# اعمال ریاضی با اعداد

# اعمال رياضي باينري: جمع

- قوانين: مانند جمع دسيمال
- با اين تفاوت كه 10 = 1+1 ← توليد نقلي

```
  +1 +1    +1
   1 0 1 1 0 1
+  0 1 1 0 0 1
-------------
   1 0 0 0 1 1 0
```

- 0+0 = 0(c0) (sum 0 with carry 0)
- 0+1 = 1+0 = 1(c0)
- 1+1 = 0(c1)
- 1+1+1 = 1(c1)

| Carry | 1 | 1 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|
| Augend | 0 | 0 | 1 | 0 | 0 | 1 |
| Addend | 0 | 1 | 1 | 1 | 1 | 1 |
| Result | 1 | 0 | 1 | 0 | 0 | 0 |

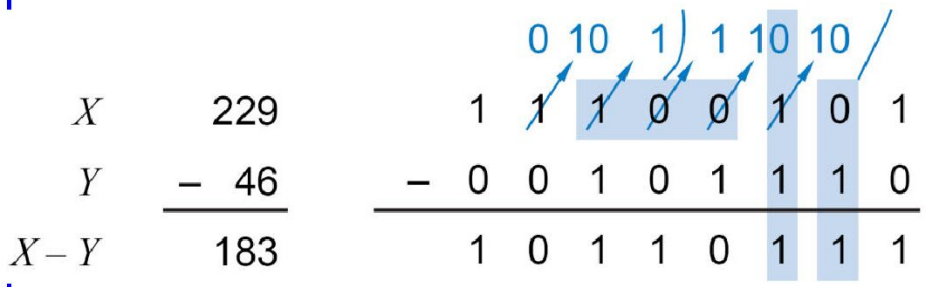
# سرریز (Overflow)

- اگر تعداد بیت ها = n و حاصل جمع n+1 بیت نیاز داشته باشد
  - ← سرریز

# اعمال رياضي باينري: تفريق

- قوانين:
  - 0-0 = 1-1 = 0 (b0) (result 0 with borrow 0)
  - 1-0 = 1 (b0)
  - 0-1 = 1 (b1)
  - …

| | | | | | |
|---|---|---|---|---|---|
| Borrow | 1 | 1 | 0 | 0 | |
| Minuend | 1 | 1 | 0 | 1 | 1 |
| Subtrahend | 0 | 1 | 1 | 0 | 1 |
| Result | 0 | 1 | 1 | 1 | 0 |



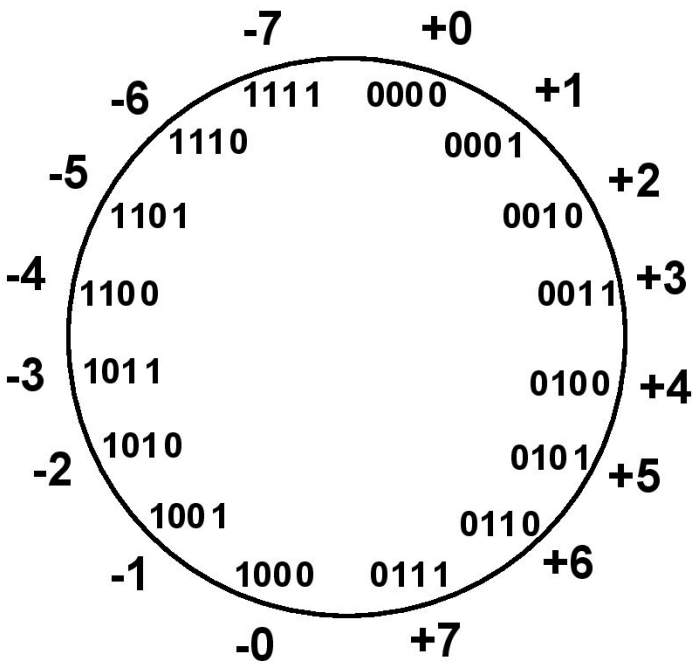
# روش انجام محاسبات

- الگوريتم هاي اعمال رياضي مبناي 10 را به خاطر آوريد.
- آنها را براي مبناي مورد نظر تعميم دهيد.
- قانون مبناي مورد نظر را به كار بريد.
  - براي باينري: 1+1=10

