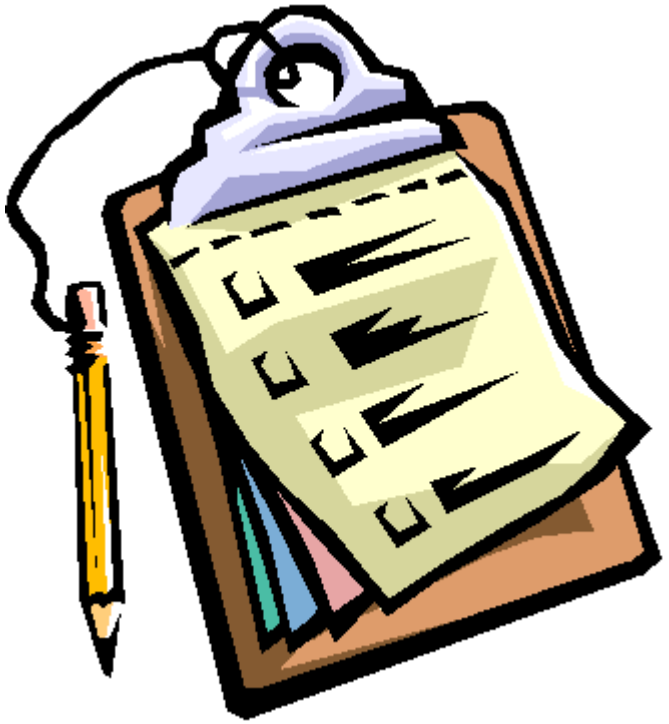
درس «مبانی کامپیوتر و برنامه‌سازی»

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 ذخیره اطلاعات در کامپیوتر

- اطلاعات در کامپیوتر به شکل‌های مختلفی ذخیره می‌شود. مثل؟
  - 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:

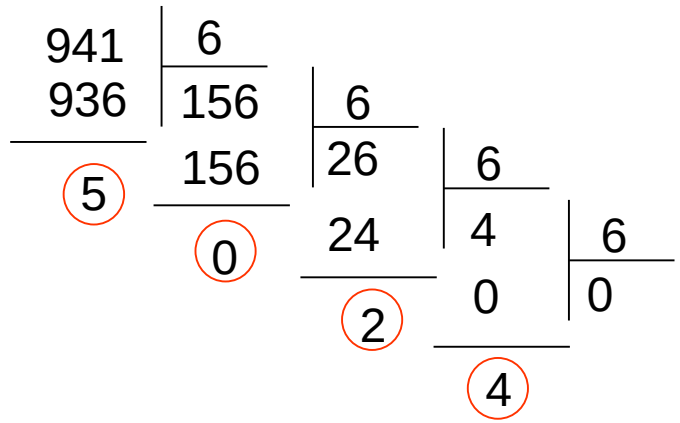
$$N = (a_{n-1}\ a_{n-2}\ \dots\ a_2\ a_1\ a_0\ )_b =$$
$$a_0 \times b^0 + a_1 \times b^1 + a_2 \times b^2 + \dots\ a_{n-1} \times b^{n-1}$$

- در مبنای b نیاز به b رقم (از ۰ تا b-1) داریم
- مثلاً یک عدد در مبنای ۶ از ارقام ۵..۰ 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

