Fundamentals of Programming CS-114

Lecture 2

Programming Languages

- Programming Language
  - A set of rules, symbols, and special words used to construct a computer program.
- **Machine Language**
  - The binary representation of the instructions that a computer's hardware can perform.
- **Assembly Language**
  - A low-level programming language in which a mnemonic is used to represent each of the machine language instructions for a specific computer.
- **High-Level Language**
  - A computer language that is more understandable and closer to standard notations than assembly language. It is more close to plain English. C/C++ are high-level languages.

Programming language rules

- Rules of **Syntax** which specify how valid instructions are written in the language.
  - They deal with the structure of an instruction
- Rules of **Semantics** which determine the meaning of the instructions (what the computer will do in response to the given instructions).
  - They deal with the content of an instruction

Bugs and Debugging

- Programming errors are called **bugs**
- The process of tracking bugs and correcting them is called **debugging**.
- Three kinds of errors can occur in a program:
  - syntax errors
  - semantic errors
  - runtime errors
- It is useful to distinguish between them in order to track them down more quickly

Syntax errors

- Most languages including c++ can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message.
  - **Syntax** refers to the structure of a program and the rules about that structure.
  - For example, in English, a sentence must begin with a capital letter and end with a period.

    this sentence contains a **syntax error**.

    So does this one

Semantic Errors

- Semantic error is an error in the content of a code.
- If there is a semantic error in your program, it will run successfully, i.e. the computer will not generate any error messages, but it will not do the right thing.
- The problem is that the program you wrote is not the program you wanted to write. The meaning of the program (its semantics) is wrong.
- Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

Runtime Errors

- The third type of error is a runtime error, so called because the error does not appear until you run the program.
- These errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened.
- Runtime errors are rare in the simple programs so it might be a while before you encounter one.

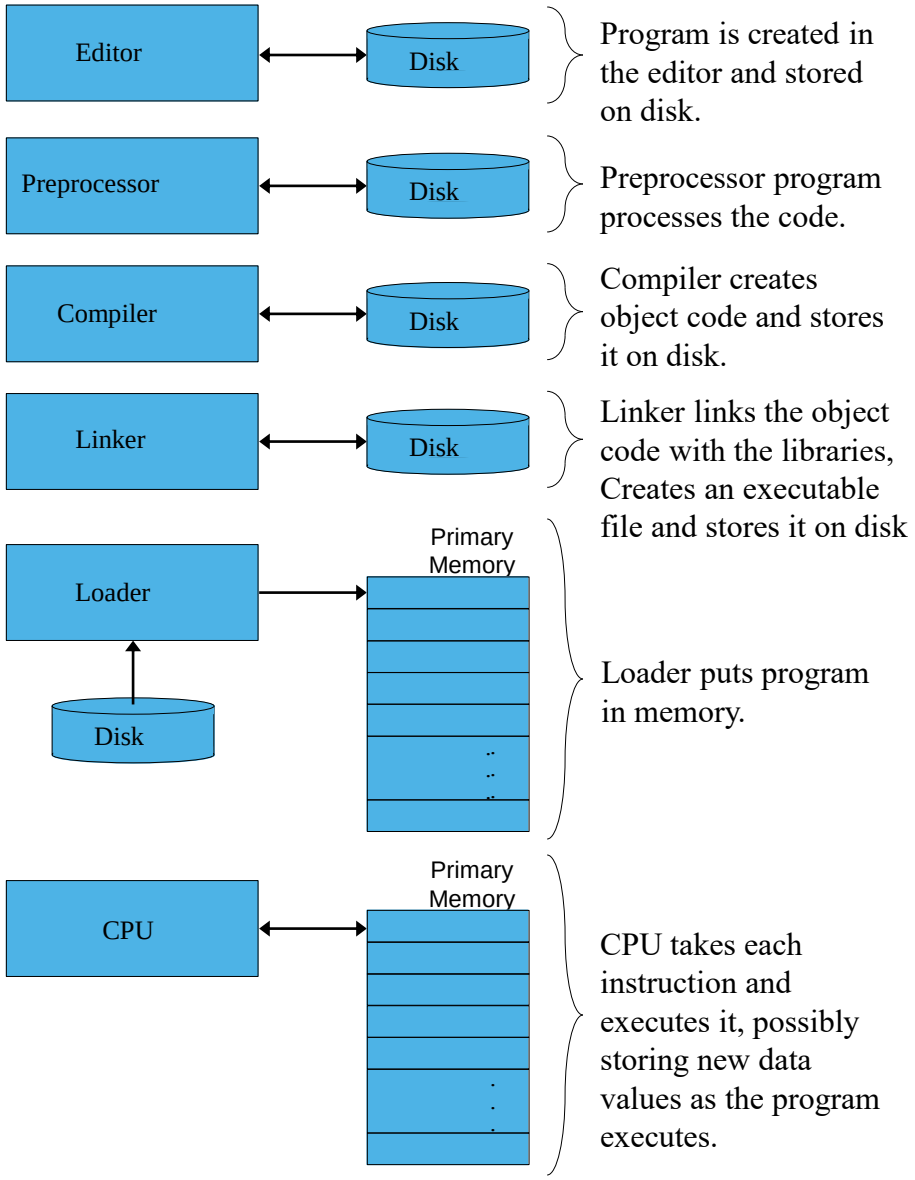
Types of Program

- Source Program
  - A program written in a human readable version, which you will write.
- Object Program
  - The machine language version of a source program in 0s and 1s.
- EXE Program
  - It is an executable program

Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



- Functions
  - Function name
  - Braces and the function body
  - Main function
- Program Statements
- Output using cout

- Directives
  - Preprocessor Directives #
  - Header Files
  - The ‘using’ Directive
  - comments

A basic program

```
// ----------------------------------------------------------
/ *hello.cpp is a demonstration program
Welcome to C++*/

#include <iostream>
using namespace std;


void main ( )
   {
        cout << "Hello C++ World ! \n";
   }
```

Comments

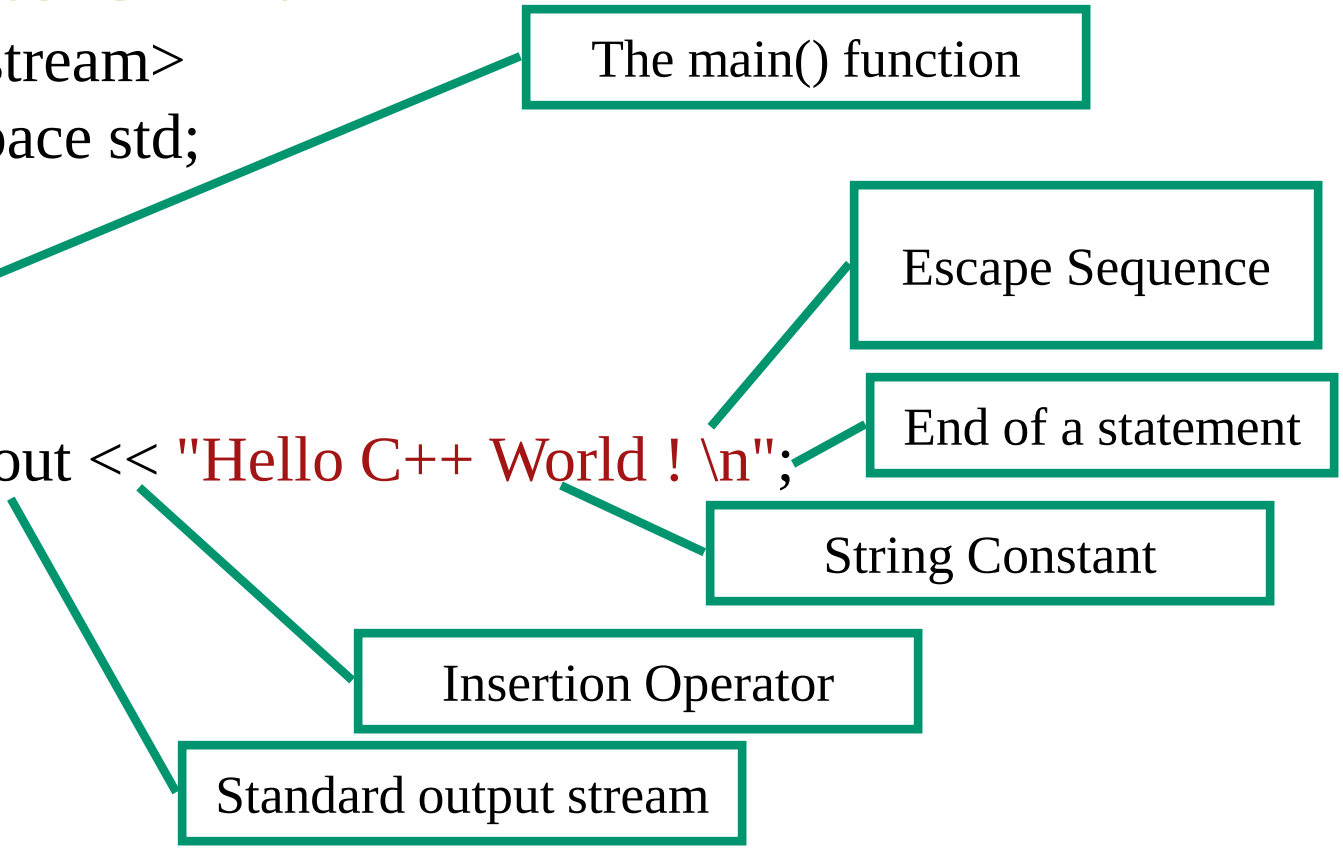
Preprocessor Directives/ Header File

The using Directive

A basic program

```
// ------------------------------------------------------
   ---
/ *hello.cpp is a demonstration program
Welcome to C++*/
#include <iostream>
using namespace std;

void main ( )
    {
            cout << "Hello C++ World ! \n";
    }
```



