دانشگاه صنعتی خواجه نصیرالدین طوسی

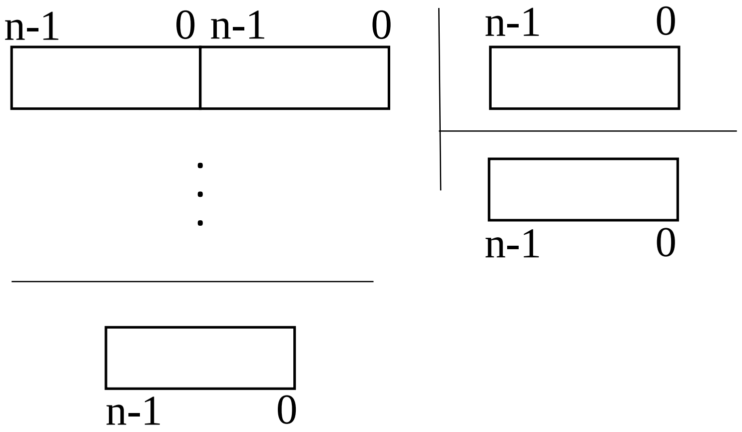
بنام خداوند جان و خرد

معماری کامپیوتر

جلسه شانزدهم: تقسیم‌کننده-محاسبات اعشاری

تقسیم‌کننده (Divider)

- حاصل تقسیم یک عدد 2n بیتی بر عددی n بیتی به n بیت فضا نیاز دارد
- روش تقسیم:
  - از مقسوم n بیت جدا می‌کنیم اگر بزرگتر از مقسوم‌علیه بود
    - یک در خارج قسمت می‌گذاریم
  - درغیراینصورت
    - صفر گذاشته و روال را برای n+1 بیت تکرار می‌کنیم



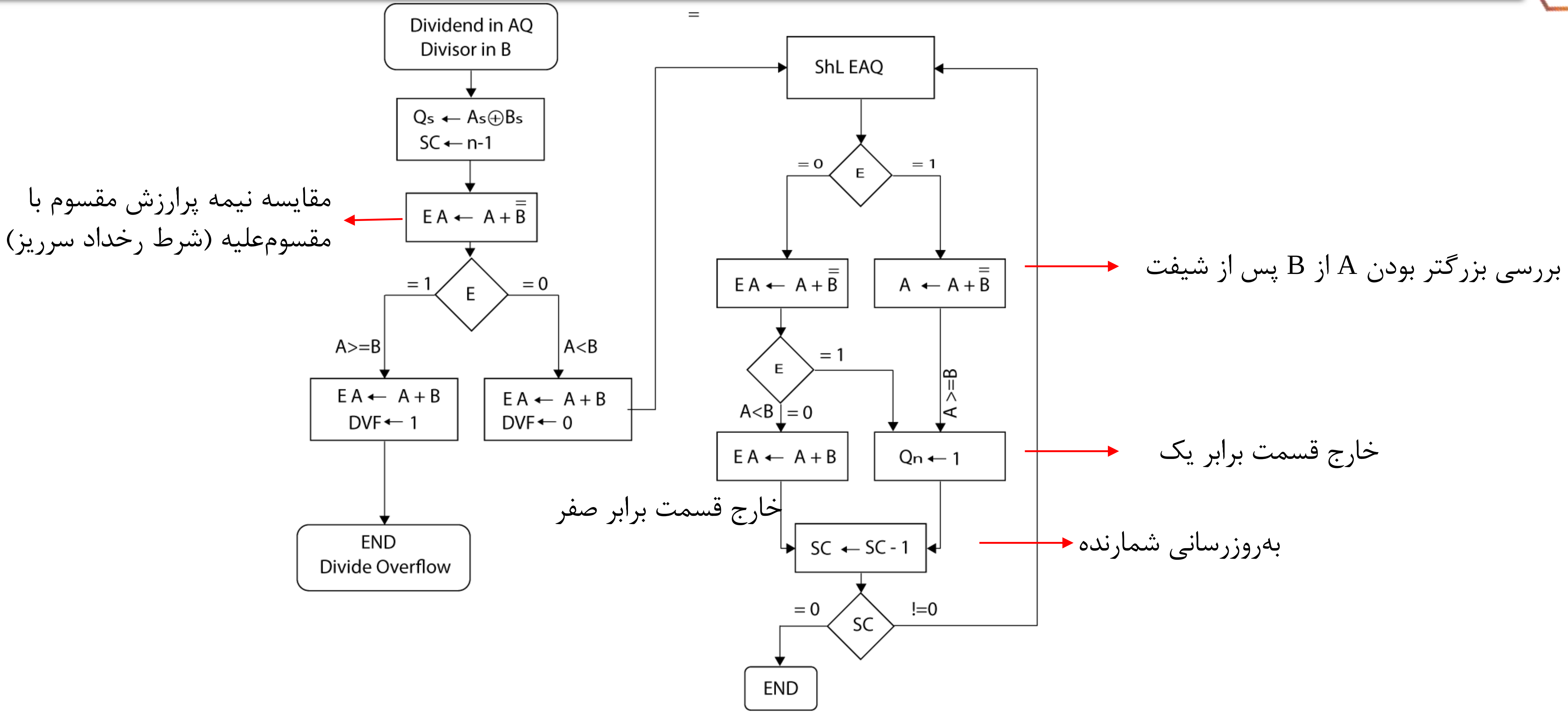
تقسیم‌کننده (Divider)

