CS F111: Computer Programming

(Second Semester 2020-21)

Lect 20: 2’s complement & Recursion Intro

Dr. Nikumani Choudhury
Asst. Prof., Dept. of Computer Sc. & Information Systems
nikumani@hyderabad.bits-pilani.ac.in

BITS Pilani

Hyderabad Campus

One’s Complement

- Change all 0’s to 1’s and 1’s to 0’s

If MSB (most significant bit) is 1 then the number is negative (same as signed magnitude)

Range is:
$-(2^{N-1} - 1) \ldots (2^{N-1} - 1)$

PDP-1, CDC 3000, …

<u>Problem:</u>

- But zero has two representations

Applications: Foundation for 2’s complement, Error detection and Correction in data communications (computation of checksums)

-4	11011
-3	11100
-2	11101
-1	11110
-0	11111
+0	00000
+1	00001
+2	00010
+3	00011
+4	00100

Two’s Complement representation

- Transformation
  - To transform a into ‘-a’, invert all bits in ‘a’ and add 1 to the result.

Range is:
-(2^{N-1}) … (2^{N-1} – 1)

IBM 360, PDP-10, modern processors…

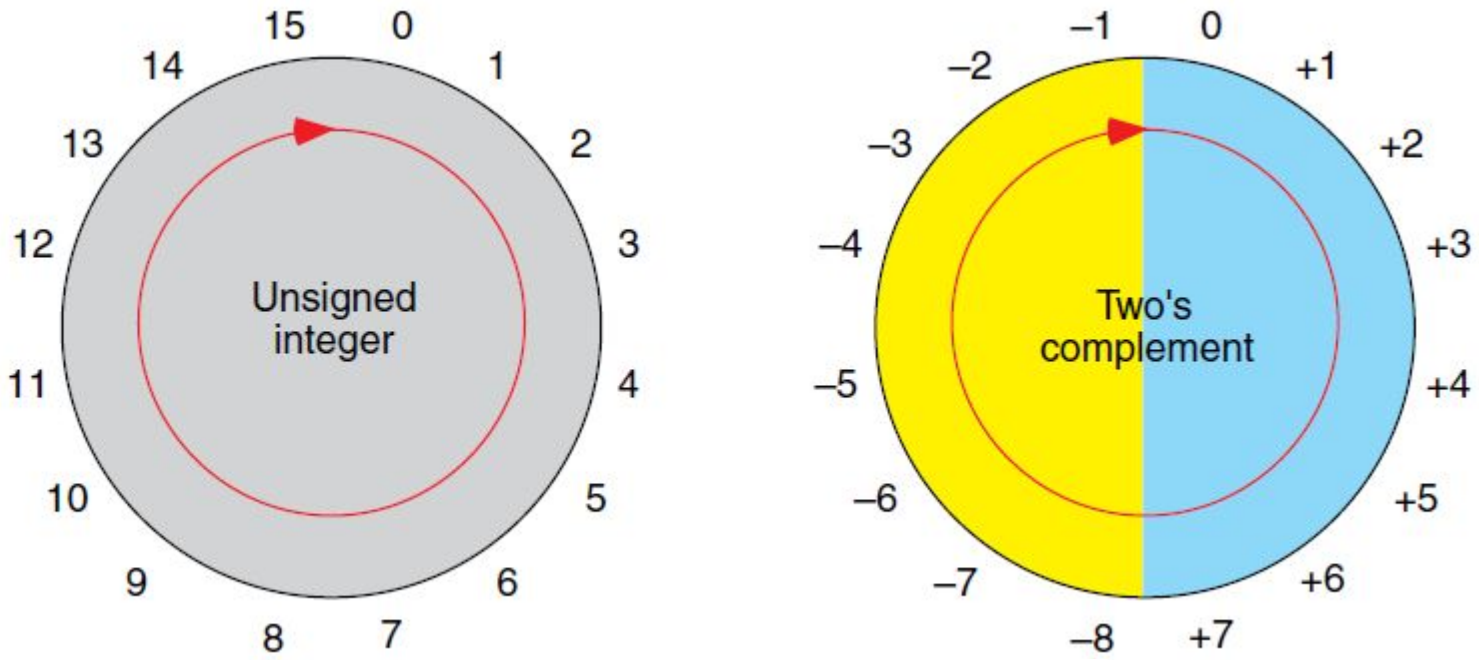
<u>Advantages:</u>

- Only one representation for zero
- Simplifies arithmetic and is used universally

-16	10000
…	…
-3	11101
-2	11110
-1	11111
0	00000
+1	00001
+2	00010
+3	00011
…	…
+15	01111

Overflow

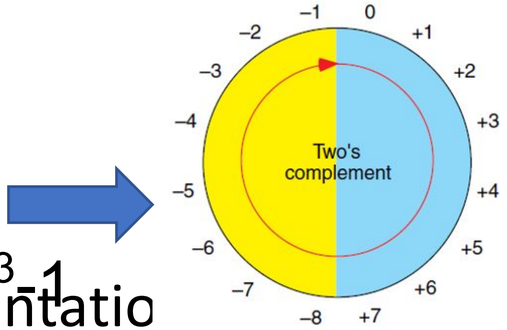
- Could be a source of concern for a programmer.
- Sometimes when we print a number, we get a surprising result.
- Numbers are stored on a limited sized word.