A basic program

- Programs typically contain the following elements:
  - Descriptive comments (double forward slash // or /* … */)
  - Include files
  - Functions including exactly one main function

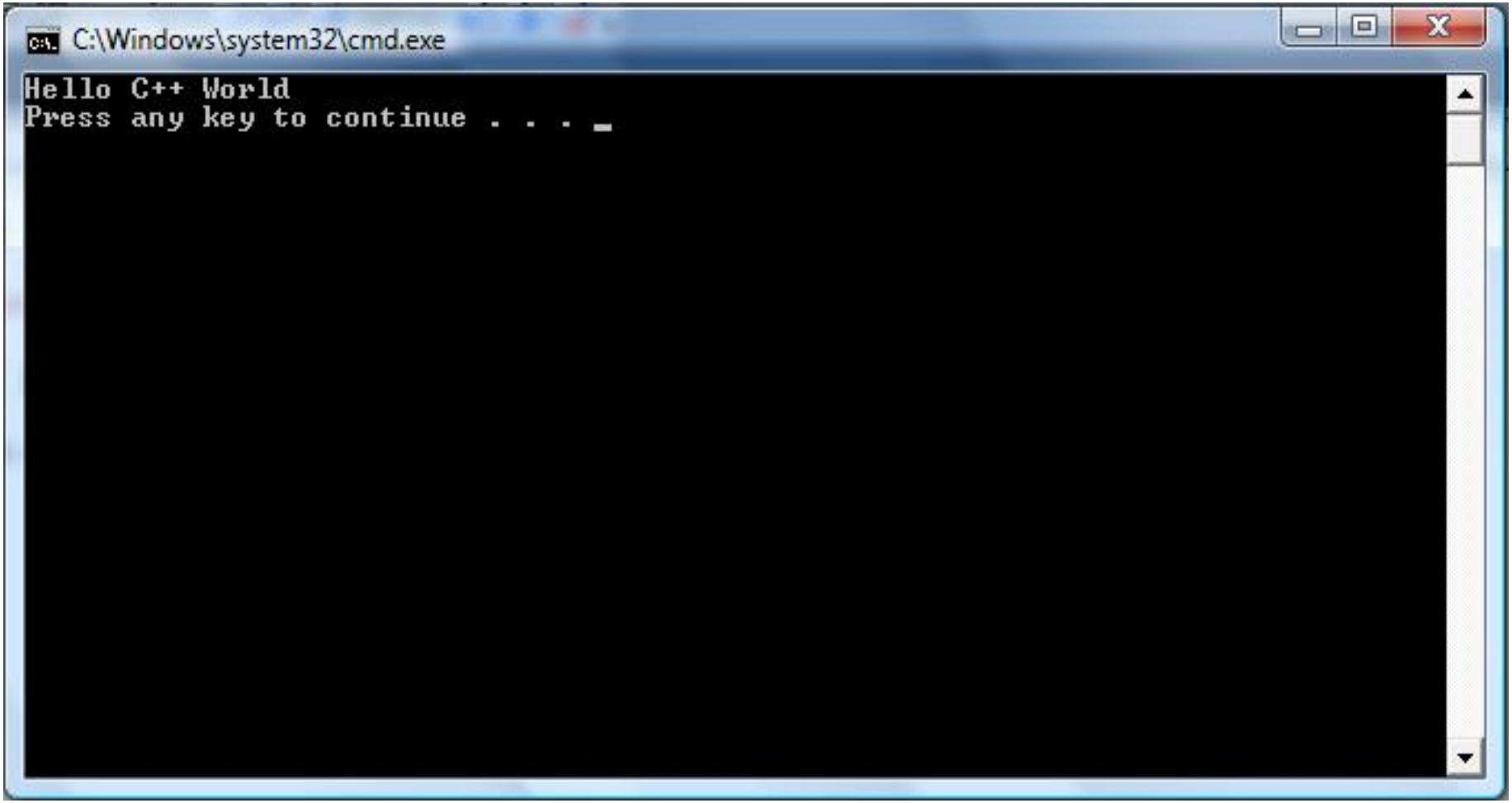
- The main function
  - Controls the execution of the program

Output Statements

cout << “Hello C++ World”;

- **cout** is an object in c++, predefined to display the standard output stream
- << insertion operator
  - It directs the contents of the variable on its right to the object on its left
- The output of this program
  - Hello C++ World

Output


C:\Windows\system32\cmd.exe
Hello C++ World
Press any key to continue . . .

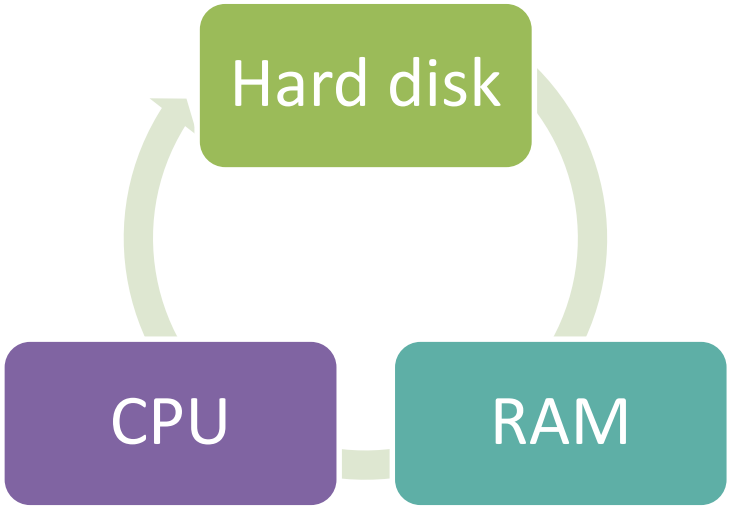
Escape Sequences

Escape Sequence	Character
\a	Bell (beep)
\b	Backspace
\n	Newline
\r	Return
\t	Tab
\\	Backslash
\‘	Single quotation mark
\“	Double quotation marks

What Happens When Your Code is Executed

- The program(your browser or music player)/code(your first C++ program) saved on the hard disk is loaded into the RAM

- From RAM, each instruction(open this website/play this song/add two numbers) is sent to the CPU to be executed



What Happens When Your Code is Executed

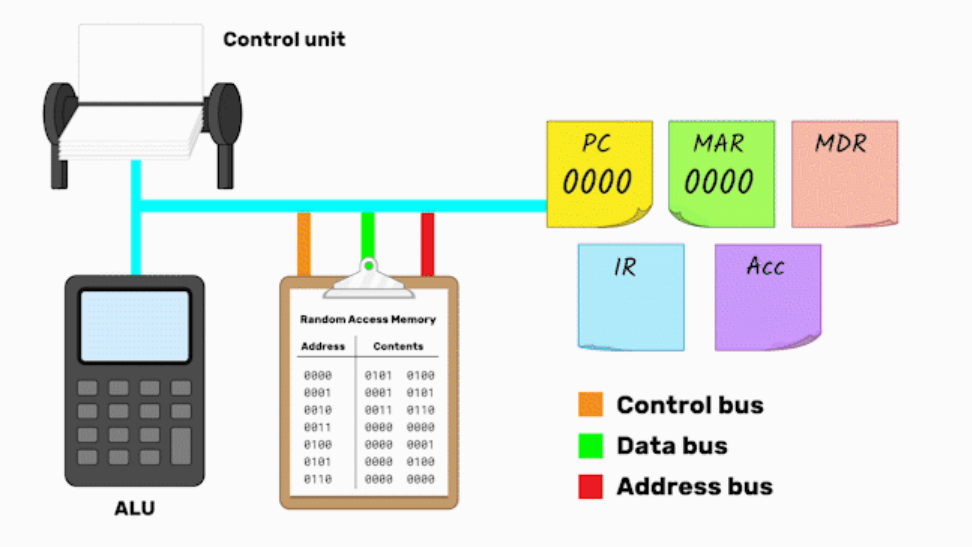
- The CPU by the virtue of its complex design is able to execute the instructions read from the RAM

- The CPU has 3 main components:
  - Arithmetic Processing Unit (ALU)
  - Control Unit
  - Buses (Highways for data)

- The running of a command/instruction can be accomplished in 3 sub-steps:
  - Fetch
  - Decode
  - Execute

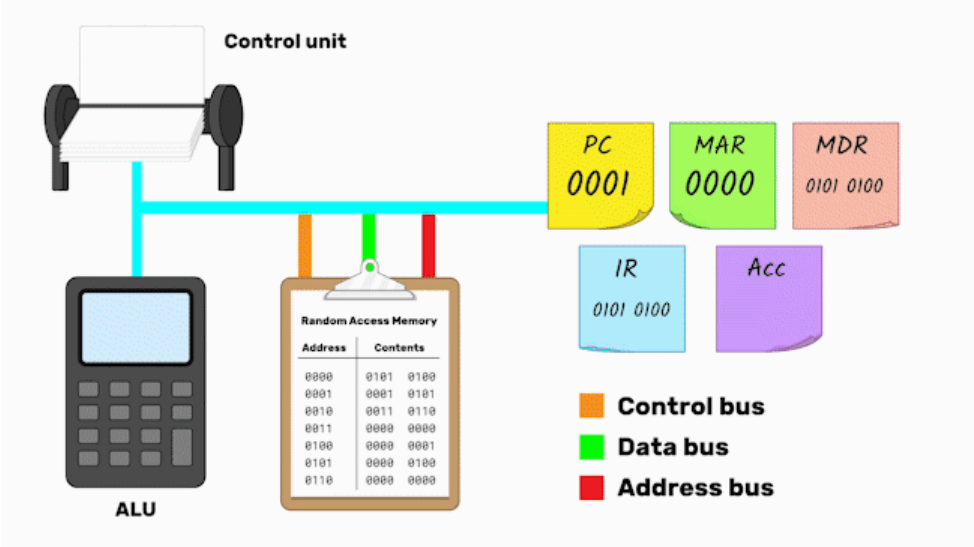
What Happens When Your Code is Executed

- Fetching entails bringing the instruction into the registers for running it
- Registers are exactly what they sound like: temporary storage



