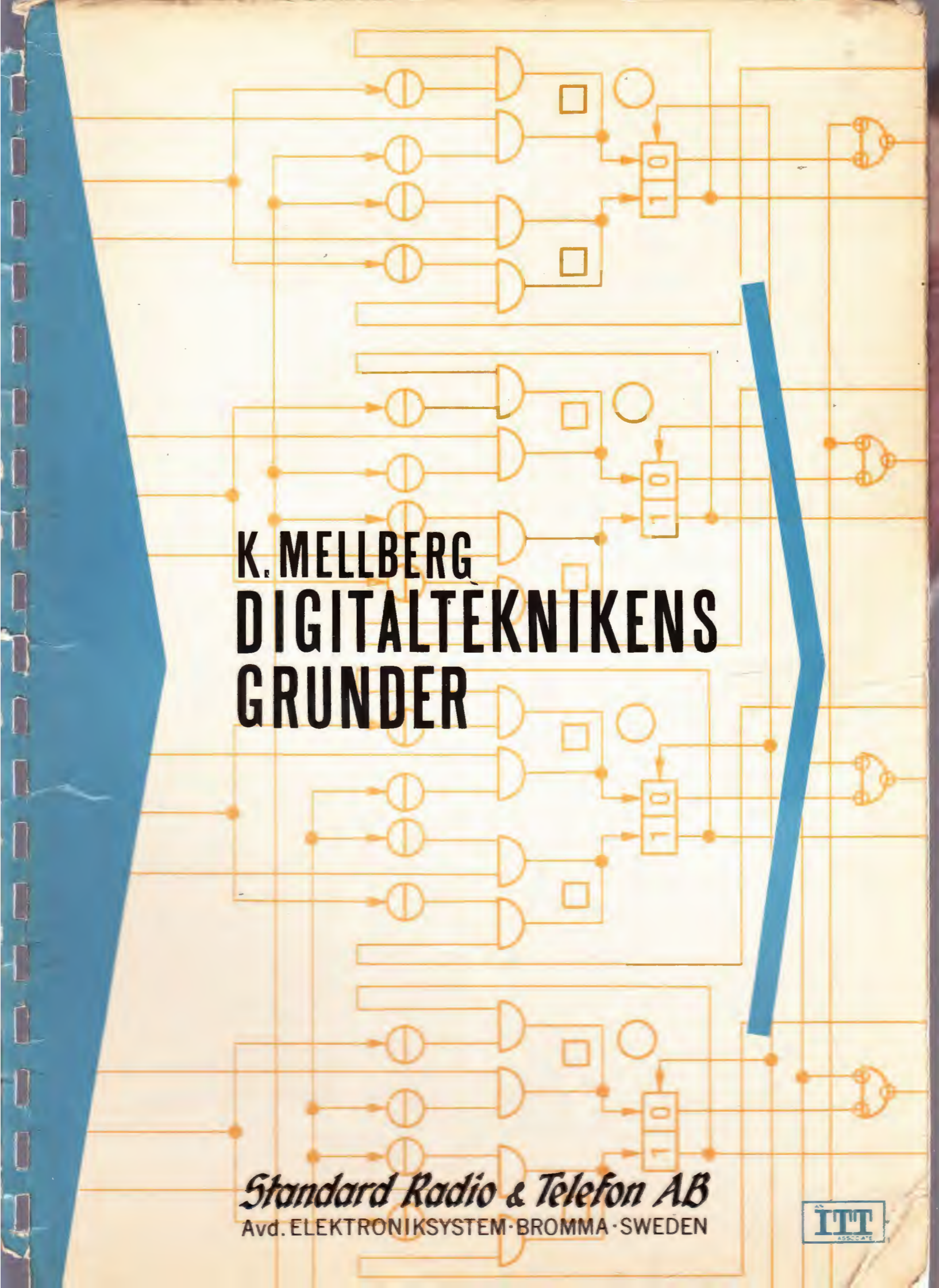


K. MELLBERG

# DIGITALTEKNIKENS GRUNDER

Standard Radio & Telefon AB
Avd. ELEKTRONIKSYSTEM · BROMMA · SWEDEN

Bengt Olofsson

# DIGITALTEKNIKENS GRUNDER

av

K. Mellberg

*Standard Radio & Telefon AB*

# INNEHÅLLSFÖRTECKNING

# INLEDNING

Den moderna datamaskintekniken, som på senare år gripit så vitt omkring sig på de flesta områden, är i sin praktiska utformning inte mer än ett par årtionden gammal. Då avses snabba elektroniska s.k. digitala computers, dvs elektroniska sifferräknemaskiner (engelska digit = siffra). Principerna för de moderna räknemaskinerna utarbetades emellertid redan under första hälften av 1800-talet av den engelske matematikern Charles Babbage. Den dåvarande teknologin var dock för outvecklad för att kunna omsätta idéerna i praktiken. Emellertid får man inte tro, att det är först under de senare åren som människan utnyttjat sig av mer eller mindre automatiska hjälpmedel för sina invecklade och tidskrävande räknebehov. Före de moderna siffermaskinerna fanns analogimaskiner av skilda typer, rent mekaniska, elektromekaniska och på senare tid elektroniska. Som namnet anger, räknar sådana maskiner inte med siffror för beräkningsstorheterna utan med analoga storheter oftast i form av kontinuerliga mekaniska rörelser eller elektroniska spänningar och strömmar. Trots att analogimaskinerna i många fall är rätt komplicerade till sin funktion, var det i synnerhet beträffande mekaniska och elektromekaniska typer lättare att med den tidigare teknologin realisera dessa än siffermaskinerna. Detta berodde till stor del på att materialuppbådet och därmed komplexiteten i uppbyggnaden blev mindre för en analogimaskin än för en siffermaskin, om användbarheten skulle vara något så när jämförlig.

För enklare typer av sifferberäkningar exempelvis additioner och multiplikationer fanns redan på 1600-talet räknesnurror av digital typ. Eftersom operatören själv fick lov att styra räkningarna, kan dessa snurror dock endast anses som halvautomatiska hjälpmedel om än avancerade sådana. Ännu längre tillbaka i tiden fanns ett enklare digitalt hjälpmedel nämligen den i medelhavsländerna och Österlandet använda kulramen, av romarna benämnd "abacus" (se fig. 1.1). I sin tidigaste form utgjordes abacusen av en tavla med skåror eller rännor, i vilka man lade små marmorkulor för att beteckna antal. Här kan vi också spåra ursprunget till begreppet kalkyl. Det latinska ordet för en stenkula enligt ovan är nämligen "calculus" (calc = kalk, kalksten; calculus = liten kalksten, marmorkula).

En mer automatiserad kulräknare och en kuriositet med modern anknytning var romarnas "hodometer" eller vägmätare, en föregångare till våra dagars taxameter. I det gamla Rom kunde man nämligen ibland hyra hjulfordon försedda med en bleckbehållare med roterande överdel, kopplad till hjulen. Över-

delen, som innehöll små stenkulor, roterade i takt med hjulen och släppte för varje varv ner en stenkula i behållaren genom ett hål i överdelen. Med hjälp av antalet nedsläppta kulor kunde då hyran beräknas efter färdens slut.

Hodometern var ett enkelt automatiskt räkneverktyg jämfört med senare tiders analogimaskiner, som ställer stora krav på precisionen hos de arbetande funktionerna, exempelvis växlar och integrerande skivor i det mekaniska fallet och motstånd och spänningar i det elektriska. Givetvis var hodometerns räkneuppgift av ett mycket enklare slag än de senares, men man kan ju leka med tanken att låta en eller flera enkla på något sätt hopkopplade kulräknare utföra de additioner och subtraktioner, i vilka man alltid kan upplösa de mest komplicerade räkneproblem. Givetvis skulle det för ett större problem åtgå en massa tid och en mängd räknare för ernående av ett resultat, men frånsett praktiska begränsningar därvidlag är det en fullt möjlig väg att gå. Till och med kan vissa typer av räkningar utföras med den enkla räknaren, där de komplicerade analogisystemen är oanvändbara. Tag som ett enkelt exempel, att en enda enhet skall dras från en miljon. Har man bara en miljon stenkulor, är det en enkel sak att ta bort en kula och sedan iakttaga förändringen. Att däremot ändra exempelvis en analogspänning med en miljondel, så att denna ändring är iakttagbart bestående, är en praktiskt omöjlig uppgift, eftersom man i praktiken åtminstone inte f. n. kan hålla spänningar konstanta med en sådan noggrannhet. En analogimaskin kan alltså inte användas för t. ex. folkräknings- eller bokföringsarbeten, där små enheter av det hela skall kunna behandlas, dvs där stor noggrannhet erfordras. Däremot kan kulräknaren och andra siffermaskiner användas för sådana ändamål, om de förses med tillräckligt stort talområde. Användningsmöjligheterna inom kontorsbranschen har också starkt bidragit till det uppsving, som siffermaskinerna rönt sedan de blev praktiskt producerbara, ett uppsving som även andra användningsområden dragit nytta av.

För att fortsätta med den enkla kulräknaren kunde man tänka sig att använda den i ett system för komplicerade beräkningar av den anledningen, att den arbetar med mycket enkelt reproducerbara fysikaliska tillstånd, som endast medger att den räknar en enhet i taget. Detta innebär, att systemet kan arbeta tillförlitligt även vid långa räkneserier, eller då många räknare arbetar samtidigt. Vad som gör systemet praktiskt oanvändbart är, att det är oerhört långsamt eller också oerhört skrymmande samt att det kan slitas ut på en relativt kort tid. Det är emellertid intressant att se, att då man på senare tid kommit fram till elektroniska anordningar som medger en avsevärd reduktion av ovannämnda nackdelar har man kunnat bygga mycket komplicerade automatiska siffermaskiner just genom att återgå till de utpräglat enkla räknemetoderna. Dessa tillåter nämligen, att delarna i de komplicerade systemen var för sig

kan arbeta efter mycket enkla fysikaliska principer, vilket är förutsättningen för att de skall kunna fungera tillförlitligt i ett komplext system. Utmärkande för de moderna siffermaskinerna är därför, att vid lösningen av komplicerade räkneproblem en mångfald i och för sig mycket enkla beräkningar görs mycket snabbt i ett stort antal enkla elektroniska kretsar med små fysikaliska dimensioner. Det är främst tack vare de senare årens utveckling av de snabba, små och driftsäkra transistorerna, som siffermaskinerna fått allt större användningsmöjligheter, inte enbart för kontorsbruk utan även för mycket snabba processregleringar, där analogsystem förut varit dominerande. En starkt bidragande orsak till siffermaskinernas popularitet är deras förmåga att genom programmering fås att själva bestämma sitt arbetssätt utan inblandning av mänskligt initiativ, vilket kraftigt höjer deras effektivitet.

## KAPITEL 1. TALREPRESENTATIONENS HISTORIA

Som tidigare nämnts, använder sig de moderna sifferräknemaskinerna av mycket enkla räknemetoder. Innan man går in på dessa kan det kanske vara roande och intressant att se, hur aritmetiken utvecklat sig genom tiderna för att slutligen bli gripbar även för själlösa maskiner.

### 1.1. Uppkomsten av siffror

Hur långt tillbaka i tiden som människan började räkna, vet man inte. Man vet endast att det var mycket länge sedan, långt innan några skrivtecken över huvud taget fanns och förmodligen också innan särskilda ord för tal uppfunnits. Studier av primitiva folkslag har lett till antagandet, att även de allra tidigaste urmänniskorna insåg betydelsen av att skilja mellan en och många och förmodligen också verkligen kunde skilja mellan många och lägre antal som en, två, tre och kanske fyra. Med stor säkerhet vet man däremot, att våra förfäder tidigt använde sina fingrar dels som minneshjälp vid räkning och dels för meddelande av tal till andra, innan några särskilda räkneord fanns. Ganska säkert var också räknandet till en början begränsat till antalet fingrar eller hos vissa folk till antalet fingrar plus antalet tår. En påminnelse om fingerräknandet får man av det latinska ordet för finger, digitus, som dessutom fått betydelsen "siffra" och går igen i benämningen digital (digitalsystem = siffersystem).

Då behovet uppstod för angivande av större mängder än antalet fingrar eller tår medgav, hittade man på ett slags grov-finsystem. För att ange exempelvis

antalet 23 kunde man öppna och sluta båda händerna två gånger (grovangivelse) och därefter visa upp tre fingrar (finangivelse). Detta förfaringssätt kan med vårt moderna skrivsätt anges med uttrycket 2 x 10 + 3. På ett naturligt sätt hade alltså en decimal (av latinska decem = tio) talbildning uppstått redan på detta tidiga stadium, och den har levt kvar och bildat grunden till vårt moderna decimalsystem.

När människan började skaffa sig ägodelar, uppstod så småningom behovet att uppteckna deras antal som stöd för minnet. Den enklaste formen av skrivtecken för sådan uppteckning var ett streck eller annat enkelt tecken för varje enhet i det räknade antalet, en metod, som kan härledas ur användningen av stickor eller utskurna skåror för tal. För större talangivelser införde man även här ett grov-finsystem på samma sätt som vid teckenspråket. Därav kom det sig, att man bildade särskilda tecken för antalet fingrar på en hand, på två händer och, då behov uppstod, även för större enheter. Många varianter av detta skrivsätt uppstod hos olika folk, men metoden går direkt igen i den för de flesta välbekanta romerska sifferskriften (egentligen etruskisk till ursprunget), där exempelvis grovangivelserna V, X och L motsvarar talen 5, 10 och 50, vilka är multipler av antalet fingrar på en hand. Talet 23 blir med denna skrift XXIII, vilket är uppbyggt exakt efter den ovannämnda fingerspråkstekniken. Hos exempelvis grekerna använde man sig också av alfabetets tecken som siffror, men där använde man olika bokstäver för varje ental, tiotal och hundratal.

De som i antiken utbildade de olika sifferskriftsystemen hade ingen tanke på att kunna använda dem för aritmetiska beräkningar. Man nöjde sig med att kunna ange antal och försökte sällan att exempelvis lägga ihop olika tal med sifferskriftens hjälp. Sådana operationer utfördes i stället med mekaniska hjälpmedel, av vilka de viktigaste var olika utvecklade former av den tidigare omnämnda abacusen. Man ansåg sig därför ha full frihet att utveckla en skrift, som enkelt angav stora tal, och åsatte talmängderna namn i form av bokstavsförkortningar, som inte visade något inbördes matematiskt samband. Kända exempel från den senare romerska sifferskriften är därvid tecknen för 100 och 1000, dvs C (av lat. centum = hundra) och M (av lat. mille = tusen). Att dessa inte visar något matematiskt samband, inser man, om man på papperet försöker utföra divisionen M/C vilket inte är mycket lättare att göra än om den skrivs i ordformen "tusen genom hundra". Ytterligare ett exempel på icke-matematiska bildningar är tecknet för exvis 900 dvs CM (jfr analoga uttryck som "klockan kvart i fem"), vilket är lättare att skriva än alternativet DCCCC men sätter stopp för alla enkla räkneregler. Visserligen lärde man sig så småningom att räkna addition och subtraktion med den romerska skriften och den blev utmärkt lämpad för dessa operationer, då uttryck av formen

CM skrevs i formen DCCCC. Den användes därför också inom kommersiell bokföring i Europa till långt efter medeltiden.*) Däremot var multiplikationer och divisioner svåra att utföra, liksom bevis av matematiska satser. Under antiken stagnerade också den rena aritmetiken, medan geometrin blomstrade.

## 1.2. Nollans revolution

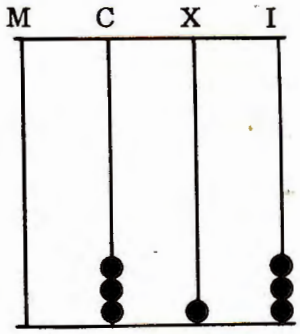
Antikens greker och romare hade alltså låst sig fast i en icke-matematisk sifferskrift, redan innan behov av komplicerade uträkningar uppstod. Värst stod det till hos grekerna, vilkas stora filosofer med förkärlek lade in magiska betydelser i sifferteckнen. Därmed fick de ännu svårare att se de matematiska sambanden mellan talen. På ett annat ställe i världen, i Indien, hade den sociala utvecklingen gått en annan väg. Den matematiskt inriktade grenen av filosofin utvecklades långsammare än i Europa. Då behoven av praktisk räkning ökade i samband med handelns tillväxt, fanns det därför inget av filosofin mystifierat sifferspråk som hos grekerna. Liksom de senare använde sig indierna av olika tecken för exempelvis varje ental, men dessa tecken ingick inte i bokstavsalfabetet utan var speceilla och alltså siffror i verklig bemärkelse. Ett par av dessa tecken tycks ha sin upprinnelse i de primitiva streckteckнen. I Österlandet angav man ursprungligen antal med hjälp av tecken bildade av vågräta streck. Siffran två skrevs då = och tre ≡. I snabbskrift övergick dessa tecken småningom till z eller ƨ resp. ƺ eller ȝ, i vilka vi lätt känner igen våra egna siffror 2 resp. 3.

Med ett sådant siffersystem är det naturligt, att man använder särskilda symboler även för antalet tiotal, hundratal osv. i stället för att som romarna upprepa ett fåtal tecken (jfr LXXXIII = 83). I Österlandet såväl som på andra ställen i världen använde man ursprungligen räknetavlor av abacustyp för att utföra beräkningar i praktiken. Så gott som överallt var dessa baserade på decimalsystemet, dvs det fanns en kolumn för ental, en för tiotal osv. Romarna betecknade dessa kolumner med motsvarande bokstäver, alltså I för ental, X för tiotal osv. Vid upptecknande av ett tal inställt på abacus skrev man då helt enkelt lika många sådana tecken som det fanns kulor i motsvarande kolumn (se fig. 1.1). I senare former minskades antalet kulor och även antalet tecken för

---

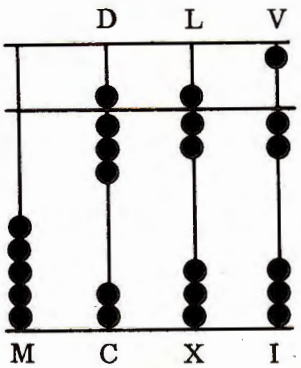
*) Följande exempel visar enkelheten i exempelvis en addition av två tal. Man behöver endast summera antalet olika tecken i de två talen och känna till sammanslagningar av typen V + V = X.

| | | |
|---|---|---|
| CCCL | XXVII | (377) |
| C | XV I | (116) |
| CCCCLXXXXIII | | (493) |




Inställt tal = CCCXIII (= 313)

Fig. 1.1




Inställt tal = DCCCLXXII (= 872)

Fig. 1.2

ett visst tal (fig. 1.2) genom användande av mellantecknen V, L osv. Enligt indiernas siffersystem borde man i stället skriva särskilda tecken för visst antal kulor i viss kolumn, dvs två kulor i entalskolumnen angavs med tecknet för två, två i tiotalskolumnen med tecknet för tjugo osv. Eftersom det i båda fallen gällde samma antal kulor fast i olika kolumner, insåg man förmodligen, att en förenkling skulle kunna göras genom användande av enbart entalstecknen och låta dessa teckens plats i talet bestämma deras talvärde på samma sätt som i räknetavlan (positionssystem). För att komma ifrån sådana tvetydigheter som att exvis 32 kunde betyda såväl 32 som 302, tvingades man av praktiska skäl att införa ett särskilt tecken för en tom kolumn i räknetavlan. Detta tecken, nollan, som utgjordes av en punkt innan det så småningom antog nuvarande form, infördes i Indien ungefär vid vår tideräknings början, en händelse som väl torde kunna betecknas som en av de viktigaste i matematikens historia. I och med att nollan infördes, blev talen sanna avbildningar av räkneramens kolumner. Behovet att använda räkneramen försvann därför snabbt, eftersom det gick lika bra att räkna med papper och penna.

