دانشگاه شهید بهشتی

درس «مبانی کامپیوتر و برنامه‌سازی»

نمایش اعداد

صادق علی‌اکبری

سرفصل مطالب

- اعداد در مبناهای مختلف
  - مبنای ۱۰، ۲ ، ۸ و ۱۶
- نمایش باینری اعداد
- نمایش اعداد علامت‌دار
- نمایش اعداد اعشاری

نحوه نمایش اطلاعات در کامپیوتر

- اطلاعات در کامپیوتر به شکل‌های مختلفی ذخیره می‌شود
  - عدد ساده، اعداد اعشاری و علامت‌دار، کاراکتر
  - متن، تصویر، ویدیو، صدا (موسیقی، صوت) و ..
- داده‌های کامپیوتری در نهایت به شکل عدد ذخیره می‌شوند
  - حتی داده‌های غیرعددی مثل تصویر یا متن
- اما نحوه ذخیره اعداد در کامپیوتر چگونه است؟
  - مثلاً در حافظه اصلی یا حافظه جانبی

مبنای ۱۰

Base 10
Decimal
Radix-10

- مبنای مورد استفاده انسانها در ریاضیات: مبنای ۱۰
- هر عدد N در مبنای ۱۰ به صورت زیر تفسیر می‌شود

$$N = (a_{n-1}\, a_{n-2} \ldots a_2\, a_1\, a_0)_{10} =$$
$$a_0 \times 10^0 + a_1 \times 10^1 + a_2 \times 10^2 + \ldots a_{n-1} \times 10^{n-1}$$

- مثال: عدد ۳۴۸۲ بصورت زیر تفسیر می گردد :

$$(3482)_{10} = 2 \times 10^0 + 8 \times 10^1 + 4 \times 10^2 + 3 \times 10^3$$

- در مبنای ۱۰ (سیستم دهدهی) نیاز به ۱۰ رقم (از ۰ تا ۹) داریم

مبنای دلخواه

- اعداد را در هر مبنای دلخواه دیگری می‌توانیم نشان دهیم
- عدد N در مبنای b به صورت زیر تفسیر می‌شود:

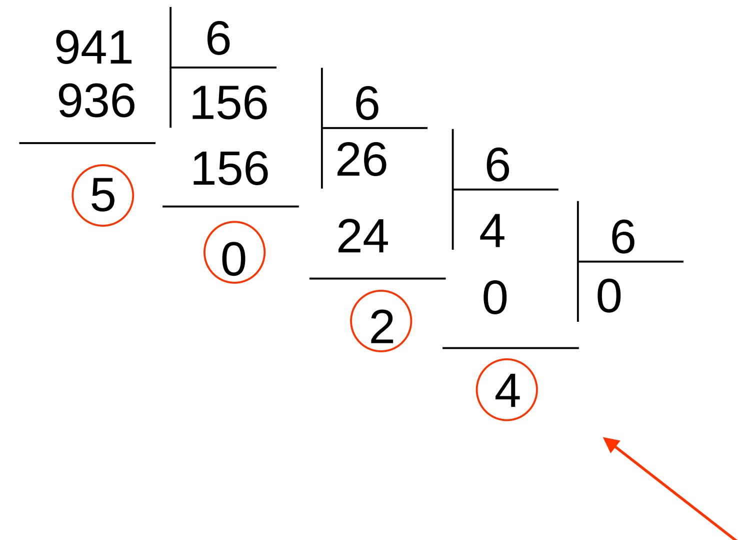
$$N = (a_{n-1}\ a_{n-2}\ \ldots\ a_2\ a_1\ a_0\ )_b =$$
$$a_0 \times b^0 + a_1 \times b^1 + a_2 \times b^2 + \ldots\ a_{n-1} \times b^{n-1}$$

- در مبنای b نیاز به b رقم (از ۰ تا b-1) داریم
- مثلاً یک عدد در مبنای ۶ از ارقام ۰..۵ تشکیل می‌شود
  - $(341)_6$ یک عدد معتبر است
  - اما $(592)_6$ نامعتبر است

تبدیل مبنا

- برای تبدیل یک عدد از مبنای ۱۰ به هر مبنای دلخواه b:
- از روش تقسیمات متوالی استفاده می گردد

$$(941)_{10} = (?)_6$$



$$\mathbf{(941)_{10} = (4205)_6}$$

تبدیل مبناها

- برای تبدیل از مبنای b به مبنای ۱۰ :
  - ارقام عدد مورد نظر را در ارزش مکانی آنها ضرب و سپس با یکدیگر جمع می‌کنیم

$$(4205)_6 = (?)_{10}$$

$$(4205)_6 = 5 \times 6^0 + 0 \times 6^1 + 2 \times 6^2 + 4 \times 6^3 = 5 + 0 + 72 + 864 = (941)_{10}$$

اهمیت مبنای ۲

- مبنای ۲ اهمیت بسیار زیادی در کامپیوترهای دیجیتال دارد
- در مبنای ۲ تنها به ۲ رقم نیاز داریم: صفر و یک
- کامپیوترهای دیجیتال هم اطلاعات را به صورت باینری نگهداری می‌کنند
  - واحد حافظه: بایت
  - هر بایت = ۸ بیت
  - هر بیت: یک خانه حافظه که یک «صفر یا یک» را نگهداری می‌کند
- بنابراین مبنای ۲ اهمیت خاصی دارد

Base 2
Binary

- مبنای ۲ = سیستم دودویی = سیستم باینری

تبدیل از مبنای ۲

- تبدیل از مبنای ۲ به ۱۰

$$(11001001)_2 = (?)_{10}$$

$$(11001001)_2 = 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$$

$$= 1 + 0 + 0 + 8 + 0 + 0 + 64 + 128 = (201)_{10}$$

تبدیل اعداد باینری

- تبدیل از مبنای ۱۰ به مبنای ۲

$$(486)_{10} = (?)_2$$

486	2	243	486	0
243	2	121	242	1
121	2	60	120	1
60	2	30	60	0
30	2	15	30	0
15	2	7	14	1
7	2	3	6	1
3	2	1	2	1
1	2	0	0	1

$$(486)_{10} = (111100110)_2$$

اعداد باینری

- هر رقم دودویی یک بیت(bit) نامیده می‌شود
- در سیستم اعداد دودویی:
  - به کم‌ارزش‌ترین بیت LSB (Least Significant Bit) گفته می‌شود
  - پرارزش‌ترین بیت: MSB (Most Significant Bit)