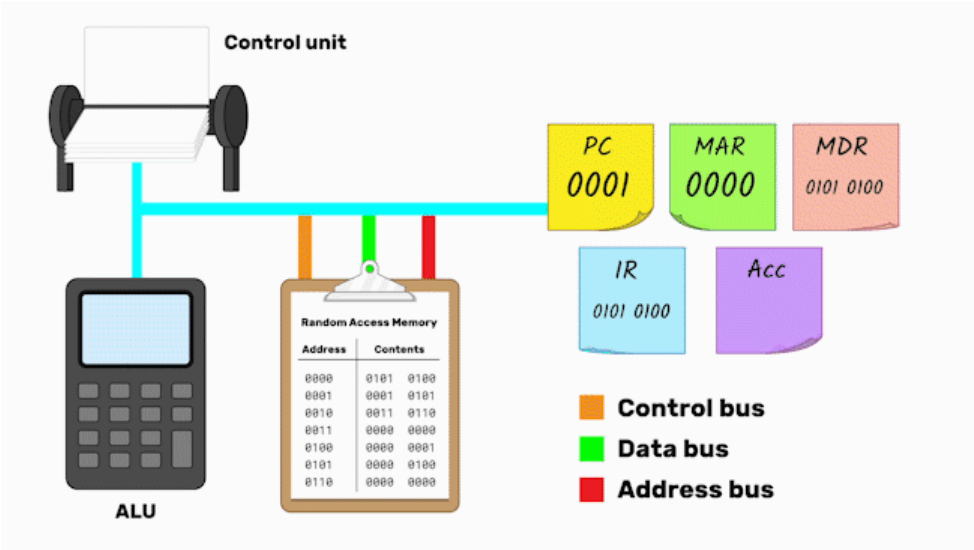
What Happens When Your Code is Executed

- In decoding, the CPU “understands” what it has to do
- Equivalent to decoding of instructions that we do as humans
- Now, the PC knows what to do



What Happens When Your Code is Executed

- Once the CPU knows what to do, it does it
- Executing a series of well-defined instructions results in what we want to do (like adding million of numbers together or finding out today's weather)



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

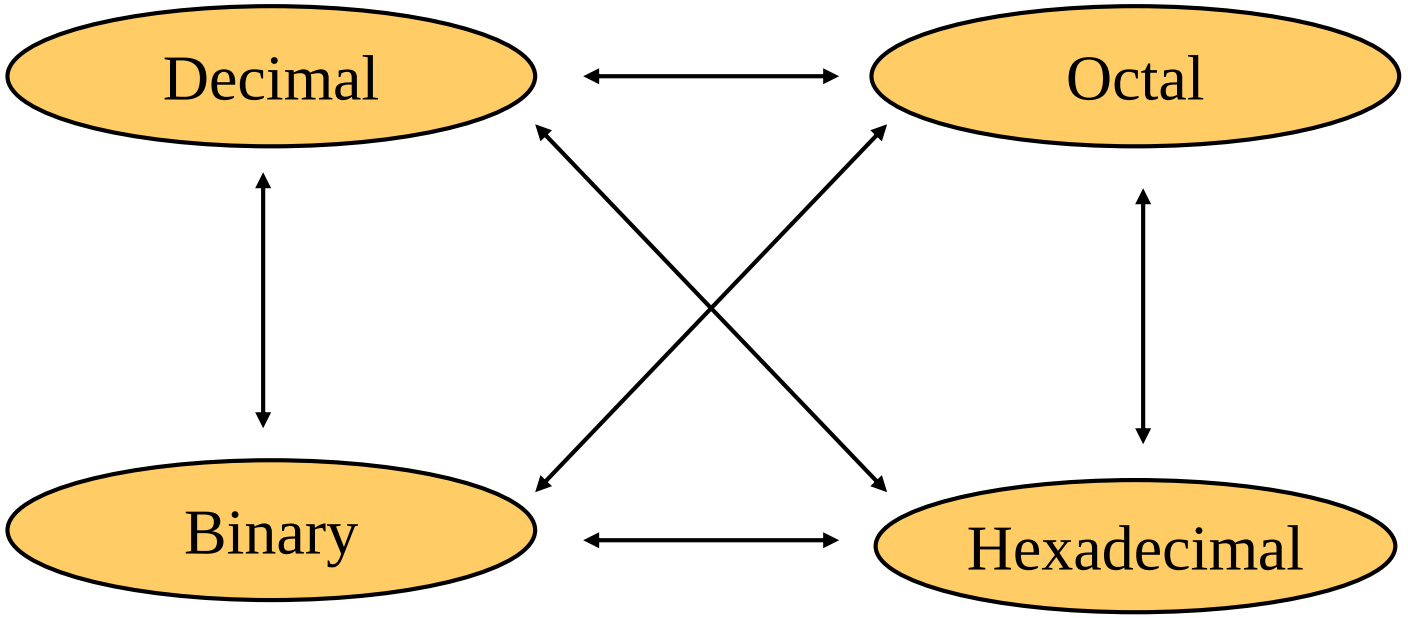
Quantities/Counting (3 of 3)

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

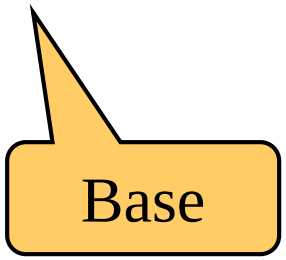
- The possibilities:



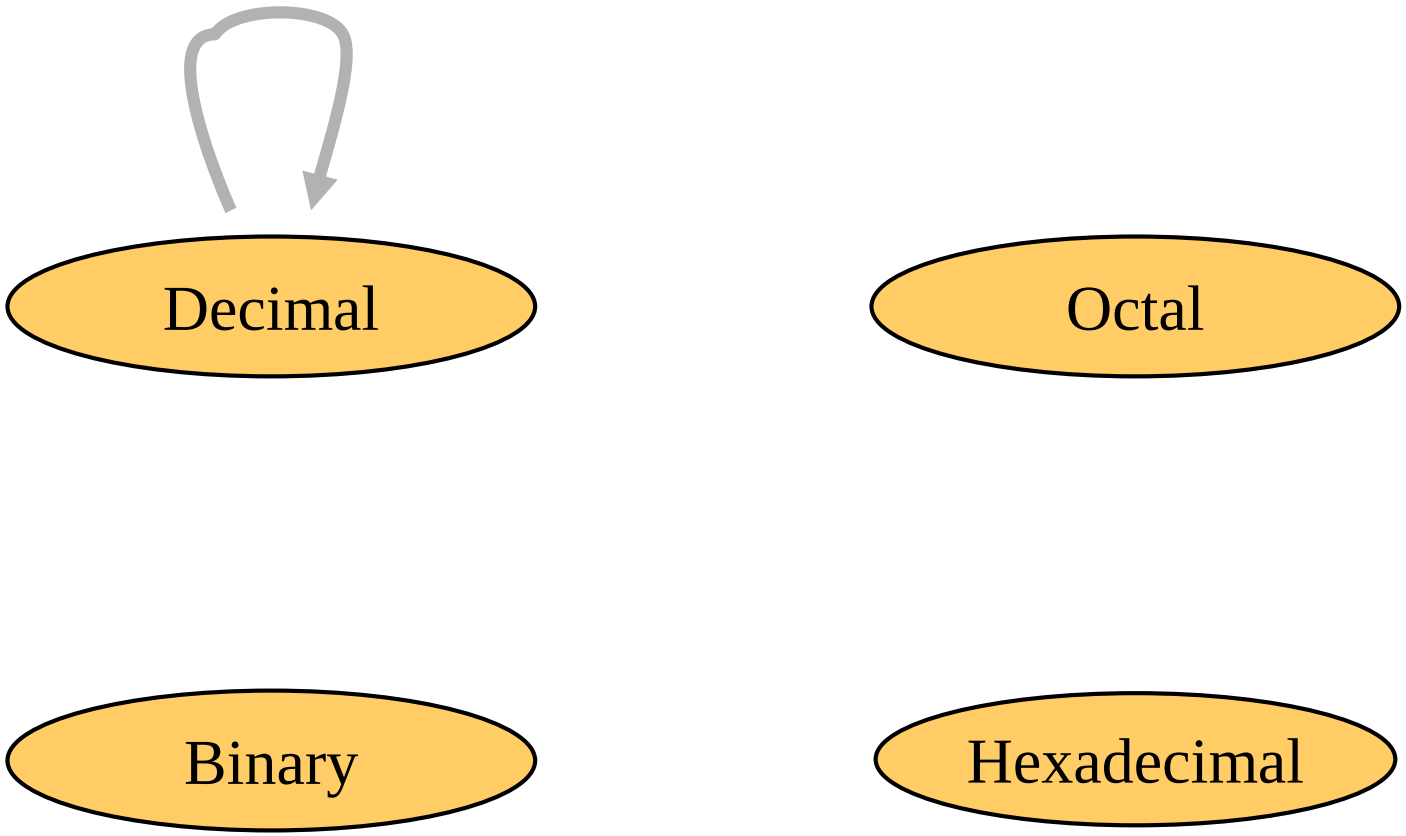
pp. 40-46

Quick Example

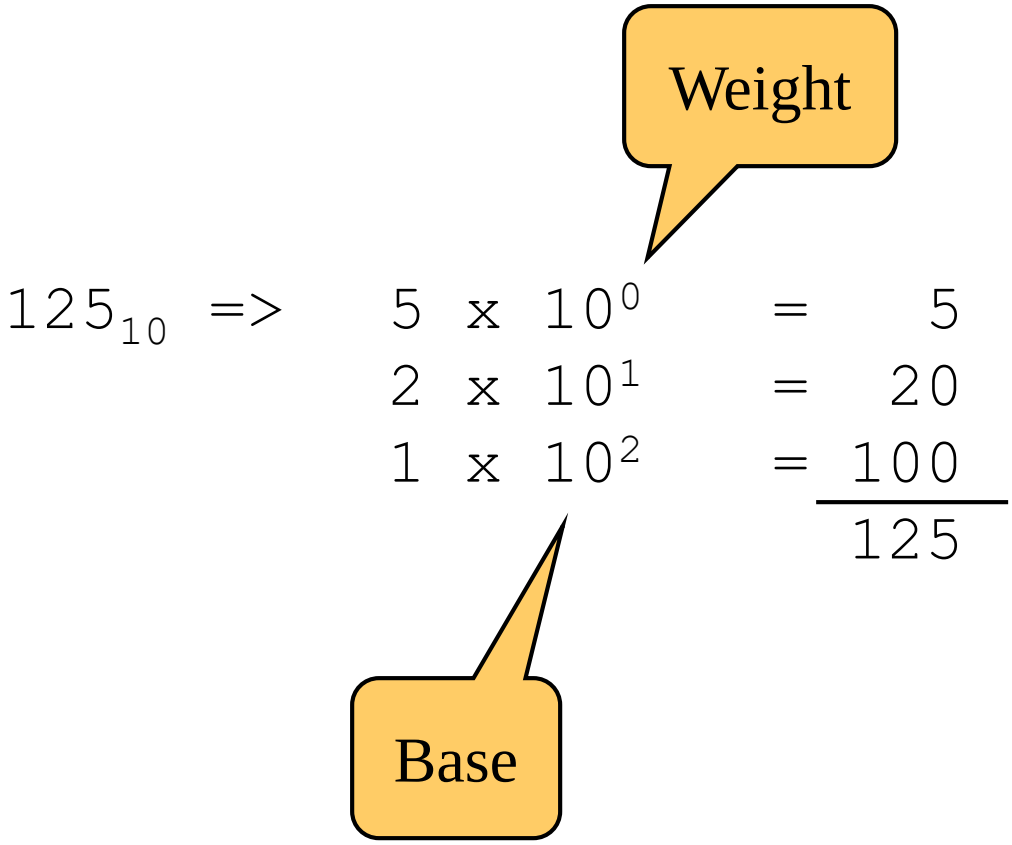
$$25_{10} = 11001_{2} = 31_{8} = 19_{16}$$



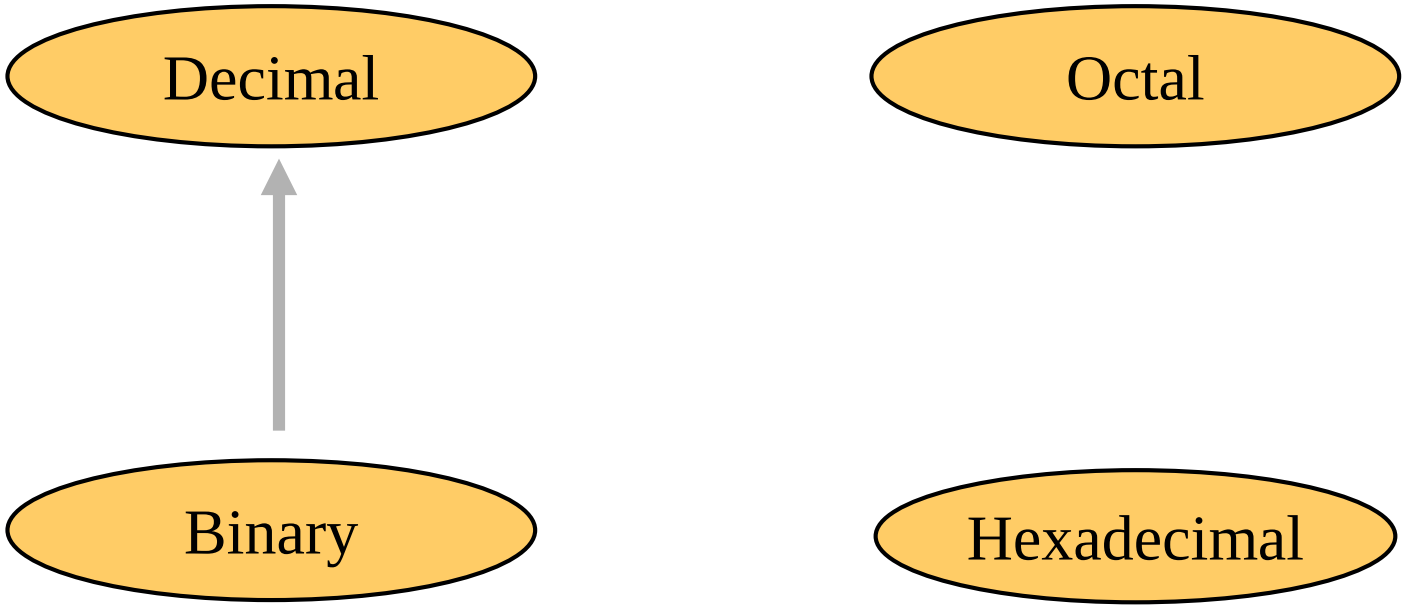
Decimal to Decimal (just for fun)