Via Arabien och morerna i Spanien fördes det indiska siffersystemet till Europa, där det emellertid tog lång tid innan den romerska sifferskriften utträng-

des. För handelsräkning och dylikt var fördelarna med det nya systemet kanske inte heller så uppenbara som för den teoretiska matematiken. Inom det senare området anammades systemet emellertid snabbt och matematikens utveckling som vetenskap tog fart. En av de uppenbara fördelarna med det indiska systemet är, att man slipper ifrån ett särskilt tecken för tio ental, som i stället skrivs 10, dvs 1 tiotal plus 0 ental. Därigenom slipper man även de särskilda tecknen för övriga jämna tiotal och man kan med hjälp av endast tio siffror skriva vilket tal som helst, varvid en siffras plats i talet anger siffrans talvärde (positionssystem). I vårt decimalsystem är alltså exvis en trea i första positionen (längst till höger) värd tre ental eller 3 x 1, i andra positionen tre tiotal 3 x 10, i tredje positionen tre hundratal 3 x 100 eller 3 x 10 x 10 osv. För varje steg åt vänster ökar alltså positionsvärdet tio gånger. Värdet tio är karakteristiskt för talsystemet och kallas detta b a s .

Ett positionssystem framhäver på ett tydligt sätt relationerna mellan olika tal. Så t.ex. är uttrycken 2 och 200 inte enbart benämningar för vissa antal, utan de visar också att det sista talet är hundra gånger så stort som det första. Sådana talrelationer, som har med faktorer och kvoter att göra, var mycket svåra att handskas med, innan positionssystemet kom till, och det var därför först då som t.ex. serieutvecklingar och infinitesimalkalkyler kunde tas i bruk som oundgängliga instrument för den högre matematikens utveckling.

## 1.3. Ett sifferspråk för maskiner

Som berörts ovan, är t.ex. decimaltalet 457 en förkortad beteckning för mängden $4 \cdot 100 + 5 \cdot 10 + 7 \cdot 1$ eller $4 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0$. Rent allmänt kan ett decimaltal med n siffror skrivas:

$$D = a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2} + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0 \qquad \ldots 1.31$$

Koefficienterna $a_i$ kan därvid väljas ur decimalsystemets tio siffror 0 – 9. Nollkoefficienterna för högre potenser utskrivs ej.

Decimalsystemet enligt ekv. 1.31 har givit människan ett utomordentligt verktyg, med vilket hon har kunnat skapa sig en hög materiell standard med allt fler och sinnrikare hjälpapparater. Räkneproblem, som beredde antikens skarpaste hjärnor svårigheter, kan idag lösas av en skolpojke. Emellertid växte behovet av praktiska räkneoperationer till en sådan volym, att det ändå blev alltför tidsödande för mänsklig arbetskraft att klara av det. Man började därför snegla på möjligheten att konstruera hjälpmedel för detta räknearbete. I våra dagar har detta också realiserats i de moderna matematikmaskinerna. För att göra dessa så enkla och tillförlitliga som möjligt har man i allmänhet

gått in för att låta dem räkna i ett ännu enklare talsystem än det decimala, nämligen det **binära talsystemet**.

I stil med ekv. 1.31 kan helt allmänt ett godtyckligt n-siffrigt tal A i ett talsystem med en enhetlig bas B skrivas:

$$A = a_{n-1} \cdot B^{n-1} + a_{n-2} \cdot B^{n-2} + \ldots + a_1 \cdot B^1 + a_0 \cdot B^0 \qquad \ldots 1.32$$

där koefficienterna eller siffervärdena $a_i$ kan variera mellan 0 och B-1.

Ju större ett systems bas är, desto mindre antal sifferpositioner behövs för angivande av en viss mängd enheter. I gengäld blir antalet olika siffervärden $a_i$ större, vilket bl.a. försvårar additions- och multiplikationstabellen. Å andra sidan ger ett stort antal siffror i ett tal svårigheter för en mänsklig iakttagare att bilda sig en uppfattning om talets storlek. För mänskligt bruk behövs därför av praktiska skäl en kompromiss mellan stor och liten bas och detta kan man väl säga gäller ganska bra i det decimala systemet. Ty även om det i början verkar svårt, så har dock människan ganska lätt för att lära in den decimala multiplikationstabellen. Sedan är det en annan sak, att man i matematikmaskinernas tidsålder ibland funnit det olyckligt att vi har skapats med tio fingrar i stället för åtta eller sexton, vilka senare fall skulle ha undanröjt en del praktiska problem som längre fram skall beröras.

Inom matematikmaskintekniken eller allmänt inom den digitala databehandlingstekniken har man följande faktorer att beakta vid val av talsystem. Tal måste alltid tänkas representerade av olika fysikaliska tillstånd t.ex. olika lägen hos mekaniska delar eller olika elektriska tillstånd i strömkretsar. Ju fler olika siffervärden ($a_i$-värden) talsystemet för en maskin har, desto fler kombinationer måste maskinen kunna vid addition eller multiplikation av två siffror och desto mer komplicerad blir den. Något som ytterligare talar för ett fåtal siffervärden är, att enkelheten och tillförlitligheten hos en maskin är större ju färre olika fysikaliska tillstånd den behöver kunna anta. Det enklaste fallet utgörs därvid av ett "till/från"-system, dvs där de fysikaliska tillstånden endast är 2. Motsvarande talsystem kan då tänkas representerat så, att $a_i = 1$ motsvaras av "till" och $a_i = 0$ av "från". Ett sådant system kallas **binärt** talsystem (av latinska bis = tvåfaldig). Det närmast liggande tillämpningsexemplet har vi inom pulstekniken, där exempelvis närvaro av en spänningspuls kan beteckna $a_i = 1$ och frånvaro $a_i = 0$. De olika siffrorna i ett binärt tal kan därvid antingen komma efter varandra i tid i samma elektriska krets (seriemetod) eller finnas samtidigt i var sin krets (parallellmetod).

Inom nämnda pulsteknik skulle man även kunna tänka sig ett system med större bas, exempelvis 10, genom att låta varje koefficient $a_i$ (0-9) representeras av

a) en puls med tio lika amplituder

eller

b) tio olika platser i tid eller rum, varav för varje värde på $a_i$ endast en plats har puls.

Vad punkt a) beträffar är för det första flernivåselektiva kretsar oftast komplicerade och för det andra är det f.n. omöjligt att utan många och dyrbara komponenter undvika amplitudförändringar hos en puls, som passerar det vanligen stora antalet kretsar i en databehandlingsmaskin, varför risken för förväxlingar mellan närbelägna amplituder är stor. Den bästa säkerheten uppnås tydligen om amplituderna endast är två vitt åtskilda (full puls eller ingen puls alls). Amplitudförsämrade pulser kan då också lätt återbildas i enkla förstärkare. Resonemanget har alltså fört tillbaka till ett binärsystem.

Vad punkt b) beträffar behövs som nämnts tio pulsplatser antingen i tid eller rum för att representera talen 0-9. I ett binärt system däremot erfordras i medeltal endast något mer än tre siffror eller pulsplatser för samma sak ($2^2 + 2^1 + 2^0 = 7$, $2^3 + 2^2 + 2^1 + 2^0 = 15$). Det binära systemet visar sig alltså även här lämpligare än det decimala.

Med tanke på ovan anförda skäl och på det faktum, att databehandlingstekniken gör intrång på allt fler andra tekniska områden, är det väl motiverat för teknikern i allmänhet och databehandlingsteknikern i synnerhet att sätta sig in i och vänja sig vid binärsystemet som sitt andra sifferspråk. Den närmaste fortsättningen avser därför att ge läsaren en viss förtrogenhet med nämnda system, som tack vare fåtalet koefficienter (0 och 1) är betydligt enklare än decimalsystemet och även kan uppvisa en del matematiska fördelar gentemot det senare.

## KAPITEL 2. BINÄR RÄKNING

### 2.1. Allmänt

Ett godtyckligt binärt tal kan enligt ekv. 1.32 skrivas:

$$A = A_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \ldots + a_1 \cdot 2 + a_0 \cdot 1 \qquad \ldots 2.11$$

där koefficienterna $a_i$ kan anta värdena 0 och 1. Det förut angivna decimaltalet blir uttryckt i potenser av $2 = 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2 + 1 \cdot 1$, och om man endast skriver ut koefficienterna $a_i$ får man det mot 457 svarande binärtalet 111001001, där som i decimalsystemet den högsta (mest signifikanta) siffran skrivs längst till vänster. För att

uttrycka det tre-siffriga decimaltalet 457 erfordras sålunda nio binära siffror (jfr under 1.3 ovan om sambandet mellan basstorlek och sifferantal). Den vedertagna benämningen för en binär siffra är en bit (svensk plur. bitar), som är en förkortning av det engelska "binary digit" (binär siffra). För att ange talet 457 erfordras alltså nio bitar.

Som synes är skillnaden mellan decimal- och binärsystem den, att man i det förra räknar med ental, tiotal, hundratal osv, medan man i det senare räknar med ental, tvåtal, fyrtal osv.

Eftersom decimal- och binärsystemen är uppbyggda efter samma princip (enl. ekv. 1.32) bör också de matematiska operationer som gäller för decimaltal även gälla för binärtal. Samma förfaringssätt kan således i båda fallen användas för de fyra enkla räknesätten.

## 2.2. Addition

### 2.2.1. Normal addition

Låt oss ta två godtyckliga binära tal för enkelhets skull med ett fåtal siffror exempelvis 1101 och 110, vilka snabbt beräknas motsvara 13 respektive 6 i decimalform. Sedan adderar vi dessa två tal på vanligt sätt kolumn för kolumn med början vid den minsta siffran (till höger). De två första kolumnerna bereder inga svårigheter, ty självklart är 1 + 0 = 1 och 0 + 1 = 1.

| | |
|---|---|
| Minnessiffror | 1 1 |
| | 1 1 0 1 |
| | + 1 1 0 |
| Summa | 1 0 0 1 1 |

I tredje kolumnen skall 1 och 1 adderas. Summan i decimalform blir ju 2, vilket i binär form motsvaras av $10_2$. Eftersom endast en siffra kan skrivas ned i kolumnens summadel, måste ettan föras över till fjärde kolumnen som minnessiffra på samma sätt som i decimalsystemet. Skillnaden är den, att i det senare systemet ingen minnessiffra bildas förrän summan i en kolumn överstiger 9, medan i det förra det sker så fort summan överstiger 1 (eftersom 1 är den största möjliga siffran i varje kolumn). I fjärde kolumnen skall således åter 1 och 1 adderas, varför även där 0 erhålles i summadelen och 1 i minne till femte kolumnen. I denna summeras 1 och 0, vilket ger 1, och slutresultatet 10011 erhålles, vilket mycket riktigt motsvarar 19 = 6 + 13.

Givetvis kan addition med mer än två tal utföras, varvid dubbla minnessiffror kan uppstå. Tag som exempel additionen av de tre talen 111 (7), 1111 (15) och

110 (6). Första kolumnen ger 1 i minne till andra kolumnen. I denna adderas 1 + 1 + 1 + 1 = 100 (4). Detta ger 0 i summadelen, 0 i minne till tredje kolumnen och 1 i minne till fjärde. Från addition i tredje kolumnen (1 + 1 + 1 = 11)(3) erhålles ytterligare 1 i minne till fjärde, som således fått minnessiffra två gånger.

| | | |
|---|---|---|
| | 1 | |
| Minnessiffror | 1 1 0 1 | |
| | 1 1 1 | |
| | 1 1 1 1 | |
| | 1 1 0 | |
| Summa | 1 1 1 0 0 | (28 = 7 + 15 + 6) |

## 2.2.2. Halvaddition

Den operation som utföres då man adderar siffrorna i en kolumn som i 2.2.1. och tar med eventuell minnessiffra från lägre position kallas fullständig addition eller heladdition. Sålunda är exempelvis 1 + 1 = 10 och 1 + 1 + 1 = 11 heladditioner. Om man däremot inte adderar minnessiffrorna, kallas operationen halvaddition. Exempelvis är 1 + 1 = 0 och 1 + 1 + 1 = 1 halvadditioner. Man finner, att en halvaddition av jämnt antal ettor ger noll och av udda antal ettor ger ett i summa. Operationen är av viss betydelse inom databehandlingstekniken, varför benämningen är värd att lägga på minnet.

## 2.3. Subtraktion

### 2.3.1. Normal subtraktion

Även subtraktion med binära tal kan utföras på gängse sätt. Vi tar samma talexempel som 2.2.1. nämligen 1101 (13) och 110 (6) och drar det senare från det förra. Vi subtraherar på vanligt sätt i kolumn efter kolumn från höger.

| | | |
|---|---|---|
| Lån | 10 10 | |
| | ~~1~~ ~~1~~ 0 1 | |
| | − 1 1 0 | |
| Rest | 1 1 1 | |

Första kolumnen ger givetvis 1 − 0 = 1. I andra kolumnen (0 − 1) måste vi låna en enhet från tredje kolumnen, vilket motsvarar två enheter i andra, dvs i binär form 10. 10 − 1 ger 1 i rest. Efter lånet från tredje kolumnen är minuenden där noll och vi måste åter låna, denna gång från fjärde kolumnen. 10 − 1

ger 1 i rest i tredje kolumnen och i fjärde får vi 0 – 0 = 0. Totala resten blir alltså 111 vilket motsvarar 7 (13 – 6).

### 2.3.2. Negativa tal. Komplement

Då begreppet komplement och räkning med komplement för många troligen är helt nytt, behandlas dessa saker tämligen ingående i detta avsnitt. För att underlätta förståelsen för det nya begreppet är dessutom framställningen till en början baserad på decimalsystemet.

Låt oss skifta på talen i 2.3.1. och i stället utföra subtraktionen 6 – 13. En snabb överblick visar oss, att resten måste bli negativ, varför vi drar det mindre talet från det större och sätter minustecken före resten, dvs vi utför operationen –(13 – 6) = –7. Antag nu att vi har en matematikmaskin, som skall utföra samma operation. Den har emellertid inte möjlighet att överblicka problemet före lösandet. Detta skulle göra den för dyrbar eller för långsam. Den kan alltså inte veta, att resultatet blir ett negativt tal, utan börjar helt rutinmässigt att dra 13 från 6. Att dra 3 från 6 går nog bra, men då 1 skall dras från 0 blir det värre. Vi måste alltså ta till något knep för att lösa problemet, och det må ju vara tillåtet, så länge slutresultatet blir riktigt.

I stället för att lägga positiva och negativa tal omkring den verkliga nollnivån skaffar vi oss andra fiktiva nollnivåer, som är så höga, att vi med säkerhet undviker negativa tal vid någon som helst subtraktion. Alla tal ovanför en sådan nivå räknar vi som positiva tal och alla under får motsvara de negativa. Talen +6 och –7 skulle alltså med exempelvis en miljon som nollnivå skrivas 1000006 respektive 999993. Det senare talet har alltså ett negativt tals karaktär trots sin positiva form men naturligtvis endast under förutsättning att ett högre tal som exempelvis ovannämnda 1000000 räknas som noll. I själva verket behöver vi inte alls binda oss vid någon viss nollnivå, så länge den är tillräckligt hög för vårt syfte. Vi skriver därför +6 och –7 som ... 00006 respektive 99993 därmed angivande att det förra talet ligger sex enheter ovanför och det senare sju enheter under en lämplig nollnivå. Talet ... 99993 benämnes komplementet till talet 7 med avseende på nollnivån ifråga. I stället för det negativa talet använder vi oss alltså här av dess nollkomplement.

Antag för enkelhets skull, att vår maskin (som här räknar med decimala siffror) inte kan räkna med större tal än tresiffriga sådana. Området ... 00000 – – ... 00999 motsvarar således de positiva talen och området ... 99999 – ... ... 99000 de negativa. Vi ser då, att de positiva talen karaktäriseras av att de innehåller nollor fr.o.m. fjärde siffran och uppåt medan de negativa innehåller nior i motsvarande positioner. För att ange maskinens hela talområde

behövs därför strängt taget endast fyra siffror, varav den högsta anger om ett tal är positivt eller negativt. Man kallar därför denna siffra teckensiffra. Våra tal +6 och −7 skriver vi nu 0 006 respektive 9 993, där understrykningen markerar teckensiffran. 9 993 är härvid till formen ($10^4$ − komplementet till 7 ($10^4 - 7$).

Låt oss se, om det nya beteckningssättet kan hjälpa oss att lösa problemet att dra 13 från 6 i vår maskin. Enligt ovan skrives talet 6 på maskinspråket som 0 006 och −13 som 9 987. Om vi nu tänker på att 9 987 i sig självt motsvarar ett negativt tal, inser vi att 0 006 och 9 987 bör adderas för att den önskade subtraktionen skall erhållas. Subtraktion ersättes alltså av addition med komplement. Resultatet blir 9 993, som ju motsvarar det negativa talet 7.

$$\begin{array}{r} 0\,0\,0\,6 \\ 9\,9\,8\,7 \\ \hline 9\,9\,9\,3 \end{array}$$

Låt oss till resultatet lägga det positiva talet 7, så får vi 0 000 = 0, eftersom ettan i femte positionen inte ryms i maskinen. Den fiktiva nollnivån kommer

$$\begin{array}{r} 0\,0\,0\,7 \\ 9\,9\,9\,3 \\ \hline 1\,0\,0\,0\,0 \end{array}$$

alltså aldrig till synes i maskinen, varför vi alltid får fullt riktiga räkneresultat.