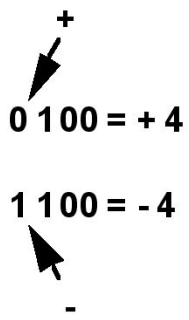
# نمایش اعداد

- **نمایش اعداد مثبت:**
  - در بیشتر سیستم ها یکسان است.
- **نمایش اعداد منفي:**
  - اندازه-علامت (Sign magnitude)
  - مکمل 1 (Ones complement)
  - مکمل 2 (Twos complement)
    - در بیشتر سیستم ها: مکمل 2
- **فرض:**
  - ماشین با کلمه هاي 4 بیتي:
    - 16 ← مقدار مختلف قابل نمایش.
    - تقریباً نیمي مثبت، نیمي منفي.

# نمایش اعداد




## اندازه-علامت:




High order bit is sign: 0 = positive (or zero), 1 = negative

Three low order bits is the magnitude: 0 (000) thru 7 (111)

Number range for n bits = $[-(2^{n-1} -1), +(2^{n-1} -1)]$

Representations for 0

Cumbersome addition/subtraction

Must compare magnitudes to determine sign of result

# نمایش اعداد

## مکمل 1:

N is positive number, then $\overline{N}$ is its negative 1's complement

$$\overline{N} = (2^n - 1) - N$$

Example: 1's complement of 7

$2^4 = 10000$

$-1 = 00001$

$1111$

$-7 = 0111$

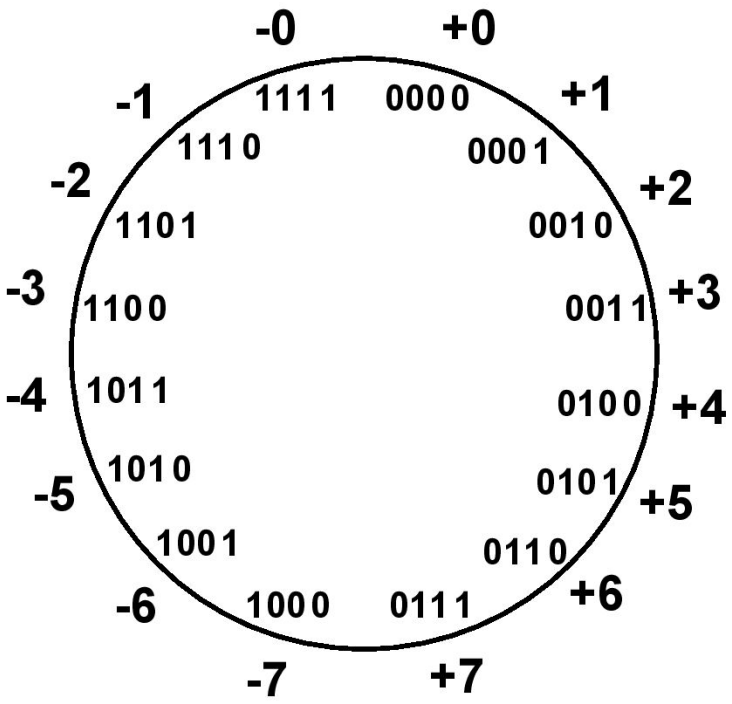
$1000$ = -7 in 1's comp.

Shortcut method:

simply compute bitwise complement

0111 -> 1000

# نمایش اعداد




## مکمل 1:

+

0 1 0 0 = + 4

1 0 1 1 = - 4

-

**Subtraction implemented by addition & 1's complement**

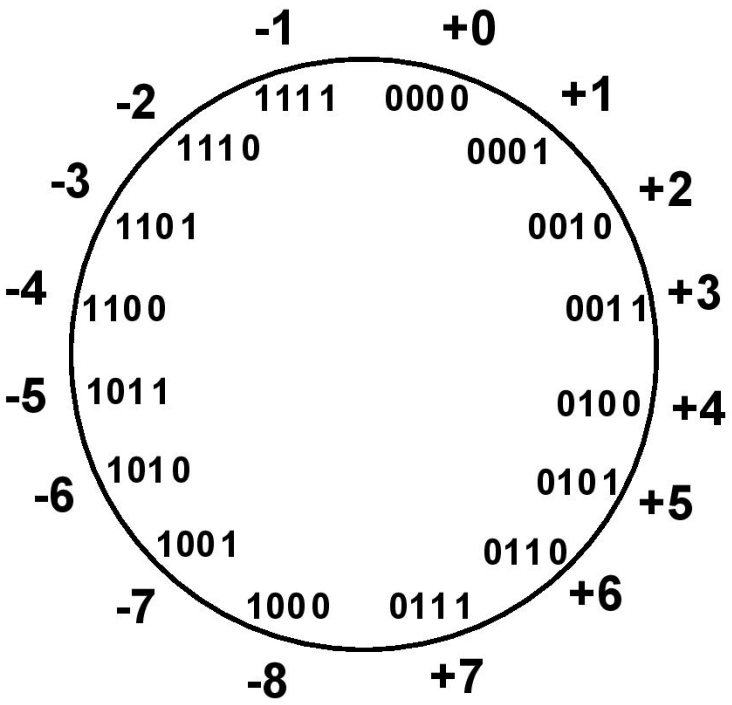
**Still two representations of 0! This causes some problems**