$$(486)_{10} = (111100110)_2$$

MSB LSB

مبناهای ۸ و ۱۶ و کاربرد آن‌ها

- مشکل اصلی مبنای ۲ : اندازه بزرگ اعداد است
- مثلاً عدد ۴۸۶
  - در مبنای ۱۰ تنها ۳ رقم دارد، ولی تبدیل به یک عدد ۹ رقمی در مبنای ۲ می‌شود
- مشاهده و محاسبه در مبنای ۲ برای انسان‌ها مشکل می‌شود
- مبنای ۸ و مبنای ۱۶، مورد توجه هستند
  - تعداد ارقام اعداد در این مبناها کمتر است
  - تبدیل اعداد باینری به این دو مبنا و برعکس بسیار ساده است
- هر سه رقم در مبنای ۲، برابر است با یک رقم در مبنای ۸ و بالعکس
- هر چهار رقم در مبنای ۲، برابر است با یک رقم در مبنای ۱۶ و بالعکس

Base 8
Octal
Oct

Base 16
Hexadecimal
Hex

تبدیل باینری به اکتال و برعکس

- تبدیل از مبنای ۲ به ۸

$$\underbrace{1\ 0}_{2}\ \underbrace{1\ 0\ 1}_{5}\ \underbrace{1\ 1\ 0}_{6} = (256)_8$$

- تبدیل از مبنای ۸ به ۲

$$(2\ 7\ 1)_8 = (10111001)_2$$

$$2 \to 010 \qquad 7 \to 111 \qquad 1 \to 001$$

تبدیل مبنای ۱۶ (هگزادسیمال)

- در این مبنا به ۱۶ رقم داریم (حروف A تا F : ارقام ۱۰ تا ۱۵)

0 1 2 3 4 5 6 7 8 9 A B C D E F

- تبدیل از مبنای ۲ به ۱۶

$$\underbrace{1\,1\,0}_{6}\;\underbrace{1\,0\,1\,1}_{B}\;\underbrace{1\,0\,0\,1}_{9} = (6B9)_{16}$$

- تبدیل از مبنای ۱۶ به ۲

$$(A\;3\;E)_{16} = (101000111110)_2$$

$A \rightarrow 1010 \quad 3 \rightarrow 0011 \quad E \rightarrow 1110$

خلاصه کاربرد مبناها

- مبنای ۱۰ : به صورت عادی توسط انسان‌ها استفاده می‌شود
- مبنای ۲ : برای ذخیره‌سازی اعداد در کامپیوتر
- مبنای ۸ و ۱۶ : نمایش ساده‌تر اعداد باینری برای انسان‌ها
  - مبنای ۸ و ۱۶ برای ذخیره اعداد در کامپیوتر استفاده نمی‌شود
- مبنای ۱۶ از مبنای ۸ پرکاربردتر است
  - هر بایت با دو عدد هگزادسیمال نمایش داده می‌شود
  - هر چهار بیت (نیم بایت) یک نیبل (nibble) خوانده می‌شود

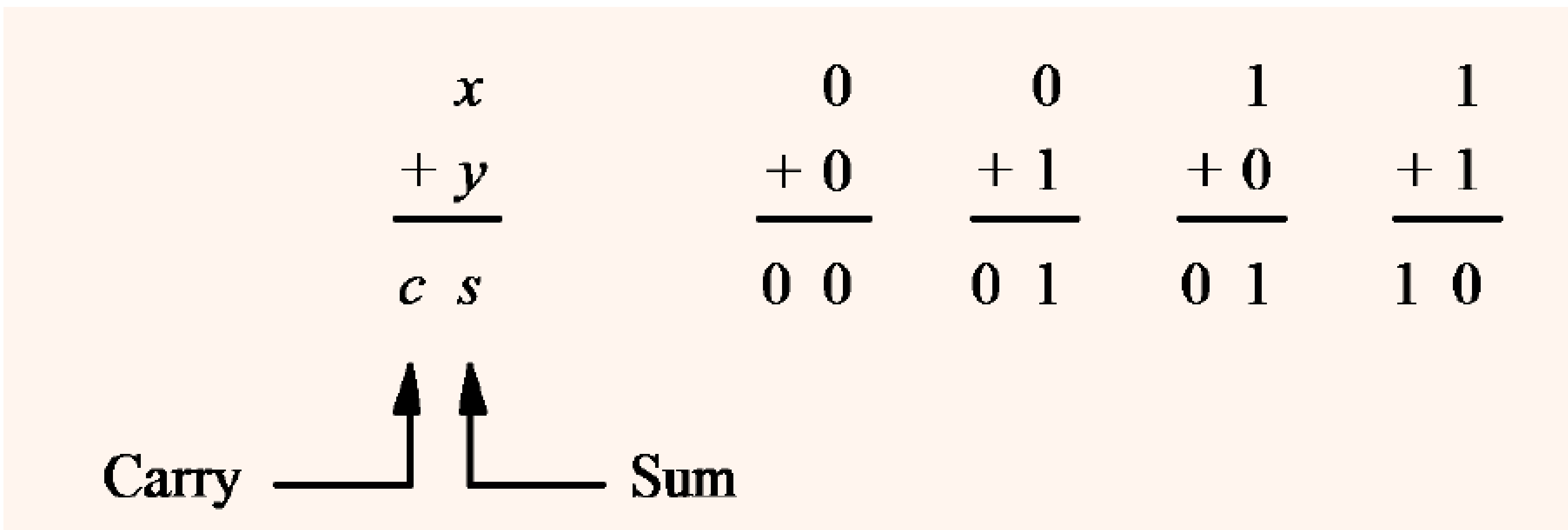
جمع اعداد بدون علامت

جمع ارقام بدون علامت

- جمع در مبنای دو:

تنها از دو رقم صفر و یک استفاده می‌شود

- پس چهار حالت ممکن است:



جمع اعداد بدون علامت (مثال)

$$
\begin{array}{rl}
X= & x_4x_3x_2x_1x_0 \\
+Y= & y_4y_3y_2y_1y_0 \\
\hline
S= & s_4s_3s_2s_1s_0
\end{array}
$$