(Ex: if the size of integer is 4-b

Overflow continued…

- Let us add +7 with +5 and see the result. Use 4-bit representation in 2’s complement form.
- What is the problem?
- Overflowed the <u>number wheel</u>



-2^3 to 2^3-1

- Stepping upto 5 bit 2’s complement representatio
- In general, if the sum of two positive numbers produces a negative result, or vice versa, an overflow has occurred, and the result is invalid in that representation.

Overflow continued…

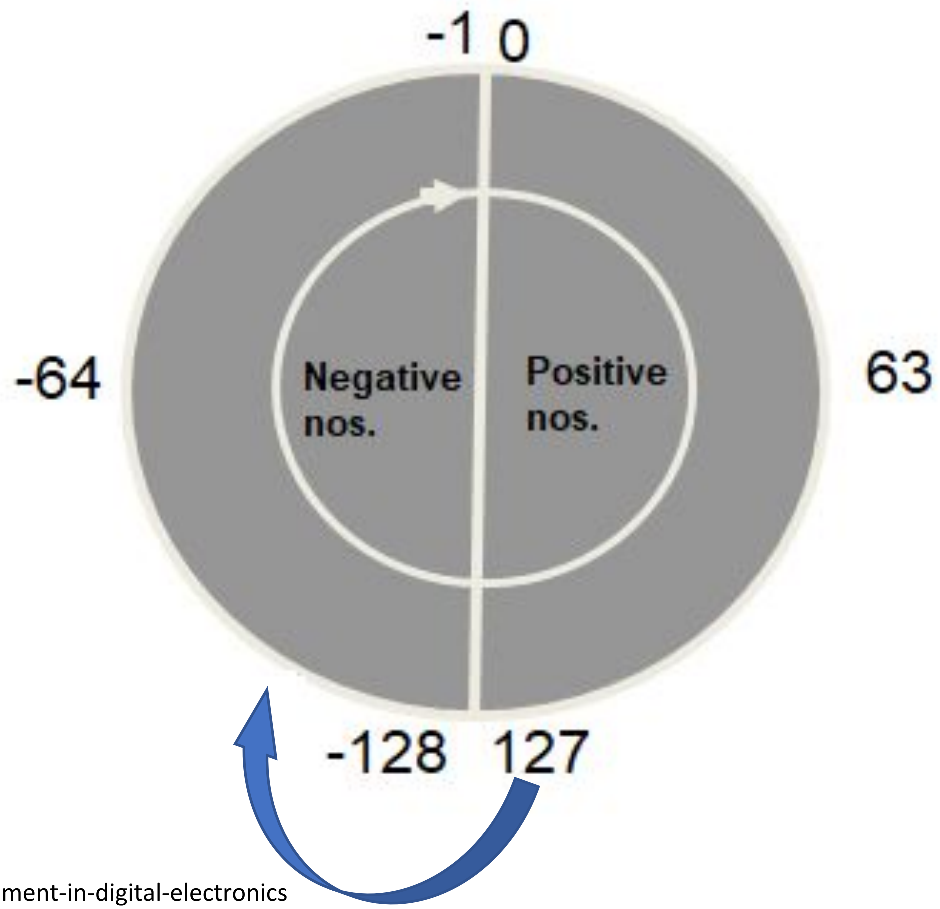
- Add 127 and 3 in 2’s complement representation with 8 bit format

