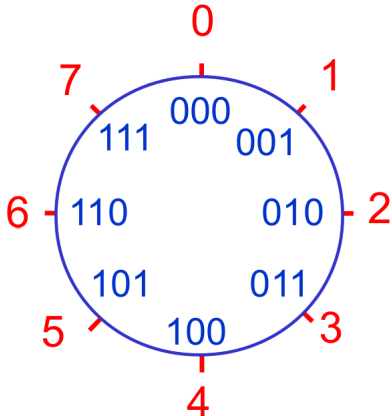


Representing values in Binary

If we have m bits, we can represent 2^m unique different values.
A useful circle :



Representing negative numbers

Sign Magnitude notation

- Use one bit for sign, others for magnitude of the number.

Representing negative numbers

Sign Magnitude notation

- Use one bit for sign, others for magnitude of the number.

	Sign Magn.
0 0 0	0
0 0 1	+1
0 1 0	+2
0 1 1	+3
1 0 0	0
1 0 1	-1
1 1 0	-2
1 1 1	-3

Representing negative numbers

Sign Magnitude notation

- Use one bit for sign, others for magnitude of the number.

	Sign Magn.
0 0 0	0
0 0 1	+1
0 1 0	+2
0 1 1	+3
1 0 0	0
1 0 1	-1
1 1 0	-2
1 1 1	-3

- using n bits: $-(2^{n-1} - 1) \dots (2^{n-1} - 1)$.
- zero has two representations.

Representing negative numbers

Ones complement notation

- for a negative number n , represent the number by the bit complement of its binary rep. using k bits.

Representing negative numbers

Ones complement notation

- for a negative number n , represent the number by the bit complement of its binary rep. using k bits.

	Sign Magn.	Ones comp.
0 0 0	0	0
0 0 1	+1	+1
0 1 0	+2	+2
0 1 1	+3	+3
1 0 0	0	-3
1 0 1	-1	-2
1 1 0	-2	-1
1 1 1	-3	0

Representing negative numbers

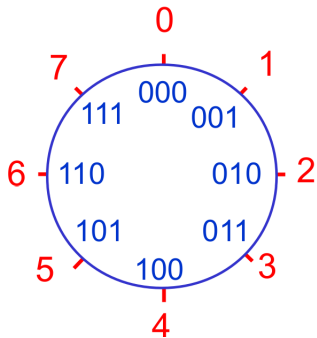
Ones complement notation

- for a negative number n , represent the number by the bit complement of its binary rep. using k bits.

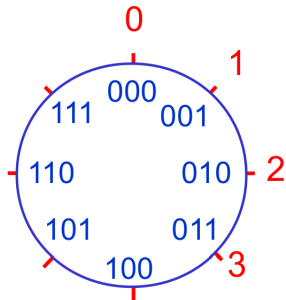
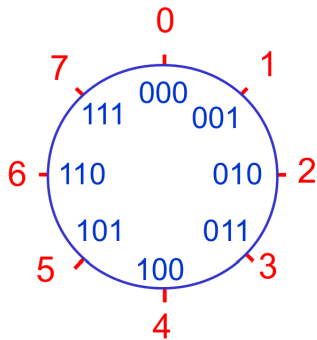
	Sign Magn.	Ones comp.
0 0 0	0	0
0 0 1	+1	+1
0 1 0	+2	+2
0 1 1	+3	+3
1 0 0	0	-3
1 0 1	-1	-2
1 1 0	-2	-1
1 1 1	-3	0

- using n bits: $-(2^{n-1} - 1) \dots (2^{n-1} - 1)$.
- zero has two representations.
- not very widely used representation.

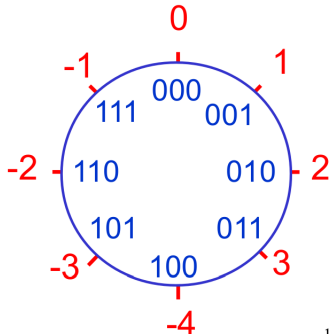
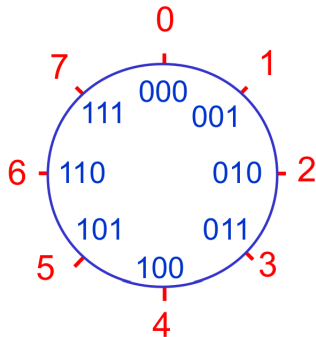
Representing negative numbers - A neat trick



Representing negative numbers - A neat trick



Representing negative numbers - A neat trick



Representing negative numbers - A neat trick

Twos complement notation

- for a negative number n , represent the number as $2^k - n$.