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



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

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

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

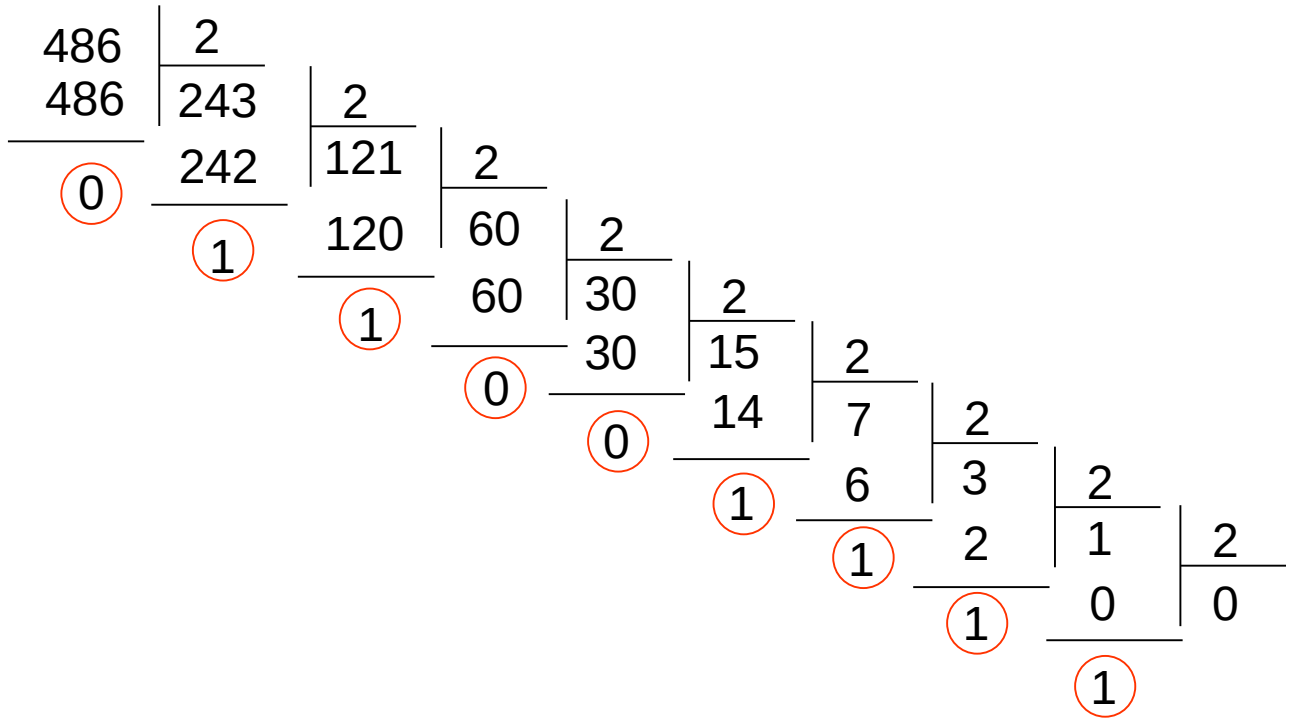
$$(11001001)_2 = 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$$

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

تبدیل اعداد باینری

- تبدیل از مبنای ۱۰ به مبنای ۲

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



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

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

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

010 111 001

تبدیل مبنای ۱۶ (هگزادسیمال)

- در این مبنا به ۱۶ رقم داریم (حروف A تا F : ارقام ۰ 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