**Some complexities in addition**

# نمایش اعداد

## مکمل 2:

*like 1's comp except shifted one position clockwise*




+

0 1 0 0 = + 4

1 1 0 0 = - 4

-

**Only one representation for 0**

**One more negative number than positive number**

# نمایش اعداد

$N^* = 2^n - N$

## مکمل 2:

Example: Twos complement of 7

$2^4 = 10000$

sub 7 = 0111

1001 = repr. of -7

Example: Twos complement of -7

$2^4 = 10000$

sub -7 = 1001

0111 = repr. of 7

Shortcut method:

Twos complement = bitwise complement + 1

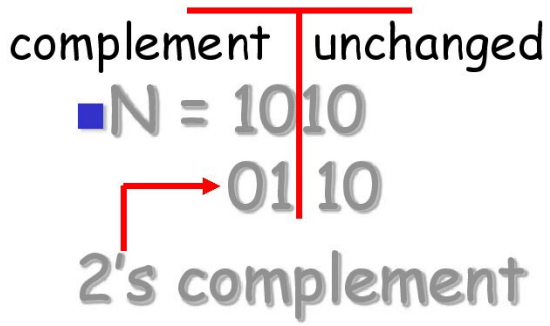
0111 -> 1000 + 1 -> 1001 (representation of -7)

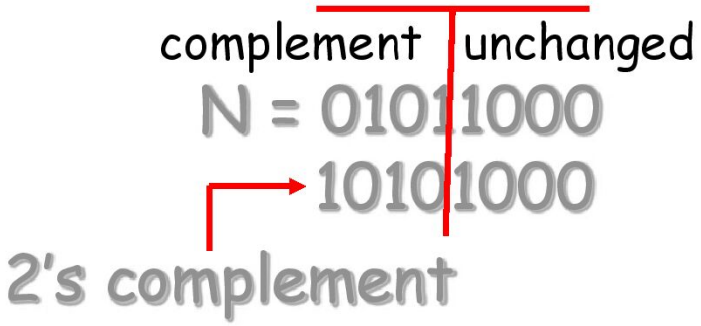
1001 -> 0110 + 1 -> 0111 (representation of 7)

# مكمل 2

- **Here's an easier way to compute the *2's* complement:**
  1. Leave all least significant 0's and first 1 unchanged.
  2. Replace 0 with 1 and 1 with 0 in all remaining higher significant bits.

- Examples:



# جمع و تفریق
# مکمل 2

| | | | |
|---|---|---|---|
| 4 | 0100 | -4 | 1100 |
| + 3 | 0011 | + (-3) | 1101 |
| 7 | 0111 | -7 | 11001 |

If
(carry-in to sign = carry-out )
then ignore carry

if
(carry-in ≠ carry-out)
then overflow

| | | | |
|---|---|---|---|
| 4 | 0100 | -4 | 1100 |
| - 3 | 1101 | + 3 | 0011 |
| 1 | 10001 | -1 | 1111 |