01111111 +127

00000011 +3

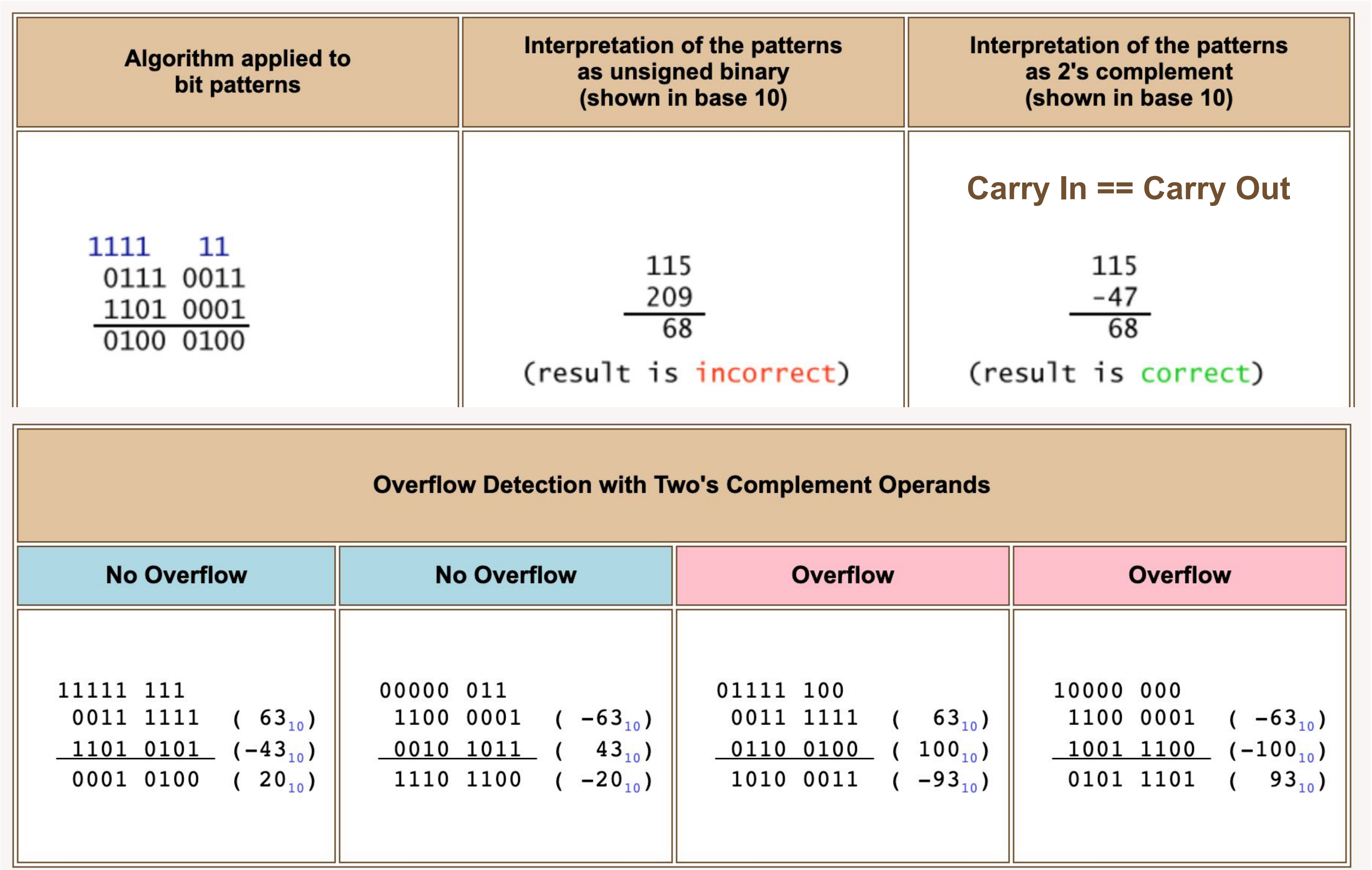
10000010

What is this number?



More: https://www.javatpoint.com/addition-and-subtraction-using-2s-complement-in-digital-electronics

2’s complement overflow



Algorithm applied to bit patterns	Interpretation of the patterns as unsigned binary (shown in base 10)	Interpretation of the patterns as 2's complement (shown in base 10)

1111   11
0111 0011
1101 0001
0100 0100
	115
	209
	68
	(result is incorrect)
		Carry In == Carry Out
		115
		-47
		68
		(result is correct)

Overflow Detection with Two's Complement Operands			
No Overflow	No Overflow	Overflow	Overflow
11111 111 0011 1111 (63_{10}) 1101 0101 (-43_{10}) 0001 0100 (20_{10})	00000 011 1100 0001 (-63_{10}) 0010 1011 (43_{10}) 1110 1100 (-20_{10})	01111 100 0011 1111 (63_{10}) 0110 0100 (100_{10}) 1010 0011 (-93_{10})	10000 000 1100 0001 (-63_{10}) 1001 1100 (-100_{10}) 0101 1101 (93_{10})

https://chortle.ccsu.edu/

Subtraction of two numbers using 2's Complement

- Negative numbers represented as 2’s Complement of Positive Numbers.