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

1010 0011 1110

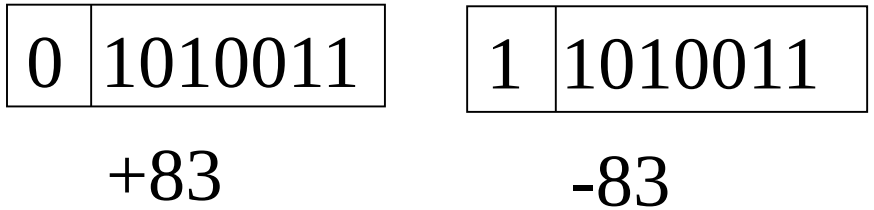
خلاصه کاربرد مبناها

- 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

اعداد علامت‌دار

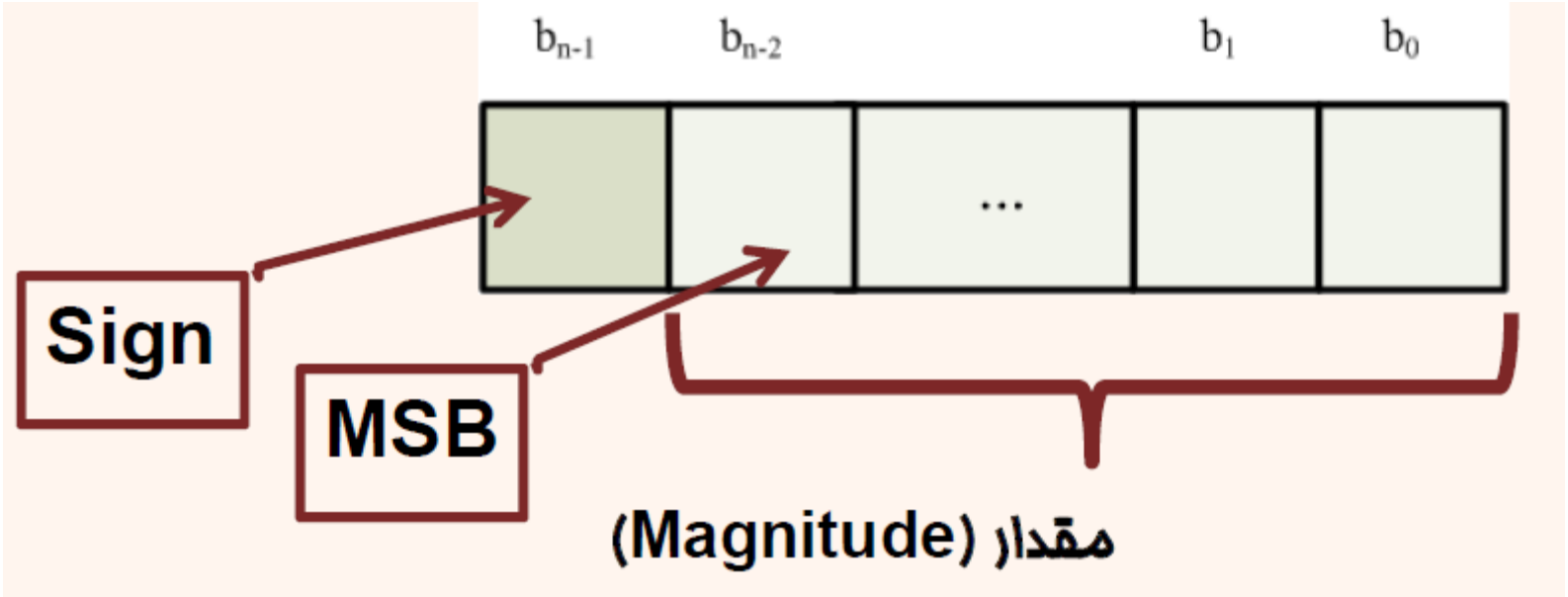
استفاده از بیت علامت

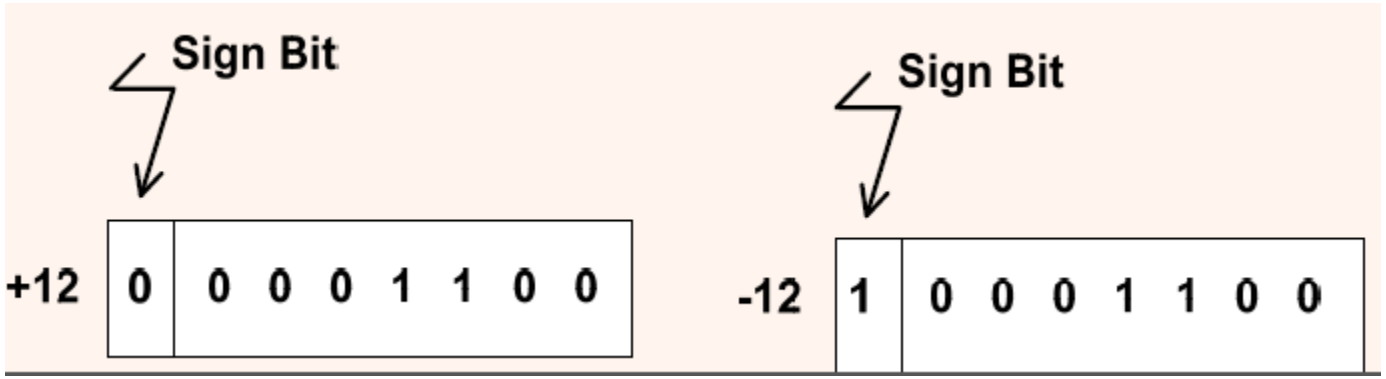
- در این روش:
  سمت‌چپ‌ترین بیت برای علامت عدد درنظر گرفته می‌شود
  سایر بیت‌ها مقدار عدد رانشان می‌دهند
- صفر بودن بیت علامت به معنای مثبت بودن و یک بودن آن به معنای منفی بودن عدد است
- با داشتن ۸ بیت می توان چه اعدادی را نمایش داد؟
  - اعداد بین +127 ... -127 را نمایش داد