- گاهی بر حسب شرایط n بیت فضا برای خارج قسمت تقسیم کم است
  - مانند تقسیم 11...11 بر 1...000
  - شرایط سرریز (overflow) در تقسیم
    - مقسوم‌علیه برابر صفر باشد که حاصل برابر بی‌نهایت شده و در n بیت نمی‌گنجد
    - اگر n بیت پرارزش مقسوم از مقسوم‌علیه بزرگتر باشد، خارج قسمت در n بیت جا نشده و سرریز رخ می‌دهد
      - مانند مثال بالا
  - شرایط یک و دو سرریز در تقسیم را می‌توان در حالت کلی مورد دوم خلاصه نمود
  - سرریز باید مدیریت شود و در پیاده‌سازی سخت‌افزاری لحاظ گردد

الگوریتم تقسیم‌کننده (Divider)

- الگوریتم برای دو عدد صحیح بدون علامت
  - مقسوم (A.Q)، مقسوم‌علیه (B)، خارج قسمت (Q)، باقی‌مانده (A)
  - چک کردن رخداد سرریز:
    - اگر A>B: سرریز رخ می‌دهد قادر به انجام تقسیم نیستیم
    - اگر A<B: رقم اول خارج قسمت صفر است و یک بیت از Q را به A اضافه می‌کنیم
      - در خارج قسمت 1 قرار داده و B را از A کم می‌کنیم
      - عملیات را تا انتها براساس شمارنده‌ای که برابر n تنظیم شده ادامه می‌دهیم

الگوریتم تقسیم‌کننده (Divider)



الگوریتم تقسیم‌کننده (Divider)

مثال: دو عدد روبرو را توسط الگوریتم divider بر یکدیگر تقسیم کنید.

X = 10011000
Y = 1100

حل:

Sc = 4 : AQ = 10011000, B = 1100

EA = A+B'+1 = 1001 + 0011+ 1 = 01101

E = 0 → Overflow = 0 → A = A + B = 1101 + 1100 = 1001

EAQ = 010011000 →SL → EAQ = 100110000

E=1 → A = EA+B'+1 = 10011 + 0011 + 1 = 0111 → Q0 = 1

Sc = 3: AQ = 01110001, B = 1100

EA = A+B'+1 = 0111 + 0011 + 1 = 01011

E = 0 → Overflow = 0 → A = A + B = 1011 + 1100 = 0111

EAQ = 001110001 → SL → EAQ = 011100010

E= 0 → EA = A+B'+1 = 1110 + 0011 + 1 = 10010 →E = 1 → Q0 = 1

الگوریتم تقسیم‌کننده (Divider)

