Hvordan virker en lommeregner?

Mogens Esrom Larsen, Københavns universitets matematiske institut

Beskedenhed er en dyd, der særlig rinder os i hu, når vi stiller krav til os selv. Det gælder også vor indsats i udregninger, - vi sparer helst på egne kræfter. Sådan har det altid været. I det gamle Babylon 2000 år før vor tidsregnings begyndelse havde man lertavler med tabeller over produkter og kvotienter, så man kunne slå op i stedet for at beregne eller huske. Lertavlerne var netop i lommeformat. Deres undskyldning for dovenskaben var nu bedre end vor tids. De brugte nemlig 60 som grundtal, hvor vi jo de sidste par tusind år har klaret os med 10. Det betyder, at "den lille tabel" hos dem gik helt op til $59 \cdot 59$ (= 3481), hvor vi kan klare os med $9 \cdot 9 = 81$. Det har været de færreste, der kunne bare den tabel udenad.

Nu har vi taget det næste skridt i retning af at gøre "den lille tabel" overkommelig at huske. Lommeregnere og computere bruger internt grundtallet 2, så deres "lille tabel" ser sådan ud:

$$1 \cdot 1 = 1$$

Til gengæld kommer regneoperationer og menteoverførsler itl at koste meget. Men det gør mindre, når man udfører 10^6 elementære operationer pr. sek.

I en lommeregner er det enkelte tal repræsenteret af en streng af bits, et såkaldt "ord", hvor hver bit kan stå i to tillinger, f.eks. tændt/slukket. Den ene stilling lader vi repræsentere "1", den anden "0". På den måde repræsenterer ordet et tal, skrevet i 2-talsystemet. Ordenes længde varierer omkring et halvt hundrede bits. Men lad os for nemheds skyld tænke os, at de er på 10 bits. Lad os videre tænke os, at de 10 bits står i positionerne:

s s s s s s t s t s

Tolker vi det, som om *s* betyder "0" og *t* betyder "1", står der tallet - i 2-talsystemet -

0 0 0 0 0 0 1 0 1 0

Det betyder - i 10-talsystemet -

$$1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 10$$

Lommeregneren har indbygget nogle grundlæggende kunster med ordene. Den kan f.eks. bytte om på *s* og *t* i et ord. Den kan også rykke alle tilstandene en (eller et andet fast antal) plads(er) til venstre. De yderste tilstande til venstre kommer så ind fra højre.

Operationen kaldes et "cyklisk skift". Skifter vi 1 plads i 10-tallet ovenfor, får vi

0 0 0 0 0 1 0 1 0 0

Det er 16 + 4 = 20. Et skift kan også udføres på den måde, at der kommer lutter 0'er eller lutter 1'er ind fra højre. Der er så tale om et "ikke cyklisk skift" eller bare et skift. Et skift svarer til at gange med 2. Det er ligesom at gange med 10 i 10-talsystemet ved at sætte et 0 bagefter.

Addition er lidt mere besværlig. Den udføres ikke af én operation, men af flere hundrede.

Lad os tænke os, at et tal er givet, og at vi ønsker at lægge et andet tal til, som er en ren potens af 2, - dvs. at tallet kun indeholder ét 1-tal i sin binære fremstilling.

Givet X:	x	x	x	x	0	1	x	x	x	x
Givet 2^4:	0	0	0	0	0	1	0	0	0	0

Dette par ændres til

Y:	x	x	x	x	0	0	x	x	x	x
2^5:	0	0	0	0	1	0	0	0	0	0

Tallet 2^4 er blevet skiftet til 2^5, mens *X* har fået sit 1-tal byttet med et 0. Denne operation kunne vi kaldet "1 in mente".

Næste gang ændrer vi parret til

Z:	x	x	x	x	1	0	x	x	x	x
O:	0	0	0	0	0	0	0	0	0	0

Denne gang slap vi med "0 + 1".

Et eksempel:

Givet184:	0	0	1	0	1	1	1	0	0	0
+8:	0	0	0	0	0	0	1	0	0	0

Parret sættes ved "1 in mente" til

176:	0	0	1	0	1	1	0	0	0	0
+16:	0	0	0	0	0	1	0	0	0	0

Som efter endnu en "1 in mente" bliver til

160:	0	0	1	0	1	0	0	0	0	0
+32:	0	0	0	0	1	0	0	0	0	0

Der efter en sidste "1 in mente"bliver til

128:	0	0	1	0	0	0	0	0	0	0
+64:	0	0	0	1	0	0	0	0	0	0

Endelig kan vi udføre "0+1"

192:	0	0	1	1	0	0	0	0	0	0
+0:	0	0	0	0	0	0	0	0	0	0

Processen ender, når vi første gang udfører "0 + 1".