**Simpler addition scheme makes twos complement the most common choice for integer number systems**

# جمع و تفریق
# مکمل 2

**Why can the carry-out be ignored?**

**-M + N when N > M:**

$$M^* + N = (2^n - M) + N = 2^n + (N - M)$$

**Ignoring carry-out is just like subtracting $2^n$**

**-M + -N where N + M < or = $2^{n-1}$**

$$-M + (-N) = M^* + N^* = (2^n - M) + (2^n - N)$$

$$= 2^n - (M + N) + 2^n$$

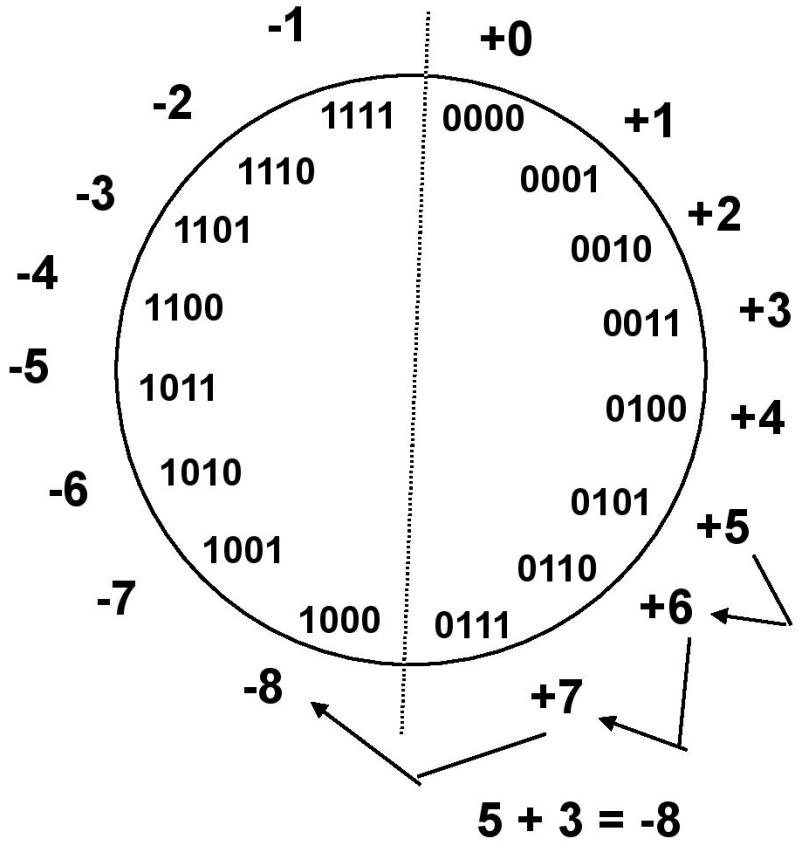
**After ignoring the carry, this is just the right twos compl. representation for -(M + N) or (M+N)***

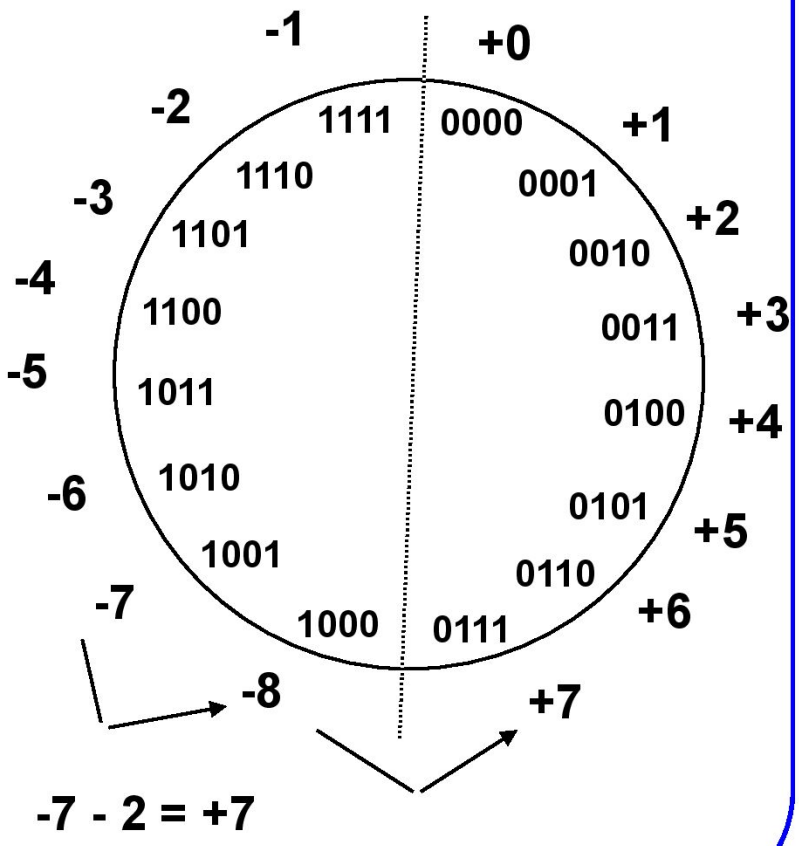
سرریز

## *Overflow Conditions*

**Add two positive numbers to get a negative number**

**or two negative numbers to get a positive number**



سرریز

## *Overflow Conditions*

| | | |
|---|---|---|
| | 0 1 1 1 | |
| 5 | 0 1 0 1 | |
| 3 | 0 0 1 1 | |
| -8 | 1 0 0 0 | |