رقم‌های نقلی (carry bits)

$$
\begin{array}{r}
\color{red}{1\ 1\ 1} \\
0\ 1\ 1\ 1\ 1 \\
+0\ 1\ 0\ 1\ 0 \\
\hline
1\ 1\ 0\ 0\ 1
\end{array}
$$

$$
\begin{array}{r}
\color{red}{1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1} \\
0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
+0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\
\hline
1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0
\end{array}
$$

سرریز (Overflow)

- برای نمایش اعداد به حافظه‌ای که در نظر گرفته شده، محدود هستیم
- در صورتی که حاصل عملیات (جمع) در فضای در نظر گرفته نگنجد:
  گفته می شود «سرریز» یا Overflow رخ داده است
- مثلاً فرض کنید یک بایت برای اعداد در نظر گرفته باشیم
- این بیت اضافه قابل ذخیره نخواهد بود
  - (در یک بایت جا نمی‌شود)
- عملاً «جمع‌کننده» چنین خواهد کرد: ۲۵۵+۱=۰

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

کوییز

- عدد π - را به صورت باینری در دو بایت نشان دهید: ۳.۱۴۱۵۹-

- روش دقیق را هنوز ندیده‌ایم
  - الان مهم این است که به حل این مسأله درست فکر کنید (شاید به پاسخ دقیق نرسید)

π=11.001001000011111101101010100010001000010110100011

مشکل: اعداد علامت‌دار و اعداد اعشاری

- به نظر شما اعداد اعشاری را چگونه نگه داریم؟
- به نظر شما اعداد منفی را چگونه نگه داریم؟
  - ایده ممیز ثابت
  - ایده بیت علامت
- روش‌های مختلفی برای این موضوع وجود دارد
  - با آن‌ها آشنا خواهیم شد

Signed & Unsigned Numbers

Integer & Real Numbers

اعداد علامت‌دار

استفاده از بیت علامت

- در این روش:
  سمت‌چپ‌ترین بیت برای علامت عدد درنظر گرفته می‌شود
  سایر بیت‌ها مقدار عدد رانشان می‌دهند
- صفر بودن بیت علامت به معنای مثبت بودن و یک بودن آن به معنای منفی بودن عدد است
- با داشتن ۸ بیت می توان چه اعدادی را نمایش داد؟
  - اعداد بین 127+ ... 127- را نمایش داد

