Computer Arkitektur
Floating Point (kort version)

Forelæsning 2
Brian Nielsen

Credits to
Randy Bryant & Dave O'Hallaron (CMU)

Vigtige læringsmål for dagens kursusgang

- Forstå mål og begrænsninger ved floating point repræsentationen.
- Præcision og afrunding
- Matematiske egenskaber (og mangel herpå: distributivitet og associativitet)

- Mindre intensivt
  - Detaljer om repræsentation
  - Normalform
  - Multiplikation og addition

Hvis / når I skal lave ”seriøse” FP applikationer, skal I sætte jer grundigt ind i hvordan det gøres rigtigt!

Design kriterier for floating-point tal

- Typiske applikationer
  - Videnskabelige beregninger og simuleringer
  - Fysik modeller, numeriske beregning, differential ligninger
  - IKKE: finansielle applikationer, som kræver præcise kr./øre beløb!!

    0.1+0.2 = 0.300000012 !?!
- Ønsker til repræsentation of reelle tal:
  - Regne så præcist som muligt
  - Håndtere meget meget store tal
  - Håndtere meget meget små tal
  - Matematisk velfunderet, så sædvanlige aritmetiske regneregler gælder

  - HW skal regne hurtigt
  - Bruge endeligt (og begrænset) antal bits (32-64 bits)
  - Skal kunne realiseres som digitale kredsløb: (FPU: floating point unit)

Ariane 5

- SW genbrugt fra Ariane 4, men Ariane 5 havde stejlere stigning
- Konvertering af 64-bit floating point til 16-bit signed integer
- Overløb!
- 500M€ nytårsraket

Patriot Missile

- Første Golf Krig
- Tid målt som 1/10 sekund, som integer
- Omregnes til sekunder ved at gange med 1/10
- 1/10 sekund =0.0001100110011001100110011001100....(lille afrundingsfejl)
- Efter 100 timers drift: total afrundingsfejl på 0.34sek =>ramte forbi Scud missil med mere end 0.5 km =>28 døde