Nu kanske den samvetsgranne läsaren lägger ihop två sådana negativa tal som exempelvis 9 405 och 9 516 och får resultatet 8 921, som närmast inte liknar någonting. Vad betyder exempelvis åttan i teckenpositionen? Jo, den betyder helt enkelt att befogenheterna är överskridna. 9 405 och 9 516 motsvarar −595 respektive −484. Summan av dessa negativa tal blir −1079. Men komplementtalens utseende anger, att det använda talområdet endast omfattar tre siffror, varför högsta siffran i resultatet inkräktar på teckensiffrans plats. På samma sätt erhålles en etta i teckenpositionen, om två för stora positiva tal adderas. För sådana operationer är tydligen talområdet för litet och bör utökas med en siffra.

Efter denna genomgång kan det vara dags att lämna decimaltalen och i stället titta på komplementbildning av binärtal. Vi antar därför, att vår maskin arbetar med binära siffror och att dess talområde för enkelhets skull är begränsat till fyra bitar (se 2.1.). Det innebär enligt det föregående, att ett tal i denna maskin bör anges med fem bitar, av vilka den femte och högsta användes som

teckensiffra. Talet 101 (5) blir således på maskinspråket $\underline{0}$ 0101, om det är positivt. Om det är negativt, skall det dras från en nollnivå. Detta sker med vanlig subtraktion, genom att man lånar från en tänkt etta i en högre position (den fiktiva nollnivån) efter samma princip som vi visat för decimaltal. Komplementet visar sig bli $\underline{1}$ 1011. För kontrollens skull lägger vi till det positiva

| | |
|---|---|
| Lån | 1 1 1 1 10 |
| | 0 0 0 0 0 |
| | −0 0 1 0 1 |
| Rest | $\underline{1}$ 1 0 1 1 |

talet igen och erhåller resultatet $\underline{0}$ 000 = 0, då ettan i sjätte positionen ($2^5$) faller utanför maskinens talområde.

$$\begin{array}{r} 1\,1\,1\,1 \\ \underline{1}\,1\,0\,1\,1 \\ \underline{0}\,0\,1\,0\,1 \\ \hline \underline{0}\,0\,0\,0\,0 \end{array}$$

Principen för komplementbildning är alltså enligt ovan subtraktion från en nollnivå. För binärtal kan emellertid komplementbildningen rent formellt förenklas. Om vi ser på nollvärdet $\underline{0}$ 0000, kan det även skrivas $\underline{1}$ 1111 + $\underline{0}$ 0001 (dvs −1 + 1). Komplementet till exempelvis $\underline{0}$ 0110 (6) kan därför bildas på följande sätt:

$$\underline{0}\ 000 - \underline{0}\ 0110 = \underline{1}\ 1111 - \underline{0}\ 0110 + \underline{0}\ 0001$$

$$\begin{array}{r} \underline{1}\,1\,1\,1\,1 \\ -\underline{0}\,0\,1\,1\,0 \\ \hline \underline{1}\,1\,0\,0\,1 \\ +\underline{0}\,0\,0\,0\,1 \\ \hline \underline{1}\,1\,0\,1\,0 \end{array}$$

Varje siffra i talet subtraheras alltså från ett, dvs man bildar ett-komplementet*) till samtliga siffror, varefter resultatet ökas med en enhet. Att bilda ett-komplementet till en binär siffra är nu synnerligen enkelt, eftersom en etta bytes mot en nolla och tvärtom. Vi kan alltså formulera en regel för komplementbildning:

Byt ettor mot nollor och nollor mot ettor samt addera en minsta enhet till resultatet.

---

*) Till skillnad från noll-komplementet ($2^n$-komplementet) som eljest avses.

Denna regel är ju synnerligen enkel och lämpar sig därför väl i en matematikmaskin. Då det gäller räkning för hand (exempelvis för kontroll av maskinella räkningar) kan det vara lämpligt med en något mera komplicerad regel, som snabbare leder till målet i det att de båda nämnda operationerna sammanslås till en. Denna regel lyder:

Byt ettor mot nollor och nollor mot ettor utom för den lägsta ettan och eventuella lägre nollor.

Överensstämmelse mellan de båda reglerna framgår av ovanstående exempel. Låt oss slutligen för fullständighetens skull utföra en subtraktion enligt den nya metoden, exempelvis

$\underline{0}\ 1101\ (13) - \underline{0}\ 0110\ (6)$:

$$\begin{array}{rl} \underline{0}\ 1\ 1\ 0\ 1 & \\ \underline{1}\ 1\ 0\ 0\ 1 & \text{Ett-komplement till } \underline{0}\ 0110 \\ +\underline{0}\ 0\ 0\ 0\ 1 & \\ \hline \underline{0}\ 0\ 1\ 1\ 1 & (7) \end{array}$$

## 2.4. Hela tal och binärdelar. Normering

Tal som är mindre än ett är vi vana att skriva som allmänna bråk eller decimalbråk. De senare är då uppbyggda enligt ekv. 1.32 men med basexponenterna negativa och koefficienterna i omvänd ordning, eftersom $a_n$ i detta fall är den minsta siffran. Exempelvis är decimalbråket 0,034 en förkortning av $0 \cdot 10^0 + 0 \cdot 10^{-1} + 3 \cdot 10^{-2} + 4 \cdot 10^{-3}$. Nu kan vi också bilda binärbråk på precis samma sätt med B = 2. Decimalbråket 0,625 exempelvis kan ersättas med binärbråket 0,101 dvs $0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$. Decimalkommat motsvaras här av ett binärkomma. Ett tal med både hela och delar exempelvis det decimala 6,75 kan då i binär form skrivas 110.11. Addition och subtraktion av sådana tal sker givetvis på förut angivet sätt. Detsamma gäller komplementbildning. Det negativa talet −10,11 kan vi alltså för vår maskin skriva som $\underline{1}01.01$ (kontrollsummering ger $\underline{0}00.00 = 0$). Som vi skall se i 2.5. och 2.6. är det vid maskinräkning med multiplikation och division ofta praktiskt att räkna enbart med tal, vars absolutvärde ligger mellan noll och ett, dvs i vår maskin tal från $\underline{1}.0000(-1)$ till $\underline{0}.1111(+15/16)$; fulla värdet +1 kan tydligen inte uppnås i detta fall, eftersom det skulle behöva skrivas likadant som komplementet för −1). Ett sådant tal kallar vi normerat tal. För att på detta sätt även kunna representera ett tal, som ligger utanför nämnda område, delar man upp talet i en taldel, som är ett normerat tal och en ex-

ponentdel som multiplicerad med taldelen ger talets verkliga värde. Talet 0 1101 exempelvis kan vi alltså skriva som $\underline{0}.1101 \cdot 2^4$. Hur man förfar med exponentdelen under räkningarnas gång skall vi se närmare under avsnitt 2.7. Av ovanstående framgår, att ett positivt normerat tal alltid har en nolla i heltalspositionen och ett negativt (komplementet) en etta. Heltalssiffran kan alltså tjänstgöra som teckensiffra och understrykningen blir överflödig, eftersom binärpunkten markerar teckensiffrans läge. För ett normerat tal gäller alltså, att det är positivt om det till formen ligger mellan noll och ett (0.00 ... – 0.11 ...) och negativt om det ligger mellan ett och två (1.00 ... – 1.11 ...). Ett negativt normerat tal uttryckes alltså formellt som ett tvåkomplement så att exempelvis –0.0110 skrives som 1.1010. Observera, att regeln gäller enbart för normerade tal. För alla övriga binärtal gäller de förut givna reglerna.

## 2.5. Multiplikation

### 2.5.1. Multiplikation av positiva tal

Multiplikation av två binära tal på från decimalsystemet känt manér illustreras av följande exempel där talet 1001 (9) skall multipliceras med talet 101 (5):

```
  1 0 0 1
    1 0 1
  -------
    1 0 0 1        1 · 2^0 · 1001
  0 0 0 0          0 · 2^1 · 1001
1 0 0 1            1 · 2^2 · 1001
-------
1 0 1 1 0 1 (45)   (1 · 2^2 + 0 · 2^1 + 1 · 2^0) 1001
```

Multiplikanden multipliceras alltså med siffra efter siffra i multiplikatorn tillsammans med respektive siffras basdignitet. Multiplikation med basdignitet görs som vanligt, genom att multiplikanden flyttas (skiftas) det antal steg, som basens exponent anger (ex: $10^2 \cdot 123 = 12300$; $2^2 \cdot 1001 = 100100$).

Låt oss nu utföra ovanstående multiplikation med talen skrivna på maskinspråk, dvs som 0 1001 och 0 0101:

```
      0 1 0 0 1
      0 0 1 0 1
    -------------
      0 1 0 0 1
    0 0 0 0 0
  0 1 0 0 1
0 0 0 0 0
0 0 0 0 0
-------------
0 0 0 1 0 1 1 0 1
```

Vi ser då, att produkten måste ha plats för åtta siffror förutom teckensiffran för att två fyrsiffriga tal vilka som helst skall kunna multipliceras med varandra. Detta följer av det kända förhållandet att en produkts talområde är summan av faktorernas talområden (i detta fall $2^4 \cdot 2^4 = 2^8$). En multiplikation med exempelvis tre faktorer skulle kräva 12 siffror i produkten osv. Det sagda gäller naturligtvis endast under förutsättning att man vill ha full noggrannhet i resultatet, dvs att man vill ha kvar samtliga siffror i detsamma. Ett sådant krav skapar tydligen orimliga problem i en maskin, som av praktiska skäl måste ha ett begränsat antal siffror. Beroende på antalet faktorer i en multiplikation skulle talområdet för faktorerna behöva skäras ned till bråkdelar av maskinens hela talområde, och det vore uppenbarligen att dåligt utnyttja maskinens kapacitet. Normalt nöjer man sig emellertid med en begränsad noggrannhet, antingen p.g.a. att de inmatade primärvärdena själva har en begränsad noggrannhet eller att man inte kan tillgodogöra sig hur noggranna resultat som helst. I vår maskin har vi bestämt oss för ett talområde på fyra siffror positivt eller negativt, vilket man kan tyda som att vi nöjer oss med en noggrannhet på $2^{-4} = 1/16$ av maximalvärdet. Resultatet 0 001001101 (45) i det ovan angivna exemplet skulle vi då kunna skriva som 0 0010 $\cdot 2^4$ (32), dvs en taldel med de fyra siffror maskinen har plats för samt en exponentdel (jfr 2.4.), som ger resultatet dess eventuella plats utanför taldelens eget talområde (se dock 2.5.4. om normalisering). Noggrannhetskravet uppfylles tydligen, eftersom skillnaden mellan riktigt och avrundat resultat (ej korrekt avrundat, se närmare 2.5.3.) är 13, dvs mindre än 1/16 av maxvärdet 240. Detta fel kallar vi a v r u n d n i n g s f e l .

Man kan ställa frågan, vilket talområde taldelen bör ha. I det givna exemplet har vi rört oss enbart med hela tal, där binärpunkten är belägen till höger om lägsta siffran. Om vi i stället skulle behöva multiplicera två blandade tal, exempelvis 010.11 och 0011.0, blir det inte lika enkelt som förut i synnerhet inte för en maskin, som man vill ge så enkla order som möjligt. Rent siffermässigt blir den fullständiga produkten 0 01000010, men då återstår utplacering av binärpunkten samt avrundning till fyra siffror. Det förstnämnda sker med ledning av binärpunktens placering i faktorerna, vilket ger produkten det exakta värdet 0 01000.010. Vid avrundningen kapas sedan sista heltalssiffran bort, och resultatet får därför utseendet 0 0100 $\cdot 2^1$. Som vi ser kan tydligen binärpunkten hamna litet var som helst i produkten beroende på dess placering i faktorerna, och det kommer sig som förut nämnts av att heltalsområdet för produkten är summan av heltalsområdena för faktorerna (områdena uttryckta i antalet siffror). Om då heltalsområdena för faktorerna vore noll, så skulle det tydligen bli så även för produkten, dvs binärpunktens läge skulle förbli oför-

ändrat vid multiplikation, vilket vore en förenkling gentemot det föregående. Som vi såg i föregående avsnitt, har de s.k. normerade talen just den egenskapen att ha noll i heltalsdelen. Låt oss därför omvandla faktorerna i det nyss behandlade exemplet så, att de kommer att bestå vardera av en taldel som är normerad och en exponentdel. Den första faktorn 010.11 skriver vi då som $0.1011 \cdot 2^2$ och den andra 0011.0 som $0.1100 \cdot 2^2$. Lägg märke till att vi placerar binärpunkten straxt till vänster om högsta ettan för att få så låga exponenter som möjligt. Multiplikation av taldelarna ger den exakta produkten 0.100000100, och multiplikation av exponentdelarna ger $2^2 \cdot 2^2 = 2^4$, dvs man adderar exponenterna. Slutresultatet efter avrundningen blir $0.1000 \cdot 2^4 = 1000$, vilket är detsamma som resultatet $0\ 0100 \cdot 2^1$ ovan.

Som vi sett kan man praktiskt underlätta räkningar, som omfattar multiplikation, genom att använda normerade tal och exponenträkning. Därför föredrar man också att göra så i stora maskiner för allmänt syfte (general purpose computers) med vilka långa och varierande räkneoperationer utförs. I mindre specialmaskiner däremot, där man utför ett mindre antal räkneoperationer och där talen ofta ligger inom ett begränsat talområde, kan det många gånger vara både enklare och billigare att räkna med talen i deras naturliga talområden, dvs utan exponentdelar.

### 2.5.2. Multiplikation av komplement

Även då det gäller multiplikation, står vi fast vid vår önskan att vid maskinräkning representera negativa tal med komplement. Det rent praktiska förfarandet kompliceras därvid i mycket ringa mån, som vi skall se, och det lämpar sig därför fortfarande för maskinräkning. Däremot är den matematiska bakgrunden något krångligare än för multiplikation av positiva tal enligt metoderna i 2.5.1.

Multiplikation av exponentdelar sker, som vi såg i 2.5.1. genom addition av exponenterna. Detta gäller alltid, varför vi här kan lämna sådana operationer åt sidan och enbart ägna oss åt multiplikation av taldelar. Taldelar antas dessutomutom vara normerade tal.

Vi utgår från en multiplikation, där multiplikanden är negativ, och skriver den som $a(-b) = -ab$. Enligt 2.4. ersätts $-b$ med 2-komplementet dvs $(2-b)$, och resultatet $-ab$ vill vi då också ha i formen $(2-ab)$. Om multiplikationen utförs på samma sätt som i 2.5.1. får vi i stället resultatet $a(2-b) = 2a - ab$, eftersom även "nollnivån" 2 multipliceras med a. Då nu a är mindre än ett, blir den nya nollnivån 2a mindre än 2, vilket ger ett felaktigt resultat. Problemet visar sig emellertid inte vara något problem, om bara skiftning av ett

komplementtal utförs på rätt sätt. Om vi exempelvis utför skiftet $0.1101 \cdot 2^{-4}$ (fyrstegs högerskift), får vi 0.00001101. På samma sätt blir $-0.1101 \cdot 2^{4} =$ $= -0.00001101$, och om vi skriver det i komplementform $1.0011 \cdot 2^{-4} =$ $= 1.11110011$, dvs de högre positionerna måste fyllas på med ettor i stället för nollor. Detta är också helt naturligt, eftersom det i själva verket är det tal som dras från nollnivån 2 som skiftas nedåt. Vid högerskift av ett tal fylls de högre positionerna på med nollor, om talet är positivt och med ettor om det är ett komplement. Eftersom varje delprodukt vid binär multiplikation erhålls genom skift av multiplikanden, behöver man endast se till att ovanstående regel följs. Förfarandet kallas skift med teckenkopiering.

I det nedan givna exemplet multipliceras 1.1101 (–3/16) med 0.1011 (11/16). I den visade uppställningen har varje delprodukt placerats rätt relativt multiplikator och multiplikand, dvs så att en viss sifferposition (kolumn) alltid har samma signifikans eller värde. Slutprodukten blir då också placerad på samma sätt och binärpunkten behåller sin plats genom hela operationen. Denna placering är den vanliga i maskiner. Vid handräkning behöver man endast för varje delprodukt se till, att den lägsta heltalssiffran (i detta fall teckensiffran) i multiplikanden skiftas ned mitt under respektive siffra i multiplikatorn.

```
  1.1 1 0 1
x 0.1 0 1 1
-----------
  1.1 1 1 1 1 1 0 1
  1.1 1 1 1 1 0 1
  0.0 0 0 0 0 0
  1.1 1 1 0 1
  0.0 0 0 0
-----------
  1.1 1 0 1 1 1 1 1    (–33/256)
```

Slutresultatet skall emellertid avrundas till fyra siffror plus teckensiffra, varvid vi får 1.1101 som motsvarar –3/16 eller –48/256. Avrundningsfelet blir då tydligen = 15/256, som är mindre än 1/16.

Det andra delfallet är, att multiplikatorn är negativ (2–a). Produktens önskade utseende är då fortfarande 2–ab, medan en normal multiplikation ger det felaktiga resultatet (2–a)b = 2b – ab. Vi tar som exempel följande uppställning:

```
  0.1 0 1 1    b
x 1.1 1 0 1    (2–a)
```

(2–a) kan också skrivas 1 + (1–a), där första ettan är teckensiffran och resten binärdelarna. Produkten kan då skrivas $[(1-a) + 1]b = (1-a)b + b$, dvs man

multiplicerar b först med binärdelarna i a och därefter med teckensiffran samt summerar. Om vi i stället vid multiplikation med teckensiffran gör denna delprodukt negativ (bildar 2-komplementet), får vi resultatet (1–a)b + (2–b) = = b–ab + 2–b = 2–ab dvs det önskade resultatet. Här visar sig alltså ettan i teckenpositionen ha negativ karaktär (se 2.7.3.). Vi formulerar då en regel:

Om multiplikatorn har en etta i teckenpositionen, skall dennas delprodukt subtraheras.