Sc = 2: AQ = 00100011, B = 1100

EA = A+B’+1 = 0010 + 0011 + 1 = 00110

E =0 → Overflow =0 → A = A + B = 0110 + 1100 = 0010

EAQ = 000100011→ SL → 001000110

E = 0 → EA = A+B’+1 = 0100 + 0011 + 1 = 01000→E=0 → Q0 = 0

EA = A+ B = 1000+ 1100 = 10100

Sc =1: AQ = 01000110, B = 1100

EA = A+B’+1 = 0100 + 0011 + 1 = 01000

E = 0 → Overflow = 0 → A = A + B = 1000 + 1100 = 0100

EAQ = 001000110 → SL → 010001100

E = 0 → EA = A+B’+1 = 1000+ 0011 + 1 = 01100 →E = 0 → Q0 = 0

EA = A + B = 1100 + 1100 = 11000

Sc = 0 → **Finish , Q = 1100 , A = 1000**

تقسیم‌کننده (Divider) علامت‌دار

- تا اینجا فرض کردیم دو عدد مثبت را بر یکدیگر تقسیم می‌کنیم
- اگر اعداد از نوع صحیح و علامت دار باشند:
  - ابتدا مشخص می‌کنیم شیوه نمایش چگونه است (مکمل ۲ یا اندازه علامت)
  - اگر اعداد مثبت باشند که همان روال قبلی را عینا تکرار می‌کنیم
  - اگر یکی یا هردو اعداد منفی بود
    - معادل مثبت آن را به‌دست آورده و عملیات تقسیم را مشابه حالت گفته شده انجام می‌دهیم
    - در انتها تعیین علامت کرده و مقادیر را به‌روزرسانی می‌کنیم

محاسبات برای اعداد اعشاری

- برای ذخیره اعداد اعشاری دو دیدگاه وجود دارد
  - اعداد اعشاری ممیز ثابت (Fixed Point Numbers)
    - پیاده‌سازی ساده
    - استفاده غیربهینه از فضای ذخیره‌سازی
  - اعداد اعشاری ممیز شناور (Floating Point Numbers)
    - پیاده‌سازی پیچیده
    - استفاده بهینه و منعطف از فضای ذخیره‌سازی

محاسبات برای اعداد اعشاری

- ممیز ثابت (Fixed Point):
  - تعداد بیت‌های تخصیص داده شده به قسمت صحیح و اعشاری ثابت است
    - بخش صحیح به‌صورت مکمل ۲
    - بخش اعشاری به‌صورت بدون علامت
    - منجر به خطا در محاسبات
    - استفاده ناکارامد از فضای ذخیره‌سازی
      - عدم توانایی ذخیره‌سازی برخی اعداد
      - خالی ماندن فضا در بیشتر موارد

قسمت صحیح	قسمت اعشاری
K بیت	t بیت

01101100

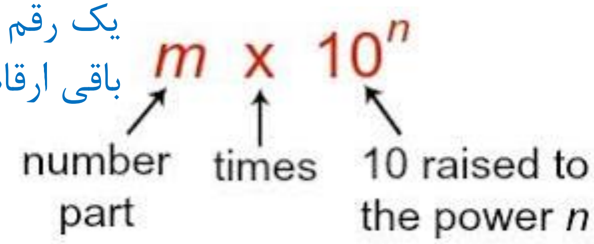
0110.1100

$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$

محاسبات برای اعداد اعشاری

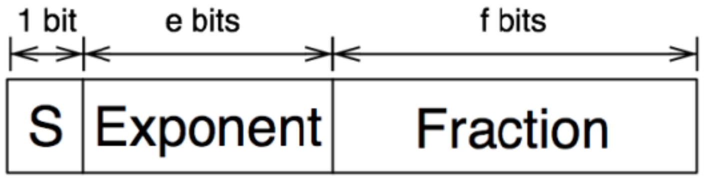
- ممیز شناور (Floating Point):
  - باهدف رفع چالش‌های نمایش ممیزثابت ارائه شده است
    - محل نقطه اعشار ثابت نیست
    - نگهداری اعداد در حافظه به‌صورت بهینه
    - کاهش خطای محاسبات نسبت به حالت نمایش ممیز ثابت
  - ایده اصلی نمایش، نگهداری اعداد به شکل علمی آن‌هاست