Representing negative numbers - A neat trick

Twos complement notation

- for a negative number n , represent the number as $2^k - n$.

	Sign Magn.	Ones comp.	Twos comp.
0 0 0	0	0	0
0 0 1	+1	+1	+1
0 1 0	+2	+2	+2
0 1 1	+3	+3	+3
1 0 0	0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	0	-1

Representing negative numbers - A neat trick

Twos complement notation

- for a negative number n , represent the number as $2^k - n$.

	Sign Magn.	Ones comp.	Twos comp.
0 0 0	0	0	0
0 0 1	+1	+1	+1
0 1 0	+2	+2	+2
0 1 1	+3	+3	+3
1 0 0	0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	0	-1

- using n bits: $-(2^{n-1}) \dots (2^{n-1} - 1)$.
- widely used representation.

Representing negative numbers

Arithmetic with these representations

	Sign Magn.	Ones comp.	Twos comp.
0 0 0	0	0	0
0 0 1	+1	+1	+1
0 1 0	+2	+2	+2
0 1 1	+3	+3	+3
1 0 0	0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	0	-1

Representing negative numbers

Arithmetic with these representations

	Sign Magn.	Ones comp.	Twos comp.
0 0 0	0	0	0
0 0 1	+1	+1	+1
0 1 0	+2	+2	+2
0 1 1	+3	+3	+3
1 0 0	0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	0	-1

- $2 + (-3)$

Representing negative numbers

Arithmetic with these representations

	Sign Magn.	Ones comp.	Twos comp.
0 0 0	0	0	0
0 0 1	+1	+1	+1
0 1 0	+2	+2	+2
0 1 1	+3	+3	+3
1 0 0	0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	0	-1

- $2 + (-3)$
- $3 + (-2)$

More examples : The case of 4 bits

	corresp. dec. oper.
$\begin{array}{r} 1 \ 1 \\ \sqrt{}\sqrt{} \\ 0110 \\ +1101 \\ \hline 10011 \end{array}$	$\begin{array}{r} +6 \\ + \ -3 \\ \hline \end{array}$
$\underbrace{10011}_{\text{correct result}} = +3$	$+3$

Example (a)

	corresp. dec. oper.
$\begin{array}{r} 0100 \\ +1001 \\ \hline 1101 \end{array}$	$\begin{array}{r} +4 \\ + \ -7 \\ \hline \end{array}$
$\underbrace{1101}_{\text{correct result}} = -3$	-3

Example (b)

More examples : The case of 4 bits

	corresp. dec. oper.
0011	+3
+0100	+ +4
0111 = +7	+7
	
correct result	
Example (c)	

	corresp. dec. oper.
1 1 1 1110	-2
+1010	+ -6
11000 = -8	-8
	
correct result	
Example (d)	

More examples : The case of 4 bits

1 √	corresp. dec. oper.
1101	-3
+1010	+ -6
10111	= +7 -9

incorrect result

Example (e)

1 √	corresp. dec. oper.
0101	+5
+0110	+ +6
1011	= -5 +11

incorrect result

Example (f)

Overflow Detection Rule : If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.

Goals for the day

- Edit, compile and execute the first C program.
- Get **simple yet useful** tasks done via C programs.
 - Add a set of numbers.
 - Find roots of a quadratic equation.
 - Multiply 2 polynomials.

Goals for the day

- Edit, compile and execute the first C program.
- Get **simple yet useful** tasks done via C programs.
 - Add a set of numbers.
 - Find roots of a quadratic equation.
 - Multiply 2 polynomials.
- Learn the **syntax** of C language.
 - Basics – structure of a C program, using standard library.
 - How to store data – variables, data types.
 - How to get inputs, how to print outputs?

Goals for the day

- Edit, compile and execute the first C program.
- Get **simple yet useful** tasks done via C programs.
 - Add a set of numbers.
 - Find roots of a quadratic equation.
 - Multiply 2 polynomials.
- Learn the **syntax** of C language.
 - Basics – structure of a C program, using standard library.
 - How to store data – variables, data types.
 - How to get inputs, how to print outputs?
- Learn about the working environment (Linux based OS).
 - editors – gedit and others.
 - compiler – gcc.
 - executing a compiled program.