Next slide…


Weight
125_{10} => 5 x 10^0 = 5
2 x 10^1 = 20
1 x 10^2 = 100
125
Base

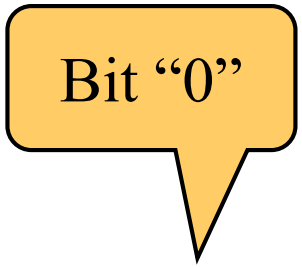
Binary to Decimal



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



101011_2 =>

$1 \times 2^0 = 1$

$1 \times 2^1 = 2$

$0 \times 2^2 = 0$

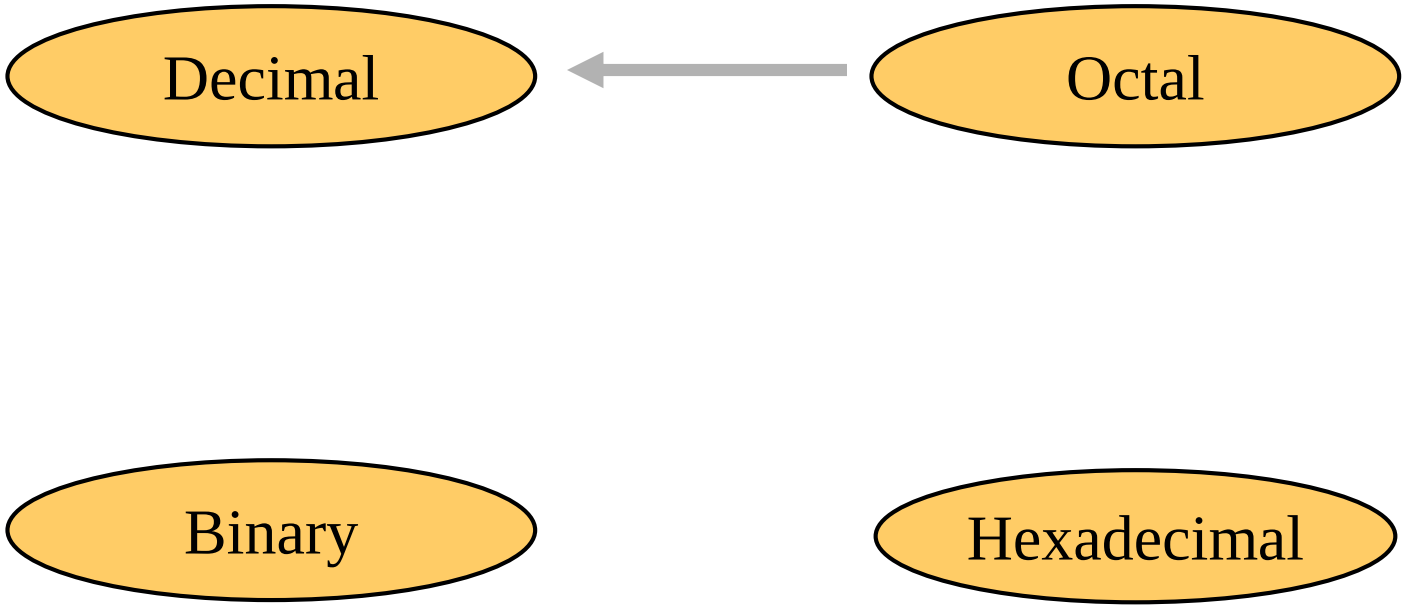
$1 \times 2^3 = 8$

$0 \times 2^4 = 0$

$1 \times 2^5 = 32$

43_{10}

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

724_8 =>

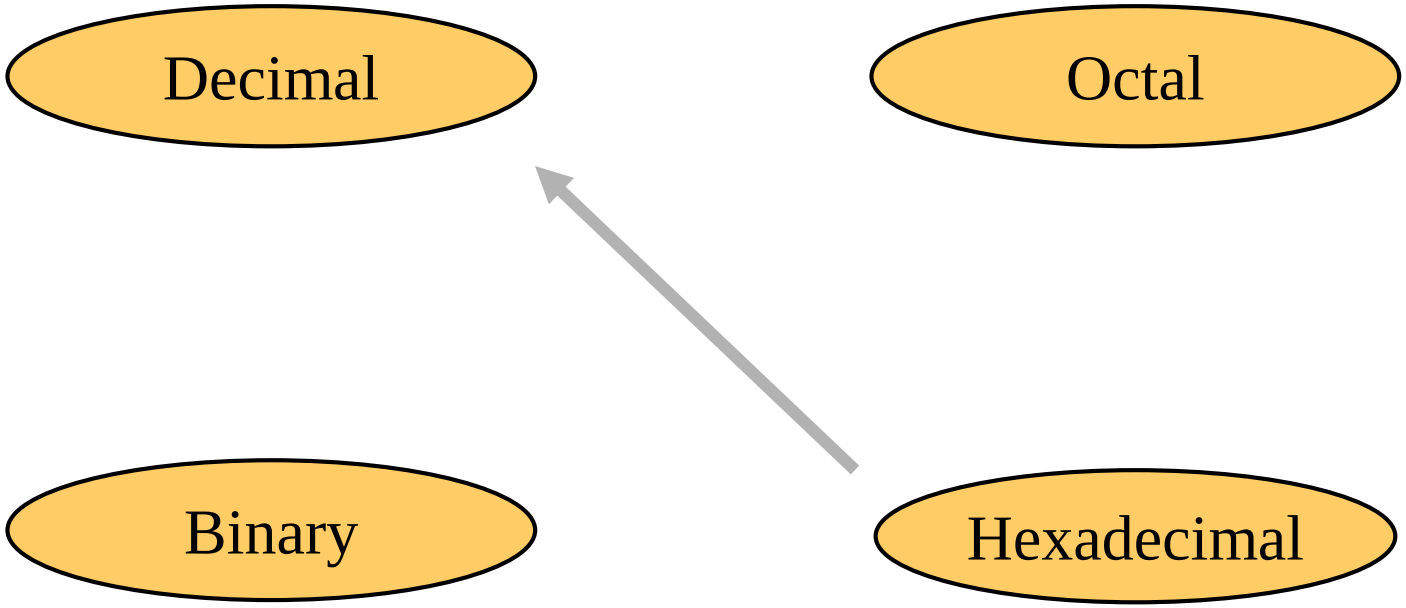
$4 \times 8^0 = 4$

$2 \times 8^1 = 16$

$7 \times 8^2 = 448$

468_{10}

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

ABC_{16} =>

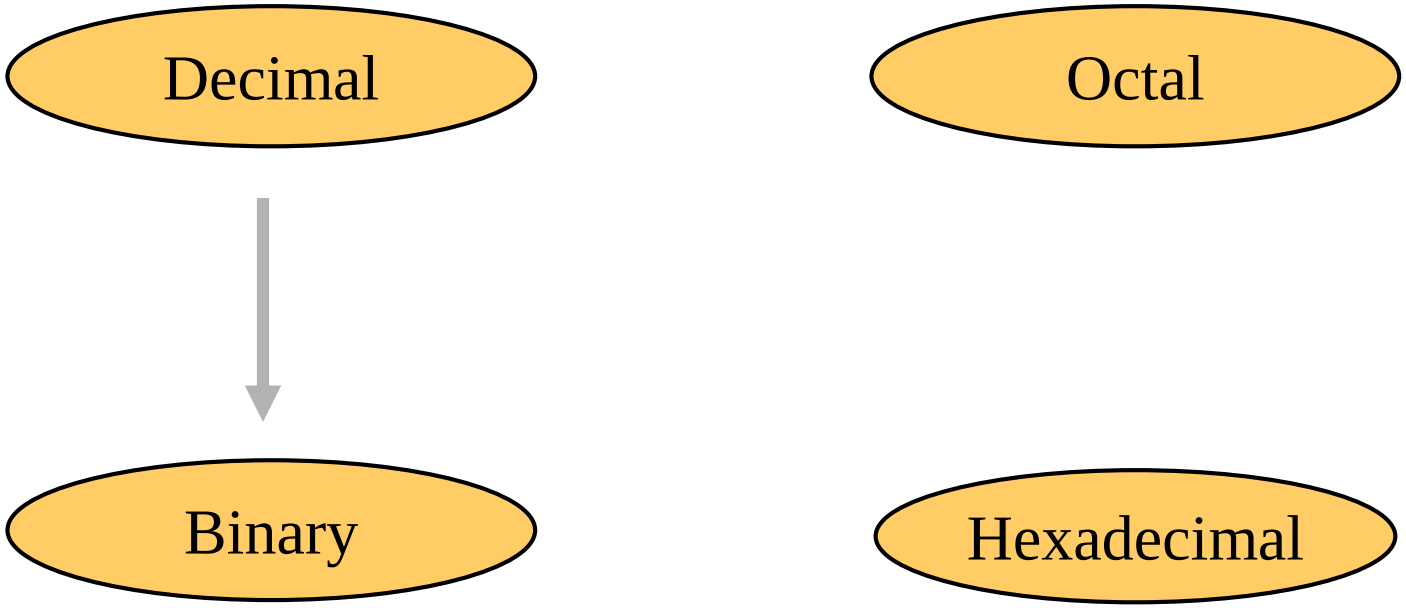
$C \times 16^0 = 12 \times 1 = 12$

$B \times 16^1 = 11 \times 16 = 176$

$A \times 16^2 = 10 \times 256 = 2560$

2748_{10}

Decimal to Binary



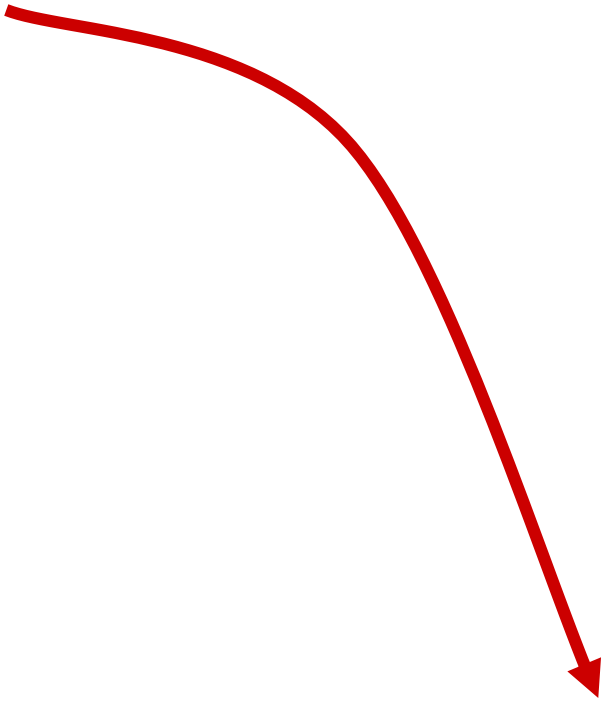