0	1010011

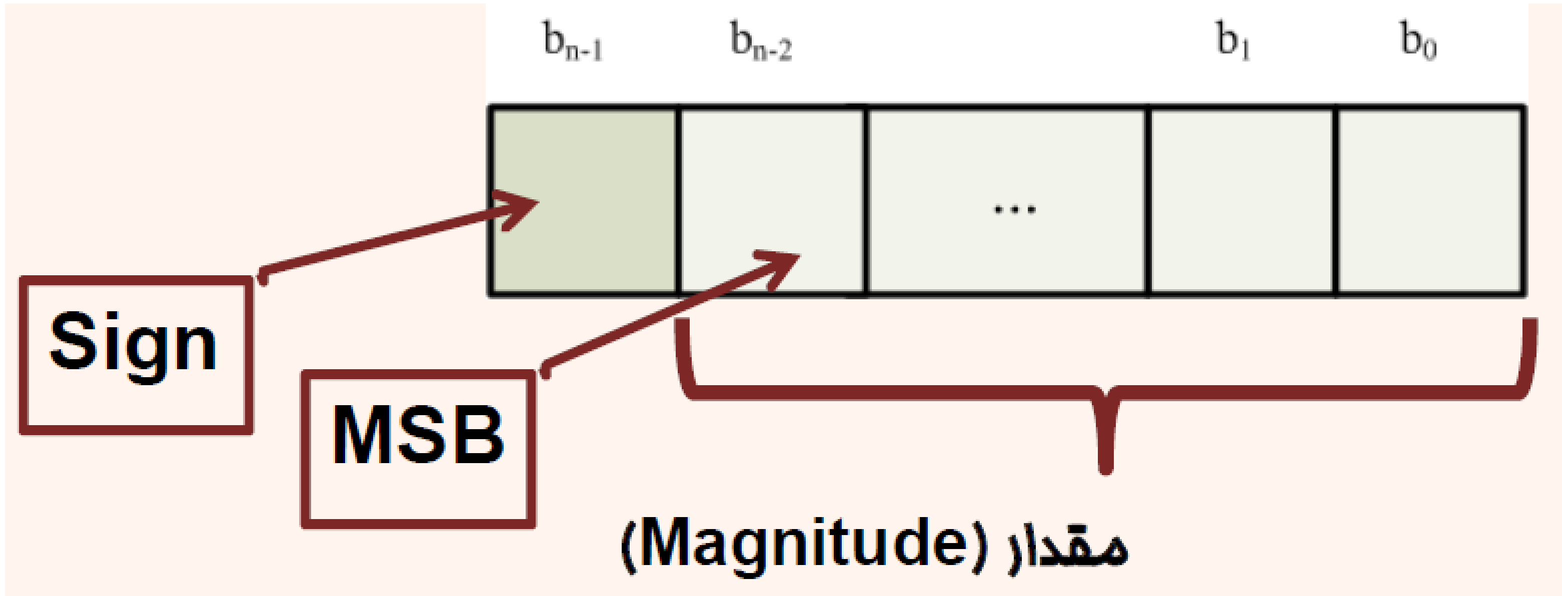
+83

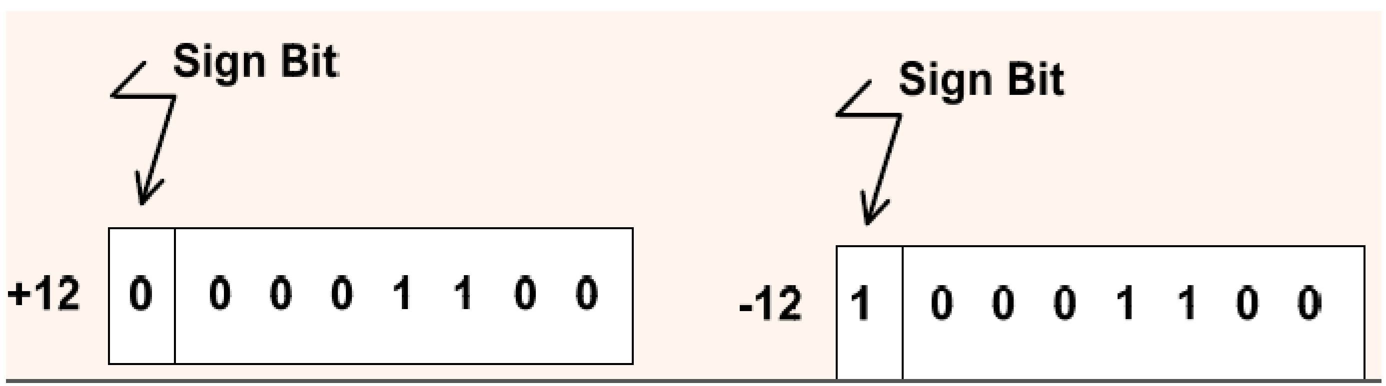
1	1010011

-83

- این روش دو مشکل اصلی دارد:
  - دو مقدار متفاوت 0+ و 0- وجود دارد
  - عمل جمع روی اعداد منفی نیاز به مدار (روش) جداگانه دارد

سیستم علامت و مقدار (استفاده از بیت علامت)





سیستم عددی پیش قدردار (Biased representation)

- در این شیوه اعداد به صورت «جمع شده با عددی دیگر (bias)» فرض می‌شوند
  - برای تعیین معنای واقعی یک عدد، bias را از آن کم می‌کنیم
- مثلاً اگر فرض کنیم bias=128
  - عدد باینری صفر ، ۱۲۸- را نشان می‌دهد
  - عدد ۱۲۸، به معنای صفر و عدد ۲۵۵، به معنای ۱۲۷ است
- اگر n بیت فضا داشته باشیم، معمولاً مقدار bias برابر است با $K=2^{(n-1)}$
  - چرا؟
  - بازه اعداد ممکن در n بیت فضا؟
    - $\mathbf{-2^{(n-1)}...2^{(n-1)}-1}$

Biased representation
Offset binary
excess-K

مشکلات سیستم پیش‌قدردار

- مشکلات؟
- به ازای هر عملیات جمع یا تفریق نیاز به تصحیح نتیجه خواهیم داشت:
  - مثال (می‌دانیم به جای مقدار x ، در واقع مقدار x+biais را ذخیره می‌کنیم) :
- (x + bias) + (y + bias) = x + y + 2*bias
  - برای تصحیح حاصل‌جمع، باید از نتیجه یک bias تفریق کنیم
- (x + bias) – (y + bias) = x – y
  - حاصل تفریق صحیح: x – y + bias

سیستم مکمل ۱

- اعداد مثبت به صورت معمولی نمایش داده می‌شوند
- نمایش اعداد منفی: قدر مطلق آن را در نظر بگیرید و سپس همه بیت‌ها را برعکس کنید
  - (کلیه صفرها را به یک و یک‌ها را به صفر تبدیل کنید)

0 1 0 1 0 0 1 1	1 0 1 0 1 1 0 0
+ 83	- 83

- برای جلوگیری از تداخل اعداد مثبت و منفی:
- فقط برای اعداد منفی باید بیت سمت چپ یک باشد
- بازه اعداد ممکن در یک بایت: از 127- تا 127+
- یکی از مشکلات این روش: دو نمایش مختلف برای 0+ و 0- وجود دارد

00000000	11111111
+0	-0

مکمل ۱

- در سیستم مکمل ۱ در واقع عدد منفی چطور محاسبه می‌شود؟ (n تعداد بیت‌ها)
- $K = (2^n - 1) - P$

1 0 0 0	→	-7	0 1 1 1	→	7
1 0 0 1	→	-6	0 1 1 0	→	6
1 0 1 0	→	-5	0 1 0 1	→	5
1 0 1 1	→	-4	0 1 0 0	→	4
1 1 0 0	→	-3	0 0 1 1	→	3
1 1 0 1	→	-2	0 0 1 0	→	2
1 1 1 0	→	-1	0 0 0 1	→	1
1 1 1 1	→	0^-	0 0 0 0	→	0^+

عملیات جمع در سیستم مکمل ۱

$$\begin{array}{r} (+5) \\ +\ (+2) \\ \hline (+7) \end{array} \qquad \begin{array}{r} 0101 \\ +\ 0010 \\ \hline 0111 \end{array}$$

$$\begin{array}{r} (-5) \\ +\ (+2) \\ \hline (-3) \end{array} \qquad \begin{array}{r} 1010 \\ +\ 0010 \\ \hline 1100 \end{array}$$

$$\begin{array}{r} (+5) \\ +\ (-2) \\ \hline (+3) \end{array} \qquad \begin{array}{r} 0101 \\ +\ 1101 \\ \hline 10010 \\ \rightarrow 1 \\ \hline 0011 \end{array}$$

$$\begin{array}{r} (-5) \\ +\ (-2) \\ \hline (-7) \end{array} \qquad \begin{array}{r} 1010 \\ +\ 1101 \\ \hline 10111 \\ \rightarrow 1 \\ \hline 1000 \end{array}$$

- در سیستم مکمل ۱، نیازمند اِعمال End-around carry هستیم
  - در مکمل دو نیازی به این کار نیست

جمع در سیستم مکمل ۱

- در این سیستم تنها نیاز به یک روش برای جمع اعداد مثبت یا منفی است
  - روش یا الگوریتم یا مدار یکسان

- امکان سرریز (Overflow)

$$\begin{array}{r} (-5) \\ +\ (-5) \\ \hline \end{array} \qquad \begin{array}{r} 1010 \\ +\ 1010 \\ \hline 10100 \\ 1 \\ \hline 0101 \end{array}$$

سیستم مکمل ۲

- اعداد مثبت مثل قبل هستند. برای اعداد منفی:
- نحوه محاسبه مکمل۲ برای اعداد منفی؟
- $K = 2^n - P$
- روش اول محاسبه: مکمل۱ را محاسبه کنیم و نتیجه را با یک جمع کنیم
- روش دوم (سریع‌تر) برای محاسبه مکمل ۲ :
  - از سمت راست شروع می‌کنیم
  - بیت هایی که برابر با صفر هستند را نادیده می‌گیریم تا به اولین مقدار یک برسیم
  - (این مقدار یک را هم نادیده می‌گیریم تا دست‌نخورده باقی بماند)
  - پس از آن بیت‌ها را برعکس می‌کنیم