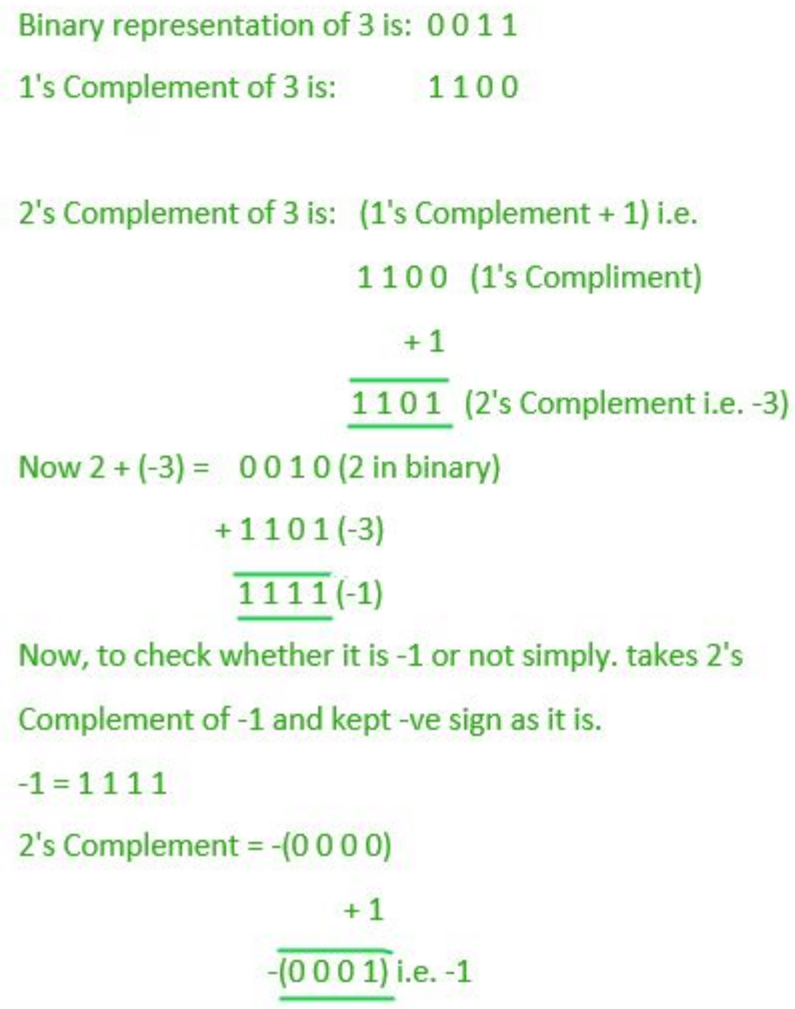
Subtract **b** from **a**. Write the expression (a-b) as:

$$(a - b) = a + (-b)$$

Now (-b) can be written as (2’s complement of b).
So the above expression can be now written as:

$$(a - b) = a + (2\text{'s complement of b})$$

So, the problem now reduces to “Add **a** to the 2’s complement of **b**“.



Binary representation of 3 is: 0 0 1 1

1's Complement of 3 is: 1 1 0 0

2's Complement of 3 is: (1's Complement + 1) i.e.

```
  1 1 0 0  (1's Compliment)
      + 1
  -------
  1 1 0 1  (2's Complement i.e. -3)
```

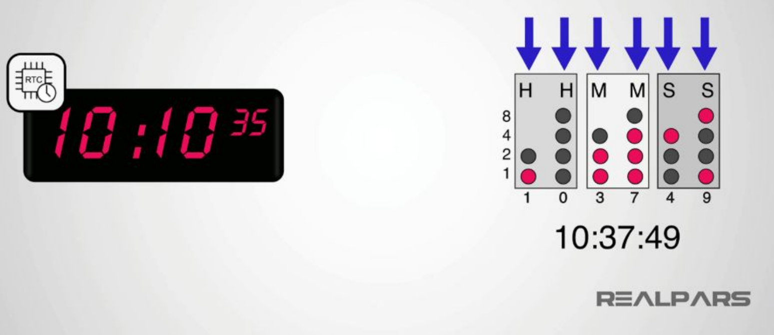
```
Now 2 + (-3) =    0 0 1 0 (2 in binary)
                + 1 1 0 1 (-3)
                  -------
                  1 1 1 1 (-1)
```

Now, to check whether it is -1 or not simply. takes 2's Complement of -1 and kept -ve sign as it is.