**Overflow**

| | | |
|---|---|---|
| | 1 0 0 0 | |
| -7 | 1 0 0 1 | |
| -2 | 1 1 1 0 | |
| 7 | 1 0 1 1 1 | |

**Overflow**

| | | |
|---|---|---|
| | 0 0 0 0 | |
| 5 | 0 1 0 1 | |
| 2 | 0 0 1 0 | |
| 7 | 0 1 1 1 | |

**No overflow**

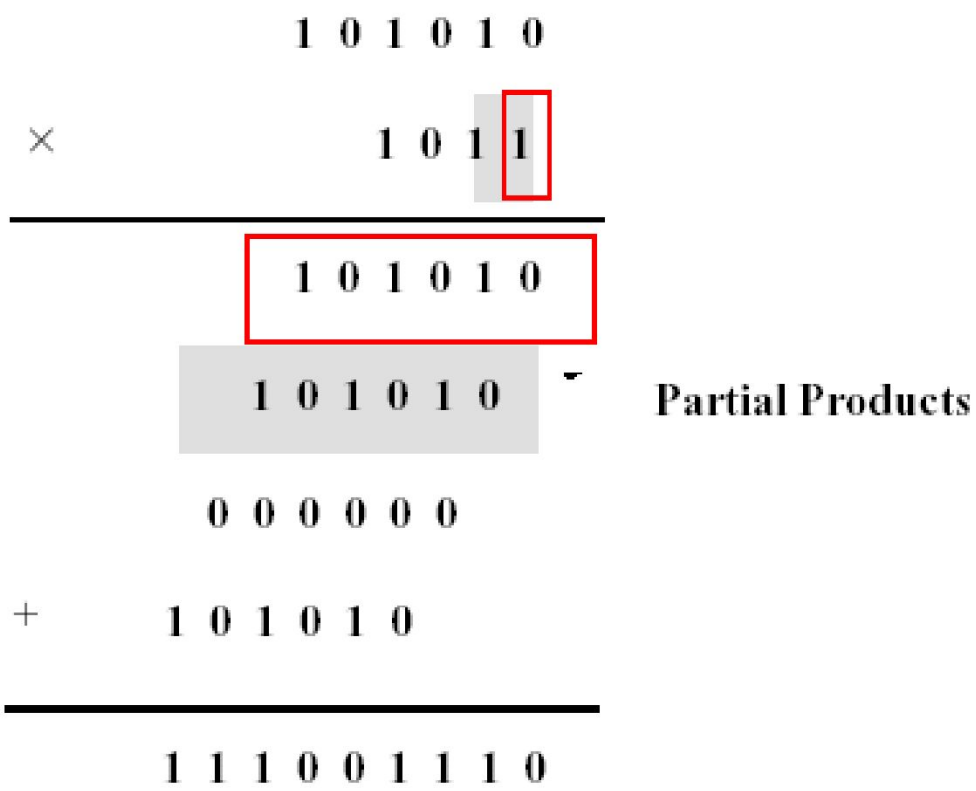
| | | |
|---|---|---|
| | 1 1 1 1 | |
| -3 | 1 1 0 1 | |
| -5 | 1 0 1 1 | |
| -8 | 1 1 0 0 0 | |

**No overflow**

**Method 1: Overflow when carry in to sign ≠ carry out**

**Method 2: Overflow when sign(A) = sign(B) ≠ sign (result)**

# The Binary Multiplication



# ضرب باينري

- **Shift-and-add algorithm, as in base 10**

| M'cand | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| M'plier | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| (1) | | | 0 | 0 | 0 | 0 | 0 |
| (2) | | 0 | 1 | 1 | 0 | 1 | |
| (3) | 0 | 1 | 1 | 0 | 1 | | |
| Sum | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