- 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 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)

1 0 1 0 1 1 0 0	0 1 0 1 0 0 1 1
- 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

11111111	00000000
-0	+0

مکمل ۱

- در سیستم مکمل۱ 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

(+5)	0101
+ (+2)	+ 0010
(+7)	0111

(–5)	1010
+ (+2)	+ 0010
(–3)	1100

(+5)	0101
+ (–2)	+ 1101
(+3)	10010
	→ 1
	0011

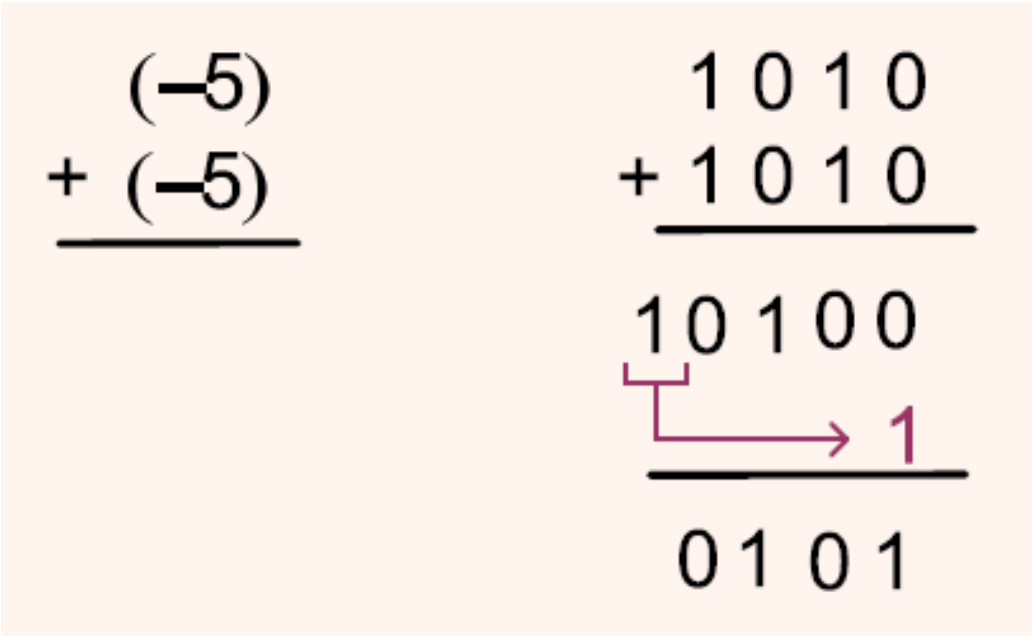
(–5)	1010
+ (–2)	+ 1101
(–7)	10111
	→ 1
	1000

- 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)