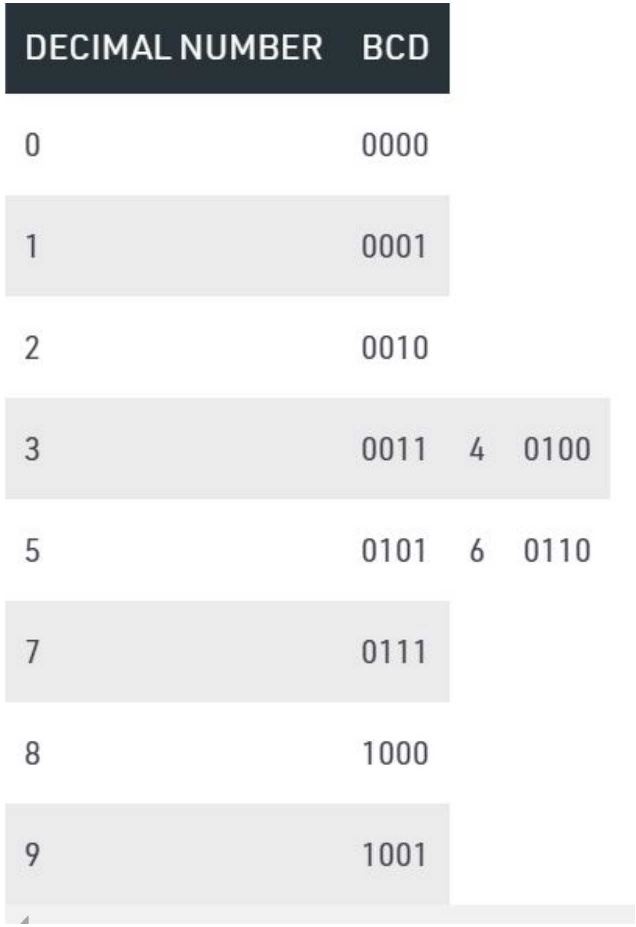
-1 = 1 1 1 1

```
2's Complement = -(0 0 0 0)
                       + 1
                 ---------
                 -(0 0 0 1) i.e. -1
```

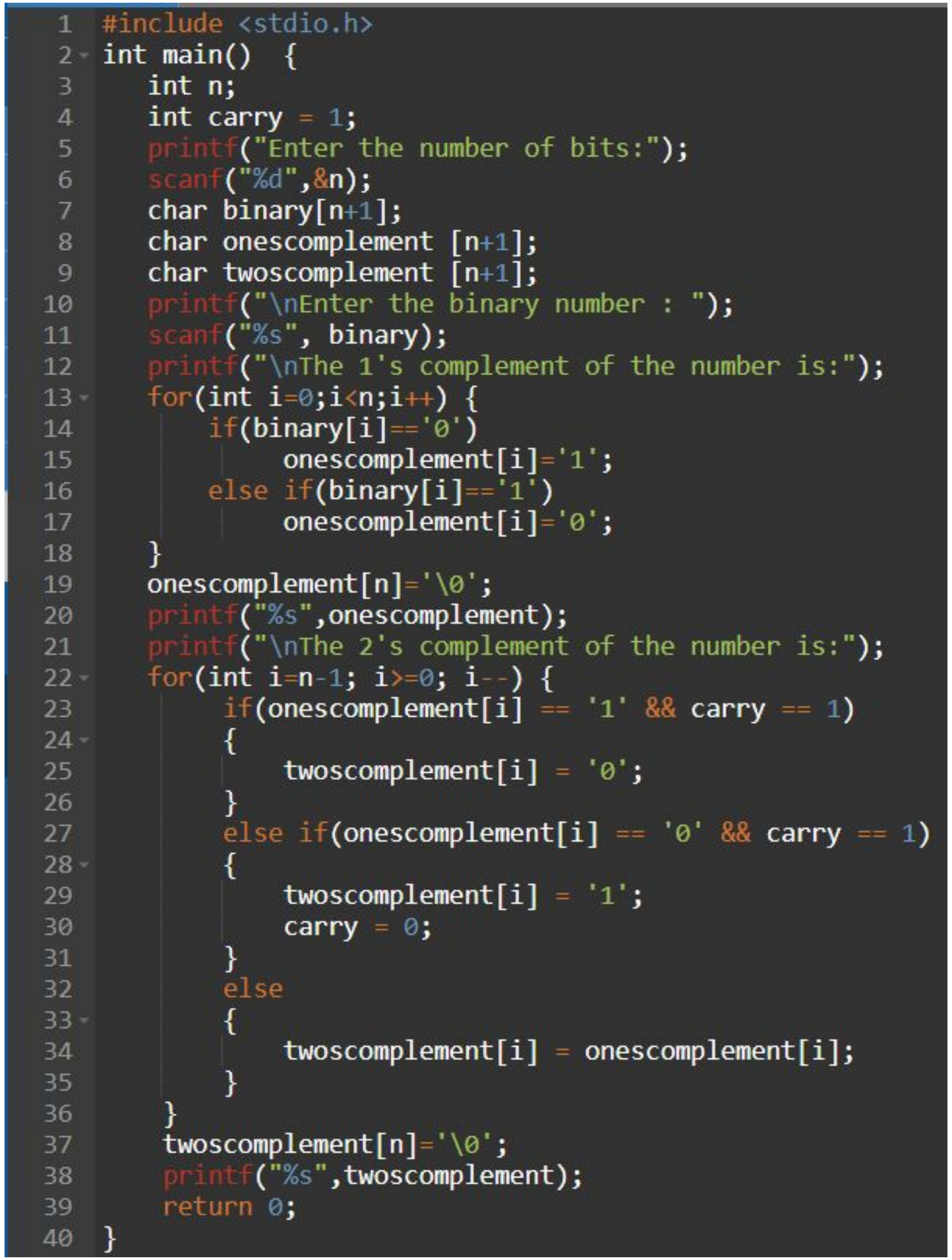
Substitution Code: Binary Coded Decimal (BCD)