Med hjälp av denna regel kan vi fullborda ovan uppställda multiplikation:

| | |
|---|---|
| 0.1 0 1 1 | (11/16) |
| x 1.1 1 0 1 | (–3/16) |
| 0.0 0 0 0 1 0 1 1 | |
| 0.0 0 1 0 1 1 | |
| 0.0 1 0 1 1 | |
| 1.0 1 0 1 | kompl. av 0.1011 |
| 1.1 1 0 1 (1 1 1 1) | (–3/16; jfr förra ex.) |

Slutligen tar vi och multiplicerar två negativa tal, och använder därvid båda reglerna samtidigt. Som exempel kan vi ta kvadraten på 1.0011 (–13/16). Produkten skall då bl.a. bli positiv.

| | |
|---|---|
| 1.0 0 1 1 | |
| x 1.0 0 1 1 | |
| 1.1 1 1 1 0 0 1 1 | |
| 1.1 1 1 0 0 1 1 | |
| 0.1 1 0 1 | kompl. av 1.0011 |
| 0.1 0 1 0 (1 0 0 1) | (10/16) |

Resultatet blir alltså avrundat 10/16, medan det riktiga resultatet är 169/256 = = 10,6/16. Det avrundade resultatet håller sig därför inom den i 2.5.1. angivna felgränsen. Som vi skall se i nästa avsnitt, kan man emellertid med samma antal siffror uppnå dubbelt så god noggrannhet som den nyss nämnda.

### 2.5.3. Korrekt avrundning

Vid avrundning som vi hittills behandlat, har vi helt enkelt tänkt oss att kapa bort de icke önskade siffrorna. Om vi som exempel tar den avkortade produkten 0.1010 i sista exemplet i 2.5.2. skulle den fullständiga produkten kunna vara både 0.10100000 och 0.10101111. I det förra fallet är det avrundade värdet exakt rätt, men i det andra är det för litet med ett belopp av 15/16 av sista

siffran i det avrundade talet, dvs så gott som hela denna siffra. Maximala avrundningsfelet är alltså ungefär minus sista siffran i slutresultatet. Man kan givetvis aldrig få bättre upplösning eller finhet i resultatet än vad sista siffran i detsamma anger (dvs stegen mellan olika talvärden är aldrig mindre än sista siffran), men man kan halvera maximalfelet genom att låta hälften ligga på positiva sidan och hälften på negativa sidan om det riktiga resultatet. I det fallet vet man då alltid, att det avrundade resultatet inte skiljer sig från det riktiga med mer än halva sista siffran. Oftast är detta att föredra, eftersom felets storlek normalt är mer avgörande än dess riktning. Metoden att på detta sätt halvera maximala avrundningsfelet känner vi igen från decimalräkningen under benämningen korrekt avrundning. Denna skulle då innebära, att exempelvis det avrundade talet 0.1010 kan representera vilket som helst av de noggrannare talen 0.10011000 ... – 0.10100111 ... dvs (0.1010 – 0.00001) – – (0.1010 + 0.00000111 ...). Med andra ord: Om vid avrundning den högsta borttagna siffran är en etta, skall det avrundade talet höjas en minsta enhet. Regeln gäller oförändrad för komplementtal, eftersom den endast bestämmer närmaste avrundningsnivå för ett icke avrundat tal. Sista exemplet i 2.5.2. ger vid noggrann uträkning produkten 0.10101001, som korrekt avrundad till fyra siffror blir 0.1011 dvs 11/16. Detta resultat ligger närmare det riktiga resultatet 10,6/16 än det vid icke korrekt avrundning erhållna 10/16 och är som sig bör i själva verket mindre än 0,5/16 fel. Givetvis måste man för korrekt avrundning få upplysning om den högsta borttagna siffran, vilket i en maskin med begränsat talområde kan verka problematiskt. Saken skall behandlas närmare i kapitlet om räknemaskiner.

### 2.5.4. Angivelsenoggrannhet. Normalisering

I avsnitt 2.5.1. nämndes, att man kan begränsa antalet siffror i en maskin tack vare att man nöjer sig med begränsade noggrannheter. Samtidigt nämndes, att om man begränsar talområdet till fyra siffror detta kunde fattas som att man nöjer sig med en noggrannhet i resultatet på $2^{-4}$ av maximalvärdet. I 2.5.3. förbättrade vi den noggrannheten till $\pm 1/2 \cdot 2^{-4}$. I båda fallen är noggrannheten angiven i absolut mått, vilket innebär att den är en viss bråkdel av maximalvärdet och oberoende av det aktuella talets värde. För ett n-siffrigt binärt tal är således den absoluta noggrannheten alltid $\pm 1/2 \cdot 2^{-n}$. I många fall kan man emellertid komma ännu längre i noggrannhet, som vi skall illustrera med ett exempel. Vi multiplicerar talet 0.1111 med 0.0011 och antar för enkelhets skull, att båda talen är exakt angivna dvs ej avrundade.

$$
\begin{array}{l}
0.1\,1\,1\,0 \\
0.0\,0\,1\,1 \\
\hline
\qquad 0\,1\,1\,1\,0 \\
\qquad\quad 0\,1\,1\,1\,0 \\
\hline
0.0\,0\,1\,0\,1\,0\,1\,0
\end{array}
$$

Resultatet blir exakt 0.0010101 och korrekt avrundat till fyra siffror 0.0011, dvs avrundningsfelet är $0.0011 - 0.0010101 = 0.0000011$ eller 3/128. Om vi i stället skriver multiplikatorn som $0.1100 \cdot 2^{-2}$ (= 0.0011), får vi det exakta resultatet $0.10101 \cdot 2^{-2}$, korrekt avrundat till fyra siffror = $0.1011 \cdot 2^{-2}$.

$$
\begin{array}{ll}
0.1\,1\,1\,0 & \\
0.1\,1\,0\,0 & \cdot 2^{-2} \\
\hline
\qquad 0\,1\,1\,1\,0 & \\
\qquad\quad 0\,1\,1\,1\,0 & \\
\hline
0.1\,0\,1\,0\,1\,0 & \cdot 2^{-2}
\end{array}
$$

Avrundningsfelet blir i det fallet $(0.1011 - 0.10101) \cdot 2^{-2} = 0.00001 \cdot 2^{-2} = 0.0000001$ eller 1/128. Samma resultat skulle vi ha erhållit, om vi i den första multiplikationen flyttat resultatets taldel två steg åt vänster, dvs skrivit resultatet som $0.10101 \cdot 2^{-2}$ och sedan avkortat. Vi ser alltså, att ju större del av talets verkliga värde taldelen utgör desto noggrannare kan talet anges. Förklaringen ligger i det förhållandet, att maximala avrundningsfelet i en n-siffrig taldel är konstant $\pm 1/2 \cdot 2^{-n}$. Detta fel multipliceras sedan med en eventuell exponentdel, varvid erhålles felet i det aktuella talet. Ju större taldelen görs för det aktuella talet, desto mindre blir exponentdelen och således också maxfelet. Förfarandet att göra taldelen (uttryckt i normerat tal) så stor som möjligt kallas normalisering. Därvid måste man komma ihåg, att det är taldelens absolutvärde som skall göras stort. Ett positivt tal skall alltså innehålla så höga ettor och ett komplementtal så höga nollor som möjligt. Ett normaliserat tal innehåller alltid en etta närmast till höger om binärpunkten om det är positivt och en nolla om det är negativt. Absolutvärdet av ett normaliserat tal ligger alltså mellan en halv och ett.

Vi skall nu försöka oss på en jämförelse mellan det förut nämnda absoluta felet, som vi kan kalla $d_a$ och det nya som vi kallar $d_n$. Det icke normaliserade talet kallar vi B (normerat) och det normaliserade $B_n \cdot 2^{-m}$ enligt ovan. Som förut nämnts är $d_a \leq \pm 1/2 \cdot 2^{-n}$, varför $d_n \leq \pm 1/2 \cdot 2^{-n} \cdot 2^{-m}$ ($d_n \leq d_a$ ty $2^{-m} \leq 1$). Men $B_n \geq 1/2$ och $B_n \cdot 2^{-m} = B$, varför $2^{-m} \leq 2\,B$. Således blir $d_n \leq \pm 2^{-n} \cdot B$, dvs $(d_n)_{max}$ är proportionellt mot talets aktuella värde B (upp

till $d_n = d_a$, se ovan). $d_n/B$ anger då den relativa noggrannheten i talet B och den är för ett normaliserat, n-siffrigt tal konstant $\leq \pm 2^{-n}$. Om $B \geq 1/2$ är $d_n = d_a$ och för $B < 1/2$ är $d_n < d_a$. Det är just för det sistnämnda fallet, som noramliseringen visar sin betydelse för felreduktionen.

## 2.6. Division

Liksom för de tre förut behandlade räknesätten skall vi först visa exempel på en division enligt gängse handräkningsmetod med två binära heltal exempelvis 1101 (13) dividerat med 1000 (8).

```
 1101   | 1000
-1000   |------
 ----   | 1.101
  1010
 -1000
 -----
    1000
   -1000
   -----
    0000
```

Vi kan också utföra divisionen med talen normerade, men då måste vi komma ihåg att göra divisorns taldel större än eller lika med dividendens för att även kvotens taldel skall bli normerad, dvs till sitt absolutvärde mindre än ett. Vi skriver därför dividenden som $0.01101 \cdot 2^5$ och divisorn som $0.1 \cdot 2^4$.

```
 0.01101  | 0.1
-  01    |-------
 ------   | 0.1101
   001
 -  01
 ------
    0001
 -    01
 -------
      00
```

Vid division subtraheras exponenterna och kvar blir exponentdelen $2^1$ ($2^5/2^4 = 2^{5-4}$). Slutresultatet blir således $0.1101 \cdot 2 = 1.101$ vilket givetvis stämmer med det första resultatet.

Multiplikation kan ses som en form av addition, där multiplikanden adderas så många gånger som multiplikatorn anger, och division som en form av subtraktion, där divisorn subtraheras från dividenden så många gånger som kvoten anger dvs tills resten är noll eller nära noll. (Som nämnts förkortas dock räk-

ningarna i praktiken genom knepet, att man i stället för att ett i sänder summera eller subtrahera $B^n$ (B = basen) stycken tal A flyttar A n steg åt vänster eller höger från kommat beroende på om n är positivt eller negativt och sedan summerar eller subtraherar det erhållna talet.)

Enligt ovan kan alltså en division a/b = c skrivas som a–bc = 0 för att bättre motsvara tillvägagångssättet. Av det senare uttrycket framgår, att resultatet blir riktigt även för komplementtal, under förutsättning att multiplikationen bc utförs enligt de regler vi uppställt i 2.5.2. Tag som exempel, att divisorn är negativ och alltså skrivs 2–b (normerat tal). Kvoten blir då också negativ och önskas i formen 2–c. Uttrycket a–(2–b)(2–c) skall alltså bli noll, om divisionen är riktig. Nu är (2–b) · (2–c) = bc enligt 2.5.2. och alltså a–(2–b)(2–c) = a–bc = = 0 enligt ovan.

Antag att a och b enligt ovan är normerade binärtal, positiva tal eller komplement och att absolutvärdet av b är större än absolutvärdet av a ($|b| > |a|$). Kvoten c blir då också ett normerat tal, som vi kan skriva $c = -c_o \cdot 2^o + c_1 \cdot 2^{-1} + \ldots + c_n \cdot 2^{-n}$. Minustecknet framför $c_o$ skriver vi med tanke på att multiplikationen bc skall bli riktig för negativt c (se 2.5.2.). Divisionen utförs då enligt ekv:

$$a–b(-c_0 + c_1 \cdot 2^{-1} + \ldots + c_n \cdot 2^{-n}) = 0$$

Division i maskin kan då utföras på följande sätt:

Ekvationen säger, att b skall skiftas ett steg i sänder åt höger och subtraheras från a eller delresten av a. Vid varje steg en subtraktion kan göras utan att resten hamnar på andra sidan noll, sätts motsvarande $c_i$ i kvoten lika med ett. Två fall ges:

1. Antag att a och b har lika tecken: (c större än 0, dvs $c_0 = 0$)

   Genom bortförkortning av minustecknen kan divisionen alltid återföras till till att a och b båda är exempelvis positiva:

   $$a–b(c_1 \cdot 2^{-1} + c_2 \cdot 2^{-2} + \ldots + c_n \cdot 2^{-n}) = 0$$

   Pröva $c_1 = 1$ (dvs b skiftas nedåt ett steg och dras från a)

   Rest > 0 ger $c_1 = 1$; rest < 0 ger $c_1 = 0$ och a återställs

   Pröva $c_2 = 1$ osv.

   Den tvivlande läsaren kan själv föra samma resonemang för negativa värden på a och b. Skillnaden blir då endast, att man närmar sig noll från motsatt sida.

2. Antag att a och b har olika tecken: (c mindre än 0, dvs $c_0 = 1$)

Genom multiplikation med minustecken kan divisionen alltid återföras till att exempelvis a är negativ och b positiv:

$$-|a| - b(-1 + c_1 \cdot 2^{-1} + c_2 \cdot 2^{-2} + \ldots + c_n \cdot 2^{-n}) = 0$$

$-|a| - b(-1) = -|a| + b$ utföres. Resten $> 0$ ty $b > |a|$.

Resten av divisionsoperationen övergår alltså till:

$$r - b(c_1 \cdot 2^{-1} + c_2 \cdot 2^{-2} + \ldots + c_n \cdot 2^{-n}) = 0; \quad r > 0$$

dvs samma slags uttryck som i fall 1. Resten av kvotsiffrorna bestäms därför på samma sätt som där.

Vi ser alltså, att så länge resultatet av en subtraktion ligger på samma sida om noll som a i fall 1 och r i fall 2, skall en etta sättas på motsvarande plats i kvoten. Men b har enligt ovan alltid samma tecken som a i fall 1 och r i fall 2. Sedan teckensiffran i kvoten bestämts ur teckenjämförelse mellan dividend och divisor, kan därför resten av kvotsiffrorna bestämmas enligt följande regel:

För varje gång resultatet av en delsubtraktion får samma tecken som divisorn, sätts en etta i motsvarande plats i kvoten.

För varje gång en subtraktion ger ett resultat som ligger på fel sida om noll, måste enligt ovan den föregående delresten återställas. Maskinen kan nämligen inte på förhand avgöra resultatets tecken (vilket sker vid division för hand, jfr 2.3.2.). I värsta fall kan alltså två operationer erfordras för varje kvotsiffra. Denna divisionsmetod kallas division med återställning.

Låt oss emellertid se på exempelvis subtraktionen

$a - b \cdot 2^{-1} = r_1$. Om $r_1 < 0$ vill vi i stället pröva $a - b \cdot 2^{-2} = r_2$.

Men $a - b \cdot 2^{-1} + b \cdot 2^{-2} = a - b \cdot 2^{-2} = r_2$. Vi behöver alltså inte återställa a för att kunna pröva andra steget. Om en delrest alltså hamnar på fel sida om noll, fortsätter vi att skifta b men adderar b till delresten, tills denna hamnar på rätt sida om noll. På så sätt erfordras aldrig mer än en operation per kvotsiffra. Första operationen i fall 2 kan också överhoppas, eftersom man när man prövar $c_1 = 1$ sammanlagt har utfört beräkningen $-|a| + b - b \cdot 2^{-1}$. Men $-|a| + b - b \cdot 2^{-1} = -|a| + b \cdot 2^{-1}$. Om kvoten är negativ, kan man alltså direkt skifta ned b ett steg och addera till dividenden. Summan av det hela blir följande regel för division utan återställning: Om dividenden respektive delresten (deldividenden)

har samma tecken som divisorn, skall den senare subtraheras från (del)dividenden eljest adderas. Vid subtraktion skall en etta sättas på respektive plats i kvoten och vid addition en nolla utom för teckensiffran, där förhållandet är omvänt.

Som exempel utför vi en fullständig division: 0.1010 (10/16) dividerat med 1.0100 (–12/16), dvs divisorn är negativ. Som förut nämnts måste vi iakttaga multiplikationsreglerna i 2.5.2. Andra regeln kommer automatiskt med vid bildandet av kvotens tecken och tillämpningen av den första framgår av divisionsexemplet i operationerna $b \cdot 2^{-n}$.

| | | |
|---|---|---|
| Olika tecken: 1 i kvottecknet | 0.1010 | 1.0100 |
| $+b \cdot 2^{-1}$ | 1.10100 | 1.00101 |
| Olika tecken: 0 i 1:a kvotsiffran | 0.01000 | |
| $+b \cdot 2^{-2}$ | 1.110100 | |
| Olika tecken: 0 i 2:a kvotsiffran | 0.000100 | |
| $+b \cdot 2^{-3}$ | 1.1110100 | |
| Lika tecken: 1 i 3:e kvotsiffran | 1.1111100 | |
| $-b \cdot 2^{-4}$ | 0.00001100 | |
| Olika tecken: 0 i 4:e kvotsiffran | 0.00000100 | |
| $+b \cdot 2^{-5}$ | 1.111110100 | |
| Lika tecken: 1 i 5:e kvotsiffran | 1.111111100 | |

Vid fortsatt dividerande finner man att siffrorna noll och ett upprepas så att kvoten blir 1.00101010 ... motsvarande –0.11010101 ... Divisionen går aldrig jämnt upp, men om vi tar med fler och fler siffror i kvoten, skall vi finna att värdet närmar sig det riktiga, som är –10/12.

Som en parentes kan vi roa oss med att ta fram kvotens exakta värde:

Sätt:

$$x = -0.11010101 \ldots$$

$$2^2 x = -11.010101 \ldots$$

$$2^4 x = -1101.010101 \ldots$$

$$2^4 - 2^2 \; x = -(1101.010101 \ldots - -11.010101 \ldots) =$$

$$= -(1101 - 11) = -1010$$

$$x = -\frac{1010}{2^4 - 2^2} = -\frac{1010}{10000-100} = -\frac{1010}{1100} = -\frac{10}{12}$$

I det visade divisionsexemplet skiftade vi hela tiden divisorn nedåt och fick mot slutet en mängd siffror i deldividenden, något som vi i en maskin vill undvika. En motsvarighet till nämnda förfarande är emellertid att skifta (del)divi-

denden uppåt. Om vi då behåller den nämnda begränsningen att dividendens absolutvärde aldrig får vara större än divisorns, kommer de dividendsiffror som skiftas uppåt utanför talområdet att vara utan betydelse, dvs alltid nollor för positiva tal och alltid ettor för komplement. Om nämligen (del)dividenden a är mindre än eller lika med divisorn b blir $a \cdot 2^{-b} \leq a$, dvs de viktiga siffrorna ligger fortfarande kvar inom talområdet. Det finns ett fall, som kanske väcker läsarens undran, och det är då dividendens första siffra är motsatsen till teckensiffran, dvs dividenden är 0.1 ... eller 1.0 ... Vid skiftning uppåt byter dividenden i dessa fall tecken. Det spelar faktiskt ingen roll, att så händer, men för att framställningen här inte skall belastas för mycket, ges beviset i avsnitt 2.7.3., där den intresserade läsaren kan finna en något utförligare behandling av komplementtalens natur.