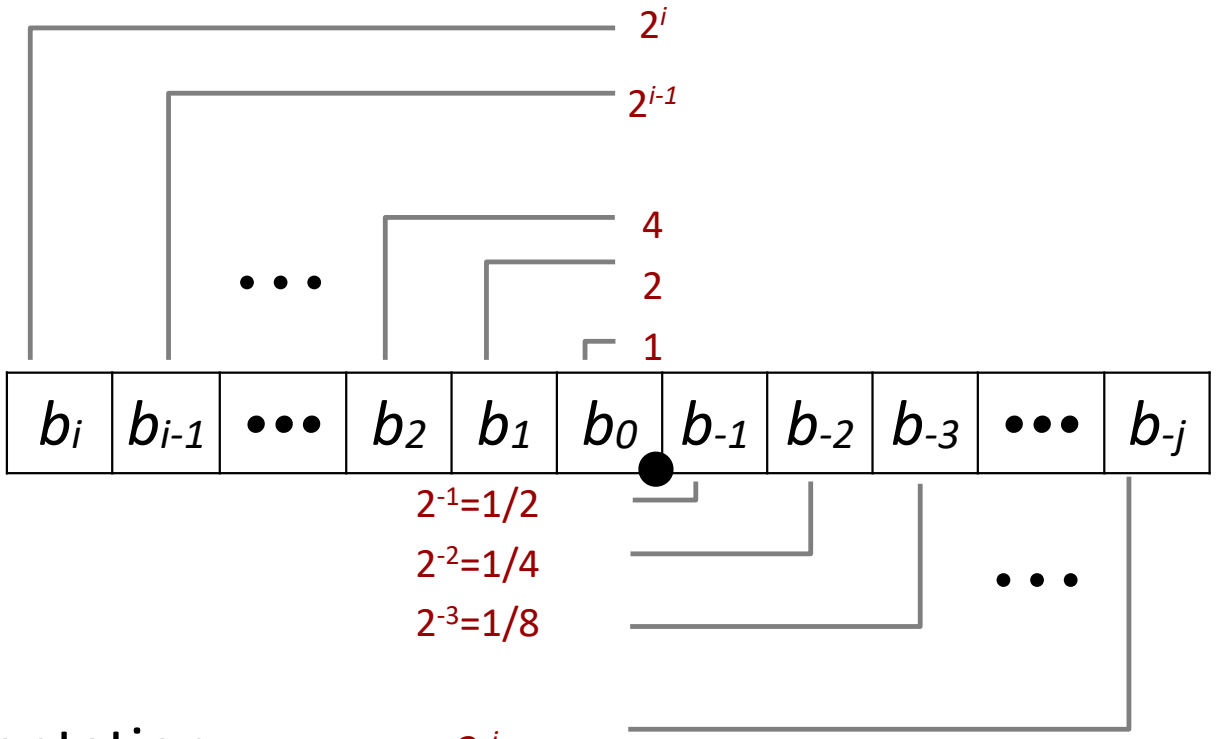
Binære brøker

Binære kommatal

- 1234.56_{10} ”Heltal.brøkdel”
- EX 1011.101_2?

Bemærk! Komma på DK ”,” vs ”.” på UK
1.234,56 Kr=?
1,234.56 Kr=?
Slides/bog bruger UK, ”.” er seperator

Binære brøker



- Repræsentation
  - Bits til højre for "komma" repræsenterer brøker som er potenser af 2
  - Repræsenterer rationelle tal:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Binære brøker: Eksempler

Værdi	**Repræsentation**
5 3/4	101.11_2
2 7/8	10.111_2
1 7/16	1.0111_2

$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 4+1+1/2+1/4$

- **Observationer**
  - Division med 2 fås ved højre-skifte (unsigned)
  - Multiplikation med 2 fås med venstre-skifte
  - Tal på formen $0.111111\ldots_2$ er lige under 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Hyppigt anvendt notation: $1.0 - \varepsilon$

Fixed point tal er bruges i mange DSP'ere.

Hvilke tal kan repræsenteres?

PP2.45

- Begrænsning #1
  - Vi kan kun repræsentere tal på formen $x/2^k$ eksakt.
    - Andre rationelle tal kræver et repeterende bit mønster
  - Værdi Repræsentation
    - 1/3 `0.0101010101[01]`$\ldots_2$
    - 1/5 `0.001100110011[0011]`$\ldots_2$
    - 1/10 `0.0001100110011[0011]`$\ldots_2$

- Begrænsning #2
  - Kun én placering af kommaet indenfor de *w* bits
    - Begrænsede tal-intervaller - hvad gør vi ved meget små tal? Meget store?

IEEE Floating Point

IEEE Floating Point

- IEEE Standard 754
  - Fremsat i 1985 som en uniform standard for floating point aritmetik
    - Tidligere var der mange forskellige formater med hver deres spidsfindigheder
  - Understøttes nu af alle større CPU'er.

- Design drevet af de numeriske egenskaber
  - God håndtering af afrunding, overløb
  - Sværere at lave hurtige operationer i hardware
    - Eksperter i numerisk analyse dominerede komiteen over hardware designere.

Floating Point Repræsentation

$-1^0 * 1.4375 * 2^7 = 184.0$ // 1*(23/16) *128
$-1^1 * 1.4375 * 2^3 = -11.5$ // -1*(23/16)*8

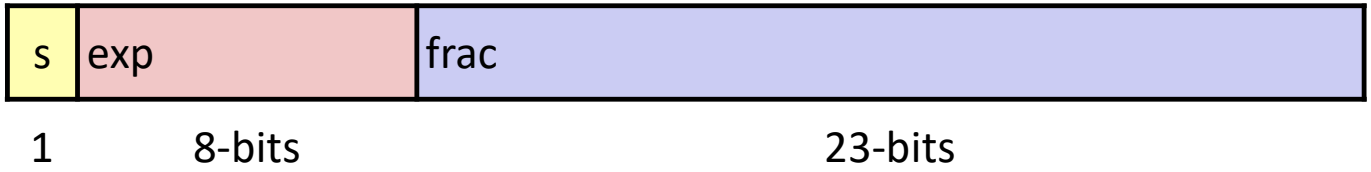
- Numerisk Format:

$$(-1)^s \cdot \boldsymbol{M} \cdot 2^{\boldsymbol{E}}$$

  - **Fortegnsbit (Sign bit)** ***s*** bestemmer om tallet er negativt eller positivt
  - **Signifikant** ***M*** : normalt et brøktal i intervallet [1.0,2.0).
  - **Exponent** ***E*** giver tallet en vægt med potens-af-2
- Indkodning som bitvektor
  - MSB s indeholder sign bit ***s***
  - exp felt indkoder ***E*** (men er ikke identisk til E)
  - frac felt indkoder ***M*** (men er ikke identisk til M)