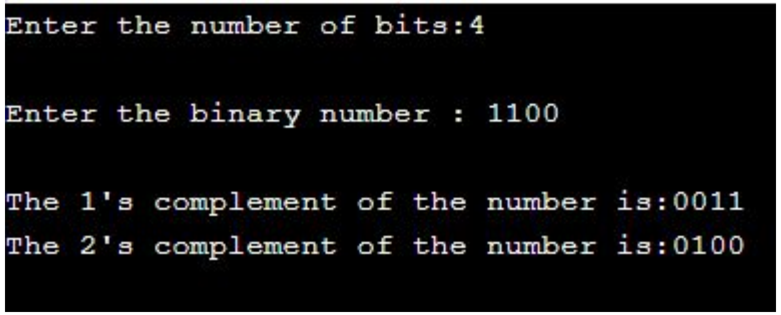
It is a process for converting decimal numbers into their binary equivalents.



DECIMAL NUMBER	BCD		
0	0000		
1	0001		
2	0010		
3	0011	4	0100
5	0101	6	0110
7	0111		
8	1000		
9	1001		



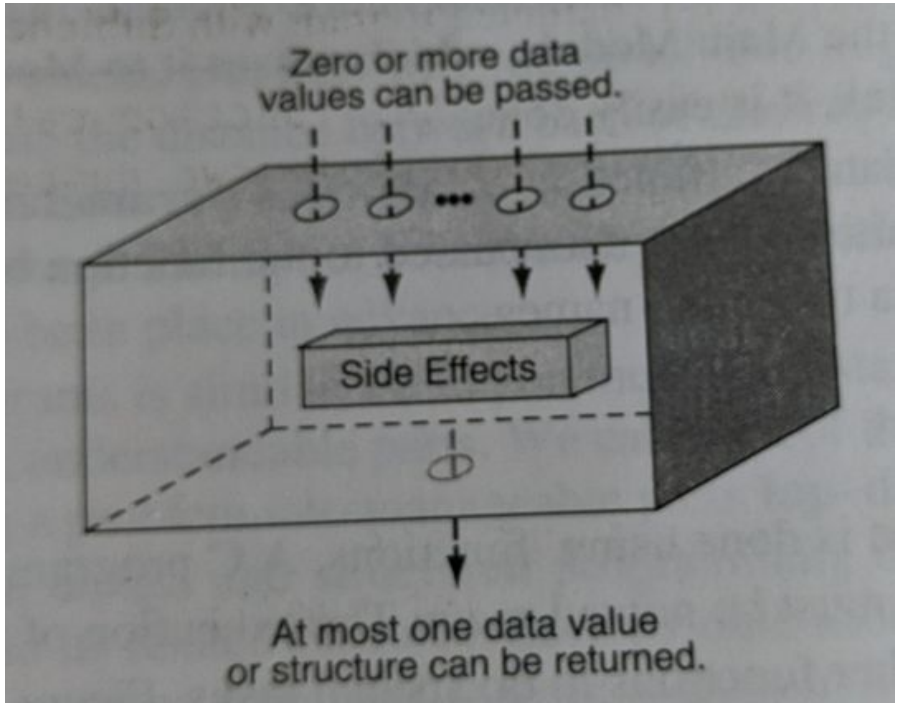
```c
#include <stdio.h>
int main()  {
   int n;
   int carry = 1;
   printf("Enter the number of bits:");
   scanf("%d",&n);
   char binary[n+1];
   char onescomplement [n+1];
   char twoscomplement [n+1];
   printf("\nEnter the binary number : ");
   scanf("%s", binary);
   printf("\nThe 1's complement of the number is:");
   for(int i=0;i<n;i++) {
       if(binary[i]=='0')
            onescomplement[i]='1';
       else if(binary[i]=='1')
            onescomplement[i]='0';
   }
   onescomplement[n]='\0';
   printf("%s",onescomplement);
   printf("\nThe 2's complement of the number is:");
   for(int i=n-1; i>=0; i--) {
        if(onescomplement[i] == '1' && carry == 1)
        {
            twoscomplement[i] = '0';
        }
        else if(onescomplement[i] == '0' && carry == 1)
        {
            twoscomplement[i] = '1';
            carry = 0;
        }
        else
        {
            twoscomplement[i] = onescomplement[i];
        }
    }
    twoscomplement[n]='\0';
    printf("%s",twoscomplement);
    return 0;
}
```



```
Enter the number of bits:4

Enter the binary number : 1100

The 1's complement of the number is:0011
The 2's complement of the number is:0100
```

