

MAT1030 – Diskret Matematikk

Forelesning 3: Litt om representasjon av tall

Dag Normann

Matematisk Institutt, Universitetet i Oslo

26. januar 2010

(Sist oppdatert: 2010-01-26 14:20)



Kapittel 3: Litt om representasjon av tall

Hva vi gjorde forrige uke

- Vi diskuterte løst hva vi skal mene med en algoritme.
- Vi så hvordan vi kan beskrive algoritmer ved hjelp av *pseudokoder*.
- Vi beskrev fem *kontrollstrukturer* som vi vil bruke i våre pseudokoder, og vi så på en del eksempler.
- Vi diskuterte kort det *dekadiske* tallsystemet, binære-, oktale og heksadesimale tall.
- Vi forventer ikke noen stor regneferdighet i tilknytning til tallsystemene, men en viss forståelse av dem.

En rask gjennomgang av kapittel 3

- Kapittel 3 inneholder mye som allerede er velkjent for informatikkstudenter.
- Vi skal derfor gå raskt gjennom de viktigste punktene.
- Resten blir overlatt til selvstudium.
- Dere kan be om hjelp hos plenumsregner eller gruppelærere, men vær forberedt på at noen av dem **ikke** har tatt noe emne i digital representasjon.
- Se forelesningsnotatene for 2008 for noe **fyldigere** forelesningsnotater.

Introduksjon

- Grunnenheten er en **bit**.
- En bit vil være i en av to tilstander: 0 eller 1.
- En **byte** er en sekvens av 8 bit, f.eks.

10011001.

- I en byte kan vi lagre $2^8 = 256$ forskjellige informasjoner.
 - Dette svarer til alle tall med to sifre i det heksadesimale systemet.

Introduksjon

- Hele tall og reelle tall er forskjellige typer tall, og “samme” tall må representeres forskjellig når det oppfattes som et heltall og når det oppfattes som et reelt tall.
- Ved å kjenne til hvordan tall **representeres**, vil vi kjenne til begrensningene og mulige feilkilder.

Hvis man skal foreta en numerisk beregning hvor antall avrundinger er i millionklassen, er det viktig å vite hvor stor feil som kan oppstå fordi representasjonen i maskinen ikke er nøyaktig. Det finnes uendelig mange tall, men bare endelig mange av dem kan representeres i en konkret maskin.

Litt om representasjon av hele tall

Når vi skal representere hele tall i en datamaskin er det tre spørsmål som må besvares:

1. Hvor mange tall ønsker vi å representere?
2. Hvilke tall ønsker vi å representere?
3. Hvordan vil vi representere dem?

Svaret på spørsmål 1 avhenger av hvor mange bit/byte vi vil sette av for å representere et enkelt tall.

Bruker vi flere bit, kan vi representere flere enkelttall, men vi vil bruke lengere tid på å manipulere tallene, og vi vil ha mindre plass til andre formål.

Litt om representasjon av hele tall

- Anta – for enkelhets skyld – at vi kun har én byte, det vil si 8 bit.
- Da kan vi representere $2^8 = 256$ forskjellige tall.
- F.eks.
 - alle tall n slik at $0 \leq n \leq 255$,
 - alle tall n slik at $-255 \leq n \leq 0$, eller
 - alle tall n slik at $-128 \leq n \leq 127$.
- Det siste er det vanligste.

Litt om representasjon av hele tall

- Den vanligste er å la det første bitet være et **fortegnsbit**, som er 1 hvis n er negativt.
- De resterende 7 bit brukes litt forskjellig avhengig av om n er negativt eller ikke.
- Hvis n er positivt, så bruker vi binærformen til n .
- Hvis n er negativt, så bruker vi binærformen til $n + 128$.

Eksempel

- La $n = -126$.
- Fortegnsbitet blir 1.
- Vi representerer $n + 128 = 2$ med 7 bit som 000 0010,
- Representasjonen blir derfor 1000 0010.