Når vi skal lægge vilkårlige tal sammen, lægger vi de forskellige 2-potenser (1-taller) fra det ene af tallene til det andet, én ad gangen.

Når lommeregneren skal multiplicere to vilkårlige tal med hinanden, multipliceres det ene efter tur med de forskellige 2-potenser i det andet. Resultaterne lægges så sammen til sidst. Multiplikationen tager derfor ca. 25 gange så lang tid som additionen.

Et eksempel, $11 \cdot 13$.

11:	0	0	0	0	0	0	1	0	1	1
13:	0	0	0	0	0	0	1	1	0	1

Vi ganger 11 med 8, dvs. det første 1-tal i 13, et skift på 3 pladser, og vi ganger 11 med 4, dvs. et skift på 2 pladser:

$11 \cdot 8$ =88:	0	0	0	1	0	1	1	0	0	0
$11 \cdot 4$ =44:	0	0	0	0	1	0	1	1	0	0

Disse to tal lægges nu sammen i 3 omgange, en for hvert af 1-tallerne. Vi får summen

132:	0	0	1	0	0	0	0	1	0	0

Vi mangler nu blot $1 \cdot 11$=11 svarende til det sidste 1-tal i 13. Disse 11 skal lægges til:

132:	0	0	1	0	0	0	0	1	0	0
11:	0	0	0	0	0	0	1	0	1	1
143:	0	0	1	0	0	0	1	1	1	1

Vi var heldige, vi slap med 4 menteoverførsler. Ialt har vi udført 10 elementære additioner, 4 "1 in mente" og 6 "0 + 1".

Nu er den virkelige repræsentation af tallene i lommeregneren noget mere sofistikeret end her antydet. Det enkelte tal repræsentes af en ''sætning'' på 4 ord, nemlig et ''fortegn'' + eller - repræsenteret af et ord på 1 bit, en ''mantisse'', der angiver de første ca. 12 decimaler i tallet, repræsenteret af et ord på ca. 40 bit, endnu et fortegn på 1 bit, og endelig en ''eksponent'', der angiver, ''hvor kommaet skal stå'', angivet ved et tal op til ca. 500, repræsenteret af et ord på ca. 9 bit, f.eks. betyder ''sætningen''

(-, 1.76, +, 2)

at tallet skal være -176 ($-1.76 \cdot 10^2$).

Fordelen er, at man billigt slipper til at regne tilstrækkeligt præcist med selv meget store og meget små tal. Af et 20-cifret tal angiver man kun de første 12 cifre, resten er erstattet af 0'er. Men det er tit godt nok. Når to tal ganges sammen, må man gange første fortegn med hinanden efter de sædvanlige regler, - · - = + o.s.v., mantisserne ganges med hinanden, som vi har set ovenfor, og eksponenterne lægges sammen, idet de regnes med det andet fortegn.

Nu kan de fleste lommeregnere jo mere end de fire regningsarter, - i hvert fald tilsyneladende. De kan kvadratrod, sinus, cosinus, logaritmer og eksponentialfunktion, og meget andet. Men, hvordan bærer de sig ad?

Når vi har tastet et tal ind, som f.eks. 2, og trykker på knappen ''kvadratrod'', står der øjeblikkeligt i displayet:

1.41421356237

Men det er nu ikke et resultat, den blot henter i hukommelsen, - det er der alligevel ikke plads til. Nej, den går frem efter en opskrift, ligesom ved multiplikation, men til kvadratrodsuddragning kunne det være denne:

$$x_{i+1} = \frac{x_i}{2} + \frac{1}{x_i} \quad ; \quad x_0 = 2.$$

Opskriften giver tallene:

$x_0 = 2$
$x_1 = 1.5$
$x_2 = 1.4167$
$x_3 = 1.4142157$
$x_4 = 1.41421356238$

Processen standser, når to tal er blevet ens, altså når $x_{i+1} = x_i$. Der er intet trylleri, kun de fire regningsarter og - den evige gentagelse! Det er et af de tricks, som elektronikken har bragt ære og værdighed, gentagelsen, eller som det hedder i fagjargonen: iterationen. Mange bække små gør en stor å. Formlen ovenfor giver altid $\sqrt{2}$, uanset værdien af $x_0 > 0$; hvis man skal finde $\sqrt{a}$, må man bruge formlen:

$$x_{i+1} = \frac{x_i}{2} + \frac{a}{2x_i} \quad ; \quad x_0 = 2.$$

Heller ikke eksponentialfunktionen, e^x, kan beregnes, i hvert fald ikke, hvis x ikke er et helt tal. Men kan vi finde en formel, der kan approksimere e^x med en fejl, der er mindre end 10^{-13}, så ses det ikke på en 12-cifret mantisse. For simpelheds skyld viser jeg et eksempel på en approximation med en fejl, der er mindre end 10^{-5}.

