MAT1030 – Diskret matematikk

Forelesning 4: Tall som data

Dag Normann

Matematisk Institutt, Universitetet i Oslo

23. januar 2008

Valg av kontaktpersoner/tillitsvalgte

- Før vi tar pause skal vi velge to til fire tillitsvalgte/kontaktpersoner.
- Kontaktpersonene skal være med på å utforme midtveisevalueringen av MAT1030.
- Vi vil også samle kontaktpersoner, foreleser, plenumsregner og/eller gruppelærerne for gjensidige tilbakemeldinger når noen ønsker det.
- Vi kan foreta valget nå eller de siste minuttene av første time.

Oppsummering av kapittel 2

I Kapittel 2 lærte vi om tall i alternative tallsystemer, i hovedsak om binære tall, oktale tall og heksadesimale tall.

Det man må kunne i tilknytning til Kapittel 2 er:

- Skrive hele tall og desimaltall på binær form.
- Finne verdien av tall skrevet på binær form.
- Utføre enkle addisjons-, subtraksjons- og multiplikasjonsstykker i binær representasjon.
- Kjenne definisjonene av oktale tall og heksadesimale tall.

Representasjon av hele tall

Mandag så vi på hvordan vi kan representere de hele tallene fra -128 til 127 ved å legge informasjon inn i en byte på 8 bits.

I boka bruker man 2 bytes og i virkeligheten gjerne 4 bytes til å representere et enkelt tall.

Prinsippene og begrunnelsen for de valg man treffer vil være de samme.

Representasjon av hele tall

Definisjon (Representasjon av hele tall)

Hvis vi har k bits 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 $2^{k-1} + a$

Representasjon av hele tall

Eksempel (Noen representasjoner med $k = 8$)

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.

La $b = -99$. Første bit må være 1.
$128 + (-99) = 29 = 11101_2$
Vi skal bruke 7 sifre, så vi bruker at

$$11101_2 = 0011101_2.$$

Det gir representasjonen 10011101 for b.

Representasjon av hele tall

Dette synes som en tungvint måte, og det er det.
Hvis vi prøver å gjennomføre de samme eksemplene ved å regne binært, ser vi en morsom effekt:

Representasjon av hele tall

Eksempel (En alternativ måte)

La $a = -23$.
$23 = 00010111_2$
Vi vil regne ut 128 - 23 binært.

$$1000000_2 - 00010111_2 = 01101001$$

så binærkoden til $128 - 23$ er 1101001.
Representasjonen av -23 blir da 11101001.

La $c = -32$
$32 = 100000_2$, og $128 - 32$ binært blir

$$10000000_2 - 100000_2 = 01100000_2.$$

Representasjonen av -32 blir da 11100000

Representasjon av hele tall

Prosessen med å finne representasjonen av $-n$ fra binærformen til n kan beskrives enda enklere, ved det som boka kaller 2-er komplementet til binærformen:(Vi bruker tilnærmet samme betegnelse som boka, selv om det ikke blir god norsk)

Definisjon

La $a_1 \cdots a_k$ være en sekvens av 0'er og 1'ere, med en 0 lengst til venstre, og med minst ett 1-tall.
2-er komplementet til sekvensen får vi ved å starte fra høyre, lese ett og ett tall og

- Alle 0'er til høyre beholdes.
- Første 1-tall fra høyre beholdes.
- Alle tall til venstre for første 1-tall endres fra 0 til 1 eller fra 1 til 0.

Representasjon av hele tall

Hvis $x_1 \cdots x_8$ er datarepresentasjonen av et positivt heltall n , er 2-er komplementet representasjonen av $-n$.

Dette gjelder selvfølgelig om vi bruker 16-bits representasjoner eller 32-bits-representasjoner også.

Representasjon av hele tall

Oppgave

Finn en pseudokode for å beregne 2-er komplementet til en sekvens av lengde 8, av lengde 16 og av lengde 32.

Oppgave

Hvis vi tar 2-er komplementet av 2-er komplementet av en sekvens $x_1 \cdots x_k$ av nuller og enere, får vi den opprinnelige sekvensen tilbake. Forklar hvorfor.

Representasjon av hele tall

Vi har valgt å representere hele tall som sekvenser av bits av fast lengde slik at

- De ikke-negative tallene vi representerer er representert ved sin binære form.
- Hvis vi kan representere a og b og $a + b$ finner vi representasjonen av $a + b$ ved å bruke standardalgoritmen for addisjon av binære tall på representasjonene til a og til b.
- Hvis vi kan representere a og b og $a - b$, finner vi representasjonen av $a - b$ ved å følge standardalgoritmen for subtraksjon av binære tall på representasjonene av a og b.