مکمل ۲

- بازه اعداد مجاز در این روش از ۱۲۸- تا ۱۲۷+ است (به ازای یک بایت حافظه)

01011010	→	10100101	→	10100110
+90				-90
		مکمل ۱		مکمل ۲

مکمل ۲

0 1 1 1 →	7
0 1 1 0 →	6
0 1 0 1 →	5
0 1 0 0 →	4
0 0 1 1 →	3
0 0 1 0 →	2
0 0 0 1 →	1

1 0 0 1 →	-7
1 0 1 0 →	-6
1 0 1 1 →	-5
1 1 0 0 →	-4
1 1 0 1 →	-3
1 1 1 0 →	-2
1 1 1 1 →	-1

عملیات جمع در سیستم مکمل ۲

$$\begin{array}{rr} (-5) & 1\,0\,1\,1 \\ +\ (+2) & +\ \ 0\,0\,1\,0 \\ \hline (-3) & 1\,1\,0\,1 \end{array}$$

$$\begin{array}{rr} (+5) & 0\,1\,0\,1 \\ +\ (+2) & +\ \ 0\,0\,1\,0 \\ \hline (+7) & 0\,1\,1\,1 \end{array}$$

$$\begin{array}{rr} (-5) & 1\,0\,1\,1 \\ +\ (-2) & +\ \ 1\,1\,1\,0 \\ \hline (-7) & 1\ 1\,0\,0\,1 \end{array}$$

ignore

$$\begin{array}{rr} (+5) & 0\,1\,0\,1 \\ +\ (-2) & +\ \ 1\,1\,1\,0 \\ \hline (+3) & 1\ 0\,0\,1\,1 \end{array}$$

ignore

سرریز

- شرط سرریز:

رقم نقلی وارد به بیت آخر با رقم نقلی خارج‌شده از آن برابر نباشد

- مثلاً هر دو مثبت بوده‌اند، ولی حاصل‌جمع منفی شده، یا هر دو منفی بوده‌اند و حاصل‌جمع مثبت شده

$$
\begin{array}{rr}
(+7) & 0\,1\,1\,1 \\
+\,(+2) & +\;0\,0\,1\,0 \\
\hline
\boxed{(+9)} & \boxed{1\,0\,0\,1}
\end{array}
$$

$c_4=0$

$c_3=1$

مزایای استفاده از مکمل ۲

- تنها یک نمایش برای صفر داریم (این سیستم 0+ و 0- ندارد)

-0	-1	+0
00000000	11111111	00000000

- برای عمل جمع اعداد مثبت و منفی از یک روش (مدار) یکسان استفاده می‌شود

53 – 22 = 53 + (-22) = 31

(00110101) – (00010110) = (00110101) + (11101010) = 1(00011111)

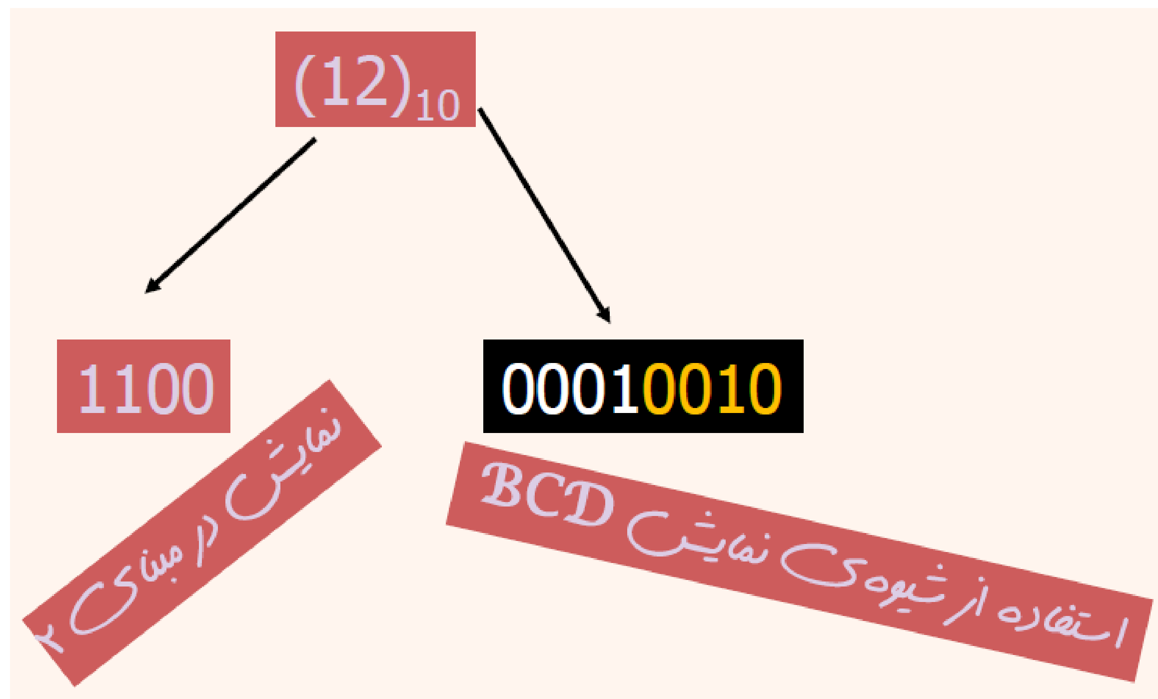
38 – 60 = 38 + (-60) = -22

(00100110) – (00111100) = (00100110) + (11000100) = (11101010)

روش BCD

- در برخی کاربردها دسترسی به ارقام ده‌دهی و تبدیل آن‌ها به مبنای دو به صورت مجزا مهم است
- روش BCD (**Binary-Coded-Decimal**)
- در این نمایش هر رقم ده‌دهی با چهار بیت نمایش داده می‌شود

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



جمع‌بندی: روش‌های نگهداری اعداد صحیح

- سیستم BCD
- سیستم علامت و مقدار (بیت علامت)
- سیستم پیش‌قدر دار
- مکمل یک
- مکمل دو

- روش مکمل۲ روشی معمول و رایج‌تر از سایر روش‌ها است (مزایای آن را دیدیم)

مثال: تفسیر یک عدد در سیستم‌های مختلف

Sequence	Two's complement	One's complement	Signed-magnitude
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0
1111	−1	−0	−7
1110	−2	−1	−6
1101	−3	−2	−5
1100	−4	−3	−4
1011	−5	−4	−3
1010	−6	−5	−2
1001	−7	−6	−1
1000	−8	−7	−0