Decimal to Binary

- 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

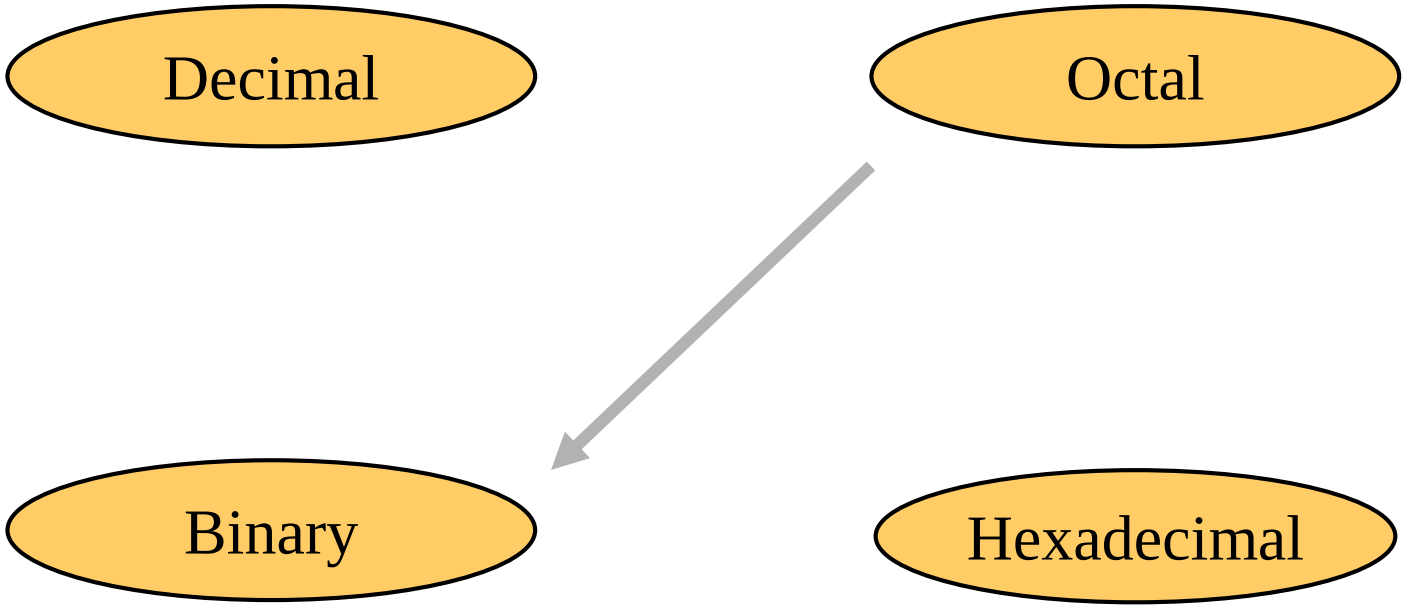
$125_{10} = ?_2$

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



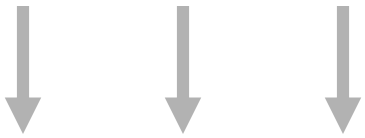
Octal to Binary

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

Example

$705_8 = ?_2$

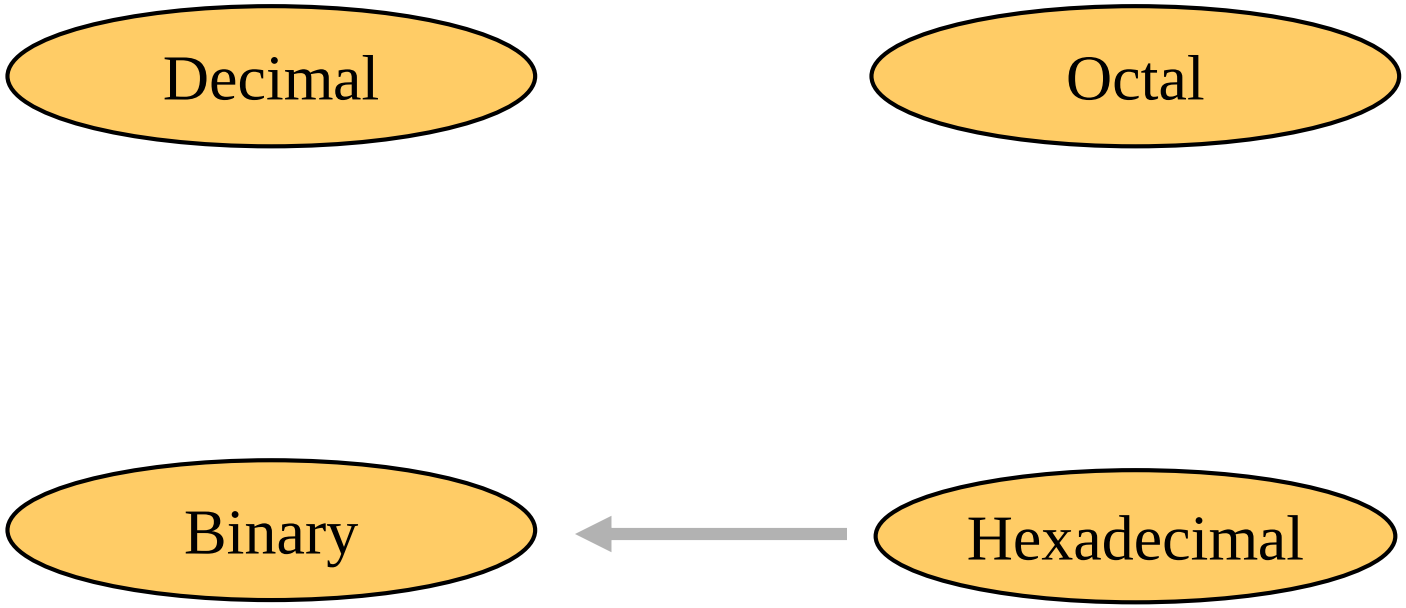
7 0 5



111 000 101

$705_8 = 111000101_2$

Hexadecimal to Binary

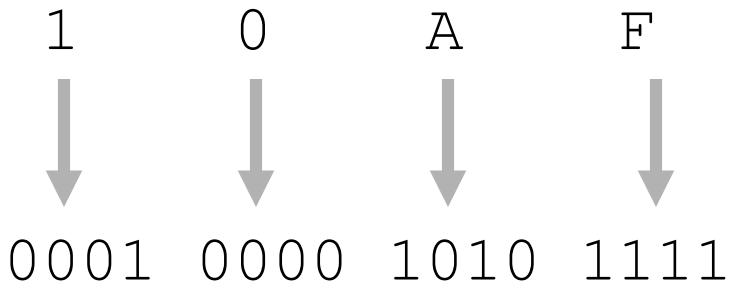


Hexadecimal to Binary

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

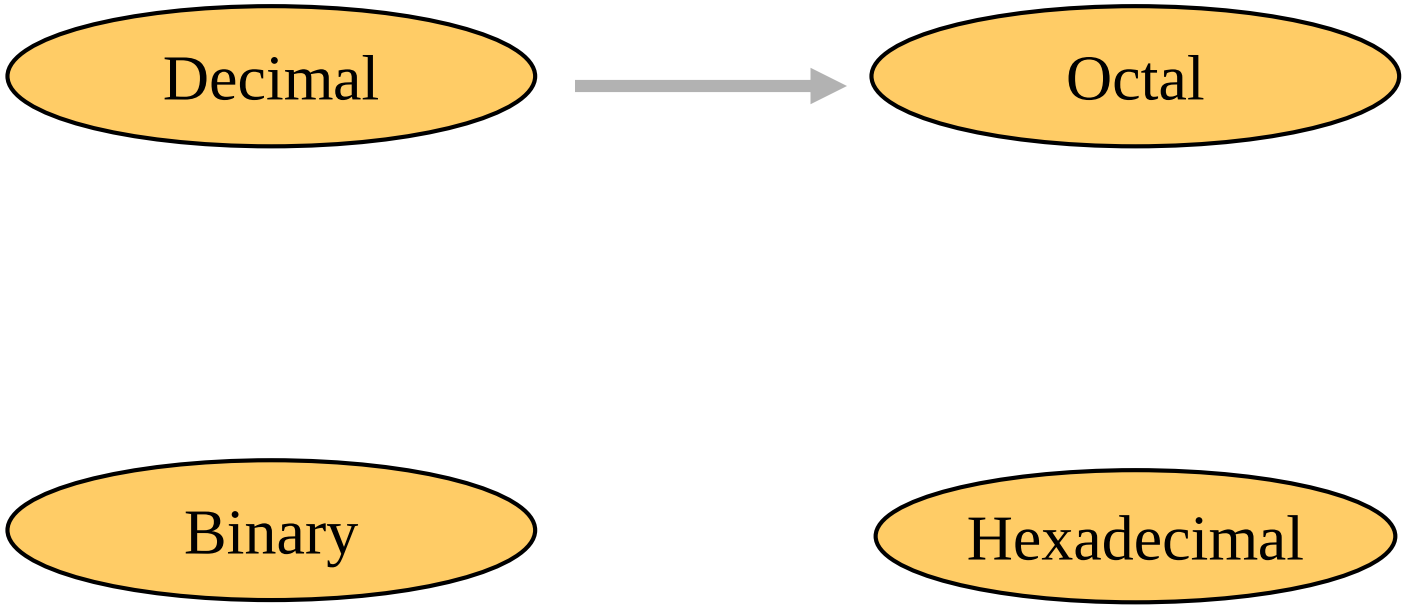
Example

$10AF_{16} = ?_{2}$



$10AF_{16} = 0001000010101111_{2}$

Decimal to Octal



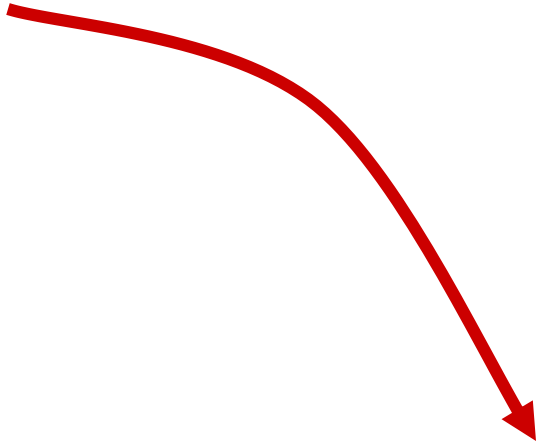
Decimal to Octal

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

Example

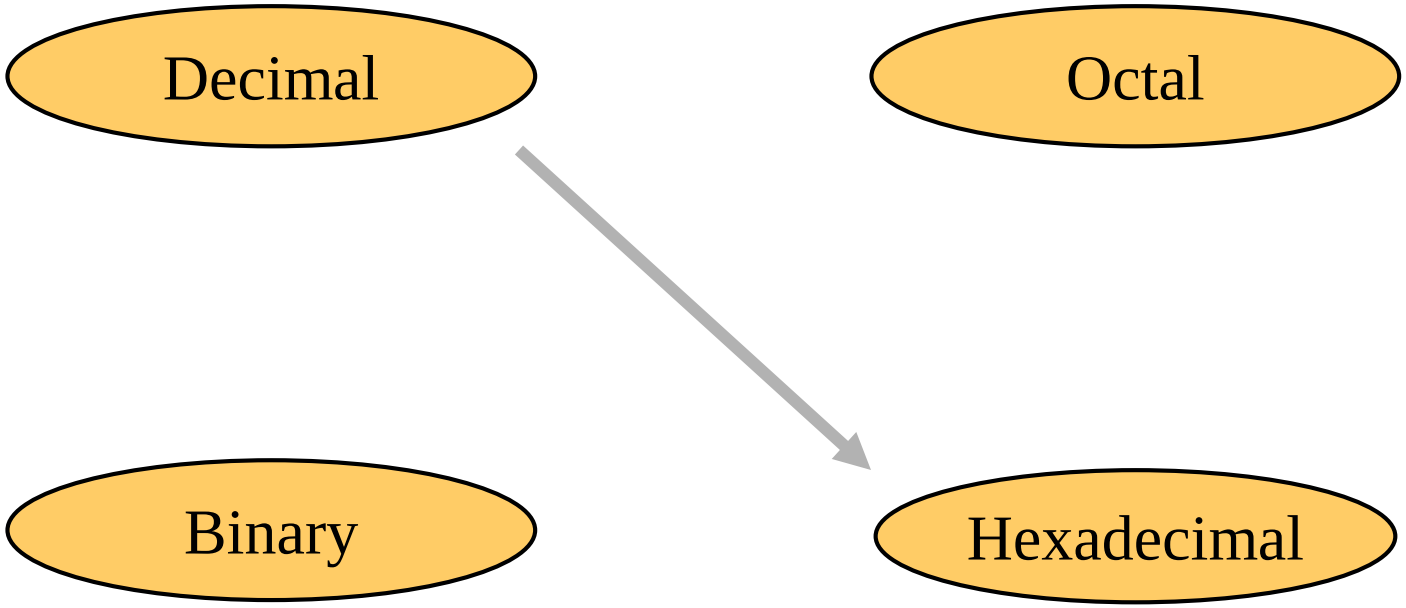