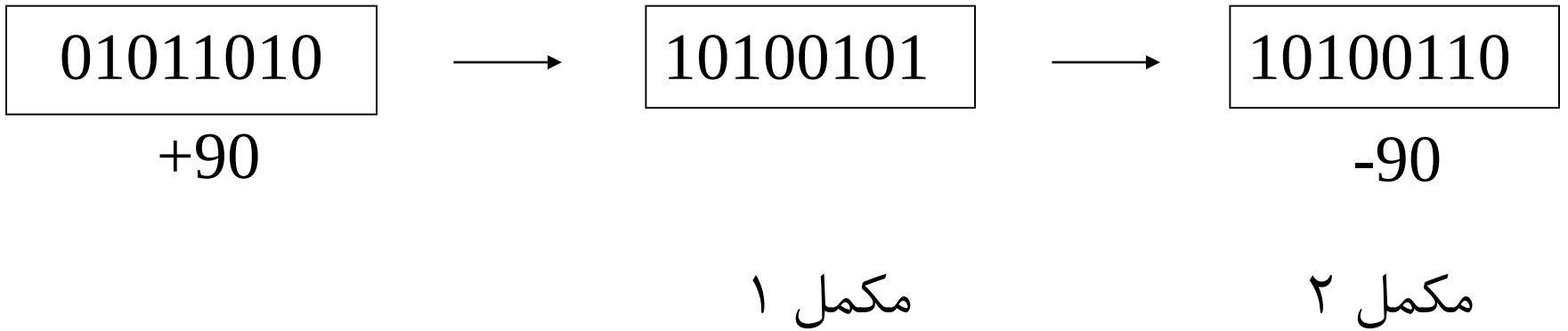


سیستم مکمل ۲

- 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)



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

(+5)	0101
+ (+2)	+ 0010
(+7)	0111

(–5)	1011
+ (+2)	+ 0010
(–3)	1101

(+5)	0101
+ (–2)	+ 1110
(+3)	10011

ignore

(–5)	1011
+ (–2)	+ 1110
(–7)	11001

ignore

سرریز (Overflow)

- 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

(+7)	0 1 1 1
+ (+2)	+ 0 0 1 0
(+9)	1 0 0 1

$c_4=0$

$c_3=1$

مزایای استفاده از مکمل ۲

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

00000000	11111111	00000000
+0	-1	-0

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

53 – 22 = 53 + (-22) = 31

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

38 – 60 = 38 + (-60) = -22

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

کوییز

۱- عملیات زیر را در سیستم مکمل ۲ انجام دهید:

(فرض کنید سیستم، ۸ بیتی است. یعنی هر عدد صحیح در یک بایت ذخیره می‌شود)

- -39 + 92

۲- حاصل‌جمع را در مبنای ۱۶ نشان دهید.

پاسخ: ۹۲+۳۹– در مبنای ۱۶ = ۳۵

- ۳۹ ← ۱۰۰۱۱۱ (یک عدد ۶ بیتی)
  - در سیستم ۸ بیتی: دو صفر در سمت چپ (پرارزش‌ترین بیت) قرار می‌دهیم: ۰۰۱۰۰۱۱۱
- ۳۹- ← مکمل ۲ : ۱۱۰۱۱۰۰۱
- عدد ۹۲ ← ۱۰۱۱۱۰۰
  - سیستم ۸ بیتی: ۰۱۰۱۱۱۰۰
- جمع بیت به بیت: ۰۰۱۱۰۱۰۱
  - رقم نقلی وارد بر پرارزش‌ترین بیت و رقم نقلی خارج شده از آن برابر هستند (هر دو: یک) پس سرریز رخ نداده
- تبدیل به مبنای ۱۶ : ۳۵

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

- $0.125 = A/2 + B/4 + C/8 + \ldots$
- $0.25 = A + B/2 + C/4 + \ldots \rightarrow \underline{A=0}$
- $0.5 = B + C/2 + .. \rightarrow \underline{B=0}$
- $1.0 = C + .. \rightarrow \underline{C=1}$
- $$(0.125)_{10} = (0.001)_2$$