Valg af præcision: muligheder

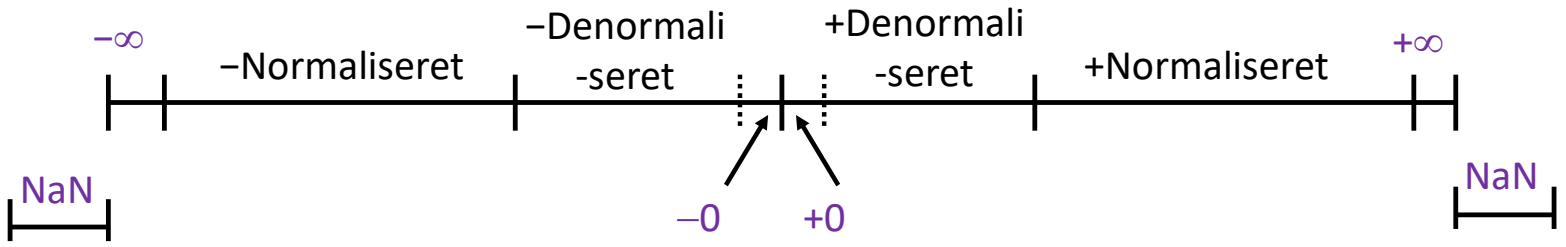
- "Single" præcision: 32 bits



- Double præcision: 64 bits

- Udvidet præcision: 80 bits (Kun Intel)

Overblik over repræsentation af floating point tal



NaN = Not a Number

2 indkodninger for M og E
”Denormaliseret repræsentation” giver extra god opløsning for tal tæt på 0

Normaliseret:
E = *e*xp – Bias
M=1+frac

De-normaliseret
E = 1 – Bias
M=frac

En værdi er “denormaliseret” når exp = 000…0

Særlige værdier

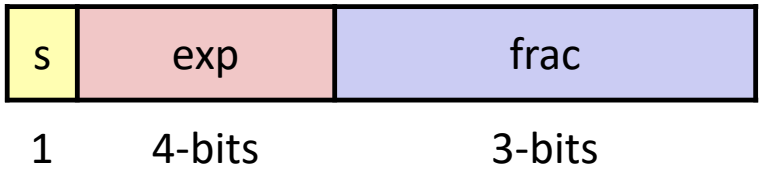
- Værdien er særlig når : **`exp`** = **`111…1`**

- Når **`exp`** = **`111…1`** og **`frac`** = **`000…0`**
  - Repræsenterer værdien ∞ **(uendelig, infinity)**
  - Gives når operationer resulterer i overløb (både positivt og negativt)
  - Fx.: 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞

- Når: **`exp`** = **`111…1`** og **`frac`** ≠ **`000…0`**
  - **Not-a-Number (NaN)**
  - Anvendes når den numeriske værdi ikke kan beregnes
  - F.x., sqrt(–1), ∞ − ∞, ∞ × 0

Eksempel :
”Lillebitte” Floating Point

“Lillebitte Float” Eksempel



- 8-bit Floating Point repræsentation
  - Den mest betydende bit angiver fortegn
  - Næste 4 bits angiver **`exp`**, med bias 7
  - Sidste 3 bits er **`frac`**

- Samme generelle format som IEEE
  - normaliseret, denormaliseret
  - repræsentation af 0, NaN, infinity

Talområde (Positive)

$v = (-1)^s \, M \cdot 2^E$

norm: E = Exp – Bias

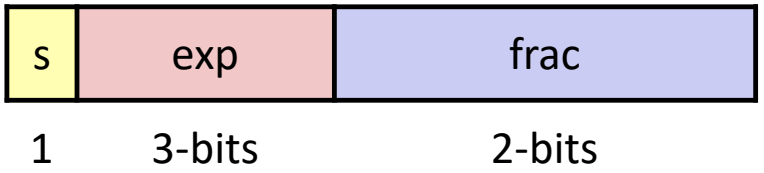
denorm: E = 1 – Bias

	s	exp	frac	E	Value	
Denormaliserede tal	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	Tættest på 0
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	største denorm
Normaliserede tal	0	0001	000	-6	8/8*1/64 = 8/512	mindste norm
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	tættest på to 1 nedenfra
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	Tættest på 1 oppe fra
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	største norm
	0	1111	000	n/a	inf	