یک رقم صحیح
باقی ارقام اعشاری



اعداد اعشاری ممیز شناور

- فرمت نمایش اعداد ممیز شناور



$$\rightarrow \mathit{Floating\ point\ number} = (-1)^S * 1.F * 2^E$$

- **S**: علامت عدد (مثبت:۰ و منفی: ۱)
- **Exponent**: نمای توانی ذخیره شده به‌صورت مکمل ۲
- **Fraction** (مانتیس): قسمت اعشاری عدد نرمال شده
  - عدد نرمال شده در قسمت صحیح تنها یک رقم دارد
  - در نمایش باینری به‌غیر از عدد صفر، بخش صحیح همواره یک است پس نیاز به ذخیره ندارد

اعداد اعشاری ممیز شناور

- شیوه نمایش اعداد در قالب ممیز شناور:
  - هنجارسازی (نرمال‌سازی) اعداد به‌صورت ممیزشناور
  - ذخیره‌سازی در ده بیت (۵ بیت مانتیس و ۴ بیت نما)

$0101.111 \rightarrow 1.\underbrace{01111}_{\text{Fraction/Mantis}} * 2^{\overset{\text{Exponent}}{2}}$

$11111 \rightarrow 1.1111 * 2^4$: S = 0, E = 0100, F = 11110

$-100.0001 \rightarrow -1.000001 * 2^2$: S = 1, E = 0010, F = 00000

$0.10101 \rightarrow 1.0101 * 2^{-1}$: S = 0, E = 1111, F = 01010

$-0.00111 \rightarrow -1.11 * 2^{-3}$: S = 1, E = 1101, F = 11000

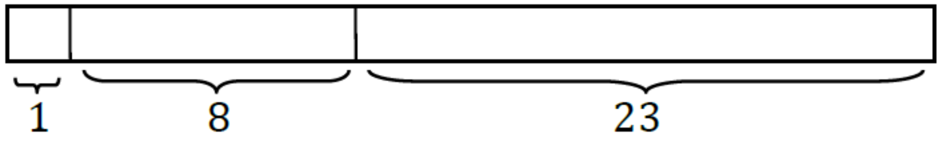
اعداد اعشاری ممیز شناور

- روال تبدیل و ذخیره اعداد به فرمت ممیز شناور:
  - عدد را هنجار (نرمال) می‌کنیم
    - همه اعداد به‌جز صفر را می‌توان هنجار کرد (نمایش صفر با کوچکترین عدد قابل نمایش)
  - نما و اعشار بدست آمده را در قالب نمایش جایگذاری می‌کنیم
    - قسمت اعشاری (مانتیس) همیشه مثبت است و به همان شکل در حافظه قرار می‌گیرد
    - قسمت نما می‌تواند مثبت یا منفی بوده و نمایش مکمل دو دارد
    - در ذخیره این دو بخش تعداد بیت تخصیص داده شده مهم است
      - رخداد سرریز یا زیرریز در نما

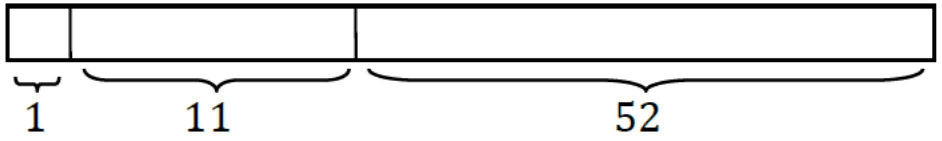
اعداد اعشاری ممیز شناور

- سایز هر بخش در فرمت ذخیره‌سازی مهم است
  - استاندارد IEEE 754 برای تنظیم فضای e و f

Single Precision(دقت ساده): 32 bits (Exponent 8 bits, Fraction 23 bits)



Double Precision(دقت مضاعف):64 bits (Exponent 11 bits, Fraction 52 bits)



اعداد اعشاری ممیز شناور

- ویژگی‌های اعداد در نمایش ممیز شناور:

$(-1)^{S}1.F*2^{e}$

- تعداد اعداد قابل نمایش
  - محدود بودن بازه نمایشی استاندارد، تعداد اعداد را معین می‌کند
- بازه اعداد قابل نمایش
  - محدوده اعداد و حداقل و حداکثر عدد مثبت قابل نمایش را معین می‌کند
- دقت محاسبات
- نمایش صفر: $(-1)^{0}1.0*2^{Emin}$