نکته درباره سیستم‌های مکمل

نامگذاری مکمل یک و مکمل دو از کجا آمده است؟

- در هر مبنایی مثل r، دو نوع مکمل وجود دارد:
  - مکمل مبنا (Radix Complement)

$$[N]_r = r^n - (N)_r$$

  - مکمل مبنای کاهش‌یافته (Diminished Radix Complement)

$$[N]_r - 1$$

- در مبنای دو (سیستم باینری) نیز دو نوع مکمل وجود دارد
  - مکمل دوم یا همان مکمل مبنا (Two's complement)
  - مکمل اول یا مکمل مبنای کاهش‌یافته (Ones' complement)

توضیح

- در سیستم ده‌دهی، مکمل کاهش‌یافته‌ی هر عدد «مکمل ۹» آن خواهد بود
  - که با تفریق هر رقم از ۹ حاصل می‌شود
- در سیستم باینری، مکمل کاهش‌یافته هر عدد «مکمل یک» آن است

Some examples of 9's complement are;

Decimal Number	9's Complement
59	40
894	105
6578	3421
2063	7936

توضیح

- Radix Complement = r's complement
- Diminished Radix Complement = (r-1)'s complement
  - مکمل کاهش یافته عدد x: عددی که اگر با x جمع شود، $r^n - 1$ حاصل شود
    - مبنا: r
    - تعداد بیت‌ها : n
  - مقدار $r^n - 1$: مثلاً تمام‌یک در مبنای ۲ و تمام۹ در مبنای ۱۰
- Binary (r=2)➔ ones' complement
- Decimal (r=10) ➔ nines' complement.

مطالعه و بررسی بیشتر

- درباره مطالب زیر جستجو، مطالعه و فکر کنید:
- چرا روش سریعتری که برای محاسبه مکمل۲ بیان شد، صحیح است؟
  - از سمت راست شروع می‌کنیم، بیت هایی که برابر با صفر هستند را نادیده می‌گیریم تا به اولین مقدار یک برسیم، پس از آن بیت‌ها را برعکس می‌کنیم
- چرا در روش مکمل۱، باید از End-around carry استفاده کرد؟
- روش BCD چه کاربردهایی دارد؟ چگونه می‌توان آن را بهینه‌تر کرد؟

نمایش اعداد حقیقی

نمایش باینری اعداد اعشاری

- روش تفسیر یک عدد اعشاری در مبنای r

$$x_{k-1}x_{k-2}\ldots x_1x_0 \,.\, x_{-1}x_{-2}\ldots x_{-l}\;) = \sum_{i=-l}^{k-1} x_i r^i$$

- مثال:
  - $(1101.101)_2 = (?)_{10}$

$$
\begin{aligned}
1101.101_2 &= 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}\\
&= 1\times 8 + 1\times 4 + 0\times 2 + 1\times 1 + 1\times 0.5 + 0\times 0.25 + 1\times 0.125\\
&= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125\\
&= 13.625_{10}
\end{aligned}
$$

نحوه تبدیل بخش اعشاری به باینری

- $(0.125)_{10} = (?)_2$
- رقم‌های باینری:

A	B	C	D	E	F

- $0.125 = A/2 + B/4 + C/8 + \ldots$
- $\boxed{0}.25 = A + B/2 + C/4 + \ldots \rightarrow \underline{A=0}$
- $\boxed{0}.5 = B + C/2 + .. \rightarrow \underline{B=0}$
- $\boxed{1}.0 = C + .. \rightarrow \underline{C=1}$
- $$\mathbf{(0.125)_{10} = (0.001)_2}$$

مثال دیگر از نحوه تبدیل اعداد اعشاری به باینری

- $(0.83)_{10} = (?)_2$
  - $0.83 * 2 = \boxed{1}.66$
  - $0.66 * 2 = \boxed{1}.32$
  - $0.32 * 2 = \boxed{0}.64$
  - $0.64 * 2 = \boxed{1}.28$
  - …

- نمایش دقیق برای این عدد وجود ندارد

- $(0.83)_{10} = (0.11010100011110101110000\ldots)_2$

مشکل نمایش باینری اعداد اعشاری

- فرض کنید حافظه محدودی را برای ذخیره یک عدد اعشاری در نظر گرفته باشیم
  - مثلاً یک بایت، یا چهار بایت، یا هشت بایت
- تعداد اعداد اعشاری بینهایت است
  - حتی اگر بازه اعداد اعشاری را محدود کنیم
- برای ذخیره اعداد حقیقی (اعشاری) دقت کامل ممکن نیست
- بنابراین برای ذخیره یا نمایش کامپیوتری هر عدد حقیقی:
  - نزدیک‌ترین عدد قابل نمایش به عدد موردنظر درنظر گرفته می‌شود

سیستم ممیز ثابت

- در این شیوه جایگاه ممیز، ثابت است
- مثلاً هنگام ذخیره عدد در دو بایت:
  - یک بایت را برای بخش صحیح و یک بایت را برای بخش اعشاری در نظر بگیریم
- برای اعداد علامت‌دار هم از هر یک از روش‌هایی که دیدیم، می‌توانیم استفاده کنیم
  - سیستم علامت و مقدار، مکمل ۲ یا ...

سیستم ممیز ثابت: مثال

- عدد 13.625 را در یک بایت جا دهید
  - فرض: یک بیت علامت، چهار بیت برای بخش صحیح و سه بیت برای بخش اعشاری
- پاسخ:
  - بیت علامت : 0
  - چهار بیت صحیح : 1101
  - سه بیت اعشاری: 101

0	1	1	0	1	1	0	1

6 **D**

دقت (precision) در ممیز ثابت

- دقت در ممیز ثابت: وابسته به تعداد بیت‌هایی که به بخش اعشاری اختصاص یافته
- اگر بخواهیم دقت اعشاری را افزایش دهیم:

  (فرض: فضای ثابتی داریم، مثلاً یک بایت)

  - فضای مربوط به بخش اعشاری را بزرگ‌تر می‌کنیم (بیت‌های بیشتر)
  - پس فضای مربوط به بخش صحیح کوچک‌تر می‌شود
  - بازه‌ی اعداد قابل نمایش کاهش می‌یابد
- با افزایش بازه‌ی اعداد: کاهش دقت