- **Check: 13 * 6 = 78**

# Binary-Coded Decimal (BCD)

- A decimal code: Decimal numbers (0..9) are coded using 4-bit distinct binary words
- Observe that the codes 1010 .. 1111 (decimal 10..15) are NOT represented (invalid BCD codes)

**TABLE 1-3**
**Binary-Coded Decimal (BCD)**

| Decimal Symbol | BCD Digit |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Table 1-3 Binary-Coded Decimal (BCD)

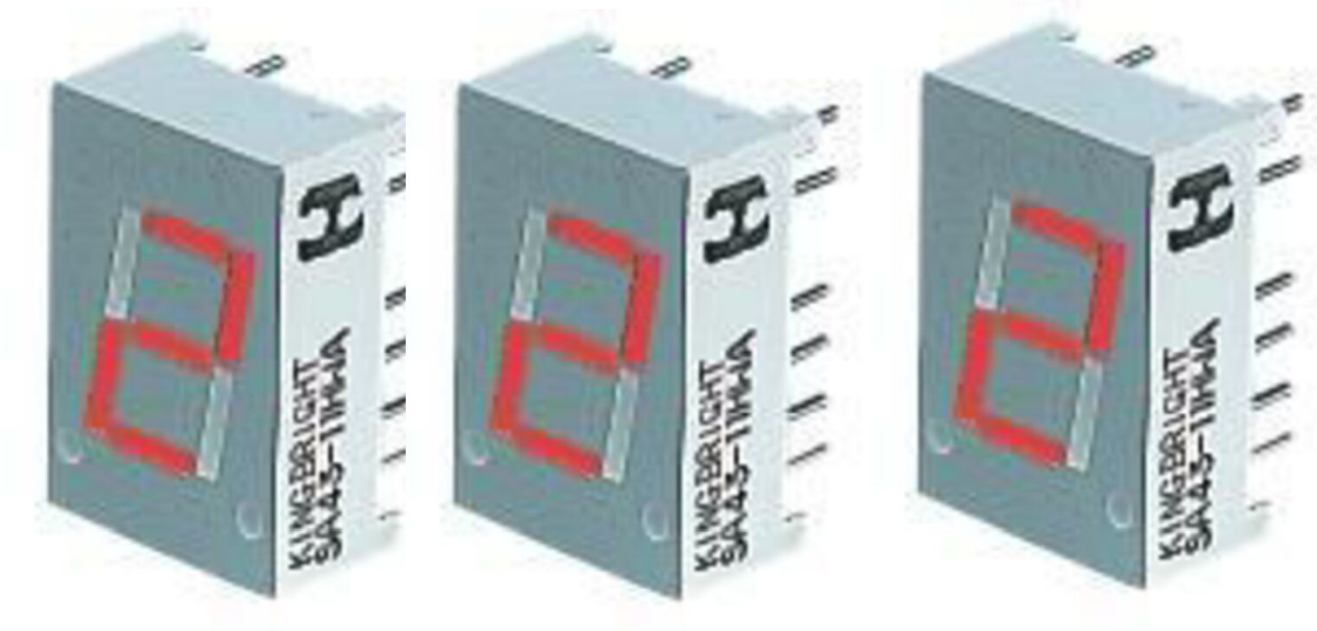
# Binary-Coded Decimal

- To code a number with *n* decimal digits, we need *4n* bits in BCD

  e.g. $(365)_{10} = (0011\ 0110\ 0101)_{BCD}$

- This is different from converting to binary, which is $(365)_{10} = (101101101)_2$
- Clearly, BCD requires more bits. BUT, it is easier to understand/interpret

# Application



# BCD Addition

Case 1:

| | |
|---|---|
| 0001 | 1 |
| 0101 | 5 |
| (0) 0110 | (0) 6 |

Case 2:

| | |
|---|---|
| 0110 | 6 |
| 0101 | 5 |
| (0) 1011 | (1) 1 |

WRONG!

Case 3:

| | |
|---|---|
| 1000 | 8 |
| 1001 | 9 |
| (1) 0001 | (1) 7 |

Note that for cases 2 and 3, adding a factor of 6 (0110) gives us the correct result.