$1234_{10} = ?_{8}$

8	1234	
8	154	2
8	19	2
8	2	3
	0	2



$1234_{10} = 2322_{8}$

Decimal to Hexadecimal



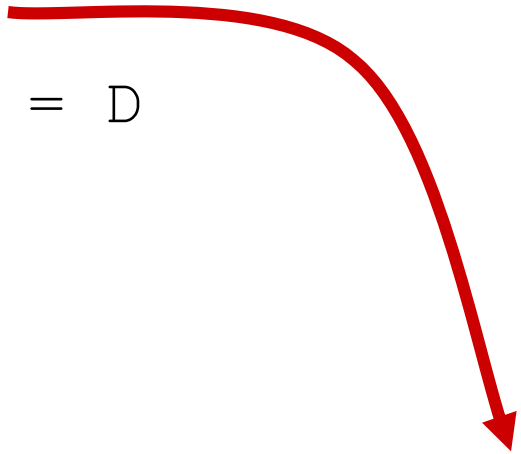
Decimal to Hexadecimal

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

Example

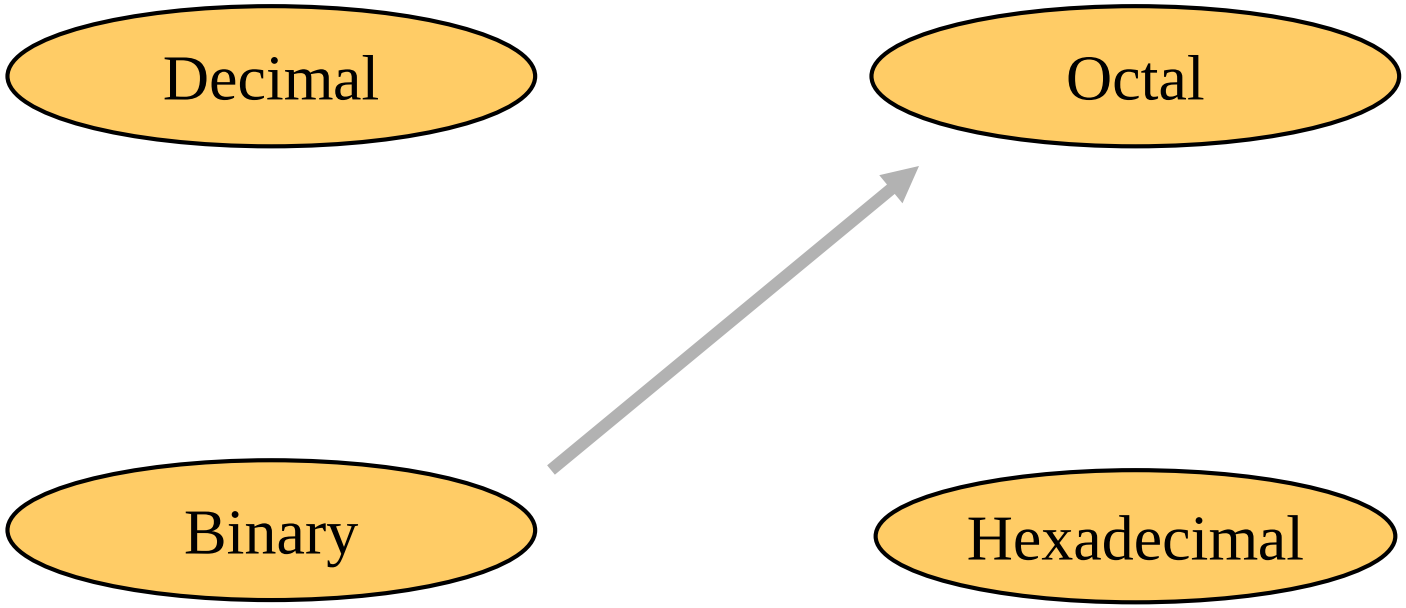
$1234_{10} = ?_{16}$

16	1234	
16	77	2
16	4	13 = D
	0	4



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

Binary to Octal



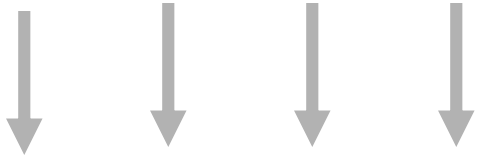
Binary to Octal

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

Example

$1011010111_2 = ?_8$

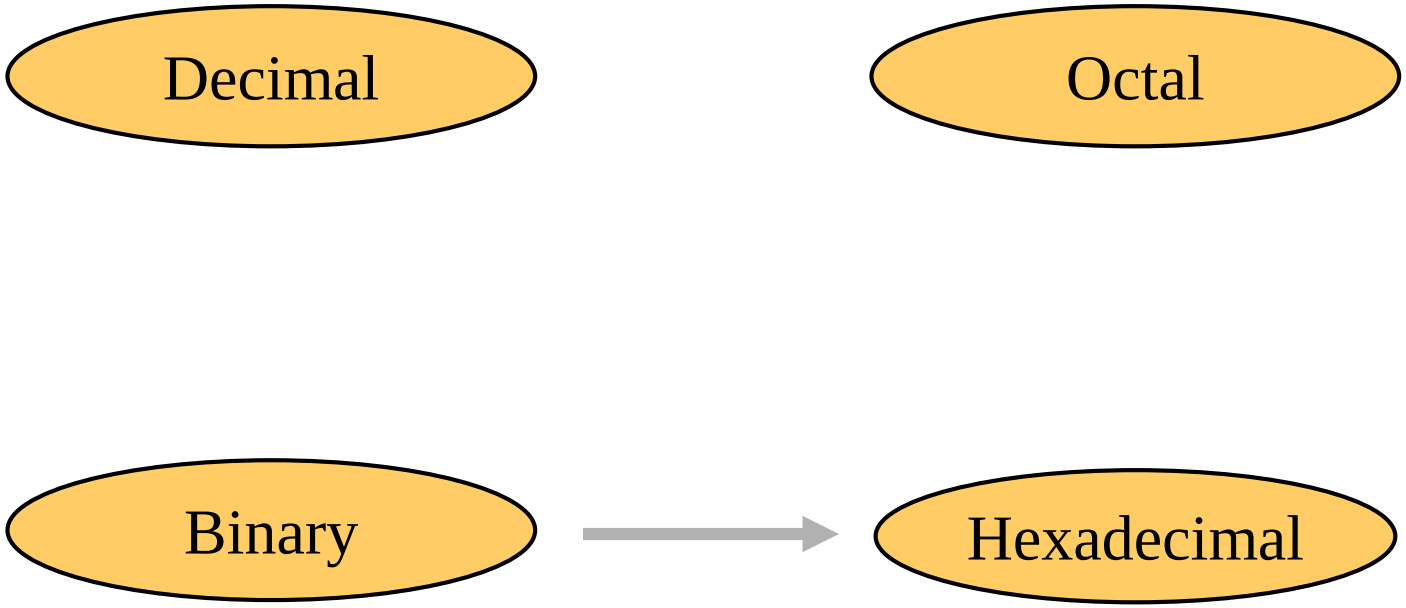
1 011 010 111



1 3 2 7

$1011010111_2 = 1327_8$

Binary to Hexadecimal



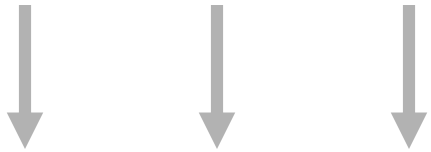
Binary to Hexadecimal

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

Example

$1010111011_2 = ?_{16}$