مشکل سیستم ممیز ثابت

- در کاربردهای علمی استفاده از اعداد بسیار بزرگ و یا اعداد بسیار کوچک لازم است
- با استفاده از سیستم «ممیز ثابت» امکان همزمان برای این دو وجود ندارد:
  ۱- نمایش بازه‌ی گسترده اعداد
  ۲- دقت بالا وجود ندارد
- برای چنین کاربردهایی سیستم « ممیز شناور» پیشنهاد شده است

سیستم ممیز شناور

- اعداد به شکل «نماد علمی نرمال» به نمایش داده می‌شوند
- در مبنای دو، نماد علمی نرمال عددی به این شکل است: $\pm 1.F \times 2^{E}$
- مثال:

- $+1.0011 * 2^5$

- در این شیوه، باید این اطلاعات را به نحوی نگهداری کنیم:
  علامت (sign)، ضریب (coefficient) و توان (exponent)
- شیوه‌های مختلفی برای نگهداری این مقادیر ارائه شده است
- **استاندارد IEEE 754** برای یکسان‌سازی این فرایند، یک شیوه ارائه کرد
  - این استاندارد، برای نمایش اعداد حقیقی رایج شده است

استاندارد IEEE 754 برای اعداد ممیز شناور

- IEEE Standard 754 (Floating Point Numbers)
  - قالب نگهداری اعداد حقیقی را مشخص کرده است
  - دو شیوه‌ی زیر در این استاندارد ارائه شده است:
  - دقت معمولی (single-precision)
    - در این شیوه از ۳۲ بیت (چهار بایت) برای نگهداری عدد حقیقی استفاده می‌شود
  - دقت مضاعف (double precision)
    - در این شیوه از ۶۴ بیت (هشت بایت) استفاده می‌شود

روش IEEE 754

- نمایش **توان** (exponent) : از شیوه‌ی پیش قدردار استفاده می‌کند
- برای **علامت** (sign) : یک بیت در نظر گرفته می‌شود (شیوه بیت علامت)
- برای **ضریب** (coefficient یا mantissa): فقط بخش اعشاری نگهداری می‌شود
  - بخش صحیح (عدد ۱) اصلاً نگهداری نمی‌شود: «یکِ پنهان» (hidden one)
  - روش باینری معمولی برای نمایش این بخش استفاده می‌شود

$$X = (-1)^{\boxed{S}} \times (1 + \boxed{\text{Fraction}}) \times 2^{(\boxed{\text{Exponent}} - \text{Bias})}$$

مرور دو شیوه‌ی استاندارد IEEE 754

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

S	Exponent	Fraction

$$X = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Single: Bias = 127; Double: Bias = 1023

دقت معمولی (single precision)

IEEE 754 Floating Point Standard

s	e=exponent	m=mantissa
1 bit	8 bits	23 bits

$$\text{number} = (-1)^s * (1.m) * 2^{e-127}$$

مثال (single precision)

- عدد ۴۸۶ را بصورت دودویی اعشاری نشان دهید
- ابتدا آن را به مبنای ۲ تبدیل می کنیم:

$$(486)_{10} = (111100110)_2$$

- اکنون داریم :

$$111100110 = 1.11100110 \times 2^8$$

s : 0 e : 8+127 = 135 = 10000111 m : 11100110

این عدد به صورت دقیق ذخیره می‌شود

- جواب نهایی:

0	10000111	11100110000000000000000

مثال بعدی (single precision)

- عدد 121.640625- را به صورت دودویی اعشاری نشان دهید

- ابتدا عدد را به مبنای ۲ می بریم:

$$(121.640625)_{10} = (1111001.101001)_2 = (1.111001101001 \times 2^6)_2$$

$s : 1 \quad e : 6 + 127 = 133 = 10000011 \quad m : 111001101001$

- پاسخ نهایی:

این عدد به صورت دقیق ذخیره می‌شود

1	10000011	11100110100100000000000

مثال بعدی (single precision)

- عدد $+1.1$

$$(1.1)_{10} = (1.\,00011001100110011001101\ldots)_2$$
$$= (1.\,00011001100110011001101\ldots \times 2^0)_2$$

$$\mathrm{s} : 0 \quad \mathbf{e} : 0 + 127 = 127 = 01111111 \quad \mathbf{m} : 00011001100110011001101\ \ldots$$

این عدد به صورت **<u>تقریبی</u>** ذخیره می‌شود

- پاسخ نهایی:

0 01111111 00011001100110011001100

تمرین

- $(41A00000)_{IEEE754(single)} = (\ ?)_b = (?)_d$

تمرین

$$(41A00000)_{IEEE754(single)}=(\ ?)_b=(?)_d$$

4	1	A	0	0	0	0	0

0 10000011 01000000000000000000000

S E+Bias F

1 = negative
0 = positive

131 $\xrightarrow{-127}$ E 4

$(0.01)_2$ → $(1.01)_2$ M=1.F

$$=(\ 1.01\times 2^4)_b=(10100)_b=(20)_d$$

تمرین

- $(C17B0000)_{IEEE754(single)}=(\ ?)_b=(?)_d$

تمرین

$$(C17B0000)_{IEEE754(single)}=(\ ?)_b=(?)_d$$

C	1	7	B	0	0	0	0

$$\underbrace{1}_{S}\ \underbrace{10000010}_{E+Bias}\ \underbrace{11110110000000000000000}_{F}$$

1 = negative
0 = positive

E+Bias → 130

$130 \xrightarrow{-127} 3$

E = 3

F → $(0.1111011)_2$ → $(1.1111011)_2$

M=1.F

$$=(\ -1.1111011\times2^3)_b=(-1111.1011)_b=(-15.6875)_d$$

بازه اعداد در استاندارد IEEE 754

level	width	range at full precision
single precision	32 bits	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
double precision	64 bits	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$

s	e=exponent	m=mantissa

نمایش مقادیر خاص اعشاری

- اگر $0<e<255$ آن‌گاه $\text{عدد} = (-1)^s \times (1.m) \times 2^{e-127}$
  - اما اگر e برابر با صفر یا ۲۵۵ باشد، تفسیر دیگری خواهیم داشت
- اگر $e=255$ و $m=0$ آن‌گاه عدد بینهایت است
  - (بسته به مقدار s مثبت یا منفی بینهایت)
- اگر $e=255$ و m غیرصفر باشد آن‌گاه حاصل یک عدد نیست (NaN یا Not a Number)
- اگر $e=0$ باشد آن‌گاه $\text{عدد} = (-1)^s \times (0.m) \times 2^{-126}$
  - که به آن عدد غیرنرمال (unnormalized) گفته می‌شود (hidden one ندارد)
  - نکته: در این حالت مقدار bias را ۱۲۶ گرفته‌اند (دلیل ظریفی دارد، که بگذریم)
  - **به خصوص اگر $e=0$ و $m=0$ آن‌گاه $\text{عدد} = 0$**
    - که البته بسته به میزان s مقدار آن 0+ یا 0- خواهد بود