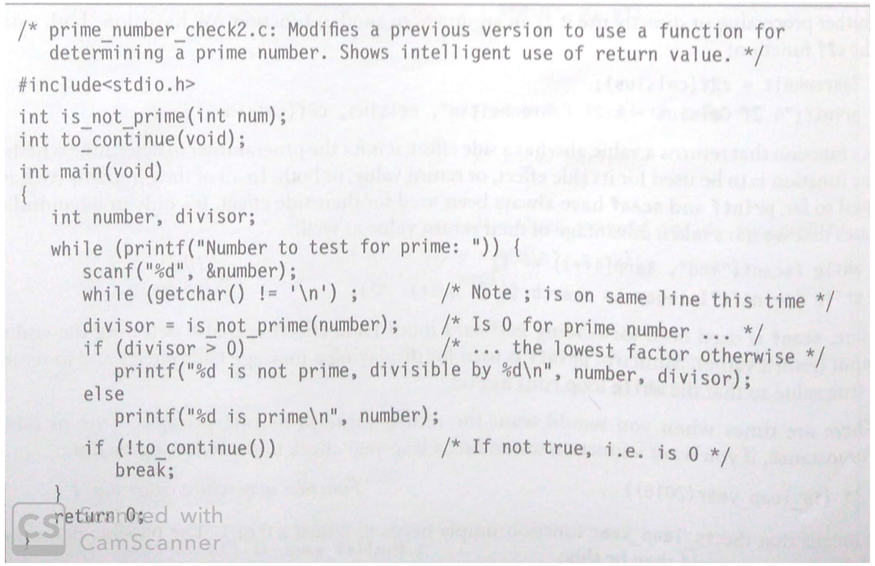
Modular Programing: User Defined Functions

- A function is a block of code that performs a specific task.
- C allows you to define functions according to your need in addition to a set of built-in library functions.
- Advantages (Manageable code, Reuse code, to protect data)
- A function in C can have a return value, a side effect, or both. The side effect occurs before the value is returned. A function can be called for a value, or for its side effects or both.

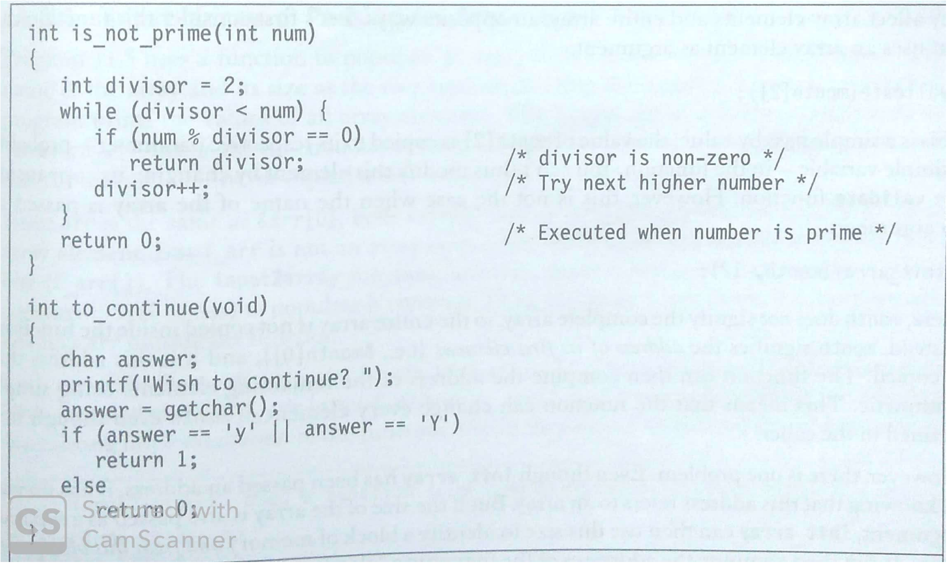


Zero or more data values can be passed.
Side Effects
At most one data value or structure can be returned.