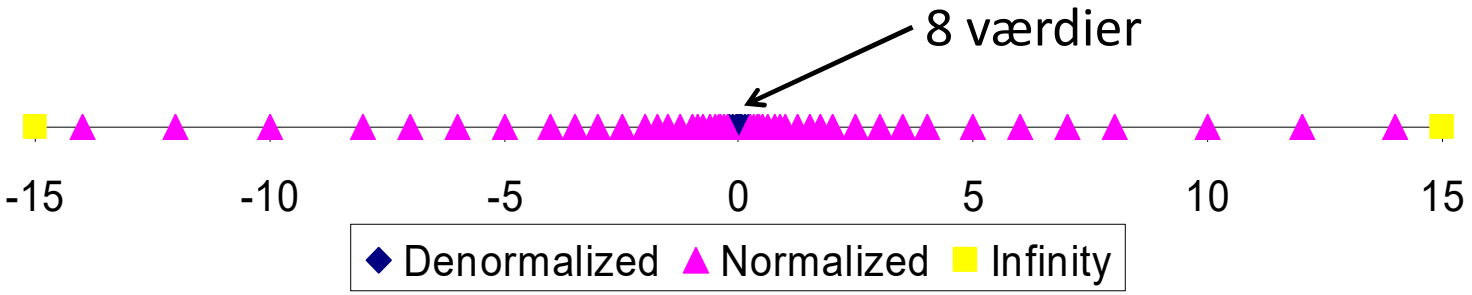
NB!

Fordeling af Værdier

- 6-bit IEEE-lignende format
  - e = 3 eksponent bits
  - f = 2 brøk bits
  - Bias er $2^{3-1}-1 = 3$

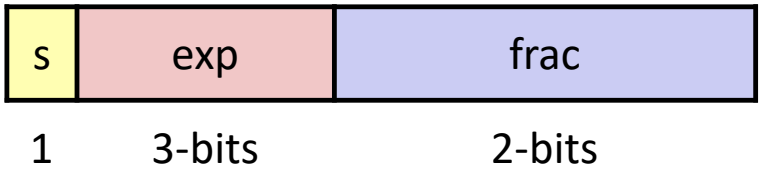


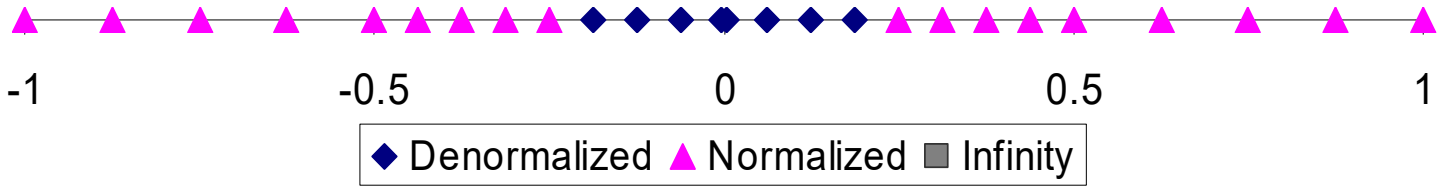
- Bemærk at fordelingen bliver tættere tæt på 0.



Fordeling af Værdier (zoom ind på -1…1)

- 6-bit IEEE-lignende format
  - e = 3 eksponent bits
  - f = 2 brøk bits
  - Bias = 3





Egenskaber ved IEEE formatet

- Floatpoint repræsentation af +0 same som Integer 0
  - Alle bits = 0

- Vi kan (næsten) anvende Integer sammenligning
  - Skal først sammenligne fortegnsbit
  - Skal håndtere “–0” = 0
  - NaNs er problematiske
    - Bliver større end alle andre værdier
    - Hvad skal resultatet af sammenligningen være?
  - Ellers OK
    - Denorm vs. normalized
    - Normalized vs. infinity

Afrunding

Grundlæggende Floating Point Operationer

- $x +_f y = Round(x + y)$

- $x \times_f y = Round(x \times y)$

- Grundlæggende idé
  - Beregn exakte resultat
  - Indpas det i den valgte præcision
    - Muligvis overflow, hvis eksponenten er for stor
    - Muligvis afrunding, så det passer med **`frac`**

Afrunding

- Afrundings former (illustreret som € afrunding)

	€1.40	€1.60	€1.50	€2.50	–€1.50
• Mod nul (Towards zero)	€1	€1	€1	€2	–€1
• Rund ned (Round down) (−∞))	€1	€1	€1	€2	–€2
• Rund up (round up (+∞))	€2	€2	€2	€3	–€1
• Nærmeste lige (Nearest Even)	€1	€2	€2	€2	–€2

”Nærmeste lige” regel
- <”halv vejs” : rund ned
- >”halv vejs”: rund op
- == ”halv vejs”: afrund til nærmeste lige tal.