Vi utför samma division som nyss men skiftar (del)dividenderna, varvid vi slopar de siffror som skiftas utanför det antagna talområdet (fyra siffror plus teckensiffra).

Före varje skift jämföres tecknen hos (del)dividenden och divisorn för erhållande av motsvarande kvotsiffra:

| | | | |
|---|---|---|---|
| 1 i kvoten | 0.1010 | 1.0100 | |
| Skift | 1.0100 | 1.00101 ... | |
| Addition | 1.0100 | | |
| 0 i kvoten | 0.1000 | | |
| Skift | 1.0000 | | |
| Addition | 1.0100 | | |
| 0 i kvoten | 0.0100 | | |
| Skift | 0.1000 | | |
| Addition | 1.0100 | | |
| 1 i kvoten | 1.1100 | | |
| Skift | 1.1000 | | |
| Subtraktion | 0.1100 | | |
| 0 i kvoten | 0.0100 | | |
| Skift | 0.1000 | | |
| Addition | 1.0100 | | |
| 1 i kvoten | 1.1100 | o.s.v. | |

Vi sammanfattar nu en fullständig divisionsregel:

Tillse först att divisorns absolutvärde är minst lika stort som dividendens. Vid divisionens utförande skiftas

(del)dividenden stegvis uppåt, och före varje skift jämföres tecknen hos (del)dividend och divisor. Vid lika tecken subtraheras divisorn från (del)dividenden, eljest adderas den. Vid lika tecken sätts en etta i respektive position i kvoten och vid olika tecken en nolla utom för teckensiffran, där förhållandet är omvänt.

## 2.7. Speciella synpunkter

### 2.7.1. Allmänt

Framställningen i det föregående har kanske tyckts läsaren väl ingående behandla en del elementär folkskolematematik, även om den varit i binär form. Det visar sig emellertid, att de flesta utför de fyra enkla räknesätten rent rutinmässigt utan att känna till den matematiska bakgrunden till förfarandena. Denna bakgrund är dock nödvändig för den, som vill lära sig förstå uppbyggnaden av och arbetssättet hos en sifferräknemaskin. Detta gäller i synnerhet då man för att förenkla uppbyggnaden av en maskin tillgriper vissa varianter av räkneförfarandena, som bl.a. komplementräkning och division i de föregående avsnitten givit exempel på.

Om man ibland, t.ex. för att verifiera ett räkneresultat under provning av en maskin, finner sig nödsakad att för hand räkna med binära tal, bör man givetvis i möjligaste mån välja det snabbaste och mest invanda räkneförfarandet. För en människa med sin förmåga till överblick av ett räkneproblem, är det sålunda lättare och mera påtagligt att exempelvis räkna med negativa tal i stället för med komplement och att utföra division med nedåtskift, som vi lärt i skolan i stället för med uppåtskift som är lämpligare för en maskin. I det första fallet överföres då efter räkningarnas utförande ett eventuellt negativt resultat till komplementform för jämförelse med maskinresultatet. I det fall man steg för steg vill kontrollera en maskinräkning, exempelvis division, kan det dock vara lämpligt att räkna på samma sätt som maskinen.

Läsaren undrar kanske, om det ändå är nödvändigt med komplementräkning även då det är fråga om en maskin. Kan man inte utföra en vanlig subtraktion i en sådan? Jo, visst kan man det. Det går lika enkelt att konstruera en maskin som subtraherar som en som adderar, men eftersom man knappast kan undvara addition, skulle en sådan maskin behöva innehålla två typer räkneenheter. Den blir då dyrare och mera komplicerad, utan att man i de flesta fall vunnit något. Som det påpekades i avsnitt 2.3.2. kan nämligen en maskin inte utan vidare avgöra storleksförhållandet mellan subtrahend och minuend. Är därför subtrahenden större än minuenden, blir resultatet ett komplementtal (genom

att maskinen lånar en etta från en tänkt position ovanför teckenpositionen). Vi kommer alltså inte undan komplementtalen, och man får väl säga, att det i regel inte lönar sig att konstruera en maskin för både addition och normal subtraktion.

Nedan följer som avslutning av detta kapitel en mera ingående diskussion av exponenträkningens möjligheter och komplementtalens natur samt en enkel regel för omvandling av decimaltal till binärtal.

## 2.7.2. Exponenträkning (Flytande räkning)

I avsnitten om multiplikation och division har helt flyktigt exponenträkningen berörts. Som vi minns från avsnitt 2.4, multiplicerar man en normerad taldel med en exponentdel för att få talets verkliga värde. Exponentdelen är då en viss dignitet av basen i det använda talsystemet, dvs i det här fallet $2^n$. Eftersom basen alltid är densamma under räkningarnas gång, är det bara den variabla exponenten n man behöver räkna med i exponentdelen. Vid multiplikation och division av två tal $B_1 \cdot 2^{n1}$ och $B_2 \cdot 2^{n2}$ erhålls resultaten $B_1 \cdot B_2 \cdot 2^{n1} \cdot 2^{n2} = B_1B_2 \cdot 2^{n1+n2}$ respektive $B_1 \cdot 2^{n1}/B_2 \cdot 2^{n2} = B_1/B_2 \cdot 2^{n1-n2}$, vilket visar att taldelar och exponentdelar kan behandlas var för sig. Vid multiplikation av taldelarna skall då exponenterna adderas och vid division divisorns exponent subtraheras från dividendens. Vid addition och subtraktion skall man endast se till, att exponenterna är lika. Vi kan alltså dela varje tal i två delar, som alltid hänger ihop men vid räkning behandlas var för sig, och exempelvis skriva ovannämnda tal som $B_1$, $n_1$ respektive $B_2$, $n_2$. De angivna operationernas resultat blir då $B_1 \cdot B_2$, $n_1+n_2$ respektive $B_1/B_2$, $n_1-n_2$. Då är det också lätt att ange exponentdelarna med binära tal. Antag som exempel att vår fyrsiffriga maskin för exponentdelen har det tresiffriga talområdet 1.000 – –0.111 (dvs $2^{-8} - 2^7$). Märk väl, att vi i detta fall använder binärpunkten enbart för att skilja mellan teckensiffra och övriga siffror, då exponenten då det gäller de fyra enkla räknesätten alltid är ett heltal. (För att få enhetlighet med det föregående undvikes det lika riktiga skrivsättet 1 000 – 0 111). Vårt talområde har alltså utvidgat sig till $1.0000 \cdot 2^7 - 0.1111 \cdot 2^7$ eller –128 – +120, och det minsta tal skilt från noll, som vi kan ange, är $\pm 0.0001 \cdot 2^{-8}$ eller $\pm 2^{-12}$ (1/4096). Detta talområde har vi åstadkommit med sammanlagt nio siffror (fem för taldelen och fyra för exponentdelen), medan vi skulle ha behövt tjugo siffror, om vi inte hade haft exponentdel ($2^{-12} = 128 \cdot 2^{-19}$, tjugonde siffran är teckensiffra). Förklaringen till detta fenomen ligger däri, att vi i det senare fallet aldrig behöver riskera ett större absolut avrundningsfel än $\pm 1/2 \cdot 2^{-12}$ (se 2.5.3.), medan vi i det förra i bästa fall (genom utnyttjan-

de av normalisering, se 2.5.4.) får räkna med ett maximalt relativt avrundningsfel av $\pm 2^{-4}$ av det aktuella talvärdet (ned till $\pm 0.1000 \cdot 2^{-8}$, därunder växer det relativa felet, eftersom exponentområdet är fullt utnyttjat). I exponentfallet blir då det absoluta felet större än i det andra fallet, om talet är större till sitt absolutvärde än $2^{-8}$. Genom att tillåta avrundningsfelet att variera med talvärdet kan man således göra en avsevärd vinst i förhållandet mellan antalet bitar och talområdet. Detta innebär dels givetvis en inbesparing i materialuppbåd för en maskin, men dels också tidsbesparing vid problemlösningar, eftersom multiplikations- och divisionstiderna (i en seriemaskin även additionstiderna) i stort sett är proportionella mot antalet bitar i taldelen. Taldels- och exponentdelsoperationerna kan antingen göras efter varandra i samma beräkningsenhet eller också samtidigt i skilda beräkningsenheter. I det förra fallet vinner man något i materialuppbåd och i det senare något i tid.

Ovan angivna räkneförfarande kallas också "räkning med flytande binärpunkt" eller också kort och gott "flytande räkning".

Det må här än en gång betonas, vad som nämndes i slutet av avsnitt 2.5.1., att fördelar och nackdelar med exponenträkning måste värderas för varje typ av maskin. I en maskin för allmänna beräkningar är fördelarna stora, medan i en specialmaskin olika faktorer såsom önskad typ av noggrannhet, form hos in- och utgångsdata, eventuell inblandning av analogiräkning m.m. påverkar ställningstagandet.

### 2.7.3. Mer om teckensiffran. Spilltal och deras egenskaper

Som vi lärt i det föregående, kan högsta sifferpositionen användas till att indikera talets tecken, varvid nolla ger positivt och etta negativt tecken. Här gäller det emellertid att hålla isär begreppen skenbar och verklig matematisk egenskap hos teckensiffran och främst då ettan. Om vi håller oss till normerade tal, har en etta i teckenpositionen vid additioner skenbart värdet –1. Tar vi som exempel talet 1.1101 (–0.0011) kan vi nämligen tänka oss det som $-1 + 0.1101$ ($= -0.0011$) eftersom båda uttrycken ger noll, om vi lägger till talet 0.0011. Denna egenskap är dock endast skenbar och kommer sig helt och hållet av att alla ettor som bildas ovanför teckensiffran slopas. Det verkliga matematiska värdet hos teckenettan är emellertid +1, så som komplementtalen skrivs. Komplementet till talet a får alltså det verkliga värdet $1 + (1-a)$ i stället för $-1 + (1-a)$, som är matematiskt identiskt med $-a$. Vid multiplikation med ett komplementtal som multiplikator ger teckenettans positiva värde då en felaktig delprodukt, eftersom exempelvis $1 \cdot b$ inte ens sken-

bart kan fås att överensstämma med $-1 \cdot b$. Därför måste man göra teckensiffrans delprodukt negativ för att få rätt slutresultat, som vi såg i avsnitt 2.5.2. Samma fenomen uppträder vid bildandet av kvotens teckensiffra vid division.

Teckensiffrans egenskap av ordinär siffra ger upphov till ett intressant förhållande. Antag, att vi skall lösa problemet $a+b-c=d$, där a, b, c och d är positiva normerade tal men där $a+b > 1$, låt oss säga $a+b = 1+e$. Vi kan beräkna d på två sätt antingen genom att först dra c från b och lägga resten till a eller också genom att först lägga ihop a och b och sedan dra bort c. Men vid summeringen av a och b får vi ett tal större än ett, dvs vi hamnar utanför talområdet och får formellt ett komplementtal p.g.a. ettan i teckenpositionen. Enligt våra definitioner är resultatet alltså negativt men matematiskt är det ett tal mellan ett och två. Eftersom d skall vara mindre än ett, måste c vara större än e, säg $c = e+\epsilon$. Det riktiga värdet på d blir då $1+e-(e+\epsilon) = 1-\epsilon$. Om vi nu först alltså adderar a och b, får vi komplementtalet 1+e. Sedan drar vi ifrån c, genom att lägga till dess 2-komplement. Då får vi $1+e+2-c = 2+1+e-(e+\epsilon) = 1-\epsilon$, eftersom tvåan försvinner ovanför teckenpositionen. Vi har alltså fått ett riktigt slutresultat, genom att teckensiffran uppträtt som vanlig siffra med positivt värde då a+b blev större än ett. Ett tal, som på detta sätt överskrider talområdet, kallar vi spilltal och den siffra, som skjuter över och hamnar i teckenpositionen, kallar vi spill. Man kan givetvis också få negativa spilltal, som formellt blir positiva, eftersom en nolla hamnar i teckenpositionen i detta fall. Om vi bibehåller ovan angivna relationer för a, b, c och d men i stället skriver problemet som $-(a+b)+c=d$, så får vi, då vi bildar $-(a+b)$, ett negativt spilltal $-(1+e)$. Rätta värdet på $-d$ blir $-(1+e)+e+\epsilon = -(1-\epsilon)$. Låt oss nu utföra beräkningen av $-d$ på maskinsätt med komplementräkning. För $-(a+b)$ får vi då $2-(a+b) = 2-(1+e) = 1-e$, dvs ett tal mellan 0 och 1 och således formellt positivt. Lägger vi nu till $c=e+\epsilon$, får vi $1-e+e+\epsilon = 2-(1-\epsilon)$, som är komplementet till $(1-\epsilon)$ eller .d. Resultatet blir alltså riktigt även i detta fall. Vi har alltså funnit det intressanta förhållandet, att om slutresultatet av en beräkning ligger inom talområdet, kan överskridande av talområdet genom spill tillåtas under beräkningen.

För att ge exempel på det sagda utför vi beräkningen 0.0111 + 0.1100 − 0.0101 (dvs 7/16 + 12/16 − 5/16):

| | | |
|---|---|---|
| 0.0 1 1 1 | | |
| +0.1 1 0 0 | | |
| 1.0 0 1 1 | Spilltal 19/16 | |
| +1.1 0 1 1 | (−0.0101) | |
| 0.1 1 1 0 | | |

Resultatet blir alltså det riktiga 14/16.

Om vi i det ovan uppställda problemet a+b−c, låter b=a, får vi 2a−c som vi till formen känner igen från divisionsförfarandet, där deldividenden skiftas uppåt ett steg och därefter minsaks (eller ökas) med divisorn. Riktigheten i detta förfarande har alltså här fått sin bekräftelse.

## 2.7.4. Enkel regel för omvandling av decimaltal till binärtal

Omvandling från binärtal till decimaltal kan man lätt göra med hjälp av ekv. 2.11. Många gånger kan det emellertid också finnas behov av att snabbt omvandla ett decimaltal till ett binärt tal. Därför skall vi ta fram en enkel regel för detta förfarande. Antag att vi har en talstorhet A som uttryckt i binär form är $A=a_n \cdot 2^n + \ldots + a_0 \cdot 2^0$. Om $a_0=0$, är A tydligen jämnt delbart med två, eljest erhålles $a_0$ som en rest = 1 vid division med två. Vi gör samma bedömning för övriga a-koefficienter enligt följande schema:

$A = a_n \cdot 2^n + \ldots + a_1 \cdot 2 + a_0$

A/2 ger $a_0$ i rest

Sätt $A_1 = \frac{A-a_0}{2} = a_n \cdot 2^{n-1} + \ldots + a_2 \cdot 2 + a_1$

$A_1/2$ ger $a_1$ i rest

osv.

På detta sätt kan man förfara (n+1) gånger, eftersom $A_{n+1} = 0$. Förfarandet är alltså en upprepad division av A med två. Den rest som erhålles vid varje division, utgör motsvarande binärsiffra. Dessa siffror erhålles i ordning från den lägsta till den högsta.

Låt oss slutligen med ledning av ovanstående bilda den binära motsvarigheten till 457:

| | |
|---|---|
| 457/2 = 228 rest 1 | $a_0 = 1$ |
| 228/2 = 114 rest 0 | $a_1 = 0$ |
| 114/2 = 57 rest 0 | $a_2 = 0$ |
| 57/2 = 28 rest 1 | $a_3 = 1$ |
| 28/2 = 14 rest 0 | $a_4 = 0$ |

$$
\begin{aligned}
14/2 &= 7 \text{ rest } 0 \quad a_5 = 0 \\
7/2 &= 3 \text{ rest } 1 \quad a_6 = 1 \\
3/2 &= 1 \text{ rest } 1 \quad a_7 = 1 \\
1/2 &= 0 \text{ rest } 1 \quad a_8 = 1
\end{aligned}
$$

Den binära motsvarigheten till 457 är således 111001001 (jfr avsnitt 2.1.).

Ovan angivna omvandlingsregel gäller allmänt och inte endast för binärtal. Om man vill göra en omvandling till ett tal med basen B, utför man upprepade divisioner av decimaltalet med B och restsiffrorna ger därvid de önskade koefficienterna på samma sätt som ovan.

Omvandling av decimala bråk till motsvarande binära bråk är något krångligare än då det gäller heltal. Anledningen är den, att ett decimalbråk är angivet i minuspotenser av tio medan för ett binärbråk gäller minuspotenser av två (ex: $0.625 = 625/1000$ eller $5/8 = 5 \cdot 2^{-3} = 0.101$ binärt). Före omvandlingen bör man därför omforma ett decimalbråk, så att det anger minuspotenser av två varefter täljaren kan omvandlas på sättet som angivits ovan. Som exempel kan vi omvandla decimalbråket 0.4 med en önskad noggrannhet av 6 bitar, dvs binärbråket skall anges i 64-delar. Antalet 64-delar erhålles lätt genom uträkningen $\frac{4}{10} \cdot \frac{6.4}{6.4} = \frac{25.6}{64}$ närmast avrundat till 26/64. Täljaren 26 omvandlas sedan på ovan angivet sätt till det binära heltalet 11010, vilket insatt som binärbråk för 64-delar blir 0.011010.

## KAPITEL 3. SYMBOLISK LOGIK OCH LOGISKA KRETSAR

### 3.1. Inledning

#### 3.1.1. Allmänt

Alla har vi väl stött på ordet logik och vet, även om vi har svårt med en närmare precisering, att det har med slutledningskonst att göra. Logiken kommer ofta till uttryck i det dagliga livet om än i enkla former. Låt oss exempelvis anta, att vi önskar få tag på en viss person per telefon. Vi vet, att personen i fråga befinner sig på sitt arbete på dagen. Vi vet vidare, att vi vill ha tag på honom just på dagen. Genom enkel slutledningskonst bedömer vi då, att vi bör ringa till personens arbete och inte till hans hem eller någon annanstans. Utan att kanske tänka på det har vi gjort en logisk slutledning av grundläggande typ. Har vi sedan förmågan att göra flera dylika slutledningar, som var för sig är enkla men som sammanhänger med varandra på ett mer eller

mindre komplicerat sätt, har vi även möjligheter att åstadkomma t.o.m. sådana tankebravader, som man får uppleva i detektivromaner.