Først skriver vi

$$e^x = 2^{h+b},$$

hvor h er et helt tal og b en brøk så $-½ < b \leq ½$. Så er

$$x = (h+b) \cdot \ln 2,$$

hvor ln 2 = 0.69314718056, den ''naturlige logaritme'' til 2, som er bestemt af $e^{\ln 2} = 2$. Vi finder derfor h og b ved at dele x med ln 2. Er f.eks. $x = 10$, finder lommeregneren let

$$10/\ln 2 = 14.4269504089,$$

altså $h = 14$ og $b = 0.4269504089$.

Nu er 2^{14} jo i lommeregneren 100000000000000, og multiplikation med den er blot et skift. Så

$$e^x = 2^{h+b} = 2^h \cdot 2^b = 2^h \cdot e^{b \ln 2},$$

hvor faktoren 2^h er næsten gratis. For at bestemme cifrene i e^x er det derfor nok at bestemme cifrene i $e^{b \ln 2}$; altså nok at kende e^x, når $-½ \ln 2 < b \leq ½ \ln 2$.

I eksemplet ($x = 10$) er

$$b \ln 2 = 0.295939472168,$$

som lommeregneren finder ved en enkelt multiplikation.

I det interessante interval kan vi bruge funktionen

$$f(x) = \frac{x^2+6x+12}{x^2-6x+12}$$

som netop approksimerer bedre end 10^{-5}.

Lommeregneren finder let

$$f(b \ln 2) = 1.344385,$$

Den korrekte værdi er 1.34438878143, så fejlen er som lovet, nemlig ca. $½ \cdot 10^{-5}$.

$f(b \ln 2)$ skal så ganges med $2^{14} = 16384$, men det er jo blot et skift. Det giver værdien

22026.4

Den korrekte værdi for e^{10} er 22026.4657948, - fundet på tilsvarende måde med en mere kompliceret brøk. Fejlen er nu 0.07, - vi har jo ganget $½ \cdot 10^{-5}$ med 16384.

Fremtidens lommeregner.

Den tal-repræsentation, vi her har beskrevet, har været brugt i de sidste 30 år. Den udmærker sig ved at udnytte regnemaskinens hukommelse praktisk taget maksimalt. Og det har hidtil været ønskeligt, fordi hukommelsen var dyr, mens regningerne gik - for alle at se - svimlende hurtigt.

Men nu, hvor hukommelserne er kommet ned i pris, og hvor mere sofistikerede opgaver begynder at trænge sig på med meget lange regnetider, melder det spørgsmål sig naturligt: vil en mere ekstensiv udnyttelse af hukommelsen kunne tillade, at de elementære regneoperationer organiseres, så hastigheden forøges væsentligt?

På Boeing Aerospace Company's forskningsafdeling har man følt et stigende behov for mindskede regnetider, og det forlyder, at de tre forskere derfra, Capps, Falk og Houk har foreslået, at man går over til at anvende ''kinesiske restklasser'' i den interne talrepræsentation i hukommelsen. Og det er tænkeligt, at det er vejen frem. Jeg skal prøve at forklare, hvordan.

Lad mig først forklare, hvordan restklasseregning kan gå langt hurtigere end almindelig regning, - på bekostning af hukommelsen! Man lader et bit-mønster altså et ord med f.eks. 23 bits repræsentere tallene 0-22, sådan at 7 har et

1-tal på plads nr. 7:

7=00000000000000001000000

Hvor ord af denne længde før kunne repræsentere millioner af forskellige tal, kan de nu kun repræsentere 23 forskellige.

Vi skal nu lægge et andet tal, f.eks. 17 til. Vi foretager et cyklisk skift af 7 på 17 pladser til venstre. Herved får vi tallet

1=00000000000000000000001

Denne form for regning kaldes ''restklasseregning'' i restklasserne ved deling med 23. Vi regner altså $24 \equiv 1\ (23)$, fordi de to tal giver samme rest, nemlig 1, ved deling med 23. På samme måde kan man gange. Vi regner $7 \cdot 17 = 119 \equiv 4\ (23)$, fordi $119 = 5 \cdot 23 + 4$. Addition gik hurtigt, fordi den blev udført ved hjælp at et skift.

Multiplikation kan i virkeligheden udføres næsten lige så hurtigt, som vi skal se.

