85 | **U** | 117 | **u** |
| 54 | **6** | 86 | **V** | 118 | **v** |
| 55 | **7** | 87 | **W** | 119 | **w** |
| 56 | **8** | 88 | **X** | 120 | **x** |
| 57 | **9** | 89 | **Y** | 121 | **y** |
| 58 | **:** | 90 | **Z** | 122 | **z** |
| 59 | **;** | 91 | **[** | 123 | **{** |
| 60 | **<** | 92 | **** | 124 | **\|** |
| 61 | **=** | 93 | **]** | 125 | **}** |
| 62 | **>** | 94 | **^** | 126 | **~** |
| 63 | **?** | 95 | **_** | | |

## Extended ASCII characters

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 128 | Ç | 160 | á | 192 | └ | 224 | Ó |
| 129 | ü | 161 | í | 193 | ┴ | 225 | ß |
| 130 | é | 162 | ó | 194 | ┬ | 226 | Ô |
| 131 | â | 163 | ú | 195 | ├ | 227 | Ò |
| 132 | ä | 164 | ñ | 196 | ─ | 228 | õ |
| 133 | à | 165 | Ñ | 197 | ┼ | 229 | Õ |
| 134 | å | 166 | ª | 198 | ã | 230 | µ |
| 135 | ç | 167 | º | 199 | Ã | 231 | þ |
| 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Þ |
| 137 | ë | 169 | ® | 201 | ╔ | 233 | Ú |
| 138 | è | 170 | ¬ | 202 | ╩ | 234 | Û |
| 139 | ï | 171 | ½ | 203 | ╦ | 235 | Ù |
| 140 | î | 172 | ¼ | 204 | ╠ | 236 | ý |
| 141 | ì | 173 | ¡ | 205 | ═ | 237 | Ý |
| 142 | Ä | 174 | « | 206 | ╬ | 238 | ¯ |
| 143 | Å | 175 | » | 207 | ¤ | 239 | ´ |
| 144 | É | 176 | ░ | 208 | ð | 240 | ≡ |
| 145 | æ | 177 | ▒ | 209 | Ð | 241 | ± |
| 146 | Æ | 178 | ▓ | 210 | Ê | 242 | ‗ |
| 147 | ô | 179 | │ | 211 | Ë | 243 | ¾ |
| 148 | ö | 180 | ┤ | 212 | È | 244 | ¶ |
| 149 | ò | 181 | Á | 213 | ı | 245 | § |
| 150 | û | 182 | Â | 214 | Í | 246 | ÷ |
| 151 | ù | 183 | À | 215 | Î | 247 | ¸ |
| 152 | ÿ | 184 | © | 216 | Ï | 248 | ° |
| 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | ¨ |
| 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 155 | ø | 187 | ╗ | 219 | █ | 251 | ¹ |
| 156 | £ | 188 | ╝ | 220 | ▄ | 252 | ³ |
| 157 | Ø | 189 | ¢ | 221 | ¦ | 253 | ² |
| 158 | × | 190 | ¥ | 222 | Ì | 254 | ■ |
| 159 | ƒ | 191 | ┐ | 223 | ▀ | 255 | nbsp |