Vill vi då definiera begreppet logik, så kan vi säga, att det är vetenskapen om hur man med utgångspunkt från givna förutsättningar kommer fram till riktiga slutsatser.

### 3.1.2. Symbolisk logik

Ovannämnda personsökningsfall kan uttryckas mycket enklare. Antag, att förutsättningen att personen befinner sig på sitt arbete på dagen betecknas med A, att vi söker honom på dagen med B och att vi bör ringa till arbetet med C, så kan vi helt kort uttrycka vår slutledning på följande sätt:

om A och B gäller, så gäller C ..... 3.1.21

Vi har här tagit ett viktigt steg i och med att vi ersatt förutsättningarna och den logiska slutsatsen med symboler. På grund av logikens enkla grundkaraktär (antingen gäller ett påstående, eller så gäller det inte) ligger det nära till hands att göra på det viset. Det ligger också nära till hands att försöka behandla dessa symbolers samspel (logiska satser) matematiskt och på den vägen lösa invecklade slutledningsproblem. Algebran har ju i många avseenden karaktären av slutledning.

Många matematiker var också redan under medeltiden inne på tanken att behandla logiken matematiskt. Först i och med Leibniz kan man dock säga, att någon nämnvärd utveckling skedde. Ett genomgående fel, som de flesta gjorde, var emellertid att försöka behandla logiska symboler exakt på samma sätt som algebraiska med alla de fyra enkla räknesätten, vilket medför oöverstigliga svårigheter.

En man, som hade förmågan att se de praktiska antaganden som behövde göras, var engelsmannen George Boole, och han var också den förste, som satte den matematiska behandlingen av logiskt tänkande i ett användbart system. Hans mest berömda arbete i ämnet är "An Investigation of the Laws of Thought" (titeln något förkortad), som utkom år 1854. Även om senare matematiker infört en del förbättringar, så har den symbolik Boole där använde (Boole's algebra) bildat grunden för den moderna s.k. logikkalkylen, vilken fått vidsträckt användning bl.a. för matematisk analys och syntes av databehandlande organ.

Boole insåg betydelsen av satsen om "allt eller intet" i logiken, dvs att varje förutsättning antingen är helt sann eller helt falsk. Han gav därför sin symbolik en binär karaktär, som möjliggjorde den matematiska behandlingen. Samma karaktär återfinnes även hos elementen i binära datautrustningar. Man har

funnit, att funktionerna hos och samspelet mellan sådana element på ett klart sätt kan uttryckas med hjälp av Boole's symbolik, och framför allt är det möjligt att med algebraiska metoder dels förenkla befintliga binära kretsar och dels konstruera nya sådana uppfyllande önskade logiska villkor. Sådana kretsar, vilka med hjälp av bestämda regler kan komma fram till logiskt riktiga slutsatser för givna förutsättningar, kallas med ett gemensamt namn "logiska kretsar".

## 3.2. Boole's algebra

### 3.2.1. Urval

Antag, att vi har 100 kulor av vilka hälften är vita, klassen X, och resten svarta. Dessutom är hälften av samtliga kulor av trä, klassen Y, och resten av plast. Fördelningen av de senare är sådan, att 25 träkulor är vita och 25 svarta och likadant för plastkulorna. Vill vi ha en vit kula, har vi möjlighet att välja i hela klassen X, dvs bland 50 kulor. Vill vi dessutom ha en träkula, har vi bara möjlighet att välja i den del, som är gemensam för både X och Y, dvs bland 25 kulor. Eftersom X omfattar halva antalet kulor och Y halva, har den möjliga delen av kulor begränsats till $1/2 \cdot 1/2 = 1/4$. Villkoret, att urvalet skall tillhöra b å d e klassen X o c h klassen Y, kan således algebraiskt uttryckas som en produkt $X \cdot Y$ eller klassen $X \cdot Y$.

Resonemanget kan överföras till fallet i avsnitt 3.1.1. Så länge vi endast vet att vi vill ha tag på personen i fråga, kan vi tänka oss flera platser att ringa till, men om vi dessutom får veta att han befinner sig på sitt arbete, kan vi genast begränsa möjligheterna.

Om man i kulfallet ovan först väljer ut klassen X och sedan därur klassen Y eller först klassen Y och därur klassen X, får man tydligen samma resultat. Alltså kan man säga, att för logisk multiplikation gäller:

$$X \cdot Y = Y \cdot X \qquad \ldots\ldots\ldots 3.2.11$$

Innebörden av det logiska uttrycket har alltså ingen betydelse för den symboliska formen. Utmärkande för Boole's tillvägagångssätt vid bildandet av den logiska algebran är, att han endast tar hänsyn till logiska omfångsrelationer, dvs omfattningen av logiska möjligheter och bortser från betydelsen av ett logiskt uttryck, tills detta är färdigbehandlat och skall tolkas. De regler för behandling av logiska uttryck som han härledde gäller därför, även om symbolerna har olika innebörd i olika fall.

En intressant egenskap hos den logiska algebran kan härledas på följande sätt: om man i kulfallet först väljer ut klassen X och därefter ur denna återigen

klassen X, så får man fortfarande klassen X. Motsvarande algebraiska uttryck blir:

$$X \cdot X = X \text{ eller } X^2 = X \qquad \ldots\ldots\ldots 3.2.12$$

Här har vi en utmärkande skillnad mellan den logiska och matematiska algebran.

Generellt kan ett obegränsat antal klasser ingå i ett logiskt uttryck. Fördelningen av en klass i de övriga behöver inte heller vara likformig. Detta har i kulexemplet antagits endast för att enklare illustrera analogin mellan ett logiskt "både-och"-villkor och en algebraisk multiplikation. En i ett logiskt uttryck ingående klassymbol betecknar i själva verket inte en talstorhet (som exempelvis 1/2 eller 1/4) utan karaktären hos en förutsättning eller ett villkor.

En logisk produkt anger som vi sett ett begränsat urval av möjligheter. Dessa möjligheter kan vi kalla b e t i n g a d e, eftersom varje möjlighet är beroende av en eller flera andra. Om vi skall kunna välja en kula av klassen X, är nämligen beroende av om kulan även tillhör klassen Y.

En logisk produkt kallas även för en logisk "och"-funktion (egentligen "både-och"-funktion).

### 3.2.2. Sammanslagning

En annan typ av villkor man kan tänka sig, är villkor med a l t e r n a t i v a möjligheter. Låt oss ta ett annat kulexempel. Vi har 30 st kulor, varav 10 är vita, 10 svarta och 10 röda. Vi vill nu välja en kula vilken som helst, bara den inte är röd. Den kan då antingen vara vit eller svart. Detta ger oss en möjlighet att välja mellan 20 kulor eller 2/3 av hela antalet. Låt oss beteckna klasserna i tur och ordning med X, Y och Z, där vardera klassen tydligen omfattar 1/3 av kulorna. Får vi välja både vitt och svart, har vi däremot möjlighet att välja bland 2/3 av kulorna eller i klassen X + Y. Villkoret att en kula skall tillhöra någon av klasserna X e l l e r Y, kan alltså algebraiskt uttryckas som en sammanslagning eller summa X + Y.

Med samma resonemang som för uttrycket 3.2.11. inser man, att följande likhet gäller:

$$X + Y = Y + X \qquad \ldots\ldots\ldots 3.2.21$$

Likaså gäller:

$$X + X = X, \qquad \ldots\ldots\ldots 3.2.22$$

eftersom man tydligen aldrig kan få m e r än klassen X att välja bland, om man vill ha en vit kula.

Boole själv tillät inte uttryck av formen X + X bland annat av följande skäl: I det nyss angivna kulexemplet antog vi, att inga av klasserna X, Y och Z hade någon kula gemensam. I det första exemplet med vita och svarta kulor av trä eller plast fanns det däremot kulor, som tillhörde både klasserna X och Y. Låt oss göra en sammanslagning även i det fallet genom att anta, att vi vill ha en kula vilken som helst bara det inte är en svart plastkula. Detta ger oss 75 möjliga kulor att välja bland eller 3/4 av samtliga. Dessa kulor är då antingen vita, av trä eller bådadera. Klasserna X och Y ger emellertid vid addition $X + Y = 1/2 + 1/2 = 1$, dvs samtliga kulor skulle vara godtagbara. Det är lätt att se, var felet ligger. Om vi först väljer ut klassen X och sedan klassen Y, så tar vi den för X och Y gemensamma delen $X \cdot Y$ två gånger, dvs vi måste ta bort en sådan del för att få rätt resultat. Vi får då $X + Y - X \cdot Y = 1/2 + 1/2 - 1/4 = 3/4$. Rent formellt säger emellertid minustecknet framför termen $X \cdot Y$, att klassen $X \cdot Y$ inte får komma ifråga (se avsnitt 3.2.3.), vilket enligt ovan måste vara fel. För att undvika tolkningssvårigheter förutsatte då Boole, att klasserna inte fick ha någon del gemensam, vilket stämmer med vårt andra kulexempel. Enligt Boole hade då inte heller uttrycket 3.2.22 någon mening. Man kan emellertid inte alltid undvika uttryck av den formen vid manipulationer med logiska uttryck, varför Boole i sådana fall fick använda diverse konstgrepp vid tolkningen.

Senare matematiker undvek nämnda svårigheter genom att helt enkelt förutsätta, att ett uttryck X + Y omfattar klasserna X eller Y eller båda även om de delvis sammanföll. Gör man så, måste man emellertid ta konsekvenserna av att termen $X \cdot Y$ utelämnas, dvs man kan inte längre behandla de logiska symbolerna som tal eller kvantiteter, vilket redan nämnts i sjätte stycket i avsnitt 3.2.1. Med införande av nu nämnda konstgrepp blir uttrycket 3.2.22. av grundläggande betydelse och har en symmetrisk likhet med uttrycket 3.2.12,, vilket också är grundläggande för denna algebra.

En logisk summa kallas även för en logisk "eller"-funktion. Uttryck av den ovannämnda formen $X + Y - X \cdot Y$ har också de fått stor användning inom binärtekniken och kallas då för "antingen"-funktioner (egentligen "antingen-eller", motsvarande engelska uttryck är "exclusive or"). De är emellertid inte några grundfunktioner utan sammansatta av en "eller"- och en "och"-funktion.

För att illustrera en konkret logisk sats med "eller"-villkor kan vi ta följande mening:

"Om jag får kaffe eller te, dricker jag gärna".

Denna sats kan tydligen symboliseras av uttrycket:

$$A + B = C,$$

om A får beteckna tillgången till kaffe, B tillgången till te och C villkoret att jag gärna dricker.

Vi kan ta ytterligare ett exempel med tillämpning av båda de genomgångna funktionerna:

"Får jag kaffe eller te, dricker jag gärna, om jag är törstig."

Här tillägger vi villkoret D, betecknande att jag är törstig. Innebörden av meningen är annorlunda uttryckt:

"Endast om jag är både törstig och får kaffe eller te, dricker jag gärna."

Detta kan vi uttrycka på följande sätt:

$$D \cdot (A + B) = C$$

Men vi kan även säga:

"Endast om jag både är törstig och får kaffe eller endast om jag både är törstig och får te, dricker jag gärna."

I det fallet blir motsvarande logiska uttryck:

$$D \cdot A + D \cdot B = C$$

dvs $$D \cdot (A + B) = D \cdot A + D \cdot B \qquad \ldots\ldots\ldots 3.2.23$$

Av denna ekvation och ekvation 3.2.21. framgår, att de logiska operationerna är distributiva och kommutativa som i den vanliga algebran.

### 3.2.3. Negation

En uteslutningsoperation eller negation har vi redan sett exempel på i föregående avsnitt. En negation av klassen X kan man skriva som $-X$ och kalla "icke-X". Denna klass betecknar då alla element, som inte tillhör klassen X. Operationen har i denna form numera knappast någon användning i den symboliska logiken, utan en överförd form används, vilket skall framgå av följande avsnitt.

### 3.2.4. Binär tillämpning av Boole's algebra

Av de ovan givna kulexemplen framgick, att klasserna, eller om vi så vill kalla dem, urvalen X, Y eller Z borde anges som delar i ett helt eller "allting" representerat av talet 1. Detta insåg Boole gav den bästa överensstämmelsen mellan logiska och vanliga algebraiska operationer. I detta system betecknar talet 0 "ingenting" eller en klass utan element. Med dessa tolkningar gäller såväl för logisk som vanlig algebra:

$$1 \cdot X = X \qquad \ldots\ldots\ldots 3.2.41$$

dvs klassen X utvald ur det hela ger klassen X.

$$0 \cdot X = 0 \qquad \ldots\ldots\ldots 3.2.42$$

dvs klassen X utvald ur ingenting ger ingenting.

I detta fall uttrycker man negationen till klassen X, dvs "icke-X" eller "allt utom X" som 1-X. Ur detta får man:

$$X + (1 - X) = 1 \qquad \ldots\ldots\ldots 3.2.43$$

dvs summan av allt som är X och allt som inte är X, är allting.

$$X \cdot (1 - X) = X - X^2 = X - X = 0 \qquad \ldots\ldots\ldots 3.2.44$$

dvs klassen som både innehåller och inte innehåller X är noll, den finns inte.

Lagen 3.2.43. tillåter en manipulation, som är mycket viktig vid behandling av logiska uttryck. Antag, att vi har en klass Y. För denna gäller:

$$Y = Y \cdot 1 = Y \left[ X + (1-X) \right] = Y \cdot X + Y \cdot (1-X) \ldots\ldots\ldots 3.2.45$$

Man kan alltså införa en godtycklig symbol, utan att ändra giltigheten av det logiska uttrycket. Det erhållna uttrycket kan tolkas som att Y antingen måste höra ihop med en godtycklig klass X eller också med icke-X.

Negationen 1–X brukar ofta förkortas till $X^1$ (X-prim) eller $\overline{X}$ (X-streck). Den kallas även inverteringen eller logiska komplementet (med karaktären 1-komplement) av X.

Även uttryck av formen 1+X förekommer i Boole's algebra. Eftersom 1 eller allting redan omfattar klassen X, inser man att ett tillägg av denna klass inte ger något utöver allting eller 1, varför följande uttryck gäller:

$$1 + X = 1 \qquad \ldots\ldots\ldots 3.2.46$$

För att göra sin symbolik algebraiskt användbar föreslog Boole helt enkelt, att de villkor eller förutsättningar som ersattes av symbolerna skulle uttryckas i sådan form att symbolerna endast kunde anta värdena 0 eller 1. Under den förutsättningen vore det möjligt att behandla ett logiskt uttryck med alla erforderliga algebraiska hjälpmedel och efter behandlingen återföra detsamma till den logiska tolkningen. Villkoren eller förutsättningarna ges då lämpligen sådan form, att de kan betecknas som "sanna" eller "falska" för varje tillämpningsfall. "Sann" kan då exempelvis betecknas med en etta och "falsk" med en nolla.

Om vi går tillbaka till inledningen och ser på uttrycket 3.1.21. så skrivs det symboliskt:

$$A \cdot B = C$$

Om de av A och B ersatta påståendena är sanna, sätter vi A och B lika med 1.

$$A \cdot B = 1 \cdot 1 = C$$

dvs C blir också 1 eller sann.

Genom senare matematikers tolkning av ett uttryck av formen A + B som klassen omfattande A eller B eller båda (jfr femte stycket avsnitt 3.2.2.) uppträder ett par viktiga och intressanta relationer mellan uttryck av formen A + B och uttryck av formen A·B, förutsatt att A och B är binära storheter.

Att A och B b å d a är sanna, kan vi uttrycka som $A \cdot B = 1$ enligt ovan. Negerar vi förhållandet, innebär det, att A och B i c k e b å d a är sanna. Minst en av dem måste vara falsk eller 0 dvs $A \cdot B = 0$. Negationen kan som vi sett även uttryckas som $1 - A \cdot B$ eller $\overline{A \cdot B}$ dvs allt, som inte tillhör den klass där A och B båda är sanna. Eftersom $A \cdot B$ är falskt enligt antagnadet, måste $\overline{A \cdot B}$ vara sant eller $\overline{A \cdot B} = 1$ (rent algebraiskt: $1 - A \cdot B = 1 - 0 = 1$). Negationen till ett falskt förhållande är ett sant förhållande. Eftersom antingen A eller B eller båda är falska (även det sista måste kunna gälla, varför förutsättningen enligt föregående stycke är nödvändig), är negationen av A eller negationen av B eller båda sanna dvs $\overline{A} + \overline{B} = 1$. Då nu $\overline{A \cdot B} = 1$ och $\overline{A} + \overline{B} = 1$ gäller för samma förutsättning (att A och B i c k e b å d a är sanna), måste $\overline{A \cdot B} = \overline{A} + \overline{B}$. Genom liknande resonemang kan relationen utsträckas till godtyckligt många symboler, varför allmänt gäller:

$$\overline{X_1 \cdot X_2 \cdot X_3 \ldots} = \overline{X}_1 + \overline{X}_2 + \overline{X}_3 + \ldots \qquad \ldots\ldots\ldots 3.2.47$$

Denna regel är mycket viktig vid manipulationer med logiska uttryck. En motsvarande regel för en negerad "eller"-funktion kan härledas på liknande sätt: om A eller B eller båda (dvs minst en av dem) är sanna, skriver vi $A + B = 1$. Negationen av detta förhållande dvs $A + B = 0$ eller $\overline{A + B} = 1$ innebär, att ingen av A eller B är sann eller om man så vill att negationen av A och negationen av B båda är sanna dvs $\overline{A} \cdot \overline{B} = 1$. Alltså är $\overline{A + B} = \overline{A} \cdot \overline{B}$ eller allmänt:

$$\overline{X_1 + X_2 + X_3 + \ldots} = \overline{X}_1 \, \overline{X}_2 \, \overline{X}_3 \ldots \qquad \ldots\ldots\ldots 3.2.48$$

Ekvationerna 3.2.47. och 3.2.48. illustrerar den s.k. d u a l i t e t s p r i n c i p e n (De Morgans teorem), som säger att inverteringen av en "och"- eller "eller"-funktion ger den andra funktionen av de inverterade symbolerna.

I detta avsnitt har framgått, hur användbar Boole's algebra är för algebraisk behandling av villkor och förutsättningar med "tvåläges"-tillstånd vilket gör den utomordentligt lämplig för analys av binära tillstånd exempelvis i en binär databehandlingsutrustning.