Representasjon av hele tall

Vi skal se på ett eksempel:

Eksempel (8 bits representasjon)

- Vi skal finne representasjonen av $23 + (-47) = -24$.
- Representasjonen av 23 er 00010111.
- 128 - 47 = 81 og binærformen til 81 er 1010001, så -47 er representert av 11010001.
- Binærformen til 47 er 00101111, og 11010001 er også 2's komplementet til 00101111.
- Legger vi sammen 00010111 og 11010001 som binære tall får vi 11101000.
- $1101000_2 = 64 + 32 + 8 = 104$
- Det betyr at 11101000 er representasjonen av $-(128 - 104) = -24$.

Representasjon av hele tall

Hva skjer hvis vi regner oss ut over de tallene vi har representert?

-128 har representasjon 10000000.
Dette er også binærtallet til 128.
Hvis vi legger 1 til 127 via representasjonen 01111111, får vi representasjonen 10000000 for -128.

I en viss forstand vil representasjonene identifisere 128 og -128, og tall-linjen erstattes av en tall-sirkel hvor vi regner rundt og rundt.

Disse betraktningene har liten direkte relevans for informatikk, og vi skal ikke forfølge dem videre.

Representasjon av reelle tall

Når vi skal representere reelle tall i en datamaskin har vi andre hensyn å ta enn når vi representerer hele tall.
Vi ønsker å representere tall slik at vi kan bruke maskinene til å regne på

- Avstander i verdensrommet.
- Vekt av himmellegemer (med beregning av gravitasjonskrefter).
- Værdata.
- Strekk- og brekkbelastninger på brokonstruksjoner.
- Molekyl- og atomvekter.
- Konstanter i forbindelse med massevirkningsloven.
- Mye, mye mer.

Representasjon av reelle tall

- Siden vi bare kan representere endelig mange reelle tall i en maskin, må vi regne med avrundede størrelser.
- I endel utregninger, eksempelvis under utarbeidelse av værprognoser, må vi gjenta enkelte operasjoner mange, mange ganger, og avrundingsfeilene kan vokse.
- Vi trenger derfor meget stor presisjon på de tallene vi kan representere, slik at det er god nok presisjon selv etter mange avrundinger.

Representasjon av reelle tall

Vi ønsker datamaskiner som arbeider fort og uten å bruke for mye plass, men som samtidig kan brukes til et vidt spekter av regneoppgaver.
I valg av hvordan vi representerer reelle tall i datamaskiner, må vi ta hensyn til følgende:

- Vi skal kunne representere både de positive og negative tallene og både de svært store (små) tallene og tall svært nært 0.
- Vi vil ha høy presisjon på de tallene vi representerer.
- Det skal finnes effektive algoritmer som "oversetter" de vanlige aritmetiske operasjonene til representasjoner.

Representasjon av reelle tall

For å forstå prinsippene for representasjon av 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 stedenfor å skrive

$$230850100000000000000$$

kan vi skrive

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

eller

$$2,308501E20$$

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

Representasjon av reelle tall

Vi vil skrive tallene på normalisert binær form:

- Et fortegn, $+$ eller $-$.
- Et binært desimaluttrykk på formen

  $$0,1\cdots$$

  kalt signifikanden.
- En eksponentdel

  $$2^e$$

  hvor e er et heltall.

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 antydet en større nøyaktighet enn det vi trolig har grunnlag for.

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

Representasjon av reelle tall

Vi har prosedyrer for å utføre aritmetikk på tall på normalisert binær form:

- Addisjon

  Om nødvendig,flytt komma i det ene tallet slik at vi får samme eksponent.

  Legg sammen signifikandene

  Normaliser resultatet om nødvendig.

- Multiplikasjon

  Multipliser signifikandene

  Adder eksponentene

  Normaliser om nødvendig.

Representasjon av reelle tall

Det finnes selvfølgelig tilsvarende prosedyrer for subtraksjon og divisjon.

Normalisert representasjon er like godt egnet til multiplikasjon og divisjon som til addisjon og subtraksjon.

Representasjon av reelle tall

Hvis man vil danne seg et inntrykk av hva disse binære eksponentene betyr i 10-tallsystemet, bruker vi formelen

$$2^k = 10^{k \cdot \log 2}$$

og minner om at $\log 2 = 0,301$ sånn omtrent.

Representasjon av reelle tall

Vi vil følge boka i beskrivelsen av hvordan reelle tall representeres på en datamaskin.
Et interessant aspekt er at vi ikke vil representere tallet 0 på maskinen.

- Vi bruker 32 bits til å representere ett tall.
- Det første bit'et bruker vi til å representere fortegnet.
- De neste 8 bit'ene bruker vi til å representere eksponenten.
- De siste 23 bit'ene bruker vi til å representere signifikanden.

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 siffre i ti-tallsystemet, så det er den nøyaktigheten vi regner med.

  Avrundingsfeil vil da være i størrelsesorden 2^{-23}.