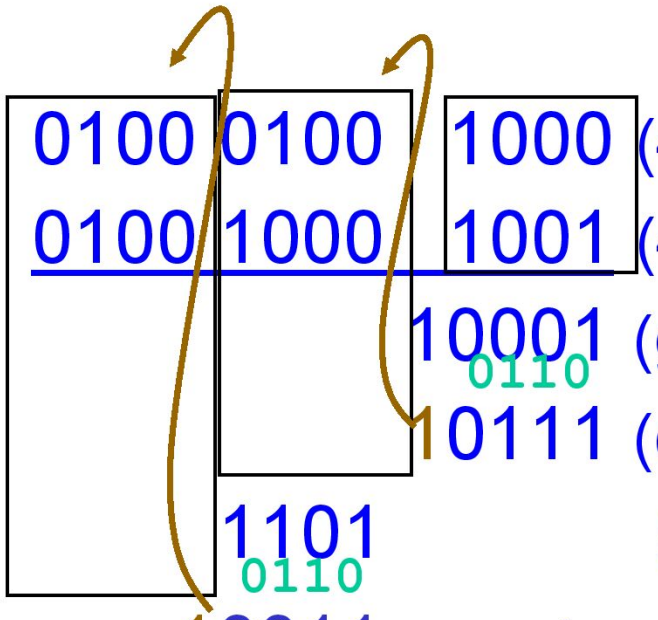
# BCD Addition (cont.)

- **BCD addition is therefore performed as follows**
  1) Add the two BCD digits together using normal binary addition
  2) Check if correction is needed
     a) 4-bit sum is in range of 1010 to 1111
     b) carry out of MSB = 1
  3) If correction is required, add 0110 to 4-bit sum to get the correct result;

     → **BCD carry out = 1**

# BCD Addition (cont.)

- **Example: Add 448 and 489 in BCD.**



0100 0100 1000 (448 in BCD)
0100 1000 1001 (489 in BCD)
10001 (greater than 9, add 6)
0110
10111 (carry 1 into middle digit)
1101 (greater than 9, add 6)
0110
10011 (carry 1 into leftmost digit)
1001 0011 0111 (BCD coding of $937_{10}$)

# BCD Negative Number Representation

- **Similar to binary negative number representation but for r = 10.**
  - BCD 9's complement
    - invert each BCD digit (0→9, 1 → 8, 2 → 7,3 → 6, …7 → 2, 8 → 1, 9 → 0)
  - BCD 10's complement
    - $-N \equiv 10^n - N$;  9's complement + 1

# Excess-3

- مانند BCD ولي هر رقم 3+
  - جمع سرراست تر
  - self-comlpement code
    - (مکمل هر رقم = مکمل 9 آن)

| Decimal Digit | BCD | | | | Excess-3 | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | W | X | Y | Z |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

# ASCII character code

- We also need to represent letters and other symbols → alphanumeric codes
- ASCII = American Standard Code for Information Interchange. Also known as Western European
- It contains 128 characters:
  - 94 printable ( 26 upper case and 26 lower case letters, 10 digits, 32 special symbols)
  - 34 non-printable (for control functions)
- Uses 7-bit binary codes to represent each of the 128 characters

# ASCII Table

| $b_3b_2b_1b_0$ | Row (hex) | $b_6b_5b_4$ (column) 000 0 | 001 1 | 010 2 | 011 3 | 100 4 | 101 5 | 110 6 | 111 7 |
|---|---|---|---|---|---|---|---|---|---|
| 0000 | 0 | NUL Null | DLE | SP Space | 0 | @ | P | ` | p |
| 0001 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | 2 | STX | DC2 | " | 2 | B | R | b | r |
| 0011 | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0101 | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | 6 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | 7 | BEL Bell | ETB | ' | 7 | G | W | g | w |
| 1000 | 8 | BS BkSpc | CAN | ( | 8 | H | X | h | x |
| 1001 | 9 | HT Tab | EM | ) | 9 | I | Y | i | y |
| 1010 | A | LF Line Fd | SUB | * | : | J | Z | j | z |
| 1011 | B | VT | ESC Escape | + | ; | K | [ | k | { |
| 1100 | C | FF | FS | , | < | L | \ | l | \| |
| 1101 | D | CR Crg Ret | GS | - | = | M | ] | m | } |
| 1110 | E | SO | RS | . | > | N | ^ | n | ~ |
| 1111 | F | SI | US | / | ? | O | _ | o | DEL |