## 3.3. Logiska kretsar

### 3.3.1. Allmänt

En databehandlingsutrustning är generellt ett organ för behandling av data, som exempelvis kan vara uppgifter om en pågående process eller storheter i ett rent matematiskt problem. De behandlade data kan då användas för vidare styrning av den nämnda processen eller utgöra svar på det matematiska problemet. I utrustningen behövs ett antal mindre organ, som kontrollerar behandlingen efter vissa uppställda regler. Dessa regler samspelar på olika sätt och bildar en behandlingsgång, under vilken data förändras till den slutliga formen. Samspelet mellan reglerna är ofta beroende dels av karaktären hos ursprungsdata och dels av karaktären hos olika delresultat i behandlingsgången. Kontrollorganen skall därför ha förmågan att göra vissa slutledningar efter bestämda regler.

Grundtypen för kretsar, som används för sådan kontroll i digitala elektroniska behandlingsutrustningar, är den s.k. grinden (eng. gate). Den definieras som "en krets med en utgång och ett variabelt antal ingångar, som är så konstruerad att utgången aktiveras endast då en bestämd uppsättning av ingångstillstånd uppfylls". Samspelet mellan sådana grindar kan teoretiskt behandlas, och i synnerhet är den booleska symboliken användbar, eftersom kontrolltillstånden kan representeras med termer innehållande de diskreta binärtalen 0 och 1. Dessa siffror kan beteckna tillståndet hos många vanliga bistabila element exempelvis ledande eller oledande tillstånd, uppladdat eller ouppladdat, tillslaget eller frånslaget (jfr avsnitt 1.3.).

Med grindkretsar kan de tre logiska grundfunktionerna enligt avsnitt 3.2. dvs "och"-, "eller" – samt "icke"-funktionerna realiseras. Tillsammans med funktionen "minnas" räcker de för bildandet av samtliga logiska villkor i binära kontrollkretsar.

Ibland skiljer man i terminologin mellan en kontrollkrets med minnesfunktion och en sådan utan. Den förra, allmännare typen kallas då kopplingskrets och den senare kombinatorisk krets. På grund av minnesfunktionens speciella karaktär behandlas endast den senare typen av kretsar i detta kapitel. Minneselement behandlas i kapitlen "Sekvenskretsar" och "Minnen".

### 3.2.2. De logiska grundfunktionerna

#### 3.3.2.1. Kretssymboler

Fig. 3.1 visar den vid SRT använda generella symbolen för en grindkrets utan närmare angivande av funktionen. Tillstånden på ingångarna, ingångsvariab-

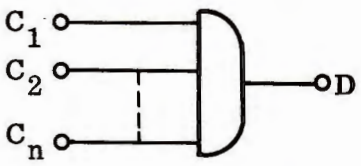
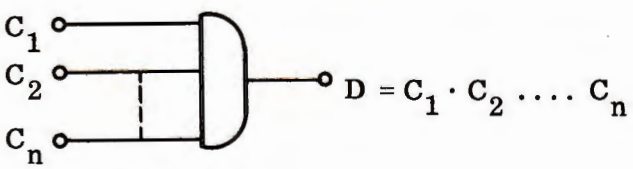


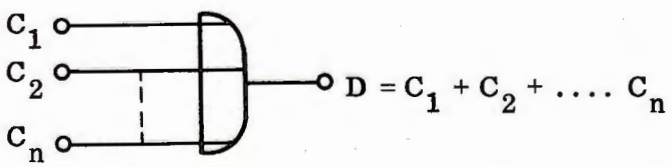

Fig. 3.1

lerna (motsvarande förutnämnda klasser), betecknas med $C_i$ och tillståndet på utgången, utgångsfunktionen, med D. $C_i$ och D antas kunna ha endast värdena 0 eller 1 i samtliga funktioner. Detta överensstämmer inte alltid med förhållandena i en fysikalisk krets, eftersom en sådan inte momentant kan övergå från ett tillstånd till ett annat. Vill man alltså fysikaliskt realisera ett logiskt villkor med föränderliga ingångsvariabler, måste man ta hänsyn till stig- och falltider hos kretsarna, dvs i praktiken kan alltid en minnesfunktion komma med i bilden. I de flesta tillämpningar kan sådana problem undvikas genom omsorgsfullt urval och noggrann dimensionering av kretsarna. I ett fall är förhållandet emellertid av betydelse nämligen i mycket snabba utrustningar, där signalernas löptider på ledningarna är av samma storleksordning som omkopplingsperioderna i kretsarna.

Enligt fig. 3.1 är D alltså en funktion av de olika $C_i$ och kan anta värdena 0 eller 1 beroende på värdena hos $C_i$. Funktionen kan därvid vara av "och"-, "eller"- eller "icke"-typ. Fig. 3.2 visar de vid SRT använda grindsymbolerna för de tre logiska grundfunktionerna, vilka överensstämmer med av SEK (Svenska Elektriska Kommissionen) föreslagen svensk standard (se SEN 0116). "Icke"-funktionen har i sin grundform endast en ingång och en utgång och används ofta till att negera en ingång till en "och"- eller "eller"-funktion. Ringbeteckningen är därför lämplig, eftersom den kan appliceras direkt på den negerade ingången till en "och"- eller "eller"-grind (se exempel i fig. 3.2).

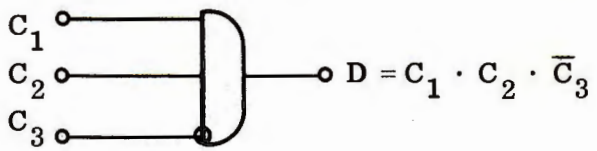



"Och"-funktion




"Eller"-funktion


"Icke"-funktion




"Och"-funktion med negerad ingång

Fig. 3.2

3.3.2.2. Sanningstabeller för grundfunktionerna

Vid uppställandet av villkoren för en logisk krets som skall konstrueras, har man normalt givet ett antal ingångsvariabler samt utgångsfunktionens beroende av dessa. Beroendet kan översiktligt illustreras i en s.k. sanningstabell. En sådan innehåller alla kombinationer av ingångsvariablerna och motsvarande värden för utgångsfunktionen. Ur sanningstabellen plockar man ut de kombinationer, för vilka utgångsfunktionen exempelvis är sann och sätter upp den logiska ekvationen för dessa kombinationer. Därmed har man åskådliggjort det önskade villkoret i algebraisk form. Man kan också plocka ut de kombinationer, som ger en falsk utgångsfunktion, och sätta upp ekvationen för dem. Man får i båda fallen samma villkor, vilket inses av förhållandet att ett villkor som för vissa kombinationer ger sann utgång måste ge falsk utgång för alla övriga kombinationer.

Nedan är sanningstabellerna för grundfunktionerna uppställda. För enkelhets skull har ingångsvariablerna begränsats till två, $C_1$ och $C_2$, för "och"- och "eller"-funktionerna. "Icke"-funktionen har enligt föregående avsnitt endast en ingång.

| $C_1$ | $C_2$ | $D = C_1 \cdot C_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

"Och"-funktion

D är sann, om $C_1$ och $C_2$ är sanna

"Och"-funktionen eller den "logiska produkten" bildas alltså på samma sätt som en aritmetisk produkt (jfr ekv. 3.2.41 och 3.2.42).

| $C_1$ | $C_2$ | $D = C_1 + C_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

"Eller"-funktion

D är sann, om $C_1$ eller $C_2$ är sann

"Eller"-funktionen eller den "logiska summan" bildas alltså på samma sätt som en aritmetisk summa utom i det avseendet, att den maximala summan är 1 (jfr ekv. 3.2.46).

| C | $D = \overline{C}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

"Icke"-funktion
D är sann, om C icke är sann

"Icke"-funktionen eller det "logiska komplementet" bildas alltså på samma sätt som 1-komplementet av en binär siffra (jfr fjärde stycket i avsnitt 3.2.4).

### 3.3.2.3. Några räkneregler

I föregående avsnitt berördes räknereglerna för de logiska grundfunktionerna. I komplexa system kan dessa förekomma tillsammans i olika former. Vid den matematiska behandlingen av ett komplext uttryck utnyttjar man sig av nedan uppställda förhållanden eller regler, som gäller för en godtycklig logisk variabel C. De kan direkt härledas ur räknereglerna för grundfunktionerna och är för övrigt närmare förklarade i det tidigare avsnittet 3.2.

Regel 1: $C + 0 = C$ (se första stycket i avsnitt 3.2.4.)
Regel 2: $C \cdot 0 = 0$ (se ekv. 3.2.42)
Regel 3: $C + 1 = 1$ (se ekv. 3.2.46)
Regel 4: $C \cdot 1 = 1$ (se ekv. 3.2.41)
Regel 5: $C + C = C$ (se ekv. 3.2.22)
Regel 6: $C \cdot C = C$ (se ekv. 3.2.12)
Regel 7: $C + \overline{C} = 1$ (se ekv. 3.2.43)
Regel 8: $C \cdot \overline{C} = 0$ (se ekv. 3.2.44)
Regel 9: $\overline{\overline{C}} = C$ (enligt definition på negering)
Regel 10: $\overline{C_1 \cdot C_2} = \overline{C}_1 + \overline{C}_2$ (se ekv. 3.2.47)
Regel 11: $\overline{C_1 + C_2} = \overline{C}_1 \cdot \overline{C}_2$ (se ekv. 3.2.48)

I föregående avsnitt nämndes, att man får samma logiska villkor ur en sanningstabell vare sig man sätter upp den logiska ekvationen för ett sant värde på utgångsfunktionen eller för ett falskt värde. Med hjälp av Regel 10 kan vi som exempel bevisa, att detta gäller för "och"-funktionen vars sanningstabell finns uppställd i föregående avsnitt. I denna uppmärksammar vi, att D är falsk om $C_1$ eller $C_2$ eller båda är falska. Detta är villkoret för "eller"-funktionen $\overline{D} = \overline{C}_1 + \overline{C}_2$, varur enligt Regel 10 erhålles $\overline{D} = \overline{C_1 \cdot C_2}$ eller efter negering av båda leden $D = C_1 \cdot C_2$.

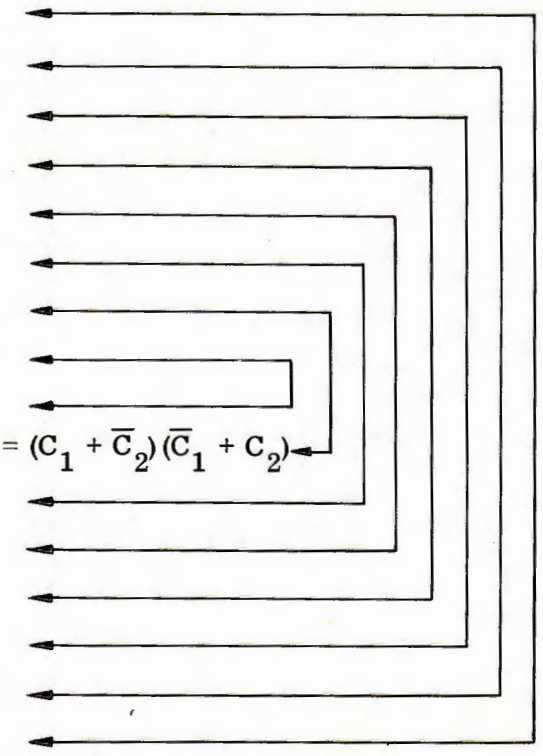
### 3.3.3. Godtyckliga funktioner av två logiska variabler

#### 3.3.3.1. Funktionerna och deras inbördes samband. Dualitetsprincipen

Det kan vid första anblicken synas onödigt att utveckla en speciell algebra för storheter, som endast kan anta de enkla värdena 0 och 1. Men låt oss se, hur många funktioner som kan erhållas redan för så litet som två ingångsvariabler. Det finns tydligen $2^2 = 4$ olika kombinationsmöjligheter för dessa ingångsvariabler och för varje kombination kan utgångsfunktionen anta de två värdena 0 och 1. Totalt kan då $2^4 = 16$ olika funktioner (enkla eller sammansatta grundfunktioner) erhållas, vilket närmare framgår nedan. För tre ingångsvariabler stiger antalet möjliga funktioner till $2^{2^3} = 256$. Ett systematiskt hjälpmedel kan alltså vara av stort värde, då det gäller att för ett visst tillämpningsfall sortera ut den rätta funktionen eller kretsen.

De 16 funktioner, som enligt ovan kan erhållas för två variabler $C_1$ och $C_2$, har var sin sanningstabell. Nedan har samtliga 16 uppställts tillsammans med motsvarande logiska ekvationer. Observera att tabellerna här placerats i horisontell led och inte vertikalt som i avsnitt 3.3.2.2. Multiplikationstecknen i de logiska produkterna har borttagits för att skrivsättet skall bli enklare.

| | | | |
|---|---|---|---|
| 0 0 1 1 | $C_1$ | } Ingångs- | |
| 0 1 0 1 | $C_2$ | } variabler | |
| 0 0 0 0 | $D_0$ | $= 0$ | |
| 0 0 0 1 | $D_1$ | $= C_1C_2$ | |
| 0 0 1 0 | $D_2$ | $= C_1\overline{C}_2$ | |
| 0 0 1 1 | $D_3$ | $= C_1$ | |
| 0 1 0 0 | $D_4$ | $= \overline{C}_1C_2$ | |
| 0 1 0 1 | $D_5$ | $= C_2$ | |
| 0 1 1 0 | $D_6$ | $= \overline{C}_1C_2 + C_1\overline{C}_2$ | |
| 0 1 1 1 | $D_7$ | $= C_1 + C_2$ | |
| 1 0 0 0 | $D_8$ | $= \overline{C}_1\overline{C}_2$ | |
| 1 0 0 1 | $D_9$ | $= \overline{C}_1\overline{C}_2 + C_1C_2 = (C_1 + \overline{C}_2)(\overline{C}_1 + C_2)$ | |
| 1 0 1 0 | $D_{10}$ | $= \overline{C}_2$ | |
| 1 0 1 1 | $D_{11}$ | $= C_1 + \overline{C}_2$ | |
| 1 1 0 0 | $D_{12}$ | $= \overline{C}_1$ | |
| 1 1 0 1 | $D_{13}$ | $= \overline{C}_1 + C_2$ | |
| 1 1 1 0 | $D_{14}$ | $= \overline{C}_1 + \overline{C}_2$ | |
| 1 1 1 1 | $D_{15}$ | $= 1$ | |



Den första och den sista funktionen i tabellen har ingen egentlig logisk innebörd. I en fysikalisk krets kan de anses motsvara avbrott respektive kortslutning.

I tabellen ovan känner vi igen den enkla "och"-funktionen i $D_1$, den enkla "eller"-funktionen i $D_7$ samt "Icke"-funktionen i $D_{10}$ och $D_{12}$.

I tabellen är funktioner $D_i$ och $D_{(15-i)}$ parvis sammanbundna med pilar. Om vi ser på två sådana sammanbundna funktioner, så finner vi att de två värdekombinationerna (till vänster) är varandras 1-komplement samt att de båda logiska ekvationernas högra led kan bildas ur varandra genom följande byten:

$$0 \text{ —— } 1$$
$$+ \text{ —— } \cdot$$
$$C_1 \text{ —— } \overline{C}_1$$
$$C_2 \text{ —— } \overline{C}_2$$

Här har vi alltså flera enkla exempel på den redan förut omnämnda dualitetsprincipen (se slutet av avsnitt 3.2.4.). Denna innebär tydligen bland annat, att en "och"-funktion av ett slags variabler fungerar som en "eller"-funktion av dessa variablers komplement ($D_1$ och $D_{14}$). $D_{14}$ är nämligen komplementet av $D_1$ (vilket bl.a. framgår av värdekombinationerna till vänster). Alltså gäller:

$$D_{14} = \overline{D}_1 = \overline{C}_1 + \overline{C}_2$$

Samma förhållande framgår redan av Regel 10 i avsnitt 3.3.2.3., som säger:

$$\overline{D}_1 = \overline{C_1 C_2} = \overline{C}_1 + \overline{C}_2$$

Av sambandet mellan $D_7$ och $D_8$ samt Regel 11 framgår likaså, att en "eller"-funktion fungerar som en "och"-funktion för variablernas komplement.

Dualitetsprincipen är lätt att fatta rent förnuftsmässigt. Vad gäller inverteringen av $D_1$ kan man nämligen resonera så, att $D_1$ är sann om $C_1$ och $C_2$ är sanna, varför $D_1$ är falsk, om någon av $C_1$ eller $C_2$ är falsk. Likaså gäller för inverteringen av $D_7$, att $D_7$ är sann, om någon av $C_1$ eller $C_2$ är sann, varför $D_7$ är falsk endast om $C_1$ och $C_2$ båda är falska.

### 3.3.3.2. Elementarprodukter. Den kanoniska formen

Av tabellen i föregående avsnitt framgår, att funktionerna $D_1$, $D_2$, $D_4$ och $D_8$ har endast en etta vardera i respektive sanningstabell, dvs de är sanna för vardera endast en variabelkombination. Sådana funktioner som samtliga är "och"-funktioner, kallas elementarprodukter eller elementarkombinationer. Varje funktion kan tydligen bildas som en logisk summa av sådana elementarprodukter (en elementarprodukt för varje etta i sanningstabellen). $D_7$ kan exempelvis skrivas:

$$D_7 = \overline{C}_1 C_2 + C_1 \overline{C}_2 + C_1 C_2$$

Ekvationen kan givetvis hyfsas till den form den har i tabellen:

$$D_7 = \overline{C}_1 C_2 + C_1 \overline{C}_2 + C_1 C_2 + C_1 C_2 = C_2(\overline{C}_1 + C_1) + C_1(\overline{C}_2 + C_2) = C_1 \cdot 1 + C_2 \cdot 1 = C_1 + C_2$$

En funktion som är uttryckt som en summa av elementarprodukter sägs vara skriven i kanonisk form.