بهبود نمایش اعداد اعشاری ممیز شناور

- مقایسه دو عدد اعشاری ممیز شناور
  - مقایسه توان‌ها که در قالب مکمل دو ذخیره شده‌اند
    - برای سهولت در مقایسه آن‌ها را با آفست 2^{e-1} جمع می‌کنیم تا مثبت شود (عملیات bias)
    - با این تغییر ترتیب اعداد ثابت مانده و مقایسه ساده می‌شود

4 + مکمل‌دو	مکمل دو	عدد
111	011	۳
110	010	۲
101	001	۱
100	000	۰
011	111	-۱
010	110	-۲
001	101	-۳
000	100	-۴

آفست: قدر مطلق بزرگترین عدد قابل نمایش در نما

نمایش اعداد خاص در سیستم ممیز شناور

- اعداد پراستفاده مانند e، π، Nan(not a number)و ...
- ذخیره در حالت خاص با هدف افزایش دقت
- در سیستم نمایش ممیز شناور نماها را با $2^{e-1}-x$ جمع می‌کنیم
  - x مکان را برای ذخیره‌سازی اعداد خاص با دقت مناسب رزرو می‌کنیم
  - مثال: x=1 و نماها سه‌بیتی باشند (بایاس = جمع با سه)
    - محدوده نمایش نما را به ۳- تا ۳+ محدود کرده و حالت ۴- (۱۱۱) را برای نمایش اعداد خاص درنظر می‌گیریم
    - ذخیره کد اعداد خاص در جدول راهنما

نمایش اعداد در سیستم ممیز شناور

- مقدار بایاس در هر سیستم نمایش برحسب تعریف مشخص است
- استخراج عدد ممیز شناور بایاس شده:

$(-1)^{S}1.F*2^{e-bias}$

- تعداد اعداد اعشاری در بازه صفر تا یک بی‌نهایت است پس قادر به نمایش همه اعداد نیستیم

- **قرارداد نمایش**
  - Bias #1 = 2^{e-1}
  - Bias #2: $2^{e-1}-1$: رزرو محدوده یک نما برای اعداد خاص

نمایش اعداد در سیستم ممیز شناور

- فرض کنیم طول بخش‌های fraction و نما برابر f و e بیت باشند:

- **حداقل مقدار اعشار(F_{min})** : صفر (تمامی ارقام برابر صفر) به‌دلیل بی‌علامت بودن این بخش
- **حداکثر مقدار اعشار(F_{max}):** تمامی ارقام اعشار برابر یک ($1-2^{-f}$)
- **حداقل مقدار نما(E_{min}):** باتوجه به علامت‌دار بودن این بخش برابر -2^{e-1}
- **حداکثر مقدار نما(E_{max}):** باتوجه به علامت‌دار بودن این بخش برابر $2^{e-1}-1$

نمایش اعداد در سیستم ممیز شناور

- کمترین مقدار مثبت قابل ذخیره در سیستم ممیز شناور:

$$\varepsilon = N_{min} = 1.F_{min} * 2^{E_{min}} \rightarrow 1.0 * 2^{-2^{e-1}}$$

- عدد ε به‌عنوان کوچکترین مقدار قابل نمایش، در برخی سیستم‌ها معادل صفر لحاظ می‌شود
  - درصورتی که e=8، این عدد معادل 2^{-128} می‌باشد که خیلی کوچک است