Litt om representasjon av hele tall

- Et viktig poeng med **representasjoner** er at vi så enkelt som mulig skal kunne regne på det som representeres.
- Man kan være fristet til å representere et negativt tall, $-n$, med fortegnbit 1 og deretter binærformen til n .
- Det er to grunner til at vi vil gjøre det anderledes.
 - Vi ville få to representasjoner av 0 og ingen av -128 .
 - Vi ville ikke kunne brukt samme prosedyrer for addisjon og subtraksjon for positive og for negative tall.
 - Dette illustreres best hvis vi bare bruker fire bit i representasjonen:

Litt om representasjon av hele tall

| Heltall | Representasjon med fire bit |
|---------|-----------------------------|
| -8 | 1000 |
| -7 | 1001 |
| -6 | 1010 |
| -5 | 1011 |
| -4 | 1100 |
| -3 | 1101 |
| -2 | 1110 |
| -1 | 1111 |
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |

Litt om representasjon av hele tall

Definisjon (Representasjon av hele tall)

Hvis vi har k bit til disposisjon for å representere hele tall, ($k = 8$, $k = 16$ og $k = 32$ er de mest aktuelle) representerer vi alle hele tall a slik at $-2^{k-1} \leq a < 2^{k-1}$ på følgende måte:

- Hvis $a \geq 0$, er første bit 0 og resten er det binære tallet for a med $k - 1$ sifre
- Hvis $a < 0$ er første bit 1 og resten er det binære tallet for $2^{k-1} + a$

Litt om representasjon av hele tall

Eksempel

- La $a = -23$. Da er $a < 0$, så første bit må være 1.
 - $128 + (-23) = 105$ og $1101001_2 = 105$
 - Det gir at representasjonen av a er 11101001.
-
- Dette synes som en tungvint måte, og det er det.
 - Hvis vi gjennomfører eksemplet ved å regne binært, ser kommer vi frem til det som kalles **toerkomplementet**.
(Vi regner eksemplet binært på tavlen.)

Litt om representasjon av hele tall

Definisjon (Toerkomplement)

- La $a_1 \cdots a_k$ være en sekvens av 0'er og 1'ere, med en 0'er lengst til venstre, og med minst én 1'er.
- **toerkomplementet** til sekvensen får vi ved å starte fra høyre, lese ett og ett siffer og
 1. Alle 0'er til høyre beholdes.
 2. Første 1'er fra høyre beholdes.
 3. Alle siffer til venstre for første 1'er endres fra 0 til 1 eller fra 1 til 0.
- Hvis $x_1 \cdots x_8$ er datarepresentasjonen av et positivt heltall n , er toerkomplementet representasjonen av $-n$.
- Dette gjelder selvfølgelig om vi bruker 16-bit representasjoner eller 32-bit-representasjoner også.

Litt om representasjon av reelle tall

For å forstå prinsippene for representasjon av reelle tall i en maskin, kan vi se på hvordan reelle tall fremstilles på en lommeregner eller i en tekst som omhandler store eller små tall. I stedet for å skrive

2308501000000000000000

kan vi skrive

$0,2308501 \cdot 10^{21}$

eller

2,308501E20

Med unntak av at vi vil bruke binære tall i stedet for vanlige tall fra titallsystemet, vil vi kode reelle tall via tre informasjonsbiter, **fortegn**, **sifrene brukt** og **eksponent**.

Litt om representasjon av reelle tall

Vi vil skrive tallene på **normalisert binær form**:

- Et fortegn, + eller –.
- Et *binært* desimaluttrykk på formen

$$0,1\dots$$

kalt **signifikanden**.

- En **eksponentdel**

$$2^n$$

hvor n er et heltall.

- Vi har enkle prosedyrer for å utføre aritmetikk på tall på normalisert binær form.

Litt om representasjon av reelle tall

Eksempel

a) $0,101100101 \cdot 2^{-3}$ er på normalisert binærform.

Vi kunne skrevet dette som $0,000101100101$.

b) $101,0101101$ er ikke på normalisert binærform.

Da burde vi skrevet $0,1010101101 \cdot 2^3$

c) $0,1000010100001 \cdot 2^{28}$ er på normalisert binær form.

I dette tilfellet er det liten grunn til å skrive tallet på eksakt form, og det ville også antydnet en større nøyaktighet enn det vi trolig har grunnlag for.

d) $0,11010 \cdot 2^{-87}$ er på normalisert binær form.

Litt om representasjon av reelle tall

- Vi følger læreboken i beskrivelsen av hvordan reelle tall representeres på en datamaskin.
- Vi bruker 32 bit til å representere ett reelt tall.
- Det første bitet bruker vi til å representere **fortegnet**.
- De neste 8 bit bruker vi til å representere **eksponenten**.
- De siste 23 bit bruker vi til å representere **signifikanden**.