Man kan se væk fra $23 \equiv 0\ (23)$, - ganger vi med 0, får vi 0. Til de 22 andre tal, 1-22, laver vi en ''logaritmetabel'' af restklasser ved deling med 22:

rest (23)	logaritme (22)	rest (23)	logaritme (22)
1	0	12	20
2	2	13	14
3	16	14	21
4	4	15	17
5	1	16	8
6	18	17	7
7	19	18	12
8	6	19	15
9	10	20	5
10	3	21	13
11	9	22	11

Når vi skal udregne $7 \cdot 17$, slår vi op i logaritmetabellen, $\log(7) = 19$, $\log(17) = 7$; dem lægger vi sammen og får 4 ($26 \equiv 4\ (22)$); resultatet finder vi i logaritmetabellen ved at finde 4 i logaritmetavlen og oversætte det til 23-resten, der tilfældigvis også er ''4''.

Vi kan udføre disse tabelopslag meget hurtigt, derfor kan restklasse-multiplikation udføres næsten så hurtigt som addition, og næsten så hurtigt som et skift.

Vi laver nu en række centralenheder, der kan regne hurtigt med restklasser hørende til forskellige primtal. For hvert primtal må man have en ''logaritmetabel'', jeg angiver som eksempel tabellen for 47:

rest	log	rest	log	rest	log	rest	log	rest	log
1	0	11	7	21	6	31	3	41	15
2	18	12	10	22	25	32	44	42	24
3	20	13	11	23	5	33	27	43	13
4	36	14	4	24	28	34	34	44	43
5	1	15	21	25	2	35	33	45	41
6	38	16	26	26	29	36	30	46	23
7	32	17	16	27	14	37	42		
8	8	18	12	28	22	38	17		
9	40	19	45	29	35	39	31		
10	19	20	37	30	39	40	9		

For eksempel $7 \cdot 17$; $\log(7) = 32$, $\log(17) = 16$, $32 + 16 = 48 \equiv 2\ (46)$ svarende til 25; thi $119 = 2 \cdot 47 + 25$ eller $119 \equiv 25\ (47)$.

Restklasseregning

To positive hele tal, n og m, kaldes *kongruente modulo* et tredie, p, hvis deres differens, m-n, er delelig med p. Altså, hvis der findes et helt tal q, sådan at

$$m - n = q \cdot p$$

Vi skriver i så fald

$$m \equiv n \pmod{p}$$

For eksempel $2 \equiv 7 \pmod 5$, fordi $7 - 2 = 1 \cdot 5$.

Vi repræsenterer nu hvert tal ved dets sæt af rester ved deling med de valgte primtal. Når så vi ønsker at danne summen eller produktet, gør vi det for hvert primtal ved restklasseregning. Derved fås netop det sæt af rester, som karakteriserer det ønskede resultat.

Ikke alene kan disse regninger udføres hurtigt og uden menteoverførsler, men netop fordi de er uafhængige af hinanden, kan de udføres samtidig, - hvilket kan udnyttes på moderne ''vektorprocessorer''.

At det fungerer går tilbage til oldtidens Kina, hvor metoden omtales i det 3. årh. af Sun-Tzï, - hvorfor metoden også kaldes ''kinesisk restklasseregning''.

Som et simpelt eksempel kan vi udregne $27 \cdot 29$ ved hjælp af restklasserne for 23 og 47.

27 har resterne 4 (23) og 27 (47).

29 har resterne 6 (23) og 29 (47).

Logaritmerne er hhv. $\log(4) = 4$ (22), og $\log(6) = 18$ (22); summen heraf er 0 (22), der svarer til klassen 1 (23).

$\log(27) = 14$ (46) og $\log(29) = 35$ (46), så summen er 3 (46), der svarer til 31 (47).

Produktet har altså resterne 1 (23) og 31 (47). $27 \cdot 29$ er derfor det (mindste) tal, der har disse to rester. Altså findes p, q, så

$$27 \cdot 29 = 1 + p \cdot 23 = 31 + q \cdot 47$$

Denne ligning løses fikst ved at tage resterne efter deling med 23. Vi får

$$1 \equiv 8 + q\ (23)$$

altså

$$q = 23 - 7 = 16.$$

Derfor

$$27 \cdot 29 = 31 + 16 \cdot 47 = 783.$$

(Vi har tilsyneladende brugt lige så meget krudt på at finde $16 \cdot 47$, som det havde kostet at udregne $27 \cdot 29$, men man må huske, at denne anstrengelse kun udføres til slut; vi sparer den i de tusinder af mellemregninger, som er brugt til at iterere os frem til sin x, - eller hvad opgaven nu var.)

Dette er måske endnu et eksempel på, at elektronikkens tidsalder kan forvandle et stykke matematik fra kuriøsitet til fundament.

Mogens Esrom Larsen er ansat ved Københavns Universitets matematiske institut som lektor uden særlige kvalifikationer.