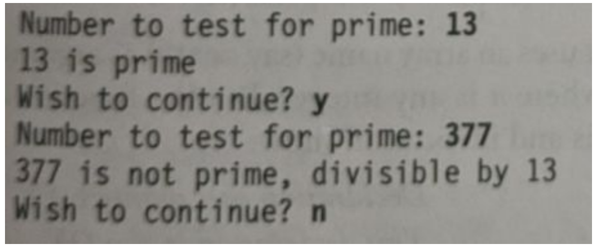
An Example



```
/* prime_number_check2.c: Modifies a previous version to use a function for
   determining a prime number. Shows intelligent use of return value. */
#include<stdio.h>
int is_not_prime(int num);
int to_continue(void);
int main(void)
{
   int number, divisor;
   while (printf("Number to test for prime: ")) {
      scanf("%d", &number);
      while (getchar() != '\n') ;            /* Note ; is on same line this time */
      divisor = is_not_prime(number);        /* Is 0 for prime number ... */
      if (divisor > 0)                       /* ... the lowest factor otherwise */
         printf("%d is not prime, divisible by %d\n", number, divisor);
      else
         printf("%d is prime\n", number);
      if (!to_continue())                    /* If not true, i.e. is 0 */
         break;
   }
   return 0;
}
```



```
int is_not_prime(int num)
{
   int divisor = 2;
   while (divisor < num) {
      if (num % divisor == 0)
         return divisor;                 /* divisor is non-zero */
      divisor++;                         /* Try next higher number */
   }
   return 0;                             /* Executed when number is prime */
}
int to_continue(void)
{
   char answer;
   printf("Wish to continue? ");
   answer = getchar();
   if (answer == 'y' || answer == 'Y')
      return 1;
   else
      return 0;
}
```



```
Number to test for prime: 13
13 is prime
Wish to continue? y
Number to test for prime: 377
377 is not prime, divisible by 13
Wish to continue? n
```

Recursion

- Repetitive algorithms: implemented using either loops or recursion
- Ex:
- fact(n) =

1 if n = 0

n*(n-1)*(n-2)*(n-3)*(n-4) … 3*2*1

if n > 0

Is it iterative or recursive?

- fact(n) =

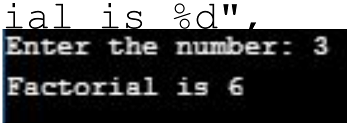
1 if n = 0

n*fact(n-1) if n > 0

- Designing recursive functions:
  - Divide the problem into subsets: 6! □ 5!, 4! etc…
  - Connect the subsets in a progressive manner: n! = n*(n-1)!
  - Identify the base case: for n!, it is at n=1.

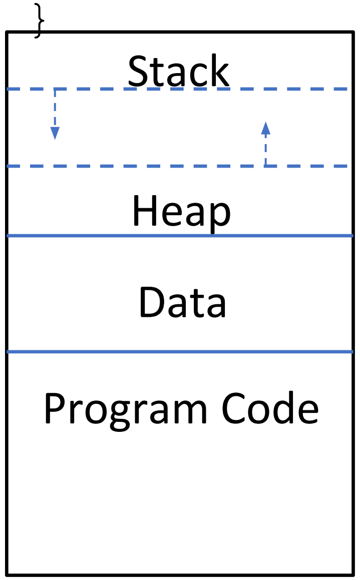
Recursive Functions: Fact

```
#include <stdio.h>
int fact (int n)  {
  if (n == 0)
     return 1;
  else
     return n * fact (n-1);
}

int main()  {
  int num;
  printf ("Enter the
number: ");

  scanf("%d", &num);
  printf("Factorial is %d",

fact(num));
return 0;
}
```



```
Enter the number: 3
Factorial is 6
```

```
#include <stdio.h>
int main(void)  {
  static long n;
  printf ("Call number %ld\n",
n++);
  main();
return 0;
}
```

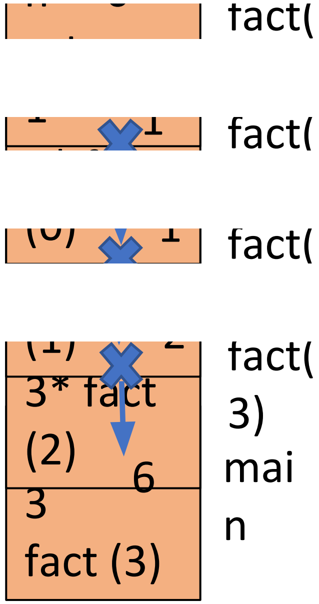


Stack

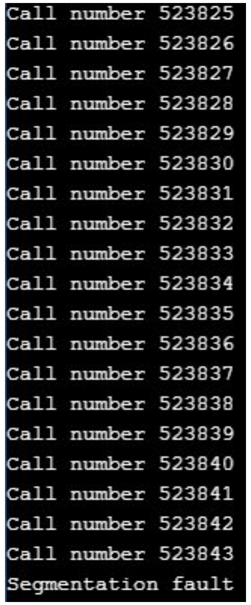
Heap

Data

Program Code



…



```
Call number 523825
Call number 523826
Call number 523827
Call number 523828
Call number 523829
Call number 523830
Call number 523831
Call number 523832
Call number 523833
Call number 523834
Call number 523835
Call number 523836
Call number 523837
Call number 523838
Call number 523839
Call number 523840
Call number 523841
Call number 523842
Call number 523843
Segmentation fault
```

STACK

OVERFLOW