Litt om representasjon av reelle tall

- For **fortegnet** bruker vi 0 for + og 1 for –.
- For **signifikanden** bruker vi det 23-sifrede binære tallet som står bak komma.
 - Dette svarer til mellom 7 og 8 sifre i titallsystemet, så det er den nøyaktigheten vi regner med.
 - Avrundingsfeil vil da være i størrelsesorden 2^{-23} .
- For å representere **eksponenten**, bruker vi 8 bit.
 - Det gir oss mulighet til å fange opp $2^8 = 256$ forskjellige eksponenter.
 - Vi har bruk for å representere omtrent like mange negative som positive eksponenter.
 - Standard metode er at man representerer en eksponent ved summen av binærformen og 01111111_2 , hvor $01111111_2 = 127$.
 - Dette plasserer representasjonen av 0 omtrent midt i, og gir oss mulighet for å representere alle eksponenter fra -127 til 128 .

Litt om representasjon av reelle tall

Eksempel

Vi vil finne datarepresentasjonen av tallet $2,5_{10}$

Først må vi skrive tallet på binær form:

$$2,5_{10} = 10,1_2.$$

Den normaliserte binære formen er

$$0,101 \cdot 2^2.$$

Litt om representasjon av reelle tall

Eksempel (fortsatt)

Da vil vi bruke 0 for å representere fortegnet, 10000001 for å representere eksponenten og 101000000000000000000000 til å representere signifikanden.

$2,5_{10}$ representeres da av bitsekvensen

01000000 11010000 00000000 00000000

fordelt på 4 byte.

Litt om representasjon av reelle tall

Eksempel

Finn representasjonen av

$$-\frac{1}{3}.$$

Binærformen med 23 gjeldende siffer er

$$\left(-\frac{1}{3}\right)_{10} = -0,010101010101010101010101_2.$$

Normalisert binær form blir da

$$-0,10101010101010101010101 \cdot 2^{-1}.$$

Litt om representasjon av reelle tall

Eksempel (fortsatt)

Da må vi bruke

- 1 for å representere fortegnet.
- 01111110 for å representere eksponenten.
- 10101010101010101010101 for å representere signifikanden.

Representasjonen, fordelt på 4 byte, blir da

10111111 01010101 01010101 01010101.

Kapittel 4: Logikk

Kapittel 4 i læreboka gir en kort innføring i viktige deler av logikken.

Hvorfor skal informatikkstudenter lære noe om logikk?

- von Neumanns konstruksjon av datamaskinen er basert på utsagnslogikk og logiske porter.
- Grunnarkitekturen av datamaskiner har ikke endret seg mye siden den tid, selv om de teknologiske forbedringene har vært enorme.
- Vi bruker språket fra logikk til å formulere forutsetninger P i kontrollstrukturer som

If P then ... else ...

- Vi bruker logikk for automatisk å sikre at forskjellige data lagt inn i en base er forenlige med hverandre.
- Vi trenger logikk for å kunne definere hva som menes med korrekthet av et program.
- Kunstig intelligens, sikkerhetsprotokoller for utveksling av informasjon og mye annen avansert bruk av datamaskiner bygger på logikk.

Logikk

- Logikk, slik vi kjenner faget i dag, har sin opprinnelse fra gresk filosofi og vitenskap.
- Tanken var at et resonnement må kunne stykkes opp i enkelte tankestrøg, **grunnresonnement**, hvor det vil være lett å bestemme om grunnresonnementene er riktige eller representerer feilslutninger.
- Da kan man finne ut av hvilke deler av et argument som baserer seg på ren tankekraft, og hva som baserer seg på unevnte forutsetninger.
- Aristoteles formulerte en del **sylogismer** som skulle gi oss de lovlige logiske slutningene.
- Vi skal ikke gå inn på den klassiske syllogismelæren, men etterhvert legge grunnlaget for den.

Eksempel

- a) Jeg rekker ikke middagen.
Du kommer senere hjem enn meg.
Altså rekker ikke du middagen.
- b) Jeg rekker ikke middagen.
Hvis jeg ikke rekker middagen, rekker ikke du middagen.
Altså rekker ikke du middagen.

I eksempel a) har vi en skjult forutsetning, mens i eksempel b) er argumentet logisk sett riktig. Vi kan skrive dette argumentet om til et annet, hvor b) holder, men a) ikke holder.

Enda et eksempel

Eksempel

- a) Jeg liker ikke Bamsemums.
Du liker alt jeg liker.
Altså liker ikke du Bamsemums.
- b) Jeg liker ikke Bamsemums.
Hvis jeg ikke liker Bamsemums, liker ikke du Bamsemums.
Altså liker ikke du Bamsemums.

Her er det opplagt en feil i argument a), mens b) holder fortsatt. Det vil være vanskelig å programmere en generell [argumentsjekker](#) som vil godta a) i det første eksemplet, men ikke i det andre.