### 3.3.4. Funktioner av ett godtyckligt antal variabler

Ett godtyckligt antal, n stycken, ingångsvariabler kan bilda $2^n$ olika kombinationer. Nedan är en tabell uppställd visande kombinationerna för 3 variabler $C_1$, $C_2$ och $C_3$ samt sanningstabellerna för några av de första utgångsfunktionerna.

| | | |
|---|---|---|
| 0 0 0 0 1 1 1 1 | $C_1$ | Ingångs-variabler |
| 0 0 1 1 0 0 1 1 | $C_2$ | |
| 0 1 0 1 0 1 0 1 | $C_3$ | |
| 0 0 0 0 0 0 0 0 | $D_0 = 0$ | |
| 0 0 0 0 0 0 0 1 | $D_1 = C_1 C_2 C_3$ | |
| 0 0 0 0 0 0 1 0 | $D_2 = C_1 C_2 \overline{C}_3$ | |
| 0 0 0 0 0 0 1 1 | $D_3 = C_1 C_2$ | |
| 0 0 0 0 0 1 0 0 | $D_4 = C_1 \overline{C}_2 C_3$ | o.s.v. |

$D_1$, $D_2$, $D_4$, $D_8$ osv är även här elementarprodukter men nu i form av "och"-funktioner för tre variabler. För ett system med n variabler är en elementarprodukt allmänt en "och"-funktion för dessa n variabler. Helt analogt med vad som framhölls i föregående avsnitt kan varje logisk funktion av n variabler uttryckas i kanonisk form med dessa n variabler.

Eftersom antalet elementarprodukter för n variabler är $2^n$ (en för varje kombination av variablerna), kan varje funktion av dessa variabler uttryckas som en summa av $N=2^n$ stycken elementarprodukter $E_i$:

$$D = k_0 \cdot E_0 + k_1 \cdot E_1 + \ldots + k_{N-1} \cdot E_{N-1}$$

Koefficienterna $k_i$ är 1 eller 0 beroende på om elementarprodukten i fråga ger en etta eller nolla i D. Funktionen D karaktäriseras då tydligen helt av de N stycken $k_i$.

Då det nu finns N stycken $k_i$ med vardera två möjliga värden, är antalet möjliga kombinationer av dessa $k_i$ tydligen $= 2^N = 2^{2^n}$, vilket samtidigt är antalet möjliga logiska funktioner D (jfr början av avsnitt 3.3.3.1.).

Ett bekvämt och lättskrivet sätt att ange en logisk funktion är att använda numret på funktionen i en uppställning av sanningstabeller av den typ som visats i avsnitt 3.3.3.1. En "eller"-funktion av två variabler kan då skrivas exempelvis $D_{7(2)}$, där tvåan anger antalet variabler. Ur denna beteckning kan funktionens sanningsschema lätt härledas genom översättning av det decimala indexet givet av antalet variabler. $D_{7(2)}$ ger då:

$k_0 = k_1 = k_2 = 1$; $E_0 = C_1C_2$ (börja från höger i sanningstabellen)

$k_3 = 0$ $E_1 = C_1\overline{C}_2$

$E_2 = \overline{C}_1C_2$

Alltså är $D_{7(2)} = C_1C_2 + C_1\overline{C}_2 + \overline{C}_1C_2$ (jfr föregående avsnitt)

## 3.3.5. Byggelement för logiska kretsar

### 3.3.5.1. Rent logiska kretssymboler

I de två föregående avsnitten har framgått, att varje logisk funktion kan uttryckas som en summa av elementarprodukter. Varje logisk krets av kombinatorisk typ (se slutet av avsnitt 3.3.1) kan då också bildas av grindkretsarna för grundfunktionerna "och", "eller" samt "icke".

Låt oss som exempel konstruera den logiska kretsen för en "binär adderare" eller enklare "adder". En sådan bildar en summasiffra $S_n$ och en minnessiffra $C_{n+1}$ av två ingångsvariabler $A_n$ och $B_n$ och en minnessiffra $C_n$ från närmast lägre sifferposition (jfr avsnitt 2.2.). I en enkel adder matas de olika positionerna (index n) in efter varandra i tid (serieadder). Ibland använder man också en adder för varje position (parallelladder). I det förra fallet måste addern minnas $C_n$ från en tidpunkt till en annan, medan i det senare $C_n$ tas från steget närmast lägre men under en och samma tidpunkt för samtliga steg.

Följande sanningstabell för addition kan uppställas:

| $A_n$ | $B_n$ | $C_n$ | $S_n$ | $C_{n+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

De logiska ekvationerna för summan $S_n$ och minnessiffran $C_{n+1}$ för närmast högre position blir då:

$$S_n = \overline{A}_n\overline{B}_nC_n + \overline{A}_nB_n\overline{C}_n + A_n\overline{B}_n\overline{C}_n + A_nB_nC_n$$

$$C_{n+1} = \overline{A}_nB_nC_n + A_n\overline{B}_nC_n + A_nB_n\overline{C}_n + A_nB_nC_n$$

Vi förenklar först $C_{n+1}$ enligt följande med hjälp av reglerna i avsnitt 3.3.2.3.:

$$C_{n+1} = C_n(\overline{A}_nB_n + A_n\overline{B}_n) + A_nB_n(\overline{C}_n + C_n) + 2\,A_nB_nC_n =$$

$$= A_nB_n + C_n(\overline{A}_nB_n + A_nB_n) + C_n(A_n\overline{B}_n + A_nB_n) = A_nB_n +$$

$$+ C_nB_n(\overline{A}_n + A_n) + C_nA_n(\overline{B}_n + B_n) =$$

$$= A_nB_n + A_nC_n + B_nC_n$$

Därefter förenklar vi $S_n$:

$$S_n = A_nB_nC_n + \overline{A}_n\overline{B}_nC_n + \overline{A}_n\overline{C}_nB_n + \overline{B}_n\overline{C}_nA_n =$$

$$= A_nB_nC_n + (A_n + B_n + C_n)(\overline{A}_n\overline{B}_n + \overline{A}_n\overline{C}_n + \overline{B}_n\overline{C}_n) =$$

$$= A_nB_nC_n + (A_n + B_n + C_n)(\overline{A_n + B_n} + \overline{A_n + C_n} + \overline{B_n + C_n}) =$$

$$= A_nB_nC_n + (A_n + B_n + C_n)\overline{(A_n + B_n)(A_n + C_n)(B_n + C_n)} =$$

$$= A_nB_nC_n + (A_n + B_n + C_n)\overline{(A_nB_n + A_nC_n + B_nC_n)} =$$

$$= A_nB_nC_n + (A_n + B_n + C_n)\overline{C}_{n+1}$$

Med ledning av de förenklade logiska uttrycken för $S_n$ och $C_{n+1}$ kan vi upprita följande "symboliska schema":

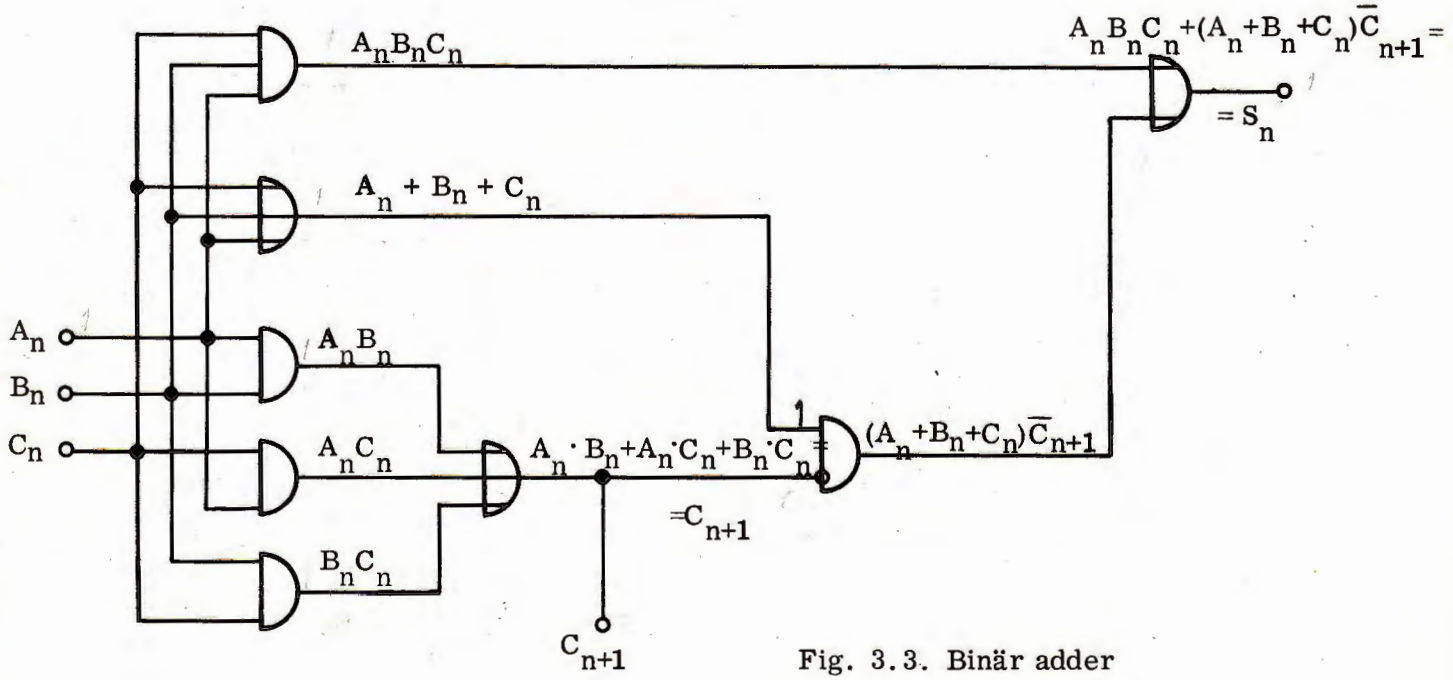



Fig. 3.3. Binär adder

Givetvis kan man också upprita adderns symbolschema direkt efter de icke hyfsade uttrycken för $S_n$ och $C_{n+1}$. Schemat blir emellertid i det fallet mer komplicerat. För $C_{n+1}$ behövs nämligen fyra "och"-grindar samt en "eller"-grind och för $S_n$ ytterligare tre "och"-grindar samt en "eller"-grind, vilket ger sammanlagt nio grindar. Dessutom måste i praktiken en "icke"-funktion normalt betraktas som en separat krets, en inverterare, ofta en fasvändande förstärkare. I det senare fallet skulle tre sådana inverteringar behövas mot en enda enligt fig. 3.3. En grind och två inverterare har således inbesparats med lösningen enligt fig. 3.3. Denna ger också i medeltal färre ingångar per grind, vilket är komponentbesparande.

### 3.3.5.2. Praktiska logiska kretssymboler

Med hjälp av logiska symbolschemor av den typ som visats i fig. 3.3 kan man fullständigt specificera funktionen hos de kombinatoriska kretsarna i en binär elektronisk utrustning. Efter ett sådant schema skall man också kunna bygga upp utrustningen fysikaliskt. Då har man emellertid behov av att införa en del andra kretssymboler, som förutom den logiska funktionen hos respektive krets även anger en del praktiska utförandeformer. Dessa är ofta betingade av behovet av signalförstärkning i mera komplicerade kretskombinationer, eftersom varje krets av passiv typ dämpar signalerna för de logiska variablerna. De vanligaste passiva grindarna byggs av dioder och motstånd, medan aktiva kretsar utgörs av förstärkare. De senare kan ofta samtidigt med förstärkningsfunktionen bilda en logisk funktion, varigenom kretsmässiga förenklingar ibland kan erbjudas.

Antalet specialsymboler kan variera högst betydligt med kraven på en utrustning och utrustningens art. Normalt är de heller inte av intresse för den rent logiska funktionen, varför här endast skall anges ett par av de vanligaste, som används vid SRT.

Som nämndes i slutet av föregående avsnitt får man normalt lov att betrakta en "icke"-funktion som en separat krets, inverter eller fasvändare. I en vanlig fasvändande förstärkare får man både fasvändning och förstärkning, varför en "icke"-funktion med fördel symboliseras med ett förstärkartecken enligt fig. 3.4.

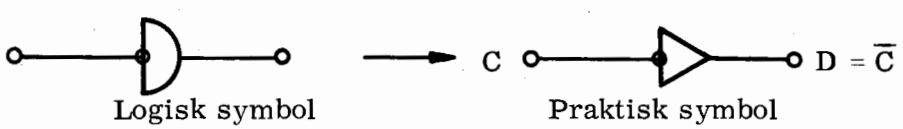
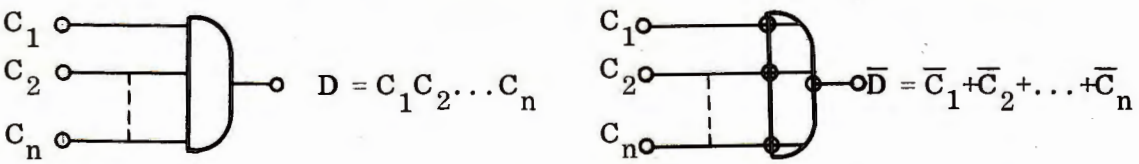



Fig. 3.4.

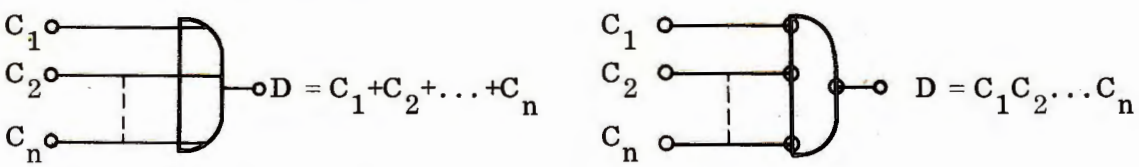
Ibland kan man emellertid behöva förstärkning utan invertering. Eftersom de vanligaste förstärkarna med god återställning av en spänningsnivå är fasvändande, behövs tydligen två steg för förstärkning utan invertering. Ett sådant slöseri med förstärkare kan många gånger undvikas genom att tilldela variablerna olika polariteter i olika kretsar. Normalt tänker man sig en viss polaritet hos variabelsignalerna, om motsvarande variabler har värdet 1 (är sanna) och denna polaritet är ofta den positiva. Om emellertid i fig. 3.4 variabeln C ges negativ polaritet och funktionen D positiv polaritet för värdet 1, så blir kretsen rent förstärkande.

På grund av de i praktiken nödiga förstärkningarna med åtföljande invertering kan det också vara önskvärt att ha "och"- och "eller"-grindar för inverterade variabler. Som vi såg i avsnitt 3.3.3.1. kan grindarna för de normala variablerna även användas för de inverterade, om man bara byter funktion. Det skulle alltså strängt taget inte behövas några nya symboler för grindar för inverterade variabler. En "och"-funktion skulle i det fallet emellertid betecknas med en "eller"-funktion av normalt polariserade variabler, varför kretsens logiska innebörd inte direkt skulle framgå, och detta kan ibland vara besvärligt vid tolkningen av ett schema.

Vill man alltså underlätta schemaläsningen, kan man införa de inverterade symbolerna enligt fig. 3.5.



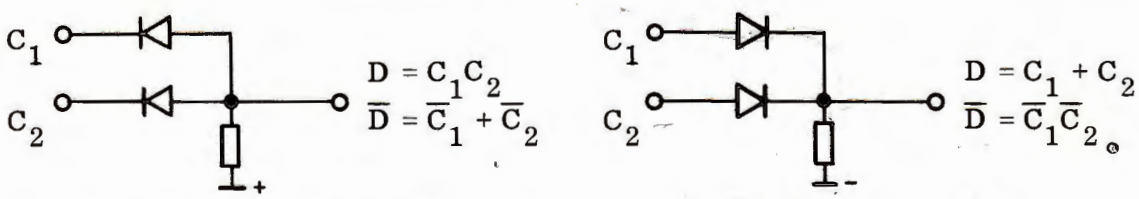

"Och"-grind för normal polaritet "Eller"-grind för invers polaritet




"Eller"-grind för normal polaritet "Och"-grind för invers polaritet

Fig. 3.5

Ringen på en grindutgång betecknar att även utgångsfunktionen skall vara inverterad för att funktionssambandet skall gälla.




Positiv "och"-grind eller negativ "eller"-grind

a)

Positiv "eller"-grind eller negativ "och"-grind

b)

Fig. 3.6

Dualitetsprincipen som den framstår i fig. 3.5 illustreras närmare i fig. 3.6 med ett par exempel på hur grindar fysikaliskt realiseras med diodkretsar. I fig. 3.6 a måste både $C_1$ och $C_2$ vara positiva, för att D skall vara positiv. D är alltså negativ, om någon av $C_1$ eller $C_1$ är negativ. I fig. 3.6 b är D positiv, om någon av $C_1$ eller $C_2$ är positiv, och negativ om $C_1$ och $C_2$ är negativa.

Inverteraren i fig. 3.4 kan även användas för att ange de signifikanta polariteterna på in- och utgång. Beteckningen i fig. 3.4 kan man tolka som att D=0 om C=1 och tvärtom, men också som att D=1 om $\overline{C}$=1 dvs den inverterade polariteten är sann på ingången men den rättvända är sann på utgången. Beteckningen enligt fig. 3.7 kan även tänkas, om de signifikanta polariteterna växlas.

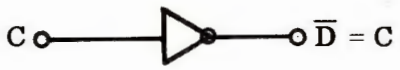



Fig. 3.7

En inverterande förstärkares fysikaliska egenskaper gör att den i vissa fall ensam kan bilda mer än en logisk funktion. Ett exempel har man i den s.k. inverterande och-kretsen (eng. nor circuit). Med exempelvis en P-N-P-transistor som förstärkande element utnyttjar denna krets förhållandet, att transistorn leder (ger positiv utspänning) så fort någon av ingångarna är negativ, dvs utgången blir negativ, om alla ingångarna är positiva. Fig. 3, 8 visar den normala och inversa symbolen för en sådan förstärkare med två ingångar.

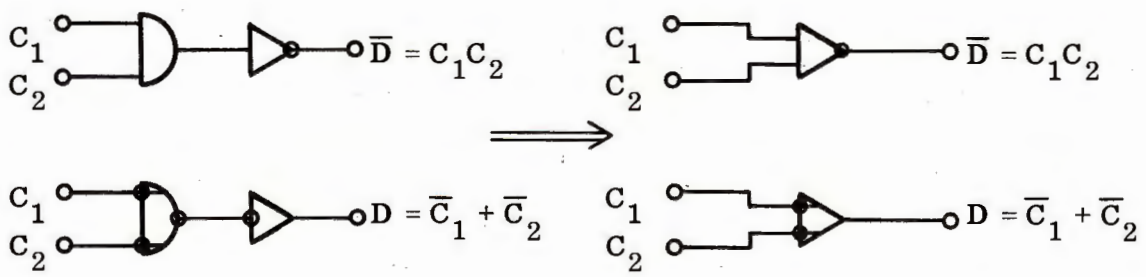



Fig. 3.8