# ASCII Control Codes

| Control codes | | | |
|---|---|---|---|
| NUL | Null | DLE | Data link escape |
| SOH | Start of heading | DC1 | Device control 1 |
| STX | Start of text | DC2 | Device control 2 |
| ETX | End of text | DC3 | Device control 3 |
| EOT | End of transmission | DC4 | Device control 4 |
| ENQ | Enquiry | NAK | Negative acknowledge |
| ACK | Acknowledge | SYN | Synchronize |
| BEL | Bell | ETB | End transmitted block |
| BS | Backspace | CAN | Cancel |
| HT | Horizontal tab | EM | End of medium |
| LF | Line feed | SUB | Substitute |
| VT | Vertical tab | ESC | Escape |
| FF | Form feed | FS | File separator |
| CR | Carriage return | GS | Group separator |
| SO | Shift out | RS | Record separator |
| SI | Shift in | US | Unit separator |
| SP | Space | DEL | Delete or rubout |

# Unicode

- Established standard (16-bit alphanumeric code) for international character sets
- Since it is 16-bit, it has 65,536 codes
- Represented by 4 Hex digits
- ASCII is between $0000_{16}$ .. $007B_{16}$

# Unicode Table

http://www.unicode.org/charts/

# Unicode

| | | | | |
|---|---|---|---|---|
| 062B 1579 | ث 062C 1580 | ج 062D 1581 | ح 062E 1582 | خ |
| 0633 1587 | س 0634 1588 | ش 0635 1589 | ص0636 1590 | ض |
| 063B 1595 | ك 063C 1596 | ک 063D 1597 | ى 063E 1598 | ى |
| 0643 1603 | ك 0644 1604 | ل 0645 1605 | م 0646 1606 | ن |
| 064B 1611 | ً 064C 1612 | ٌ 064D 1613 | ٍ 064E 1614 | َ |
| 0653 1619 | ٓ 0654 1620 | ُ 0655 1621 | ِ 0656 1622 | ِ |
| 065B 1627 | ٛ 065C 1628 | ٜ 065D 1629 | ٞ 065E 1630 | ً |
| 0663 1635 | ٣ 0664 1636 | ٤ 0665 1637 | ٥ 0666 1638 | ٦ |
| 066B 1643 | , 066C 1644 | , 066D 1645 | * 066E 1646 | ـ |
| 0673 1651 | إ 0674 1652 | ٔ 0675 1653 | أ 0676 1654 | ؤ |
| 067B 1659 | ٻ 067C 1660 | ټ 067D 1661 | ٹ 067E 1662 | پ |
| 0683 1667 | چ 0684 1668 | ڃ 0685 1669 | څ 0686 1670 | چ |
| 068B 1675 | ڋ 068C 1676 | ڈ 068D 1677 | ډ 068E 1678 | ڈ |

# ASCII Parity Bit

- Parity coding is used to detect errors in data communication and processing
  - An $8^{th}$ bit is added to the 7-bit ASCII code
- Even (Odd) parity: set the parity bit so as to make the # of 1's in the 8-bit code even (odd)

# ASCII Parity Bit (cont.)

- **For example:**
  - Make the 7-bit code 1011011 an 8-bit even parity code → 11011011
  - Make the 7-bit code 1011011 an 8-bit odd parity code → 01011011
- **Error Checking:**
  - Both even and odd parity codes can detect an odd number of error.
    - An even number of errors goes undetected.

# Gray Codes

- **Gray codes are *minimum change* codes**
  - From one numeric representation to the next, only one bit changes
  - Applications:
    - Later.

# Gray Codes (cont.)

| Binary | Gray |
|---|---|
| 0000 | 0000 |
| 0001 | 0001 |
| 0010 | 0011 |
| 0011 | 0010 |
| 0100 | 0110 |
| 0101 | 0111 |
| 0110 | 0101 |
| 0111 | 0100 |
| 1000 | 1100 |
| 1001 | 1101 |
| 1010 | 1111 |
| 1011 | 1110 |
| 1100 | 1010 |
| 1101 | 1011 |
| 1110 | 1001 |
| 1111 | 1000 |

| Binary | Gray |
|---|---|
| 00 | 00 |
| 01 | 01 |
| 10 | 11 |
| 11 | 10 |

| Binary | Gray |
|---|---|
| 000 | 000 |
| 001 | 001 |
| 010 | 011 |
| 011 | 010 |
| 100 | 110 |
| 101 | 111 |
| 110 | 101 |
| 111 | 100 |