First C program

```
#include <stdio.h>

/* My first C program */

void main() {
    printf("Hello World!\n");
}
```

First C program

```
#include <stdio.h>

/* My first C program */

void main() {
    printf("Hello World!\n");
}
```

- `stdio.h` : standard library of input and output.
- `main` : a function that every C program must have.
- `printf` : a useful library function to print several things in C.

First C program

```
#include <stdio.h>

/* My first C program */

void main() {
    printf("Hello World!\n");
}
```

- `stdio.h` : standard library of input and output.
- `main` : a function that every C program must have.
- `printf` : a useful library function to print several things in C.

To do anything more useful than merely printing we need to have more operations / commands and storage to store temporary computations.

Variables in C

Add 2 numbers x and y and store the value in z .

- Define two *variables* x and y .
 - What type of values can x and y take?

Variables in C

Add 2 numbers x and y and store the value in z .

- Define two *variables* x and y .
 - What type of values can x and y take?

integer, positive integers, fractions?

Variables in C

Add 2 numbers x and y and store the value in z .

- Define two *variables* x and y .
 - What type of values can x and y take?
integer, positive integers, fractions?
 - What do x and y represent?

Variables in C

Add 2 numbers x and y and store the value in z .

- Define two *variables* x and y .
 - What type of values can x and y take?
- What do x and y represent?

integer, positive integers, fractions?

Say marks in Maths and marks in Physics respectively.

Variables in C

Add 2 numbers x and y and store the value in z .

- Define two *variables* x and y .
 - What type of values can x and y take?
integer, positive integers, fractions?
 - What do x and y represent?
Say marks in Maths and marks in Physics respectively.
- Use the $+$ operator defined to sum up the values of x and y .

Variables in C

Add 2 numbers x and y and store the value in z .

- Define two *variables* x and y .
 - What type of values can x and y take?
integer, positive integers, fractions?
 - What do x and y represent?
Say marks in Maths and marks in Physics respectively.
- Use the $+$ operator defined to sum up the values of x and y .
- Use an *assignment* operator to store the value in z .

Sum of 2 numbers

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x;
    int y;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

Sum of 2 numbers

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x;
    int y;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

- **int** : defines that x, y, z are of type integers.
- **$z = x+y$** : evaluates $x+y$ and stores it in z .

Sum of 2 numbers

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x;
    int y;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

- **int** : defines that x, y, z are of type integers.
- **$z = x + y$** : evaluates $x + y$ and stores it in z .
- What will be output if we print z ?

Sum of 2 numbers

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x;
    int y;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

- **int** : defines that x, y, z are of type integers.
- **$z = x+y$** : evaluates $x+y$ and stores it in z .
- What will be output if we print z ?
- **Initialization** or **reading** of x and y missing.

Sum of 2 numbers

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x;
    int y;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

- **int** : defines that x, y, z are of type integers.
- **$z = x + y$** : evaluates $x + y$ and stores it in z .
- What will be output if we print z ?
- **Initialization** or **reading** of x and y missing.

Sum of 2 numbers – with initialization

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x = 98;
    int y = 99;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

- **int** : defines that x, y, z are of type integers.
- **$z = x+y$** : evaluates $x+y$ and stores it in z .
- What will be output if we print z ?

Sum of 2 numbers – with initialization

```
#include <stdio.h>

/* sum 2 integers */

main() {
    int x = 98;
    int y = 99;
    int z;

    z = x+y;
    printf("%d\n", z);
}
```

- **int** : defines that x, y, z are of type integers.
- **$z = x+y$** : evaluates $x+y$ and stores it in z .
- What will be output if we print z ?

Basic operators in C

- Arithmetic operators: $+$, $-$, $*$, $/$

Basic operators in C

- Arithmetic operators: $+$, $-$, $*$, $/$
- Modulus operator: $\%$
 $x \% y$: remainder when x is divided by y .

Basic operators in C

- Arithmetic operators: $+$, $-$, $*$, $/$
- Modulus operator: $\%$
 $x \% y$: remainder when x is divided by y .
- Assignment operator: $=$

Basic operators in C

- Arithmetic operators: `+`, `-`, `*`, `/`
 - Modulus operator: `%`
`x % y` : remainder when `x` is divided by `y`.
 - Assignment operator: `=`
-

Input statement: scanf

`scanf(format-string, &var1, &var2, ... , &var3);`

- scanf is a function which allows us to accept inputs.
- Usually functions take **fixed** number of parameters/arguments.
- scanf takes variable number of arguments.
- Notice the **&** preceeding the variables.