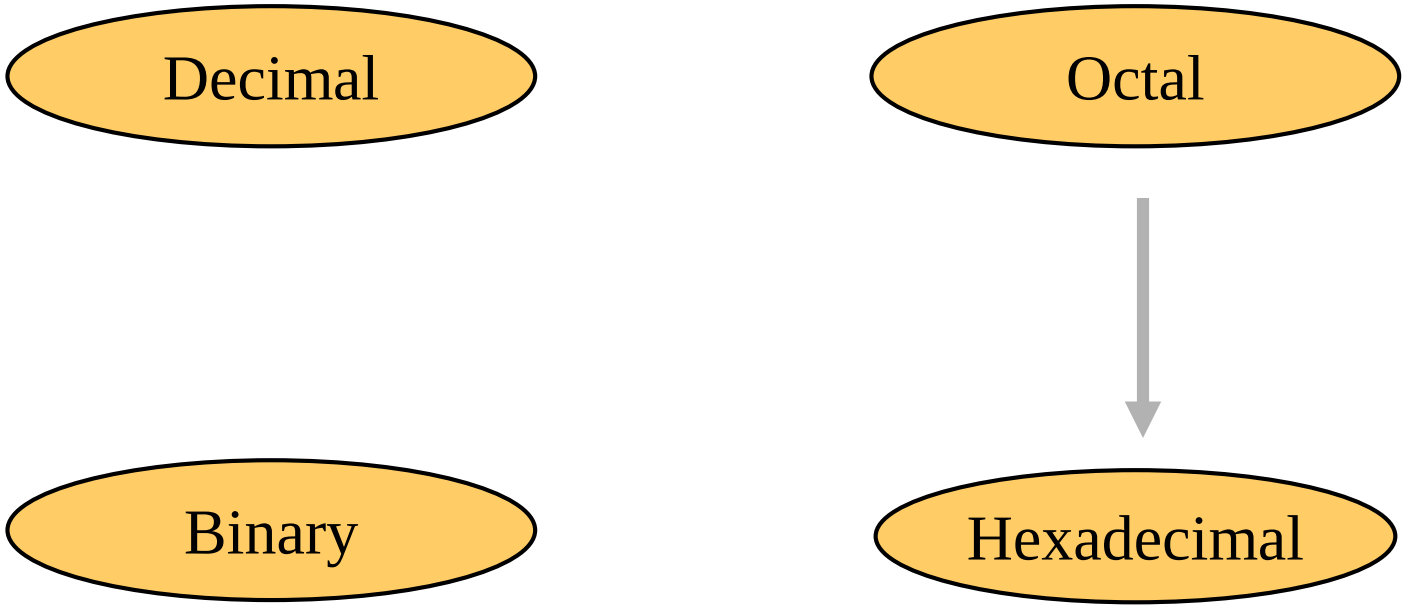
10 1011 1011



2 B B

$1010111011_2 = 2BB_{16}$

Octal to Hexadecimal

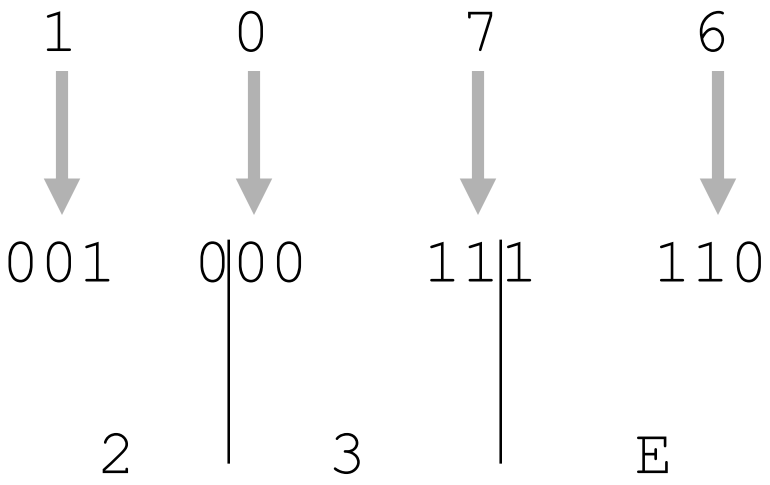


Octal to Hexadecimal

- Technique
  - Use binary as an intermediary

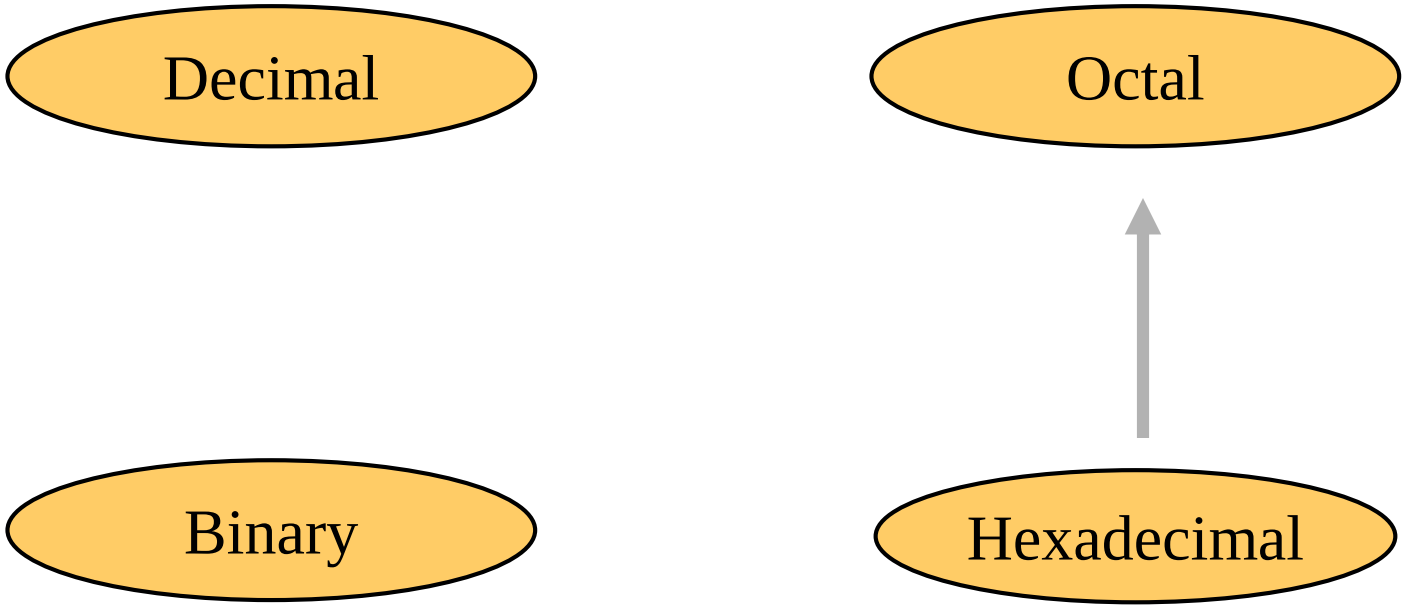
Example

$1076_8 = ?_{16}$



$1076_8 = 23E_{16}$

Hexadecimal to Octal

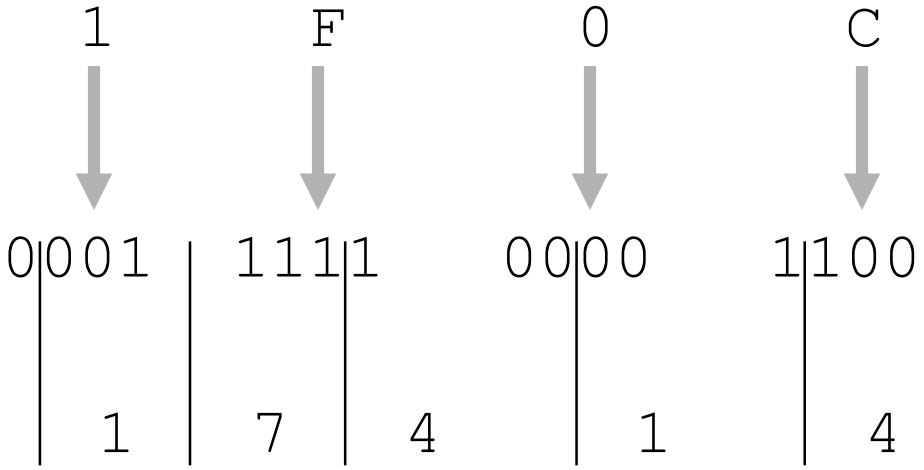


Hexadecimal to Octal

- Technique
  - Use binary as an intermediary

Example

$1F0C_{16} = ?_{8}$



$1F0C_{16} = 17414_{8}$

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

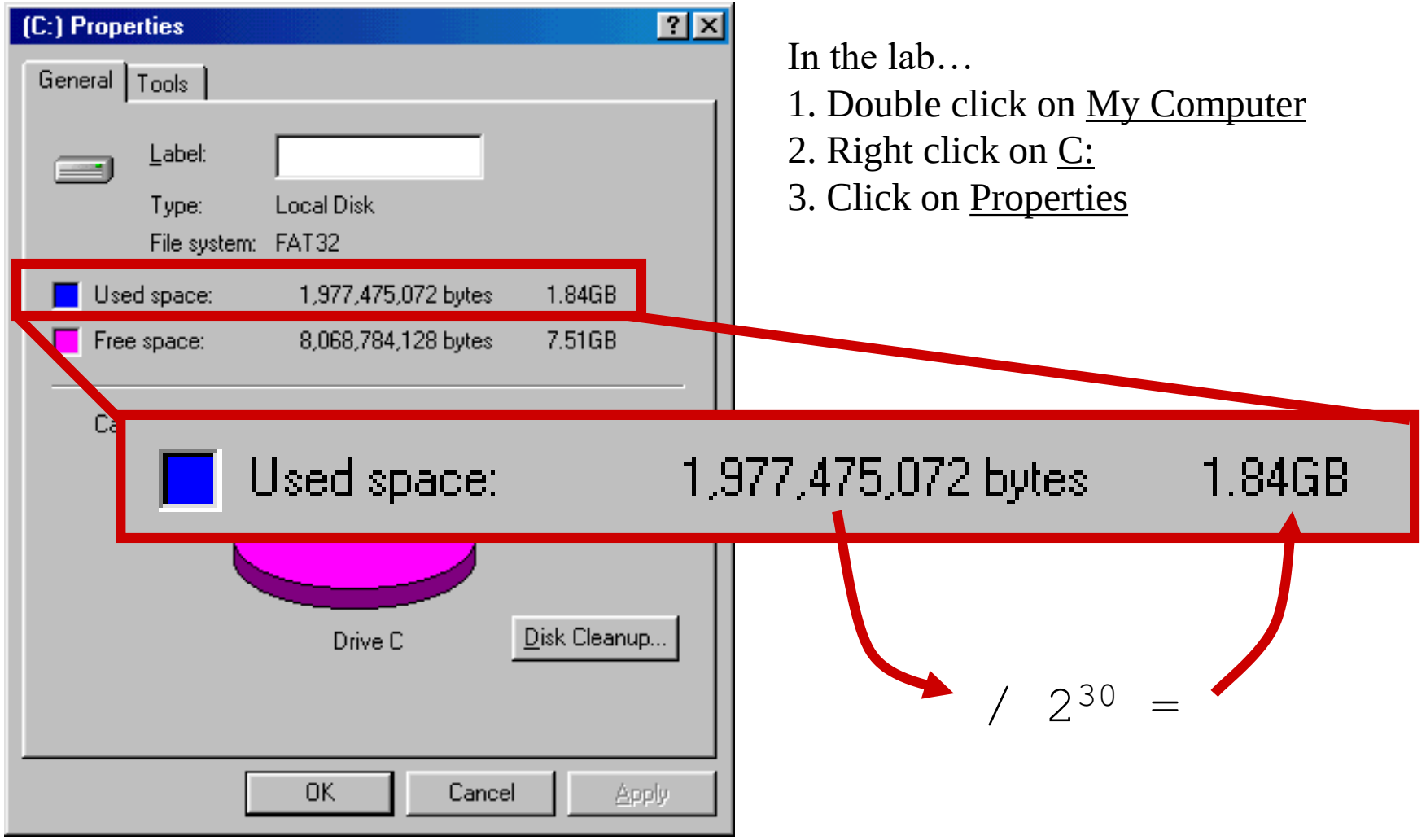
Common Powers (2 of 2)

- 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. memory, 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

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…

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

Thank you