Nærmeste-lige er standart-indstillingen

Analyse af nærmeste-lige afrunding

- Standart-indstillingen
  - Svært at få anden form uden brug af assembler, se dog `fesetround()` i C99.
  - De andre former giver statistisk bias
    - Fx., sum af en serie positive tal giver konsistent/systematisk over- eller under-estimering
    - Kan bruges til at finde øvre/nedre grænser for x: $x^- \leq x \leq x^+$
- Anvendelse på andre decimaler / Bit positioner
  - Når præcist "halvvejs" mellem to mulig værdier,
    - Afrund så mindst betydende ciffer er lige
  - Fx., afrund til nærmeste hundrede-dele

7.8949999	7.89	(Mindre end "halv vejs")
7.8950001	7.90	(Mere end "halv vejs")
7.8950000	7.90	("halv vejs" —rund op)
7.8850000	7.88	("halv vejs" —rund ned)

Matematiske egenskaber

Mathematiske egenskaber ved FP add

- Sammenlign med egenskaberne i en Abel'sk gruppe
  - Lukket under addition? ***Ja***
    - Men kan give "infinity" eller "NaN"
  - Kommutativ? ***Ja***
  - Associativ? ***Nej***
    - Overløb og upræcished forårsaget af afrunding
    - `(3.14+1e10)-1e10 = 0, 3.14+(1e10-1e10) = 3.14`
  - 0 er additiv identitet? ***Ja***
  - Hvert element har en additiv invers? ***Næsten***
    - Undtaget "infinit" og "NaN"s
- Monotonicitet ***Næsten***
  - $a \geq b \Rightarrow a+c \geq b+c$?
    - Undtaget "infinit" og "NaN"s

Implikationer for compilere:

Mathematiske egenskaber ved FP Mult

- Sammenlign med "kommutativ ring"
  - Lukket under multiplikation? ***Ja***
    - Men kan give "infinity" eller "NaN"
  - Multiplikation er kommutativ? ***Ja***
  - Multiplikation is associativ? ***Nej***
    - Mulighed for overløb, upræcist forårsaget af afrunding
    - Fx: `(1e20*1e20)*1e-20= inf, 1e20*(1e20*1e-20)= 1e20`
  - 1 er multiplikativ identit? ***Ja***
  - Multiplikation distribuerer over addition? ***Nej***
    - Mulighed for overløb, upræcished forårsaget af afrunding
    - `1e20*(1e20-1e20)= 0.0, 1e20*1e20 – 1e20*1e20 = NaN`
- Monotonisitet ***Næsten***
  - $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c$?
    - Undtaget "infinity" og "NaN"s

Implikationer for compilers

Floating Point in C

- C garanterer to præcisioner
  - **`float`** single præcision
  - **`double`** double præcision
- Type-konvertering (Casting)
  - Casting mellem **`int`**, **`float`**, og **`double`** ændrer bit repræsentation
  - **`double`**/**`float`** → **`int`**
    - Trunkerer brøken
    - Som ved afrunding mod nul
    - Ej defineret for infinity, NaN: sættes generelt til TMin
  - **`int`** → **`double`**
    - præcis konvertering, sålænge **`int`** har ordstørrelse på ≤ 53 bits
  - **`int`** → **`float`**
    - Afrunder i overensstemmelse med anvendte afrundingsform (default: ”Round to nearest even”).

Resumé

- IEEE Floating Point har veldefinerede matematiske egenskaber
- Repræsenter tal på formen $M \times 2^E$
- Kan ræsonnere om resultatet uafhængigt af FP implementation
  - Som ”eksakt resultat, dernæst afrundet”
- Ikke det samme som aritmetik på reelle tal
  - Bryder associativitet/distributivitet
  - Gør det svært for compilers, numeriske applikationer, og programmører

Hvis / når I skal lave ”seriøse” FP applikationer, skal I sætte jer grundigt ind i hvordan det gøres rigtigt!