  Siden maskinen ofte bare vil ignorere siffere for langt ute, og ikke forhøye på vanlig måte, blir avrundingsfeilen større enn man først kunne tro.
- Man kan bruke dobbel presisjon, det vil si 64 bits, til å representere et tall, hvis det er fare for for mange avrundingsfeil.

  Vi skal ikke se på detaljene her.

Representasjon av reelle tall

- For å representere eksponenten, bruker vi 8 bits.

  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.

Representasjon av reelle tall

Vi kunne brukt representasjonen vår av alle tall fra -128 til 127 som vi beskrev for "lekedatamaskinen", og det kunne endog vært en teknologisk sett bedre løsning, ettersom vi har bruk for å addere og subtrahere eksponenter.

Standard metode er imidlertid at man representerer en eksponent ved summen av binærformen og 01111111, 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.

Representasjon av reelle tall

Eksempel

Vi vil finne datarepresentasjonen av tallet $1,25_{10}$

Først må vi skrive tallet på binær form:

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

Den normaliserte binære formen er

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

Representasjon av reelle tall

Eksempel (fortsatt)

Da vil vi bruke 0 for å representere fortegnet, 10000001 for å representere eksponenten og 10100000000000000000000 til å representere signifikanden.

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

$$01000000\ 11010000\ 00000000\ 00000000$$

fordelt på 4 bytes.

Representasjon av reelle tall

Eksempel

Finn representasjonen av

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

Binærformen med 23 gjeldende siffer er

$$(-\frac{1}{3})_{10} = -0,01010101010101010101010_2.$$

Normalisert binær form blir da

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

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 8 bytes, blir da

$$10111111\ 01010101\ 01010101\ 01010101.$$

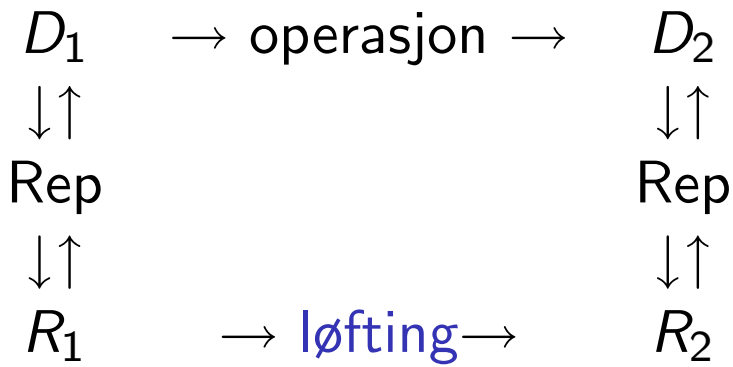
Representasjon av reelle tall

- De "reelle tallene" vi regner med i en datamaskin, kalles ofte flytende binære punkter.
- Det er vanlig å regne med slike flytende punkter med enkel eller dobbel presisjon.
- Ulempen er at man ikke har god kontroll på feilene som gjøres, det vil si, man vet ofte ikke hva usikkerheten i resultatet er.
- Det arbeides, på forskningsnivå, med andre tilnærminger til beregninger over reelle tall hvor man har bedre kontroll på usikkerhetene i svarene.

Generelt om representasjoner

Anta at D_1 og D_2 er to datatyper i "den virkelige verden".

Vi ønsker å finne effektive programmer for visse operasjoner som sender data av type D_1 til data av type D_2



Det er viktig at løftingen kan utføres raskt og uten bruk av for mye ressurser.

Det er ikke alltid slik at det bare er en måte å representere et objekt på, men en representant kan bare representere ett objekt.

OVER TIL 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

  $$\textbf{If } P \textbf{ then } \cdots \textbf{ else } \cdots$$

- 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 resonement må kunne stykkes opp i enkelte tankesprang, grunnresonement , hvor det vil være lett å bestemme om grunnresonementene 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 syllogismer som skulle gi oss de lovlige logiske slutningene.
- Vi skal ikke gå inn på den klassiske syllogismelæren, men etterhvert legge grunnlaget for den.

Logikk

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 eksembel b) er argumentet logisk sett riktig.
Vi kan skrive dette argumentet om til et annet, hvor b) holder, men a) ikke holder.

Logikk

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.

Logikk

Som en tommelfingerregel kan vi si at

> Et argument er logisk holdbart hvis vi kan bytte ut delformuleringer som ikke inneholder noe av den logiske strukturen med andre formuleringer uten at argumentet blir feil.

Hovedutfordringen blir å bestemme hva som tilhører den logiske strukturen og hva vi kan forandre på for å bruke testen over.

Logikk

Før vi starter på fagstoffet skal vi trekke frem fire navn som er viktige for utviklingen av logikk i det 20. århundre og for sammenfiltringen av matematisk logikk og teoretisk informatikk:

- Toralf Skolem
- Kurt Gödel
- Alan Turing
- John von Neumann