مثال دیگر از نحوه تبدیل اعداد اعشاری به باینری

- $(0.83)_{10} = (?)_2$
  - $0.83 * 2 = 1.66$
  - $0.66 * 2 = 1.32$
  - $0.32 * 2 = 0.64$
  - $0.64 * 2 = 1.28$
  - ...
- نمایش دقیق برای این عدد وجود ندارد
- $(0.83)_{10} = (0.11010100011110101011100 0...)_2$

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

مشکل سیستم ممیز ثابت

- تعداد ارقام اصلی و ارقام اعشاری ثابت است
- دقت اعشاری و همچنین اندازه عدد موردنظر، محدود و مشخص خواهند بود

- پیشنهاد: استفاده از سیستم ممیز شناور (floating point)
  - سیستم کارآمدتری است
  - هم می‌تواند اعداد بسیار بزرگ را نشان دهد
  - و هم اعداد بسیار کوچک اعشاری (با دقت فراوان)

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

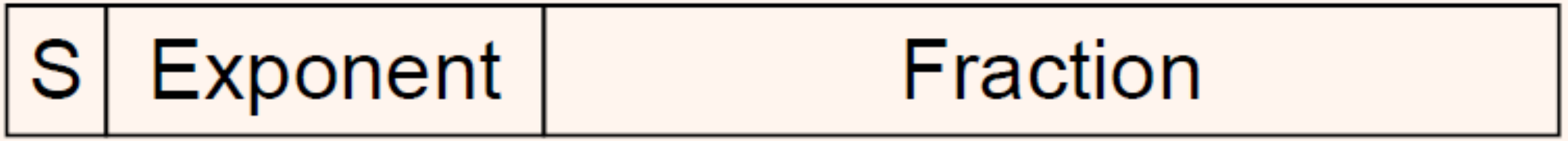
- نمایش **توان** (exponent) : از شیوه‌ی پیش قدردار استفاده می‌کند
- برای **علامت** (sign) : یک بیت در نظر گرفته می‌شود (شیوه بیت علامت)
- برای **ضریب** (coefficient یا mantissa): فقط بخش اعشاری نگهداری می‌شود
  - بخش صحیح (عدد ۱) اصلاً نگهداری نمی‌شود: «یکِ پنهان» (hidden one)
  - روش باینری معمولی برای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

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

مرور دو شیوه‌ی استاندارد IEEE 754

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits



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

Single: Bias = 127; Double: Bias = 1023

دقت معمولی (single precision)

IEEE 754 Floating Point Standard

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

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

- جواب نهایی:

این عدد به صورت دقیق ذخیره می‌شود

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 e : 6 + 127 = 133 = 10000011 m : 111001101001

- پاسخ نهایی:

این عدد به صورت دقیق ذخیره می‌شود

1	10000011	11100110100100000000000

مثال بعدی (single precision)

- عدد 1.1+

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

s : 0 **e** : 0 + 127 = 127 = 01111111 **m** : 00011001100110011001101 ...

این عدد به صورت **تقریبی** ذخیره می‌شود

- پاسخ نهایی:

0 01111111 00011001100110011001100

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 $\pm 1.80 \times 10^{308}$

نمایش کاراکترها و متن

نمایش متن

- متن (text) رشته‌ای از کاراکترها است (String)
- برای ذخیره متن در کامپیوتر، هر کاراکتر را با یک عدد نشان می‌دهیم
  - به جای متن (رشته) ، مجموعه‌ای از اعداد را پشت‌سرهم ذخیره می‌کنیم
- 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 (ASCII)

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

ASCII		
Dec	Hex	Result
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII		
Dec	Hex	Result
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

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 مختلف
- نمایش باینری اعداد
- نمایش اعداد علامت‌دار
- نمایش اعداد اعشاری
  - استاندارد IEEE 754

مطالعه بیشتر

- مجموعه اسلایدهای کامل‌تر همین مبحث (اسلایدهای حاضر خلاصه شده بودند)
- اسلایدهای 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
- درباره موضوع این درس جستجو کنید. مثال:
  - Two's complement
  - Hexadecimal
  - ...

پایان