استاندارد IEEE 754

Name	Common name	Base
binary16	Half precision	2
binary32	Single precision	2
binary64	Double precision	2
binary128	Quadruple precision	2
decimal32		10
decimal64		10
decimal128		10

- آن‌چه ما دیدیم
و معمولاً در نرم‌افزارها و سخت‌افزارها استفاده می‌شود،
بخشی از استاندارد IEEE 754 است
که تحت عنوان
single precision و double precision
شناخته می‌شوند

IEEE 754-2008

برخی از سخت‌افزارهای جدید هم آن را پشتیبانی می‌کنند (مثل IBM Z10)

نمایش کاراکترها و متن

نمایش متن

- متن (text) رشته‌ای از کاراکترها است (String)
- هر کاراکتر را با یک عدد نشان می‌دهیم
- متن (رشته) را با مجموعه‌ای از اعداد پشت‌سرهم نشان می‌دهیم
- برخلاف سایر داده‌ها (مثل عدد صحیح یا عدد حقیقی معمولی)، حافظه لازم برای نگهداری یک متن ثابت نیست (variable-length encodings)
- در ادامه روش‌های نگهداری یک کاراکتر را می‌بینیم

نحوه نمایش کاراکتر: روش کد اسکی (ASCII Code)

- ASCII: American Standard Code for Information Interchange

- این روش، یک کاراکتر را در یک بایت نمایش می‌دهد
- مجموعاً ۲۵۶ کاراکتر را پشتیبانی می‌کند
  - در واقع ۱۲۸ کاراکتر اول در اسکی تعیین شده است (۷ بیت)
- کاراکترهای اسکی:
  - ارقام (0 تا 9)
  - حروف کوچک و بزرگ انگلیسی
  - علائم و کاراکترهای کنترلی

جدول اَسکی

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td>NUL</td><td>DLE</td><td>space</td><td>0</td><td>@</td><td>P</td><td>`</td><td>p</td></tr>
<tr><td>1</td><td>SOH</td><td>DC1
XON</td><td>!</td><td>1</td><td>A</td><td>Q</td><td>a</td><td>q</td></tr>
<tr><td>2</td><td>STX</td><td>DC2</td><td>"</td><td>2</td><td>B</td><td>R</td><td>b</td><td>r</td></tr>
<tr><td>3</td><td>ETX</td><td>DC3
XOFF</td><td>#</td><td>3</td><td>C</td><td>S</td><td>c</td><td>s</td></tr>
<tr><td>4</td><td>EOT</td><td>DC4</td><td>$</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td></tr>
<tr><td>5</td><td>ENQ</td><td>NAK</td><td>%</td><td>5</td><td>E</td><td>U</td><td>e</td><td>u</td></tr>
<tr><td>6</td><td>ACK</td><td>SYN</td><td>&</td><td>6</td><td>F</td><td>V</td><td>f</td><td>v</td></tr>
<tr><td>7</td><td>BEL</td><td>ETB</td><td>'</td><td>7</td><td>G</td><td>W</td><td>g</td><td>w</td></tr>
<tr><td>8</td><td>BS</td><td>CAN</td><td>(</td><td>8</td><td>H</td><td>X</td><td>h</td><td>x</td></tr>
<tr><td>9</td><td>HT</td><td>EM</td><td>)</td><td>9</td><td>I</td><td>Y</td><td>i</td><td>y</td></tr>
<tr><td>A</td><td>LF</td><td>SUB</td><td>*</td><td>:</td><td>J</td><td>Z</td><td>j</td><td>z</td></tr>
<tr><td>B</td><td>VT</td><td>ESC</td><td>+</td><td>;</td><td>K</td><td>[</td><td>k</td><td>{</td></tr>
<tr><td>C</td><td>FF</td><td>FS</td><td>,</td><td><</td><td>L</td><td>\</td><td>l</td><td>|</td></tr>
<tr><td>D</td><td>CR</td><td>GS</td><td>-</td><td>=</td><td>M</td><td>]</td><td>m</td><td>}</td></tr>
<tr><td>E</td><td>SO</td><td>RS</td><td>.</td><td>></td><td>N</td><td>^</td><td>n</td><td>~</td></tr>
<tr><td>F</td><td>SI</td><td>US</td><td>/</td><td>?</td><td>O</td><td>_</td><td>o</td><td>del</td></tr>
</table>

استاندارد یونیکد (Unicode)

- تعداد کاراکترهای بیشتری را پوشش می‌دهد
  - از جمله کاراکترهای فارسی، ژاپنی و …
  - ۱۲۸ کاراکتر اول آن با ACII مطابق است
- در یک بایت جا نمی‌شود
  - کاراکترها را در یک تا چهار بایت جا می‌دهد
  - روش‌های UTF-8 ، UTF-16 و UTF-32
- بیشتر بخوانید:
  - یونیکد یک Character Set است
  - UTF-8 یک Encoding است

جمع‌بندی

جمع‌بندی

- اعداد در مبناهای مختلف (مبنای ۱۰، ۲ ، ۸ و ۱۶)
  - اهمیت و جایگاه مبناهای ممختلف
- نمایش باینری اعداد
- نمایش اعداد علامت‌دار
- نمایش اعداد اعشاری
  - استاندارد IEEE 754

مطالعه بیشتر

- اسلایدهای بسیار خوب دکتر محمودی
- http://faculties.sbu.ac.ir/~a_mahmoudi/ITP_93_1.htm
- صفحات ویکیپدیا
  - مثل: https://en.wikipedia.org/wiki/IEEE_floating_point
- هر آن چه می‌خواهید جستجو کنید. مثال:
  - Two's complement
  - Hexadecimal
  - ...

پایان