- دومین کوچکترین عدد پس از ε چگونه بدست می‌آید؟

نمایش اعداد در سیستم ممیز شناور

- دومین کوچکترین عدد پس از ε
  - افزایش مقدار fraction به‌اندازه 2^{-f}

$$N_{min+1} = \left(1 + 2^{-f}\right) * 2^{-2^{e-1}}$$

  - تفاضل N_{min} و N_{min+1}:

$$\Delta_1 = 2^{-f} * 2^{-2^{e-1}}$$

- روال افزایش گام به گام fraction را تا انتها ادامه می‌دهیم:

$$N_{min+2^f-1} = \left(2 - 2^{-f}\right) * 2^{-2^{e-1}}$$

- فاصله هردو عدد متوالی در این‌حالت برابر Δ_1 است

نمایش اعداد در سیستم ممیز شناور

- با بیشینه شدن fraction، برای افزایش باید e را زیاد کنیم:
  - عدد بزرگتر بعدی:
  - فاصله با عدد قبلی:
  - به‌دلیل افزایش یک واحدی نما $\Delta_2 = 2\Delta_1$
  - در نتیجه هرچه از صفر فاصله بگیریم، فاصله اعداد بیشتر می‌شود
    - فاصله بین اعداد نمادی از دقت نمایش

$$N = 1.0 * 2^{-2^{e-1}+1}$$

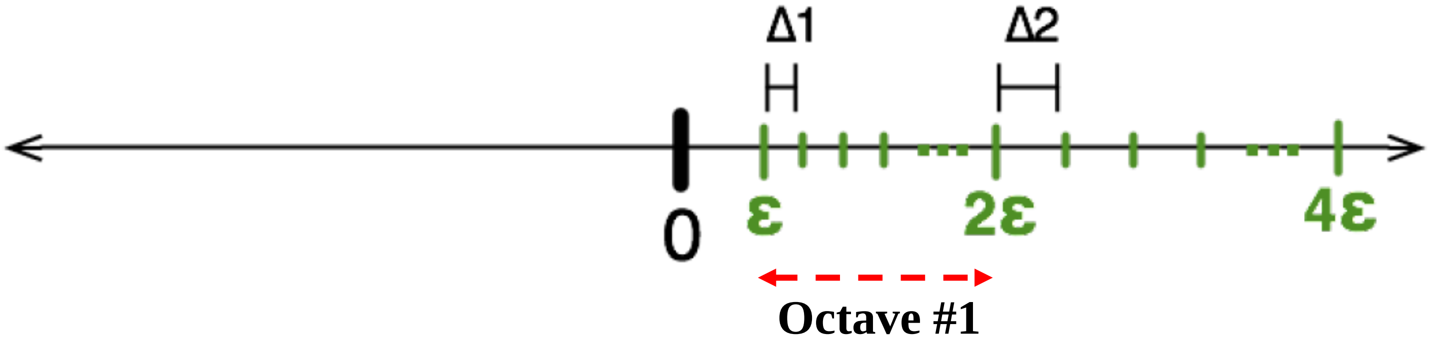
$$\Delta_2 = 2^{-f} * 2^{-2^{e-1}+1}$$

نمایش اعداد در سیستم ممیز شناور

- اکتاو: سطوح نمای مختلف در نمایش ممیز شناور
  - تعداد اعداد قابل نمایش در اکتاوها یکسان است (2^f)
  - هرچه اکتاو بالاتر باشد، فاصله اعداد آن بیشتر هستند

$$\Delta_1 = 2^{-f} * 2^{-2^{e-1}}, \Delta_i = 2^{i-1} * \Delta_1$$

- اعداد قابل نمایش در سیستم ممیز شناور



نمایش اعداد در سیستم ممیز شناور

- ویژگی‌های سیستم نمایش ممیز شناور(سیستم bias #2)
  - **تعداد اعداد قابل نمایش:** $2 * 2^f * 2^e - 1$
  - **بازه اعداد قابل نمایش:** $[-N_{max}, N_{max}]$
  - **حداقل و حداکثر اعداد:** $N_{max} = (2-2^{-f}) * 2^{Emax}$, $N_{min} = -N_{max}$
  - **دقت اعداد ذخیره شده:** Δ که برحسب نما تعیین می‌شود

مقایسه سیستم ممیز ثابت و ممیز شناور

- هزینه سخت‌افزاری کمتر نمایش ممیز ثابت نسبت به ممیز شناور
- تاخیر محاسبات کمتر نمایش ممیز ثابت نسبت به ممیز شناور
- عدم انعطاف‌پذیری نمایش ممیز ثابت نسبت به ممیز شناور
- قابلت نمایش اعداد خاص مانند e، π، Nan … در سیستم نمایش ممیز شناور